

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.3b (2018/05/18); documentation date: 2018/05/18.

From source file `xint.dtx`. Time-stamp: <18-05-2018 at 19:33:37 CEST>.

Contents

1	Package <code>xintkernel</code> implementation	4
2	Package <code>xinttools</code> implementation	18
3	Package <code>xintcore</code> implementation	54
4	Package <code>xint</code> implementation	113
5	Package <code>xintbinhex</code> implementation	154
6	Package <code>xintgcd</code> implementation	166
7	Package <code>xintfrac</code> implementation	179
8	Package <code>xintseries</code> implementation	262
9	Package <code>xintcfrac</code> implementation	271
10	Package <code>xintexpr</code> implementation	294

This is 1.3b of 2018/05/18.

- Release 1.3b of 2018/05/18:
 - `\xintUniformDeviate`,
 - `random()`, `grand()`, `randrange()`,
 - and their support macros `\XINTinRandomFloatSdigits`, `\XINTinRandomFloatSixteen`, `\xintRandomDigits`, `\xintiiRandRange`, `\xintiiRandRangeAtoB`,
 - also `\xintXRandomDigits`,
 - functions defined via `\xintdeffunc` (et al.) to be without argument can be used as `foo()`: `foo(nil)` syntax not anymore required.
- Release 1.3a of 2018/03/07:
 - removes from `xintcore`, `xint` and `xintfrac` the whole deprecation mechanism, as there are no more currently any deprecated macro,
 - adds `ifone()` and `ifint()` conditionals to the expression parsers,
 - has a completely redone `\XINT_factortens`, in the style of 1.2 release (but about 100 digits at least are needed for noticeable speed gain),
 - and last but not least fixes via addition of `\xintExpandArgs` the meaning of user defined functions, which in case of recursivity (as made possible at 1.3) were badly inefficient for lack of expansion of their arguments.
- Release 1.3 of 2018/03/01:
 - removed all macros previously deprecated at 1.2o,
 - modified addition, subtraction and modulo operations to use a least common multiple for the denominator of the result,

Contents

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

Improvements and new features:

- `xinttools` macro `\xintListWithSep` is faster (first update since 1.04-2013/04/25...).
 - `divmod()` function added to the `xintexpr` parsers,
 - `\xintdefvar`, `\xintdeffloatvar`, `\xintdefiivar` extended to allow multiple simultaneous assignments.
- Release 1.2o of 2017/08/29 deprecated those macros from `xintcore` and `xint` which filtered their arguments via `\xintNum`. Currently these macros execute as formerly but raise an error message. This deprecation is overruled for most if `xintfrac` is loaded as it provides their proper definitions. Some however (like `\xintiAdd`) remain deprecated even if loading `xintfrac`. All these deprecated macros are destined to be removed at some future release.
A few macros got renamed (e.g. `\xintNot` became `\xintNOT`.) Former names emit a deprecation error and will get removed at some future release.
 - Release 1.2n of 2017/08/06 removed the `xintbinhex` dependencies upon `xintcore`; the package now depends upon, and loads, only `xintkernel`. The allowed maximal size for the inputs of the base conversion macros got increased. The speed got slightly improved.
 - Release 1.2m of 2017/07/31 rewrote entirely the `xintbinhex` module in the style of the techniques from 1.2 `xintcore`. The new macros expand faster but their inputs are now limited to a few thousand characters (the earlier routines, which dated back to 1.08 could handle (slowly) tens of thousands of digits.)
 - Release 1.2l of 2017/07/26 refactored the subtraction and also `\xintiiCmp` got a rewrite. It should certainly use `\pdfstrcmp` for dramatic speed-up but I do not want to have to worry about multi-engine usage.
Some utility routines in `xintcore` manipulating blocks of eight digits and still in $O(N^2)$ style got rewritten analogously to the 1.2i version of macros such as `\xintInc`. Also `\xintiNum` was revisited.
The arithmetic macros of `xintcore` and all macros of `xintfrac` using `\XINT_infrac` were made compatible with arguments using non-delimited `\the\numexpr` or `\the\mathcode` etc... But `\xintiiAbs` and `\xintiiOpp` were not modified (to avoid some overhead) as well as routines such as `\xintInc` which are primarily for internal usage.
 - Release 1.2i of 2016/12/13 rewrote some legacy macros like `\xintDSR` or `\xintDecSplit` in the style of the techniques of 1.2. But this means also that they got limited to about 22480 digits for the former and 19970 digits for the latter (this is with the input stack size at 5000 and

Contents

the maximal expansion depth at 10000.) This is not really an issue from the point of view of calling macros (such as `\xintTrunc`, `\xintRound`), because they usually had since 1.2 their own limitation at about 19950 digits from other code parts (such as division.) The macro `\xintXTrunc` (which is not f-expandable however) can produce tens of thousands of digits and it escapes these limitations. Old macros such as `\xintLength` are not limited either (incidentally it got a lifting in 1.2i.) The macros from `xinttools` (`\xintKeep`, `\xintTrim`, `\xintNthElt`) also are not limited (but slower.)

- Release 1.2 of 2015/10/10 entirely rewrote the core arithmetic routines located in `xintcore`. The parser of `xintexpr` got faster and the limitation at 5000 digits per number was removed.
- Extensive changes in release 1.1 of 2014/10/28 were located in `xintexpr`. Also with that release, packages `xintkernel` and `xintcore` were extracted from `xinttools` and `xint`, and `\xintAdd` was modified to not multiply denominators blindly.

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

Warning: pay attention when looking at the code to the catcode configuration as found in `\XINTD_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.

1 Package *xintkernel* implementation

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

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

First appeared as a separate package with release 1.1.

1.2i adds `\xintreplicate`, `\xintgobble`, `\xintLengthUpTo` and `\xintLastItem`, and improves the efficiency of `\xintLength`.

1.3b adds `\xintUniformDeviate`.

1.1 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

```

1 Package *xintkernel* implementation

```

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%
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   {%

```

1 Package *xintkernel* implementation

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

Other modules could possibly be loaded under a different catcode regime.
116 \def\XINTsetupcatcodes {% for use by other modules
117     \edef\XINT_restorecatcodes_endinput
118     {%
119         \XINT_restorecatcodes\noexpand\endinput %
120     }%
121     \XINT_setcatcodes
122 }%
```

1.2 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

1 Package *xintkernel* implementation

extra precautions.

1.09c uses e-TeX `\ifdefined`.

```
123 \ifdefined\ProvidesPackage
124 \let\XINT_providespackage\relax
125 \else
126 \def\XINT_providespackage #1#2[#3]%
127     {\immediate\write-1{Package: #2 #3}%
128      \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
129 \fi
130 \XINT_providespackage
131 \ProvidesPackage {xintkernel}%
132 [2018/05/18 1.3b Paraphernalia for the xint packages (JFB)]%
```

1.3 Constants

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

1.4 (WIP) `\xint_texuniformdeviate` and needed counts

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

1.5 Token management utilities

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

1 Package *xintkernel* implementation

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

1.6 “gob til” macros and UD style fork

```
209 \long\def\xint_gob_til_R #1\R {}%
210 \long\def\xint_gob_til_W #1\W {}%
211 \long\def\xint_gob_til_Z #1\Z {}%
```


1 Package `xintkernel` implementation

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

1.7 `\xint_afterfi`

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

1.8 `\xint_bye`, `\xint_Bye`

`\xint_Bye` is new with 1.2i for `\xintDSRr` and `\xintRound`. Also `\xint_bye_thenstop`.

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

1.9 `\xintdothis`, `\xintorthat`

New with 1.1. Public names without underscores with 1.2. Used as `\if..\xint_dothis{..}\fi` <multiple times> followed by `\xint_orthat{...}`. To be used with less probable things first.

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

1.10 `\xint_zapspace`s

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

```
\xint_zapspace foo<space>\xint_gobble_i
```

Will remove some brace pairs (but not spaces inside them). By the way the `\zap@spaces` of LaTeX2e handles unexpectedly things such as `\zap@spaces 1 {22} 3 4 \@empty` (spaces are not all removed). This does not happen with `\xint_zapspace`s.

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

1 Package *xintkernel* implementation

start with. Of course brace-stripping may occur. And this iterates: each time a #2 is removed, either we then have spaces and next #1 will be empty, or we have no spaces and #1 will end at the first space. Ultimately #2 will be `\xint_gobble_i`.

This is not really robust as it may switch the expansion order of macros, and the `\xint_zapspaces` token might end up being fetched up by a macro. But it is enough for our purposes, for example:

```
\the\numexpr \xint_zapspaces 1 2 \xint_gobble_i\relax
```

expands to 12, and not 12\relax.

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

Made long with 1.2i.

```
241 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
```

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

1.11 `\odef`, `\oodef`, `\fdef`

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

```
243 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
```

```
244 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
```

```
245     \expandafter\expandafter\expandafter#1%
```

```
246     \expandafter\expandafter\expandafter }%
```

```
247 \def\xintfdef #1#2%
```

```
248     {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&@#2}}%
```

```
249 \ifdefined\odef\else\let\odef\xintodef\fi
```

```
250 \ifdefined\oodef\else\let\oodef\xintoodef\fi
```

```
251 \ifdefined\fdef\else\let\fdef\xintfdef\fi
```

1.12 `\xintReverseOrder`

`\xintReverseOrder`: does NOT expand its argument.

Attention: removes brace pairs.

For contents with only digit tokens a faster reverse macro is provided as `\xintReverseDigits` from 1.2 `xintcore.sty`.

At 1.3a, `\XINT_factortens` from `xintfrac.sty` was redone in a faster `\numexpr`-expansion style, and it stopped using this. There remains only two calls from `xintgcd.sty` to `\XINT_rord_main`.

For comma separated items, 1.2g has (not user documented) `\xintCSVReverse` in `xinttools.sty`.

```
252 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
```

```
253 \long\def\xintreverseorder #1%
```

```
254 {%
```

```
255     \XINT_rord_main {}#1%
```

```
256     \xint:
```

```
257     \xint_bye\xint_bye\xint_bye\xint_bye
```

```
258     \xint_bye\xint_bye\xint_bye\xint_bye
```

```
259     \xint:
```

```
260 }%
```

```
261 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
```

```
262 {%
```

```
263     \xint_bye #9\XINT_rord_cleanup\xint_bye
```

```
264     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
```

```
265 }%
```

```
266 \def\XINT_rord_cleanup #1{%
```

```
267 \long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
```

```

268 {%
269   \expandafter#1\xint_gob_til_xint: ##1%
270 }}\XINT_rord_cleanup { }%

```

1.13 `\xintLength`

`\xintLength` does NOT expand its argument. See `\xintNthElt{0}` from `xinttools.sty` which f-expands its argument.

1.2g has (not user documented) `\xintCSVLength` in `xinttools.sty`.

1.2i has rewritten this venerable macro. New code is about 40% faster across all lengths. Was again slightly changed for 1.2j (cosmetic).

```

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

```

1.14 `\xintLastItem`

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

```

290 \def\xintLastItem {\romannumeral0\xintlastitem }%
291 \long\def\xintlastitem #1%
292 {%
293   \XINT_last_loop {}.#1%
294   {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%
295   {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
296   {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
297   {\xint:\XINT_last_loop_endg}{\xint:\XINT_last_loop_endh}\xint_bye
298 }%
299 \long\def\xint_last_loop #1.#2#3#4#5#6#7#8#9%
300 {%
301   \xint_gob_til_xint: #9%
302   {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
303   \XINT_last_loop {#9}.%
304 }%
305 \long\def\xint_last_loop_enda #1#2\xint_bye{ #1}%

```

1 Package *xintkernel* implementation

```
306 \long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
307 \long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
308 \long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
309 \long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
310 \long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
311 \long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
312 \long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%
```

1.15 `\xintLengthUpTo`

1.2i for use by `\xintKeep` and `\xintTrim`.

`\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))`.

Does not expand its second argument (it is used by `\xintKeep` and `\xintTrim` with already expanded argument). Not a user macro. 1.2j rewrote the ending and changed the loop interface. The argument `N` **must be non-negative**.

```
313 \def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
314 \long\def\xintlengthupto #1#2%
315 {%
316   \expandafter\XINT_lengthupto_loop
317   \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
318     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
319     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
320 }%
321 \def\XINT_lengthupto_loop_a #1%
322 {%
323   \xint_UDsignfork
324   #1\XINT_lengthupto_gt
325   -\XINT_lengthupto_loop
326   \krof #1%
327 }%
328 \long\def\XINT_lengthupto_gt #1\xint_bye.{-0}%
329 \long\def\XINT_lengthupto_loop #1.#2#3#4#5#6#7#8#9%
330 {%
331   \xint_gob_til_xint: #9\XINT_lengthupto_finish_a\xint:%
332   \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
333 }%
334 \def\XINT_lengthupto_finish_a\xint:\expandafter\XINT_lengthupto_loop_a
335   \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
336 {%
337   \expandafter\XINT_lengthupto_finish_b\the\numexpr #1-#9\xint_bye
338 }%
339 \def\XINT_lengthupto_finish_b #1#2.%
340 {%
341   \xint_UDsignfork
342   #1{-0}%
343   -{ #1#2}%
344   \krof
345 }%
```

1.16 `\xintreplicate`

Added with 1.2i. This is cloned from `\prg_replicate:nn` from `expl3`, 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 `|#1|` replications when input `#1` is negative and then activates an error triggering macro; here we simply do nothing when `#1` is negative.

When `#1` is already explicit digits (even `#1=0`, but non-negative) one can call the macro directly as `\romannumeralXINT_rep #1\endcsname {foo}` to skip the `\numexpr`.

Expansion must be triggered by a `\romannumeral`.

```

346 \def\xintreplicate#1%
347   {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
348 \def\XINT_replicate #1{\xint_UDsignfork
349     #1\XINT_rep_neg
350     -\XINT_rep
351     \krof #1}%
352 \long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
353 \def\XINT_rep #1{\csname XINT_rep_f#1\XINT_rep_a}%
354 \def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
355 \def\XINT_rep_\XINT_rep_a{\endcsname}%
356 \long\expandafter\def\csname XINT_rep_0\endcsname #1%
357   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
358 \long\expandafter\def\csname XINT_rep_1\endcsname #1%
359   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
360 \long\expandafter\def\csname XINT_rep_2\endcsname #1%
361   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
362 \long\expandafter\def\csname XINT_rep_3\endcsname #1%
363   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
364 \long\expandafter\def\csname XINT_rep_4\endcsname #1%
365   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
366 \long\expandafter\def\csname XINT_rep_5\endcsname #1%
367   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
368 \long\expandafter\def\csname XINT_rep_6\endcsname #1%
369   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
370 \long\expandafter\def\csname XINT_rep_7\endcsname #1%
371   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
372 \long\expandafter\def\csname XINT_rep_8\endcsname #1%
373   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
374 \long\expandafter\def\csname XINT_rep_9\endcsname #1%
375   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
376 \long\expandafter\def\csname XINT_rep_f0\endcsname #1%
377   {\xint_c_}%
378 \long\expandafter\def\csname XINT_rep_f1\endcsname #1%
379   {\xint_c_ #1}%
380 \long\expandafter\def\csname XINT_rep_f2\endcsname #1%
381   {\xint_c_ #1#1}%
382 \long\expandafter\def\csname XINT_rep_f3\endcsname #1%
383   {\xint_c_ #1#1#1}%
384 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
385   {\xint_c_ #1#1#1#1}%
386 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
```

```

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

```

1.17 `\xintgobble`

Added with 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.

Like for `\xintreplicate`, a `\romannumeral` is needed to trigger expansion.

I wrote in a similar spirit an `\xintcount`. But it proved slower than the upgraded 1.2i `\xintLength` in all the range up to thousands of tokens.

```

396 \def\xintgobble #1%
397   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
398 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
399 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
400 \def\XINT_gobble_b #1.#2%
401   {\expandafter\XINT_gobble_c
402     \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
403     \the\numexpr #2+\xint_c_i.#1.}%
404 \def\XINT_gobble_c #1.#2.#3.%
405   {\csname XINT_g#2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
406 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
407 \expandafter\let\csname XINT_g10\endcsname\endcsname
408 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
409 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
410 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
411 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
412 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
413 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
414 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
415 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
416 \expandafter\let\csname XINT_g20\endcsname\endcsname
417 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
418   {\endcsname}%
419 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
420   {\expandafter\noexpand\csname XINT_g21\endcsname}%
421 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
422   {\expandafter\noexpand\csname XINT_g22\endcsname}%
423 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
424   {\expandafter\noexpand\csname XINT_g23\endcsname}%
425 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
426   {\expandafter\noexpand\csname XINT_g24\endcsname}%
427 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
428   {\expandafter\noexpand\csname XINT_g25\endcsname}%
429 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
430   {\expandafter\noexpand\csname XINT_g26\endcsname}%

```

1 Package *xintkernel* implementation

```
431 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
432 {\expandafter\noexpand\csname XINT_g27\endcsname}%
433 \expandafter\let\csname XINT_g30\endcsname\endcsname
434 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
435 {\expandafter\noexpand\csname XINT_g28\endcsname}%
436 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
437 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
438 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
439 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
440 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
441 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
442 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
443 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
444 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
445 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
446 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
447 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
448 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
449 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
450 \expandafter\let\csname XINT_g40\endcsname\endcsname
451 \expandafter\edef\csname XINT_g41\endcsname
452 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
453 \expandafter\edef\csname XINT_g42\endcsname
454 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
455 \expandafter\edef\csname XINT_g43\endcsname
456 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
457 \expandafter\edef\csname XINT_g44\endcsname
458 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
459 \expandafter\edef\csname XINT_g45\endcsname
460 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
461 \expandafter\edef\csname XINT_g46\endcsname
462 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
463 \expandafter\edef\csname XINT_g47\endcsname
464 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
465 \expandafter\edef\csname XINT_g48\endcsname
466 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
467 \expandafter\let\csname XINT_g50\endcsname\endcsname
468 \expandafter\edef\csname XINT_g51\endcsname
469 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
470 \expandafter\edef\csname XINT_g52\endcsname
471 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
472 \expandafter\edef\csname XINT_g53\endcsname
473 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
474 \expandafter\edef\csname XINT_g54\endcsname
475 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
476 \expandafter\edef\csname XINT_g55\endcsname
477 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
478 \expandafter\edef\csname XINT_g56\endcsname
479 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
480 \expandafter\edef\csname XINT_g57\endcsname
481 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
482 \expandafter\edef\csname XINT_g58\endcsname
```

1 Package *xintkernel* implementation

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

1.18 (WIP) `\xintUniformDeviate`

1.3b. See user manual for related information.

```
501 \ifdefined\xint_texuniformdeviate
502   \expandafter\xint_firstoftwo
503 \else\expandafter\xint_secondoftwo
504 \fi
505 {%
506   \def\xintUniformDeviate#1%
507     {\the\numexpr\expandafter\XINT_uniformdeviate_sgnfork\the\numexpr#1\xint:}%
508   \def\XINT_uniformdeviate_sgnfork#1%
509     {%
510       \if-#1\XINT_uniformdeviate_neg\fi \XINT_uniformdeviate{ }#1%
511     }%
512   \def\XINT_uniformdeviate_neg\fi\XINT_uniformdeviate#1-%
513     {%
514       \fi-\numexpr\XINT_uniformdeviate\relax
515     }%
516   \def\XINT_uniformdeviate#1#2\xint:
517     {% (
518       \expandafter\XINT_uniformdeviate_a\the\numexpr%
519         -\xint_texuniformdeviate\xint_c_ii^vii%
520         -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
521         -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
522         -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
523         +\xint_texuniformdeviate#2\xint:/#2)*#2\xint:+#2\fi\relax#1%
524     }%
525   \def\XINT_uniformdeviate_a #1\xint:
526     {%
527       \expandafter\XINT_uniformdeviate_b\the\numexpr#1-(#1%
528     }%
529   \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
530 }%
```


1 Package *xintkernel* implementation

```
531 {%
532 \def\xintUniformDeviate#1%
533 {%
534     \the\numexpr
535     \XINT_expandableerror{No uniformdeviate at engine level, returning 0.}%
536     0\relax
537 }%
538 }%
```

1.19 `\xintMessage`, `\ifxintverbose`

1.2c added it for use by `\xintdefvar` and `\xintdeffunc` of `xintexpr`. 1.2e uses `\write128` rather than `\write16` for compatibility with future extended range of output streams, in LuaTeX in particular.

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

1.20 (WIP) Expandable error message

Incorporated in 1.2l release, but really belongs to next major release.

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. Utiliser `\FPE: ?` (mais ce n'est pas uniquement du « floating point »)

Always used in context where expansion was launched by a `\romannumeral0` or `\romannumeral`^^@`.

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

2 Package `xinttools` implementation

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

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`.

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  \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

```

2 Package `xinttools` implementation

```
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
```

2.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2018/05/18 1.3b 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!
```

2.3 `\xintgodef`, `\xintgoodef`, `\xintgfdef`

1.09i. For use in `\xintAssign`.

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

2.4 `\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 }%
```

2 Package *xinttools* implementation

```
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\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{ }%
```

2.5 `\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}%

```

2.6 \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{ }%

```

2.7 `\xintZapSpaces`

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

```

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

```

2.8 `\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}%

```

2.9 `\xintCSVtoList`, `\xintCSVtoListNonStripped`

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

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

```

141 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
142 \long\def\xintcsvtolist #1{\expandafter\xintApply
143     \expandafter\xintzapspaceb
144     \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%

```

2 Package *xinttools* implementation

```

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

```

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

```

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

```

2 Package *xinttools* implementation

```
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}}%
```

2.10 `\xintListWithSep`

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

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

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

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

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



```

232 \long\def\XINT_lws_e_iii\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6\xint_bye
233   { #2#1#3#1#4#1#5}%
234 \long\def\XINT_lws_e_ii\xint_bye\XINT_lws_loop_b #1#2#3#4#5\xint_bye
235   { #2#1#3#1#4}%
236 \long\def\XINT_lws_e_i\xint_bye\XINT_lws_loop_b #1#2#3#4\xint_bye
237   { #2#1#3}%
238 \long\def\XINT_lws_e\xint_bye\XINT_lws_loop_b #1#2#3\xint_bye
239   { #2}%

```

2.11 `\xintNthElt`

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

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

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

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

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

```

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

```

2 Package *xinttools* implementation

```
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{ }%
```

2.12 `\xintKeep`

First included in release 1.09m.

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

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

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

`\xintKeepNoExpand` does not expand the L argument.

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

```
285 \def\xintKeep          {\romannumeral0\xintkeep }%
286 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
287 \long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
288   \expandafter{\romannumeral`&&@#2}}%
289 \def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.}%
290 \def\XINT_keep_a #1%
291 {%
292   \xint_UDzerominusfork
293   #1-\XINT_keep_keepnone
294   0#1\XINT_keep_neg
295   0-{\XINT_keep_pos #1}%
296   \krof
297 }%
298 \long\def\XINT_keep_keepnone .#1{ }%
299 \long\def\XINT_keep_neg #1.#2%
300 {%
301   \expandafter\XINT_keep_neg_a\the\numexpr
302   #1-\numexpr\XINT_length_loop
303   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
304   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
305   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.#2%
306 }%
307 \def\XINT_keep_neg_a #1%
308 {%
```

2 Package *xinttools* implementation

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

2.13 `\xintKeepUnbraced`

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

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

Modified for 1.2i like `\xintKeep`.

```
351 \def\xintKeepUnbraced {\romannumeral0\xintkeepunbraced }%
352 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
```

2 Package *xinttools* implementation

```

353 \long\def\xintkeepunbraced #1#2%
354   {\expandafter\XINT_keepunbr_a\the\numexpr #1\expandafter.%
355     \expandafter{\romannumeral`&&@#2}}%
356 \def\xintkeepunbracednoexpand #1%
357   {\expandafter\XINT_keepunbr_a\the\numexpr #1.}%
358 \def\XINT_keepunbr_a #1%
359 {%
360   \xint_UDzerominusfork
361     #1-\XINT_keep_keeptime
362     0#1\XINT_keep_neg
363     0-{\XINT_keepunbr_pos #1}%
364   \krof
365 }%
366 \long\def\XINT_keepunbr_pos #1.#2%
367 {%
368   \expandafter\XINT_keepunbr_loop
369   \the\numexpr#1-\XINT_lengthupto_loop
370   #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
371     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
372     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
373   -\xint_c_viii.{}#2\xint_bye%
374 }%
375 \def\XINT_keepunbr_loop #1#2.%
376 {%
377   \xint_gob_til_minus#1\XINT_keepunbr_loop_end-%
378   \expandafter\XINT_keepunbr_loop
379   \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
380 }%
381 \long\def\XINT_keepunbr_loop_pickeight
382   #1#2#3#4#5#6#7#8#9{{#1#2#3#4#5#6#7#8#9}}%
383 \def\XINT_keepunbr_loop_end-\expandafter\XINT_keepunbr_loop
384   \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
385   {\csname XINT_keepunbr_end#1\endcsname}%
386 \long\expandafter\def\csname XINT_keepunbr_end1\endcsname
387   #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
388 \long\expandafter\def\csname XINT_keepunbr_end2\endcsname
389   #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
390 \long\expandafter\def\csname XINT_keepunbr_end3\endcsname
391   #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
392 \long\expandafter\def\csname XINT_keepunbr_end4\endcsname
393   #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
394 \long\expandafter\def\csname XINT_keepunbr_end5\endcsname
395   #1#2#3#4#5\xint_bye { #1#2#3#4}%
396 \long\expandafter\def\csname XINT_keepunbr_end6\endcsname
397   #1#2#3#4\xint_bye { #1#2#3}%
398 \long\expandafter\def\csname XINT_keepunbr_end7\endcsname
399   #1#2#3\xint_bye { #1#2}%
400 \long\expandafter\def\csname XINT_keepunbr_end8\endcsname
401   #1#2\xint_bye { #1}%

```

2.14 `\xintTrim`

First included in release 1.09m.

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

```

402 \def\xintTrim          {\romannumeral0\xinttrim }%
403 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
404 \long\def\xinttrim #1#2{\expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
405          \expandafter{\romannumeral`&&#2}}%
406 \def\xinttrimnoexpand #1{\expandafter\XINT_trim_a\the\numexpr #1.}%
407 \def\XINT_trim_a #1%
408 {%
409   \xint_UDzerominusfork
410     #1-\XINT_trim_trimnone
411     0#1\XINT_trim_neg
412     0-{\XINT_trim_pos #1}%
413   \krof
414 }%
415 \long\def\XINT_trim_trimnone .#1{ #1}%
416 \long\def\XINT_trim_neg #1.#2%
417 {%
418   \expandafter\XINT_trim_neg_a\the\numexpr
419   #1-\numexpr\XINT_length_loop
420   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
421   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
422   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
423   .{ }#2\xint_bye
424 }%
425 \def\XINT_trim_neg_a #1%
426 {%
427   \xint_UDsignfork
428     #1{\expandafter\XINT_keep_loop\the\numexpr-\xint_c_viii+}%
429     -\XINT_trim_trimall
430   \krof
431 }%
432 \def\XINT_trim_trimall#1{%
433 \def\XINT_trim_trimall {\expandafter#1\xint_bye}%
434 }\XINT_trim_trimall{ }%

```

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

```

435 \long\def\XINT_trim_pos #1.#2%
436 {%

```

2 Package *xinttools* implementation

```
437 \expandafter\XINT_trim_pos_done\expandafter\space
438 \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
439 #2\xint:\xint:\xint:\xint:\xint:%
440 \xint:\xint:\xint:\xint:\xint:%
441 \xint_bye
442 }%
443 \def\XINT_trim_loop #1#2.%
444 {%
445 \xint_gob_til_minus#1\XINT_trim_finish-%
446 \expandafter\XINT_trim_loop\the\numexpr#1#2\XINT_trim_loop_trimnine
447 }%
448 \long\def\XINT_trim_loop_trimnine #1#2#3#4#5#6#7#8#9%
449 {%
450 \xint_gob_til_xint: #9\XINT_trim_toofew\xint:-\xint_c_ix.%
451 }%
452 \def\XINT_trim_toofew\xint:{*\xint_c_}%
453 \def\XINT_trim_finish#1{%
454 \def\XINT_trim_finish-%
455 \expandafter\XINT_trim_loop\the\numexpr-##1\XINT_trim_loop_trimnine
456 {%
457 \expandafter\expandafter\expandafter#1%
458 \csname xint_gobble_\romannumeral\numexpr\xint_c_ix-##1\endcsname
459 }}\XINT_trim_finish{ }%
460 \long\def\XINT_trim_pos_done #1\xint:#2\xint_bye {#1}%
```

2.15 `\xintTrimUnbraced`

1.2a. Modified in 1.2i like `\xintTrim`

```
461 \def\xintTrimUnbraced {\romannumeral0\xinttrimunbraced }%
462 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
463 \long\def\xinttrimunbraced #1#2%
464 {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
465 \expandafter{\romannumeral`&&@#2}}%
466 \def\xinttrimunbracednoexpand #1%
467 {\expandafter\XINT_trimunbr_a\the\numexpr #1.}%
468 \def\XINT_trimunbr_a #1%
469 {%
470 \xint_UDzerominusfork
471 #1-\XINT_trim_trimnone
472 0#1\XINT_trimunbr_neg
473 0-{\XINT_trim_pos #1}%
474 \krof
475 }%
476 \long\def\XINT_trimunbr_neg #1.#2%
477 {%
478 \expandafter\XINT_trimunbr_neg_a\the\numexpr
479 #1-\numexpr\XINT_length_loop
480 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
481 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
482 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
483 .{#2\xint_bye
484 }%
```

```

485 \def\XINT_trimunbr_neg_a #1%
486 {%
487   \xint_UDsignfork
488     #1{\expandafter\XINT_keeppunbr_loop\the\numexpr-\xint_c_viii+%
489     -\XINT_trim_trimall
490   \krof
491 }%

```

2.16 `\xintApply`

`\xintApply` `{\macro}{a}{b}...{z}` returns `{\macro{a}}...{\macro{b}}` where each instance of `\macro` is *f*-expanded. The list itself is first *f*-expanded and may thus be a macro. Introduced with release 1.04.

```

492 \def\xintApply          {\romannumeral0\xintapply }%
493 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
494 \long\def\xintapply #1#2%
495 {%
496   \expandafter\XINT_apply\expandafter {\romannumeral`&&@#2}%
497   {#1}%
498 }%
499 \long\def\XINT_apply #1#2{\XINT_apply_loop_a }{#2}#1\xint_bye }%
500 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a }{#1}#2\xint_bye }%
501 \long\def\XINT_apply_loop_a #1#2#3%
502 {%
503   \xint_bye #3\XINT_apply_end\xint_bye
504   \expandafter
505   \XINT_apply_loop_b
506   \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
507 }%
508 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a }{#2{#1}}}%
509 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
510   \expandafter #1#2#3{ #2}%

```

2.17 `\xintApplyUnbraced`

`\xintApplyUnbraced` `{\macro}{a}{b}...{z}` returns `\macro{a}...{\macro{z}}` where each instance of `\macro` is *f*-expanded using `\romannumeral-`0`. The second argument may be a macro as it is itself also *f*-expanded. No braces are added: this allows for example a non-expandable `\def` in `\macro`, without having to do `\gdef`. Introduced with release 1.06b.

```

511 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
512 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
513 \long\def\xintapplyunbraced #1#2%
514 {%
515   \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
516   {#1}%
517 }%
518 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a }{#2}#1\xint_bye }%
519 \long\def\xintapplyunbracednoexpand #1#2%
520   {\XINT_applyunbr_loop_a }{#1}#2\xint_bye }%
521 \long\def\XINT_applyunbr_loop_a #1#2#3%
522 {%

```

```

523 \xint_bye #3\XINT_applyunbr_end\xint_bye
524 \expandafter\XINT_applyunbr_loop_b
525 \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
526 }%
527 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
528 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
529 \expandafter #1#2#3{ #2}%

```

2.18 \xintSeq

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

```

530 \def\xintSeq {\romannumeral0\xintseq }%
531 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
532 \def\XINT_seq_chkopt #1%
533 {%
534 \ifx [#1\expandafter\XINT_seq_opt
535 \else\expandafter\XINT_seq_noopt
536 \fi #1%
537 }%
538 \def\XINT_seq_noopt #1\xint_bye #2%
539 {%
540 \expandafter\XINT_seq\expandafter
541 {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
542 }%
543 \def\XINT_seq #1#2%
544 {%
545 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
546 \expandafter\xint_firstoftwo_thenstop
547 \or
548 \expandafter\XINT_seq_p
549 \else
550 \expandafter\XINT_seq_n
551 \fi
552 {#2}{#1}%
553 }%
554 \def\XINT_seq_p #1#2%
555 {%
556 \ifnum #1>#2
557 \expandafter\expandafter\expandafter\XINT_seq_p
558 \else
559 \expandafter\XINT_seq_e
560 \fi
561 \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
562 }%
563 \def\XINT_seq_n #1#2%
564 {%
565 \ifnum #1<#2
566 \expandafter\expandafter\expandafter\XINT_seq_n
567 \else
568 \expandafter\XINT_seq_e
569 \fi

```


2 Package *xinttools* implementation

```

570     \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
571 }%
572 \def\XINT_seq_e #1#2#3{ }%
573 \def\XINT_seq_opt [\xint_bye #1]#2#3%
574 {%
575     \expandafter\XINT_seqo\expandafter
576     {\the\numexpr #2\expandafter}\expandafter
577     {\the\numexpr #3\expandafter}\expandafter
578     {\the\numexpr #1}%
579 }%
580 \def\XINT_seqo #1#2%
581 {%
582     \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
583     \expandafter\XINT_seqo_a
584     \or
585     \expandafter\XINT_seqo_pa
586     \else
587     \expandafter\XINT_seqo_na
588     \fi
589     {#1}{#2}%
590 }%
591 \def\XINT_seqo_a #1#2#3{ {#1}}%
592 \def\XINT_seqo_o #1#2#3#4{ #4}%
593 \def\XINT_seqo_pa #1#2#3%
594 {%
595     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
596     \expandafter\XINT_seqo_o
597     \or
598     \expandafter\XINT_seqo_pb
599     \else
600     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
601     \fi
602     {#1}{#2}{#3}{#1}%
603 }%
604 \def\XINT_seqo_pb #1#2#3%
605 {%
606     \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
607 }%
608 \def\XINT_seqo_pc #1#2%
609 {%
610     \ifnum #1>#2
611         \expandafter\XINT_seqo_o
612     \else
613         \expandafter\XINT_seqo_pd
614     \fi
615     {#1}{#2}%
616 }%
617 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
618 \def\XINT_seqo_na #1#2#3%
619 {%
620     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
621     \expandafter\XINT_seqo_o

```

2 Package *xinttools* implementation

```
622 \or
623   \xint_afterfi{\expandafter\space\xint_gobble_iv}%
624 \else
625   \expandafter\XINT_seqo_nb
626 \fi
627 {#1}{#2}{#3}{#1}}%
628 }%
629 \def\XINT_seqo_nb #1#2#3%
630 {%
631   \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
632 }%
633 \def\XINT_seqo_nc #1#2%
634 {%
635   \ifnum #1<#2
636     \expandafter\XINT_seqo_o
637   \else
638     \expandafter\XINT_seqo_nd
639   \fi
640   {#1}{#2}}%
641 }%
642 \def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%
```

2.19 `\xintloop`, `\xintbreakloop`, `\xintbreakloopaddo`, `\xintloopskiptonext`

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

```
643 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
644 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
645   #1\xintloop_again\fi\xint_gobble_i {#1}}%
646 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%
647 \long\def\xintbreakloopaddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
648 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
649   #2\xintloop_again\fi\xint_gobble_i {#2}}%
```

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

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

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

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

2 Package *xinttools* implementation

```
661 \long\def\xintloopindex #1\xintloop_again\fi\xint_gobble_iii #2%
662     {#2#1\xintloop_again\fi\xint_gobble_iii {#2}}%
663 \long\def\xintbracedloopindex #1\xintloop_again\fi\xint_gobble_iii #2%
664     {{{#2}#1\xintloop_again\fi\xint_gobble_iii {#2}}}%
665 \long\def\xintouteriloopindex #1\xintloop_again
666     #2\xintloop_again\fi\xint_gobble_iii #3%
667     {#3#1\xintloop_again #2\xintloop_again\fi\xint_gobble_iii {#3}}%
668 \long\def\xintbracedouteriloopindex #1\xintloop_again
669     #2\xintloop_again\fi\xint_gobble_iii #3%
670     {{{#3}#1\xintloop_again #2\xintloop_again\fi\xint_gobble_iii {#3}}}%
671 \long\def\xintloopskipnext #1\xintloop_again\fi\xint_gobble_iii #2#3{%
672     \expandafter\xintloop_again_b \the\numexpr#2+#3.#3.}%
673 \long\def\xintloopskipandredo #1\xintloop_again\fi\xint_gobble_iii #2#3#4{%
674     #4\xintloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%
```

2.21 `\XINT_xflet`

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

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

```
708 \fi
709 }%
```

2.22 `\xintApplyInline`

1.09a: `\xintApplyInline\macro{a}{b}...{z}` has the same effect as executing `\macro{a}` and then applying again `\xintApplyInline` to the shortened list `{b}...{z}` until nothing is left. This is a non-expandable command which will result in quicker code than using `\xintApplyUnbraced`. It *f*-expands its second (list) argument first, which may thus be encapsulated in a macro.

Rewritten in 1.09c. *Nota bene*: uses catcode 3 Z as privated list terminator.

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

2.23 `\xintFor`, `\xintFor*`, `\xintBreakFor`, `\xintBreakForAndDo`

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

2 Package *xinttools* implementation

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

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

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

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

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

2 Package *xinttools* implementation

```

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

```

```

836 }%
837 \def\XINT_for_last?yes
838 {%
839 \let\xintifForLast\xint_firstoftwo
840 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
841 }%

```

2.24 `\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`).

```

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

```

2 Package *xinttools* implementation

```

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

```



```
931 \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
932 }%
```

2.25 `\xintForpair`, `\xintForthree`, `\xintForfour`

1.09c.

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

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

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

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

2 Package `xinttools` implementation

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

2.26 `\xintAssign`, `\xintAssignArray`, `\xintDigitsOf`

`\xintAssign {a}{b}..{z}\to\A\B...Z` resp. `\xintAssignArray {a}{b}..{z}\to\U`.
`\xintDigitsOf=\xintAssignArray`.

1.1c 2015/09/12 has (belatedly) corrected some "features" of `\xintAssign` which didn't like the case of a space right before the `"\to"`, or the case with the first token not an opening brace and the subsequent material containing brace groups. The new code handles gracefully these situations.

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

2 Package *xinttools* implementation

```

1018     \expandafter\XINT_assign_c
1019 \else\expandafter\XINT_assign_f
1020 \fi
1021 }%
1022 \long\def\XINT_assign_f #1\xint:\to #2%
1023 {%
1024     \XINT_assign_def #2{#1}%
1025 }%
1026 \long\def\XINT_assign_c #1%
1027 {%
1028     \def\xint_temp {#1}%
1029     \ifx\xint_temp\xint_bracedstopper
1030         \expandafter\XINT_assign_e
1031     \else
1032         \expandafter\XINT_assign_d
1033     \fi
1034 }%
1035 \long\def\XINT_assign_d #1\to #2%
1036 {%
1037     \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1038     \XINT_assign_c #1\to
1039 }%
1040 \def\XINT_assign_e #1\to {}%
1041 \def\xintRelaxArray #1%
1042 {%
1043     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1044     \escapechar -1
1045     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1046     \XINT_restoreescapechar
1047     \xintilooop [\csname\xint_arrayname 0\endcsname+-1]
1048     \global
1049     \expandafter\let\csname\xint_arrayname\xintilooopindex\endcsname\relax
1050     \ifnum \xintilooopindex > \xint_c_
1051     \repeat
1052     \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1053     \global\let #1\relax
1054 }%
1055 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1056     \XINT_flet_zapsp }%
1057 \def\XINT_assignarray_fork
1058 {%
1059     \let\XINT_assignarray_def\def
1060     \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1061         \else\expandafter\XINT_assignarray
1062     \fi
1063 }%
1064 \def\XINT_assignarray_opt [#1]%
1065 {%
1066     \ifcsname #1def\endcsname
1067         \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1068     \else
1069         \expandafter\let\expandafter\XINT_assignarray_def

```

2 Package *xinttools* implementation

```

1070             \csname xint#1def\endcsname
1071     \fi
1072     \XINT_assignarray
1073 }%
1074 \long\def\XINT_assignarray #1\to #2%
1075 {%
1076     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1077     \escapechar -1
1078     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1079     \XINT_restoreescapechar
1080     \def\xint_itemcount {0}%
1081     \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
1082     \csname\xint_arrayname 00\expandafter\endcsname
1083     \csname\xint_arrayname 0\expandafter\endcsname
1084     \expandafter {\xint_arrayname}#2%
1085 }%
1086 \long\def\XINT_assignarray_loop #1%
1087 {%
1088     \def\xint_temp {#1}%
1089     \ifx\xint_temp\xint_bracedstopper
1090         \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1091             \expandafter{\the\numexpr\xint_itemcount}%
1092         \expandafter\expandafter\expandafter\XINT_assignarray_end
1093     \else
1094         \expandafter\def\expandafter\xint_itemcount\expandafter
1095             {\the\numexpr\xint_itemcount+\xint_c_i}%
1096         \expandafter\XINT_assignarray_def
1097             \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1098             \expandafter{\xint_temp }%
1099         \expandafter\XINT_assignarray_loop
1100     \fi
1101 }%
1102 \def\XINT_assignarray_end #1#2#3#4%
1103 {%
1104     \def #4##1%
1105     {%
1106         \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1107     }%
1108     \def #1##1%
1109     {%
1110         \ifnum ##1<\xint_c_
1111             \xint_afterfi{\XINT_expandableerror{Array index negative: 0 > ##1} }%
1112         \else
1113             \xint_afterfi {%
1114                 \ifnum ##1>#2
1115                     \xint_afterfi
1116                     {\XINT_expandableerror{Array index beyond range: ##1 > #2} }%
1117                 \else\xint_afterfi
1118             {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1119             \fi}%
1120         \fi
1121     }%

```

```
1122 }%
1123 \let\xintDigitsOf\xintAssignArray
```

2.27 `\xintExpandArgs`

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

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

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

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

These macros will remain undocumented in the user manual:

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

-- crucial design elements may change in future:

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

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

2. the way to represent the empty list. I was tempted for matter of optimization and synchronization with *xintexpr* context to require the empty list to be always represented by a space token and to not let the macros admit a completely empty input. But there were complications so for the time being 1.2j does accept truly empty output (it is not distinguished from an input equal to a space token) and produces empty output for empty list. This means that the status of the «nil» object for the *xintexpr* parsers is not completely clarified (currently it is represented by a space token).

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

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

Kept items (with no leading spaces; but first item special as it will have lost a leading space due to f-expansion) will lose a brace pair under `\xintKeep:f:csv` if the first argument was positive and strictly less than the length of the list. This differs of course from `\xintKeep` (which always

braces items it outputs when used with positive first argument) and also from `\xintKeepUnbraced` in the case when the whole list is kept. Actually the case of singleton list is special, and brace removal will happen then.

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

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

2.28.1 `\xintLength:f:csv`

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

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

Must first check if empty list.

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

2.28.2 `\xintLengthUpTo:f:csv`

1.2j. `\xintLengthUpTo:f:csv{N}{comma-list}`. No ending comma. Returns -0 if `length>N`, else returns difference `N-length`. ****N must be non-negative!****

Attention to the dot after `\xint_bye` for the loop interface.

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

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

```

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

```

2.28.3 `\xintKeep:f:csv`

1.2g 2016/03/17. Redone for 1.2j with use of `\xintLengthUpTo:f:csv`. Same code skeleton as `\xintKeep` but handling comma separated but non terminated lists has complications. The `\xintKeep` in case of a negative #1 uses `\xintgobble`, we don't have that for comma delimited items, hence we do a special loop here (this style of loop is surely competitive with `xintgobble` for a few dozens items and even more). The loop knows before starting that it will not go too far.

```

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

```

2 Package *xinttools* implementation

```

1200 \xint_UDzerominusfork
1201     #1-\XINT_keep:f:csv_keepnone
1202     0#1\XINT_keep:f:csv_neg
1203     0-\XINT_keep:f:csv_pos #1}%
1204 \krof
1205 }%
1206 \long\def\XINT_keep:f:csv_keepnone .#1{,}%
1207 \long\def\XINT_keep:f:csv_neg #1.#2%
1208 {%
1209     \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
1210     \romannumeral0%
1211     \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
1212     #1-\numexpr\XINT_length:f:csv_a
1213     #2\xint:,\xint:,\xint:,\xint:,%
1214     \xint:,\xint:,\xint:,\xint:,\xint:,%
1215     \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1216     \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1217     .#2\xint_bye
1218 }%
1219 \def\XINT_keep:f:csv_neg_a #1%
1220 {%
1221     \xint_UDsignfork
1222     #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
1223     -\XINT_keep:f:csv_keepall
1224 \krof
1225 }%
1226 \def\XINT_keep:f:csv_keepall #1.{ }%
1227 \long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
1228 \def\XINT_keep:f:csv_trimloop #1#2.%
1229 {%
1230     \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
1231     \expandafter\XINT_keep:f:csv_trimloop
1232     \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1233 }%
1234 \long\def\XINT_keep:f:csv_trimloop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
1235 \def\XINT_keep:f:csv_trimloop_finish-%
1236     \expandafter\XINT_keep:f:csv_trimloop
1237     \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1238     {\csname XINT_trim:f:csv_finish#1\endcsname}%
1239 \long\def\XINT_keep:f:csv_pos #1.#2%
1240 {%
1241     \expandafter\XINT_keep:f:csv_pos_fork
1242     \romannumeral0\XINT_lengthupto:f:csv_a
1243     #1.#2\xint:,\xint:,\xint:,\xint:,%
1244     \xint:,\xint:,\xint:,\xint:,%
1245     \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1246     \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1247     .#1.{ }#2\xint_bye%
1248 }%
1249 \def\XINT_keep:f:csv_pos_fork #1#2.%
1250 {%
1251     \xint_UDsignfork

```


2 Package *xinttools* implementation

```
1252     #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1253     -\XINT_keep:f:csv_pos_keeppall
1254     \krof
1255 }%
1256 \long\def\XINT_keep:f:csv_pos_keeppall #1.#2#3\xint_bye{,#3}%
1257 \def\XINT_keep:f:csv_loop #1#2.%
1258 {%
1259     \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
1260     \expandafter\XINT_keep:f:csv_loop
1261     \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1262 }%
1263 \long\def\XINT_keep:f:csv_loop_pickeight
1264     #1#2,#3,#4,#5,#6,#7,#8,#9,{{#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
1265 \def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
1266     \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1267     {\csname XINT_keep:f:csv_end#1\endcsname}%
1268 \long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
1269     #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {#1,#2,#3,#4,#5,#6,#7,#8}%
1270 \long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
1271     #1#2,#3,#4,#5,#6,#7,#8\xint_bye {#1,#2,#3,#4,#5,#6,#7}%
1272 \long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname
1273     #1#2,#3,#4,#5,#6,#7\xint_bye {#1,#2,#3,#4,#5,#6}%
1274 \long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
1275     #1#2,#3,#4,#5,#6\xint_bye {#1,#2,#3,#4,#5}%
1276 \long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
1277     #1#2,#3,#4,#5\xint_bye {#1,#2,#3,#4}%
1278 \long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
1279     #1#2,#3,#4\xint_bye {#1,#2,#3}%
1280 \long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
1281     #1#2,#3\xint_bye {#1,#2}%
1282 \long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
1283     #1#2\xint_bye {#1}%
```

2.28.4 `\xintTrim:f:csv`

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

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

2 Package *xinttools* implementation

```

1300 \long\def\XINT_trim:f:csv_neg #1.#2%
1301 {%
1302   \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
1303   #1-\numexpr\XINT_length:f:csv_a
1304   #2\xint:,\xint:,\xint:,\xint:,%
1305   \xint:,\xint:,\xint:,\xint:,\xint:,%
1306   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1307   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1308   .{ }#2\xint_bye
1309 }%
1310 \def\XINT_trim:f:csv_neg_a #1%
1311 {%
1312   \xint_UDsignfork
1313   #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1314   -\XINT_trim:f:csv_trimall
1315   \krof
1316 }%
1317 \def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
1318 \long\def\XINT_trim:f:csv_pos #1.#2%
1319 {%
1320   \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
1321   \romannumeral0%
1322   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1323   #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1324   \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
1325 }%
1326 \def\XINT_trim:f:csv_loop #1#2.%
1327 {%
1328   \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
1329   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
1330 }%
1331 \long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1332 {%
1333   \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
1334 }%
1335 \def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
1336 \def\XINT_trim:f:csv_finish-%
1337   \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
1338 {%
1339   \csname XINT_trim:f:csv_finish#1\endcsname
1340 }%
1341 \long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
1342   #1,#2,#3,#4,#5,#6,#7,#8,{ }%
1343 \long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
1344   #1,#2,#3,#4,#5,#6,#7,{ }%
1345 \long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
1346   #1,#2,#3,#4,#5,#6,{ }%
1347 \long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
1348   #1,#2,#3,#4,#5,{ }%
1349 \long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
1350   #1,#2,#3,#4,{ }%
1351 \long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname

```

2 Package *xinttools* implementation

```
1352 #1,#2,#3,{ }%
1353 \long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname
1354 #1,#2,{ }%
1355 \long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
1356 #1,{ }%
1357 \expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
1358 \long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%
```

2.28.5 `\xintNthEltPy:f:csv`

Counts like Python starting at zero. Last refactored with 1.2j. Attention, makes currently no effort at removing leading spaces in the picked item.

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

2 Package *xinttools* implementation

```
1399 \long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
1400 \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%
1401 }\XINT_nthelt:f:csv_pos_done{ }%
```

This strange thing is in case the picked item was the last one, hence there was an ending `\xint:` (we could not put a comma earlier for matters of not confusing empty list with a singleton list), and we do this here to activate brace-stripping of item as all other items may be brace-stripped if picked. This is done for coherence. Of course, in the context of the `xintexpr.sty` parsers, there are no braces in list items...

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

2.28.6 `\xintReverse:f:csv`

1.2g. Contrarily to `\xintReverseOrder` from `xintkernel.sty`, this one expands its argument. Handles empty list too. 2016/03/17. Made `\long` for 1.2j.

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

2.28.7 `\xintFirstItem:f:csv`

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

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

2.28.8 `\xintLastItem:f:csv`

Added with 1.2k, based on and sharing code with `xintkernel`'s `\xintLastItem` from 1.2i. Output empty if input empty. `f`-expands its argument (hence first item, if not protected.) For use by `last()` in `\xintexpr`-essions with to some extent `\xintNewExpr` compatibility.

```

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

```

2.28.9 Public names for the undocumented csv macros

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

```

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

```

3 Package *xintcore* implementation

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

Got split off from *xint* with release 1.1.

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

The commenting continues (2018/05/18) 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.

3.1 Catcodes, ε - \TeX and reload detection

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

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

```

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

3 Package *xintcore* implementation

```
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
```

3.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2018/05/18 1.3b Expandable arithmetic on big integers (JFB)]%
```

3.3 (WIP!) Error conditions and exceptions

As per the [Mike Cowlishaw/IBM's General Decimal Arithmetic Specification](http://speleotrove.com/decimal/decarith.html)

<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.

3 Package *xintcore* implementation

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}}%
```


3 Package *xintcore* implementation

```
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_firstofone_thenstop{#5}%
83 }%
84 }%
85%% \let\xintUse\xint_firstofthree_thenstop % defined in xint.sty
86 \def\XINT_ifFlagRaised #1{%
87 \ifcsname XINT_#1Flag_ON\endcsname
88 \expandafter\xint_firstoftwo
89 \else
90 \expandafter\xint_secondoftwo
91 \fi}%
92 \def\XINT_resetFlag #1%
93 {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
94 \def\XINT_resetFlags {% WIP
95 \XINT_resetFlag{InvalidOperation}% also from DivisionUndefined
96 \XINT_resetFlag{DivisionByZero}%
97 \XINT_resetFlag{Underflow}% (\xintiiPow with negative exponent)
98 \XINT_resetFlag{Overflow}% not encountered so far in xint code 1.21
99 % .. others ..
100 }%
101 \def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
102%% NOT IMPLEMENTED! WORK IN PROGRESS! (ALL SIGNALS TRAPPED, NO HANDLERS USED)
103 \catcode`. 11
104 \let\XINT_Clamped.handler\xint_firstofone % WIP
105 \def\XINT_InvalidOperation.handler#1{NaN}% WIP
106 \def\XINT_ConversionSyntax.handler#1{NaN}% WIP
107 \def\XINT_DivisionByZero.handler#1{SignedInfinity(#1)}% WIP
108 \def\XINT_DivisionImpossible.handler#1{NaN}% WIP
109 \def\XINT_DivisionUndefined.handler#1{NaN}% WIP
110 \let\XINT_Inexact.handler\xint_firstofone % WIP
111 \def\XINT_InvalidContext.handler#1{NaN}% WIP
112 \let\XINT_Rounded.handler\xint_firstofone % WIP
113 \let\XINT_Subnormal.handler\xint_firstofone% WIP
114 \def\XINT_Overflow.handler#1{NaN}% WIP
115 \def\XINT_Underflow.handler#1{NaN}% WIP
116 \catcode`. 12
```

3.4 Counts for holding needed constants

```
117 \ifdefined\m@ne\let\xint_c_mone\m@ne
118 \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 \fi
119 \ifdefined\xint_c_x^viii\else
120 \csname newcount\endcsname\xint_c_x^viii \xint_c_x^viii 100000000
121 \fi
122 \ifdefined\xint_c_x^ix\else
123 \csname newcount\endcsname\xint_c_x^ix \xint_c_x^ix 1000000000
```

```

124 \fi
125 \newcount\xint_c_x^viii_mone      \xint_c_x^viii_mone      99999999
126 \newcount\xint_c_xii_e_viii      \xint_c_xii_e_viii     1200000000
127 \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.

3.5 `\XINT_cuz_small`

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

```

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

```

3.6 `\xintNum`

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.

```

133 \def\xintiNum {\romannumeral0\xintinum }%
134 \def\xintinum #1%
135 {%
136   \expandafter\XINT_num_cleanup\the\numexpr\expandafter\XINT_num_loop
137   \romannumeral`&&@#1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
138 }%
139 \def\xintNum {\romannumeral0\xintnum }%
140 \let\xintnum\xintinum
141 \def\XINT_num #1%
142 {%
143   \expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
144   #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
145 }%
146 \def\XINT_num_loop #1#2#3#4#5#6#7#8#9%
147 {%
148   \xint_gob_til_xint: #9\XINT_num_end\xint:
149   #1#2#3#4#5#6#7#8#9%
150   \ifnum \numexpr #1#2#3#4#5#6#7#8#9+\xint_c_ = \xint_c_

```

3 Package *xintcore* implementation

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.

```
151     \expandafter\XINT_num_loop
152     \else
```

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

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

3.7 `\xintiiSgn`

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

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

```
158 \def\xintiiSgn {\romannumeral0\xintiisgn }%
159 \def\xintiisgn #1%
160 {%
161     \expandafter\XINT_sgn \romannumeral`&&@#1\xint:
162 }%
163 \def\XINT_sgn #1#2\xint:
164 {%
165     \xint_UDzerominusfork
166     #1-{ 0}%
167     0#1{-1}%
168     0-{ 1}%
169     \krof
170 }%
171 \def\XINT_Sgn #1#2\xint:
172 {%
173     \xint_UDzerominusfork
174     #1-{0}%
175     0#1{-1}%
176     0-{1}%
177     \krof
178 }%
179 \def\XINT_cntSgn #1#2\xint:
180 {%
181     \xint_UDzerominusfork
182     #1-\xint_c_
183     0#1\xint_c_mone
184     0-\xint_c_i
185     \krof
186 }%
```

3.8 `\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".

3 Package *xintcore* implementation

```
187 \def\xintiiOpp {\romannumeral0\xintiiopp }%
188 \def\xintiiopp #1%
189 {%
190   \expandafter\XINT_opp \romannumeral`&&@#1%
191 }%
192 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
193 \def\XINT_opp #1%
194 {%
195   \xint_UDzerominusfork
196   #1-{\ 0}%      zero
197   0#1{ }%      negative
198   0-{-#1}%     positive
199   \krof
200 }%
```

3.9 `\xintiiAbs`

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

```
201 \def\xintiiAbs {\romannumeral0\xintiiabs }%
202 \def\xintiiabs #1%
203 {%
204   \expandafter\XINT_abs \romannumeral`&&@#1%
205 }%
206 \def\XINT_abs #1%
207 {%
208   \xint_UDsignfork
209   #1{ }%
210   -{ #1}%
211   \krof
212 }%
```

3.10 `\xintFDg`

FIRST DIGIT.

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

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

```
213 \def\xintFDg {\romannumeral0\xintfdg }%
214 \def\xintfdg #1{\expandafter\XINT_fdg \romannumeral`&&@#1\xint:\Z}%
215 \def\XINT_FDg #1%
216   {\romannumeral0\expandafter\XINT_fdg\romannumeral`&&\xintnum{#1}\xint:\Z }%
217 \def\XINT_fdg #1#2#3\Z
218 {%
219   \xint_UDzerominusfork
220   #1-{\ 0}%      zero
221   0#1{ #2}%     negative
222   0-{\ #1}%     positive
223   \krof
224 }%
```

3.11 `\xintLDg`

LAST DIGIT.

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

1.2o deprecates `\xintiiLDg`, gives to `\xintLDg` former meaning of `\xintiiLDg`.

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

```

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

```

3.12 `\xintDouble`

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

```

245 \def\xintDouble {\romannumeral0\xintdouble}%
246 \def\xintdouble #1{\expandafter\XINT_dbl_fork\romannumeral`&&@#1%
247     \xint_bye2345678\xint_bye*\xint_c_ii\relax}%
248 \def\XINT_dbl_fork #1%
249 {%
250     \xint_UDsignfork
251     #1\XINT_dbl_neg
252     -\XINT_dbl
253     \krof #1%
254 }%
255 \def\XINT_dbl_neg-{\expandafter-\romannumeral0\XINT_dbl}%
256 \def\XINT_dbl #1{%
257 \def\XINT_dbl ##1##2##3##4##5##6##7##8%
258     {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8\XINT_dbl_a}%
259 }\XINT_dbl{ }%
260 \def\XINT_dbl_a #1#2#3#4#5#6#7#8%
261     {\expandafter\XINT_dbl_e\the\numexpr 1#1#2#3#4#5#6#7#8\XINT_dbl_a}%
262 \def\XINT_dbl_e#1{* \xint_c_ii\if#13+\xint_c_i\fi\relax}%

```

3.13 `\xintHalf`

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

```

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

```

3.14 `\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.

```

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

```

3.15 `\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.

```

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

```

3.16 `\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.

```

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

```

3.17 `\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.

```

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

```

3 Package *xintcore* implementation

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

3.18 `\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.

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

Blocks of eight digits

The lingua of release 1.2.

3.19 `\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.

3 Package *xintcore* implementation

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.

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

3.20 `\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`

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

3.21 `\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.

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

3 Package *xintcore* implementation

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

3.22 `\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.

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

3.23 `\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.

```
414 \def\XINT_div_unsepQ_delim {\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\Z}%
415 \def\XINT_div_unsepQ
416 {%
417 \expandafter\XINT_div_unsepQ_x\the\numexpr0\XINT_unsep_loop
418 }%
419 \def\XINT_div_unsepQ_x #1{%
420 \def\XINT_div_unsepQ_x 0##1##2##3##4##5##6##7##8##9%
421 {%
422 \xint_gob_til_Z ##9\XINT_div_unsepQ_one\Z
423 \xint_gob_til_eightzeroes ##1##2##3##4##5##6##7##8\XINT_div_unsepQ_y 00000000%
424 \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax ##9%
425 }}\XINT_div_unsepQ_x{ }%
426 \def\XINT_div_unsepQ_y #1{%
427 \def\XINT_div_unsepQ_y ##1\relax ##2##3##4##5##6##7##8##9%
428 }%
```

```

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

```

3.24 \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 1.21, 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.

```

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

```

3.25 \XINT_zeroes_forviii

\romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}000001\W
produces a string of k 0's such that k+length(#1) is smallest bigger multiple of eight.

```

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

```

3.26 \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 1.21 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.

```

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

```

3.27 `\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`.

```
453 \def\XINT_sepbyviii_andcount
454 {%
455   \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
456 }%
457 \def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
458 {%
459   1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
460 }%
461 \def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
462 \def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_\xint:}%
463 \def\XINT_sepbyviii_andcount_b #1\xint:#2!#3!#4!#5!#6!#7!#8!#9!%
464 {%
465   #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
466   !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
467   #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
468   \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii\xint:%
469 }%
470 \def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
471   #2+\xint_c_viii\xint:#3#4\W {\expandafter\xint:\the\numexpr #2+#3\xint:}%
```

3.28 `\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`.

```
472 \def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
473 {%
474   \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
475 }%
476 \def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
477 {%
478   #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
479   1#1\expandafter\xint:\the\numexpr 1\XINT_rsepbyviii
```

3 Package *xintcore* implementation

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

3.29 `\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`).

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

3.30 `\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.

```
495 \def\XINT_sepandrev_andcount
496 {%
497   \expandafter\XINT_sepandrev_andcount_a\the\numexpr 1\XINT_rsepbyviii
498 }%
499 \def\XINT_sepandrev_andcount_a {\XINT_sepandrev_andcount_b 0!{}}%
500 \def\XINT_sepandrev_andcount_b #1!#2#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
501 {%
502   \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
503   \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_i!%
504   {#9!#8!#7!#6!#5!#4!#3!#2}%
505 }%
```

3 Package *xintcore* implementation

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

3.31 `\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.

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

3.32 `\XINT_unrevbyviii`

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

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

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

```
519 \def\XINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
520 {%
521   \xint_gob_til_R #9\XINT_unrevbyviii_a\R
522   \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
523 }%
524 \def\XINT_unrevbyviii_a#1{%
525 \def\XINT_unrevbyviii_a\R\XINT_unrevbyviii ##1##2\W
526   {\expandafter#1\romannumeral`&&\xint_gob_til_sc ##1}%
527 }\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`.

```
528 \def\XINT_smallunrevbyviii 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
529 {%
530   \expandafter\XINT_cuz_small\xint_gob_til_sc #8#7#6#5#4#3#2#1%
531 }%
```

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.)

3 Package *xintcore* implementation

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.

3.33 `\xintiiAdd`

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

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

3 Package *xintcore* implementation

```

573          \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
574  \X #1%
575 }%
576 \let\XINT_add_plusplus \XINT_add_pp_a
577 \def\XINT_add_pp_b #1\xint:#2\X #3\xint:
578 {%
579   \expandafter\XINT_add_checklengths
580   \the\numexpr #1\expandafter\xint:%
581   \romannumeral0\expandafter\XINT_sepandrev_andcount
582   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
583   #3\XINT_rsepyviii_end_A 2345678%
584   \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
585   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
586   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
587   1;!1;!1;!1;!1\W #21;!1;!1;!1;!1\W
588   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
589 }%

```

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

```

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

```

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

Call: \the\numexpr\XINT_add_a 2#11;!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).

3 Package *xintcore* implementation

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

3 Package *xintcore* implementation

```
666 \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
667 }%
668 \def\XINT_add_hi;%
669 \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W
670 {%
671 \XINT_add_k #1#3!%
672 }%
673 \def\XINT_add_i #1#2\xint:%
674 {%
675 1#2\expandafter!\the\numexpr\XINT_add_a #1%
676 }%

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

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

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

3.34 `\xintiiCmp`

Moved from `xint.sty` to `xintcore.sty` and rewritten for 1.2l.

1.2l's `\xintiiCmp` is robust against non terminated input.

1.2o deprecates `\xintCmp`, with `xintfrac` loaded it will get overwritten anyhow.

```
685 \def\xintiiCmp {\romannumeral0\xintiicmp }%
686 \def\xintiicmp #1\expandafter\XINT_iicmp\romannumeral`&&@#1\xint:%}
687 \def\XINT_iicmp #1#2\xint:#3%
688 {%
689 \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
690 }%
691 \def\XINT_icmp #1#2\xint:#3%
692 {%
693 \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
694 }%
695 \def\XINT_cmp_nfork #1#2%
696 {%
697 \xint_UDzerofork
698 #1\XINT_cmp_firstiszero
699 #2\XINT_cmp_secondiszero
700 0{}%
701 \krof
702 \xint_UDsignsfork
703 #1#2\XINT_cmp_minusminus
704 #1-\XINT_cmp_minusplus
705 #2-\XINT_cmp_plusminus
706 --\XINT_cmp_plusplus
707 \krof #1#2%
708 }%
```

3 Package *xintcore* implementation

```

709 \def\XINT_cmp_firstiszero #1\krof 0#2#3\xint:#4\xint:
710 {%
711   \xint_UDzerominusfork
712     #2-{\ 0}%
713     0#2{ 1}%
714     0-{\ -1}%
715   \krof
716 }%
717 \def\XINT_cmp_secondiszero #1\krof #20#3\xint:#4\xint:
718 {%
719   \xint_UDzerominusfork
720     #2-{\ 0}%
721     0#2{ -1}%
722     0-{\ 1}%
723   \krof
724 }%
725 \def\XINT_cmp_plusminus #1\xint:#2\xint:{ 1}%
726 \def\XINT_cmp_minusplus #1\xint:#2\xint:{ -1}%
727 \def\XINT_cmp_minusminus
728   --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}}}%
729 \def\XINT_cmp_plusplus #1#2#3\xint:
730 {%
731   \expandafter\XINT_cmp_pp
732   \the\numexpr\expandafter\XINT_sepbyviii_andcount
733   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
734   #2#3\XINT_sepbyviii_end 2345678\relax
735     \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
736     \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
737   #1%
738 }%
739 \def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
740 {%
741   \expandafter\XINT_cmp_checklengths
742   \the\numexpr #2\expandafter\xint:%
743   \the\numexpr\expandafter\XINT_sepbyviii_andcount
744   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
745   #3\XINT_sepbyviii_end 2345678\relax
746     \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
747     \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
748   #1;!1;!1;!1;!1;\W
749 }%
750 \def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
751 {%
752   \ifnum #1=#3
753     \expandafter\xint_firstoftwo
754   \else
755     \expandafter\xint_secondoftwo
756   \fi
757   \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;\W
758 }%
759 \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
760 {%

```

3 Package *xintcore* implementation

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

3.35 `\xintiiSub`

Entirely rewritten for 1.2.

Refactored at 1.2l. I was initially aiming at clinching some internal format of the type `1<8digits>!...!<8digits>!` for chaining the arithmetic operations (as a preliminary step to decided upon some internal format for `xintfrac` macros), thus I wanted to uniformize delimiters in particular and have some core macros inputting and outputting such formats. But the way division is implemented makes it currently very hard to obtain a satisfactory solution. For subtraction I got there almost, but there was added overhead and, as the core sub-routine still assumed the shorter number will be positioned first, one would need to record the length also in the basic internal format, or add the overhead to not make assumption on which one is shorter. I thus but back-tracked my steps but in passing I improved the efficiency (probably) in the worst case branch.

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

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

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

```
787 \def\xintiiSub {\romannumeral0\xintiiSub }%
788 \def\xintiiSub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\xint:}%
789 \def\XINT_iisub #1#2\xint:#3%
790 {%
791   \expandafter\XINT_sub_nfork\expandafter
```

3 Package *xintcore* implementation

```

792   #1\romannumeral`&&@#3\xint:#2\xint:
793 }%
794 \def\XINT_isub #1#2\xint:#3%
795 {%
796   \expandafter\XINT_sub_nfork\expandafter
797   #1\romannumeral0\xintnum{#3}\xint:#2\xint:
798 }%
799 \def\XINT_sub_nfork #1#2%
800 {%
801   \xint_UDzerofork
802   #1\XINT_sub_firstiszero
803   #2\XINT_sub_secondiszero
804   0{}}%
805   \krof
806   \xint_UDsignsfork
807   #1#2\XINT_sub_minusminus
808   #1-\XINT_sub_minusplus
809   #2-\XINT_sub_plusminus
810   --\XINT_sub_plusplus
811   \krof #1#2%
812 }%
813 \def\XINT_sub_firstiszero #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
814 \def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
815 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
816 \def\XINT_sub_plusplus #1#2%
817   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
818 \def\XINT_sub_minusplus #1#2%
819   {\expandafter-\romannumeral0\XINT_add_pp_a { }#2}%
820 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a { }{}}%
821 \def\XINT_sub_mm_a #1#2#3\xint:
822 {%
823   \expandafter\XINT_sub_mm_b
824   \romannumeral0\expandafter\XINT_sepandrev_andcount
825   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
826   #2#3\XINT_rsepbyviii_end_A 2345678%
827   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
828   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
829   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
830   \X #1%
831 }%
832 \def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
833 {%
834   \expandafter\XINT_sub_checklengths
835   \the\numexpr #1\expandafter\xint:%
836   \romannumeral0\expandafter\XINT_sepandrev_andcount
837   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
838   #3\XINT_rsepbyviii_end_A 2345678%
839   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
840   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
841   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
842   1;!1;!1;!1;!1\W
843   #21;!1;!1;!1;!1\W

```

3 Package *xintcore* implementation

```

844 1;!1\R!1\R!1\R!1\R!%
845 1\R!1\R!1\R!1\R!\W
846 }%
847 \def\XINT_sub_checklengths #1\xint:#2\xint:%
848 {%
849 \ifnum #2>#1
850 \expandafter\XINT_sub_exchange
851 \else
852 \expandafter\XINT_sub_aa
853 \fi
854 }%
855 \def\XINT_sub_exchange #1\W #2\W
856 {%
857 \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
858 }%
859 \def\XINT_sub_aa
860 {%
861 \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
862 }%

```

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`.

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

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

Call: `\the\numexpr`

```

\XINT_sub_a 1#11;!1;!1;!1;!1\W
#21;!1;!1;!1;!1\W

```

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

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

`\numexpr` governed expansion stops with various possibilities:

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

Various type of post actions are then needed:

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

3 Package *xintcore* implementation

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.

```
864 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
865 {%
866   \XINT_sub_b
867   #1!#6!#2!#7!#3!#8!#4!#9!%
868   #5\W
869 }%
```

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

```
870 \def\XINT_sub_b #1#2#3#4!#5!%
871 {%
872   \xint_gob_til_sc #3\XINT_sub_bi ;%
873   \expandafter\XINT_sub_c\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
874 }%
875 \def\XINT_sub_c 1#1#2\xint:%
876 {%
877   1#2\expandafter!\the\numexpr\XINT_sub_d #1%
878 }%
879 \def\XINT_sub_d #1#2#3#4!#5!%
880 {%
881   \xint_gob_til_sc #3\XINT_sub_di ;%
882   \expandafter\XINT_sub_e\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
883 }%
884 \def\XINT_sub_e 1#1#2\xint:%
885 {%
886   1#2\expandafter!\the\numexpr\XINT_sub_f #1%
887 }%
888 \def\XINT_sub_f #1#2#3#4!#5!%
889 {%
890   \xint_gob_til_sc #3\XINT_sub_fi ;%
891   \expandafter\XINT_sub_g\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
892 }%
893 \def\XINT_sub_g 1#1#2\xint:%
894 {%
895   1#2\expandafter!\the\numexpr\XINT_sub_h #1%
896 }%
897 \def\XINT_sub_h #1#2#3#4!#5!%
898 {%
899   \xint_gob_til_sc #3\XINT_sub_hi ;%
900   \expandafter\XINT_sub_i\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
901 }%
902 \def\XINT_sub_i 1#1#2\xint:%
903 {%
904   1#2\expandafter!\the\numexpr\XINT_sub_a #1%
905 }%
906 \def\XINT_sub_bi;%
907   \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3\xint:%
908   #4!#5!#6!#7!#8!#9!\W
```

3 Package *xintcore* implementation

```

909 {%
910   \XINT_sub_k #1#2!#5!#7!#9!%
911 }%
912 \def\XINT_sub_di;%
913   \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3\xint:
914   #4!#5!#6!#7!#8\W
915 {%
916   \XINT_sub_k #1#2!#5!#7!%
917 }%
918 \def\XINT_sub_fi;%
919   \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
920   #4!#5!#6\W
921 {%
922   \XINT_sub_k #1#2!#5!%
923 }%
924 \def\XINT_sub_hi;%
925   \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3\xint:
926   #4\W
927 {%
928   \XINT_sub_k #1#2!%
929 }%

B terminated. Have we reached the end of A (necessarily at least as long as B) ? (we are computing
A-B, digits of B come first).
  If not, then we are certain that even if there is carry it will not propagate beyond the end of
  A. But it may propagate far transforming chains of 00000000 into 99999999, and if it does go to the
  final block which possibly is just 1<00000001>!, we will have those eight zeros to clean up.
  If A and B have the same length (in base 10^8) then arbitrarily many zeros might have to be cleaned
  up, and if A<B, the whole result will have to be complemented first.

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

937 \def\XINT_sub_l_carry 1#1!{\ifcase #1
938   \expandafter \XINT_sub_l_Id
939   \or \expandafter \XINT_sub_l_Ic
940   \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
941 \def\XINT_sub_l_Ib #1;#2\W {-\xint_c_i+#1;!1\XINT_sub_fix_none!}%
942 \def\XINT_sub_l_Ic 1#1!1#2#3!#4;#5\W
943 {%
944   \xint_gob_til_sc #2\XINT_sub_l_Ica;%
945   1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
946 }%

```

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`

3 Package *xintcore* implementation

```

947 \def\XINT_sub_l_Ica#1\W
948 {%
949     1;!1\XINT_sub_fix_cuz!%
950     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
951     \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
952 }%
953 \def\XINT_sub_l_Id 1#1!%
954     {199999999\expandafter!\the\numexpr \XINT_sub_l_Id_a}%
955 \def\XINT_sub_l_Id_a 1#1!{\ifcase #1
956     \expandafter \XINT_sub_l_Id
957     \or \expandafter \XINT_sub_l_Id_b
958     \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
959 \def\XINT_sub_l_Id_b 1#1!#2#3!#4;#5\W
960 {%
961     \xint_gob_til_sc #2\XINT_sub_l_Ida;%
962     1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
963 }%
964 \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.

```

965 \def\XINT_sub_p;\XINT_sub_l #1#2\W #3\W
966 {%
967     \xint_UDzerofork
968     #1{1;!1\XINT_sub_fix_neg!%
969     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
970     \xint_bye2345678\xint_bye1099999988\relax}% A - B, B > A
971     0{1;!1\XINT_sub_fix_cuz!%
972     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
973     \krof
974     \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
975 }%

```

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.

```

976 \def\XINT_sub_fix_none;{\XINT_cuz_small}%
977 \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.

```

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

```

3 Package *xintcore* implementation

```
985 {%
986   \expandafter\XINT_sub_comp_clean
987   \the\numexpr \xint_c_xi_e_viii_mone-#1#2#3#4#5#6#7#8\XINT_sub_comp_loop
988 }%
```

#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.

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

3.36 `\xintiiMul`

Completely rewritten for 1.2.

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

```
990 \def\xintiiMul {\romannumeral0\xintiimul }%
991 \def\xintiimul #1%
992 {%
993   \expandafter\XINT_iimul\romannumeral`&&@#1\xint:
994 }%
995 \def\XINT_iimul #1#2\xint:#3%
996 {%
997   \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
998 }%
```

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

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

3 Package *xintcore* implementation

```

1026      \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1027      \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1028      \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1029  \W #1%
1030 }%
1031 \def\XINT_mul_pre_b #1\xint:#2\W #3\xint:
1032 {%
1033   \expandafter\XINT_mul_checklengths
1034   \the\numexpr #1\expandafter\xint:%
1035   \romannumeral0\expandafter\XINT_sepandrev_andcount
1036   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
1037   #3\XINT_rsepyviii_end_A 2345678%
1038   \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1039   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1040   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1041   1;! \W #21;!%
1042   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
1043 }%

```

Cooking recipe, 2015/10/05.

```

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

```

3 Package *xintcore* implementation

```
1076 {\fi\fi\XINT_mul_start #31;!\W #2}%
```

```
1077 \def\XINT_mul_start
```

```
1078 {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1;!\W}%
```

```
1079 \def\XINT_mul_out
```

```
1080 {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
```

Call:

```
\the\numexpr \XINT_mul_loop 100000000!1;!\W #1;!\W #2;!
```

where #1 and #2 are (globally reversed) blocks $1\langle 8\rangle!$. Its is generally more efficient if #1 is the shorter one, but a better recipe is implemented in `\XINT_mul_checklengths`. One may call `\XINT_mul_loop` directly (but multiplication by zero will produce many 100000000! blocks on output).

Ends after having produced: $1\langle 8\rangle!\dots 1\langle 8\rangle!1;!$. 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.

```
1081 \def\XINT_mul_loop #1\W #2\W 1#3!%
```

```
1082 {%
```

```
1083 \xint_gob_til_sc #3\XINT_mul_e ;%
```

```
1084 \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
```

```
1085 #1\W #2\W
```

```
1086 }%
```

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

```
1087 \def\XINT_mul_a #1\W #2\W
```

```
1088 {%
```

```
1089 \expandafter\XINT_mul_b\the\numexpr
```

```
1090 \XINT_add_a \xint_c_ii #21;!1;!1;!\W #11;!1;!1;!\W\W
```

```
1091 }%
```

```
1092 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
```

```
1093 \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 $1\langle 8\rangle$ carry $\langle 4\rangle$ high digits $\langle 4\rangle$ low digits $\langle 8\rangle$.

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

```
1094 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
```

```
1095 {%
```

```
1096 \expandafter\XINT_minimulwc_b
```

```
1097 \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
```

```
1098 #3*#4#5#6#7+#2*#8\xint:
```

```
1099 #2*#4#5#6#7\xint:%
```

```
1100 }%
```

3 Package *xintcore* implementation

```

1101 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1102 {%
1103   \expandafter\XINT_minimulwc_c
1104   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1105 }%
1106 \def\XINT_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1107 {%
1108   1#6#7\expandafter!%
1109   \the\numexpr\expandafter\XINT_smallmul_a
1110   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1111 }%
1112 \def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!}%
1113 \def\XINT_smallmul_a #1\xint:#2\xint:#3!#4!%
1114 {%
1115   \xint_gob_til_sc #4\XINT_smallmul_e;%
1116   \XINT_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1117 }%
1118 \def\XINT_smallmul_e;\XINT_minimulwc_a 1#1\xint:#2;#3!%
1119   {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!;!}%
1120 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%

```

```

1121 \def\XINT_verysmallmul #1\xint:#2!#3!%
1122 {%
1123   \xint_gob_til_sc #3\XINT_verysmallmul_e;%
1124   \expandafter\XINT_verysmallmul_a
1125   \the\numexpr #2*#3+#1\xint:#2!%
1126 }%
1127 \def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
1128   #1+#2#3\xint:#4!%
1129 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
1130 \def\XINT_verysmallmul_f #1!1{1\relax}%
1131 \def\XINT_verysmallmul_a #1#2\xint:%
1132 {%
1133   \unless\ifnum #1#2<\xint_c_x^ix
1134     \expandafter\XINT_verysmallmul_bi\else
1135     \expandafter\XINT_verysmallmul_bj\fi
1136     \the\numexpr \xint_c_x^ix+#1#2\xint:%
1137 }%
1138 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
1139 \def\XINT_verysmallmul_cj 1#1#2\xint:%
1140   {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:%}
1141 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1142   {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2\xint:%}

```

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

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

```

1143 \def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
1144 {%
1145   \expandafter\XINT_minimul_b
1146   \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%

```

3 Package *xintcore* implementation

```

1147 }%
1148 \def\XINT_minimul_b #1#2#3#4#5\xint:#6\xint:%
1149 {%
1150     \expandafter\XINT_minimul_c
1151     \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%
1152 }%
1153 \def\XINT_minimul_c #1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1154 {%
1155     #6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1156 }%

```

3.37 \xintiiDivision

Completely rewritten for 1.2.

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

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

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

```

1157 \def\xintiiDivision {\romannumeral0\xintiidivision }%
1158 \def\xintiidivision #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:%}
1159 \def\XINT_iidivision #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1160     \romannumeral`&&@#3\xint:#2\xint:%}

```

On regarde les signes de A et de B.

```

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

```

3 Package *xintcore* implementation

```

1177 {%
1178   \xint_UDsignfork
1179       #1\xINT_iidivision_aneg
1180       -{\XINT_iidivision_apos #1}%
1181   \krof
1182 }%
```

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.

```

1183 \def\xINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
1184 \def\xINT_iidivision_aneg #1\xint:#2\xint:
1185   {\expandafter
1186     \XINT_iidivision_aneg_b\romannumeral0\xINT_div_prepare {#1}{#2}{#1}}%
1187 \def\xINT_iidivision_aneg_b #1#2{\if0\xINT_Sgn #2\xint:
1188     \expandafter\xINT_iidivision_aneg_rzero
1189     \else
1190     \expandafter\xINT_iidivision_aneg_rpos
1191     \fi {#1}{#2}}%
1192 \def\xINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1193 \def\xINT_iidivision_aneg_rpos #1%
1194 {%
1195   \expandafter\xINT_iidivision_aneg_end\expandafter
1196     {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1197 }%
1198 \def\xINT_iidivision_aneg_end #1#2#3%
1199 {%
1200   \expandafter\xint_exchangetwo_keepbraces
1201   \expandafter{\romannumeral0\xINT_sub_mm_a {}}{#3\xint:#2\xint:}{#1}% r-> b-r
1202 }%
```

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.

```

1203 \def\xINT_div_prepare #1%
1204 {%
1205   \XINT_div_prepare_a #1\R\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1206 }%
1207 \def\xINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1208 {%
1209   \xint_gob_til_R #9\xINT_div_prepare_small\R
1210   \XINT_div_prepare_b #9%
1211 }%
```

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).

```

1212 \def\xINT_div_prepare_small\R #1!#2%
1213 {%
1214   \ifcase #2
1215   \or\expandafter\xINT_div_BisOne
```

3 Package *xintcore* implementation

```

1216 \or\expandafter\XINT_div_BisTwo
1217 \else\expandafter\XINT_div_small_a
1218 \fi {#2}%
1219 }%
1220 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1221 \def\XINT_div_BisTwo #1#2%
1222 {%
1223 \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1224 \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1225 }%
1226 \def\XINT_div_BisTwo_a #1#2%
1227 {%
1228 \expandafter{\romannumeral0\XINT_half
1229 #2\xint_bye\xint_Bye345678\xint_bye
1230 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1231 }%

```

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.

```

1232 \def\XINT_div_small_a #1#2%
1233 {%
1234 \expandafter\XINT_div_small_b
1235 \the\numexpr #1/\xint_c_ii\expandafter
1236 \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
1237 \romannumeral0%
1238 \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
1239 #2\XINT_sepbyviii_Z_end 2345678\relax
1240 }%

```

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

```

1241 \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 $A < 10^8$.

```

1242 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1243 {%
1244 \xint_gob_til_R #9\XINT_div_smallsmall\R
1245 \expandafter\XINT_div_dosmallldiv
1246 \the\numexpr\expandafter\XINT_sepbyviii_Z
1247 \romannumeral0\XINT_zeroes_forviii
1248 #1#2#3#4#5#6#7#8#9%
1249 }%

```

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

```

1250 \def\XINT_div_smallsmall\R
1251 \expandafter\XINT_div_dosmallldiv
1252 \the\numexpr\expandafter\XINT_sepbyviii_Z
1253 \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1254 {{\XINT_div_dosmallsmall}}{#1}}%
1255 \def\XINT_div_dosmallsmall #1\xint:1#2!#3%

```


3 Package *xintcore* implementation

```

1256 {%
1257   \expandafter\XINT_div_smallsmallend
1258   \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
1259 }%
1260 \def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
1261   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%

```

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

```

1262 \def\XINT_div_dosmalldiv
1263   {{\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 $8N$.

```

1264 \def\XINT_div_prepare_b
1265   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1266 \def\XINT_div_prepare_c #1!%
1267 {%
1268   \XINT_div_prepare_d #1.00000000!{#1}%
1269 }%
1270 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1271 {%
1272   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1273 }%
1274 \def\XINT_div_prepare_e #1!#2!#3#4%
1275 {%
1276   \XINT_div_prepare_f #4#3\X {#1}{#3}%
1277 }%

```

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.

```

1278 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1279 {%
1280   \expandafter\XINT_div_prepare_g
1281   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1282   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1283   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1284   \xint:\romannumeral0\XINT_sepandrev_andcount
1285   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1286   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1287   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1288   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1289   \X
1290 }%
1291 \def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
1292 {%

```

3 Package *xintcore* implementation

```

1293 \expandafter\XINT_div_prepare_h
1294 \the\numexpr\expandafter\XINT_sepbyviii_andcount
1295 \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1296 #8#7\XINT_sepbyviii_end 2345678\relax
1297 \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1298 \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
1299 {#1}{#2}{#3}{#4}{#5}{#6}%
1300 }%
1301 \def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6#7#8%
1302 {%
1303 \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1304 }%

```

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.

```

1305 \def\XINT_div_start_a #1#2%
1306 {%
1307 \ifnum #1 < #2
1308 \expandafter\XINT_div_zeroQ
1309 \else
1310 \expandafter\XINT_div_start_b
1311 \fi
1312 {#1}{#2}%
1313 }%
1314 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1315 {%
1316 \expandafter\XINT_div_zeroQ_end
1317 \romannumeral0\XINT_unsep_cuzsmall
1318 #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
1319 }%
1320 \def\XINT_div_zeroQ_end #1\xint:#2%
1321 {\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»

```

1322 \def\XINT_div_start_b #1#2#3#4#5#6%
1323 {%
1324 \expandafter\XINT_div_finish\the\numexpr
1325 \XINT_div_start_c {#2}\xint:#3\xint:{#6}{#1}{#2}{#4}{#5}{#6}%
1326 }%
1327 \def\XINT_div_finish
1328 {%
1329 \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1330 }%
1331 \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».

```

1332 \def\XINT_div_finish_b #1%
1333 {%
1334 \if0#1%
1335 \expandafter\XINT_div_finish_bRzero

```

3 Package *xintcore* implementation

```

1336 \else
1337 \expandafter\XINT_div_finish_bRpos
1338 \fi
1339 #1%
1340 }%
1341 \def\XINT_div_finish_bRzero 0\xint:#1#2{{#1}{0}}%
1342 \def\XINT_div_finish_bRpos #1\xint:#2#3%
1343 {%
1344 \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3\xint:{#2}%
1345 }%
1346 \def\XINT_div_cleanR #100000000\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.

```

1347 \def\XINT_div_start_c #1%
1348 {%
1349 \ifnum #1>\xint_c_vi
1350 \expandafter\XINT_div_start_ca
1351 \else
1352 \expandafter\XINT_div_start_cb
1353 \fi {#1}%
1354 }%
1355 \def\XINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
1356 {%
1357 \expandafter\XINT_div_start_c\expandafter
1358 {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!\xint:%
1359 }%
1360 \def\XINT_div_start_cb #1%
1361 {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
1362 \def\XINT_div_start_c_i #1\xint:#2!%
1363 {\XINT_div_start_c_ #1#2!\xint:}%
1364 \def\XINT_div_start_c_ii #1\xint:#2!#3!%
1365 {\XINT_div_start_c_ #1#2!#3!\xint:}%
1366 \def\XINT_div_start_c_iii #1\xint:#2!#3!#4!%
1367 {\XINT_div_start_c_ #1#2!#3!#4!\xint:}%
1368 \def\XINT_div_start_c_iv #1\xint:#2!#3!#4!#5!%
1369 {\XINT_div_start_c_ #1#2!#3!#4!#5!\xint:}%
1370 \def\XINT_div_start_c_v #1\xint:#2!#3!#4!#5!#6!%
1371 {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!\xint:}%
1372 \def\XINT_div_start_c_vi #1\xint:#2!#3!#4!#5!#6!#7!%
1373 {\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».

```

1374 \def\XINT_div_start_c_ 1#1!#2\xint:#3\xint:#4#5#6%
1375 {%
1376 \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1377 }%

```

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

3 Package *xintcore* implementation

```

1378 \def\XINT_div_I_a #1#2%
1379 {%
1380   \expandafter\XINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
1381 }%
1382 \def\XINT_div_I_b #1%
1383 {%
1384   \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1385 }%

```

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».

```

1386 \def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1387 \def\XINT_div_I_c #1\xint:#2#3%
1388 {%
1389   \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
1390 }%

```

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

```

1391 \def\XINT_div_I_da #1\xint:%
1392 {%
1393   \ifnum #1>\xint_c_ix
1394     \expandafter\XINT_div_I_dp
1395   \else
1396     \ifnum #1<\xint_c_
1397       \expandafter\expandafter\expandafter\XINT_div_I_dN
1398     \else
1399       \expandafter\expandafter\expandafter\XINT_div_I_db
1400     \fi
1401   \fi
1402 }%

```

attention très mauvaises notations avec _b et _db.

```

1403 \def\XINT_div_I_dN #1\xint:%
1404 {%
1405   \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
1406 }%
1407 \def\XINT_div_I_db #1\xint:#2#3#4#5%
1408 {%
1409   \expandafter\XINT_div_I_dc\expandafter #1%
1410   \romannumeral0\expandafter\XINT_div_sub\expandafter
1411     {\romannumeral0\XINT_rev_nounsep {}}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1412     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1413   \Z {#4}{#5}%
1414 }%

```

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

```

1415 \def\XINT_div_I_dc #1#2%
1416 {%
1417   \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1418   #1#2%

```

3 Package *xintcore* implementation

```

1419 }%
1420 \def\XINT_div_I_dd #1-\Z
1421 {%
1422   \if #11\expandafter\XINT_div_I_dz\fi
1423   \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
1424 }%
1425 \def\XINT_div_I_dz #1XX#2#3#4%
1426 {%
1427   1#4\XINT_div_I_g {#2}%
1428 }%
1429 \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»

1430 \def\XINT_div_I_dP #1\xint:#2#3#4#5#6%
1431 {%
1432   1#6+#1\expandafter\XINT_div_I_g\expandafter
1433   {\romannumeral0\expandafter\XINT_div_sub\expandafter
1434     {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1435     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1436   }%
1437 }%

  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»

1438 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1439 {%
1440   \expandafter !\the\numexpr
1441   \ifnum#2=#3
1442     \expandafter\XINT_div_exittofinish
1443   \else
1444     \expandafter\XINT_div_I_h
1445   \fi
1446   {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
1447 }%

  {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»

1448 \def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
1449 {%
1450   1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
1451   \romannumeral0\XINT_div_unsepR #2#3%
1452   \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R\xint:
1453 }%

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

```

3 Package *xintcore* implementation

```

1454 \def\XINT_div_I_h #1\xint:#2!#3\xint:#4#5%
1455 {%
1456   \XINT_div_II_b #1#2!\xint:{#5}{#4}{#3}{#5}%
1457 }%

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

1458 \def\XINT_div_II_b #1#2!#3!%
1459 {%
1460   \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1461   \XINT_div_II_c #1{1#2}{#3}%
1462 }%

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

1463 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
1464 {%
1465   \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1466 }%

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

1467 \def\XINT_div_II_c #1#2#3#4%
1468 {%
1469   \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1470   #1\xint:#2!#3!#4!{#1}{#2}{#3!#4!%
1471 }%
1472 \def\XINT_div_xmini #1%
1473 {%
1474   \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1475 }%
1476 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1477 {%
1478   \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1479 }%
1480 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1481 {%
1482   \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1483 }%

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

1484 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000\xint:50000000!#1!#2!{#1!}%

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

1485 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%
1486 {%
1487   \expandafter\XINT_div_II_e
1488   \romannumeral0\expandafter\XINT_div_sub\expandafter
1489   {\romannumeral0\XINT_rev_nounsep }{#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1490   {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#91;!}%
1491   \xint:{#6}{#7}{#9}{#1#2#3#4#5}%
1492 }%

```

3 Package *xintcore* implementation

$\alpha.x', y, B, q_1, \{\{x'y\}, x, K, L\}, \alpha', B, \llcorner$. Attention la soustraction spéciale doit maintenir les blocs $1 < 8$!

```
1493 \def\XINT_div_II_e 1#1!%
1494 {%
1495   \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
1496   \XINT_div_II_f 1#1!%
1497 }%

10000000! alpha sur K chiffres. #2=x', #3=y, #4=B, #5=q1, #6=\{\{x'y\}, x, K, L\}, #7=alpha', B\llcorner -> \{x'y\}x, K, L
(à diminuer de 1), \{\alpha sur K\}B\{q_1\}\{\alpha'\}B\llcorner
```

```
1498 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%
1499 {%
1500   \XINT_div_II_k #6\{#1\}\{#4\}\{#5\}%
1501 }%
```

$1 < a_1 > ! 1 < a_2 > !$, α (sur $K+1$ blocs de 8). $x', y, B, q_1, \{\{x'y\}, x, K, L\}, \alpha', B, \llcorner$.
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`.

```
1502 \def\XINT_div_II_f #1!#2!#3\xint:%
1503 {%
1504   \XINT_div_II_fa \{#1!#2!\}\{#1!#2!#3\}%
1505 }%
1506 \def\XINT_div_II_fa #1#2#3#4%
1507 {%
1508   \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1\{#2\}%
1509 }%
```

$\#1=q, \#2=\alpha (K+4), \#3=B, \#4=q_1, \{\{x'y\}, x, K, L\}, \alpha', BQ\llcorner -> 1$ puis nouveau q sur 8 chiffres. nouvel α sur K blocs, $B, \{\{x'y\}, x, K, L\}, \alpha', BQ\llcorner$

```
1510 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1511 {%
1512   \expandafter \XINT_div_II_h
1513   \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1514   \xint:\expandafter\expandafter\expandafter
1515   \{\expandafter\xint_gob_til_exclam
1516   \romannumeral0\expandafter\XINT_div_sub\expandafter
1517   \{\romannumeral0\XINT_rev_nounsep \}\#6\R!\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1518   \{\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#71;!}\}%
1519   \{#7\}%
1520 }%
```

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

```
1521 \def\XINT_div_II_h 1#1\xint:#2#3#4%
1522 {%
1523   \XINT_div_II_k #4\{#2\}\{#3\}\{#1\}%
1524 }%
```

3 Package *xintcore* implementation

$\{x'y\}x,K,L$ à diminuer de 1, $\alpha, B\{q\}\alpha',B\langle c\rangle \rightarrow$ nouveau $L.K,x',y,x,\alpha.B,q,\alpha',B,\langle c\rangle$
 $\rightarrow\{LK\{x'y\}x\},x,a,\alpha.B,q,\alpha',B,\langle c\rangle$

```
1525 \def\XINT_div_II_k #1#2#3#4#5%
1526 {%
1527   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
1528 }%
1529 \def\XINT_div_II_l #1\xint:#2#3#4#5!#6!%
1530 {%
1531   \XINT_div_II_m {{#1}{#2}{{#3}{#4}}{#5}{{#5}{#6}}1#6!%
1532 }%
```

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

```
1533 \def\XINT_div_II_m #1#2#3#4\xint:#5#6%
1534 {%
1535   \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1536 }%
```

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

```
1537 \def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1538 {%
1539   \expandafter\XINT_div_minimulwc_b
1540   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
1541 }%
1542 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1543 {%
1544   \expandafter\XINT_div_minimulwc_c
1545   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1546 }%
1547 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1548 {%
1549   1#6#7\expandafter!%
1550   \the\numexpr\expandafter\XINT_div_smallmul_a
1551   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1552 }%
1553 \def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
1554 {%
1555   \xint_gob_til_sc #4\XINT_div_smallmul_e;%
1556   \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1557 }%
1558 \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.

```
1559 \def\XINT_div_verysmallmul #1%
1560   {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1561 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
1562   {1\relax #1100000000!}%
```


3 Package *xintcore* implementation

```

1563 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1564 {%
1565   \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1566   \expandafter\XINT_div_verysmallmul_b
1567   \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1568 }%
1569 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1570   {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1571 \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.

```

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

```

3 Package *xintcore* implementation

```

1612 {%
1613     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1614 }%
1615 \def\XINT_div_sub_h #1#2#3!#4!%
1616 {%
1617     \xint_gob_til_sc #4\XINT_div_sub_hi ;%
1618     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1619 }%
1620 \def\XINT_div_sub_i 1#1#2\xint:%
1621 {%
1622     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1623 }%
1624 \def\XINT_div_sub_bi;%
1625     \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8!#9!;! \W
1626 {%
1627     \XINT_div_sub_l #1#2!#5!#7!#9!%
1628 }%
1629 \def\XINT_div_sub_di;%
1630     \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8\W
1631 {%
1632     \XINT_div_sub_l #1#2!#5!#7!%
1633 }%
1634 \def\XINT_div_sub_fi;%
1635     \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3\xint:#4!#5!#6\W
1636 {%
1637     \XINT_div_sub_l #1#2!#5!%
1638 }%
1639 \def\XINT_div_sub_hi;%
1640     \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3\xint:#4\W
1641 {%
1642     \XINT_div_sub_l #1#2!%
1643 }%
1644 \def\XINT_div_sub_l #1%
1645 {%
1646     \xint_UDzerofork
1647     #1{-2\relax}%
1648     0\XINT_div_sub_r
1649     \krof
1650 }%
1651 \def\XINT_div_sub_r #1!%
1652 {%
1653     -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1654 }%

```

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

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

```

1655 \def\XINT_sdiv_out #1;!#2!%
1656     {\expandafter

```

3 Package *xintcore* implementation

```

1657 {\romannumeral0\XINT_unsep_cuzsmall
1658 #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
1659 {#2}}%

```

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

```

1660 \def\XINT_smalldivx_a #1\xint:1#2!1#3!%
1661 {%
1662 \expandafter\XINT_smalldivx_b
1663 \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
1664 }%
1665 \def\XINT_smalldivx_b #1#2!%
1666 {%
1667 \if0#1\else
1668 \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1669 \XINT_smalldiv_c #1#2!%
1670 }%
1671 \def\XINT_smalldiv_c #1!#2\xint:#3!#4!%
1672 {%
1673 \expandafter\XINT_smalldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
1674 }%

```

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`.

```

1675 \def\XINT_smalldiv_d #1!#2!1#3#4!%
1676 {%
1677 \xint_gob_til_sc #3\XINT_smalldiv_end ;%
1678 \XINT_smalldiv_e #1!#2!1#3#4!%
1679 }%
1680 \def\XINT_smalldiv_end;\XINT_smalldiv_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 $< 10^8$.

```

1681 \def\XINT_smalldiv_e #1!#2\xint:#3!%
1682 {%
1683 \expandafter\XINT_smalldiv_f\the\numexpr
1684 \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2\xint:#3!#1!%
1685 }%
1686 \def\XINT_smalldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
1687 {%
1688 \xint_gob_til_zero #1\XINT_smalldiv_fz 0%
1689 \expandafter\XINT_smalldiv_g
1690 \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
1691 }%
1692 \def\XINT_smalldiv_fz 0%
1693 \expandafter\XINT_smalldiv_g\the\numexpr\XINT_minimul_a
1694 9999\xint:9999!#1!99999999!#2!0!1#3!%
1695 {%
1696 \XINT_smalldiv_i \xint:#3!\xint_c_!#2!%

```

3 Package *xintcore* implementation

```

1697 }%
1698 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1699 {%
1700   \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1701 }%
1702 \def\XINT_smallldiv_h 1#1#2\xint:#3!#4!%
1703 {%
1704   \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1705 }%
1706 \def\XINT_smallldiv_i #1\xint:#2!#3!#4\xint:#5!%
1707 {%
1708   \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4\xint:#5!%
1709 }%
1710 \def\XINT_smallldiv_j #1!#2!%
1711 {%
1712   \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1713   #1!%
1714 }%

```

On boucle vers `\XINT_smallldiv_d`.

```

1715 \def\XINT_smallldiv_k #1!#2!#3\xint:#4!%
1716 {%
1717   \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
1718 }%

```

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>`.

```

1719 \def\XINT_div_mini #1\xint:#2!1#3!%
1720 {%
1721   \expandafter\XINT_div_mini_a\the\numexpr
1722   \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
1723 }%

```

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.

```

1724 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7\xint:#8!%
1725 {%
1726   \xint_gob_til_zero #1\XINT_div_mini_w 0%
1727   \expandafter\XINT_div_mini_b
1728   \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#7!#2#3#4#5#6!#7\xint:#8!%
1729 }%
1730 \def\XINT_div_mini_w 0%
1731   \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a

```

3 Package *xintcore* implementation

```
1732 9999\xint:9999!#1!99999999!#2\xint:#3!00000000!#4!%
1733 {%
1734 \xint_c_x^viii_mone+(#4+#3)/#2!%
1735 }%
1736 \def\xINT_div_mini_b #1!#2!#3!#4!#5!#6!%
1737 {%
1738 \expandafter\xINT_div_mini_c
1739 \the\numexpr #1#6-#1\xint:#2!#5!#3!#4!%
1740 }%
1741 \def\xINT_div_mini_c #1#2\xint:#3!#4!%
1742 {%
1743 \expandafter\xINT_div_mini_d
1744 \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1745 }%
1746 \def\xINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
1747 {%
1748 \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1749 }%
```

Derived arithmetic

3.38 `\xintiiQuo`, `\xintiiRem`

```
1750 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1751 \def\xintiiRem {\romannumeral0\xintiirem }%
1752 \def\xintiiquo
1753 {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1754 \def\xintiirem
1755 {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
```

3.39 `\xintiiDivRound`

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

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

```
1756 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1757 \def\xintiidivround #1{\expandafter\xINT_iidivround\romannumeral`&&@#1\xint:}%
1758 \def\xINT_iidivround #1#2\xint:#3%
1759 {\expandafter\xINT_iidivround_a\expandafter #1%
1760 \romannumeral0\xintnum{#3}\xint:#2\xint:}%
1761 \def\xINT_iidivround #1#2\xint:#3%
1762 {\expandafter\xINT_iidivround_a\expandafter #1\romannumeral`&&@#3\xint:#2\xint:}%
1763 \def\xINT_iidivround_a #1#2% #1 de A, #2 de B.
1764 {%
1765 \if0#2\xint_dothis{\XINT_iidivround_divbyzero#1#2}\fi
1766 \if0#1\xint_dothis{\XINT_iidivround_aiszero}\fi
1767 \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1768 \xint_orthat{\XINT_iidivround_bpos #1#2}%
1769 }%
1770 \def\xINT_iidivround_divbyzero #1#2#3\xint:#4\xint:
1771 {\XINT_signalcondition{DivisionByZero}{Division of #1#4 by #2#3}}{0}}%
1772 \def\xINT_iidivround_aiszero #1\xint:#2\xint:{ 0}%
1773 \def\xINT_iidivround_bpos #1%
```

```

1774 {%
1775   \xint_UDsignfork
1776     #1{\xintiiopp\XINT_iidivround_pos {}}%
1777     -{\XINT_iidivround_pos #1}%
1778   \krof
1779 }%
1780 \def\XINT_iidivround_bneg #1%
1781 {%
1782   \xint_UDsignfork
1783     #1{\XINT_iidivround_pos {}}%
1784     -{\xintiiopp\XINT_iidivround_pos #1}%
1785   \krof
1786 }%
1787 \def\XINT_iidivround_pos #1#2\xint:#3\xint:
1788 {%
1789   \expandafter\expandafter\expandafter\XINT_dsrr
1790   \expandafter\xint_firstoftwo
1791   \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1792   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1793 }%

```

3.40 \xintiiDivTrunc

1.21: \xintiiDivTrunc made robust against non terminated input.

```

1794 \def\xintiiDivTrunc {\romannumeral0\xintiiidivtrunc }%
1795 \def\xintiiidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral`&&@#1\xint:}%
1796 \def\XINT_iidivtrunc #1#2\xint:#3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1797   \romannumeral`&&@#3\xint:#2\xint:}%
1798 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1799 {%
1800   \if0#2\xint_dothis{\XINT_iidivtrunc_divbyzero#1#2}\fi
1801   \if0#1\xint_dothis\XINT_iidivtrunc_aiszero\fi
1802   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1803   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1804 }%

```

Attention to not move DivRound code beyond that point.

```

1805 \let\XINT_iidivtrunc_divbyzero\XINT_iidivround_divbyzero
1806 \let\XINT_iidivtrunc_aiszero \XINT_iidivround_aiszero
1807 \def\XINT_iidivtrunc_bpos #1%
1808 {%
1809   \xint_UDsignfork
1810     #1{\xintiiopp\XINT_iidivtrunc_pos {}}%
1811     -{\XINT_iidivtrunc_pos #1}%
1812   \krof
1813 }%
1814 \def\XINT_iidivtrunc_bneg #1%
1815 {%
1816   \xint_UDsignfork
1817     #1{\XINT_iidivtrunc_pos {}}%
1818     -{\xintiiopp\XINT_iidivtrunc_pos #1}%

```

```

1819 \krof
1820 }%
1821 \def\XINT_iidivtrunc_pos #1#2\xint:#3\xint:
1822   {\expandafter\xint_firstoftwo_thenstop
1823    \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

3.41 \xintiiModTrunc

Renamed from `\xintiiMod` to `\xintiiModTrunc` at 1.2p.

```

1824 \def\xintiiModTrunc {\romannumeral0\xintiiModTrunc }%
1825 \def\xintiiModTrunc #1{\expandafter\XINT_iimodtrunc\romannumeral`&&@#1\xint:}%
1826 \def\XINT_iimodtrunc #1#2\xint:#3{\expandafter\XINT_iimodtrunc_a\expandafter #1%
1827   \romannumeral`&&@#3\xint:#2\xint:}%
1828 \def\XINT_iimodtrunc_a #1#2% #1 de A, #2 de B.
1829 {%
1830   \if0#2\xint_dothis{\XINT_iimodtrunc_divbyzero#1#2}\fi
1831   \if0#1\xint_dothis\XINT_iimodtrunc_aiszero\fi
1832   \if-#2\xint_dothis{\XINT_iimodtrunc_bneg #1}\fi
1833   \xint_orthat{\XINT_iimodtrunc_bpos #1#2}%
1834 }%

```

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

```

1835 \let\XINT_iimodtrunc_divbyzero\XINT_iidivround_divbyzero
1836 \let\XINT_iimodtrunc_aiszero \XINT_iidivround_aiszero
1837 \def\XINT_iimodtrunc_bpos #1%
1838 {%
1839   \xint_UDsignfork
1840     #1{\xintiiopp\XINT_iimodtrunc_pos {}}%
1841     -{\XINT_iimodtrunc_pos #1}%
1842   \krof
1843 }%
1844 \def\XINT_iimodtrunc_bneg #1%
1845 {%
1846   \xint_UDsignfork
1847     #1{\xintiiopp\XINT_iimodtrunc_pos {}}%
1848     -{\XINT_iimodtrunc_pos #1}%
1849   \krof
1850 }%
1851 \def\XINT_iimodtrunc_pos #1#2\xint:#3\xint:
1852   {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
1853    {#2}{#1#3}}%

```

3.42 \xintiiDivMod

1.2p. Associated with floored division like Python's `divmod`.

```

1854 \def\xintiiDivMod {\romannumeral0\xintiiDivMod }%
1855 \def\xintiiDivMod #1{\expandafter\XINT_iidivmod\romannumeral`&&@#1\xint:}%
1856 \def\XINT_iidivmod #1#2\xint:#3{\expandafter\XINT_iidivmod_a\expandafter #1%
1857   \romannumeral`&&@#3\xint:#2\xint:}%

```

3 Package *xintcore* implementation

```
1858 \def\XINT_iidivmod_a #1#2% #1 de A, #2 de B.
1859 {%
1860   \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
1861   \if0#1\xint_dothis\XINT_iidivmod_aiszero\fi
1862   \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
1863   \xint_orthat{\XINT_iidivmod_bpos #1#2}%
1864 }%
1865 \def\XINT_iidivmod_divbyzero #1#2\xint:#3\xint:
1866 {%
1867   \XINT_signalcondition{DivisionByZero}{Division by #2 of #1#3}{}%
1868   {{0}{0}}% à revoir...
1869 }%
1870 \def\XINT_iidivmod_aiszero #1#2\xint:#3\xint:{{0}{0}}%
1871 \def\XINT_iidivmod_bneg #1%
1872 {%
1873   \expandafter\XINT_iidivmod_bneg_finish
1874   \romannumeral0\xint_UDsignfork
1875     #1{\XINT_iidivmod_bpos {}}%
1876     -{\XINT_iidivmod_bpos {-#1}}%
1877   \krof
1878 }%
1879 \def\XINT_iidivmod_bneg_finish#1#2%
1880 {%
1881   \expandafter\xint_exchangetwo_keepbraces\expandafter
1882   {\romannumeral0\xintiiopt#2}{#1}%
1883 }%
1884 \def\XINT_iidivmod_bpos #1#2\xint:#3\xint: {\xintiidivision{#1#3}{#2}}%
```

3.43 *\xintiiDivFloor*

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

```
1885 \def\xintiiDivFloor {\romannumeral0\xintiiDivFloor}%
1886 \def\xintiiDivFloor {\expandafter\xint_firstoftwo_thenstop
1887   \romannumeral0\xintiidivmod}%
```

3.44 *\xintiiMod*

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

```
1888 \def\xintiiMod {\romannumeral0\xintiiMod}%
1889 \def\xintiiMod {\expandafter\xint_secondoftwo_thenstop
1890   \romannumeral0\xintiidivmod}%
```

3.45 *\xintiiSqr*

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

```
1891 \def\xintiiSqr {\romannumeral0\xintiiSqr}%
1892 \def\xintiiSqr #1%
1893 {%
1894   \expandafter\XINT_sqr\romannumeral0\xintiiabs{#1}\xint:
```


3 Package *xintcore* implementation

And it did only a `\numexpr` on the exponent, contradicting the documentation related to the "i" convention in names.

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

```
1934 \def\xintiiPow {\romannumeral0\xintiipow }%
1935 \def\xintiipow #1#2%
1936 {%
1937   \expandafter\xint_pow\the\numexpr #2\expandafter
1938   .\romannumeral`&&@#1\xint:
1939 }%
1940 \def\xint_pow #1.#2##3\xint:
1941 {%
1942   \xint_UDzerominusfork
1943   #2-\XINT_pow_AisZero
1944   0#2\XINT_pow_Aneg
1945   0-{\XINT_pow_Apos #2}%
1946   \krof {#1}%
1947 }%
1948 \def\XINT_pow_AisZero #1#2\xint:
1949 {%
1950   \ifcase\XINT_cntSgn #1\xint:
1951     \xint_afterfi { 1}%
1952   \or
1953     \xint_afterfi { 0}%
1954   \else
1955     \xint_afterfi
1956     {\XINT_signalcondition{DivisionByZero}{Zero to power #1}{0}}%
1957   \fi
1958 }%
1959 \def\XINT_pow_Aneg #1%
1960 {%
1961   \ifodd #1
1962     \expandafter\XINT_opp\romannumeral0%
1963   \fi
1964   \XINT_pow_Apos {}{#1}%
1965 }%
1966 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1967 \def\XINT_pow_Apos_a #1#2#3%
1968 {%
1969   \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
1970   \XINT_pow_AatleastTwo {#1}#2#3%
1971 }%
1972 \def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
1973 {%
1974   \ifcase #2
1975     \xintError:thiscannothappen
1976   \or \expandafter\XINT_pow_AisOne
1977   \else\expandafter\XINT_pow_AatleastTwo
1978   \fi {#1}#2\xint:
1979 }%
1980 \def\XINT_pow_AisOne #1\xint:{ 1}%
1981 \def\XINT_pow_AatleastTwo #1%
1982 {%
```

3 Package *xintcore* implementation

```

1983 \ifcase\XINT_cntSgn #1\xint:
1984   \expandafter\XINT_pow_BisZero
1985 \or
1986   \expandafter\XINT_pow_I_in
1987 \else
1988   \expandafter\XINT_pow_BisNegative
1989 \fi
1990 {#1}%
1991 }%
1992 \def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}{Inverse power
1993   can not be represented by an integer}}{0}}%
1994 \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).

```

1995 \def\XINT_pow_I_in #1#2\xint:
1996 {%
1997   \expandafter\XINT_pow_I_loop
1998   \the\numexpr #1\expandafter\xint:%
1999   \romannumeral0\expandafter\XINT_sepandrev
2000   \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\R{10}0000001\W
2001   #2\XINT_rsepyviii_end_A 2345678%
2002   \XINT_rsepyviii_end_B 2345678\relax XX%
2003   \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
2004   1;!\W
2005   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
2006 }%
2007 \def\XINT_pow_I_loop #1\xint:%
2008 {%
2009   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
2010   \ifodd #1
2011     \expandafter\XINT_pow_II_in
2012   \else
2013     \expandafter\XINT_pow_I_squareit
2014   \fi #1\xint:%
2015 }%
2016 \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 "butcheckifsmall" 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.

```

2017 \def\XINT_pow_I_squareit #1\xint:#2\W%
2018 {%
2019   \expandafter\XINT_pow_I_loop
2020   \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2021   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2022 }%
2023 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
2024 {%
2025   \xint_gob_til_sc #2\XINT_pow_mul_small;%
2026   \XINT_mul_loop 100000000!1;!\W #1!1#2%
2027 }%

```

3 Package *xintcore* implementation

```
2028 \def\XINT_pow_mul_small;\XINT_mul_loop
2029     100000000!1;! \W #1!1;! \W
2030 {%
2031     \XINT_smallmul #1!%
2032 }%
2033 \def\XINT_pow_II_in #1\xint:#2\W
2034 {%
2035     \expandafter\XINT_pow_II_loop
2036     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2037     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
2038 }%
2039 \def\XINT_pow_II_loop #1\xint:%
2040 {%
2041     \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
2042     \ifodd #1
2043         \expandafter\XINT_pow_II_odda
2044     \else
2045         \expandafter\XINT_pow_II_even
2046     \fi #1\xint:%
2047 }%
2048 \def\XINT_pow_II_exit\ifodd #1\fi #2\xint:#3\W #4\W
2049 {%
2050     \expandafter\XINT_mul_out
2051     \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
2052 }%
2053 \def\XINT_pow_II_even #1\xint:#2\W
2054 {%
2055     \expandafter\XINT_pow_II_loop
2056     \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2057     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2058 }%
2059 \def\XINT_pow_II_odda #1\xint:#2\W #3\W
2060 {%
2061     \expandafter\XINT_pow_II_oddb
2062     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2063     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
2064 }%
2065 \def\XINT_pow_II_oddb #1\xint:#2\W #3\W
2066 {%
2067     \expandafter\XINT_pow_II_loop
2068     \the\numexpr #1\expandafter\xint:%
2069     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
2070 }%
```

3.47 *\xintiFac*

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

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

Partially rewritten with release 1.2 to benefit from the inner format of the 1.2 multiplication.

3 Package *xintcore* implementation

With current default settings of the etex memory and a.t.t.o.w (11/2015) the maximal possible computation is 5971! (which has 19956 digits).

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

```

\catcode`_ 11
\catcode`^ 11
\long\def\xint_firstofthree #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree #1#2#3{#3}%
% quickly written factorial using binary split recursive method
\def\tFac {\romannumeral-`0\tfac }%
\def\tfac #1{\expandafter\XINT_mul_out
  \romannumeral-`0\ufac {1}{#1}1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
\def\ufac #1#2{\ifcase\numexpr#2-#1\relax
  \expandafter\xint_firstofthree
\or
  \expandafter\xint_secondofthree
\else
  \expandafter\xint_thirdofthree
\fi
  {\the\numexpr\xint_c_x^viii+#1!1;!}%
  {\the\numexpr\xint_c_x^viii+#1*#2!1;!}%
  {\expandafter\vfac\the\numexpr (#1+#2)/\xint_c_ii.#1.#2.}%
}%
\def\vfac #1.#2.#3.%
{%
  \expandafter
  \wfac\expandafter
  {\romannumeral-`0\expandafter
  \ufac\expandafter{\the\numexpr #1+\xint_c_i}{#3}}%
  {\ufac {#2}{#1}}%
}%
\def\wfac #1#2{\expandafter\zfac\romannumeral-`0#2\W #1}%
\def\zfac {\the\numexpr\XINT_mul_loop 10000000!1;! \W }% core multiplication...
\catcode`_ 8
\catcode`^ 7

```

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

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

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

```

2071 \def\xintiifac {\romannumeral0\xintiifac }%
2072 \def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
2073 \def\XINT_fac_fork #1#2.%
2074 {%
2075   \xint_UDzerominusfork
2076   #1-\XINT_fac_zero
2077   0#1\XINT_fac_neg
2078   0-\XINT_fac_checksiz
2079   \krof #1#2.%
2080 }%

```

3 Package *xintcore* implementation

```

2081 \def\XINT_fac_zero #1.{ 1}%
2082 \def\XINT_fac_neg #1.{\XINT_signalcondition{InvalidOperation}{Factorial of
2083   negative: (#1)!}}{0}}%
2084 \def\XINT_fac_checksiz #1.%
2085 {%
2086   \ifnum #1>\xint_c_x^iv \xint_dothis{\XINT_fac_toobig #1.}\fi
2087   \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
2088   \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
2089   \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
2090   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
2091 }%
2092 \def\XINT_fac_toobig #1.#2\W{\XINT_signalcondition{InvalidOperation}{Factorial
2093   of too big argument: #1 > 10000}}{0}}%
2094 \def\XINT_fac_bigloop_a #1.%
2095 {%
2096   \expandafter\XINT_fac_bigloop_b \the\numexpr
2097   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2098 }%
2099 \def\XINT_fac_bigloop_b #1.#2.%
2100 {%
2101   \expandafter\XINT_fac_medloop_a
2102   \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
2103 }%
2104 \def\XINT_fac_bigloop_loop #1.#2.%
2105 {%
2106   \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
2107   \expandafter\XINT_fac_bigloop_loop
2108   \the\numexpr #1+\xint_c_ii\expandafter.%
2109   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
2110 }%
2111 \def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
2112 \def\XINT_fac_bigloop_mul #1!%
2113 {%
2114   \expandafter\XINT_smallmul
2115   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2116 }%
2117 \def\XINT_fac_medloop_a #1.%
2118 {%
2119   \expandafter\XINT_fac_medloop_b
2120   \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2121 }%
2122 \def\XINT_fac_medloop_b #1.#2.%
2123 {%
2124   \expandafter\XINT_fac_smallloop_a
2125   \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
2126 }%
2127 \def\XINT_fac_medloop_loop #1.#2.%
2128 {%
2129   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2130   \expandafter\XINT_fac_medloop_loop
2131   \the\numexpr #1+\xint_c_iii\expandafter.%
2132   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%

```

3 Package *xintcore* implementation

```
2133 }%
2134 \def\XINT_fac_medloop_mul #1!%
2135 {%
2136   \expandafter\XINT_smallmul
2137   \the\numexpr
2138     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2139 }%
2140 \def\XINT_fac_smallloop_a #1.%
2141 {%
2142   \csname
2143     XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2144   \endcsname #1.%
2145 }%
2146 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
2147 {%
2148   \XINT_fac_smallloop_loop 2.#1.100000001!1;!%
2149 }%
2150 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2151 {%
2152   \XINT_fac_smallloop_loop 3.#1.100000002!1;!%
2153 }%
2154 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2155 {%
2156   \XINT_fac_smallloop_loop 4.#1.100000006!1;!%
2157 }%
2158 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2159 {%
2160   \XINT_fac_smallloop_loop 5.#1.1000000024!1;!%
2161 }%
2162 \def\XINT_fac_smallloop_loop #1.#2.%
2163 {%
2164   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2165   \expandafter\XINT_fac_smallloop_loop
2166   \the\numexpr #1+\xint_c_iv\expandafter.%
2167   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2168 }%
2169 \def\XINT_fac_smallloop_mul #1!%
2170 {%
2171   \expandafter\XINT_smallmul
2172   \the\numexpr
2173     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2174 }%
2175 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%
```

3.48 *\XINT_useiimessage*

1.2o

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

3 Package *xintcore* implementation

```
2181     {}%  
2182 }%  
2183 \XINT_restorecatcodes_endinput%
```


4 Package `xint` implementation

.1	Package identification	114	.31	<code>\xintiiifGt</code>	126
.2	More token management	114	.32	<code>\xintiiifLt</code>	126
.3	(WIP) A constant needed by <code>\xintRandomDigits</code> et al.	114	.33	<code>\xintiiifZero</code>	126
.4	<code>\xintLen</code>	115	.34	<code>\xintiiifNotZero</code>	127
.5	<code>\xintReverseDigits</code>	115	.35	<code>\xintiiifOne</code>	127
.6	<code>\xintiiE</code>	116	.36	<code>\xintiiifOdd</code>	127
.7	<code>\xintDecSplit</code>	116	.37	<code>\xintifTrueAelseB, \xintifFalseAelseB</code>	127
.8	<code>\xintDecSplitL</code>	118	.38	<code>\xintIsTrue, \xintIsFalse</code>	128
.9	<code>\xintDecSplitR</code>	118	.39	<code>\xintNOT</code>	128
.10	<code>\xintDSHr</code>	119	.40	<code>\xintAND, \xintOR, \xintXOR</code>	128
.11	<code>\xintDSH</code>	119	.41	<code>\xintANDof</code>	128
.12	<code>\xintDSx</code>	120	.42	<code>\xintORof</code>	129
.13	<code>\xintiiEq</code>	122	.43	<code>\xintXORof</code>	129
.14	<code>\xintiiNotEq</code>	122	.44	<code>\xintiiMax</code>	129
.15	<code>\xintiiGeq</code>	122	.45	<code>\xintiiMin</code>	130
.16	<code>\xintiiGt</code>	123	.46	<code>\xintiiMaxof</code>	131
.17	<code>\xintiiLt</code>	123	.47	<code>\xintiiMinof</code>	132
.18	<code>\xintiiGtorEq</code>	123	.48	<code>\xintiiSum</code>	132
.19	<code>\xintiiLtorEq</code>	123	.49	<code>\xintiiPrd</code>	132
.20	<code>\xintiiIsZero</code>	123	.50	<code>\xintiiSquareRoot</code>	133
.21	<code>\xintiiIsNotZero</code>	123	.51	<code>\xintiiSqrt, \xintiiSqrtR</code>	140
.22	<code>\xintiiIsOne</code>	123	.52	<code>\xintiiBinomial</code>	141
.23	<code>\xintiiOdd</code>	124	.53	<code>\xintiiPFactorial</code>	146
.24	<code>\xintiiEven</code>	124	.54	<code>\xintBool, \xintToggle</code>	149
.25	<code>\xintiiMON</code>	124	.55	(WIP) <code>\xintRandomDigits</code>	150
.26	<code>\xintiiMMON</code>	124	.56	(WIP) <code>\XINT_eightrandomdigits</code>	150
.27	<code>\xintSgnFork</code>	125	.57	(WIP) <code>\xintXRandomDigits</code>	151
.28	<code>\xintiiifSgn</code>	125	.58	(WIP) <code>\xintiiRandRangeAtoB</code>	151
.29	<code>\xintiiifCmp</code>	125	.59	(WIP) <code>\xintiiRandRange</code>	151
.30	<code>\xintiiifEq</code>	126	.60	Adjustments for engines without uniformdeviate primitive	153

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

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

1.3b adds randomness related macros.

```

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

```

4 Package `xint` implementation

```
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)
```

4.1 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xint}%
46 [2018/05/18 1.3b Expandable operations on big integers (JFB)]%
```

4.2 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_firstofthree_thenstop #1#2#3{ #1}% 1.09i
51 \long\def\xint_secondofthree_thenstop #1#2#3{ #2}%
52 \long\def\xint_thirdofthree_thenstop #1#2#3{ #3}%
```

4.3 (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
```

4.4 `\xintLen`

`\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`.

```

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 }%

```

4.5 `\xintReverseDigits`

1.2.

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

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

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

```

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

```

4 Package `xint` implementation

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

4.6 `\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.

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

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

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

4.7 `\xintDecSplit`

DECIMAL SPLIT

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

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

4 Package *xint* implementation

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

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

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

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

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

```

4 Package `xint` implementation

```
169 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
170   {#1#2#3#4\XINT_split_fromleft_end_b}%
171 \expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
172   {#1#2#3#4#5\XINT_split_fromleft_end_b}%
173 \expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
174   {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
175 \expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
176   {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
177 \expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
178   {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%

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

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

4.8 `\xintDecSplitL`

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

4.9 `\xintDecSplitR`

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

```

212 \xint_bye2345678\xint_bye. .%
213 }%
214 \def\xint_splitr_finish #1.#2.{ #2}%

```

4.10 \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}

si $x \leq 0$, fait $A \rightarrow A \cdot 10^{|x|}$. si $x > 0$, et $A \geq 0$, fait $A \rightarrow \text{quo}(A, 10^x)$

si $x > 0$, et $A < 0$, fait $A \rightarrow -\text{quo}(-A, 10^x)$

(donc pour $x > 0$ c'est comme DSR itéré x fois)

\xintDSHr donne le 'reste' (si $x \leq 0$ donne zéro).

Badly named macros.

Rewritten for 1.2i, this was old code and \xintDSx has changed interface.

```

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

```

```

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

```

```

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

```

```

231 \expandafter\xint_secondoftwo_thenstop\romannumeral0\xint_dsx_xisPos
232 }%

```

4.11 \xintDSH

```

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

```

```

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

```

4 Package *xint* implementation

```
242     0#1\XINT_dsx_xisNeg_checkA
243     0-{\XINT_dsh_xisPos #1}%
244     \krof
245 }%
246 \def\XINT_dsh_xiszero #1.#2;{ #2}%
247 \def\XINT_dsh_xisPos
248 {%
  \expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
249 }%
```

4.12 \xintDSx

--> Attention le cas $x=0$ est traité dans la même catégorie que $x > 0$ <--
si $x < 0$, fait $A \rightarrow A \cdot 10^{(|x|)}$
si $x \geq 0$, et $A \geq 0$, fait $A \rightarrow \{ \text{quo}(A, 10^x) \} \{ \text{rem}(A, 10^x) \}$
si $x \geq 0$, et $A < 0$, d'abord on calcule $\{ \text{quo}(-A, 10^x) \} \{ \text{rem}(-A, 10^x) \}$
puis, si le premier n'est pas nul on lui donne le signe -
si le premier est nul on donne le signe - au second.

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

Rewritten for 1.2i, this was old code.

```
250 \def\xintDSx {\romannumeral0\xintdsx }%
251 \def\xintdsx #1#2%
252 {%
  \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
253 }%
254 }%
255 \def\XINT_dsx_fork #1%
256 {%
257     \xint_UDzerominusfork
258     #1-\XINT_dsx_xisZero
259     0#1\XINT_dsx_xisNeg_checkA
260     0-{\XINT_dsx_xisPos #1}%
261     \krof
262 }%
263 \def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%
264 \def\XINT_dsx_xisNeg_checkA #1.#2%
265 {%
266     \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%
  \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%
267 }%
268 }%
269 \def\XINT_dsx_xisNeg_Azero #1;{ 0}%
  \def\XINT_dsx_addzeros #1%
270 }%
271 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%
```


4 Package *xint* implementation

```
272 \def\XINT_dsx_addzerosnofuss #1%
273   {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%
274 \def\XINT_dsx_append #1.#2;{ #2#1}%

275 \def\XINT_dsx_xisPos #1.#2%
276 {%
277   \xint_UDzerominusfork
278     #2-\XINT_dsx_AisZero
279     0#2\XINT_dsx_AisNeg
280     0-\XINT_dsx_AisPos
281   \krof #1.#2%
282 }%
283 \def\XINT_dsx_AisZero #1;{{0}{0}}%
284 \def\XINT_dsx_AisNeg #1.-#2;%
285 {%
286   \expandafter\XINT_dsx_AisNeg_checkiffirstempty
287   \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
288 }%

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

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

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

4 Package *xint* implementation

```
313           {\romannumeral0\XINT_num {#1}}%
314 }%
315 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%
```

4.13 `\xintiiEq`

no `\xintiieq`.

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

4.14 `\xintiiNotEq`

Pour `xintexpr`. Pas de version en lowercase.

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

4.15 `\xintiiGeq`

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

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

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

This macro should have been called `\xintGEq` for example.

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

324 \def\XINT_geq #1#2\xint:#3%
325 {%
326   \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
327 }%
328 \def\XINT_geq_fork #1#2%
329 {%
330   \xint_UDzerofork
331     #1\XINT_geq_firstiszero
332     #2\XINT_geq_secondiszero
333     0}%
334   \krof
335   \xint_UDsignsfork
336     #1#2\XINT_geq_minusminus
337     #1-\XINT_geq_minusplus
338     #2-\XINT_geq_plusminus
339     --\XINT_geq_plusplus
340   \krof #1#2%
341 }%
342 \def\XINT_geq_firstiszero #1\krof 0#2#3\xint:#4\xint:
343           {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
344 \def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%

```

4 Package `xint` implementation

```
345 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
346 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus {}#1}%
347 \def\XINT_geq_minusminus --{\XINT_geq_plusplus {}{}}%
348 \def\XINT_geq_plusplus
349   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
350 \def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
351   \else\expandafter\XINT_geq_yes\fi}%
352 \def\XINT_geq_no 1{ 0}%
353 \def\XINT_geq_yes { 1}%
```

4.16 `\xintiiGt`

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

4.17 `\xintiiLt`

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

4.18 `\xintiiGtorEq`

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

4.19 `\xintiiLtorEq`

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

4.20 `\xintiiIsZero`

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

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

4.21 `\xintiiIsNotZero`

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

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

4.22 `\xintiiIsOne`

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

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

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

```

373 {%
374   \unless\if#2X\xint_dothis0\fi
375   \unless\if#11\xint_dothis0\fi
376   \xint_orthat1%
377 }%

```

4.23 `\xintiiOdd`

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

```

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

```

4.24 `\xintiiEven`

```

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

```

4.25 `\xintiiMON`

MINUS ONE TO THE POWER N

```

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

```

4.26 `\xintiiMMON`

MINUS ONE TO THE POWER N-1

```

405 \def\xintiiMMON {\romannumeral0\xintiimmon }%
406 \def\xintiimmon #1%
407 {%
408   \ifodd\xintLDg {#1} %<- intentional space

```

```

409     \xint_afterfi{ 1}%
410   \else
411     \xint_afterfi{ -1}%
412   \fi
413 }%

```

4.27 `\xintSgnFork`

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

```

414 \def\xintSgnFork {\romannumeral0\xintsgnfork }%
415 \def\xintsgnfork #1%
416 {%
417   \ifcase #1 \expandafter\xint_secondofthree_thenstop
418             \or\expandafter\xint_thirdofthree_thenstop
419             \else\expandafter\xint_firstofthree_thenstop
420   \fi
421 }%

```

4.28 `\xintiiifSgn`

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

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

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

```

422 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
423 \def\xintiiifsgn #1%
424 {%
425   \ifcase \xintiiSgn{#1}
426             \expandafter\xint_secondofthree_thenstop
427             \or\expandafter\xint_thirdofthree_thenstop
428             \else\expandafter\xint_firstofthree_thenstop
429   \fi
430 }%

```

4.29 `\xintiiifCmp`

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

```

431 \def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
432 \def\xintiiifcmp #1#2%
433 {%
434   \ifcase\xintiiCmp {#1}{#2}
435             \expandafter\xint_secondofthree_thenstop
436             \or\expandafter\xint_thirdofthree_thenstop
437             \else\expandafter\xint_firstofthree_thenstop
438   \fi
439 }%

```

4.30 `\xintiiifEq`

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

```

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

```

4.31 `\xintiiifGt`

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

```

448 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
449 \def\xintiiifgt #1#2%
450 {%
451   \if1\xintiiCmp{#1}{#2}%
452     \expandafter\xint_firstoftwo_thenstop
453   \else\expandafter\xint_secondoftwo_thenstop
454   \fi
455 }%

```

4.32 `\xintiiifLt`

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

```

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

```

4.33 `\xintiiifZero`

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

1.2o deprecates `\xintifZero`.

```

464 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
465 \def\xintiiifzero #1%
466 {%
467   \if0\xintiiSgn{#1}%
468     \expandafter\xint_firstoftwo_thenstop
469   \else
470     \expandafter\xint_secondoftwo_thenstop

```

```
471 \fi
472 }%
```

4.34 `\xintiiifNotZero`

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

4.35 `\xintiiifOne`

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

```
482 \def\xintiiifOne {\romannumeral0\xintiiifone }%
483 \def\xintiiifone #1%
484 {%
485 \if1\xintiiIsOne{#1}%
486 \expandafter\xint_firstoftwo_thenstop
487 \else
488 \expandafter\xint_secondoftwo_thenstop
489 \fi
490 }%
```

4.36 `\xintiiifOdd`

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

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

4.37 `\xintifTrueAelseB`, `\xintifFalseAelseB`

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

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

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

4.38 `\xintIsTrue`, `\xintIsFalse`

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

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

4.39 `\xintNOT`

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

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

4.40 `\xintAND`, `\xintOR`, `\xintXOR`

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

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

4.41 `\xintANDof`

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

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

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

This macro as well as `ORof` and `XORof` are actually not used by `xintexpr`, which has its own csv handling macros.

```
516 \def\xintANDof {\romannumeral0\xintandof }%
517 \def\xintandof #1{\expandafter\xINT_andof_a\romannumeral`&&@#1\xint:}%
518 \def\xINT_andof_a #1{\expandafter\xINT_andof_b\romannumeral`&&@#1!}%
519 \def\xINT_andof_b #1%
520 {\xint_gob_til_xint: #1\xINT_andof_e\xint:\xINT_andof_c #1}%
521 \def\xINT_andof_c #1!%
522 {\xintifTrueAelseB {#1}{\xINT_andof_a}{\xINT_andof_no}}%
523 \def\xINT_andof_no #1\xint:{ 0}%
524 \def\xINT_andof_e #1!{ 1}%
```


4.42 `\xintORof`

New with 1.09a. Works also with an empty list. Empty items however are not accepted.
1.2l made `\xintORof` robust against non terminated items.

```

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

```

4.43 `\xintXORof`

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

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

```

534 \def\xintXORof     {\romannumeral0\xintxorof }%
535 \def\xintxorof     #1{\expandafter\XINT_xorof_a\expandafter
536     0\romannumeral`&&@#1\xint:}%
537 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2!#1}%
538 \def\XINT_xorof_b #1%
539     {\xint_gob_til_xint: #1\XINT_xorof_e\xint:\XINT_xorof_c #1}%
540 \def\XINT_xorof_c #1!#2%
541     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
542     \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
543     {\XINT_xorof_a #2}}%
544     }%
545 \def\XINT_xorof_e #1!#2{ #2}%

```

4.44 `\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.

```

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

```

`#3#4` vient du **premier**, `#1#2` vient du **second**. I have renamed the sub-macros at 1.2m because the terminology was quite counter-intuitive; there was no bug, but still.

4 Package `xint` implementation

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

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

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

Premier des testés $|A|=-A$, second est $|B|=-B$. On veut le $\max(A,B)$, c'est donc A si $|A|<|B|$ (ou $|A|=|B|$, mais peu importe alors). Donc on peut faire cela avec `\unless`. Simple.

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

4.45 `\xintiiMin`

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

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

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

```
588 \def\xintiiMin {\romannumeral0\xintiimin }%
589 \def\xintiimin #1%
590 {%
```

4 Package `xint` implementation

```

591 \expandafter\xint_iimin \romannumeral`&&@#1\xint:
592 }%
593 \def\xint_iimin #1\xint:#2%
594 {%
595 \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
596 }%
597 \def\XINT_min_fork #1#2\xint:#3#4\xint:
598 {%
599 \xint_UDsignsfork
600 #1#3\XINT_min_minusminus % A < 0, B < 0
601 #1-\XINT_min_plusminus % B < 0, A >= 0
602 #3-\XINT_min_minusplus % A < 0, B >= 0
603 --{\xint_UDzerosfork
604 #1#3\XINT_min_zerozero % A = B = 0
605 #10\XINT_min_pluszero % B = 0, A > 0
606 #30\XINT_min_zeroplus % A = 0, B > 0
607 00\XINT_min_plusplus % A, B > 0
608 \krof }%
609 \krof
610 #3#1#2\xint:#4\xint:
611 \expandafter\xint_secondoftwo_thenstop
612 \else
613 \expandafter\xint_firstoftwo_thenstop
614 \fi
615 {#3#4}{#1#2}%
616 }%
617 \def\XINT_min_zerozero #1\fi{\xint_firstoftwo_thenstop }%
618 \def\XINT_min_zeroplus #1\fi{\xint_firstoftwo_thenstop }%
619 \def\XINT_min_pluszero #1\fi{\xint_secondoftwo_thenstop }%
620 \def\XINT_min_minusplus #1\fi{\xint_firstoftwo_thenstop }%
621 \def\XINT_min_plusminus #1\fi{\xint_secondoftwo_thenstop }%
622 \def\XINT_min_plusplus
623 {%
624 \if1\romannumeral0\XINT_geq_plusplus
625 }%
626 \def\XINT_min_minusminus --%
627 {%
628 \unless\if1\romannumeral0\XINT_geq_plusplus{}}%
629 }%

```

4.46 `\xintiiMaxof`

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

NOT compatible with empty list.

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

```

630 \def\xintiiMaxof {\romannumeral0\xintiimaxof }%
631 \def\xintiimaxof #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\xint:}%
632 \def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1!}%
633 \def\XINT_iimaxof_b #1!#2%
634 {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2!{#1}!}%
635 \def\XINT_iimaxof_c #1%

```

4 Package *xint* implementation

```
636      {\xint_gob_til_xint: #1\XINT_iimaxof_e\xint:\XINT_iimaxof_d #1}%
637 \def\XINT_iimaxof_d #1!%
638      {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
639 \def\XINT_iimaxof_e #1!#2!{ #2}%
```

4.47 `\xintiiMinof`

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

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

4.48 `\xintiiSum`

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

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

4.49 `\xintiiPrd`

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

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

4.50 `\xintiiSquareRoot`

First done with 1.08.

1.1 added `\xintiiSquareRoot`.

1.1a added `\xintiiSqrtR`.

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

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

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

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

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

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

```

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

```

4 Package `xint` implementation

```
698 \else
699 \expandafter\XINT_sqrt_bE
700 \fi
701 #1.%
702 }%

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

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

711 \def\XINT_sqrt_c #1#2%
712 {%
713 \expandafter #2%
714 \the\numexpr \ifnum #1>\xint_c_ii
715 \ifnum #1>\xint_c_vi
716 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
717 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
718 \ifnum #1>90
719 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
720 \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
721 }%

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

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

732 \def\XINT_sqrt_small_ea #1%
733 {%
734 \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
735 \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
```

4 Package *xint* implementation

```

736 \xint_orthat\XINT_sqrt_small_f #1%
737 }%
738 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
739 \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

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

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

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

755 \def\XINT_sqrt_small_g #1#2.%
756 {%
757 \if 0#1%
758 \expandafter\XINT_sqrt_small_end
759 \else
760 \expandafter\XINT_sqrt_small_h
761 \fi
762 #1#2.%
763 }%

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

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

```

4 Package `xint` implementation

```

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

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

781 \def\xINT_sqrt_big_eE_a #1.#2;#3%
782 {%
783 \expandafter\xINT_sqrt_bigormed_f
784 \romannumeral0\xINT_sqrt_small_e #2000.#3.#1;%
785 }%

786 \def\xINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
787 {%
788 \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
789 }%
790 \def\xINT_sqrt_big_e0_a #1.#2;#3#4%
791 {%
792 \expandafter\xINT_sqrt_bigormed_f
793 \romannumeral0\xINT_sqrt_small_e #20000.#3#4.#1;%
794 }%

795 \def\xINT_sqrt_bigormed_f #1#2#3;%
796 {%
797 \ifnum#3<\xint_c_ix
798 \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
799 \fi
800 \xint_orthat\xINT_sqrt_big_f #1.#2.#3;%
801 }%
802 \def\xINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
803 \def\xINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
804 \def\xINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
805 \def\xINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%

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

```


4 Package `xint` implementation

```

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

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

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

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

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

840 \def\XINT_sqrt_big_gc #1.#2.#3.%
841 {%
842   \expandafter\XINT_sqrt_big_gd
843   \romannumeral0\xintiiaadd
844     {\xintiiSub {#3000000000}{\xintDouble{\xintiiMul{#2}{#1}}00000000}%
845     {\xintiiSqr {#1}}}%
846   \romannumeral0\xintiisub{#2000000000}{#1}.%
847 }%

```

4 Package `xint` implementation

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

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

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

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

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

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

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

4 Package `xint` implementation

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

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

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

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

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

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

914 \def\XINT_sqrt_big_kc #1%
915 {%
916   \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
917   \xint_orthat\XINT_sqrt_big_kloop #1%
918 }%
919 \def\XINT_sqrt_big_kz 0.#1.%
920 {%
```

4 Package `xint` implementation

```

921 \expandafter\XINT_sqrt_big_kend
922 \romannumeral0%
923 \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
924 }%
925 \def\XINT_sqrt_big_kend #1.#2.%
926 {%
927 \expandafter{\romannumeral0\xintinc{#2}}{#1}%
928 }%

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

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

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

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

```

4.51 `\xintiiSqr`, `\xintiiSqrR`

```

953 \def\xintiiSqr {\romannumeral0\xintiisqr }%
954 \def\xintiisqr {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
955 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
956 \def\xintiiSqrR {\romannumeral0\xintiisqrtr }%
957 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%

```

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

4 Package *xint* implementation

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

4.52 `\xintiiBinomial`

2015/11/28-29 for 1.2f.

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

I really lack some kind of infinity or NaN value.

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

```
960 \def\xintiiBinomial {\romannumeral0\xintiibinomial }%
961 \def\xintiibinomial #1#2%
962 {%
963   \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
964 }%
965 \def\XINT_binom_pre #1.#2.%
966 {%
967   \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
968 }%
```

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

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

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

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

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

$y = x - k + 1$, $j = 1$, k . On va évaluer par $y/1 * (y+1)/2 * (y+2)/3$ etc... On essaie de regrouper de manière à utiliser au mieux `\numexpr`. On peut aller jusqu'à $x = 10000$ car $9999 * 10000 < 10^8$. $463 * 464 * 465 = 99896880$, $98 * 99 * 100 * 101 = 97990200$. On va vérifier à chaque étape si on dépasse un seuil. Le style de

4 Package xint implementation

l'implémentation diffère de celui que j'avais utilisé pour `\xintiiFac`. On pourrait tout-à-fait avoir une `verybigloop`, mais bon. Je rajoute aussi un `verysmall`. Le traitement est un peu différent pour elle afin d'aller jusqu'à $x=29$ (et pas seulement 26 si je suivais le modèle des autres, mais je veux pouvoir faire `binomial(29,1)`, `binomial(29,2)`, ... en `vsmall`).

```

987 \def\xINT_binom_b #1.%
988 {%
989   \ifnum #1>9999 \xint_dothis\xINT_binom_vbigloop \fi
990   \ifnum #1>463 \xint_dothis\xINT_binom_bigloop \fi
991   \ifnum #1>98 \xint_dothis\xINT_binom_medloop \fi
992   \ifnum #1>29 \xint_dothis\xINT_binom_smallloop \fi
993   \xint_orthat\xINT_binom_vsmallloop #1.%
994 }%

```

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

```

995 \def\xINT_binom_smallloop #1.#2.#3.%
996 {%
997   \ifcase\numexpr #3-#2\relax
998     \expandafter\xINT_binom_end_
999   \or \expandafter\xINT_binom_end_i
1000  \or \expandafter\xINT_binom_end_ii
1001  \or \expandafter\xINT_binom_end_iii
1002  \else\expandafter\xINT_binom_smallloop_a
1003  \fi #1.#2.#3.%
1004 }%

```

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

```

1005 \def\xINT_binom_smallloop_a #1.#2.#3.%
1006 {%
1007   \expandafter\xINT_binom_smallloop_b
1008   \the\numexpr #1+\xint_c_iv\expandafter.%
1009   \the\numexpr #2+\xint_c_iv\expandafter.%
1010   \the\numexpr #3\expandafter.%
1011   \the\numexpr\expandafter\xINT_binom_div
1012   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1013   !\romannumeral0\expandafter\xINT_binom_mul
1014   \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1015 }%
1016 \def\xINT_binom_smallloop_b #1.%
1017 {%
1018   \ifnum #1>98 \expandafter\xINT_binom_medloop \else
1019     \expandafter\xINT_binom_smallloop \fi #1.%
1020 }%

```

Ici on prend trois par trois.

```

1021 \def\xINT_binom_medloop #1.#2.#3.%
1022 {%
1023   \ifcase\numexpr #3-#2\relax
1024     \expandafter\xINT_binom_end_

```

4 Package *xint* implementation

```
1025 \or \expandafter\XINT_binom_end_i
1026 \or \expandafter\XINT_binom_end_ii
1027 \else\expandafter\XINT_binom_medloop_a
1028 \fi #1.#2.#3.%
1029 }%
1030 \def\XINT_binom_medloop_a #1.#2.#3.%
1031 {%
1032 \expandafter\XINT_binom_medloop_b
1033 \the\numexpr #1+\xint_c_iii\expandafter.%
1034 \the\numexpr #2+\xint_c_iii\expandafter.%
1035 \the\numexpr #3\expandafter.%
1036 \the\numexpr\expandafter\XINT_binom_div
1037 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1038 !\romannumeral0\expandafter\XINT_binom_mul
1039 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1040 }%
1041 \def\XINT_binom_medloop_b #1.%
1042 {%
1043 \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1044 \expandafter\XINT_binom_medloop \fi #1.%
1045 }%
```

Ici on prend deux par deux.

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

Et finalement un par un.

```
1070 \def\XINT_binom_vbigloop #1.#2.#3.%
1071 {%
1072 \ifnum #3=#2
```

4 Package *xint* implementation

```

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

```

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

```

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

```


4 Package *xint* implementation

```

1119 1;!%
1120 }%
1121 \def\XINT_binom_div #1!1;!%
1122 {%
1123 \expandafter\XINT_smalldivx_a
1124 \the\numexpr #1/\xint_c_ii\expandafter\xint:
1125 \the\numexpr \xint_c_x^viii+#1!%
1126 }%

```

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

```

1127 \def\XINT_binom_vsmallmuldiv #1!#2!1#3!\{\xint_c_x^viii+#2*#3/#1!}%

```

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

```

1128 \def\XINT_binom_end_iii #1.#2.#3.%
1129 {%
1130 \expandafter\XINT_binom_finish
1131 \the\numexpr\expandafter\XINT_binom_div
1132 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1133 !\romannumeral0\expandafter\XINT_binom_mul
1134 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1135 }%
1136 \def\XINT_binom_end_ii #1.#2.#3.%
1137 {%
1138 \expandafter\XINT_binom_finish
1139 \the\numexpr\expandafter\XINT_binom_div
1140 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1141 !\romannumeral0\expandafter\XINT_binom_mul
1142 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1143 }%
1144 \def\XINT_binom_end_i #1.#2.#3.%
1145 {%
1146 \expandafter\XINT_binom_finish
1147 \the\numexpr\expandafter\XINT_binom_div
1148 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1149 !\romannumeral0\expandafter\XINT_binom_mul
1150 \the\numexpr #1*(#1+\xint_c_i)!%
1151 }%
1152 \def\XINT_binom_end_ #1.#2.#3.%
1153 {%
1154 \expandafter\XINT_binom_finish
1155 \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1156 !\romannumeral0\XINT_binom_mul #1!%
1157 }%
1158 \def\XINT_binom_finish #1;!0!%
1159 {\XINT_unsep_cuzsmall #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%

```

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

```

1160 \def\XINT_binom_vsmallend_iii #1.%
1161 {%

```

4 Package `xint` implementation

```

1162 \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1163 \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1164 }%
1165 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1166 {%
1167 \expandafter\XINT_binom_vsmallfinish
1168 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1169 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1170 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1171 }%
1172 \def\XINT_binom_vsmallend_ii #1.%
1173 {%
1174 \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1175 \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1176 }%
1177 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1178 {%
1179 \expandafter\XINT_binom_vsmallfinish
1180 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1181 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1182 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1183 }%
1184 \def\XINT_binom_vsmallend_i #1.%
1185 {%
1186 \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1187 \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1188 }%
1189 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1190 {%
1191 \expandafter\XINT_binom_vsmallfinish
1192 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1193 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1194 !\the\numexpr #1*(#1+\xint_c_i)!%
1195 }%
1196 \def\XINT_binom_vsmallend_ #1.%
1197 {%
1198 \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1199 \expandafter\XINT_binom_vsmallend_b \fi #1.%
1200 }%
1201 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1202 {%
1203 \expandafter\XINT_binom_vsmallfinish
1204 \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
1205 }%
1206 \def\XINT_binom_vsmallfinish#1{%
1207 \def\XINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
1208 }\XINT_binom_vsmallfinish{ }%

```

4.53 `\xintiiPFactorial`

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

4 Package *xint* implementation

```

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

```

4 Package `xint` implementation

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

4.54 `\xintBool`, `\xintToggle`

1.09c

4 Package `xint` implementation

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

4.55 (WIP) `\xintRandomDigits`

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

```
1355 \def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
1356 \def\xintrandomdigits#1%
1357 {%
1358     \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
1359 }%
1360 \def\XINT_randomdigits#1\xint:
1361 {%
1362     \expandafter\XINT_randomdigits_a
1363     \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1364 }%
1365 \def\XINT_randomdigits_a#1\xint:#2\xint:
1366 {%
1367     \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
1368     \romannumeral\XINT_replicate #1\endcsname \csname
1369     XINT_rdg\endcsname
1370 }%
1371 \def\XINT_rdg
1372 {%
1373     \expandafter\XINT_rdg_aux\the\numexpr%
1374     \xint_c_nine_x^viii%
1375     -\xint_texuniformdeviate\xint_c_ii^vii%
1376     -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1377     -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1378     -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1379     +\xint_texuniformdeviate\xint_c_x^viii%
1380     \relax%
1381 }%
1382 \def\XINT_rdg_aux#1{XINT_rdg\endcsname}%
1383 \let\XINT_XINT_rdg\endcsname
```

4.56 (WIP) `\XINT_eightrandomdigits`

1.3b.

```
1384 \def\XINT_eightrandomdigits
1385 {%
1386     \expandafter\xint_gobble_i\the\numexpr%
1387     \xint_c_nine_x^viii%
1388     -\xint_texuniformdeviate\xint_c_ii^vii%
1389     -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1390     -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1391     -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1392     +\xint_texuniformdeviate\xint_c_x^viii%
1393     \relax%
```

1394 }%

4.57 (WIP) `\xintXRandomDigits`

1.3b.

```

1395 \def\xintXRandomDigits#1%
1396 {%
1397   \csname xint_gobble_\expandafter\XINT_xrandomdigits\the\numexpr#1\xint:
1398 }%
1399 \def\XINT_xrandomdigits#1\xint:
1400 {%
1401   \expandafter\XINT_xrandomdigits_a
1402   \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1403 }%
1404 \def\XINT_xrandomdigits_a#1\xint:#2\xint:
1405 {%
1406   \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
1407   \romannumeral`&&\romannumeral
1408   \XINT_replicate #1\endcsname\XINT_eightrandomdigits
1409 }%

```

4.58 (WIP) `\xintiRandRangeAtoB`

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

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

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

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

```

1410 \def\xintiRandRangeAtoB{\romannumeral`&&\xintiirandrangeAtoB}%
1411 \def\xintiirandrangeAtoB#1%
1412 {%
1413   \expandafter\XINT_randrangeAtoB_a\romannumeral`&&#1\xint:
1414 }%
1415 \def\XINT_randrangeAtoB_a#1\xint:#2%
1416 {%
1417   \xintiadd{\expandafter\XINT_randrange
1418             \romannumeral0\xintiisub{#2}{#1}\xint:}%
1419   {#1}%
1420 }%

```

4.59 (WIP) `\xintiRandRange`

1.3b. Support for `randrange()`.

```

1421 \def\xintiRandRange{\romannumeral`&&\xintiirandrange}%
1422 \def\xintiirandrange#1%
1423 {%
1424   \expandafter\XINT_randrange\romannumeral`&&#1\xint:
1425 }%
1426 \def\XINT_randrange #1%

```

4 Package *xint* implementation

```

1427 {%
1428   \xint_UDzerominusfork
1429   #1-\XINT_randrange_err:empty
1430   0#1\XINT_randrange_err:empty
1431   0-\XINT_randrange_a
1432   \krof #1%
1433 }%
1434 \def\XINT_randrange_err:empty#1\xint:
1435 {%
1436   \XINT_expandableerror{Empty range for randrange.} 0%
1437 }%
1438 \def\XINT_randrange_a #1\xint:
1439 {%
1440   \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
1441 }%
1442 \def\XINT_randrange_b #1.%
1443 {%
1444   \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
1445   \xint_orthat{\XINT_randrange_c #1.}%
1446 }%
1447 \def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
1448 {%
1449   \expandafter\XINT_randrange_d
1450   \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
1451     {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
1452   #2#3#4#5#6#7#8#9\xint:#1\xint:
1453 }%

```

This raises following annex question: immediately after setting the seed is it possible for `\xintUniformDeviante{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.

```

1454 \def\XINT_randrange_d #1\xint:#2\xint:
1455 {%
1456   \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
1457   \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
1458   \xint_orthat\XINT_randrange_e #1\xint:
1459 }%
1460 \def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
1461 {%
1462   \the\numexpr#1\expandafter\relax
1463   \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
1464 }%

```

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...

```

1465 \def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
1466 {%
1467   \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
1468 }%

```


4 Package *xint* implementation

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.

```
1469 \def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
1470 {%
1471   \expandafter\XINT_randrange_B
1472   \romannumeral0\xinrandomdigits{#2-\xint_c_viii}\xint:
1473   #3\xint:#2.#1\xint:
1474 }%
1475 \def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
1476 {%
1477   \xintiiiflt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
1478   #4#1\xint:#3.#4#2\xint:
1479 }%
1480 \def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
1481 \def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%
```

4.60 Adjustments for engines without uniformdeviate primitive

1.3b.

```
1482 \ifdefined\xint_texuniformdeviate
1483 \else
1484   \def\xinrandomdigits#1%
1485   {%
1486     \XINT_expandableerror
1487     {No uniformdeviate at engine level, returning 0.} 0%
1488   }%
1489   \let\xintXRandomDigits\xintRandomDigits
1490   \def\XINT_randrange#1\xint:
1491   {%
1492     \XINT_expandableerror
1493     {No uniformdeviate at engine level, returning 0.} 0%
1494   }%
1495 \fi
1496 \XINT_restorecatcodes_endinput%
```

5 Package *xintbinhex* implementation

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

The commenting is currently (2018/05/18) 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.

5.1 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

```

5 Package *xintbinhex* implementation

```
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
```

5.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xintbinhex}%
46 [2018/05/18 1.3b Expandable binary and hexadecimal conversions (JFB)]%
```

5.3 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}%
```

5 Package *xintbinhex* implementation

```
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
```

5.4 Helper macros

5.4.1 `\XINT_zeroes_foriv`

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

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

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

5.5 `\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
```

5 Package *xintbinhex* implementation

```

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!}%

```

5 Package *xintbinhex* implementation

```

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...

5 Package *xintbinhex* implementation

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 }%
```

5.6 `\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 { }%

```

5.7 `\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 }%

```


5 Package *xintbinhex* implementation

```

288 \def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_main}%
289 \def\XINT_htd_main #1\xint:
290 {%
291   \expandafter\XINT_htd_startb
292   \the\numexpr\expandafter\XINT_htd_starta
293   \romannumeral0\XINT_zeroes_foriv
294   #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
295   #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
296 }%
297 \def\XINT_htd_starta #1#2#3#4{"#1#2#3#4+100000!}%
298 \def\XINT_htd_startb 1#1%
299 {%
300   \if#10\expandafter\XINT_htd_startba\else
301     \expandafter\XINT_htd_startbb
302   \fi 1#1%
303 }%
304 \def\XINT_htd_startba 10#1!\{\XINT_htd_again #1%
305   \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%
306 \def\XINT_htd_startbb 1#1#2!\{\XINT_htd_again #1!#2%
307   \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%

```

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

```

308 \def\XINT_htd_again #1\XINT_htd_nextfour #2%
309 {%
310   \xint_bye #2\XINT_htd_finish\xint_bye
311   \expandafter\XINT_htd_A\the\numexpr
312   \XINT_htd_a #1\XINT_htd_nextfour #2%
313 }%
314 \def\XINT_htd_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
315 {%
316   #1\expandafter\XINT_htd_update
317   \the\numexpr #2\expandafter\XINT_htd_update
318   \the\numexpr #3\expandafter\XINT_htd_update
319   \the\numexpr #4\expandafter\XINT_htd_update
320   \the\numexpr #5\expandafter\XINT_htd_update
321   \the\numexpr #6\expandafter\XINT_htd_update
322   \the\numexpr #7\expandafter\XINT_htd_update
323   \the\numexpr #8\expandafter\XINT_htd_update
324   \the\numexpr #9\expandafter\XINT_htd_update
325   \the\numexpr \XINT_htd_a
326 }%
327 \def\XINT_htd_nextfour #1#2#3#4%
328 {%
329   *\xint_c_ii^xvi+"#1#2#3#4+10000000000\relax\xint_bye!%
330   2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour
331 }%

```

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

```

332 \def\XINT_htd_update 1#1#2#3#4#5#6!%

```

5 Package *xintbinhex* implementation

```
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{ }%
```

5.8 *\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 }%

```

5.9 \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 $\text{ceil}(\text{size}(\text{input})/4)$, 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 }%
```

5.10 `\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 { }%
```

5.11 `\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.

5 Package *xintbinhex* implementation

```
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%
```

6 Package *xintgcd* implementation

.1	Catcodes, ε -TeX and reload detection . . .	166	.7	<code>\xintBezoutAlgorithm</code>	174
.2	Package identification	167	.8	<code>\xintGCDof</code>	176
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	167	.9	<code>\xintLCMof</code>	176
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	168	.10	<code>\xintTypesetEuclideanAlgorithm</code>	176
.5	<code>\xintBezout</code>	168	.11	<code>\xintTypesetBezoutAlgorithm</code>	177
.6	<code>\xintEuclideanAlgorithm</code>	172			

The commenting is currently (2018/05/18) very sparse. Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the `xinttools` `\xintloop` rather than the Plain TeX or \mathbb{E} TeX's `\loop`.

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

Breaking change at 1.2p: `\xintBezout{A}{B}` formerly had output $\{A\}{B}\{U\}{V}\{D\}$ with $AU-BV=D$, now it is $\{U\}{V}\{D\}$ with $AU+BV=D$.

6.1 Catcodes, ε -TeX and reload detection

The code for reload detection was initially copied from \mathbb{E} TeX's packages, then modified.

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

```

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

```

```

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

```

6.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintgcd}%
46 [2018/05/18 1.3b Euclidean algorithm with xint package (JFB)]%

```

6.3 *\xintGCD*, *\xintiGCD*

1.09a added *\xintnum* filtering from *\xintiabs*. This is a bit overhead but makes it easier for the *gcd* function in *\xintexpr*.

1.1a defines *\xintiGCD* to avoid overhead in *\xintiexpr*.

1.2p: but 1.2o deprecated *\xintiAbs*, and in fact *\xintGCD* should not have any *\xintNum* overhead for consistency with other *xint* macros. But well, it would be breaking change to modify this now. We can not use *\xintiAbs* which will create a fraction $A/1$, so we use *\xintNum* directly.

```

47 \def\xintGCD {\romannumeral0\xintgcd }%
48 \def\xintgcd #1#2{\xintiigcd {\xintNum{#1}}{\xintNum{#2}}}%
49 \def\xintiGCD {\romannumeral0\xintiigcd }%
50 \def\xintiigcd #1%
51 {%
52   \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiabs{#1}}%
53 }%
54 \def\XINT_iigcd #1#2%
55 {%
56   \expandafter\XINT_gcd_fork\romannumeral0\xintiabs{#2}\Z #1\Z
57 }%

```

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

```

58 \def\XINT_gcd_fork #1#2\Z #3#4\Z
59 {%
60   \xint_UDzerofork
61     #1\XINT_gcd_BisZero
62     #3\XINT_gcd_AisZero
63     0\XINT_gcd_loop
64   \krof
65   {#1#2}{#3#4}%
66 }%
67 \def\XINT_gcd_AisZero #1#2{ #1}%
68 \def\XINT_gcd_BisZero #1#2{ #2}%

```

```

69 \def\XINT_gcd_CheckRem #1#2\Z
70 {%
71   \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop {#1#2}%
72 }%
73 \def\XINT_gcd_end0\XINT_gcd_loop #1#2{ #2}%

```

#1=B, #2=A. \XINT_div_prepare{#1}{#2} divides A by B.

```

74 \def\XINT_gcd_loop #1#2%
75 {%
76   \expandafter\expandafter\expandafter
77     \XINT_gcd_CheckRem
78   \expandafter\xint_secondoftwo
79   \romannumeral0\XINT_div_prepare {#1}{#2}\Z
80   {#1}%
81 }%

```

6.4 `\xintLCM`, `\xintiLCM`

See comments of `\xintGCD`.

```

82 \def\xintLCM {\romannumeral0\xintlcm}%
83 \def\xintlcm #1#2{\xintiilcm{\xintNum{#1}}{\xintNum{#2}}}%
84 \def\xintiLCM {\romannumeral0\xintiilcm}%
85 \def\xintiilcm #1%
86 {%
87   \expandafter\XINT_iilcm\expandafter{\romannumeral0\xintiabs{#1}}%
88 }%
89 \def\XINT_iilcm #1#2%
90 {%
91   \expandafter\XINT_lcm_fork\romannumeral0\xintiabs{#2}\Z #1\Z
92 }%
93 \def\XINT_lcm_fork #1#2\Z #3#4\Z
94 {%
95   \xint_UDzerofork
96   #1\XINT_lcm_BisZero
97   #3\XINT_lcm_AisZero
98   0\expandafter
99   \krof
100  \XINT_lcm_notzero\expandafter{\romannumeral0\XINT_gcd_loop {#1#2}{#3#4}}%
101  {#1#2}{#3#4}%
102 }%
103 \def\XINT_lcm_AisZero #1#2#3#4#5{ 0}%
104 \def\XINT_lcm_BisZero #1#2#3#4#5{ 0}%
105 \def\XINT_lcm_notzero #1#2#3{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

6.5 `\xintBezout`

`\xintBezout{#1}{#2}` produces $\{U\}\{V\}\{D\}$ with $UA+VB=D$, $D = \text{PGCD}(A,B)$ (non-positive), where *#1* and *#2* f-expand to big integers *A* and *B*.

I had not checked this macro for about three years when I realized in January 2017 that `\xintBezout{A}{B}` was buggy for the cases $A = 0$ or $B = 0$. I fixed that blemish in 1.21 but overlooked the

6 Package *xintgcd* implementation

other blemish that $\text{\xintBezout}\{A\}\{B\}$ with A multiple of B produced a coefficient U as -0 in place of 0.

Hence I rewrote again for 1.2p. On this occasion I modified the output of the macro to be $\{U\}\{V\}\{D\}$ with $AU+BV=D$, formerly it was $\{A\}\{B\}\{U\}\{V\}\{D\}$ with $AU - BV = D$. This is quite breaking change!

Note in particular change of sign of V.

I don't know why I had designed this macro to contain $\{A\}\{B\}$ in its output. Perhaps I initially intended to output $\{A//D\}\{B//D\}$ (but forgot), as this is actually possible from outcome of the last iteration, with no need of actually dividing. Current code however arranges to skip this last update, as U and V are already furnished by the iteration prior to realizing that the last non-zero remainder was found.

Also 1.2l raised `InvalidOperation` if both A and B vanished, but I removed this behaviour at 1.2p.

```
106 \def\xintBezout {\romannumeral0\xintbezout }%
107 \def\xintbezout #1%
108 {%
109   \expandafter\XINT_bezout\expandafter {\romannumeral0\xintnum{#1}}%
110 }%
111 \def\XINT_bezout #1#2%
112 {%
113   \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
114 }%
```

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

```
115 \def\XINT_bezout_fork #1#2\Z #3#4\Z
116 {%
117   \xint_UDzerosfork
118   #1#3\XINT_bezout_botharezero
119   #10\XINT_bezout_secondiszero
120   #30\XINT_bezout_firstiszero
121   00\xint_UDsignsfork
122   \krof
123     #1#3\XINT_bezout_minusminus % A < 0, B < 0
124     #1-\XINT_bezout_minusplus % A > 0, B < 0
125     #3-\XINT_bezout_plusminus % A < 0, B > 0
126     --\XINT_bezout_plusplus % A > 0, B > 0
127   \krof
128   {#2}{#4}#1#3% #1#2=B, #3#4=A
129 }%
130 \def\XINT_bezout_botharezero #1\krof#2#300{{0}{0}{0}}%
131 \def\XINT_bezout_firstiszero #1\krof#2#3#4#5%
132 {%
133   \xint_UDsignfork
134   #4{{0}{-1}}{#2}}%
135   -{{0}{1}}{#4#2}}%
136   \krof
137 }%
138 \def\XINT_bezout_secondiszero #1\krof#2#3#4#5%
139 {%
140   \xint_UDsignfork
141   #5{{-1}{0}}{#3}}%
142   -{{1}{0}}{#5#3}}%
```

6 Package *xintgcd* implementation

```

143 \krof
144 }%

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

145 \def\XINT_bezout_minusminus #1#2#3#4%
146 {%
147 \expandafter\XINT_bezout_mm_post
148 \romannumeral0\expandafter\XINT_bezout_preloop_a
149 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
150 }%
151 \def\XINT_bezout_mm_post #1#2%
152 {%
153 \expandafter\XINT_bezout_mm_postb\expandafter
154 {\romannumeral0\xintiiopp{#2}}{\romannumeral0\xintiiopp{#1}}%
155 }%
156 \def\XINT_bezout_mm_postb #1#2{\expandafter{#2}{#1}}%

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

157 \def\XINT_bezout_minusplus #1#2#3#4%
158 {%
159 \expandafter\XINT_bezout_mp_post
160 \romannumeral0\expandafter\XINT_bezout_preloop_a
161 \romannumeral0\XINT_div_prepare {#1}{#4#2}{#1}%
162 }%
163 \def\XINT_bezout_mp_post #1#2%
164 {%
165 \expandafter\xint_exchangetwo_keepbraces\expandafter
166 {\romannumeral0\xintiiopp {#2}}{#1}%
167 }%

plusminus A < 0, B > 0

168 \def\XINT_bezout_plusminus #1#2#3#4%
169 {%
170 \expandafter\XINT_bezout_pm_post
171 \romannumeral0\expandafter\XINT_bezout_preloop_a
172 \romannumeral0\XINT_div_prepare {#3#1}{#2}{#3#1}%
173 }%
174 \def\XINT_bezout_pm_post #1{\expandafter{\romannumeral0\xintiiopp{#1}}}%

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

175 \def\XINT_bezout_plusplus #1#2#3#4%
176 {%
177 \expandafter\XINT_bezout_preloop_a
178 \romannumeral0\XINT_div_prepare {#3#1}{#4#2}{#3#1}%
179 }%

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

```

6 Package *xintgcd* implementation

```

sinon mise à jour
  vv, v = q * vv + v, vv
  uu, u = q * uu + u, uu
  e = -e
  puis calcul quotient reste et itération

```

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

```

180 \def\xINT_bezout_preloop_a #1#2#3%
181 {%
182   \if0#1\xint_dothis\xINT_bezout_preloop_exchange\fi
183   \if0#2\xint_dothis\xINT_bezout_preloop_exit\fi
184   \xint_orthat{\expandafter\xINT_bezout_loop_B}%
185   \romannumeral0\xINT_div_prepare {#2}{#3}{#2}{#1}110%
186 }%
187 \def\xINT_bezout_preloop_exit
188   \romannumeral0\xINT_div_prepare #1#2#3#4#5#6#7%
189 {%
190   {0}{1}{#2}%
191 }%
192 \def\xINT_bezout_preloop_exchange
193 {%
194   \expandafter\xint_exchangetwo_keepbraces
195   \romannumeral0\expandafter\xINT_bezout_preloop_A
196 }%
197 \def\xINT_bezout_preloop_A #1#2#3#4%
198 {%
199   \if0#2\xint_dothis\xINT_bezout_preloop_exit\fi
200   \xint_orthat{\expandafter\xINT_bezout_loop_B}%
201   \romannumeral0\xINT_div_prepare {#2}{#3}{#2}{#1}%
202 }%
203 \def\xINT_bezout_loop_B #1#2%
204 {%
205   \if0#2\expandafter\xINT_bezout_exitA
206   \else\expandafter\xINT_bezout_loop_C
207   \fi {#1}{#2}%
208 }%

```

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

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

```

209 \def\xINT_bezout_loop_C #1#2#3#4#5#6#7%
210 {%
211   \expandafter\xINT_bezout_loop_D\expandafter
212     {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
213     {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%

```

```

214   {#2}{#3}{#4}{#5}%
215 }%
216 \def\XINT_bezout_loop_D #1#2%
217 {%
218   \expandafter\XINT_bezout_loop_E\expandafter{#2}{#1}%
219 }%
220 \def\XINT_bezout_loop_E #1#2#3#4%
221 {%
222   \expandafter\XINT_bezout_loop_b
223   \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
224 }%
225 \def\XINT_bezout_loop_b #1#2%
226 {%
227   \if0#2\expandafter\XINT_bezout_exita
228   \else\expandafter\XINT_bezout_loop_c
229   \fi {#1}{#2}%
230 }%
231 \def\XINT_bezout_loop_c #1#2#3#4#5#6#7%
232 {%
233   \expandafter\XINT_bezout_loop_d\expandafter
234     {\romannumeral0\xintiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
235     {\romannumeral0\xintiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%
236     {#2}{#3}{#4}{#5}%
237 }%
238 \def\XINT_bezout_loop_d #1#2%
239 {%
240   \expandafter\XINT_bezout_loop_e\expandafter{#2}{#1}%
241 }%
242 \def\XINT_bezout_loop_e #1#2#3#4%
243 {%
244   \expandafter\XINT_bezout_loop_B
245   \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
246 }%

```

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

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

```

247 \def\XINT_bezout_exita #1#2#3#4#5#6#7{{-#5}{#4}{#3}}%
248 \def\XINT_bezout_exita #1#2#3#4#5#6#7{{#5}{-#4}{#3}}%

```

6.6 \xintEuclideanAlgorithm

Pour Euclide: $\{N\}{A}\{D=r(n)\}{B}\{q_1\}{r_1}\{q_2\}{r_2}\{q_3\}{r_3}\dots\{q_N\}{r_N=0}$
 $u_{<2n>} = u_{<2n+3>}u_{<2n+2>} + u_{<2n+4>}$ à la n ième étape.

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

```

249 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm }%
250 \def\xinteucclideanalgorithm #1%
251 {%
252   \expandafter\XINT_euc\expandafter{\romannumeral0\xintiabs{\xintNum{#1}}}%
253 }%
254 \def\XINT_euc #1#2%

```

6 Package *xintgcd* implementation

```

255 {%
256   \expandafter\XINT_euc_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
257 }%

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

258 \def\XINT_euc_fork #1#2\Z #3#4\Z
259 {%
260   \xint_UDzerofork
261     #1\XINT_euc_BisZero
262     #3\XINT_euc_AisZero
263     0\XINT_euc_a
264   \krof
265   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
266 }%

Le {} pour protéger {{A}{B}} si on s'arrête après une étape (B divise A). On va renvoyer :
{N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}...{qN}{rN=0}

267 \def\XINT_euc_AisZero #1#2#3#4#5#6{{1}{0}{#2}{#2}{0}{0}}%
268 \def\XINT_euc_BisZero #1#2#3#4#5#6{{1}{0}{#3}{#3}{0}{0}}%

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

269 \def\XINT_euc_a #1#2#3%
270 {%
271   \expandafter\XINT_euc_b\the\numexpr #1+\xint_c_i\expandafter.%
272   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
273 }%

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

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

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

278 \def\XINT_euc_c #1#2\Z
279 {%
280   \xint_gob_til_zero #1\XINT_euc_end0\XINT_euc_a
281 }%

{n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}...}\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}

282 \def\XINT_euc_end0\XINT_euc_a #1#2#3#4\Z%
283 {%
284   \expandafter\XINT_euc_end_a
285   \romannumeral0%
286   \XINT_rord_main {#4}{#1}{#3}}%

```

```

287 \xint:
288 \xint_bye\xint_bye\xint_bye\xint_bye
289 \xint_bye\xint_bye\xint_bye\xint_bye
290 \xint:
291 }%
292 \def\xINT_euc_end_a #1#2#3{{#1}{#3}{#2}}%

```

6.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

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

```

293 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
294 \def\xintbezoutalgorithm #1%
295 {%
296 \expandafter \XINT_bezalg
297 \expandafter{\romannumeral0\xintiiabs{\xintNum{#1}}}%
298 }%
299 \def\xINT_bezalg #1#2%
300 {%
301 \expandafter\xINT_bezalg_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
302 }%

```

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

```

303 \def\xINT_bezalg_fork #1#2\Z #3#4\Z
304 {%
305 \xint_UDzerofork
306 #1\xINT_bezalg_BisZero
307 #3\xINT_bezalg_AisZero
308 0\xINT_bezalg_a
309 \krof
310 0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{\}\Z
311 }%
312 \def\xINT_bezalg_AisZero #1#2#3\Z{{1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
313 \def\xINT_bezalg_BisZero #1#2#3#4\Z{{1}{0}{0}{1}{#3}{#3}{1}{0}{0}{0}{0}{1}}%

```

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

```

314 \def\xINT_bezalg_a #1#2#3%
315 {%
316 \expandafter\xINT_bezalg_b\the\numexpr #1+\xint_c_i\expandafter.%
317 \romannumeral0\xINT_div_prepare {#2}{#3}{#2}%
318 }%

```

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

```

319 \def\xINT_bezalg_b #1.#2#3#4#5#6#7#8%
320 {%
321 \expandafter\xINT_bezalg_c\expandafter
322 {\romannumeral0\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%

```

6 Package *xintgcd* implementation

```

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

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

326 \def\XINT_beza1g_c #1#2#3#4#5#6%
327 {%
328     \expandafter\XINT_beza1g_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
329 }%

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

330 \def\XINT_beza1g_d #1#2#3#4#5#6#7#8%
331 {%
332     \XINT_beza1g_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{#3}{#4}{#1}{#6}}%
333 }%

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

334 \def\XINT_beza1g_e #1#2\Z
335 {%
336     \xint_gob_til_zero #1\XINT_beza1g_end0\XINT_beza1g_a
337 }%

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

338 \def\XINT_beza1g_end0\XINT_beza1g_a #1#2#3#4#5#6#7#8\Z
339 {%
340     \expandafter\XINT_beza1g_end_a
341     \romannumeral0%
342     \XINT_rord_main {}#8{{#1}{#3}}%
343     \xint:
344     \xint_bye\xint_bye\xint_bye\xint_bye
345     \xint_bye\xint_bye\xint_bye\xint_bye
346     \xint:
347 }%

{N}{D}{A}{B}{q1}{r1}{alpha1=q1}{beta1=1}{q2}{r2}{alpha2}{beta2}
...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

348 \def\XINT_beza1g_end_a #1#2#3#4{{#1}{#3}{0}{1}}{#2}{#4}{1}{0}}%

```

6.8 `\xintGCdof`

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

```

349 \def\xintGCdof      {\romannumeral0\xintgcdof }%
350 \def\xintgcdof     #1{\expandafter\XINT_gcdoof_a\romannumeral`&&@#1\xint:}%
351 \def\XINT_gcdoof_a #1{\expandafter\XINT_gcdoof_b\romannumeral`&&@#1!}%
352 \def\XINT_gcdoof_b #1!#2{\expandafter\XINT_gcdoof_c\romannumeral`&&@#2!{#1!}%
353 \def\XINT_gcdoof_c #1{\xint_gob_til_xint: #1\XINT_gcdoof_e\xint:\XINT_gcdoof_d #1}%
354 \def\XINT_gcdoof_d #1!{\expandafter\XINT_gcdoof_b\romannumeral0\xintgcd {#1}}%
355 \def\XINT_gcdoof_e #1!#2!{ #2}%

```

6.9 `\xintLCMof`

New with 1.09a

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

```

356 \def\xintLCMof     {\romannumeral0\xintlcmof }%
357 \def\xintlcmof     #1{\expandafter\XINT_lcmof_a\romannumeral`&&@#1\xint:}%
358 \def\XINT_lcmof_a #1{\expandafter\XINT_lcmof_b\romannumeral`&&@#1!}%
359 \def\XINT_lcmof_b #1!#2{\expandafter\XINT_lcmof_c\romannumeral`&&@#2!{#1!}%
360 \def\XINT_lcmof_c #1{\xint_gob_til_xint: #1\XINT_lcmof_e\xint:\XINT_lcmof_d #1}%
361 \def\XINT_lcmof_d #1!{\expandafter\XINT_lcmof_b\romannumeral0\xintlcm {#1}}%
362 \def\XINT_lcmof_e #1!#2!{ #2}%

```

6.10 `\xintTypesetEuclideanAlgorithm`

TYPESETTING

Organisation:

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

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

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

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

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

```

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

```



```

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

```

6.11 \xintTypesetBezoutAlgorithm

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

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

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

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

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

```

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

```

6 Package *xintgcd* implementation

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

7 Package `xintfrac` implementation

.1	Catcodes, ε - \TeX and reload detection	179	.41	<code>\xintBinomial</code>	213
.2	Package identification	180	.42	<code>\xintPFactorial</code>	214
.3	<code>\XINT_cntSgnFork</code>	181	.43	<code>\xintPrd</code>	214
.4	<code>\xintLen</code>	181	.44	<code>\xintDiv</code>	214
.5	<code>\XINT_outfrac</code>	181	.45	<code>\xintDivFloor</code>	215
.6	<code>\XINT_inFrac</code>	182	.46	<code>\xintDivTrunc</code>	215
.7	<code>\XINT_frac_gen</code>	184	.47	<code>\xintDivRound</code>	215
.8	<code>\XINT_factortens</code>	186	.48	<code>\xintModTrunc</code>	215
.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>\xintIsOne</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>	187	.49	<code>\xintDivMod</code>	216
.10	<code>\xintRaw</code>	189	.50	<code>\xintMod</code>	217
.11	<code>\xintPRaw</code>	189	.51	<code>\xintIsOne</code>	218
.12	<code>\xintRawWithZeros</code>	190	.52	<code>\xintGeq</code>	218
.13	<code>\xintDecToString</code>	190	.53	<code>\xintMax</code>	220
.14	<code>\xintFloor</code> , <code>\xintiFloor</code>	191	.54	<code>\xintMaxof</code>	220
.15	<code>\xintCeil</code> , <code>\xintiCeil</code>	191	.55	<code>\xintMin</code>	221
.16	<code>\xintNumerator</code>	191	.56	<code>\xintMinof</code>	221
.17	<code>\xintDenominator</code>	191	.57	<code>\xintCmp</code>	222
.18	<code>\xintFrac</code>	192	.58	<code>\xintAbs</code>	223
.19	<code>\xintSignedFrac</code>	192	.59	<code>\xintOpp</code>	223
.20	<code>\xintFwOver</code>	193	.60	<code>\xintSgn</code>	223
.21	<code>\xintSignedFwOver</code>	193	.61	Floating point macros	223
.22	<code>\xintREZ</code>	194	.62	<code>\xintFloat</code>	224
.23	<code>\xintE</code>	195	.63	<code>\XINTinFloat</code> , <code>\XINTinFloatS</code>	225
.24	<code>\xintIrr</code> , <code>\xintPIrr</code>	195	.64	<code>\xintPFloat</code>	232
.25	<code>\xintifInt</code>	197	.65	<code>\XINTinFloatFracdigits</code>	233
.26	<code>\xintJrr</code>	197	.66	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code>	234
.27	<code>\xintTFrac</code>	198	.67	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code>	235
.28	<code>\xintTrunc</code> , <code>\xintiTrunc</code>	199	.68	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code>	236
.29	<code>\xintTTrunc</code>	201	.69	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code>	236
.30	<code>\xintNum</code>	201	.70	<code>\xintFloatPow</code> , <code>\XINTinFloatPow</code>	237
.31	<code>\xintRound</code> , <code>\xintiRound</code>	201	.71	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code>	241
.32	<code>\xintXTrunc</code>	202	.72	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code>	245
.33	<code>\xintDigits</code>	209	.73	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatPFactorial</code>	250
.34	<code>\xintAdd</code>	209	.74	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBinomial</code>	254
.35	<code>\xintSub</code>	211	.75	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code>	256
.36	<code>\xintSum</code>	211	.76	<code>\xintFloatE</code> , <code>\XINTinFloatE</code>	258
.37	<code>\xintMul</code>	211	.77	<code>\XINTinFloatMod</code>	259
.38	<code>\xintSqr</code>	212	.78	<code>\XINTinFloatDivFloor</code>	259
.39	<code>\xintPow</code>	212	.79	<code>\XINTinFloatDivMod</code>	260
.40	<code>\xintFac</code>	213	.80	<code>\xintifFloatInt</code>	260
			.81	(WIP) <code>\XINTinRandomFloatS</code>	260
			.82	(WIP) <code>\XINTinRandomFloatSixteen</code>	261

The commenting is currently (2018/05/18) very sparse.

7.1 Catcodes, ε - \TeX and reload detection

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

7 Package *xintfrac* implementation

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

7.2 Package identification

```

49 \XINT_providespackage
50 \ProvidesPackage{xintfrac}%
51 [2018/05/18 1.3b Expandable operations on fractions (JFB)]%

```

7.3 `\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.

```

52 \def\XINT_cntSgnFork #1%
53 {%
54   \ifcase #1\expandafter\xint_secondofthree
55     \or\expandafter\xint_thirdofthree
56     \else\expandafter\xint_firstofthree
57   \fi
58 }%

```

7.4 `\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.

```

59 \def\xintLen {\romannumeral0\xintlen }%
60 \def\xintlen #1%
61 {%
62   \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
63 }%
64 \def\XINT_flen#1{\def\XINT_flen ##1##2##3%
65 }%
66   \expandafter#1%

67   \the\numexpr \XINT_abs##1+%
68   \XINT_len_fork ##2##3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
69   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
70   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-\xint_c_i
71   \relax
72 }}\XINT_flen{ }%

```

7.5 `\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.

```

73 \def\XINT_outfrac #1#2#3%
74 {%
75   \ifcase\XINT_cntSgn #3\xint:
76     \expandafter \XINT_outfrac_divisionbyzero
77   \or
78     \expandafter \XINT_outfrac_P

```

```

79   \else
80     \expandafter \XINT_outfrac_N
81   \fi
82   {#2}{#3}[#1]%
83 }%
84 \def\XINT_outfrac_divisionbyzero #1#2%
85 {%
86   \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}{0/1[0]}%
87 }%
88 \def\XINT_outfrac_P#1{%
89 \def\XINT_outfrac_P ##1##2%
90   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi#1##1/##2}%
91 }\XINT_outfrac_P{ }%
92 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
93 \def\XINT_outfrac_N #1#2%
94 {%
95   \expandafter\XINT_outfrac_N_a\expandafter
96   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
97 }%
98 \def\XINT_outfrac_N_a #1#2%
99 {%
100   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
101 }%

```

7.6 `\XINT_inFrac`

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

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

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

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

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

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

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

7 Package *xintfrac* implementation

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`.

```

102 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
103 \def\XINT_infrac #1%
104 {%
105   \expandafter\XINT_infrac_fork\romannumeral`&&@#1\xint:\XINT_W[\XINT_W\XINT_T
106 }%
107 \def\XINT_infrac_fork #1[#2%
108 {%
109   \xint_UDXINTWfork
110   #2\XINT_frac_gen          % input has no brackets [N]
111   \XINT_W\XINT_infrac_res_a % there is some [N], must be strict A[N] or A/B[N] input
112   \krof
113   #1[#2%
114 }%
115 \def\XINT_infrac_res_a #1%
116 {%
117   \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
118 }%

```

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

```

119 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
120 \def\XINT_infrac_res_b #1/#2%
121 {%
122   \xint_UDXINTWfork
123   #2\XINT_infrac_res_ca      % it was A[N] input
124   \XINT_W\XINT_infrac_res_cb % it was A/B[N] input
125   \krof
126   #1/#2%
127 }%

```

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.

```

128 \def\XINT_infrac_res_ca #1[#2]\xint:\XINT_W[\XINT_W\XINT_T

```

```

129   {\expandafter{\the\numexpr #2}{#1}{1}}%
130 \def\XINT_infrac_res_cb #1/#2[%
131   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[]%
132 \def\XINT_infrac_res_cc #1~#2[#3]\xint:/\XINT_W[\XINT_W\XINT_T
133   {\expandafter{\the\numexpr #3}{#2}{#1}}%

```

7.7 \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`.

```

134 \def\XINT_frac_gen #1/#2%
135 {%
136   \xint_UDXINTWfork
137   #2\XINT_frac_gen_A      % there was no /
138   \XINT_W\XINT_frac_gen_B % there was a /
139   \krof
140   #1/#2%
141 }%

```

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

```

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

```


7 Package *xintfrac* implementation

```

168     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
169     ~#2#1!%
170 }%
171 \def\xINT_frac_gen_C #1!#2.#3%
172 {%
173     \xint_UDXINTWfork
174     #3\xINT_frac_gen_Ca
175     \XINT_W\xINT_frac_gen_Cb
176     \krof
177     #1!#2.#3%
178 }%
179 \def\xINT_frac_gen_Ca #1~#2!#3e#4e#5\xINT_T
180 {%
181     \expandafter\xINT_frac_gen_F\the\numexpr #4-#1\expandafter
182     ~\romannumeral0\expandafter\xINT_num_cleanup\the\numexpr\xINT_num_loop
183     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
184 }%
185 \def\xINT_frac_gen_Cb #1.#2e%
186 {%
187     \expandafter\xINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
188 }%
189 \def\xINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\xINT_T
190 {%
191     \expandafter\xINT_frac_gen_F\the\numexpr #5-#2-%

192     \numexpr\xINT_length_loop
193     #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
194     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
195     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye

196     \relax\expandafter~%
197     \romannumeral0\expandafter\xINT_num_cleanup\the\numexpr\xINT_num_loop
198     #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
199     ~#4#1~%
200 }%
201 \def\xINT_frac_gen_F #1~#2%
202 {%
203     \xint_UDzerominusfork
204     #2-\XINT_frac_gen_Gdivbyzero
205     0#2{\XINT_frac_gen_G -{}}%
206     0-{\XINT_frac_gen_G }#2}%
207     \krof #1~%
208 }%
209 \def\xINT_frac_gen_Gdivbyzero #1~~#2~%
210 {%
211     \expandafter\xINT_frac_gen_Gdivbyzero_a
212     \romannumeral0\expandafter\xINT_num_cleanup\the\numexpr\xINT_num_loop
213     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%
214 }%
215 \def\xINT_frac_gen_Gdivbyzero_a #1~#2~%

```



```

258 \def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
259 \def\XINT_factortens_e #1..#2.%
260   {\expandafter.\expandafter\XINT_factortens_c
261     \the\numexpr\xint_c_ix+#2.}%

```

7.9 `\xintEq`, `\xintNotEq`, `\xintGt`, `\xintLt`, `\xintGtorEq`, `\xintLtorEq`, `\xintIsZero`, `\xintIsNotZero`, `\xintIsOne`, `\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`.

```

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

```

7 Package *xintfrac* implementation

```

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

```

7 Package *xintfrac* implementation

```
353     \expandafter\xint_firstoftwo_thenstop
354   \fi
355 }%
356 \def\xintifOne {\romannumeral0\xintifone }%
357 \def\xintifone #1%
358 {%
359   \if1\xintIsOne{#1}%
360     \expandafter\xint_firstoftwo_thenstop
361   \else
362     \expandafter\xint_secondoftwo_thenstop
363   \fi
364 }%
365 \def\xintifOdd {\romannumeral0\xintifodd }%
366 \def\xintifodd #1%
367 {%
368   \if\xintOdd{#1}1%
369     \expandafter\xint_firstoftwo_thenstop
370   \else
371     \expandafter\xint_secondoftwo_thenstop
372   \fi
373 }%
```

7.10 `\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.

```
374 \def\xintRaw {\romannumeral0\xintraw }%
375 \def\xintraw
376 {%
377   \expandafter\XINT_raw\romannumeral0\XINT_infrac
378 }%
379 \def\XINT_raw #1#2#3{ #2/#3[#1]}%
```

7.11 `\xintPRaw`

1.09b

```
380 \def\xintPRaw {\romannumeral0\xintpraw }%
381 \def\xintpraw
382 {%
383   \expandafter\XINT_praw\romannumeral0\XINT_infrac
384 }%
385 \def\XINT_praw #1%
386 {%
387   \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
388 }%
389 \def\XINT_praw_A #1#2#3%
390 {%
391   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
392     \else\expandafter\xint_secondoftwo
393   \fi { #2[#1]}{ #2/#3[#1]}%
394 }%
```

```

395 \def\XINT_praw_a\XINT_praw_A #1#2#3%
396 {%
397   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
398       \else\expandafter\xint_secondoftwo
399   \fi { #2}{ #2/#3}%
400 }%

```

7.12 `\xintRawWithZeros`

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

```

401 \def\xintRawWithZeros {\romannumeral0\xintraewithzeros }%
402 \def\xintraewithzeros
403 {%
404   \expandafter\XINT_raw_fork\romannumeral0\XINT_infrac
405 }%

```

```

406 \def\XINT_raw_fork #1%
407 {%
408   \ifnum#1<\xint_c_
409     \expandafter\XINT_rawz_Ba
410   \else
411     \expandafter\XINT_rawz_A
412   \fi
413   #1.%
414 }%
415 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
416 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
417   \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
418 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

7.13 `\xintDecToString`

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

```

419 \def\xintDecToString{\romannumeral0\xintdectosttring}%
420 \def\xintdectosttring#1{\expandafter\XINT_dectostr\romannumeral0\xintra{#1}}%
421 \def\XINT_dectostr #1/#2[#3]{\xintiifZero {#1}%
422   \XINT_dectostr_z
423   {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
424     \else\expandafter\XINT_dectostr_b
425   \fi}%
426   #1/#2[#3]}%
427 }%
428 \def\XINT_dectostr_z#1[#2]{ 0}%
429 \def\XINT_dectostr_a#1/#2[#3]{%
430   \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
431   \xint_orthat{\xintiie{#1}#3}}%
432 }%
433 \def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow

```

```

434 \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
435 \xint_orthat{\xintiie{#1}{#3}/#2}%
436 }%

```

7.14 `\xintFloor`, `\xintiFloor`

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

```

437 \def\xintFloor {\romannumeral0\xintfloor }%
438 \def\xintfloor #1% devrais-je faire \xintREZ?
439   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}./1[0]}%
440 \def\xintiFloor {\romannumeral0\xintifloor }%
441 \def\xintifloor #1%
442   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}.}%
443 \def\XINT_ifloor #1/#2.{\xintiigo {#1}{#2}}%

```

7.15 `\xintCeil`, `\xintiCeil`

1.09a

```

444 \def\xintCeil {\romannumeral0\xintceil }%
445 \def\xintceil #1{\xintiiopp {\xintFloor {\xintOpp{#1}}}}%
446 \def\xintiCeil {\romannumeral0\xinticeil }%
447 \def\xinticeil #1{\xintiiopp {\xintiFloor {\xintOpp{#1}}}}%

```

7.16 `\xintNumerator`

```

448 \def\xintNumerator {\romannumeral0\xintnumerator }%
449 \def\xintnumerator
450 {%
451   \expandafter\XINT_numer\romannumeral0\XINT_infrac
452 }%
453 \def\XINT_numer #1%
454 {%
455   \ifcase\XINT_cntSgn #1\xint:
456     \expandafter\XINT_numer_B
457   \or
458     \expandafter\XINT_numer_A
459   \else
460     \expandafter\XINT_numer_B
461   \fi
462   {#1}%
463 }%
464 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
465 \def\XINT_numer_B #1#2#3{ #2}%

```

7.17 `\xintDenominator`

```

466 \def\xintDenominator {\romannumeral0\xintdenominator }%
467 \def\xintdenominator
468 {%
469   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
470 }%

```

```

471 \def\XINT_denom_fork #1%
472 {%

473   \ifnum#1<\xint_c_
474     \expandafter\XINT_denom_B
475   \else
476     \expandafter\XINT_denom_A
477   \fi
478   #1.%
479 }%
480 \def\XINT_denom_A #1.#2#3{ #3}%
481 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

7.18 `\xintFrac`

Useless typesetting macro.

```

482 \def\xintFrac {\romannumeral0\xintfrac }%
483 \def\xintfrac #1%
484 {%
485   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
486 }%
487 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
488 \catcode`^=7
489 \def\XINT_fracfrac_B #1#2\Z
490 {%
491   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
492 }%
493 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
494 {%
495   \if1\XINT_isOne {#3}%
496     \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
497   \fi
498   \space
499   \frac {#2}{#3}%
500 }%
501 \def\XINT_fracfrac_D #1#2#3%
502 {%
503   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
504   \space
505   \frac {#2}{#3}#1%
506 }%
507 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

7.19 `\xintSignedFrac`

```

508 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
509 \def\xintsignedfrac #1%
510 {%
511   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
512 }%
513 \def\XINT_sgnfrac_a #1#2%

```



```

514 {%
515   \XINT_sgnfrac_b #2\Z {#1}%
516 }%
517 \def\XINT_sgnfrac_b #1%
518 {%
519   \xint_UDsignfork
520   #1\XINT_sgnfrac_N
521   -{\XINT_sgnfrac_P #1}%
522   \krof
523 }%
524 \def\XINT_sgnfrac_P #1\Z #2%
525 {%
526   \XINT_fracfrac_A {#2}{#1}%
527 }%
528 \def\XINT_sgnfrac_N
529 {%
530   \expandafter-\romannumeral0\XINT_sgnfrac_P
531 }%

```

7.20 `\xintFwOver`

```

532 \def\xintFwOver {\romannumeral0\xintfwover }%
533 \def\xintfwover #1%
534 {%
535   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
536 }%
537 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
538 \def\XINT_fwover_B #1#2\Z
539 {%
540   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
541 }%
542 \catcode`^=11
543 \def\XINT_fwover_C #1#2#3#4#5%
544 {%
545   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
546   \else\xint_afterfi { #4}%
547   \fi
548 }%
549 \def\XINT_fwover_D #1#2#3%
550 {%
551   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
552   \else\xint_afterfi { #2\cdot }%
553   \fi
554   #1%
555 }%

```

7.21 `\xintSignedFwOver`

```

556 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
557 \def\xintsignedfwover #1%
558 {%
559   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
560 }%
561 \def\XINT_sgnfwover_a #1#2%
562 {%

```

```

563 \XINT_sgnfwover_b #2\Z {#1}%
564 }%
565 \def\XINT_sgnfwover_b #1%
566 {%
567 \xint_UDsignfork
568 #1\XINT_sgnfwover_N
569 -{\XINT_sgnfwover_P #1}%
570 \krof
571 }%
572 \def\XINT_sgnfwover_P #1\Z #2%
573 {%
574 \XINT_fwover_A {#2}{#1}%
575 }%
576 \def\XINT_sgnfwover_N
577 {%
578 \expandafter-\romannumeral0\XINT_sgnfwover_P
579 }%

```

7.22 \xintREZ

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

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

```

580 \def\xintREZ {\romannumeral0\xintrez }%
581 \def\xintrez
582 {%
583 \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
584 }%
585 \def\XINT_rez_A #1#2%
586 {%
587 \XINT_rez_AB #2\Z {#1}%
588 }%
589 \def\XINT_rez_AB #1%
590 {%
591 \xint_UDzerominusfork
592 #1-\XINT_rez_zero
593 0#1\XINT_rez_neg
594 0-{\XINT_rez_B #1}%
595 \krof
596 }%
597 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
598 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
599 \def\XINT_rez_B #1\Z
600 {%
601 \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
602 }%
603 \def\XINT_rez_C #1.#2.#3#4%
604 {%
605 \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
606 }%
607 \def\XINT_rez_D #1.#2.#3.%
608 {%
609 \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%

```

```
610 }%
611 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%
```

7.23 \xintE

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

```
612 \def\xintE {\romannumeral0\xinte }%
613 \def\xinte #1%
614 {%
615   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
616 }%
617 \def\XINT_e #1#2#3#4%
618 {%
619   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
620 }%
621 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%
```

7.24 \xintIrr, \xintPIrr

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

```
622 \def\xintIrr {\romannumeral0\xintirr }%
623 \def\xintPIrr{\romannumeral0\xintpirr }%
624 \def\xintirr #1%
625 {%
626   \expandafter\XINT_irr_start\romannumeral0\xintraewithzeros {#1}\Z
627 }%
628 \def\xintpirr #1%
629 {%
630   \expandafter\XINT_pirr_start\romannumeral0\xintraew{#1}%
631 }%
632 \def\XINT_irr_start #1#2/#3\Z
633 {%
634   \if0\XINT_isOne {#3}%
635     \xint_afterfi
636       {\xint_UDsignfork
637         #1\XINT_irr_negative
638         -{\XINT_irr_nonneg #1}%
639         \krof}%
640   \else
641     \xint_afterfi{\XINT_irr_denomiseone #1}%
642   \fi
643   #2\Z {#3}%
644 }%
645 \def\XINT_pirr_start #1#2/#3[%
646 {%
647   \if0\XINT_isOne {#3}%
648     \xint_afterfi
649       {\xint_UDsignfork
650         #1\XINT_irr_negative
651         -{\XINT_irr_nonneg #1}%
```

7 Package *xintfrac* implementation

```

652         \krof}%
653     \else
654         \xint_afterfi{\XINT_irr_denomisine #1}%
655     \fi
656     #2\Z {#3}[%
657 }%
658 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
659 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
660 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
661 \def\XINT_irr_D #1#2\Z #3#4\Z
662 {%
663     \xint_UDzerosfork
664         #3#1\XINT_irr_indeterminate
665         #30\XINT_irr_divisionbyzero
666         #10\XINT_irr_zero
667         00\XINT_irr_loop_a
668     \krof
669     {#3#4}{#1#2}{#3#4}{#1#2}%
670 }%
671 \def\XINT_irr_indeterminate #1#2#3#4#5%
672 {%
673     \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{{0/1}%
674 }%
675 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
676 {%
677     \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{{0/1}%
678 }%
679 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
680 \def\XINT_irr_loop_a #1#2%
681 {%
682     \expandafter\XINT_irr_loop_d
683     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
684 }%
685 \def\XINT_irr_loop_d #1#2%
686 {%
687     \XINT_irr_loop_e #2\Z
688 }%
689 \def\XINT_irr_loop_e #1#2\Z
690 {%
691     \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
692 }%
693 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
694 {%
695     \expandafter\XINT_irr_loop_exitb\expandafter
696     {\romannumeral0\xintiique {#3}{#2}}%
697     {\romannumeral0\xintiique {#4}{#2}}%
698 }%
699 \def\XINT_irr_loop_exitb #1#2%
700 {%
701     \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
702 }%
703 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

7.25 `\xintifInt`

```

704 \def\xintifInt  {\romannumeral0\xintifint }%
705 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawwithzeros {#1}.}%
706 \def\XINT_ifint #1/#2.%
707 {%
708   \if 0\xintiiRem {#1}{#2}%
709   \expandafter\xint_firstoftwo_thenstop
710   \else
711   \expandafter\xint_secondoftwo_thenstop
712   \fi
713 }%

```

7.26 `\xintJrr`

```

714 \def\xintJrr {\romannumeral0\xintjrr }%
715 \def\xintjrr #1%
716 {%
717   \expandafter\XINT_jrr_start\romannumeral0\xintraawwithzeros {#1}\Z
718 }%
719 \def\XINT_jrr_start #1#2/#3\Z
720 {%
721   \if0\XINT_isOne {#3}\xint_afterfi
722     {\xint_UDsignfork
723       #1\XINT_jrr_negative
724       -{\XINT_jrr_nonneg #1}%
725       \krof}%
726   \else
727     \xint_afterfi{\XINT_jrr_denomisine #1}%
728   \fi
729   #2\Z {#3}%
730 }%
731 \def\XINT_jrr_denomisine #1\Z #2{ #1/1}% changed in 1.08
732 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
733 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
734 \def\XINT_jrr_D #1#2\Z #3#4\Z
735 {%
736   \xint_UDzerosfork
737     #3#1\XINT_jrr_indeterminate
738     #30\XINT_jrr_divisionbyzero
739     #10\XINT_jrr_zero
740     00\XINT_jrr_loop_a
741   \krof
742   {#3#4}{#1#2}1001%
743 }%
744 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
745 {%
746   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
747 }%
748 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
749 {%
750   \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%
751 }%
752 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08

```

7 Package *xintfrac* implementation

```

753 \def\XINT_jrr_loop_a #1#2%
754 {%
755   \expandafter\XINT_jrr_loop_b
756   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
757 }%
758 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
759 {%
760   \expandafter \XINT_jrr_loop_c \expandafter
761     {\romannumeral0\xintiiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
762     {\romannumeral0\xintiiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
763     {#2}{#3}{#4}{#5}%
764 }%
765 \def\XINT_jrr_loop_c #1#2%
766 {%
767   \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
768 }%
769 \def\XINT_jrr_loop_d #1#2#3#4%
770 {%
771   \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
772 }%
773 \def\XINT_jrr_loop_e #1#2\Z
774 {%
775   \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
776 }%
777 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
778 {%
779   \XINT_irr_finish {#3}{#4}%
780 }%

```

7.27 `\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`.

```

781 \def\xintTFrac {\romannumeral0\xinttfrac }%
782 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
783 \def\XINT_tfrac_fork #1%
784 {%
785   \xint_UDzerominusfork
786     #1-\XINT_tfrac_zero
787     0#1{\xintiiopp\XINT_tfrac_P }%
788     0-{\XINT_tfrac_P #1}%
789   \krof
790 }%
791 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
792 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
793   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

7.28 `\xintTrunc`, `\xintiTrunc`

1.2i release notes: ever since its inception this macro was stupid for a decimal input: it did not handle it separately from the general fraction case $A/B[N]$ with $B>1$, hence ended up doing divisions by powers of ten. But this meant that nesting `\xintTrunc` with itself was very inefficient.

1.2i version is better. However it still handles $B>1$, $N<0$ via adding zeros to B and dividing with this extended B . A possibly more efficient approach is implemented in `\xintXTrunc`, but its logic is more complicated, the code is quite longer and making it f-expandable would not shorten it... I decided for the time being to not complicate things here.

```

794 \def\xintTrunc  {\romannumeral0\xinttrunc }%
795 \def\xintiTrunc {\romannumeral0\xintitrunc}%
796 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
797 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
798 \def\XINT_trunc #1.#2#3%
799 {%
800   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
801 }%
802 \def\XINT_trunc_a #1#2#3#4.#5%
803 {%
804   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
805   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
806   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
807 }%
808 \def\XINT_trunc_zero #1.#2.{ 0}%

809 \def\XINT_trunc_b      {\expandafter\XINT_trunc_B\the\numexpr}%
810 \def\XINT_trunc_sp_b  {\expandafter\XINT_trunc_sp_B\the\numexpr}%

811 \def\XINT_trunc_B #1%
812 {%
813   \xint_UDsignfork
814   #1\XINT_trunc_C
815   -\XINT_trunc_D
816   \krof #1%
817 }%

818 \def\XINT_trunc_sp_B #1%
819 {%
820   \xint_UDsignfork
821   #1\XINT_trunc_sp_C
822   -\XINT_trunc_sp_D
823   \krof #1%
824 }%

825 \def\XINT_trunc_C -#1.#2#3%
826 {%

```

7 Package *xintfrac* implementation

```

827 \expandafter\XINT_trunc_CE
828 \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
829 }%
830 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

831 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.%}
832 \def\XINT_trunc_sp_Ca #1%
833 {%
834 \xint_UDsignfork
835 #1{\XINT_trunc_sp_Cb -}%
836 -{\XINT_trunc_sp_Cb \space#1}%
837 \krof
838 }%
839 \def\XINT_trunc_sp_Cb #1#2.#3.%
840 {%
841 \expandafter\XINT_trunc_sp_Cc

842 \romannumeral0\expandafter\XINT_split_fromright_a
843 \the\numexpr#3-\numexpr\XINT_length_loop
844 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
845 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
846 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
847 .#2\xint_bye2345678\xint_bye..#1%
848 }%

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

```



```

872     -{\XINT_trunc_F \space#1}%
873   \krof
874 }%
875 \def\XINT_trunc_sp_E #1%
876 {%
877   \xint_UDsignfork
878     #1{\XINT_trunc_sp_F -}%
879     -{\XINT_trunc_sp_F\space#1}%
880   \krof
881 }%
882 \def\XINT_trunc_F #1#2.#3#4%
883   {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
884     \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
885 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
886 \def\XINT_itrunc_G #1#2.#3#4.{\if#10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
887 \def\XINT_trunc_G #1.#2#3.%
888 {%
889   \expandafter\XINT_trunc_H
890   \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
891 }%
892 \def\XINT_trunc_H #1.#2.%
893 {%
894   \ifnum #1 > \xint_c_
895     \xint_afterfi {\XINT_trunc_Ha {#2}}%
896   \else
897     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
898   \fi
899 }%
900 \def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
901 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
902 \def\XINT_trunc_Hb #1#2#3%
903 {%
904   \expandafter #3\expandafter0\expandafter.%

905   \romannumeral\xintreplicate{#1}0#2%
906 }%

```

7.29 `\xintTTrunc`

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

```

907 \def\xintTTrunc {\romannumeral0\xintttrunc }%
908 \def\xintttrunc {\xintitrunc\xint_c_}%

```

7.30 `\xintNum`

```
909 \let\xintnum \xintttrunc
```

7.31 `\xintRound`, `\xintiRound`

Modified in 1.2i.

7 Package *xintfrac* implementation

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`.

```
910 \def\xintRound {\romannumeral0\xintround }%
911 \def\xintiRound {\romannumeral0\xintiround }%
912 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
913 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
914 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
915 \def\XINT_round_aa #1.#2.#3#4%
916 {%
917   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
918 }%
919 \def\XINT_round_a #1#2#3#4.%
920 {%
921   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
922   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
923   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
924 }%
925 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
926 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
927 \def\XINT_round_B #1.%
928   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%
```

7.32 `\xintXTrunc`

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

- no more use of `\xintiloop` from `xinttools.sty` (replaced by `\xintreplicate...` from `xintkernel.sty`),
 - no more use in $O>N>-D$ 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.

```
929 \def\xintXTrunc #1#2%
930 {%
931   \expandafter\XINT_xtrunc_a
932   \the\numexpr #1\expandafter.\romannumeral0\xintraw
933 }%
934 \def\XINT_xtrunc_a #1.% ?? faire autre chose
935 {%
936   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
937 }%

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

939 \def\XINT_xtrunc_c #1%
940 {%
941   \xint_UDzerominusfork
```

7 Package *xintfrac* implementation

```

942     #1-\XINT_xtrunc_zero
943     0#1{-\XINT_xtrunc_d {}}%
944     0-{\XINT_xtrunc_d #1}%
945     \krof
946 }%[
947 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

948 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
949 {%
950     \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W
951     !{#4};{#5}{#2}{#1#3}%
952 }%
953 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
954 {%
955     \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
956     \XINT_xtrunc_prepare_b #9%
957 }%
958 \def\XINT_xtrunc_prepare_small\R #1!#2;%
959 {%
960     \ifcase #2
961     \or\expandafter\XINT_xtrunc_BisOne
962     \or\expandafter\XINT_xtrunc_BisTwo
963     \or
964     \or\expandafter\XINT_xtrunc_BisFour
965     \or\expandafter\XINT_xtrunc_BisFive
966     \or
967     \or
968     \or\expandafter\XINT_xtrunc_BisEight
969     \fi\XINT_xtrunc_BisSmall {#2}%
970 }%

971 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
972     {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
973 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
974 {%
975     \expandafter\XINT_xtrunc_sp_e\expandafter
976     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
977     {\romannumeral0\xintiimul 5{#4}}{#3}%
978 }%
979 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
980 {%
981     \expandafter\XINT_xtrunc_sp_e\expandafter
982     {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
983     {\romannumeral0\xintiimul {25}{#4}}{#3}%
984 }%
985 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
986 {%
987     \expandafter\XINT_xtrunc_sp_e\expandafter
988     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
989     {\romannumeral0\xintdouble {#4}}{#3}%

```

7 Package *xintfrac* implementation

```

990 }%
991 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
992 {%
993   \expandafter\XINT_xtrunc_sp_e\expandafter
994   {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
995   {\romannumeral0\xintiimul {125}{#4}}{#3}%
996 }%
997 \def\XINT_xtrunc_BisSmall #1%
998 {%
999   \expandafter\XINT_xtrunc_e\expandafter
1000   {\expandafter\XINT_xtrunc_small_a
1001   \the\numexpr #1/\xint_c_ii\expandafter
1002   .\the\numexpr \xint_c_x^viii+#1!}%
1003 }%

1004 \def\XINT_xtrunc_small_a #1.#2!#3%
1005 {%
1006   \expandafter\XINT_div_small_b\the\numexpr #1\expandafter
1007   \xint:\the\numexpr #2\expandafter!%
1008   \romannumeral0\XINT_div_small_ba #3\R\R\R\R\R\R\R\R\R\{10}0000001\W
1009   #3\XINT_sepbyviii_Z_end 2345678\relax
1010 }%

1011 \def\XINT_xtrunc_prepare_b
1012   {\expandafter\XINT_xtrunc_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1013 \def\XINT_xtrunc_prepare_c #1!%
1014 {%
1015   \XINT_xtrunc_prepare_d #1.00000000!{#1}%
1016 }%
1017 \def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1018 {%
1019   \expandafter\XINT_xtrunc_prepare_e
1020   \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1021 }%
1022 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
1023 {%
1024   \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1025 }%
1026 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1027 {%
1028   \expandafter\XINT_xtrunc_prepare_g\expandafter
1029   \XINT_div_prepare_g
1030   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1031   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1032   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1033   \xint:\romannumeral0\XINT_sepandrev_andcount
1034   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1035   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1036   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1037   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W

```

7 Package *xintfrac* implementation

```

1038 \X
1039 }%

1040 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

1041 \def\XINT_xtrunc_e #1#2%
1042 {%
1043   \ifnum #2<\xint_c_
1044     \expandafter\XINT_xtrunc_I
1045   \else
1046     \expandafter\XINT_xtrunc_II
1047   \fi #2\xint:{#1}%
1048 }%

1049 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1050 {%
1051   \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1052 }%

1053 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1054 {%
1055   \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1056 }%

1057 \def\XINT_xtrunc_I_b #1%
1058 {%
1059   \xint_UDsignfork
1060   #1\XINT_xtrunc_IA_c
1061   -\XINT_xtrunc_IB_c
1062   \krof #1%
1063 }%

1064 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1065 {%
1066   \expandafter\XINT_xtrunc_IA_d
1067   \the\numexpr#2-\xintLength{#6}\xint:{#6}%
1068   \expandafter\XINT_xtrunc_IA_xd
1069   \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1070 }%

1071 \def\XINT_xtrunc_IA_d #1%
1072 {%

```

7 Package *xintfrac* implementation

```
1073 \xint_UDsignfork
1074 #1\xINT_xtrunc_IAA_e
1075 -\XINT_xtrunc_IAB_e
1076 \krof #1%
1077 }%

1078 \def\xINT_xtrunc_IAA_e -#1\xint:#2%
1079 {%
1080 \romannumeral0\xINT_split_fromleft
1081 #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1082 }%

1083 \def\xINT_xtrunc_IAB_e #1\xint:#2%
1084 {%
1085 0.\romannumeral\xINT_rep#1\endcsname0#2%
1086 }%

1087 \def\xINT_xtrunc_IA_xd #1\xint:#2\xint:%
1088 {%
1089 \expandafter\xINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1090 }%

1091 \def\xINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
1092 {%
1093 \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
1094 }%

1095 \def\xINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
1096 {%
1097 \expandafter\xINT_xtrunc_IB_d
1098 \romannumeral0\xINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1099 }%

1100 \def\xINT_xtrunc_IB_d #1.#2.#3%
1101 {%
1102 \expandafter\xINT_xtrunc_IA_d\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1103 }%

1104 \def\xINT_xtrunc_II #1\xint:%
1105 {%
1106 \expandafter\xINT_xtrunc_II_a\romannumeral\xintreplicate{#1}0\xint:%
1107 }%
```


7 Package *xintfrac* implementation

```
1153 \def\XINT_xtrunc_sp_e #1%
1154 {%
1155   \ifnum #1<\xint_c_
1156     \expandafter\XINT_xtrunc_sp_I
1157   \else
1158     \expandafter\XINT_xtrunc_sp_II
1159   \fi #1\xint:%
1160 }%

1161 \def\XINT_xtrunc_sp_I -#1\xint:#2#3%
1162 {%
1163   \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
1164 }%

1165 \def\XINT_xtrunc_sp_I_a #1%
1166 {%
1167   \xint_UDsignfork
1168   #1\XINT_xtrunc_sp_IA_b
1169   -\XINT_xtrunc_sp_IB_b
1170   \krof #1%
1171 }%

1172 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1173 {%
1174   \expandafter\XINT_xtrunc_sp_IA_c
1175   \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1176 }%

1177 \def\XINT_xtrunc_sp_IA_c #1%
1178 {%
1179   \xint_UDsignfork
1180   #1\XINT_xtrunc_sp_IAA
1181   -\XINT_xtrunc_sp_IAB
1182   \krof #1%
1183 }%

1184 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1185 {%
1186   \romannumeral0\XINT_split_fromleft
1187   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1188 }%
```


7 Package *xintfrac* implementation

```
1189 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1190 {%
1191   0.\romannumeral\XINT_rep#1\endcsname0#2%
1192 }%

1193 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1194 {%
1195   \expandafter\XINT_xtrunc_sp_IB_c
1196   \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1197 }%

1198 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1199 {%
1200   \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1201 }%

1202 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1203 {%
1204   #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1205 }%
```

7.33 `\xintDigits`

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

```
1206 \mathchardef\XINTdigits 16
1207 \def\xintDigits #1#2%
1208   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=%}%
1209 \def\xinttheDigits {\number\XINTdigits }%
```

7.34 `\xintAdd`

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

```
1210 \def\xintAdd {\romannumeral0\xintadd }%
1211 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintra #1}%
1212 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1213 \def\XINT_fadd_Azero #1{\xintra }%
1214 \def\XINT_fadd_a #1/#2[#3]#4%
1215   {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1216 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1217 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1218 \def\XINT_fadd_c #1/#2[#3]#4%
1219 {%
1220   \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
```

7 Package *xintfrac* implementation

```

1221 }%
1222 \def\XINT_fadd_Aa #1%
1223 {%
1224   \xint_UDzerominusfork
1225     #1-\XINT_fadd_B
1226     0#1\XINT_fadd_Bb
1227     0-\XINT_fadd_Ba
1228   \krof #1%
1229 }%
1230 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1231 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1232 {%
1233   \expandafter\XINT_fadd_C\expandafter
1234     {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1235     {#7}{#5}{#4}[#2]%
1236 }%
1237 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1238 {%
1239   \expandafter\XINT_fadd_C\expandafter
1240     {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1241     {#5}{#7}{#6}[#3]%
1242 }%
1243 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine1?
1244 \def\XINT_fadd_C #1#2#3%
1245 {%
1246   \expandafter\XINT_fadd_D_b
1247   \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1248 }%

```

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.

```

1249 \def\XINT_fadd_D_a #1#2%
1250 {%
1251   \expandafter\XINT_fadd_D_b
1252   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1253 }%
1254 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1255 \def\XINT_fadd_D_c #1#2\Z
1256 {%
1257   \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1258 }%
1259 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1260 {%
1261   \expandafter\XINT_fadd_E
1262   \romannumeral0\xintiiquo {#3}{#2}.#2}%
1263 }%
1264 \def\XINT_fadd_E #1.#2#3%
1265 {%
1266   \expandafter\XINT_fadd_F
1267   \romannumeral0\xintiimul{#1}{#3}.\xintiiquo{#3}{#2}{#1}%
1268 }%
1269 \def\XINT_fadd_F #1.#2#3#4#5%

```

```

1270 {%
1271   \expandafter\XINT_fadd_G
1272   \romannumeral0\xintiiadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
1273 }%
1274 \def\XINT_fadd_G #1{%
1275 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1276 }\XINT_fadd_G{ }%

```

7.35 `\xintSub`

Since 1.3 will use least common multiple of denominators.

```

1277 \def\xintSub {\romannumeral0\xintsub }%
1278 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra #1}%
1279 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1280 \def\XINT_fsub_Azero #1{\xintopp }%
1281 \def\XINT_fsub_a #1/#2[#3]#4%
1282   {\expandafter\XINT_fsub_b\romannumeral0\xintra #4}{#3}{#1}{#2}}%
1283 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1284   #1-\XINT_fadd_Bzero
1285   0#1\XINT_fadd_c
1286   0-\XINT_fadd_c -#1}%
1287 \krof }%

```

7.36 `\xintSum`

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

Empty items are not accepted by this macro.

```

1288 \def\xintSum {\romannumeral0\xintsum }%
1289 \def\xintsum #1{\expandafter\XINT_fsumexpr\romannumeral`&&#1\xint:}%
1290 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1291 \def\XINT_fsum_loop_a #1#2%
1292 {%
1293   \expandafter\XINT_fsum_loop_b \romannumeral`&&#2\xint:#{#1}%
1294 }%
1295 \def\XINT_fsum_loop_b #1%
1296 {%
1297   \xint_gob_til_xint: #1\XINT_fsum_finished\xint:\XINT_fsum_loop_c #1%
1298 }%
1299 \def\XINT_fsum_loop_c #1\xint:#2%
1300 {%
1301   \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1302 }%
1303 \def\XINT_fsum_finished #1\xint:\xint:#2{ #2}%

```

7.37 `\xintMul`

```

1304 \def\xintMul {\romannumeral0\xintmul }%
1305 \def\xintmul #1{\expandafter\XINT_fmula\romannumeral0\xintra #1}.%
1306 \def\XINT_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%
1307 \def\XINT_fmula_a #1[#2].#3%

```

7 Package *xintfrac* implementation

```
1308   {\expandafter\XINT_fmulo\romannumeral0\xintra #3}{#1[#2.]}%
1309 \def\XINT_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1310 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]%
1311 {%
1312   \expandafter\XINT_fmulo_d
1313   \expandafter{\the\numexpr #3+#6\expandafter}%
1314   \expandafter{\romannumeral0\xintiimulo {#5}{#2}}%
1315   {\romannumeral0\xintiimulo {#4}{#1}}%
1316 }%
1317 \def\XINT_fmulo_d #1#2#3%
1318 {%
1319   \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1320 }%
1321 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1322 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%
```

7.38 `\xintSqr`

1.1 modifs comme `xintMul`.

```
1323 \def\xintSqr {\romannumeral0\xintsqr }%
1324 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintra #1}}%
1325 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1326 \def\XINT_fsqr_a #1/#2[#3]%
1327 {%
1328   \expandafter\XINT_fsqr_b
1329   \expandafter{\the\numexpr #3+#3\expandafter}%
1330   \expandafter{\romannumeral0\xintiisqr {#2}}%
1331   {\romannumeral0\xintiisqr {#1}}%
1332 }%
1333 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1334 \def\XINT_fsqr_zero #1{ 0/1[0]}%
```

7.39 `\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.

```
1335 \def\xintipow #1#2%
1336 {%
1337   \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1338   .\romannumeral0\xintnum{#1}\xint:
1339 }%
1340 \def\xintPow {\romannumeral0\xintpow }%
1341 \def\xintpow #1%
1342 {%
1343   \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1344 }%
1345 \def\XINT_fpow #1#2%
1346 {%
1347   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
```

```

1348 }%
1349 \def\XINT_fpow_fork #1#2\Z
1350 {%
1351   \xint_UDzerominusfork
1352   #1-\XINT_fpow_zero
1353   0#1\XINT_fpow_neg
1354   0-\XINT_fpow_pos #1}%
1355   \krof
1356   {#2}%
1357 }%
1358 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1359 \def\XINT_fpow_pos #1#2#3#4#5%
1360 {%
1361   \expandafter\XINT_fpow_pos_A\expandafter
1362   {\the\numexpr #1#2*#3\expandafter}\expandafter
1363   {\romannumeral0\xintiipow {#5}{#1#2}}%
1364   {\romannumeral0\xintiipow {#4}{#1#2}}%
1365 }%
1366 \def\XINT_fpow_neg #1#2#3#4%
1367 {%
1368   \expandafter\XINT_fpow_pos_A\expandafter
1369   {\the\numexpr -#1*#2\expandafter}\expandafter
1370   {\romannumeral0\xintiipow {#3}{#1}}%
1371   {\romannumeral0\xintiipow {#4}{#1}}%
1372 }%
1373 \def\XINT_fpow_pos_A #1#2#3%
1374 {%
1375   \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1376 }%
1377 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

7.40 `\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`.

```

1378 \def\xintFac {\romannumeral0\xintfac}%
1379 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

7.41 `\xintBinomial`

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

```

1380 \def\xintBinomial {\romannumeral0\xintbinomial}%
1381 \def\xintbinomial #1#2%
1382 {%
1383   \expandafter\XINT_binom_pre
1384   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]}%
1385 }%

```

7.42 `\xintPFactorial`

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

```

1386 \def\xintpfactorial #1#2%
1387 {%
1388   \expandafter\XINT_pfac_fork
1389   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1390 }%
1391 \def\xintPFactorial {\romannumeral0\xintpfactorial}%
1392 \def\xintpfactorial #1#2%
1393 {%
1394   \expandafter\XINT_pfac_fork
1395   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1396 }%

```

7.43 `\xintPrd`

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

```

1397 \def\xintPrd {\romannumeral0\xintprd }%
1398 \def\xintprd #1{\expandafter\XINT_fprdexpr \romannumeral`&&@#1\xint:}%
1399 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1400 \def\XINT_fprod_loop_a #1#2%
1401 {%
1402   \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\xint:{#1}%
1403 }%
1404 \def\XINT_fprod_loop_b #1%
1405 {%
1406   \xint_gob_til_xint: #1\XINT_fprod_finished\xint:\XINT_fprod_loop_c #1%
1407 }%
1408 \def\XINT_fprod_loop_c #1\xint:#2%
1409 {%
1410   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1411 }%
1412 \def\XINT_fprod_finished#1\xint:\xint:#2{ #2}%

```

7.44 `\xintDiv`

```

1413 \def\xintDiv {\romannumeral0\xintdiv }%
1414 \def\xintdiv #1%
1415 {%
1416   \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1417 }%
1418 \def\XINT_fdiv #1#2%
1419   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1420 \def\XINT_fdiv_A #1#2#3#4#5#6%
1421 {%
1422   \expandafter\XINT_fdiv_B
1423   \expandafter{\the\numexpr #4-#1\expandafter}%
1424   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1425   {\romannumeral0\xintiimul {#3}{#5}}%

```

```

1426 }%
1427 \def\XINT_fdiv_B #1#2#3%
1428 {%
1429   \expandafter\XINT_fdiv_C
1430   \expandafter{#3}{#1}{#2}%
1431 }%
1432 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

7.45 `\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`.

```

1433 \def\xintDivFloor   {\romannumeral0\xintdivfloor }%
1434 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%

```

7.46 `\xintDivTrunc`

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

```

1435 \def\xintDivTrunc   {\romannumeral0\xintdivtrunc }%
1436 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

7.47 `\xintDivRound`

1.1

```

1437 \def\xintDivRound   {\romannumeral0\xintdivround }%
1438 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

7.48 `\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).

```

1439 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1440 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1441 \def\XINT_modtrunc_a #1#2.#3%
1442   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1443 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1444 {%
1445   \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1446   \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1447   \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1448   \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1449 }%
1450 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1451 {%
1452   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}{0/1[0]}%
1453 }%
1454 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%

```

```

1455 \def\XINT_modtrunc_bneg #1%
1456 {%
1457   \xint_UDsignfork
1458     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1459     -{\XINT_modtrunc_pos #1}%
1460   \krof
1461 }%
1462 \def\XINT_modtrunc_bpos #1%
1463 {%
1464   \xint_UDsignfork
1465     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1466     -{\XINT_modtrunc_pos #1}%
1467   \krof
1468 }%

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).

1469 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1470 {%
1471   \expandafter\XINT_modtrunc_pos_a
1472   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1473   \romannumeral0\expandafter\XINT_mod_D_b
1474   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1475   {#1#5}{#7-#4}{#2}{#4-#7}%
1476 }%
1477 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

7.49 \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.

```

1478 \def\xintDivMod {\romannumeral0\xintdivmod }%
1479 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1480 \def\XINT_divmod_a #1#2.#3%
1481   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1482 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1483 {%
1484   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1485   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1486   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1487   \xint_orthat{\XINT_divmod_bpos #1#2}%
1488 }%
1489 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1490 {%
1491   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
1492   {\0}{0/1[0]}% à revoir...
1493 }%
1494 \def\XINT_divmod_aiszero #1.{\0}{0/1[0]}%
1495 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)

```



```

1496 {%
1497   \expandafter\XINT_divmod_bneg_finish
1498   \romannumeral0\xint_UDsignfork
1499     #1{\XINT_divmod_bpos {}}%
1500     -{\XINT_divmod_bpos {-#1}}%
1501   \krof
1502 }%
1503 \def\XINT_divmod_bneg_finish#1#2%
1504 {%
1505   \expandafter\xint_exchangetwo_keepbraces\expandafter
1506   {\romannumeral0\xintiioopp#2}{#1}%
1507 }%
1508 \def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1509 {%
1510   \expandafter\XINT_divmod_bpos_a
1511   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1512   \romannumeral0\expandafter\XINT_mod_D_b
1513   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1514   {#1#5}{#7-#4}{#2}{#4-#7}%
1515 }%
1516 \def\XINT_divmod_bpos_a #1.#2#3#4%
1517 {%
1518   \expandafter\XINT_divmod_bpos_finish
1519   \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
1520 }%
1521 \def\XINT_divmod_bpos_finish #1#2#3{#1}{#2#3}%

```

7.50 `\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.

```

1522 \def\xintMod {\romannumeral0\xintmod }%
1523 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1524 \def\XINT_mod_a #1#2.#3%
1525   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1526 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1527 {%
1528   \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1529   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1530   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1531   \xint_orthat{\XINT_mod_bpos #1#2}%
1532 }%

```

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

```

1533 \let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero
1534 \let\XINT_mod_aiszero \XINT_modtrunc_aiszero
1535 \def\XINT_mod_bneg #1% f % -g = -((-f) % g), for g > 0
1536 {%
1537   \xintiioopp\xint_UDsignfork

```

7 Package *xintfrac* implementation

```

1538     #1{\XINT_mod_bpos {}}%
1539     -{\XINT_mod_bpos {-#1}}%
1540     \krof
1541 }%
1542 \def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1543 {%
1544     \expandafter\XINT_mod_bpos_a
1545     \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1546     \romannumeral0\expandafter\XINT_mod_D_b
1547     \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1548     {#1#5}{#7-#4}{#2}{#4-#7}%
1549 }%
1550 \def\XINT_mod_D_a #1#2%
1551 {%
1552     \expandafter\XINT_mod_D_b
1553     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1554 }%
1555 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1556 \def\XINT_mod_D_c #1#2\Z
1557 {%
1558     \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1559 }%
1560 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1561 {%
1562     \expandafter\XINT_mod_E
1563     \romannumeral0\xintiico {#3}{#2}.#2}%
1564 }%
1565 \def\XINT_mod_E #1.#2#3%
1566 {%
1567     \expandafter\XINT_mod_F
1568     \romannumeral0\xintiimul{#1}{#3}.\xintiico{#3}{#2}{#1}%
1569 }%
1570 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1571 {%
1572     {#1}{\xintie{\xintimul{#4}{#3}}{#5}}%
1573     {\xintie{\xintimul{#6}{#2}}{#7}}%
1574 }%
1575 \def\XINT_mod_bpos_a #1.#2#3#4{\xintirem {#3}{#4}/#2[#1]}%

```

7.51 *\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.

```

1576 \def\xintIsOne {\romannumeral0\xintisone }%
1577 \def\xintisone #1{\expandafter\XINT_fracisone
1578     \romannumeral0\xintraewithzeros{#1}\Z }%
1579 \def\XINT_fracisone #1/#2\Z
1580     {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

7.52 *\xintGeq*

```

1581 \def\xintGeq {\romannumeral0\xintgeq }%

```

7 Package *xintfrac* implementation

```

1582 \def\xintgeq #1%
1583 {%
1584   \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1585 }%
1586 \def\XINT_fgeq #1#2%
1587 {%
1588   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1589 }%
1590 \def\XINT_fgeq_A #1%
1591 {%
1592   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1593   \XINT_fgeq_B #1%
1594 }%
1595 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1596 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1597 {%
1598   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1599   \expandafter\XINT_fgeq_C\expandafter
1600   {\the\numexpr #7-#3\expandafter}\expandafter
1601   {\romannumeral0\xintiimul {#4#5}{#2}}%
1602   {\romannumeral0\xintiimul {#6}{#1}}%
1603 }%
1604 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1605 \def\XINT_fgeq_C #1#2#3%
1606 {%
1607   \expandafter\XINT_fgeq_D\expandafter
1608   {#3}{#1}{#2}%
1609 }%
1610 \def\XINT_fgeq_D #1#2#3%
1611 {%
1612   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1613   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1614   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1615 }%
1616 \def\XINT_fgeq_E #1%
1617 {%
1618   \xint_UDsignfork
1619   #1\XINT_fgeq_Fd
1620   -{\XINT_fgeq_Fn #1}%
1621   \krof
1622 }%

1623 \def\XINT_fgeq_Fd #1\Z #2#3%
1624 {%
1625   \expandafter\XINT_fgeq_Fe
1626   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1627 }%
1628 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1629 \def\XINT_fgeq_Fn #1\Z #2#3%
1630 {%
1631   \expandafter\XINT_fgeq_Fo
1632   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:

```

```
1633 }%
1634 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%
```

7.53 `\xintMax`

```
1635 \def\xintMax {\romannumeral0\xintmax }%
1636 \def\xintmax #1%
1637 {%
1638   \expandafter\XINT_fmax\expandafter {\romannumeral0\xintra #1}}%
1639 }%
1640 \def\XINT_fmax #1#2%
1641 {%
1642   \expandafter\XINT_fmax_A\romannumeral0\xintra #2}#1%
1643 }%
1644 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1645 {%
1646   \xint_UDsignsfork
1647     #1#5\XINT_fmax_minusminus
1648     -#5\XINT_fmax_firstneg
1649     #1-\XINT_fmax_secondneg
1650     --\XINT_fmax_nonneg_a
1651   \krof
1652   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1653 }%
1654 \def\XINT_fmax_minusminus --%
1655   {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1656 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1657 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1658 \def\XINT_fmax_nonneg_a #1#2#3#4%
1659 {%
1660   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1661 }%
1662 \def\XINT_fmax_nonneg_b #1#2%
1663 {%
1664   \if0\romannumeral0\XINT_fgeq_A #1#2%
1665     \xint_afterfi{ #1}%
1666   \else \xint_afterfi{ #2}%
1667   \fi
1668 }%
```

7.54 `\xintMaxof`

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

```
1669 \def\xintMaxof {\romannumeral0\xintmaxof }%
1670 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\xint:}%
1671 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}!}%
1672 \def\XINT_maxof_b #1!#2%
1673   {\expandafter\XINT_maxof_c\romannumeral`&&@#2!{#1}!}%
1674 \def\XINT_maxof_c #1%
1675   {\xint_gob_til_xint: #1\XINT_maxof_e\xint:\XINT_maxof_d #1}%
1676 \def\XINT_maxof_d #1!%
1677   {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1678 \def\XINT_maxof_e #1!#2!{ #2}%
```

7.55 `\xintMin`

```

1679 \def\xintMin {\romannumeral0\xintmin }%
1680 \def\xintmin #1%
1681 {%
1682   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra #1}}%
1683 }%
1684 \def\XINT_fmin #1#2%
1685 {%
1686   \expandafter\XINT_fmin_A\romannumeral0\xintra #2}#1%
1687 }%
1688 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1689 {%
1690   \xint_UDsignsfork
1691     #1#5\XINT_fmin_minusminus
1692     -#5\XINT_fmin_firstneg
1693     #1-\XINT_fmin_secondneg
1694     --\XINT_fmin_nonneg_a
1695   \krof
1696   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1697 }%
1698 \def\XINT_fmin_minusminus --%
1699   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1700 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1701 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1702 \def\XINT_fmin_nonneg_a #1#2#3#4%
1703 {%
1704   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1705 }%
1706 \def\XINT_fmin_nonneg_b #1#2%
1707 {%
1708   \if0\romannumeral0\XINT_fgeq_A #1#2%
1709     \xint_afterfi{ #2}%
1710   \else \xint_afterfi{ #1}%
1711   \fi
1712 }%

```

7.56 `\xintMinof`

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

```

1713 \def\xintMinof {\romannumeral0\xintminof }%
1714 \def\xintminof #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\xint:}%
1715 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}!}%
1716 \def\XINT_minof_b #1!#2%
1717   {\expandafter\XINT_minof_c\romannumeral`&&@#2!{#1}!}%
1718 \def\XINT_minof_c #1%
1719   {\xint_gob_til_xint: #1\XINT_minof_e\xint:\XINT_minof_d #1}%
1720 \def\XINT_minof_d #1!%
1721   {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1722 \def\XINT_minof_e #1!#2!{ #2}%

```

7.57 `\xintCmp`

```

1723 \def\xintCmp {\romannumeral0\xintcmp }%
1724 \def\xintcmp #1%
1725 {%
1726   \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintra #1}}%
1727 }%
1728 \def\XINT_fcmp #1#2%
1729 {%
1730   \expandafter\XINT_fcmp_A\romannumeral0\xintra #2}#1%
1731 }%
1732 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1733 {%
1734   \xint_UDsignsfork
1735     #1#5\XINT_fcmp_minusminus
1736     -#5\XINT_fcmp_firstneg
1737     #1-\XINT_fcmp_secondneg
1738     --\XINT_fcmp_nonneg_a
1739   \krof
1740   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1741 }%
1742 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1743 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1744 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1745 \def\XINT_fcmp_nonneg_a #1#2%
1746 {%
1747   \xint_UDzerosfork
1748     #1#2\XINT_fcmp_zerozero
1749     0#2\XINT_fcmp_firstzero
1750     #10\XINT_fcmp_secondzero
1751     00\XINT_fcmp_pos
1752   \krof
1753   #1#2%
1754 }%
1755 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1756 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1757 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1758 \def\XINT_fcmp_pos #1#2#3#4%
1759 {%
1760   \XINT_fcmp_B #1#3#2#4%
1761 }%
1762 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1763 {%
1764   \expandafter\XINT_fcmp_C\expandafter
1765   {\the\numexpr #6-#3\expandafter}\expandafter
1766   {\romannumeral0\xintiimul {#4}{#2}}%
1767   {\romannumeral0\xintiimul {#5}{#1}}%
1768 }%
1769 \def\XINT_fcmp_C #1#2#3%
1770 {%
1771   \expandafter\XINT_fcmp_D\expandafter
1772   {#3}{#1}{#2}%
1773 }%

```

```

1774 \def\XINT_fcmp_D #1#2#3%
1775 {%
1776   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1777   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1778   { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1779 }%
1780 \def\XINT_fcmp_E #1%
1781 {%
1782   \xint_UDsignfork
1783   #1\XINT_fcmp_Fd
1784   -{\XINT_fcmp_Fn #1}%
1785   \krof
1786 }%

1787 \def\XINT_fcmp_Fd #1\Z #2#3%
1788 {%
1789   \expandafter\XINT_fcmp_Fe
1790   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1791 }%
1792 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1793 \def\XINT_fcmp_Fn #1\Z #2#3%
1794 {%
1795   \expandafter\XINT_fcmp_Fo
1796   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1797 }%
1798 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

7.58 `\xintAbs`

```

1799 \def\xintAbs {\romannumeral0\xintabs }%
1800 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

7.59 `\xintOpp`

```

1801 \def\xintOpp {\romannumeral0\xintopp }%
1802 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

7.60 `\xintSgn`

```

1803 \def\xintSgn {\romannumeral0\xintsgn }%
1804 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\xint:}%

```

7.61 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.

7.62 `\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

7 Package *xintfrac* implementation

that the output was a power of ten but not detected to be a rounding up of the original fraction. The value was ok, but printed 1.0...0eN with P-1 zeroes, not 10.0...0e(N-1).

I decided it was not worth the effort to enhance the algorithm to detect with 100% fiability all cases of rounding up to next power of ten, hence 1.2k dropped this.

To avoid duplication of code, and any extra burden on `\XINTinFloat`, which is the macro used internally by the float macros for parsing their inputs, we simply make now `\xintFloat` a wrapper of `\XINTinFloat`.

```
1805 \def\xintFloat    {\romannumeral0\xintfloat }%
1806 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
1807 \def\XINT_float_chkopt #1%
1808 {%
1809   \ifx [#1\expandafter\XINT_float_opt
1810     \else\expandafter\XINT_float_noopt
1811   \fi #1%
1812 }%
1813 \def\XINT_float_noopt #1\xint:%
1814 {%
1815   \expandafter\XINT_float_post
1816   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
1817 }%
1818 \def\XINT_float_opt [\xint:#1]%
1819 {%
1820   \expandafter\XINT_float_opt_a\the\numexpr #1.%
1821 }%
1822 \def\XINT_float_opt_a #1.#2%
1823 {%
1824   \expandafter\XINT_float_post
1825   \romannumeral0\XINTinfloat[#1]{#2}#1.%
1826 }%
1827 \def\XINT_float_post #1%
1828 {%
1829   \xint_UDzerominusfork
1830   #1-\XINT_float_zero
1831   0#1\XINT_float_neg
1832   0-\XINT_float_pos
1833   \krof #1%
1834 }%
1835 \def\XINT_float_zero #1]#2.{ 0.e0}%
1836 \def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
1837 \def\XINT_float_pos #1#2[#3]#4.%
1838 {%
1839   \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
1840 }%
1841 \def\XINT_float_pos_done #1.#2;{ #2e#1}%
```

7.63 `\XINTinFloat`, `\XINTinFloatS`

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

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

7 Package *xintfrac* implementation

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`.

```
1842 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1843 \def\XINTinfloat
1844   {\expandafter\XINT_infloat_clean\romannumeral0\XINT_infloat}%
1845 \def\XINT_infloat_clean #1%
1846   {\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 $\text{abs}(\#2)$, ici le $\#2$ peut avoir un signe) qui est $< P$

```
1847 \def\XINT_infloat_clean_a !#1.#2[#3]%
1848 {%
1849   \expandafter\XINT_infloat_done
1850   \the\numexpr #3-#1\expandafter.%
1851   \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
1852 }%
1853 \def\XINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
1854 \def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
1855 \def\XINTinfloatS
1856   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
1857 \def\XINT_infloatS_clean #1%
1858   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
1859 \def\XINT_infloatS_clean_a !#1.{ }%
```

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

```
1860 \def\XINT_infloat [#1]#2%
1861 {%
1862   \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
1863   \romannumeral0\XINT_infrac {#2}%
1864 }%
```

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

```
1865 \def\XINT_infloat_a #1.#2#3#4%
1866 {%
```

7 Package *xintfrac* implementation

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

```
1867 \if1\XINT_is_One#4XY%
1868 \expandafter\XINT_infloat_sp
1869 \else\expandafter\XINT_infloat_fork
1870 \fi #3.{#1}{#2}{#4}%
1871 }%
```

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.

```
1872 \def\XINT_infloat_sp #1%
1873 {%
1874 \xint_UDzerominusfork
1875 #1-\XINT_infloat_spzero
1876 0#1\XINT_infloat_spneg
1877 0-\XINT_infloat_sppos
1878 \krof #1%
1879 }%
```

Attention surtout pas 0/1[0] ici.

```
1880 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
1881 \def\XINT_infloat_spneg-%
1882 {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
1883 \def\XINT_infloat_spnegend #1%
1884 {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
1885 \def\XINT_infloat_spneg_needzeros -!#1.#!#1.-}%
```

in: A. $\{P\}\{N\}\{1\}$
out: P-L.A.P.N.

```
1886 \def\XINT_infloat_sppos #1.#2#3#4%
1887 {%
1888 \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1889 }%
```

#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.

```
1890 \def\XINT_infloat_sp_b #1%
1891 {%
1892 \xint_UDzerominusfork
1893 #1-\XINT_infloat_sp_quick
1894 0#1\XINT_infloat_sp_c
1895 0-\XINT_infloat_sp_needzeros
1896 \krof #1%
1897 }%
```

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

```
1898 \def\XINT_infloat_sp_quick 0.#1.#2.#3.{ #1[#3]}%
```

7 Package *xintfrac* implementation

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 zeros.

```
1899 \def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

L-P=#1.A=#2#3.P=#4.N=#5.

Ici P<L. Il va falloir arrondir. Attention si on va à la puissance de 10 suivante. En #1 on a L-P qui est >0. L'exposant final sera N+L-P, sauf dans le cas spécial, il sera alors N+L-P+1. L'ajustement final est fait par `\XINT_infloat_Y`.

```
1900 \def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
```

```
1901 {%
```

```
1902   \expandafter\XINT_infloat_Y
```

```
1903   \the\numexpr #5+#1\expandafter.%
```

```
1904   \romannumeral0\expandafter\XINT_infloat_sp_round
```

```
1905   \romannumeral0\XINT_split_fromleft
```

```
1906   (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
```

```
1907 }%
```

```
1908 \def\XINT_infloat_sp_round #1.#2.%
```

```
1909 {%
```

```
1910   \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
```

```
1911 }%
```

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.

```
1912 \def\XINT_infloat_fork #1%
```

```
1913 {%
```

```
1914   \xint_UDsignfork
```

```
1915   #1\XINT_infloat_J
```

```
1916   -\XINT_infloat_K
```

```
1917   \krof #1%
```

```
1918 }%
```

```
1919 \def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%
```

A. $\{P\}_n\{B\}$ avec B>1.

```
1920 \def\XINT_infloat_K #1.#2%
```

```
1921 {%
```

```
1922   \expandafter\XINT_infloat_L
```

```
1923   \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
```

```
1924 }%
```

|A|.P+4. $\{A\}_P\{n\}_B$. We check if A already has length <= P+4.

```
1925 \def\XINT_infloat_L #1.#2.%
```

```
1926 {%
```

```
1927   \ifnum #1>#2
```

```
1928     \expandafter\XINT_infloat_Ma
```

```
1929   \else
```

```
1930     \expandafter\XINT_infloat_Mb
```

```
1931   \fi #1.#2.%
```

```
1932 }%
```

7 Package *xintfrac* implementation

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

```
1933 \def\XINT_infloat_Ma #1.#2.#3%
1934 {%
1935   \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
1936   \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1937   #2.#1.#3}%
1938 }%
```

$|A|.P+4.\{A\}\{P\}\{n\}\{B\}$.
 Here A is short. We set $u = P+4 - |A|$, and $A' = A$ ($A' = 10^u A$)
 output: $u.A'..P+4.|A|. \{A\}\{P\}\{n\}\{B\}$

```
1939 \def\XINT_infloat_Mb #1.#2.#3%
1940 {%
1941   \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
1942   #3..#2.#1.#3}%
1943 }%
```

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

```
1944 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
1945 {%
1946   \expandafter\XINT_infloat_N
1947   \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
1948 }%
1949 \def\XINT_infloat_N #1.#2.%
1950 {%
1951   \ifnum #1>#2
1952     \expandafter\XINT_infloat_Oa
1953   \else
1954     \expandafter\XINT_infloat_Ob
1955   \fi #1.#2.%
1956 }%
```

input $|B|.P+4.\{B\}u.A'..P.|A|.n.\{A\}\{B\}$
 output $v=-0.B'..junk.|B|.u.A'..P.|A|.n.\{A\}\{B\}$

```
1957 \def\XINT_infloat_Oa #1.#2.#3%
1958 {%
1959   \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
1960   \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1961   #1.%
1962 }%
```

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

```
1963 \def\XINT_infloat_Ob #1.#2.#3%
1964 {%
1965   \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
1966 }%
```

7 Package *xintfrac* implementation

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.

```
1967 \def\xINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
1968 {%
1969   \cname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
1970   \romannumeral0\xintiiquo
1971   {\romannumeral0\xINT_dsx_addzerosnofuss
1972     {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
1973   {#3}.#9.#5.%
1974 }%
```

«quick» branch.

```
1975 \def\xINT_infloat_Qq #1.#2.%
1976 {%
1977   \expandafter\xINT_infloat_Rq
1978   \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1979 }%
1980 \def\xINT_infloat_Rq #1.#2#3.%
1981 {%
1982   \ifnum#2<\xint_c_v
1983     \expandafter\xINT_infloat_SEq
1984   \else\expandafter\xINT_infloat_SUP
1985   \fi
1986   {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
1987 }%
```

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

```
1988 \def\xINT_infloat_Q #1.#2.%
1989 {%
1990   \expandafter\xINT_infloat_R
1991   \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1992 }%
1993 \def\xINT_infloat_R #1.#2#3#4#5.%
1994 {%
1995   \if.#5.\expandafter\xINT_infloat_Sa\else\expandafter\xINT_infloat_Sb\fi
1996   #2#3#4#5.#1.%
1997 }%
```

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

```
1998 \def\xINT_infloat_Sa #1.%
1999 {%
2000   \ifnum#1>500 \xint_dothis\xINT_infloat_SUP\fi
2001   \ifnum#1<499 \xint_dothis\xINT_infloat_SEq\fi
2002   \xint_orthat\xINT_infloat_X\xint_c_
2003 }%
```

7 Package *xintfrac* implementation

```

2004 \def\XINT_infloat_Sb #1.%
2005 {%
2006   \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
2007   \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2008   \xint_orthat\XINT_infloat_X\xint_c_i
2009 }%

```

```

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

```

```

2010 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2011 {%
2012   \expandafter\XINT_infloat_SY
2013   \the\numexpr #6+#5-#4-#3+#1.#2.%
2014 }%
2015 \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.

```

2016 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2017 {%
2018   \expandafter\XINT_infloat_Y
2019   \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2020   \romannumeral0\xintinc{#2#3}.#2%
2021 }%

```

```

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

```

```

2022 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2023 {%
2024   \expandafter\XINT_infloat_Y
2025   \the\numexpr #7+#6-#5-#4+#1\expandafter.%
2026   \romannumeral`&&\romannumeral0\xintiiiflt
2027   {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
2028   {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
2029   \xint_firstofone
2030   \xintinc{#2#3}.#2%
2031 }%

```

check for rounding up to next power of ten.

```

2032 \def\XINT_infloat_Y #1{%
2033 \def\XINT_infloat_Y ##1.##2##3.##4%
2034 {%
2035   \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
2036   #1##2##3[##1]%
2037 }}\XINT_infloat_Y{ }%

```

#1=1, #2=0.

```
2038 \def\XINT_infloat_Z #1#2#3[#4]%
2039 {%
2040   \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2041 }%
2042 \def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%
```

7.64 \xintPFloat

1.1. This is a prettifying printing macro for floats.

The macro applies one simple rule: $x.yz\dots eN$ will drop scientific notation in favor of pure decimal notation if $-5 \leq N \leq 5$. 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).

```
2043 \def\xintPFloat {\romannumeral0\xintpfloat }%
2044 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2045 \def\XINT_pfloat_chkopt #1%
2046 {%
2047   \ifx [#1\expandafter\XINT_pfloat_opt
2048     \else\expandafter\XINT_pfloat_noopt
2049   \fi #1%
2050 }%
2051 \def\XINT_pfloat_noopt #1\xint:%
2052 {%
2053   \expandafter\XINT_pfloat_a
2054   \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2055 }%

2056 \def\XINT_pfloat_opt [\xint:#1]%
2057 {%
2058   \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
2059 }%
2060 \def\XINT_pfloat_opt_a #1.#2%
2061 {%
2062   \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2063 }%
2064 \def\XINT_pfloat_a #1%
2065 {%
2066   \xint_UDzerominusfork
2067     #1-\XINT_pfloat_zero
2068     0#1\XINT_pfloat_neg
2069     0-\XINT_pfloat_pos
2070   \krof #1%
2071 }%

2072 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2073 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%
```



```

2074 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2075 {%
2076   \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2077   \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2078   \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2079   \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2080   \xint_orthat\XINT_pfloat_P #1#2e#3;%
2081 }%
2082 \def\XINT_pfloat_no #1#2;{ #1.#2}%

```

This is all simpler coded, now that 1.2k's `\xintFloat` always outputs a mantissa with exactly one digit before decimal mark always.

```

2083 \def\XINT_pfloat_N #1e-#2;%
2084 {%
2085   \csname XINT_pfloat_N_\romannumeral#2\endcsname #1%
2086 }%
2087 \def\XINT_pfloat_N_i { 0.}%
2088 \def\XINT_pfloat_N_ii { 0.0}%
2089 \def\XINT_pfloat_N_iii{ 0.00}%
2090 \def\XINT_pfloat_N_iv { 0.000}%
2091 \def\XINT_pfloat_N_v { 0.0000}%

2092 \def\XINT_pfloat_P #1e#2;%
2093 {%
2094   \csname XINT_pfloat_P_\romannumeral#2\endcsname #1%
2095 }%
2096 \def\XINT_pfloat_P_ #1{ #1.}%
2097 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2098 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2099 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2100 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2101 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

2102 \def\XINT_pfloat_Ps #1e#2;%
2103 {%
2104   \csname XINT_pfloat_Ps_\romannumeral#2\endcsname #100000;%
2105 }%
2106 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.}%
2107 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.}%
2108 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.}%
2109 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.}%
2110 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.}%

```

7.65 `\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.

7 Package *xintfrac* implementation

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.

```
2111 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2112 \def\XINTinfloatfracdigits #1%
2113 {%
2114   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xintfrac{#1}}%
2115 }%
2116 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%
```

7.66 `\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.

```
2117 \def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
2118 \def\xintfloatadd      #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2119 \def\XINTinFloatAdd    {\romannumeral0\XINTinfloatadd }%
2120 \def\XINTinfloatadd    #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2121 \def\XINT_fladd_chkopt #1#2%
2122 {%
2123   \ifx [#2\expandafter\XINT_fladd_opt
2124     \else\expandafter\XINT_fladd_noopt
2125   \fi #1#2%
2126 }%
2127 \def\XINT_fladd_noopt #1#2\xint:#3%
2128 {%
2129   #1[\XINTdigits]%
2130   {\expandafter\XINT_FL_add_a
2131     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2132 }%
2133 \def\XINT_fladd_opt #1[\xint:#2]#3#4%
2134 {%
2135   \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2136 }%
2137 \def\XINT_fladd_opt_a #1.#2#3#4%
2138 {%
2139   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2140 }%
2141 \def\XINT_FL_add_a #1%
2142 {%
2143   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2144 }%
2145 \def\XINT_FL_add_zero #1.#2{#2}%[
```

7 Package *xintfrac* implementation

```

2146 \def\XINT_FL_add_b #1]#2.#3%
2147 {%
2148   \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1)%
2149 }%

```

```

2150 \def\XINT_FL_add_c #1%
2151 {%
2152   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2153 }%

```

```

2154 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2155 {%
2156   \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2157   \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2158   \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2159 }%

```

7.67 `\xintFloatSub`, `\XINTinFloatSub`

First done 1.07.

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

```

2160 \def\xintFloatSub      {\romannumeral0\xintfloatsub }%
2161 \def\xintfloatsub      #1{\XINT_fsub_chkopt \xintfloat #1\xint:}%
2162 \def\XINTinFloatSub    {\romannumeral0\XINTinfloatsub }%
2163 \def\XINTinfloatsub    #1{\XINT_fsub_chkopt \XINTinfloatS #1\xint:}%
2164 \def\XINT_fsub_chkopt #1#2%
2165 {%
2166   \ifx [#2\expandafter\XINT_fsub_opt
2167     \else\expandafter\XINT_fsub_noopt
2168   \fi #1#2%
2169 }%
2170 \def\XINT_fsub_noopt #1#2\xint:#3%
2171 {%
2172   #1[\XINTdigits]%
2173   {\expandafter\XINT_FL_add_a
2174     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2175 }%
2176 \def\XINT_fsub_opt #1[\xint:#2]#3#4%
2177 {%
2178   \expandafter\XINT_fsub_opt_a\the\numexpr #2.#1%
2179 }%
2180 \def\XINT_fsub_opt_a #1.#2#3#4%
2181 {%
2182   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2183 }%

```

7.68 `\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).

```

2184 \def\xintFloatMul    {\romannumeral0\xintfloatmul    }%
2185 \def\xintfloatmul    #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2186 \def\XINTinFloatMul  {\romannumeral0\XINTinfloatmul }%
2187 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2188 \def\XINT_flmul_chkopt #1#2%
2189 {%
2190   \ifx [#2\expandafter\XINT_flmul_opt
2191     \else\expandafter\XINT_flmul_noopt
2192   \fi #1#2%
2193 }%
2194 \def\XINT_flmul_noopt #1#2\xint:#3%
2195 {%
2196   #1[\XINTdigits]%
2197   {\expandafter\XINT_FL_mul_a
2198     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2199 }%
2200 \def\XINT_flmul_opt #1[\xint:#2]#3#4%
2201 {%
2202   \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2203 }%
2204 \def\XINT_flmul_opt_a #1.#2#3#4%
2205 {%
2206   #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.#4}}%
2207 }%
2208 \def\XINT_FL_mul_a #1[#2]#3.#4%
2209 {%
2210   \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2211 }%

2212 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

7.69 `\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.

7 Package *xintfrac* implementation

```
2213 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2214 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2215 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2216 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
2217 \def\XINT_fldiv_chkopt #1#2%
2218 {%
2219   \ifx [#2\expandafter\XINT_fldiv_opt
2220     \else\expandafter\XINT_fldiv_noopt
2221   \fi #1#2%
2222 }%

2223 \def\XINT_fldiv_noopt #1#2\xint:#3%
2224 {%
2225   #1[\XINTdigits]%
2226   {\expandafter\XINT_FL_div_a
2227     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2228 }%
2229 \def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
2230 {%
2231   \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2232 }%

2233 \def\XINT_fldiv_opt_a #1.#2#3#4%
2234 {%
2235   #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.{#3}}%
2236 }%
2237 \def\XINT_FL_div_a #1[#2]#3.#4%
2238 {%
2239   \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2240 }%

2241 \def\XINT_FL_div_b #1[#2]{#1e#2}%
```

7.70 `\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).

```
2242 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2243 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2244 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
2245 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2246 \def\XINT_flpow_chkopt #1#2%
```

7 Package *xintfrac* implementation

```

2247 {%
2248   \ifx [#2\expandafter\XINT_flpow_opt
2249     \else\expandafter\XINT_flpow_noopt
2250   \fi
2251   #1#2%
2252 }%
2253 \def\XINT_flpow_noopt #1#2\xint:#3%
2254 {%
2255   \expandafter\XINT_flpow_checkB_a
2256   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}%
2257 }%
2258 \def\XINT_flpow_opt #1[\xint:#2]%
2259 {%
2260   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2261 }%
2262 \def\XINT_flpow_opt_a #1.#2#3#4%
2263 {%
2264   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2265 }%
2266 \def\XINT_flpow_checkB_a #1%
2267 {%
2268   \xint_UDzerominusfork
2269   #1-\XINT_flpow_BisZero
2270   0#1{\XINT_flpow_checkB_b -}%
2271   0-{\XINT_flpow_checkB_b }{#1}%
2272   \krof
2273 }%
2274 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

2275 \def\XINT_flpow_checkB_b #1#2.#3.%
2276 {%
2277   \expandafter\XINT_flpow_checkB_c
2278   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2279 }%
2280 \def\XINT_flpow_checkB_c #1.#2.%
2281 {%
2282   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2283 }%

1.2f rounds input to P digits, first.

2284 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2285 {%
2286   \expandafter \XINT_flpow_aa
2287   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2288 }%

2289 \def\XINT_flpow_aa #1[#2]#3%
2290 {%

```

7 Package *xintfrac* implementation

```

2291 \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2292 \romannumeral\XINT_rep #3\endcsname0.#1.%
2293 }%

2294 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2295 \def\XINT_flpow_a #1%
2296 {%
2297 \xint_UDzerominusfork
2298 #1-\XINT_flpow_zero
2299 0#1{\XINT_flpow_b \iftrue}%
2300 0-{\XINT_flpow_b \iffalse#1}%
2301 \krof
2302 }%
2303 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2304 {%
2305 #6{\if 1#51\xint_dothis {0[0]}\fi
2306 \xint_orthat
2307 {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2308 }%
2309 }%

2310 \def\XINT_flpow_b #1#2[#3]#4#5%
2311 {%
2312 \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2313 }%

2314 \def\XINT_flpow_truncate #1.#2.#3.%
2315 {%
2316 \expandafter\XINT_flpow_truncate_a
2317 \romannumeral0\XINT_split_fromleft
2318 #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2319 }%

2320 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2321 \def\XINT_flpow_loopI #1.%
2322 {%
2323 \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2324 \ifodd #1
2325 \expandafter\XINT_flpow_loopI_odd
2326 \else
2327 \expandafter\XINT_flpow_loopI_even
2328 \fi
2329 #1.%
2330 }%

```

7 Package *xintfrac* implementation

```

2331 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2332 {%
2333   \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2334 }%

2335 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2336 {%
2337   \expandafter\XINT_flpow_loopI
2338   \the\numexpr #1/\xint_c_ii\expandafter.%
2339   \the\numexpr\expandafter\XINT_flpow_truncate
2340   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2341 }%
2342 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2343 {%
2344   \expandafter\XINT_flpow_loopII
2345   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2346   \the\numexpr\expandafter\XINT_flpow_truncate
2347   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2348 }%
2349 \def\XINT_flpow_loopII #1.%
2350 {%
2351   \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2352   \ifodd #1
2353     \expandafter\XINT_flpow_loopII_odd
2354   \else
2355     \expandafter\XINT_flpow_loopII_even
2356   \fi
2357   #1.%
2358 }%
2359 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2360 {%
2361   \expandafter\XINT_flpow_loopII
2362   \the\numexpr #1/\xint_c_ii\expandafter.%
2363   \the\numexpr\expandafter\XINT_flpow_truncate
2364   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2365 }%
2366 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2367 {%
2368   \expandafter\XINT_flpow_loopII_odda
2369   \the\numexpr\expandafter\XINT_flpow_truncate
2370   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2371   #1.#2.#3.%
2372 }%
2373 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2374 {%
2375   \expandafter\XINT_flpow_loopII
2376   \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2377   \the\numexpr\expandafter\XINT_flpow_truncate
2378   \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%

```


7 Package *xintfrac* implementation

```
2379 #1.#2.%
2380 }%
```

```
2381 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2382 {%
2383 \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2384 \the\numexpr\expandafter\XINT_flpow_truncate
2385 \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2386 }%
```

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 $0.52 \text{ ulp}(Z)$ (and worst cases can only be slightly worse than $0.51 \text{ ulp}(Z)$).

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 $0.5125 \text{ ulp}(Z)$, and at any rate it is less than $0.52 \text{ ulp}(Z)$.

```
2387 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2388 {%
2389 \expandafter\XINT_flpow_IIIend
2390 \xint_UDsignfork
2391 #5{{1/#3[-#2]}}%
2392 -{{#3[#2]}}%
2393 \krof #1%
2394 }%
```

```
2395 \def\XINT_flpow_IIIend #1#2#3%
2396 {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}%
```

7.71 `\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 \wedge 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 \wedge 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.

```
2397 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
```

7 Package *xintfrac* implementation

```

2398 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2399 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
2400 \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.

2401 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerh }%

2402 \def\XINTinfloatpowerh #1#2%
2403 {%
2404     \expandafter\XINT_flpowerh_a\romannumeral0\xintitrunc1{#2};%
2405     \XINTdigits.{#1}{\XINTinfloatS[\XINTdigits]}%
2406 }%

2407 \def\XINT_flpowerh_a #1;%
2408 {%
2409     \if0\xintLDg{#1}\expandafter\XINT_flpowerh_int
2410     \else\expandafter\XINT_flpowerh_b
2411     \fi #1.%
2412 }%
2413 \def\XINT_flpowerh_int #1%
2414 {%
2415     \if0#1\expandafter\XINT_flpower_BisZero
2416     \else\expandafter\XINT_flpowerh_i
2417     \fi #1%
2418 }%
2419 \def\XINT_flpowerh_i #10.{\expandafter\XINT_flpower_checkB_a#1.}%
2420 \def\XINT_flpowerh_b #1.%
2421 {%
2422     \expandafter\XINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2423 }%
2424 \def\XINT_flpowerh_c #1.%
2425 {%
2426     \ifodd\xintLDg{#1} %<- intentional space
2427     \expandafter\XINT_flpowerh_d\else\expandafter\XINT_flpowerh_e
2428     \fi #1.%
2429 }%
2430 \def\XINT_flpowerh_d #1.\XINTdigits.#2#3%
2431 {%
2432     \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2433 }%
2434 \def\XINT_flpowerh_finish #1%
2435     {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%

```

7 Package *xintfrac* implementation

```
2436 \def\XINT_flpowerh_e #1.%
2437   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xinthal{#1}.}%

  Start of macro. Check for optional argument.

2438 \def\XINT_flpower_chkopt #1#2%
2439 {%
2440   \ifx [#2\expandafter\XINT_flpower_opt
2441     \else\expandafter\XINT_flpower_noopt
2442     \fi
2443   #1#2%
2444 }%
2445 \def\XINT_flpower_noopt #1#2\xint:#3%
2446 {%
2447   \expandafter\XINT_flpower_checkB_a
2448   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2449 }%
2450 \def\XINT_flpower_opt #1[\xint:#2]%
2451 {%
2452   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2453 }%
2454 \def\XINT_flpower_opt_a #1.#2#3#4%
2455 {%
2456   \expandafter\XINT_flpower_checkB_a
2457   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2458 }%
2459 \def\XINT_flpower_checkB_a #1%
2460 {%
2461   \xint_UDzerominusfork
2462     #1-{\XINT_flpower_BisZero 0}%
2463     0#1{\XINT_flpower_checkB_b -}%
2464     0-{\XINT_flpower_checkB_b }#1%
2465   \krof
2466 }%
2467 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2468 \def\XINT_flpower_checkB_b #1#2.#3.%
2469 {%
2470   \expandafter\XINT_flpower_checkB_c
2471   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2472 }%

2473 \def\XINT_flpower_checkB_c #1.#2.%
2474 {%
2475   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2476 }%

2477 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2478 {%
2479   \expandafter \XINT_flpower_aa
2480   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2481 }%
```

7 Package *xintfrac* implementation

```

2482 \def\XINT_flpower_aa #1[#2]#3%
2483 {%
2484   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2485   \romannumeral\XINT_rep #3\endcsname0.#1.%
2486 }%
2487 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2488 \def\XINT_flpower_a #1%
2489 {%
2490   \xint_UDzerominusfork
2491   #1-\XINT_flpow_zero
2492   0#1{\XINT_flpower_b \iftrue}%
2493   0-{\XINT_flpower_b \iffalse#1}%
2494   \krof
2495 }%
2496 \def\XINT_flpower_b #1#2[#3]#4#5%
2497 {%
2498   \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiioOdd{#5}\fi}%
2499 }%
2500 \def\XINT_flpower_loopI #1.%
2501 {%
2502   \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2503   \ifodd\xintLDg{#1} %<- intentional space
2504   \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2505   \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2506   \romannumeral0\XINT_half
2507   #1\xint_bye\xint_Bye345678\xint_bye
2508   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2509 }%
2510 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2511 {%
2512   \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2513 }%
2514 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2515 {%
2516   \expandafter\XINT_flpower_toloopI
2517   \the\numexpr\expandafter\XINT_flpow_truncate
2518   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2519 }%
2520 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2521 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2522 {%
2523   \expandafter\XINT_flpower_toloopII
2524   \the\numexpr\expandafter\XINT_flpow_truncate
2525   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2526   #1.#2.#3.%
2527 }%
2528 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2529 \def\XINT_flpower_loopII #1.%

```

7 Package *xintfrac* implementation

```

2530 {%
2531   \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_IItoIII\fi
2532   \ifodd\xintLDg{#1} %<- intentional space
2533     \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2534   \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

2535   \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2536   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2537 }%
2538 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2539 {%
2540   \expandafter\XINT_flpower_toloopII
2541   \the\numexpr\expandafter\XINT_flpow_truncate
2542   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2543 }%
2544 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2545 {%
2546   \expandafter\XINT_flpower_loopII_odda
2547   \the\numexpr\expandafter\XINT_flpow_truncate
2548   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2549   #1.#2.#3.%
2550 }%
2551 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2552 {%
2553   \expandafter\XINT_flpower_toloopII
2554   \the\numexpr\expandafter\XINT_flpow_truncate
2555   \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2556   #4.#1.#2.%
2557 }%
2558 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2559 {%
2560   \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2561   \the\numexpr\expandafter\XINT_flpow_truncate
2562   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2563 }%

```

7.72 *\xintFloatFac*, *\XINTFloatFac*

```

2564 \def\xintFloatFac   {\romannumeral0\xintfloatfac}%
2565 \def\xintfloatfac   #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2566 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac }%
2567 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint:}%
2568 \def\XINT_flfac_chkopt #1#2%
2569 {%
2570   \ifx [#2\expandafter\XINT_flfac_opt
2571     \else\expandafter\XINT_flfac_noopt
2572   \fi
2573   #1#2%
2574 }%
2575 \def\XINT_flfac_noopt #1#2\xint:
2576 {%

```

7 Package *xintfrac* implementation

```

2577 \expandafter\XINT_FL_fac_fork_a
2578 \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]}%
2579 }%
2580 \def\XINT_flfac_opt #1[\xint:#2]%
2581 {%
2582 \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2583 }%
2584 \def\XINT_flfac_opt_a #1.#2#3%
2585 {%
2586 \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]}%
2587 }%
2588 \def\XINT_FL_fac_fork_a #1%
2589 {%
2590 \xint_UDzerominusfork
2591 #1-\XINT_FL_fac_iszero
2592 0#1\XINT_FL_fac_isneg
2593 0-\XINT_FL_fac_fork_b #1}%
2594 \krof
2595 }%
2596 \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.

```

2597 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2598 {%
2599 #5{\XINT_signalcondition{InvalidOperation}}
2600 {Factorial of negative: (-#1)!}{0[0]}%
2601 }%
2602 \def\XINT_FL_fac_fork_b #1.%
2603 {%
2604 \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2605 \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2606 \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2607 \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2608 \xint_orthat\XINT_FL_fac_small
2609 #1.%
2610 }%
2611 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2612 {%
2613 #5{\XINT_signalcondition{InvalidOperation}}
2614 {Factorial of too big: (#1)!}{0[0]}%
2615 }%

```

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

7 Package *xintfrac* implementation

flemme là).

We then want $\text{ceil}((P+k+n)/8)$. Using `\numexpr` rounding division (ARRRRRGGGHHHH), if m is a positive integer, $\text{ceil}(m/8)$ can be computed as $(m+3)/8$. Thus with $m=P+10+k$, this gives $Q<-(P+13+k)/8$. The routine actually computes $8(Q-1)$ for use in `\XINT_FL_fac_addzeros`.

With 1.2f the formula is $m=P+9+k$, $Q<-(P+12+k)/8$, and we use now $4=12-8$ rather than the earlier $5=13-8$. Whatever happens, the value computed in `\XINT_FL_fac_increaseP` is at least 8. There will always be an extra block.

Note: with `Digits:=32`; Maple gives for 200!:

```
> factorial(200.);
```

```
375
```

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

and this is the correct rounding because for 40 digits it computes

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to `xint`) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```
2616 \def\XINT_FL_fac_vbig
2617   {\expandafter\XINT_FL_fac_vbigloop_a
2618    \the\numexpr \XINT_FL_fac_increaseP \xint_c_i }%
2619 \def\XINT_FL_fac_big
2620   {\expandafter\XINT_FL_fac_bigloop_a
2621    \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii }%
2622 \def\XINT_FL_fac_med
2623   {\expandafter\XINT_FL_fac_medloop_a
2624    \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2625 \def\XINT_FL_fac_small
2626   {\expandafter\XINT_FL_fac_smallloop_a
2627    \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv }%
2628 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2629 {%
2630   #2\expandafter.\the\numexpr\xint_c_viii*%
2631   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2632    \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2633 }%
2634 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2635 \def\XINT_FL_fac_countdone #1#2\Z {#1}%
2636 \def\XINT_FL_fac_out #1;![#2]#3%
2637   {#3{\romannumeral0\XINT_mul_out
2638    #1;!1\R!1\R!1\R!1\R!1\R!%
2639    1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2640 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2641 {%
2642   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2643   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2644    \the\numexpr \xint_c_x^viii+#1.}%
2645 }%
2646 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2647 {%
2648   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2649   \expandafter\XINT_FL_fac_vbigloop_loop
2650   \the\numexpr #1+\xint_c_i\expandafter.%
```

7 Package *xintfrac* implementation

```

2651 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2652 }%
2653 \def\XINT_FL_fac_bigloop_a #1.%
2654 {%
2655 \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2656 #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2657 }%
2658 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2659 {%
2660 \expandafter\XINT_FL_fac_medloop_a
2661 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2662 }%
2663 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2664 {%
2665 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2666 \expandafter\XINT_FL_fac_bigloop_loop
2667 \the\numexpr #1+\xint_c_ii\expandafter.%
2668 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2669 }%
2670 \def\XINT_FL_fac_bigloop_mul #1!%
2671 {%
2672 \expandafter\XINT_FL_fac_mul
2673 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2674 }%
2675 \def\XINT_FL_fac_medloop_a #1.%
2676 {%
2677 \expandafter\XINT_FL_fac_medloop_b
2678 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2679 }%
2680 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2681 {%
2682 \expandafter\XINT_FL_fac_smallloop_a
2683 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2684 }%
2685 \def\XINT_FL_fac_medloop_loop #1.#2.%
2686 {%
2687 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2688 \expandafter\XINT_FL_fac_medloop_loop
2689 \the\numexpr #1+\xint_c_iii\expandafter.%
2690 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2691 }%
2692 \def\XINT_FL_fac_medloop_mul #1!%
2693 {%
2694 \expandafter\XINT_FL_fac_mul
2695 \the\numexpr
2696 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2697 }%
2698 \def\XINT_FL_fac_smallloop_a #1.%
2699 {%
2700 \csname
2701 XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2702 \endcsname #1.%

```


7 Package *xintfrac* implementation

```

2703 }%
2704 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2705 {%
2706   \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2707 }%
2708 \expandafter\def\csname XINT_FL_fac_smallloop_2\endcsname #1.#2.%
2709 {%
2710   \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2711 }%
2712 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2713 {%
2714   \XINT_FL_fac_addzeros #2.100000006!.{4.#1.}{#2}%
2715 }%
2716 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2717 {%
2718   \XINT_FL_fac_addzeros #2.100000024!.{5.#1.}{#2}%
2719 }%
2720 \def\XINT_FL_fac_addzeros #1.%
2721 {%
2722   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2723   \expandafter\XINT_FL_fac_addzeros
2724   \the\numexpr #1-\xint_c_viii.100000000!%
2725 }%

```

We will manipulate by successive **small** multiplications Q blocks $1\langle 8d\rangle!$, 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.

```

2726 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![-#4]}%
2727 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2728 {%
2729   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2730   \expandafter\XINT_FL_fac_smallloop_loop
2731   \the\numexpr #1+\xint_c_iv\expandafter.%
2732   \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2733 }%
2734 \def\XINT_FL_fac_smallloop_mul #1!%
2735 {%
2736   \expandafter\XINT_FL_fac_mul
2737   \the\numexpr
2738     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2739 }%[
2740 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
2741 \def\XINT_FL_fac_mul 1#1!%
2742   {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1]}%
2743 \def\XINT_FL_fac_mul_a #1-#2%
2744 {%
2745   \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2746   \expandafter\space\fi #11;!%
2747 }%
2748 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2749 {%
2750   \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2751 }%

```

7 Package *xintfrac* implementation

```

2752 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2753 {%
2754   \expandafter\XINT_FL_fac_minimulwc_c
2755   \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
2756 }%
2757 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2758 {%
2759   \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2760 }%
2761 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2762 {%
2763   \expandafter\XINT_FL_fac_minimulwc_e
2764   \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{#2}{#4}%
2765 }%
2766 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
2767 {%
2768   1#6#9\expandafter!%
2769   \the\numexpr\expandafter\XINT_FL_fac_smallmul
2770   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2771 }%
2772 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2773 {%
2774   \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2775   \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2776 }%

```

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.)

```

2777 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
2778 {%
2779   \ifnum #2=\xint_c_
2780     \expandafter\xint_firstoftwo\else
2781     \expandafter\xint_secondoftwo
2782   \fi
2783   {-2\relax[#4]}%
2784   {1#2\expandafter!\expandafter-\expandafter1\expandafter
2785     [\the\numexpr #4+\xint_c_viii]}%
2786 }%

```

7.73 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

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

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

```

2787 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2788 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
2789 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial }%
2790 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
2791 \def\XINT_flpfac_chkopt #1#2%

```

7 Package *xintfrac* implementation

```

2792 {%
2793   \ifx [#2\expandafter\XINT_flpfac_opt
2794     \else\expandafter\XINT_flpfac_noopt
2795   \fi
2796   #1#2%
2797 }%
2798 \def\XINT_flpfac_noopt #1#2\xint:#3%
2799 {%
2800   \expandafter\XINT_FL_pfac_fork
2801   \the\numexpr \xintNum{#2}\expandafter.%
2802   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]]}%
2803 }%
2804 \def\XINT_flpfac_opt #1[\xint:#2]%
2805 {%
2806   \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
2807 }%
2808 \def\XINT_flpfac_opt_b #1.#2#3#4%
2809 {%
2810   \expandafter\XINT_FL_pfac_fork
2811   \the\numexpr \xintNum{#3}\expandafter.%
2812   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
2813 }%
2814 \def\XINT_FL_pfac_fork #1#2.#3#4.%
2815 {%
2816   \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
2817   \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
2818   \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
2819   \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
2820   \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
2821 }%
2822 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
2823 {%
2824   #5{\XINT_signalcondition{InvalidOperation}}
2825   {pfactorial second arg too big: 9999999 < #2}{0[0]}%
2826 }%
2827 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
2828 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2829 \def\XINT_FL_pfac_neg -#1.-#2.%
2830 {%
2831   \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2832   \xint_orthat {%
2833     \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
2834     \expandafter\XINT_FL_pfac_increaseP}%
2835   \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2836 }%

```

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.

7 Package *xintfrac* implementation

```

2837 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2838 {%
2839   \expandafter\XINT_FL_pfac_a
2840   \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
2841     \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2842     /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2843     87654321\Z)/\xint_c_viii).#1.#2.%
2844 }%
2845 \def\XINT_FL_pfac_a #1.#2.#3.%
2846 {%
2847   \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2848   \the\numexpr#3\expandafter.%
2849   \romannumeral0\XINT_FL_pfac_addzeroes #1.1000000001!1;![-#1]%
2850 }%
2851 \def\XINT_FL_pfac_addzeroes #1.%
2852 {%
2853   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2854   \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
2855 }%
2856 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2857 \def\XINT_FL_pfac_b #1.%
2858 {%
2859   \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2860   \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2861   \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2862   \xint_orthat\XINT_FL_pfac_smallloop #1.%
2863 }%
2864 \def\XINT_FL_pfac_smallloop #1.#2.%
2865 {%
2866   \ifcase\numexpr #2-#1\relax
2867     \expandafter\XINT_FL_pfac_end_
2868   \or \expandafter\XINT_FL_pfac_end_i
2869   \or \expandafter\XINT_FL_pfac_end_ii
2870   \or \expandafter\XINT_FL_pfac_end_iii
2871   \else\expandafter\XINT_FL_pfac_smallloop_a
2872   \fi #1.#2.%
2873 }%
2874 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2875 {%
2876   \expandafter\XINT_FL_pfac_smallloop_b
2877   \the\numexpr #1+\xint_c_iv\expandafter.%
2878   \the\numexpr #2\expandafter.%
2879   \romannumeral0\expandafter\XINT_FL_fac_mul
2880   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2881 }%
2882 \def\XINT_FL_pfac_smallloop_b #1.%
2883 {%
2884   \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2885     \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2886 }%
2887 \def\XINT_FL_pfac_medloop #1.#2.%
2888 {%

```

7 Package *xintfrac* implementation

```

2889 \ifcase\numexpr #2-#1\relax
2890   \expandafter\XINT_FL_pfac_end_
2891 \or \expandafter\XINT_FL_pfac_end_i
2892 \or \expandafter\XINT_FL_pfac_end_ii
2893 \else\expandafter\XINT_FL_pfac_medloop_a
2894 \fi #1.#2.%
2895 }%
2896 \def\XINT_FL_pfac_medloop_a #1.#2.%
2897 {%
2898   \expandafter\XINT_FL_pfac_medloop_b
2899   \the\numexpr #1+\xint_c_iii\expandafter.%
2900   \the\numexpr #2\expandafter.%
2901   \romannumeral0\expandafter\XINT_FL_fac_mul
2902   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2903 }%
2904 \def\XINT_FL_pfac_medloop_b #1.%
2905 {%
2906   \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2907     \expandafter\XINT_FL_pfac_medloop \fi #1.%
2908 }%
2909 \def\XINT_FL_pfac_bigloop #1.#2.%
2910 {%
2911   \ifcase\numexpr #2-#1\relax
2912     \expandafter\XINT_FL_pfac_end_
2913   \or \expandafter\XINT_FL_pfac_end_i
2914   \else\expandafter\XINT_FL_pfac_bigloop_a
2915   \fi #1.#2.%
2916 }%
2917 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2918 {%
2919   \expandafter\XINT_FL_pfac_bigloop_b
2920   \the\numexpr #1+\xint_c_ii\expandafter.%
2921   \the\numexpr #2\expandafter.%
2922   \romannumeral0\expandafter\XINT_FL_fac_mul
2923   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2924 }%
2925 \def\XINT_FL_pfac_bigloop_b #1.%
2926 {%
2927   \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2928     \expandafter\XINT_FL_pfac_bigloop \fi #1.%
2929 }%
2930 \def\XINT_FL_pfac_vbigloop #1.#2.%
2931 {%
2932   \ifnum #2=#1
2933     \expandafter\XINT_FL_pfac_end_
2934   \else\expandafter\XINT_FL_pfac_vbigloop_a
2935   \fi #1.#2.%
2936 }%
2937 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
2938 {%
2939   \expandafter\XINT_FL_pfac_vbigloop
2940   \the\numexpr #1+\xint_c_i\expandafter.%

```

7 Package *xintfrac* implementation

```

2941 \the\numexpr #2\expandafter.%
2942 \romannumeral0\expandafter\XINT_FL_fac_mul
2943 \the\numexpr\xint_c_x^viii+#1!%
2944 }%
2945 \def\XINT_FL_pfac_end_iii #1.#2.%
2946 {%
2947 \expandafter\XINT_FL_fac_out
2948 \romannumeral0\expandafter\XINT_FL_fac_mul
2949 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2950 }%
2951 \def\XINT_FL_pfac_end_ii #1.#2.%
2952 {%
2953 \expandafter\XINT_FL_fac_out
2954 \romannumeral0\expandafter\XINT_FL_fac_mul
2955 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2956 }%
2957 \def\XINT_FL_pfac_end_i #1.#2.%
2958 {%
2959 \expandafter\XINT_FL_fac_out
2960 \romannumeral0\expandafter\XINT_FL_fac_mul
2961 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2962 }%
2963 \def\XINT_FL_pfac_end_ #1.#2.%
2964 {%
2965 \expandafter\XINT_FL_fac_out
2966 \romannumeral0\expandafter\XINT_FL_fac_mul
2967 \the\numexpr \xint_c_x^viii+#1!%
2968 }%

```

7.74 `\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.

```

2969 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
2970 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
2971 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial }%
2972 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
2973 \def\XINT_flbinom_chkopt #1#2%
2974 {%
2975 \ifx [#2\expandafter\XINT_flbinom_opt
2976 \else\expandafter\XINT_flbinom_noopt
2977 \fi #1#2%
2978 }%
2979 \def\XINT_flbinom_noopt #1#2\xint:#3%
2980 {%
2981 \expandafter\XINT_FL_binom_a

```

7 Package *xintfrac* implementation

```

2982 \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
2983 }%
2984 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
2985 {%
2986 \expandafter\XINT_FL_binom_a
2987 \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
2988 \the\numexpr #2.#1%
2989 }%
2990 \def\XINT_FL_binom_a #1.#2.%
2991 {%
2992 \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
2993 }%
2994 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
2995 {%
2996 \if-#5\xint_dothis \XINT_FL_binom_neg\fi
2997 \if-#1\xint_dothis \XINT_FL_binom_zero\fi
2998 \if-#3\xint_dothis \XINT_FL_binom_zero\fi
2999 \if0#1\xint_dothis \XINT_FL_binom_one\fi
3000 \if0#3\xint_dothis \XINT_FL_binom_one\fi
3001 \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
3002 \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3003 \xint_orthat\XINT_FL_binom_aa
3004 #1#2.#3#4.#5#6.%
3005 }%
3006 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3007 {%
3008 #5[#4]{\XINT_signalcondition{InvalidOperation}}
3009 {binomial with first arg negative: #3}{\{0[0]}}%
3010 }%
3011 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3012 {%
3013 #5[#4]{\XINT_signalcondition{InvalidOperation}}
3014 {binomial with first arg too big: 99999999 < #3}{\{0[0]}}%
3015 }%
3016 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3017 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3018 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3019 {%
3020 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3021 #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3022 {\XINT_FL_fac_fork_b
3023 #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3024 }%
3025 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3026 {%
3027 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3028 #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3029 {\XINT_FL_fac_fork_b
3030 #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3031 }%

```

7.75 `\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.

```

3032 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqr }%
3033 \def\xintfloatsqr      #1{\XINT_flsqr_chkopt \xintfloat #1\xint:}%
3034 \def\XINTinFloatSqrt   {\romannumeral0\XINTinfloatsqr }%
3035 \def\XINTinfloatsqr    #1{\XINT_flsqr_chkopt \XINTinfloat #1\xint:}%
3036 \def\XINT_flsqr_chkopt #1#2%
3037 {%
3038   \ifx [#2\expandafter\XINT_flsqr_opt
3039     \else\expandafter\XINT_flsqr_noopt
3040   \fi #1#2%
3041 }%
3042 \def\XINT_flsqr_noopt #1#2\xint:%
3043 {%
3044   \expandafter\XINT_FL_sqr_a
3045     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3046 }%
3047 \def\XINT_flsqr_opt #1[\xint:#2]#3%
3048 {%
3049   \expandafter\XINT_flsqr_opt_a\the\numexpr #2.#1%
3050 }%
3051 \def\XINT_flsqr_opt_a #1.#2#3%
3052 {%
3053   \expandafter\XINT_FL_sqr_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3054 }%
3055 \def\XINT_FL_sqr_a #1%
3056 {%
3057   \xint_UDzerominusfork
3058     #1-\XINT_FL_sqr_iszero
3059     0#1\XINT_FL_sqr_isneg
3060     0-{\XINT_FL_sqr_pos #1}%
3061   \krof
3062 }%[
3063 \def\XINT_FL_sqr_iszero #1]#2.#3{#3[#2]{0[0]}}%
3064 \def\XINT_FL_sqr_isneg #1]#2.#3%
3065 {%
3066   #3[#2]{\XINT_signalcondition{InvalidOperation}
3067     {Square root of negative: -#1}}{0[0]}}%
3068 }%

```


7 Package *xintfrac* implementation

```

3069 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3070 {%
3071   \expandafter\XINT_flsqrt
3072   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3073   \xint_orthat {+\xint_c_ii.#2.{}#100.#3.%
3074 }%

3075 \def\XINT_flsqrt #1.#2.%
3076 {%
3077   \expandafter\XINT_flsqrt_a
3078   \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3079 }%

3080 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3081 {%
3082   \expandafter\XINT_flsqrt_b
3083   \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3084   \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3085 }%

3086 \def\XINT_flsqrt_b #1.#2#3%
3087 {%
3088   \expandafter\XINT_flsqrt_c
3089   \romannumeral0\xintiisub
3090   {\XINT_dsx_addzeros {#1}#2;}%
3091   {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3092   {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3093 }%

3094 \def\XINT_flsqrt_c #1.#2.%
3095 {%
3096   \expandafter\XINT_flsqrt_d
3097   \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3098 }%

3099 \def\XINT_flsqrt_d #1.#2#3.%
3100 {%
3101   \ifnum #2=\xint_c_v
3102   \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3103   #2#3.#1.%
3104 }%

3105 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

```

```

3106 \def\XINT_flsqrt_f 5#1.%
3107   {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.}%
3108 \def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3109   \xint_orthat{\XINT_flsqrt_finish 5.}}%
3110 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3111   \xint_orthat{\XINT_flsqrt_finish 5.}}%

```

```

3112 \def\XINT_flsqrt_again #1.#2.%
3113 {%
3114   \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3115 }%

```

```

3116 \def\XINT_flsqrt_again_a #1.#2.#3.%
3117 {%
3118   \expandafter\XINT_flsqrt_b
3119   \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3120   \romannumeral0\XINT_sqrt_start #1.#200000000.#3.%
3121   #1.#200000000.#3.%
3122 }%

```

7.76 `\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]`.

```

3123 \def\xintFloatE   {\romannumeral0\xintfloate }%
3124 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3125 \def\XINT_floate_chkopt #1%
3126 {%
3127   \ifx [#1\expandafter\XINT_floate_opt
3128     \else\expandafter\XINT_floate_noopt
3129   \fi #1%
3130 }%
3131 \def\XINT_floate_noopt #1\xint:%
3132 {%
3133   \expandafter\XINT_floate_post
3134   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3135 }%
3136 \def\XINT_floate_opt [\xint:#1]%
3137 {%
3138   \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3139 }%
3140 \def\XINT_floate_opt_a #1.#2%
3141 {%
3142   \expandafter\XINT_floate_post
3143   \romannumeral0\XINTinfloat[#1]{#2}#1.%

```

```

3144 }%
3145 \def\XINT_floate_post #1%
3146 {%
3147   \xint_UDzerominusfork
3148   #1-\XINT_floate_zero
3149   0#1\XINT_floate_neg
3150   0-\XINT_floate_pos
3151   \krof #1%
3152 }%[
3153 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3154 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%

3155 \def\XINT_floate_pos #1#2[#3]#4.#5%
3156 {%
3157   \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3158 }%
3159 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3160 \def\XINTinfloate
3161   {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]}%
3162 \def\XINT_infloate #1[#2]#3%
3163   {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3164 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

7.77 `\XINTinFloatMod`

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

```

3165 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
3166 \def\XINTinfloatmod [#1]#2#3%
3167 {%
3168   \XINTinfloat[#1]{\xintMod
3169     {\romannumeral0\XINTinfloat[#1]{#2}}%
3170     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3171 }%

```

7.78 `\XINTinFloatDivFloor`

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

```

3172 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]}%
3173 \def\XINTinfloatdivfloor [#1]#2#3%
3174 {%
3175   \xintdivfloor
3176     {\romannumeral0\XINTinfloat[#1]{#2}}%
3177     {\romannumeral0\XINTinfloat[#1]{#3}}%
3178 }%

```

7.79 `\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.

```

3179 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]}%
3180 \def\XINTinfloatdivmod [#1]#2#3%
3181 {%
3182   \expandafter\XINT_infloatdivmod
3183   \romannumeral0\xintdivmod
3184     {\romannumeral0\XINTinfloat[#1]{#2}}%
3185     {\romannumeral0\XINTinfloat[#1]{#3}}%
3186   {#1}%
3187 }%
3188 \def\XINT_infloatdivmod #1#2#3{ #1,\XINTinFloat[#3]{#2}}%

```

7.80 `\xintifFloatInt`

```

3189 \def\xintifFloatInt {\romannumeral0\xintiffloatint}%
3190 \def\xintiffloatint #1{\expandafter\XINT_iffloatint
3191   \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
3192 \def\XINT_iffloatint #1#2/1[#3]%
3193 {%
3194   \if 0#1\xint_dothis\xint_firstoftwo_thenstop\fi
3195   \ifnum#3<\xint_c_\xint_dothis\xint_secondoftwo_thenstop\fi
3196   \xint_orthat\xint_firstoftwo_thenstop
3197 }%

```

7.81 (WIP) `\XINTinRandomFloatS`

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:

```

from random import random
from struct import pack
def binary(num):
    return ''.join('{:0>8b}'.format(c) for c in pack('>d', num))
def test(n):
    x=random()
    while 2**n*x>=1:

```

```

        x=random()
        return x, binary(x)
# ou simplement
def test(n):
    x=random()
    while 2**n*x>=1:
        x=random()
    return x, 2**53*x

```

```

3198 \def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
3199 \def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3200 \def\XINTinrandomfloatS[#1]%
3201 {%
3202   \expandafter\XINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
3203 }%
3204 \def\XINT_inrandomfloatS-#1\xint:
3205 {%
3206   \expandafter\XINT_inrandomfloatS_a
3207   \romannumeral0\xintrandomdigits{#1}[-#1]%
3208 }%

```

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.

```

3209 \def\XINT_inrandomfloatS_a#1%
3210 {%
3211   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3212   \xint_orthat{ #1}%
3213 }%[
3214 \def\XINT_inrandomfloatS_b#1%
3215 {%
3216   \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
3217   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3218   \xint_orthat{ #1}%
3219 }%[
3220 \def\XINT_inrandomfloatS_zero#1]{ 0[0]}%

```

7.82 (WIP) \XINTinRandomFloatSixteen

1.3b. Support for grand() function.

```

3221 \def\XINTinRandomFloatSixteen%
3222 {%
3223   \romannumeral0\expandafter\XINT_inrandomfloatS_a
3224   \romannumeral`&&\expandafter\XINT_eightrandomdigits
3225   \romannumeral`&&\XINT_eightrandomdigits[-16]%
3226 }%
3227 \XINT_restorecatcodes_endinput%

```

8 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection . . .	262	.7	<code>\xintRationalSeries</code>	265
.2	Package identification	263	.8	<code>\xintRationalSeriesX</code>	266
.3	<code>\xintSeries</code>	263	.9	<code>\xintFxpPtPowerSeries</code>	267
.4	<code>\xintiSeries</code>	263	.10	<code>\xintFxpPtPowerSeriesX</code>	268
.5	<code>\xintPowerSeries</code>	264	.11	<code>\xintFloatPowerSeries</code>	268
.6	<code>\xintPowerSeriesX</code>	265	.12	<code>\xintFloatPowerSeriesX</code>	270

The commenting is currently (2018/05/18) 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@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

```

8.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintseries}%
46 [2018/05/18 1.3b Expandable partial sums with xint package (JFB)]%

```

8.3 `\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 }%

```

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

```

8.5 `\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 }%

```

8.6 `\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 }%

```

8.7 `\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 }%

```

8.8 \xintRationalSeriesX

a,b,initial,ratiofunction,x

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

```

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

```

```

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

```

8.9 `\xintFxpPowerSeries`

I am not two happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to `\numexpr`.

```

210 \def\xintFxpPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213 \expandafter\XINT_fppowseries\expandafter
214 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218 \ifnum #2<#1
219 \xint_afterfi { 0}%
220 \else
221 \xint_afterfi
222 {\expandafter\XINT_fppowseries_loop_pre\expandafter
223 {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224 {#1}{#4}{#2}{#3}{#5}%
225 }%
226 \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230 \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231 \expandafter\XINT_fppowseries_loop_i\expandafter
232 {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233 {\romannumeral0\xintitrunc {#6}{\xintMul {#5{#2}}{#1}}}%
234 {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237 {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241 \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242 \expandafter\XINT_fppowseries_loop_ii\expandafter
243 {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244 {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
247 {%
248 \expandafter\XINT_fppowseries_loop_i\expandafter

```

```

249   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250   {\romannumeral0\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}}%
251   {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\xINT_fppowseries_exit_i\fi\expandafter\xINT_fppowseries_loop_ii
254   {\fi \expandafter\xINT_fppowseries_exit_ii }%
255 \def\xINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257   \xinttrunc {#7}
258   {\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}[-#7]}}%
259 }%

```

8.10 `\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 }%

```

8.11 `\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

```

8 Package *xintseries* implementation

```

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 }%

```

8.12 \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%

```

9 Package *xintcfrac* implementation

.1	Catcodes, ε - \TeX and reload detection	271	.16	\backslash xintiGtoF	283
.2	Package identification	272	.17	\backslash xintCtoCv, \backslash xintCstoCv	284
.3	\backslash xintCFrac	272	.18	\backslash xintiCstoCv	285
.4	\backslash xintGCFrac	273	.19	\backslash xintGctoCv	285
.5	\backslash xintGGCFrac	275	.20	\backslash xintiGctoCv	287
.6	\backslash xintGctoGCx	276	.21	\backslash xintFtoCv	288
.7	\backslash xintFtoCs	276	.22	\backslash xintFtoCCv	288
.8	\backslash xintFtoCx	277	.23	\backslash xintCntoF	288
.9	\backslash xintFtoC	277	.24	\backslash xintGcntoF	289
.10	\backslash xintFtoGC	278	.25	\backslash xintCntoCs	290
.11	\backslash xintFGtoC	278	.26	\backslash xintCntoGC	290
.12	\backslash xintFtoCC	279	.27	\backslash xintGcntoGC	291
.13	\backslash xintCtoF, \backslash xintCstoF	280	.28	\backslash xintCstoGC	292
.14	\backslash xintiCstoF	281	.29	\backslash xintGctoGC	292
.15	\backslash xintGctoF	281			

The commenting is currently (2018/05/18) very sparse. Release 1.09m (2014/02/26) has modified a few things: \backslash xintFtoCs and \backslash xintCntoCs insert spaces after the commas, \backslash xintCstoF and \backslash xintCstoCv authorize spaces in the input also before the commas, \backslash xintCntoCs does not brace the produced coefficients, new macros \backslash xintFtoC, \backslash xintCtoF, \backslash xintCtoCv, \backslash xintFGtoC, and \backslash xintGGCFrac.

There is partial dependency on *xinttools* due to \backslash xintCstoF and \backslash xintCsToCv.

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@xintcfrac.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
15 \expandafter
16  \ifx\csname PackageInfo\endcsname\relax
17    \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18  \else
19    \def\y#1#2{\PackageInfo{#1}{#2}}%
20  \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23  \y{xintcfrac}{\numexpr not available, aborting input}%
24  \aftergroup\endinput

```

```

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

```

9.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcfrac}%
46 [2018/05/18 1.3b Expandable continued fractions with xint package (JFB)]%

```

9.3 \xintCFrac

```

47 \def\xintCFrac {\romannumeral0\xintcfrac }%
48 \def\xintcfrac #1%
49 {%
50   \XINT_cfrac_opt_a #1\xint:
51 }%
52 \def\XINT_cfrac_opt_a #1%
53 {%
54   \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
55 }%
56 \def\XINT_cfrac_noopt #1\xint:
57 {%
58   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
59   \relax\relax
60 }%
61 \def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint:#1]%
62 {%
63   \fi\csname XINT_cfrac_opt#1\endcsname
64 }%
65 \def\XINT_cfrac_optl #1%
66 {%
67   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
68   \relax\hfill
69 }%
70 \def\XINT_cfrac_optc #1%
71 {%
72   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z

```


9 Package *xintcfrac* implementation

```

73   \relax\relax
74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77   \expandafter\XINT_cfrac_A\romannumeral0\xintraewithzeros {#1}\Z
78   \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82   \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86   \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90   \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}%
94 \def\XINT_cfrac_loop_a
95 {%
96   \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100   \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104   \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108   \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111   {\XINT_cfrac_T #5#6{#2}#4\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114   \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#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}%

9.4 \xintGCFrac

121 \def\xintGCFrac {\romannumeral0\xintgcfrac }%
122 \def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint:}%
123 \def\XINT_gcfrac_opt_a #1%

```

9 Package *xintcfrac* implementation

```

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}%

```

9.5 `\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}%
```

9.6 `\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}%
```

9.7 `\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}%

```

9.8 `\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}%

```

9.9 `\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 {}}%
```

9.10 `\xintFtoGC`

```
316 \def\xintFtoGC {\romannumeral0\xintftogc }%
317 \def\xintftogc {\xintftocx {+1/}}%
```

9.11 `\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 }%

```

9.12 `\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 }%

```

9.13 `\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 {%

```


9 Package *xintcfrac* implementation

```
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]
```

9.14 *\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]
```

9.15 *\xintGctof*

9 Package *xintcfrac* implementation

```

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]

```

9.16 \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]

```

9.17 `\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}%

```

9.18 `\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]

```

9.19 `\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 {%

```

9 Package *xintcfrac* implementation

```

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}%

```

9.20 \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]

```

9.21 `\xintFtoCv`

Still uses `\xinticstocv` `\xintFtoCs` rather than `\xintctocv` `\xintFtoC`.

```

823 \def\xintFtoCv {\romannumeral0\xintftocv }%
824 \def\xintftocv #1%
825 {%
826   \xinticstocv {\xintFtoCs {#1}}%
827 }%

```

9.22 `\xintFtoCCv`

```

828 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
829 \def\xintftoccv #1%
830 {%
831   \xintigctocv {\xintFtoCC {#1}}%
832 }%

```

9.23 `\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 }%

```

9.24 `\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 }%

```

9.25 `\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

```

9.26 `\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/{ }%

```

9.27 `\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/{ }%

```

9.28 *\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}%

```

9.29 *\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}/}%

```

9 Package *xintcfrac* implementation

```
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%
```

10 Package `xintexpr` implementation

Contents

10.1	Old comments	295
10.2	Catcodes, ε -TeX and reload detection	295
10.3	Package identification	297
10.4	Locking and unlocking	297
10.5	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	298
10.6	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	298
10.7	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	298
10.8	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiieexpr</code>	298
10.9	<code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiieexpr</code>	298
10.10	<code>\xintthe</code>	298
10.11	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiieexpr</code>	298
10.12	<code>\xintthecoords</code>	298
10.13	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	299
10.14	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	299
10.15	<code>\xinteval</code> , <code>\xintiieval</code>	299
10.16	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	299
10.17	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	300
10.18	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	300
10.19	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliieexpr</code>	300
10.20	Hooks for the functioning of <code>\xintNewExpr</code> and <code>\xintdeffunc</code>	301
10.21	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	301
10.22	<code>\XINT_expr_getnext</code> : fetching some number then an operator	302
10.23	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	303
10.24	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	310
10.25	Expansion spanning; opening and closing parentheses	312
10.26	<code> </code> , <code> </code> , <code>&</code> , <code>&&</code> , <code><</code> , <code>></code> , <code>=</code> , <code>==</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>:</code> , and <code>++</code> operators	313
10.27	Macros for list selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	320
10.28	Macros for a..b list generation	325
10.29	Macros for a..[d]..b list generation	327
10.30	The comma as binary operator	329
10.31	The minus as prefix operator of variable precedence level	329
10.32	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches	330
10.33	<code>!</code> as postfix factorial operator	331
10.34	The A/B[N] mechanism	331
10.35	<code>\XINT_expr_op_`</code> for recognizing functions	331
10.36	The <code>bool</code> , <code>togl</code> , <code>protect</code> pseudo “functions”	332
10.37	The <code>break</code> function	332
10.38	The <code>qint</code> , <code>qfrac</code> , <code>qfloat</code> “functions”	333
10.39	The <code>random()</code> and <code>grand()</code> “functions”	333
10.40	<code>\XINT_expr_op__</code> for recognizing variables	333
10.41	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	333
10.42	<code>\xintunassignvar</code>	335
10.43	<code>seq</code> and the implementation of dummy variables	336
10.44	<code>add</code> , <code>mul</code>	341
10.45	<code>subs</code>	342

10.46	rseq	343
10.47	iter	344
10.48	rrseq	346
10.49	iterr	348
10.50	Macros handling csv lists for functions with multiple comma separated arguments in expressions	350
10.51	Auxiliary wrappers for function macros	353
10.52	The num, reduce, preduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, divmod, gcd, lcm, max, min, `+`, `*`, `?`, `!`, not, all, any, xor, if, ifsgn, even, odd, first, last, len, reversed, factorial, binomial, and randrange functions	354
10.53	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	364
10.54	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	366
10.55	<code>\xintNewFunction</code>	368
10.56	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	369

This is release 1.3b of [2018/05/18].

10.1 Old comments

These general comments were last updated at the end of the 1.09x series in 2014. The principles remain in place to this day but refer to [CHANGES.html](#) for some significant evolutions since.

The first version was released in June 2013. I was greatly helped in this task of writing an expandable parser of infix operations by the comments provided in `l3fp-parse.dtx` (in its version as available in April-May 2013). One will recognize in particular the idea of the ``until'` macros; I have not looked into the actual `l3fp` code beyond the very useful comments provided in its documentation.

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

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

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

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

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

10.2 Catcodes, ε -TEX and reload detection

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

Package *xintexpr* implementation

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

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



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

10.3 Package identification

```
55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2018/05/18 1.3b Expandable expression parser (JFB)]%
58 \catcode`! 11
59 \let\XINT_Cmp \xintiiCmp
```

10.4 Locking and unlocking

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannu-``meral-`0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed 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 `\romannu-``meral` are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the `lockscan` macro after a chain of `\romannu-``meral-`0` had ended gathering digits; this uses has been replaced by direct processing inside a `\csname... \endcsname` and the macro is kept only for matters of dummy variables.

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

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

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

10.5 `\XINT_expr_wrap`, `\XINT_iiexpr_wrap`

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

10.6 `\XINT_protectii`, `\XINT_expr_usethe`

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

10.7 `\XINT_expr_print`, `\XINT_iiexpr_print`, `\XINT_boolexpr_print`

See also the `\XINT_flexpr_print` which is special, below.

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

10.8 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiexpr`

```
89 \def\xintexpr {\romannumeral0\xinteval }%
90 \def\xintiexpr {\romannumeral0\xintieval }%
91 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
92 \def\xintiexpr {\romannumeral0\xintiieval }%
```

10.9 `\xinttheexpr`, `\xinttheiexpr`, `\xintthefloatexpr`, `\xinttheiexpr`

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

10.10 `\xintthe`

```
99 \def\xintthe #1{\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&#1}%
```

10.11 `\thexintexpr`, `\thexintiexpr`, `\thexintfloatexpr`, `\thexintiexpr`

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

```
100 \let\thexintexpr \xinttheexpr
101 \let\thexintiexpr \xinttheiexpr
102 \let\thexintfloatexpr\xintthefloatexpr
103 \let\thexintiexpr \xinttheiexpr
```

10.12 `\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 `\curname` 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!

```

104 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
105     \XINT_thecoords_a
106     \expandafter\xint_gobble_iii\romannumeral0#1}%
107 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
108   {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
109     \romannumeral`&&@#1#2,!,!,^\endcsname }%
110 \def\XINT_thecoords_b #1#2,#3#4,%
111   {\xint_gob_til! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
112 \def\XINT_thecoords_c #1^{}%

```

10.13 `\xintbareeval`, `\xintbarefloateval`, `\xintbareieval`

```

113 \def\xintbareeval
114   {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
115 \def\xintbarefloateval
116   {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
117 \def\xintbareieval
118   {\expandafter\XINT_iiexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%

```

10.14 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareieval`

```

119 \def\xintthebareeval   {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
120 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
121 \def\xintthebareieval   {\expandafter\XINT_expr_unlock\romannumeral0\xintbareieval}%

```

10.15 `\xinteval`, `\xintiieval`

```

122 \def\xinteval   {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
123 \def\xintiieval {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareieval }%

```

10.16 `\xintieval`, `\XINT_iexpr_wrap`

Optional argument since 1.1.

```

124 \def\xintieval #1%
125   {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
126 \def\XINT_iexpr_noopt
127   {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
128 \def\XINT_iexpr_withopt [#1]%
129 {%
130   \expandafter\XINT_iexpr_wrap\expandafter
131   {\the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter}%
132   \romannumeral0\xintbareeval
133 }%
134 \def\XINT_iexpr_wrap #1#2%
135 {%
136   \expandafter\XINT_expr_wrap
137   \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
138 }%

```

10.17 `\xintfloateval`, `\XINT_flexpr_wrap`, `\XINT_flexpr_print`

Optional argument since 1.1

```

139 \def\xintfloateval #1%
140 {%
141   \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
142   \fi #1%
143 }%
144 \def\XINT_flexpr_noopt
145 {%
146   \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
147   \romannumeral0\xintbarefloateval
148 }%
149 \def\XINT_flexpr_withopt_a [#1]%
150 {%
151   \expandafter\XINT_flexpr_withopt_b\expandafter
152   {\the\numexpr\xint_zapspace #1 \xint_gobble_i\expandafter}%
153   \romannumeral0\xintbarefloateval
154 }%
155 \def\XINT_flexpr_withopt_b #1#2%
156 {%
157   \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
158   \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
159 }%
160 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
161 \def\XINT_flexpr_print #1%
162 {%
163   \expandafter\xintPFloat::csv
164   \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%
165 }%
166 \catcode` : 12
167   \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
168 \catcode` : 11

```

10.18 `\xintboolexpr`, `\xinttheboolexpr`, `\thexintboolexpr`

```

169 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
170   \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
171 \def\xinttheboolexpr   {\romannumeral`&&\expandafter\expandafter\expandafter
172   \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
173 \let\thexintboolexpr\xinttheboolexpr
174 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%

```

10.19 `\xintifboolexpr`, `\xintifboolfloatexpr`, `\xintifbooliiexpr`

Do not work with comma separated expressions.

```

175 \def\xintifboolexpr    #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
176 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
177 \def\xintifbooliiexpr  #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%

```

10.20 Hooks for the functioning of `\xintNewExpr` and `\xintdeffunc`

This is new with 1.3. See `\XINT_expr_redefinemacros`.

```
178 \let\XINT:NEhook:two\empty
179 \let\XINT:NEhook:one\empty
180 \let\XINT:NEhook:csv\empty
```

10.21 Macros handling csv lists on output (for `\XINT_expr_print` et al. routines)

10.21.1	<code>\XINT::_end</code>	301
10.21.2	<code>\xintCSV::csv</code>	301
10.21.3	<code>\xintSPRaw, \xintSPRaw::csv</code>	301
10.21.4	<code>\xintIsTrue::csv</code>	301
10.21.5	<code>\xintRound::csv</code>	302
10.21.6	<code>\XINTinFloat::csv</code>	302
10.21.7	<code>\xintPFloat::csv</code>	302

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

10.21.1 `\XINT::_end`

Le mécanisme est le suivant, #2 est dans des accolades et commence par `,<sp>`. Donc le gobble se débarrasse du, et le `<sp>` après brace stripping arrête un `\romannumeral0` ou `\romannumeral-`0`

```
181 \def\XINT::_end #1,#2{\xint_gobble_i #2}%
```

10.21.2 `\xintCSV::csv`

```
182 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^,%}
183 \def\XINT_csv::_a {\XINT_csv::_b {}}%
184 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
185 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
186 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

10.21.3 `\xintSPRaw, \xintSPRaw::csv`

```
187 \def\xintSPRaw {\romannumeral0\xintspraw }%
188 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
189 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
190 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
191 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
192 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^,%}
193 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
194 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
195 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
196 \if ^#1\xint_dothis\XINT::_end\fi
197 \xint_orthat\XINT_spraw::_d #1}%
198 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
199 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

10.21.4 `\xintIsTrue::csv`

```
200 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^,%}
201 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
```

```

202 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
203 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
204         \if ^#1\xint_dothis\XINT::__end\fi
205         \xint_orthat\XINT_istrue::_d #1}%
206 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
207 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2}, #1}}%

```

10.21.5 `\xintRound::csv`

```

208 \def\XINT::_:_end #1,#2#3{\xint_gobble_i #3}%
209 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
210     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
211 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
212 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
213         \if ^#1\xint_dothis\XINT::__end\fi
214         \xint_orthat\XINT_round::_d #1}%
215 \def\XINT_round::_d #1,#2{%
216     \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
217     \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
218 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3}, #1}}%

```

10.21.6 `\XINTinFloat::csv`

```

219 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
220     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
221 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
222 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
223         \if ^#1\xint_dothis\XINT::__end\fi
224         \xint_orthat\XINT_infloat::_d #1}%
225 \def\XINT_infloat::_d #1,#2%
226     {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
227 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3}, #1}}%

```

10.21.7 `\xintPFloat::csv`

```

228 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
229     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
230 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
231 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
232         \if ^#1\xint_dothis\XINT::__end\fi
233         \xint_orthat\XINT_pfloat::_d #1}%
234 \def\XINT_pfloat::_d #1,#2%
235     {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint:#2]{#1},{#2}}%
236 \def\XINT_pfloat::_e #1,#2#3{\XINT_pfloat::_b {#2}{#3}, #1}}%

```

10.22 `\XINT_expr_getnext`: fetching some number then an operator

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then `\romannumeral-`0` expansion.

```

237 \def\XINT_expr_getnext #1%
238 {%
239     \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
240 }%
241 \def\XINT_expr_getnext_a #1%
242 {% screens out sub-expressions and \count or \dimen registers/variables

```

```

243 \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
244 \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
245 \expandafter\XINT_expr_countetc
246 \else
247 \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
248 \fi
249 #1%
250 }%
251 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

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

```

252 \def\XINT_expr_countetc #1%
253 {%
254 \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
255 \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
256 \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
257 \ifx\fontchardp#1\else\ifx\fontcharic#1\else
258 \XINT_expr_unpackvar
259 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
260 \expandafter\XINT_expr_getnext\number #1%
261 }%
262 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
263 \expandafter\XINT_expr_getnext\number #1%
264 {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
265 \expandafter\XINT_expr_getop\csname .=#1\endcsname }%
266 \begingroup
267 \lccode`*=`#
268 \lowercase{\endgroup
269 \def\XINT_expr_getnextfork #1{%
270 \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
271 \if#1[\xint_dothis {\xint_c_xviii ({})}\fi
272 \if#1+\xint_dothis \XINT_expr_getnext \fi
273 \if#1.\xint_dothis {\XINT_expr_startdec}\fi
274 \if#1-\xint_dothis -\fi
275 \if#1(\xint_dothis {\xint_c_xviii ({})}\fi
276 \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
277 }%
278 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

10.23 The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser

10.23.1	Integral part (skipping zeroes)	304
10.23.2	Fractional part	306
10.23.3	Scientific notation	307
10.23.4	Hexadecimal numbers	308
10.23.5	Parsing names of functions and variables	309

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with `\xintiexpr` which should never be given a `[n]` inside its `\csname.=<digits>\endcsname` storage of numbers (because its arithmetic uses the `ii` macros which know nothing about the `[N]` notation). Hence if the parser has only seen digits when hitting something else than the dot or `e` (or `E`), it will not insert a `[0]`. Thus we very slightly compromise the efficiency of `\xintexpr` and `\xintfloatexpr` in order to be able to share the same code with `\xintiexpr`.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside `\xintexpr` or `\xintiexpr`. On the other hand if a dot or a `e` (or `E`) is met, then the (common) parser has no scruples ending this number with a `[n]`, this will provoke an error later if that was within an `\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 pick only one token (could have been braced stuff), thus here it will be `+` or `*`, and via `\XINT_expr_op_`` this into becomes a suitable `\XINT_{expr|iiexpr|fexpr}_func_+` (or `*`). Documentation of 1.1 said to use ``+`(...)`, but ``+`(...)` is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```

279 \catcode96 11 % `
280 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
281 {%
282   \if )#1\xint_dothis \XINT_expr_gotnil \fi
283   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
284   \if `#1\xint_dothis {\XINT_expr_onliteral_}\fi
285   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
286   \xint_orthat \XINT_expr_scanfunc #1%
287 }%
288 \def\XINT_expr_gotnil{\expandafter\XINT_expr_getop\csname.= \endcsname}%
289 \def\XINT_expr_onliteral_` #1#2#3({\xint_c_xviii `{#2}}%
290 \catcode96 12 % `
291 \def\XINT_expr_startint #1%
292 {%
293   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
294 }%
295 \def\XINT_expr_scanint_a #1#2%
296   {\expandafter\XINT_expr_getop\csname.=#1%
297   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
298 \def\XINT_expr_gobz_a #1%
299   {\expandafter\XINT_expr_getop\csname.=%
300   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
301 \def\XINT_expr_startdec #1%
302   {\expandafter\XINT_expr_getop\csname.=%
303   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

10.23.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligeable. I don't think the doubled `\string` is a serious penalty.

Package *xintexpr* implementation

```

304 \def\XINT_expr_scanint_b #1%
305 {%
306   \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
307   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
308   \string#1\XINT_expr_scanint_d
309 }%
310 \def\XINT_expr_scanint_d #1%
311 {%
312   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
313 }%
314 \def\XINT_expr_scanint_endbycs#1#2\XINT_expr_scanint_d{\endcsname #1}%

```

With 1.2d the tacit multiplication in front of a variable name or function name is now done with a higher precedence, intermediate between the common one of $*$ and $/$ and the one of $^$. Thus $x/2y$ is like $x/(2y)$, but x^2y is like x^2*y and $2y!$ is not $(2y)!$ but $2*y!$.

Finally, 1.2d has moved away from the `_scan` macros all the business of the tacit multiplication in one unique place via `\XINT_expr_getop`. For this, the ending token is not first given to `\string` as was done earlier before handing over back control to `\XINT_expr_getop`. Earlier we had to identify the catcode `11 !` signaling a sub-expression here. With no `\string` applied we can do it in `\XINT_expr_getop`. As a corollary of this displacement, parsing of big numbers should be a tiny bit faster now.

Extended for 1.2l to ignore underscore character `_` if encountered within digits; so it can serve as separator for better readability.

```

315 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
316 {%
317   \if _#1\xint_dothis\XINT_expr_scanint_d\fi
318   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
319   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
320   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
321   \xint_orthat {\endcsname #1}%
322 }%
323 \def\XINT_expr_startdec_a .#1%
324 {%
325   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
326 }%
327 \def\XINT_expr_scandec_a #1%
328 {%
329   \if .#1\xint_dothis{\endcsname .}\fi
330   \xint_orthat {\XINT_expr_scandec_b 0.#1}%
331 }%
332 \def\XINT_expr_gobz_scanint_b #1%
333 {%
334   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
335   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
336   \string#1\XINT_expr_scanint_d
337 }%
338 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
339 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
340 {%
341   \if _#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
342   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
343   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi

```

```

344 \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
345 \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
346 \xint_orthat {0\endcsname #1}%
347 }%
348 \def\XINT_expr_gobz_scanint_d #1%
349 {%
350 \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
351 }%
352 \def\XINT_expr_gobz_startdec_a .#1%
353 {%
354 \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
355 }%
356 \def\XINT_expr_gobz_scandec_a #1%
357 {%
358 \if .#1\xint_dothis{0\endcsname.}\fi
359 \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
360 }%

```

10.23.2 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 ?

```

361 \def\XINT_expr_scandec_b #1.#2%
362 {%
363 \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
364 \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
365 \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
366 }%
367 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
368 \the\numexpr#3-\xint_c_i.#[#3]\endcsname #1}%
369 \def\XINT_expr_scandec_d #1.#2%
370 {%
371 \expandafter\XINT_expr_scandec_b
372 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
373 }%
374 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
375 {%
376 \if _#1\xint_dothis{\XINT_expr_scandec_d#3.}\fi
377 \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
378 \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
379 \xint_orthat {#[#3]\endcsname #1}%
380 }%
381 \def\XINT_expr_gobz_scandec_b #1.#2%
382 {%
383 \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
384 \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
385 \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
386 {\expandafter\XINT_expr_gobz_scandec_b}%

```

```

387   {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
388 }%
389 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
390 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
391 {%
392   \if   _#1\xint_dothis{\XINT_expr_gobz_scandec_b #4.}\fi
393   \if   e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
394   \if   E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
395   \xint_orthat {0[0]\endcsname #1}%
396 }%

```

10.23.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

```

397 \def\XINT_expr_scanexp_a #1#2%
398 {%
399   #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
400 }%
401 \def\XINT_expr_scanexp_b #1%
402 {%
403   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
404   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
405   \string#1\XINT_expr_scanexp_d
406 }%
407 \def\XINT_expr_scanexpr_endbycs#1#2\XINT_expr_scanexp_d []\endcsname #1}%
408 \def\XINT_expr_scanexp_d #1%
409 {%
410   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
411 }%
412 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
413 {%
414   \if   _#1\xint_dothis \XINT_expr_scanexp_d \fi
415   \if   +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
416   \if   -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
417   \xint_orthat {}]\endcsname #1}%
418 }%
419 \def\XINT_expr_scanexp_bb #1%
420 {%
421   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
422   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
423   \string#1\XINT_expr_scanexp_db
424 }%
425 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db []\endcsname #1}%
426 \def\XINT_expr_scanexp_db #1%
427 {%
428   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
429 }%
430 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db
431 {%
432   \if _#1\xint_dothis\XINT_expr_scanexp_d\fi
433   \xint_orthat{}]\endcsname #1}%

```

434 }%

10.23.4 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.

```

435 \def\XINT_expr_scanhex_I #1% #1="
436 {%
437   \expandafter\XINT_expr_getop\csname.=\expandafter
438   \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
439 }%
440 \def\XINT_expr_scanhexI_a #1%
441 {%
442   \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
443   \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
444   \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
445 }%
446 \def\XINT_expr_scanhexI_aa #1%
447 {%
448   \if\ifnum`#1>`/
449     \ifnum`#1>`9
450     \ifnum`#1>`@
451     \ifnum`#1>`F
452     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
453     \expandafter\XINT_expr_scanhexI_b
454   \else
455     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
456     \if.#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
457     \xint_orthat % gather what we got so far, leave catcode 12 #1 in stream
458     {\xint_afterfi {.>\endcsname\endcsname}}%
459   \fi
460   #1%
461 }%
462 \def\XINT_expr_scanhexI_b #1#2%
463 {%
464   #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
465 }%
466 \def\XINT_expr_scanhexI_bgob #1#2%
467 {%
468   \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
469 }%
470 \def\XINT_expr_scanhex_transition .#1%
471 {%
472   \expandafter.\expandafter.\expandafter
473   \XINT_expr_scanhexII_a\romannumeral`&&@#1%
474 }%
475 \def\XINT_expr_scanhexII_a #1%
476 {%

```

Package *xintexpr* implementation

```
477 \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
478 \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
479 \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
480 }%
481 \def\XINT_expr_scanhexII_aa #1%
482 {%
483 \if\ifnum`#1>`/
484 \ifnum`#1>`9
485 \ifnum`#1>`@
486 \ifnum`#1>`F
487 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
488 \expandafter\XINT_expr_scanhexII_b
489 \else
490 \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
491 \xint_orthat{\xint_afterfi {\endcsname\endcsname}}%
492 \fi
493 #1%
494 }%
495 \def\XINT_expr_scanhexII_b #1#2%
496 {%
497 #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
498 }%
499 \def\XINT_expr_scanhexII_bgob #1#2%
500 {%
501 \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
502 }%
```

10.23.5 Parsing names of functions and variables

```
503 \def\XINT_expr_scanfunc
504 {%
505 \expandafter\XINT_expr_func\romannumeral`&&@\XINT_expr_scanfunc_a
506 }%
507 \def\XINT_expr_scanfunc_a #1#2%
508 {%
509 \expandafter #1\romannumeral`&&@\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
510 }%
```

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.

```

511 \def\xINT_expr_scanfunc_b #1%
512 {%
513   \ifx !#1\xint_dothis{(_)\fi
514   \ifcat \relax#1\xint_dothis{(_)\fi
515   \if (#1\xint_dothis{\xint_firstoftwo{(`)\fi
516   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
517   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
518   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
519   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
520   \xint_orthat {(_}%
521   #1%
522 }%

```

Comments written 2015/11/12: earlier there was an `\ifcsname` test for checking if we had a variable in front of a `(`, for tacit multiplication for example in `x(y+z(x+w))` to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in `expr`, `iiexpr` or `floatexpr`. The `\xint_c_xviii` causes all fetching operations to stop and control is handed over to the routines which will be `expr`, `iiexpr` ou `floatexpr` specific, i.e. the `\XINT_{expr|iiexpr|flexpr}_op_{`|_}` which are invoked by the `until_<op>_b` macros earlier in the stream. Functions may exist for one but not the two other parsers. Variables are declared via one parser and usable in the others, but naturally `\xintiexpr` has its restrictions.

Thinking about this again I decided to treat a priori cases such as `x(...)` as functions, after having assigned to each variable a low-weight macro which will convert this into `_getop\.=<value of x>*(...)`. To activate that macro at the right time I could for this exploit the "onliteral" intercept, which is parser independent (1.2c).

This led to me necessarily to rewrite partially the `seq`, `add`, `mul`, `subs`, `iter` ... routines as now the variables fetch only one token. I think the thing is more efficient.

1.2c had `\def\xINT_expr_func #1(#2{\xint_c_xviii #2{#1}}`

In `\XINT_expr_func` the `#2` is `_` if `#1` must be a variable name, or `#2=``` if `#1` must be either a function name or possibly a variable name which will then have to be followed by tacit multiplication before the opening parenthesis.

The `\xint_c_xviii` is there because `_op_`` must know in which parser it works. Dispendious for `_`. Hence I modify for 1.2d.

```

523 \def\xINT_expr_func #1(#2{\if _#2\xint_dothis\xINT_expr_op__\fi
524   \xint_orthat{\xint_c_xviii #2}{#1}}%

```

10.24 `\XINT_expr_getop`: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit multiplication also in front of variable or functions names starting with a letter, not only a `@` or a `_` as was already the case. This is for `(x+y)z` situations. It also applies higher precedence in cases like `x/2y` or `x/2@`, or `x/2max(3,5)`, or `x/2\xintexpr 3\relax`.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if `\XINT_expr_getop` as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like `(a+b)/(c+d)(e+f)` will first multiply the last two parenthesized terms.

Package *xintexpr* implementation

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!

```
525 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
526 {%
527   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
528 }%
529 \catcode`* 11
530 \def\XINT_expr_getop_a #1#2%
531 {%
532   \ifx   \relax #2\xint_dothis\xint_firstofthree\fi
533   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
534   \ifnum\xint_c_ix<1\string#2 \xint_dothis\xint_secondofthree\fi
535   \if   _#2\xint_dothis      \xint_secondofthree\fi
536   \if   @#2\xint_dothis      \xint_secondofthree\fi
537   \if   (#2\xint_dothis      \xint_secondofthree\fi
538   \ifcat a#2\xint_dothis      \xint_secondofthree\fi
539   \xint_orthat \xint_thirdofthree
540   {\XINT_expr_foundend #1}%
541   {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
542   {\expandafter\XINT_expr_getop_b \string#2#1}%
543 }%
544 \catcode`* 12
545 \def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.
```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```
546 \def\XINT_expr_getop_b #1%
547 {%
548   \if '#1\xint_dothis{\XINT_expr_binopwrld }\fi
549   \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
550   \xint_orthat      {\XINT_expr_scanop_a #1}%
551 }%
552 \def\XINT_expr_binopwrld #1#2' {\expandafter\XINT_expr_foundop_a
553   \csname XINT_expr_itself_\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
554 \def\XINT_expr_scanop_a #1#2#3%
555   {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
556 \def\XINT_expr_scanop_b #1#2#3%
557 {%
558   \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
559   \ifcsname XINT_expr_itself_#1#3\endcsname
560   \xint_dothis
561     {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
562   \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
563 }%
564 \def\XINT_expr_scanop_c #1#2#3%
565 {%
566   \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
567 }%
```

```

568 \def\XINT_expr_scanop_d #1#2#3%
569 {%
570 \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
571 \ifcsname XINT_expr_itself_#1#3\endcsname
572 \xint_dothis
573     {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
574 \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
575 }%
576 \def\XINT_expr_foundop_a #1%
577 {%
578     \ifcsname XINT_expr_precedence_#1\endcsname
579         \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
580         \expandafter #1%
581     \else
582         \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
583     \fi
584 }%
585 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
586 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

10.25 Expansion spanning; opening and closing parentheses

Version 1.1 had a hack inside the until macros for handling the omit and abort in iterations over dummy variables. This has been removed by 1.2c, see the subsection where omit and abort are discussed.

```

587 \catcode`) 11
588 \def\XINT_tmpa #1#2#3#4%
589 {%
590     \def##1##1%
591     {%
592         \xint_UDsignfork
593             ##1{\expandafter#1\romannumeral`&&@#3}%
594             -{##2##1}%
595         \krof
596     }%
597     \def##2##1##2%
598     {%
599         \ifcase ##1\expandafter\XINT_expr_done
600         \or\xint_afterfi{\XINT_expr_extra_)
601             \expandafter #1\romannumeral`&&\XINT_expr_getop }%
602         \else
603         \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
604         \fi
605     }%
606 }%
607 \def\XINT_expr_extra_) {\xintError:removed }%
608 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
609     \expandafter\XINT_tmpa
610     \csname XINT_#1_until_end_a\endcsname\expandafter\endcsname
611     \csname XINT_#1_until_end_b\endcsname\expandafter\endcsname
612     \csname XINT_#1_op_-vi\endcsname
613     {#1}%

```



```

614 }%
615 \def\XINT_tmpa #1#2#3#4#5#6%
616 {%
617   \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
618   \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
619   \def #3##1{\xint_UDsignfork
620     ##1{\expandafter #3\romannumeral`&&#5}%
621     -{#4##1}%
622     \krof }%
623   \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_)
624     \or \csname XINT_#6_op_##2\expandafter\endcsname
625     \else
626     \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
627     \fi
628   }%
629 }%
630 \def\XINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%

```

We should be using `until_()` notation to stay synchronous with `until_+`, `until_*` etc..., but I found that `until_()` was more telling.

```

631 \catcode` ) 12
632 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
633   \expandafter\XINT_tmpa
634   \csname XINT_#1_op_(\expandafter\endcsname
635   \csname XINT_#1_oparen\expandafter\endcsname
636   \csname XINT_#1_until_)_a\expandafter\endcsname
637   \csname XINT_#1_until_)_b\expandafter\endcsname
638   \csname XINT_#1_op_-vi\endcsname
639   {#1}%
640 }%
641 \expandafter\let\csname XINT_expr_precedence_)\endcsname\xint_c_i

```

10.26 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, -, *, /, ^, **, //, /:, .., ..[,]..,][,][: ,:], and ++ operators

10.26.1	Square brackets for lists, the !? for omit and abort, and the ++ postfix construct	313
10.26.2	The , &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iiexpr operators	315
10.26.3	The]+,]-,]*,]/,]^, +[, -[, *[, /[, and ^[list operators	317
10.26.4	The 'and', 'or', 'xor', and 'mod' as infix operator words	319
10.26.5	The , &&, **, **[,]** operators as synonyms	319

10.26.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct

This is all very clever and only need setting some suitable precedence levels, if only I could understand what I did in 2014... just joking. Notice that `op_)` macros are defined here in the `\xintFor` loop.

There is some clever business going on here with the letter `a` for handling constructs such as `[3..5]*2` (I think...).

1.2c has replaced 1.1's private dealings with "`^C`" (which was done before dummy variables got implemented) by use of "`!?`". See discussion of omit and abort.

Package *xintexpr* implementation

```
642 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
643 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
644 \let\XINT_expr_precedence_a \xint_c_xviii
645 \let\XINT_expr_precedence_!? \xint_c_ii
646 \expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i
```

Comments added 2015/11/13 Here we have in particular the mechanism for post action on lists via `op_]` The `precedence_]` is the one of a closing parenthesis. We need the closing parenthesis to do its job, hence we can not define a `op_]+` operator for example, as we want to assign it the precedence of addition not the one of closing parenthesis. The trick I used in 1.1 was to let the `op_]` insert the letter `a`, this letter exceptionnally also being a legitimate operator, launch the `_getop` and let it find a `a*`, `a+`, `a/`, `a-`, `a^`, `a**` operator standing for `]*`, `]+`, `]/`, `]^`, `]**` postfix item by item list operator. I thought I had in mind an example to show that having defined `op_a` and `precedence_a` for the letter `a` caused a reduction in syntax for this letter, but it seems I am lacking now an example.

2015/11/18: for 1.2d I accelerate `\XINT_expr_op_]` to jump over the `\XINT_expr_getop_a` which now does tacit multiplications also in front of letters, for reasons of things like, $(x+y)z$, hence it must not see the "a". I could have used a `catcode12` a possibly, but anyhow jumping straight to `\XINT_expr_scanop_a` skips a few expansion steps (up to the potential price of less conceptual programming if I change things in the future.)

```
647 \catcode`. 11 \catcode`= 11 \catcode`+ 11
648 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
649   \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
650   \expandafter\let\csname XINT_#1_op_;\endcsname \space
651   \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
652   \expandafter\let\csname XINT_#1_op_a\endcsname \XINT_expr_getop
```

1.1 2014/10/29 did `\expandafter\.=+\xintiCeil` which transformed it into `\romannumeral0\xinticeil`, which seems a bit weird. This exploited the fact that dummy variables macros could back then pick braced material (which in the case at hand here ended being `{\romannumeral0\xinticeil...}` and were submitted to two expansions. The result of this was to provide a not value which got expanded only in the first loop of the `:_A` and following macros of `seq`, `iter`, `rseq`, etc...

Anyhow with 1.2c I have changed the implementation of dummy variables which now need to fetch a single locked token, which they do not expand.

The `\xintiCeil` appears a bit dispendious, but I need the starting value in a `\numexpr` compatible form in the iteration loops.

```
653   \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
654   {\expandafter\XINT_expr_foundend \expandafter
655     {\expandafter\.=+\csname .=\XINT:NEhook:one\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
656 }%
657 \catcode`. 12 \catcode`= 12 \catcode`+ 12
```

1.2d adds the `***` for tying via tacit multiplication, for example $x/2y$. Actually I don't need the `_itself` mechanism for `***`, only a precedence.

```
658 \catcode`& 12
659 \xintFor* #1 in {==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{.}{..}{.[]}{.[]..}%
660   {+[]{-[]}{*[]{/[]}{**[]}{^[]}{a+}{a-}{a*}{a/}{a**}{a^}%
661   [][][:]{:]}{!?}{++}{++)}%{***}}
662   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
663 \catcode`& 7
664 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii
```

10.26.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iexpr operators

1.2d needed some room between /, * and ^. Hence precedence for ^ is now at 9

```

665 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
666 {%
667   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iexpr
668   {% keep value, get next number and operator, then do until
669     \expandafter #2\expandafter ##1%
670     \romannumeral`&&\expandafter\XINT_expr_getnext }%
671   \def #2#1##2% \XINT_expr_until_<op>_a ou flexpr ou iexpr
672   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
673     -{#3##1##2}%
674     \krof }%
675   \def #3#1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iexpr
676   {% either execute next operation now, or first do next (possibly unary)
677     \ifnum ##2>#7%
678     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
679       \csname XINT_#8_op_##3\endcsname {##4}}%
680     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
681       \csname .=#9#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
682     \fi }%
683   \let #7#5%
684 }%
685 \def\XINT_expr_defbin_b #1#2#3#4#5%
686 {%
687   \expandafter\XINT_expr_defbin_c
688   \csname XINT_#1_op_#2\expandafter\endcsname
689   \csname XINT_#1_until_#2_a\expandafter\endcsname
690   \csname XINT_#1_until_#2_b\expandafter\endcsname
691   \csname XINT_#1_op_#4\expandafter\endcsname
692   \csname xint_c_#3\expandafter\endcsname
693   \csname #5\expandafter\endcsname
694   \csname XINT_expr_precedence_#2\endcsname {#1}\XINT:NEhook:two
695 }%
696 \XINT_expr_defbin_b {expr} | {iii}{vi} {xintOR}%
697 \XINT_expr_defbin_b {flexpr} | {iii}{vi} {xintOR}%
698 \XINT_expr_defbin_b {iexpr} | {iii}{vi} {xintOR}%
699 \XINT_expr_defbin_b {expr} & {iv}{vi} {xintAND}%
700 \XINT_expr_defbin_b {flexpr} & {iv}{vi} {xintAND}%
701 \XINT_expr_defbin_b {iexpr} & {iv}{vi} {xintAND}%
702 \XINT_expr_defbin_b {expr} {xor}{iii}{vi} {xintXOR}%
703 \XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
704 \XINT_expr_defbin_b {iexpr}{xor}{iii}{vi} {xintXOR}%
705 \XINT_expr_defbin_b {expr} < {v}{vi} {xintLt}%
706 \XINT_expr_defbin_b {flexpr} < {v}{vi} {xintLt}%
707 \XINT_expr_defbin_b {iexpr} < {v}{vi} {xintiiLt}%
708 \XINT_expr_defbin_b {expr} > {v}{vi} {xintGt}%
709 \XINT_expr_defbin_b {flexpr} > {v}{vi} {xintGt}%
710 \XINT_expr_defbin_b {iexpr} > {v}{vi} {xintiiGt}%
711 \XINT_expr_defbin_b {expr} = {v}{vi} {xintEq}%
712 \XINT_expr_defbin_b {flexpr} = {v}{vi} {xintEq}%

```

Package *xintexpr* implementation

```

713 \XINT_expr_defbin_b {iiexpr} = {v}{vi} {xintiiEq}%
714 \XINT_expr_defbin_b {expr} {<=} {v}{vi} {xintLtorEq}%
715 \XINT_expr_defbin_b {flexpr}{<=} {v}{vi} {xintLtorEq}%
716 \XINT_expr_defbin_b {iiexpr}{<=} {v}{vi} {xintiiLtorEq}%
717 \XINT_expr_defbin_b {expr} {>=} {v}{vi} {xintGtorEq}%
718 \XINT_expr_defbin_b {flexpr}{>=} {v}{vi} {xintGtorEq}%
719 \XINT_expr_defbin_b {iiexpr}{>=} {v}{vi} {xintiiGtorEq}%
720 \XINT_expr_defbin_b {expr} {!=} {v}{vi} {xintNotEq}%
721 \XINT_expr_defbin_b {flexpr}{!=} {v}{vi} {xintNotEq}%
722 \XINT_expr_defbin_b {iiexpr}{!=} {v}{vi} {xintiiNotEq}%
723 \XINT_expr_defbin_b {expr} {//} {vii}{vii}{xintDivFloor}% CHANGED IN 1.2p!
724 \XINT_expr_defbin_b {flexpr}{//} {vii}{vii}{XINTinFloatDivFloor}% "
725 \XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivFloor}% "
726 \XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}% "
727 \XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{XINTinFloatMod}% "
728 \XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}% "
729 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
730 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
731 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
732 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%
733 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
734 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
735 \XINT_expr_defbin_b {expr} * {vii}{vii}{xintMul}%
736 \XINT_expr_defbin_b {flexpr} * {vii}{vii}{XINTinFloatMul}%
737 \XINT_expr_defbin_b {iiexpr} * {vii}{vii}{xintiiMul}%
738 \XINT_expr_defbin_b {expr} / {vii}{vii}{xintDiv}%
739 \XINT_expr_defbin_b {flexpr} / {vii}{vii}{XINTinFloatDiv}%
740 \XINT_expr_defbin_b {iiexpr} / {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
741 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
742 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
743 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
744 \XINT_expr_defbin_b {expr} {..[]}{iii}{vi} {xintSeqA::csv}%
745 \XINT_expr_defbin_b {flexpr}{..[]}{iii}{vi} {XINTinFloatSeqA::csv}%
746 \XINT_expr_defbin_b {iiexpr}{..[]}{iii}{vi} {xintiiSeqA::csv}%
747 \def\XINT_expr_defbin_b #1#2#3#4#5%
748 {%
749 \expandafter\XINT_expr_defbin_c
750 \csname XINT_#1_op_#2\expandafter\endcsname
751 \csname XINT_#1_until_#2_a\expandafter\endcsname
752 \csname XINT_#1_until_#2_b\expandafter\endcsname
753 \csname XINT_#1_op_#4\expandafter\endcsname
754 \csname xint_c_#3\expandafter\endcsname
755 \csname #5\expandafter\endcsname
756 \csname XINT_expr_precedence_#2\endcsname {#1}{}%
757 }%
758 \XINT_expr_defbin_b {expr} {..} {iii}{vi} {xintSeq::csv}%
759 \XINT_expr_defbin_b {flexpr}{..} {iii}{vi} {xintSeq::csv}%
760 \XINT_expr_defbin_b {iiexpr}{..} {iii}{vi} {xintiiSeq::csv}%
761 \XINT_expr_defbin_b {expr} {[]}..{iii}{vi} {xintSeqB::csv}%
762 \XINT_expr_defbin_b {flexpr}{[]}..{iii}{vi} {XINTinFloatSeqB::csv}%
763 \XINT_expr_defbin_b {iiexpr}{[]}..{iii}{vi} {xintiiSeqB::csv}%

```

10.26.3 The `+`, `-`, `*`, `/`, `^`, `+`, `-`, `*`, `/`, and `^` list operators

`\XINT_expr_binop_inline_b` This handles acting on comma separated values (no need to bother about spaces in this context; expansion in a `\csname...\endcsname`).

```

764 \def\XINT_expr_binop_inline#1%
765   {\XINT_expr_binop_inline_a{\expandafter\XINT:NEhook:two\expandafter#1}}%
766 \def\XINT_expr_binop_inline_a
767   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b }%
768 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
769 \def\XINT_expr_binop_inline_c #1{%
770   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
771   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
772   \xint_orthat\XINT_expr_binop_inline_d #1}%
773 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
774 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
775 \def\XINT_expr_binop_inline_end #1,#2{%
776 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
777 {%
778   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
779   {% keep value, get next number and operator, then do until
780     \expandafter #2\expandafter ##1%
781     \romannumeral`&&\expandafter\XINT_expr_getnext }%
782   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
783   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4%
784     -{#3##1##2}}%
785     \krof }%
786   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
787   {% either execute next operation now, or first do next (possibly unary)
788     \ifnum ##2>#7%
789     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
790       \csname XINT_#8_op_##3\endcsname {##4}}%
791     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
792       \csname .=\expandafter\XINT_expr_binop_inline\expandafter
793         {\expandafter#6\expandafter\xint_exchangetwo_keepbraces\expandafter
794           {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
795         \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname }%
796     \fi }%
797   \let #7#5%
798 }%
799 \def\XINT_expr_deflistopr_b #1#2#3#4%
800 {%
801   \expandafter\XINT_expr_deflistopr_c
802   \csname XINT_#1_op_#2\expandafter\endcsname
803   \csname XINT_#1_until_#2_a\expandafter\endcsname
804   \csname XINT_#1_until_#2_b\expandafter\endcsname
805   \csname XINT_#1_op_#3\expandafter\endcsname
806   \csname xint_c_#3\expandafter\endcsname
807   \csname #4\expandafter\endcsname
808   \csname XINT_expr_precedence_#2\endcsname {#1}%
809 }%

```

This is for `[x..y]*z` syntax etc.... Attention that with 1.2d, precedence level of `^` raised to `ix` to make room for `***`.

```

810 \XINT_expr_deflistopr_b {expr} {a+}{vi} {xintAdd}%
811 \XINT_expr_deflistopr_b {expr} {a-}{vi} {xintSub}%
812 \XINT_expr_deflistopr_b {expr} {a*}{vii} {xintMul}%
813 \XINT_expr_deflistopr_b {expr} {a/}{vii} {xintDiv}%
814 \XINT_expr_deflistopr_b {expr} {a^}{ix} {xintPow}%
815 \XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
816 \XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
817 \XINT_expr_deflistopr_b {iiexpr}{a*}{vii} {xintiiMul}%
818 \XINT_expr_deflistopr_b {iiexpr}{a/}{vii} {xintiiDivRound}%
819 \XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
820 \XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
821 \XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
822 \XINT_expr_deflistopr_b {flexpr}{a*}{vii} {XINTinFloatMul}%
823 \XINT_expr_deflistopr_b {flexpr}{a/}{vii} {XINTinFloatDiv}%
824 \XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%
825 \def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
826 {%
827   \def #1#1{\expandafter#2\expandafter##1\romannumeral`&&@%
828     \expandafter #3\romannumeral`&&\XINT_expr_getnext }%
829   \def #2##1##2##3##4%
830   {% either execute next operation now, or first do next (possibly unary)
831     \ifnum ##2>#6%
832     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
833       \csname XINT_#7_op_##3\endcsname {##4}}%
834     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
835       \csname .=\expandafter\XINT_expr_binop_inline\expandafter
836         {\expandafter#5\expandafter
837           {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
838         \romannumeral`&&\XINT_expr_unlock ##4,^,\endcsname }%
839     \fi }%
840   \let #6#4%
841 }%
842 \def\XINT_expr_deflistopl_b #1#2#3#4%
843 {%
844   \expandafter\XINT_expr_deflistopl_c
845   \csname XINT_#1_op_#2\expandafter\endcsname
846   \csname XINT_#1_until_#2\expandafter\endcsname
847   \csname XINT_#1_until_)_a\expandafter\endcsname
848   \csname xint_c_#3\expandafter\endcsname
849   \csname #4\expandafter\endcsname
850   \csname XINT_expr_precedence_#2\endcsname {#1}%
851 }%

```

This is for $z*[x..y]$ syntax etc...

```

852 \XINT_expr_deflistopl_b {expr} {+[]}{vi} {xintAdd}%
853 \XINT_expr_deflistopl_b {expr} {-[]}{vi} {xintSub}%
854 \XINT_expr_deflistopl_b {expr} {*[]}{vii} {xintMul}%
855 \XINT_expr_deflistopl_b {expr} {/[]}{vii} {xintDiv}%
856 \XINT_expr_deflistopl_b {expr} {^[]}{ix} {xintPow}%
857 \XINT_expr_deflistopl_b {iiexpr}{+[]}{vi} {xintiiAdd}%
858 \XINT_expr_deflistopl_b {iiexpr}{-[]}{vi} {xintiiSub}%
859 \XINT_expr_deflistopl_b {iiexpr}{*[]}{vii} {xintiiMul}%

```

```

860 \XINT_expr_deflistopl_b {iiexpr}{/[]{}{vii}{xintiiDivRound}}%
861 \XINT_expr_deflistopl_b {iiexpr}{^[]{}{ix} {xintiiPow}}%
862 \XINT_expr_deflistopl_b {fexpr}{+[]{}{vi} {XINTinFloatAdd}}%
863 \XINT_expr_deflistopl_b {fexpr}{-[]{}{vi} {XINTinFloatSub}}%
864 \XINT_expr_deflistopl_b {fexpr}{*[]{}{vii}{XINTinFloatMul}}%
865 \XINT_expr_deflistopl_b {fexpr}{/[]{}{vii}{XINTinFloatDiv}}%
866 \XINT_expr_deflistopl_b {fexpr}{^[]{}{ix} {XINTinFloatPowerH}}%

```

10.26.4 The 'and', 'or', 'xor', and 'mod' as infix operator words

```

867 \xintFor #1 in {and,or,xor,mod} \do {%
868   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
869 \expandafter\let\csname XINT_expr_precedence_and\endcsname
870   \csname XINT_expr_precedence_&\endcsname
871 \expandafter\let\csname XINT_expr_precedence_or\endcsname
872   \csname XINT_expr_precedence_|\endcsname
873 \expandafter\let\csname XINT_expr_precedence_mod\endcsname
874   \csname XINT_expr_precedence_/\endcsname
875 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
876   \expandafter\let\csname XINT_#1_op_and\endcsname
877     \csname XINT_#1_op_&\endcsname
878   \expandafter\let\csname XINT_#1_op_or\endcsname
879     \csname XINT_#1_op_|\endcsname
880   \expandafter\let\csname XINT_#1_op_mod\endcsname
881     \csname XINT_#1_op_/\endcsname
882 }%

```

10.26.5 The ||, &&, **, **[,]** operators as synonyms

```

883 \expandafter\let\csname XINT_expr_precedence_==\endcsname
884   \csname XINT_expr_precedence_=\endcsname
885 \expandafter\let\csname XINT_expr_precedence_&string&\endcsname
886   \csname XINT_expr_precedence_&\endcsname
887 \expandafter\let\csname XINT_expr_precedence_||\endcsname
888   \csname XINT_expr_precedence_|\endcsname
889 \expandafter\let\csname XINT_expr_precedence_**\endcsname
890   \csname XINT_expr_precedence_^\endcsname
891 \expandafter\let\csname XINT_expr_precedence_a**\endcsname
892   \csname XINT_expr_precedence_a^\endcsname
893 \expandafter\let\csname XINT_expr_precedence_**[\endcsname
894   \csname XINT_expr_precedence_^[endcsname
895 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
896   \expandafter\let\csname XINT_#1_op_==\endcsname
897     \csname XINT_#1_op_=\endcsname
898   \expandafter\let\csname XINT_#1_op_&string&\endcsname
899     \csname XINT_#1_op_&\endcsname
900   \expandafter\let\csname XINT_#1_op_||\endcsname
901     \csname XINT_#1_op_|\endcsname
902   \expandafter\let\csname XINT_#1_op_**\endcsname
903     \csname XINT_#1_op_^\endcsname
904   \expandafter\let\csname XINT_#1_op_a**\endcsname
905     \csname XINT_#1_op_a^\endcsname
906   \expandafter\let\csname XINT_#1_op_**[\endcsname
907     \csname XINT_#1_op_^[endcsname

```

908 }%

10.27 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

10.27.1	<code>\xintListSel:x:csv</code>	322
10.27.2	<code>\xintListSel:f:csv</code>	323
10.27.3	<code>\xintKeep:x:csv</code>	324
10.27.4	<code>\xintKeep:f:csv</code>	325
10.27.5	<code>\xintTrim:f:csv</code>	325
10.27.6	<code>\xintNthEltPy:f:csv</code>	325
10.27.7	<code>\xintLength:f:csv</code>	325
10.27.8	<code>\xintReverse:f:csv</code>	325

Python slicing was first implemented for 1.1 (27 octobre 2014). But it used `\xintCSVtoList` and `\xintListWithSep{,}` to convert back and forth to token lists for use of `\xintKeep`, `\xintTrim`, `\xintNthElt`. Not very efficient! Also `[list][a:b]` was Python like but not `[list][N]` which counted items starting at one, and returned the length for $N=0$.

Release 1.2g changed this so `[list][N]` now counts starting at zero and `len(list)` computes the number of items. Also 1.2g had its own f-expandable macros handling directly the comma separated lists. They are located into `xinttools.sty`.

1.2j improved the `xinttools.sty` macros and furthermore it made the Python slicing in expressions a bit more efficient still by exploiting in some cases that expansion happens in `\csname...\endcsname` and does not have to be f-expandable. But the f-expandable variants must be kept for use by `\xintNewExpr` and `\xintdeffunc`.

```

909 \def\XINT_tmpa #1#2#3#4#5#6%
910 {%
911   \def #1##1% \XINT_expr_op_][
912   {%
913     \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
914   }%
915   \def #2##1##2% \XINT_expr_until_][_a
916   {\xint_UDsignfork
917     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
918     -{#3##1##2}%
919   \krof }%
920   \def #3##1##2##3##4% \XINT_expr_until_][_b
921   {%
922     \ifnum ##2>#5%
923       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
924         \csname XINT_#6_op_##3\endcsname {##4}}%
925     \else
926       \xint_afterfi
927       {\expandafter ##2\expandafter ##3\csname
928         .=\expandafter\xintListSel:x:csv % will be \xintListSel:f:csv in \xintNewExpr output
929         \romannumeral`&&@\XINT_expr_unlock ##4;% selector
930         \XINT_expr_unlock ##1;\endcsname % unlock already pre-positioned for \xintNewExpr
931       }%
932     \fi
933   }%
934   \let #5\xint_c_ii
935 }%
936 \xintFor #1 in {expr,flexpr,iiexpr} \do {%

```



```

937 \expandafter\XINT_tmpa
938   \csname XINT_#1_op_][\expandafter\endcsname
939   \csname XINT_#1_until_][_a\expandafter\endcsname
940   \csname XINT_#1_until_][_b\expandafter\endcsname
941   \csname XINT_#1_op_-vi\expandafter\endcsname
942   \csname XINT_expr_precedence_][\endcsname {#1}%
943 }%
944 \def\XINT_tmpa #1#2#3#4#5#6%
945 {%
946   \def #1##1% \XINT_expr_op_:
947   {%
948     \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
949   }%
950   \def #2##1##2% \XINT_expr_until:_a
951   {\xint_UDsignfork
952     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
953     -{##3##1##2}%
954   \krof }%
955   \def #3##1##2##3##4% \XINT_expr_until:_b
956   {%
957     \ifnum ##2>#5%
958       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
959         \csname XINT_#6_op_##3\endcsname {##4}}%
960     \else
961       \xint_afterfi
962       {\expandafter ##2\expandafter ##3\csname
963         .:=\XINT:NEhook:one\xintNum{\XINT_expr_unlock ##1};%
964         \XINT:NEhook:one\xintNum{\XINT_expr_unlock ##4}%
965         \endcsname
966       }%
967     \fi
968   }%
969   \let #5\xint_c_iii
970 }%
971 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
972 \expandafter\XINT_tmpa
973   \csname XINT_#1_op_:\expandafter\endcsname
974   \csname XINT_#1_until:_a\expandafter\endcsname
975   \csname XINT_#1_until:_b\expandafter\endcsname
976   \csname XINT_#1_op_-vi\expandafter\endcsname
977   \csname XINT_expr_precedence_:\endcsname {#1}%
978 }%
979 \catcode`[ 11 \catcode`] 11
980 \let\XINT_expr_precedence_:\xint_c_iii
981 \def\XINT_expr_op_:] #1%
982 {%
983   \expandafter\xint_c_i\expandafter )%
984   \csname .:=\XINT:NEhook:one\xintNum{\XINT_expr_unlock #1}\endcsname
985 }%
986 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
987 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
988 \let\XINT_expr_precedence_][: \xint_c_iii

```

At the end of the replacement text of `\XINT_expr_op_][:`, the `:` after index 0 must be catcode 12, else will be mistaken for the start of variable by expression parser (as `<digits><variable>` is allowed by the syntax and does tacit multiplication).

```
989 \edef\XINT_expr_op_][: #1{\xint_c_ii\noexpand\XINT_expr_itself_][#10\string :}%
990 \let\XINT_flexpr_op_][: \XINT_expr_op_][:
991 \let\XINT_iiexpr_op_][: \XINT_expr_op_][:
992 \catcode`[ 12 \catcode`] 12
```

10.27.1 `\xintListSel:x:csv`

1.2j. Because there is `\xintKeep:x:csv` which is faster than `\xintKeep:f:csv`.

```
993 \def\xintListSel:x:csv #1%
994 {%
995   \if ]\noexpand#1\xint_dothis\XINT_listsel:_s\fi
996   \if : \noexpand#1\xint_dothis\XINT_listxsel:_:\fi
997   \xint_orthat {\XINT_listsel:_nth #1}%
998 }%
999 \def\XINT_listsel:_s #1#2;#3;%
1000 {%
1001   \if-#1\expandafter\xintKeep:f:csv\else\expandafter\xintTrim:f:csv\fi
1002   {#1#2}{#3}%
1003 }%
1004 \def\XINT_listsel:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%
```

`\XINT_listsel:_nth` and `\XINT_listsel:_s` located in `\xintListSel:f:csv`.

```
1005 \def\XINT_listxsel:_: #1#2;#3#4;%
1006 {%
1007   \xint_UDsignsfork
1008     #1#3\XINT_listxsel:_N:N
1009     #1-\XINT_listxsel:_N:P
1010     -#3\XINT_listxsel:_P:N
1011     --\XINT_listxsel:_P:P
1012   \krof #1#2;#3#4;%
1013 }%
1014 \def\XINT_listxsel:_P:P #1;#2;#3;%
1015 {%
1016   \unless\ifnum #1<#2 \expandafter\xint_gobble_iii\fi
1017   \xintKeep:x:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1018 }%
1019 \def\XINT_listxsel:_N:N #1;#2;#3;%
1020 {%
1021   \expandafter\XINT_listxsel:_N:N_a
1022   \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength:f:csv{#3};#3;%
1023 }%
1024 \def\XINT_listxsel:_N:N_a #1;#2;#3;%
1025 {%
1026   \unless\ifnum #1>\xint_c_ \expandafter\xint_gobble_iii\fi
1027   \xintKeep:x:csv{#1}{\xintTrim:f:csv{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1028 }%
1029 \def\XINT_listxsel:_N:P #1;#2;#3;{\expandafter\XINT_listxsel:_N:P_a
```

```

1030             \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
1031 \def\xINT_listxsel:_N:P_a #1#2;%
1032   {\if -#1\expandafter\xINT_listxsel:_O:P\fi\xINT_listxsel:_P:P #1#2;}%
1033 \def\xINT_listxsel:_O:P\xINT_listxsel:_P:P #1;{\XINT_listxsel:_P:P 0;}%
1034 \def\xINT_listxsel:_P:N #1;#2;#3;{\expandafter\xINT_listxsel:_P:N_a
1035   \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
1036 \def\xINT_listxsel:_P:N_a #1#2;#3;%
1037   {\if -#1\expandafter\xINT_listxsel:_P:O\fi\xINT_listxsel:_P:P #3;#1#2;}%
1038 \def\xINT_listxsel:_P:O\xINT_listxsel:_P:P #1;#2;{\XINT_listxsel:_P:P #1;0;}%

```

10.27.2 `\xintListSel:f:csv`

1.2g. Since 1.2j this is needed only for `\xintNewExpr` and user defined functions. Some extras compared to `\xintListSel:x:csv` because things may not yet have been expanded in the `\xintNewExpr` context.

```

1039 \def\xintListSel:f:csv #1%
1040 {%
1041   \if ]\noexpand#1\xint_dothis{\expandafter\xINT_listsel:_s\romannumeral`&&@\}\fi
1042   \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
1043   \xint_orthat {\XINT_listsel:_nth #1}%
1044 }%
1045 \def\xINT_listsel:_: #1;#2;%
1046 {%
1047   \expandafter\xINT_listsel:_:a
1048   \the\numexpr #1\expandafter;\the\numexpr #2\expandafter;\romannumeral`&&@%
1049 }%
1050 \def\xINT_listsel:_:a #1#2;#3#4;%
1051 {%
1052   \xint_UDsignsfork
1053     #1#3\xINT_listsel:_N:N
1054     #1-\XINT_listsel:_N:P
1055     -#3\xINT_listsel:_P:N
1056     --\XINT_listsel:_P:P
1057   \krof #1#2;#3#4;%
1058 }%
1059 \def\xINT_listsel:_P:P #1;#2;#3;%
1060 {%
1061   \unless\ifnum #1<#2 \xint_afterfi{\expandafter\space\xint_gobble_iii}\fi
1062   \xintKeep:f:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1063 }%
1064 \def\xINT_listsel:_N:N #1;#2;#3;%
1065 {%
1066   \unless\ifnum #1<#2 \expandafter\xINT_listsel:_N:N_abort\fi
1067   \expandafter\xINT_listsel:_N:N_a
1068   \the\numexpr#1+\xintLength:f:csv{#3}\expandafter;\the\numexpr#2-#1;#3;%
1069 }%
1070 \def\xINT_listsel:_N:N_abort #1;#2;#3;{ }%
1071 \def\xINT_listsel:_N:N_a #1;#2;#3;%
1072 {%
1073   \xintKeep:f:csv{#2}{\xintTrim:f:csv{\ifnum#1<\xint_c_\xint_c_\else#1\fi}{#3}}%
1074 }%
1075 \def\xINT_listsel:_N:P #1;#2;#3;{\expandafter\xINT_listsel:_N:P_a

```

```

1076 \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
1077 \def\xINT_listsel:_N:P_a #1#2;%
1078   {\if -#1\expandafter\xINT_listsel:_O:P\fi\xINT_listsel:_P:P #1#2;}%
1079 \def\xINT_listsel:_O:P\xINT_listsel:_P:P #1;{\XINT_listsel:_P:P 0;}%
1080 \def\xINT_listsel:_P:N #1;#2;#3;{\expandafter\xINT_listsel:_P:N_a
1081   \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
1082 \def\xINT_listsel:_P:N_a #1#2;#3;%
1083   {\if -#1\expandafter\xINT_listsel:_P:O\fi\xINT_listsel:_P:P #3;#1#2;}%
1084 \def\xINT_listsel:_P:O\xINT_listsel:_P:P #1;#2;{\XINT_listsel:_P:P #1;0;}%

```

10.27.3 \xintKeep:x:csv

1.2j. This macro is used only with positive first argument.

```

1085 \def\xintKeep:x:csv #1#2%
1086 {%
1087   \expandafter\xint_gobble_i
1088   \romannumeral0\expandafter\xINT_keep:x:csv_pos
1089   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1090 }%
1091 \def\xINT_keep:x:csv_pos #1.#2%
1092 {%
1093   \expandafter\xINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1094   #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1095   \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1096 }%
1097 \def\xINT_keep:x:csv_loop #1%
1098 {%
1099   \xint_gob_til_minus#1\xINT_keep:x:csv_finish-%
1100   \XINT_keep:x:csv_loop_pickeight #1%
1101 }%
1102 \def\xINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1103 {%
1104   ,#2,#3,#4,#5,#6,#7,#8,#9%
1105   \expandafter\xINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1106 }%
1107 \def\xINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1108 {%
1109   \csname XINT_keep:x:csv_finish#1\endcsname
1110 }%
1111 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
1112   #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1113 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1114   #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
1115 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1116   #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
1117 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1118   #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
1119 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1120   #1,#2,#3,{,#1,#2,#3\xint_Bye}%
1121 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1122   #1,#2,{,#1,#2\xint_Bye}%
1123 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname

```

```
1124 #1,{,#1\xint_Bye}%
1125 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye
```

10.27.4 `\xintKeep:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.27.5 `\xintTrim:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.27.6 `\xintNthEltPy:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.27.7 `\xintLength:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.27.8 `\xintReverse:f:csv`

1.2g, code in `xinttools.sty`.

10.28 Macros for a..b list generation

10.28.1	<code>\xintSeq::csv</code>	325
10.28.2	<code>\xintiSeq::csv</code>	326

Ne produit que des listes d'entiers inférieurs à la borne de TeX ! mais sous la forme $N/1[0]$ en ce qui concerne `\xintSeq::csv`.

10.28.1 `\xintSeq::csv`

Commence par remplacer `a` par `ceil(a)` et `b` par `floor(b)` et renvoie ensuite les entiers entre les deux, possiblement en décroissant, et extrémités comprises. Si $a=b$ est non entier en obtient donc `ceil(a)` et `floor(a)`. Ne renvoie jamais une liste vide.

Note: le `a..b` dans `\xintfloatexpr` utilise cette routine.

```
1126 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
1127 \def\xintseq::csv #1#2%
1128 {%
1129   \expandafter\XINT_seq::csv\expandafter
1130   {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
1131   {\the\numexpr \xintiFloor{#2}}%
1132}%
1133 \def\XINT_seq::csv #1#2%
1134 {%
1135   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1136   \expandafter\XINT_seq::csv_z
1137   \or
```

```

1138     \expandafter\XINT_seq::csv_p
1139   \else
1140     \expandafter\XINT_seq::csv_n
1141   \fi
1142   {#2}{#1}%
1143 }%
1144 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1145 \def\XINT_seq::csv_p #1#2%
1146 {%
1147   \ifnum #1>#2
1148     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1149   \else
1150     \expandafter\XINT_seq::csv_e
1151   \fi
1152   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1153 }%
1154 \def\XINT_seq::csv_n #1#2%
1155 {%
1156   \ifnum #1<#2
1157     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1158   \else
1159     \expandafter\XINT_seq::csv_e
1160   \fi
1161   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1162 }%
1163 \def\XINT_seq::csv_e #1,{ }%

```

10.28.2 `\xintiiseq::csv`

```

1164 \def\xintiiseq::csv {\romannumeral0\xintiiseq::csv }%
1165 \def\xintiiseq::csv #1#2%
1166 {%
1167   \expandafter\XINT_iiseq::csv\expandafter
1168     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1169 }%
1170 \def\XINT_iiseq::csv #1#2%
1171 {%
1172   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1173   \expandafter\XINT_iiseq::csv_z
1174   \or
1175   \expandafter\XINT_iiseq::csv_p
1176   \else
1177   \expandafter\XINT_iiseq::csv_n
1178   \fi
1179   {#2}{#1}%
1180 }%
1181 \def\XINT_iiseq::csv_z #1#2{ #1}%
1182 \def\XINT_iiseq::csv_p #1#2%
1183 {%
1184   \ifnum #1>#2
1185     \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1186   \else
1187     \expandafter\XINT_seq::csv_e

```

```

1188 \fi
1189 \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1190 }%
1191 \def\xINT_iiseq::csv_n #1#2%
1192 {%
1193 \ifnum #1<#2
1194 \expandafter\expandafter\expandafter\xINT_iiseq::csv_n
1195 \else
1196 \expandafter\xINT_seq::csv_e
1197 \fi
1198 \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1199 }%
1200 \def\xINT_seq::csv_e #1,{ }%

```

10.29 Macros for a..[d].b list generation

10.29.1	<code>\xintSeqA::csv</code> , <code>\xintiiSeqA::csv</code> , <code>\XINTinFloatSeqA::csv</code>	327
10.29.2	<code>\xintSeqB::csv</code>	327
10.29.3	<code>\xintiiSeqB::csv</code>	328
10.29.4	<code>\XINTinFloatSeqB::csv</code>	328

Contrarily to `a..b` which is limited to small integers, this works with `a`, `b`, and `d` (big) fractions. It will produce a «nil» list, if `a>b` and `d<0` or `a<b` and `d>0`.

10.29.1 `\xintSeqA::csv`, `\xintiiSeqA::csv`, `\XINTinFloatSeqA::csv`

```

1201 \def\xintSeqA::csv #1%
1202 {\expandafter\xINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%
1203 \def\xINT_seqa::csv #1#2{\expandafter\xINT_seqa::csv_a \romannumeral0\xintra {#2};#1;}%
1204 \def\xintiiSeqA::csv #1{\expandafter\xINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
1205 \def\xINT_iiseqa::csv #1#2{\expandafter\xINT_seqa::csv_a\romannumeral`&&@#2;#1;}%
1206 \def\xINTinFloatSeqA::csv #1{\expandafter\xINT_flseqa::csv\expandafter
1207 {\romannumeral0\xINTinfloat [\XINTdigits]{#1}}}%
1208 \def\xINT_flseqa::csv #1#2%
1209 {\expandafter\xINT_seqa::csv_a\romannumeral0\xINTinfloat [\XINTdigits]{#2};#1;}%
1210 \def\xINT_seqa::csv_a #1{\xint_UDzerominusfork
1211 #1-{z}%
1212 0#1{n}%
1213 0-{p}%
1214 \krof #1}%

```

10.29.2 `\xintSeqB::csv`

With one year late documentation, let's just say, the `#1` is `\XINT_expr_unlock\.=Ua;b`; with `U=z` or `n` or `p`, `a=step`, `b=start`.

```

1215 \def\xintSeqB::csv #1#2%
1216 {\expandafter\xINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1217 \def\xINT_seqb::csv #1#2{\expandafter\xINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1218 \def\xINT_seqb::csv_a #1#2;#3;#4!{\expandafter\xINT_expr_seq_empty?
1219 \romannumeral0\cename XINT_seqb::csv_#1\endcename {#3}{#4}{#2}}%
1220 \def\xINT_seqb::csv_p #1#2#3%
1221 {%
1222 \xintifCmp {#1}{#2}{, #1\expandafter\xINT_seqb::csv_p\expandafter}%
1223 {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

`\romannumeral0` stopped by `\endcsname`, `XINT_expr_seq_empty?` constructs "nil".

```

1224   {\romannumeral0\xintadd {#3}{#1}{#2}{#3}%
1225 }%
1226 \def\XINT_seqb::csv_n #1#2#3%
1227 {%
1228   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1229   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1230   {\romannumeral0\xintadd {#3}{#1}{#2}{#3}%
1231 }%
1232 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

10.29.3 `\xintiiSeqB::csv`

```

1233 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1234 \def\XINT_iiseqb::csv #1#2#3#4%
1235   {\expandafter\XINT_iiseqb::csv_a
1236   \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1237   \romannumeral`&&\XINT_expr_unlock #4!}%
1238 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1239   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1240 \def\XINT_iiseqb::csv_p #1#2#3%
1241 {%
1242   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1243   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1244   {\romannumeral0\xintiiadd {#3}{#1}{#2}{#3}%
1245 }%
1246 \def\XINT_iiseqb::csv_n #1#2#3%
1247 {%
1248   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1249   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1250   {\romannumeral0\xintiiadd {#3}{#1}{#2}{#3}%
1251 }%
1252 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

10.29.4 `\XINTinFloatSeqB::csv`

```

1253 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1254   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1255 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&#2#1!}%
1256 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1257   \romannumeral`&&\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1258 \def\XINT_flseqb::csv_p #1#2#3%
1259 {%
1260   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1261   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1262   {\romannumeral0\XINTinfloatadd {#3}{#1}{#2}{#3}%
1263 }%
1264 \def\XINT_flseqb::csv_n #1#2#3%
1265 {%
1266   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1267   {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1268   {\romannumeral0\XINTinfloatadd {#3}{#1}{#2}{#3}%
1269 }%

```



```
1270 \def\XINT_flseqb::csv_z #1#2#3{,#1}%
```

10.30 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```
1271 \def\XINT_tmpa #1#2#3#4#5#6%
1272 {%
1273   \def #1##1% \XINT_expr_op_,
1274   {%
1275     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
1276   }%
1277   \def #2##1##2% \XINT_expr_until_,_a
1278   {\xint_UDsignfork
1279     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
1280     -{#3##1##2}%
1281   \krof }%
1282   \def #3##1##2##3##4% \XINT_expr_until_,_b
1283   {%
1284     \ifnum ##2>\xint_c_ii
1285       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
1286                     \csname XINT_#6_op_##3\endcsname {##4}}%
1287     \else
1288       \xint_afterfi
1289       {\expandafter ##2\expandafter ##3%
1290        \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1291     \fi
1292   }%
1293   \let #5\xint_c_ii
1294 }%
1295 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1296 \expandafter\XINT_tmpa
1297   \csname XINT_#1_op_,\expandafter\endcsname
1298   \csname XINT_#1_until_,_a\expandafter\endcsname
1299   \csname XINT_#1_until_,_b\expandafter\endcsname
1300   \csname XINT_#1_op_-vi\expandafter\endcsname
1301   \csname XINT_expr_precedence_,\endcsname {#1}%
1302 }%
```

10.31 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```
1303 \def\XINT_tmpa #1#2#3%
1304 {%
1305   \expandafter\XINT_tmpb
1306   \csname XINT_#1_op_-#3\expandafter\endcsname
1307   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1308   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1309   \csname xint_c_#3\endcsname {#1}#2%
1310 }%
1311 \def\XINT_tmpb #1#2#3#4#5#6%
1312 {%
```

```

1313 \def #1% \XINT_expr_op_-<level>
1314 {% get next number+operator then switch to _until macro
1315 \expandafter #2\romannumeral`&&\XINT_expr_getnext
1316 }%
1317 \def #2##1% \XINT_expr_until_-<l>_a
1318 {\xint_UDsignfork
1319 ##1{\expandafter #2\romannumeral`&&#1}%
1320 -{#3##1}%
1321 \krof }%
1322 \def #3##1##2##3% \XINT_expr_until_-<l>_b
1323 {% _until tests precedence level with next op, executes now or postpones
1324 \ifnum #1>#4%
1325 \xint_afterfi {\expandafter #2\romannumeral`&&%
1326 \csname XINT_#5_op_##2\endcsname {##3}}%
1327 \else
1328 \xint_afterfi {\expandafter ##1\expandafter ##2%
1329 \csname .=%
1330 \XINT:NEhook:one#6{\XINT_expr_unlock ##3}\endcsname }%
1331 \fi
1332 }%
1333 }%

```

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.

```

1334 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1335 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1336 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{\vi}{vii}{viii}{ix}}%

```

10.32 ? as two-way and ?? as three-way conditionals with braced branches

In 1.1, I overload ? with ??, as : will be used for list extraction, problem with (stuff)??(1){0} for example, one should put a space (stuff)? ?(1){0} will work. Small idiosyncrasy. (which has been removed in 1.2h, there is no problem anymore with (test)??(1){0}, however (test)??{!}(x) is not accepted; but (test)??(x){!(x)} is or even with ?(?!(x).)

syntax: ?{yes}{no} and ??{<0}{=0}{>0}.

The difficulty is to recognize the second ? without removing braces as would be the case with standard parsing of operators. Hence the ? operator is intercepted in \XINT_expr_getop_b.

1.2h corrects a bug in \XINT_expr_op_? which in context like (test)?{foo}{bar} would provoke expansion of \foo, or also with (test)??{bar} would result in an error. The fix also solves the (test)??(1){0} issue mentioned above.

```

1337 \let\XINT_expr_precedence_? \xint_c_x
1338 \def\XINT_expr_op_? #1#2%
1339 {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
1340 \def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
1341 \def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
1342 \else\expandafter\xint_bye\fi }%
1343 \def\XINT_expr_op_?checkc #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1344 \def\XINT_expr_op_?a #1#2#3%
1345 {%
1346 \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1347 }%

```

```

1348 \let\XINT_flexpr_op_?\XINT_expr_op_?
1349 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1350 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
1351 {%
1352     \xintiiifSgn {\XINT_expr_unlock #1}%
1353     {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1354 }%

```

10.33 ! as postfix factorial operator

```

1355 \let\XINT_expr_precedence_! \xint_c_x
1356 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1357     \csname .=\XINT:NEhook:one\xintFac{\XINT_expr_unlock #1}\endcsname }%
1358 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1359     \csname .=\XINT:NEhook:one\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1360 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1361     \csname .=\XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #1}\endcsname }%

```

10.34 The A/B[N] mechanism

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

```

1362 \catcode`[ 11
1363 \catcode`* 11
1364 \let\XINT_expr_precedence_[ \xint_c_vii
1365 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1366     \csname .=\xintE{\XINT_expr_unlock #1}%
1367     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1368 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1369     \csname .=\xintiiE{\XINT_expr_unlock #1}%
1370     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1371 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1372     \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1373     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1374 \catcode`[ 12
1375 \catcode`* 12

```

10.35 \XINT_expr_op_` for recognizing functions

The "onliteral" intercepts is for bool, togl, protect, ... but also for add, mul, seq, etc... Genuine functions have expr, iiexpr and flexpr versions (or only one or two of the three).

With 1.2c "onliteral" is also used to disambiguate variables from functions. However as I use only a \ifcsname test, in order to be able to re-define a variable as function, I move the check for being a function first. Each variable name now has its onliteral_<name> associated macro which is

the new way tacit multiplication in front of a parenthesis is implemented. This used to be decided much earlier at the time of `\XINT_expr_func`.

The advantage of our choices for 1.2c is that the same name can be used for a variable or a function, the parser will apply the correct interpretation which is decided by the presence or not of an opening parenthesis next.

```

1376 \def\XINT_tmpa #1#2#3{%
1377   \def #1##1%
1378   {%
1379     \ifcsname XINT_#3_func_##1\endcsname
1380     \xint_dothis{\expandafter\expandafter
1381       \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1382     \ifcsname XINT_expr_onliteral_##1\endcsname
1383     \xint_dothis{\csname XINT_expr_onliteral_##1\endcsname}\fi
1384     \xint_orthat{\XINT_expr_unknown_function {##1}%
1385       \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1386   }%
1387 }%
1388 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {##1}}%
1389 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1390   \expandafter\XINT_tmpa
1391     \csname XINT_#1_op_`\expandafter\endcsname
1392     \csname XINT_#1_oparen\endcsname
1393     {##1}%
1394 }%
1395 \def\XINT_expr_func_unknown #1#2#3%
1396   {\expandafter #1\expandafter #2\csname .=0\endcsname }%

```

10.36 The `bool`, `togl`, `protect` pseudo “functions”

`bool`, `togl` and `protect` use delimited macros. They are not true functions, they turn off the parser to gather their “variable”.

```

1397 \def\XINT_expr_onliteral_bool #1)%
1398   {\expandafter\XINT_expr_getop\csname .=\xintBool{##1}\endcsname }%
1399 \def\XINT_expr_onliteral_togl #1)%
1400   {\expandafter\XINT_expr_getop\csname .=\xintToggle{##1}\endcsname }%
1401 \def\XINT_expr_onliteral_protect #1)%
1402   {\expandafter\XINT_expr_getop\csname .=\detokenize{##1}\endcsname }%

```

10.37 The `break` function

`break` is a true function, the parsing via expansion of the succeeding material proceeded via `_oparen` macros as with any other function.

```

1403 \def\XINT_expr_func_break #1#2#3%
1404   {\expandafter #1\expandafter #2\csname .=?\romannumeral`&&@\XINT_expr_unlock #3\endcsname }%
1405 \let\XINT_flexpr_func_break \XINT_expr_func_break
1406 \let\XINT_iiexpr_func_break \XINT_expr_func_break

```

10.38 The `qint`, `qfrac`, `qfloat` “functions”

New with 1.2. Allows the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general.

```

1407 \def\XINT_expr_onliteral_qint #1)%
1408     {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1409 \def\XINT_expr_onliteral_qfrac #1)%
1410     {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1411 \def\XINT_expr_onliteral_qfloat #1)%
1412     {\expandafter\XINT_expr_getop\csname .=\XINTinFloatdigits{#1}\endcsname }%

```

10.39 The `random()` and `grand()` “functions”

1.3b. Function-like syntax but with no argument currently, so let's use fast parsing which requires though the closing parenthesis to be explicit.

```

1413 \def\XINT_expr_onliteral_random #1)%
1414     {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSdigits\endcsname }%
1415 \def\XINT_expr_onliteral_grand #1)%
1416     {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSixteen\endcsname }%

```

10.40 `\XINT_expr_op__` for recognizing variables

The 1.1 mechanism for `\XINT_expr_var_<varname>` has been modified in 1.2c. The `<varname>` associated macro is now only expanded once, not twice. We arrive here via `\XINT_expr_func`.

```

1417 \def\XINT_expr_op__ #1% op__ with two _'s
1418     {%
1419         \ifcsname XINT_expr_var_#1\endcsname
1420             \expandafter\xint_firstoftwo
1421         \else
1422             \expandafter\xint_secondoftwo
1423         \fi
1424         {\expandafter\expandafter\expandafter
1425          \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1426         {\XINT_expr_unknown_variable {#1}%
1427          \expandafter\XINT_expr_getop\csname .=\0\endcsname}%
1428     }%
1429 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1430 \let\XINT_flexpr_op__ \XINT_expr_op__
1431 \let\XINT_iexpr_op__ \XINT_expr_op__

```

10.41 User defined variables: `\xintdefvar`, `\xintdefiivar`, `\xintdeffloatvar`

1.1 An active `:` character will be a pain with our delimited macros and I almost decided not to use `:=` but rather `=` as assignation operator, but this is the same problem inside expressions with the modulo operator `/:`, or with `babel+frenchb` with all high punctuation `?`, `!`, `:`, `;`.

Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with `@` or an underscore are reserved.

Note (2015/11/11): although defined since october 2014 with 1.1, they were only very briefly mentioned in the user documentation, I should have expanded more. I am now adding functions to variables, and will rewrite entirely the documentation of `xintexpr.sty`.

Package xintexpr implementation

1.2c adds the "onliteral" macros as we changed our tricks to disambiguate variables from functions if followed by a parenthesis, in order to allow function names to have precedence on variable names.

I don't issue warnings if a an attempt to define a variable name clashes with a pre-existing function name, as I would have to check `expr`, `iiexpr` and also `floatexpr`. And anyhow overloading a function name with a variable name is allowed, the only thing to know is that if an opening parenthesis follows it is the function meaning which prevails.

2015/11/13: I now first do an a priori complete expansion of #1, and then apply `\detokenize` to the result, and remove spaces.

2015/11/21: finally I do not detokenize the variable name. Because this complicated the `\xintunassignvar` if it did the same and we wanted to use it to redeclare a letter as dummy variable.

Documentation of 1.2d said that the tacit multiplication always was done with increased precedence, but I had not at that time made up my mind for the case of `variable(stuff)` and pushed to CTAN early because I need to fix the bug I had introduced in 1.2c which itself I had pushed to CTAN early because I had to fix the 1.2 bug with subtraction...

Finally I decide to do it indeed. Hence for 1.2e. This only impacts situations such as `A/B(stuff)`, which are thus interpreted as `A/(B*(stuff))`.

1.2p (2017/12/01) extends `\xintdefvar` for simultaneous assignments to multiple variables.

```
1432 \catcode`* 11
1433 \def\xINT_expr_defvar_one #1#2%
1434 {%
1435   \expandafter\edef\csname XINT_expr_var_#1\endcsname
1436     {\expandafter\noexpand#2}%
1437   \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1438     {\XINT_expr_precedence_*** *\expandafter\noexpand#2}%
1439   \ifxintverbose\xintMessage{xintexpr}{Info}
1440     {Variable "#1" defined with value \expandafter\xINT_expr_unlock#2.}%
1441   \fi
1442 }%
1443 \catcode`* 12
1444 \def\xINT_expr_defvar #1#2#3;%
1445 {%
1446   \edef\xINT_expr_tmpa{#2}%
1447   \edef\xINT_expr_tmpa{\xint_zapspaces_o\xINT_expr_tmpa}%
1448   \edef\xINT_expr_tmpc{\xintCSVLength{\XINT_expr_tmpa}}%
1449   \ifcase\xINT_expr_tmpc
1450     \xintMessage {xintexpr}{Warning}
1451     {Aborting: impossible to declare variable with empty name.}%
1452   \or
1453     \edef\xINT_expr_tmpb{\romannumeral0#1#3\relax}%
1454     \XINT_expr_defvar_one\xINT_expr_tmpa\xINT_expr_tmpb
1455   \else
1456     \edef\xINT_expr_tmpb
1457       {\expandafter\xINT_expr_unlock\romannumeral0#1#3\relax}%
1458     \edef\xINT_expr_tmpd{\xintCSVLength{\XINT_expr_tmpb}}%
1459     \ifnum\xINT_expr_tmpc=\XINT_expr_tmpd\space
1460       \xintAssignArray\xintCSVtoList\xINT_expr_tmpa\to\xINT_expr_tmpvar
1461       \xintAssignArray
1462       \xintApply\xINT_expr_lockit{\xintCSVtoList\xINT_expr_tmpb}%
1463       \to\xINT_expr_tmpval
1464     \def\xINT_expr_tmpd{1}%
1465     \xintloop
```

```

1466     \expandafter\XINT_expr_defvar_one
1467     \csname XINT_expr_tmpvar\XINT_expr_tmpd\expandafter\endcsname
1468     \csname XINT_expr_tmpval\XINT_expr_tmpd\endcsname
1469     \ifnum\XINT_expr_tmpd<\XINT_expr_tmpc\space
1470     \edef\XINT_expr_tmpd{\the\numexpr\XINT_expr_tmpd+1}%
1471     \repeat
1472     \xintRelaxArray\XINT_expr_tmpvar
1473     \xintRelaxArray\XINT_expr_tmpval
1474     \else
1475     \xintMessage {xintexpr}{Warning}
1476     {Aborting: mismatch between number of variables (\XINT_expr_tmpc)
1477     and number of values (\XINT_expr_tmpd).}%
1478     \fi
1479     \fi
1480 }%
1481 \catcode`: 12
1482 \def\xintdefvar #1:={\XINT_expr_defvar\xintbareeval {#1}}%
1483 \def\xintdefiivar #1:={\XINT_expr_defvar\xintbareiieval {#1}}%
1484 \def\xintdeffloatvar #1:={\XINT_expr_defvar\xintbarefloateval {#1}}%
1485 \catcode`: 11

```

10.42 \xintunassignvar

1.2e. Currently not possible to genuinely ``undefine'' a variable, all we can do is to let it stand for zero and generate an error. The reason is that I chose to use \ifcsname tests in \XINT_expr_op__ and \XINT_expr_op_`.

```

1486 \def\xintunassignvar #1{%
1487     \edef\XINT_expr_tmpa{#1}%
1488     \edef\XINT_expr_tmpa {\xint_zapspace_o\XINT_expr_tmpa}%
1489     \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1490     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1491     \expandafter\XINT_expr_makedummy\XINT_expr_tmpa
1492     \ifxintverbose\xintMessage {xintexpr}{Info}%
1493     {Character \XINT_expr_tmpa\space usable as dummy variable (if with catcode letter).}%
1494     \fi
1495     \else
1496     \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1497     {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
1498     \expandafter\edef\csname XINT_expr_onliteral_\XINT_expr_tmpa\endcsname
1499     {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
1500     \ifxintverbose\xintMessage {xintexpr}{Info}
1501     {Variable \XINT_expr_tmpa\space has been ``unassigned''.}%
1502     \fi
1503     \fi
1504     \else
1505     \xintMessage {xintexpr}{Warning}
1506     {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1507     \fi
1508 }%
1509 \def\XINT_expr_undefined #1{\xintError:replaced_by_zero\xint_gobble_i {#1}}%

```

10.43 seq and the implementation of dummy variables

10.43.1	All letters usable as dummy variables	336
10.43.2	omit and abort	337
10.43.3	The special variables @, @1, @2, @3, @4, @@, @@(1), . . . , @@@, @@@(1), . . . for recursion	338
10.43.4	<code>\XINT_expr_onliteral_seq</code>	339
10.43.5	<code>\XINT_expr_onliteral_seq_a</code>	339
10.43.6	<code>\XINT_isbalanced_a</code> for <code>\XINT_expr_onliteral_seq_a</code>	339
10.43.7	<code>\XINT_allexpr_func_seqx</code>	340
10.43.8	Evaluation over list, <code>\XINT_expr_seq:_a</code> with break, abort, omit	340
10.43.9	Evaluation over ++ generated lists with <code>\XINT_expr_seq:_A</code>	341

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain `\xintNewExpr` !)

The `\XINT_expr_onliteral_seq_a` parses: "expression, variable=list)" (when it is called the opening (has been swallowed, and it looks for the ending one.) Both expression and list may themselves contain parentheses and commas, we allow nesting. For example "x^2,x=1..10)", at the end of seq_a we have {variable{expression}}{list}, in this example {x{x^2}}{1..10}, or more complicated "seq(add(y,y=1..x),x=1..10)" will work too. The variable is a single lowercase Latin letter.

The complications with `\xint_c_xviii` in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iexpr.

10.43.1 All letters usable as dummy variables

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..[d]..b, or for seq with dummy variable where omit has omitted everything may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of `\XINT_expr_onliteral_<name>`.

In 1.1 a letter variable say X was acting as a delimited macro looking for !X{stuff} and then would expand the stuff inside a `\csname.=...\endcsname`. I don't think I used the possibilities this opened and the 1.2c version has stuff_already encapsulated thus a single token. Only one expansion, not two is then needed in `\XINT_expr_op__`.

I had to accordingly modify seq, add, mul and subs, but fortunately realized that the @, @1, etc... variables for rseq, rrseq and iter already had been defined in the way now also followed by the Latin letters as dummy variables.

The 1.2e `\XINT_expr_makedummy` was adjoined `\xintnewdummy` by 1.2k for a public interface. It should not be used with multi-letter argument. The add, mul, seq, etc... can only work with one-letter long dummy variable. And this will almost certainly not change.

Also 1.2e does the tacit multiplication x(stuff)->x*(stuff) in its higher precedence form. Things are easy now that variables always fetch a single already locked value `\.=<number>`.

The tacit multiplication in case of the ```nil''` variable doesn't make much sense but we do it anyhow.

```
1510 \catcode`* 11
1511 \def\XINT_expr_makedummy #1%
```



```

1512 {%
1513   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1514     {##2##1\relax !#1##2}%
1515   \expandafter\def\csname XINT_expr_onliteral_#1\endcsname ##1\relax !#1##2%
1516     {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
1517 }%
1518 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1519 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1520 \def\xintnewdummy #1{%
1521   \XINT_expr_makedummy{#1}%
1522   \ifxintverbose\xintMessage {xintexpr}{Info}%
1523     {Character #1 now usable as dummy variable (if with catcode letter).}%
1524   \fi
1525 }%
1526 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1527 \edef\XINT_expr_onliteral_nil
1528   {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname (}%
1529 \catcode`* 12

```

10.43.2 omit and abort

June 24 and 25, 2014.

Added comments 2015/11/13:

Et la documentation ? on n'y comprend plus rien. Trop rusé.

```
\def\XINT_expr_var_omit #1\relax !{1^C!{}}{.}\relax !}
```

```
\def\XINT_expr_var_abort #1\relax !{1^C!{}}{.\^}\relax !}
```

C'était accompagné de `\XINT_expr_precedence_^C=0` et d'un hack au sein même des macros until de plus bas niveau.

Le mécanisme sioux était le suivant: `^C` est déclaré comme un opérateur de précedence nulle. Lorsque le parseur trouve un "omit" dans un seq ou autre, il va insérer dans le stream `\XINT_expr_getop` suivi du texte de remplacement. Donc ici on avait un 1 comme place holder, puis l'opérateur `^C`. Celui-ci étant de précedence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le `until_end_b` (et le `until_)_b`) qui confronté à `^C`, va se relancer à zéro, le `getnext` va trouver le `!{}}{.}\relax !}` et ensuite il y aura `\relax`, et le résultat sera `\.=!` pour omit ou `\.=^` pour abort. Les routines des boucles seq, iter, etc... peuvent alors repérer le ! ou ^ et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais `^C` a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car `seq(2^C, C=1..5)` est alors impossible. De toute façon ce `^C` était à usage interne uniquement.

Il me faut un symbole d'opérateur qui ne rentre pas en conflit. Bon je vais prendre `!?`. Ensuite au lieu de hacker `until_end`, il vaut mieux lui donner précedence 2 (mais ça ne pourra pas marcher à l'intérieur de parenthèses il faut d'abord les fermer manuellement) et lui associer un simplement un op spécial. Je n'avais pas fait cela peut-être pour éviter d'avoir à définir plusieurs macros. Le #1 dans la définition de `\XINT_expr_op_!?` est le résultat de l'évaluation forcée précédente.

Attention que les premier ! doivent être de catcode 12 sinon ils signalent une sous-expression qui déclenche une multiplication tacite.

```

1530 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
1531 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
1532 \def\XINT_expr_op_!#? #1#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
1533 \let\XINT_iiexpr_op_!#? \XINT_expr_op_!#?
1534 \let\XINT_flexpr_op_!#? \XINT_expr_op_!#?

```

10.43.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

October 2014: I had completely forgotten what the @@@ etc... stuff were supposed to do: this is for nesting recursions! (I was mad back in June). @@(N) gives the Nth back, @@@(N) gives the Nth back of the higher recursion!

1.2c adds the needed "onliterate" now that tacit multiplication between a variable and a (has a new mechanism. 1.2e does this tacit multiplication with higher precedence.

For the record, the ~ has catcode 3 in this code.

```

1535 \catcode`? 3 \catcode`* 11
1536 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1537 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1538 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1539 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1540 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1541 \def\XINT_expr_onliterate_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2)%
1542 \expandafter\let\csname XINT_expr_onliterate_@1\endcsname \XINT_expr_onliterate_@
1543 \expandafter\def\csname XINT_expr_onliterate_@2\endcsname #1~#2#3%
1544         {\XINT_expr_precedence_*** *#3(#1~#2#3)%
1545 \expandafter\def\csname XINT_expr_onliterate_@3\endcsname #1~#2#3#4%
1546         {\XINT_expr_precedence_*** *#4(#1~#2#3#4)%
1547 \expandafter\def\csname XINT_expr_onliterate_@4\endcsname #1~#2#3#4#5%
1548         {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5)%
1549 \catcode`* 12
1550 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1551 {%
1552   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1553     {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1554 }%
1555 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1556 {%
1557   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1558     {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1559 }%
1560 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1561 {%
1562   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1563     {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1564 }%
1565 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1566 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1567 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
1568 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1569 {%
1570   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1571     {\XINT_expr_unlock#3}{#5}#4~#5?%
1572 }%
1573 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1574 {%
1575   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1576     {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1577 }%
1578 \def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%

```

```

1579 {%
1580   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1581   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1582 }%
1583 \catcode`? 11

```

10.43.4 \XINT_expr_onliteral_seq

```

1584 \def\XINT_expr_onliteral_seq
1585   {\expandafter\XINT_expr_onliteral_seq_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1586 \def\XINT_expr_onliteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

10.43.5 \XINT_expr_onliteral_seq_a

```

1587 \def\XINT_expr_onliteral_seq_a #1#2,%
1588 {%
1589   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1590     \expandafter\XINT_expr_onliteral_seq_c
1591     \or\expandafter\XINT_expr_onliteral_seq_b
1592     \else\expandafter\xintError:we_are_doomed
1593   \fi {#1#2},%
1594 }%
1595 \def\XINT_expr_onliteral_seq_b #1,{\XINT_expr_onliteral_seq_a {#1,}}%
1596 \def\XINT_expr_onliteral_seq_c #1,#2#3% #3 pour absorber le =
1597 {%
1598   \XINT_expr_onliteral_seq_d {#2{#1}}{}%
1599 }%
1600 \def\XINT_expr_onliteral_seq_d #1#2#3)%
1601 {%
1602   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1603     \or\expandafter\XINT_expr_onliteral_seq_e
1604     \else\expandafter\xintError:we_are_doomed
1605   \fi
1606   {#1}{#2#3}%
1607 }%
1608 \def\XINT_expr_onliteral_seq_e #1#2{\XINT_expr_onliteral_seq_d {#1}{#2}}%

```

10.43.6 \XINT_isbalanced_a for \XINT_expr_onliteral_seq_a

Expands to `\xint_c_mone` in case a closing `)` had no opening (matching it, to `\@ne` if opening `)` had no closing `)` matching it, to `\z@` if expression was balanced.

```

1609 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1610 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1611 \def\XINT_isbalanced_b #1)#2%
1612   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%
1613   if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1
1614 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1615   {\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.

```
1616 \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.

```
1617 \def\xINT_isbalanced_d #1)#2%
```

```
1618   {\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.

```
1619 \def\xINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%
```

10.43.7 \XINT_allexpr_func_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT_expr_seq:_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT_allexp_seqx, #2 is the list, evaluated and encapsulated, #3 is the dummy variable, #4 is the expression to evaluate repeatedly.

A special case is a list generated by <variable>+: then #2 is {\.=+\.=<start>}

```
1620 \def\xINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareeval }%
```

```
1621 \def\xINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
```

```
1622 \def\xINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
```

```
1623 \def\xINT_allexpr_seqx #1#2#3#4%
```

```
1624 {%
```

```
1625   \expandafter \XINT_expr_getop
```

```
1626   \csname .=\expandafter\xINT_expr_seq:_aa
```

```
1627     \romannumeral`&&@\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
```

```
1628 }%
```

```
1629 \def\xINT_expr_seq:_aa #1{\if +#1\expandafter\xINT_expr_seq:_A\else
```

```
1630   \expandafter\xINT_expr_seq:_a\fi #1}%
```

10.43.8 Evaluation over list, \XINT_expr_seq:_a with break, abort, omit

The #2 here is \...bareeval <expression>\relax !<variable name>. The #1 is a comma separated list of values to assign to the dummy variable. The \XINT_expr_seq_empty? intervenes immediately after handling of firstvalue.

1.2c has rewritten to a large extent this and other similar loops because the dummy variables now fetch a single encapsulated token (apart from a good means to lose a few hours needlessly -- as I have had to rewrite and review most everything, this change could make the thing more efficient if the same variable is used many times in an expression, but we are talking micro-seconds here anyhow.)

```
1631 \def\xINT_expr_seq:_a #1!#2{\expandafter\xINT_expr_seq_empty?
```

```
1632   \romannumeral0\xINT_expr_seq:_b {#2}#1,^,}%
```

```
1633 \def\xINT_expr_seq:_b #1#2#3,{%
```

```
1634   \if ,#2\xint_dothis\xINT_expr_seq:_noop\fi
```

```
1635   \if ^#2\xint_dothis\xINT_expr_seq:_end\fi
```

```
1636   \xint_orthat{\expandafter\xINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
```

```
1637 }%
```

```
1638 \def\xINT_expr_seq:_noop\csname.=#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
```

```

1639 \def\XINT_expr_seq:_end \csname.=^\endcsname #1{}%
1640 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1641 \def\XINT_expr_seq:_d #1{\if #1^\xint_dothis\XINT_expr_seq:_abort\fi
1642         \if #1?\xint_dothis\XINT_expr_seq:_break\fi
1643         \if #1!\xint_dothis\XINT_expr_seq:_omit\fi
1644         \xint_orthat{\XINT_expr_seq:_goon #1}}%
1645 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{ }%
1646 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1647 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1648 \def\XINT_expr_seq:_goon #1!#2#3#4{,#1\XINT_expr_seq:_b {#4}}%

```

If all is omitted or list is empty, `_empty?` will fetch within the `#1` a `\endcsname` token and construct "nil" via `<space>\endcsname`, if not `#1` will be a comma and the gobble will swallow the space token and the extra `\endcsname`.

```

1649 \def\XINT_expr_seq_empty? #1{%
1650 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1651 \XINT_expr_seq_empty? { }%

```

10.43.9 Evaluation over ++ generated lists with `\XINT_expr_seq:_A`

This is for index lists generated by `n++`. The starting point will have been replaced by its ceil (added: in fact with version 1.1. the ceil was not yet evaluated, but `_var_<letter>` did an expansion of what they fetch). We use `\numexpr` rather than `\xintInc`, hence the indexing is limited to small integers.

The 1.2c version of `n++` produces a `#1` here which is already a single `\.=<value>` token.

```

1652 \def\XINT_expr_seq:_A +#1!%
1653   {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1654 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1655 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1656         \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1657         \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1658         \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1659 \def\XINT_expr_seq:_Abort #1!#2#3#4{ }%
1660 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1661 \def\XINT_expr_seq:_Omit #1!#2#3%
1662   {\expandafter\XINT_expr_seq:_D
1663     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1664 \def\XINT_expr_seq:_Goon #1!#2#3%
1665   {,#1\expandafter\XINT_expr_seq:_D
1666     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%

```

10.44 add, mul

1.2c uses more directly the `\xintiiAdd` etc... macros and has `opxadd/opxmul` rather than a single `opx`. This is less conceptual as I use explicitly the associated macro names for `+`, `*` but this makes other things more efficient, and the code more readable.

```

1667 \def\XINT_expr_onliteral_add
1668   {\expandafter\XINT_expr_onliteral_add_f\romannumeral`&&@\XINT_expr_onliteral_seq_a {}}%
1669 \def\XINT_expr_onliteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1670 \def\XINT_expr_onliteral_mul

```

```
1671 {\expandafter\XINT_expr_onliteral_mul_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1672 \def\XINT_expr_onliteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%
```

10.44.1 `\XINT_expr_func_opxadd`, `\XINT_flexpr_func_opxadd`, `\XINT_iiexpr_func_opxadd` and `same` for `mul`

modified 1.2c.

```
1673 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1674 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1675 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1676 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1677 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1678 \def\XINT_iiexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiMul 1}}%
```

#1=bareval etc, #2={Add0} ou {Mul1}, #3=liste encapsulée, #4=la variable, #5=expression

```
1679 \def\XINT_allexpr_opx #1#2#3#4#5%
1680 {%
1681   \expandafter\XINT_expr_getop
1682   \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1683     \romannumeral`&&\XINT_expr_unlock #3!{#1#5}\relax !#4}{#2}\endcsname
1684 }%
1685 \def\XINT_expr_op:_a #1!#2#3{\XINT_expr_op:_b #3{#2}#1,^,}%
```

#2 in \XINT_expr_op:_b is the partial result of computation so far, not locked. A noop with have #4=, and #5 the next item which we need to recover. No need to be very efficient for that in op:_noop. In op:_d, #4 is \xintAdd or similar.

```
1686 \def\XINT_expr_op:_b #1#2#3#4#5,{%
1687   \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1688   \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1689   \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1690 }%
1691 \def\XINT_expr_op:_c #1#2#3#4%
1692   {\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1693 \def\XINT_expr_op:_d #1!#2#3#4#5%
1694   {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1695     {\romannumeral`&&\XINT:NEhook:two#4{\XINT_expr_unlock#1}{#5}}}%
```

The replacement text had `expr_seq:_b` rather than `expr_op:_b` due to a left-over from copy-paste. This made `add` and `mul` fail with an empty range for the variable (or "nil" in the list of values). Fixed in 1.2h.

```
1696 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,}%
1697 \def\XINT_expr_op:_end \csname.=^\endcsname #1#2#3{#3}%
```

10.45 `subs`

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

```
1698 \def\XINT_expr_onliteral_subs
1699   {\expandafter\XINT_expr_onliteral_subs_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1700 \def\XINT_expr_onliteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
```

```

1701 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1702 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1703 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1704 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1705 {% #3 is the dummy variable, #4 is the expression to evaluate
1706     \expandafter\expandafter\expandafter\XINT_expr_getop
1707     \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1708 }%
1709 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

10.46 rseq

10.46.1	<code>\XINT_expr_rseqx</code>	...	343
10.46.2	<code>\XINT_expr_rseqy</code>	...	343
10.46.3	<code>\XINT_expr_rseq:_a</code> etc...	...	344
10.46.4	<code>\XINT_expr_rseq:_A</code> etc...	...	344

When `func_rseq` has its turn, initial segment has been scanned by `oparen`, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion. Notice that the ; is discovered during standard parsing mode, it may be for example `{;}` or arise from expansion as `rseq` does not use a delimited macro to locate it.

Here and in `rrseq` and `iter`, 1.2c adds also use of `\xintthebareeval`, etc...

```

1710 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval }%
1711 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1712 \def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval }%
1713 \def\XINT_allexpr_rseq #1#2#3%
1714 {%
1715     \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1716     #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1717 }%

```

10.46.1 `\XINT_expr_rseqx`

The `(#5)` is for ++ mechanism which must have its closing parenthesis.

```

1718 \def\XINT_expr_rseqx #1#2#3#4#5%
1719 {%
1720     \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1721 }%

```

10.46.2 `\XINT_expr_rseqy`

`#1`=valeurs pour variable (locked), `#2`=toutes les valeurs initiales (csv,locked), `#3`=variable, `#4`=expr, `#5`=`\xintthebareeval` ou `\xintthebarefloateval` ou `\xintthebareiieval`

```

1722 \def\XINT_expr_rseqy #1#2#3#4#5%
1723 {%
1724     \expandafter \XINT_expr_getop
1725     \csname .=\XINT_expr_unlock #2%
1726     \expandafter\XINT_expr_rseq:_aa
1727         \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1728 }%

```

```
1729 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1730 \expandafter\XINT_expr_rseq:_a\fi #1}%
```

10.46.3 \XINT_expr_rseq:_a etc...

```
1731 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,}%
1732 \def\XINT_expr_rseq:_b #1#2#3#4,{%
1733 \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1734 \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1735 \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname.#3#4\endcsname
1736 {#1}{#2}%
1737 }%
1738 \def\XINT_expr_rseq:_noop\csname.#,\#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,}%
1739 \def\XINT_expr_rseq:_end \csname.#^\endcsname #1#2{%}
1740 \def\XINT_expr_rseq:_c #1#2#3%
1741 {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
1742 \def\XINT_expr_rseq:_d #1{%
1743 \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1744 \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1745 \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1746 \xint_orthat{\XINT_expr_rseq:_goon #1}%
1747 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1748 \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1749 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1750 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{}%
1751 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%
```

10.46.4 \XINT_expr_rseq:_A etc...

n++ for rseq. With 1.2c dummy variables pick a single token.

```
1752 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1753 \def\XINT_expr_rseq:_D #1#2#3%
1754 {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
1755 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1756 \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1757 \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1758 \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1759 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1760 {,#1\expandafter\XINT_expr_rseq:_D
1761 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1762 \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1763 \def\XINT_expr_rseq:_Omit #1!#2#3~#4#5%
1764 {\expandafter\XINT_expr_rseq:_D
1765 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1766 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{%}
1767 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%
```

10.47 iter

10.47.1	\XINT_expr_iterx	345
10.47.2	\XINT_expr_itory	345
10.47.3	\XINT_expr_iter:_a etc...	345
10.47.4	\XINT_expr_iter:_A etc...	346

Prior to 1.2g, the `iter` keyword was what is now called `iterr`, analogous with `rrseq`. Somehow I forgot an `iter` functioning like `rseq` with the sole difference of printing only the last iteration. Both `rseq` and `iter` work well with list selectors, as `@` refers to the whole comma separated sequence of the initial values. I have thus deliberately done the backwards incompatible renaming of `iter` to `iterr`, and the new `iter`.

```
1768 \def\XINT_expr_func_iter  {\XINT_allexpr_iter \xintbareeval      \xintthebareeval      }%
1769 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1770 \def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval   \xintthebareiieval   }%
1771 \def\XINT_allexpr_iter #1#2#3%
1772 {%
1773   \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
1774   #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1775 }%
```

10.47.1 \XINT_expr_iterx

The `(#5)` is for `++` mechanism which must have its closing parenthesis.

```
1776 \def\XINT_expr_iterx #1#2#3#4#5%
1777 {%
1778   \expandafter\XINT_expr_itery\romannumeral0#1(#5)\relax #3#4#2%
1779 }%
```

10.47.2 \XINT_expr_itery

`#1=valeurs pour variable (locked)`, `#2=toutes les valeurs initiales (csv,locked)`, `#3=variable`, `#4=expr`, `#5=\xintthebareeval` ou `\xintthebarefloateval` ou `\xintthebareiieval`

```
1780 \def\XINT_expr_itery #1#2#3#4#5%
1781 {%
1782   \expandafter \XINT_expr_getop
1783   \csname .=%
1784   \expandafter\XINT_expr_iter:_aa
1785   \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1786 }%
1787 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1788                               \expandafter\XINT_expr_iter:_a\fi #1}%

```

10.47.3 \XINT_expr_iter:_a etc...

```
1789 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,}%
1790 \def\XINT_expr_iter:_b #1#2#3#4,{%
1791   \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1792   \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1793   \xint_orthat{\expandafter\XINT_expr_iter:_c}%
1794   \csname.=#3#4\endcsname {#1}{#2}%
1795 }%
1796 \def\XINT_expr_iter:_noop\csname.=#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,}%
1797 \def\XINT_expr_iter:_end \csname.=#^1\endcsname #1#2{\XINT_expr:_unlock #1}%
1798 \def\XINT_expr_iter:_c #1#2#3%
1799   {\expandafter\XINT_expr_iter:_d\romannumeral`&&#3#1~#2{#3}}%
1800 \def\XINT_expr_iter:_d #1{%
```

```

1801 \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1802 \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1803 \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1804 \xint_orthat{\XINT_expr_iter:_goon #1}%
1805 \def\XINT_expr_iter:_goon #1!#2#3~#4#5%
1806 {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
1807 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1808 \def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
1809 \def\XINT_expr_iter:_break #1!#2#3~#4#5#6^,{#1}%

```

10.47.4 `\XINT_expr_iter:_A` etc...

`n++` for `iter`. With 1.2c dummy variables pick a single token.

```

1810 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
1811 \def\XINT_expr_iter:_D #1#2#3%
1812 {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
1813 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1814 \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1815 \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1816 \xint_orthat{\XINT_expr_iter:_Goon #1}%
1817 \def\XINT_expr_iter:_Goon #1!#2#3~#4#5%
1818 {\expandafter\XINT_expr_iter:_D
1819 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1820 \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1821 \def\XINT_expr_iter:_Omit #1!#2#3~#4#5%
1822 {\expandafter\XINT_expr_iter:_D
1823 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1824 \def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
1825 \def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%

```

10.48 `rrseq`

10.48.1	<code>\XINT_expr_rrseqx</code>	346
10.48.2	<code>\XINT_expr_rrseqy</code>	347
10.48.3	<code>\XINT_expr_rrseq:_a</code> etc.	347
10.48.4	<code>\XINT_expr_rrseq:_A</code> etc.	348

When `func_rrseq` has its turn, initial segment has been scanned by `oparen`, the `;` mimicking the rôle of a closing parenthesis, and stopping further expansion.

```

1826 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1827 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1828 \def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1829 \def\XINT_allexpr_rrseq #1#2#3%
1830 {%
1831 \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
1832 #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}%
1833 }%

```

10.48.1 `\XINT_expr_rrseqx`

The `(#5)` is for `++` mechanism which must have its closing parenthesis.

```

1834 \def\XINT_expr_rrseqx #1#2#3#4#5%
1835 {%
1836   \expandafter\XINT_expr_rrseqy\romannumeral0#1(#5)\expandafter\relax
1837   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1838     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1839   #3#4#2%
1840 }%

```

10.48.2 \XINT_expr_rrseqy

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintthebareeval ou \xintthebare-floateval ou \xintthebareiieval

```

1841 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
1842 {%
1843   \expandafter \XINT_expr_getop
1844   \csname .=\XINT_expr_unlock #3%
1845   \expandafter\XINT_expr_rrseq:_aa
1846     \romannumeral`&&@\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1847 }%
1848 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1849   \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

10.48.3 \XINT_expr_rrseq:_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1850 \catcode`? 3
1851 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1852 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1853   \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1854   \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1855   \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname.=#3#4\endcsname
1856   {#1}{#2}%
1857 }%
1858 \def\XINT_expr_rrseq:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1859 \def\XINT_expr_rrseq:_end \csname.=^\endcsname #1#2{%}
1860 \def\XINT_expr_rrseq:_c #1#2#3%
1861   {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
1862 \def\XINT_expr_rrseq:_d #1{%
1863   \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1864   \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1865   \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1866   \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1867 }%
1868 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1869   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1870 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1871 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{,%}
1872 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%

```

10.48.4 `\XINT_expr_rrseq:_A` etc...

`n++` for `rrseq`. With 1.2C, the `#1` in `\XINT_expr_rrseq:_A` is a single token.

```
1873 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1874 \def\XINT_expr_rrseq:_D #1#2#3%
1875   {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1876 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1877   {,#1\expandafter\XINT_expr_rrseq:_D
1878     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1879     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1880 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1881   {\expandafter\XINT_expr_rrseq:_D
1882     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1883 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{}%
1884 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1885 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1886     \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1887     \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1888     \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%
```

10.49 `iterr`

10.49.1	<code>\XINT_expr_iterrx</code>	348
10.49.2	<code>\XINT_expr_iterry</code>	348
10.49.3	<code>\XINT_expr_iterr:_a</code> etc.	349
10.49.4	<code>\XINT_expr_iterr:_A</code> etc.	349

```
1889 \def\XINT_expr_func_iterr {\XINT_allexpr_iterr \xintbareeval \xintthebareeval }%
1890 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval \xintthebarefloateval }%
1891 \def\XINT_iiexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval \xintthebareiieval }%
1892 \def\XINT_allexpr_iterr #1#2#3%
1893 {%
1894   \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
1895   #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
1896 }%
```

10.49.1 `\XINT_expr_iterrx`

The `(#5)` is for `++` mechanism which must have its closing parenthesis.

```
1897 \def\XINT_expr_iterrx #1#2#3#4#5%
1898 {%
1899   \expandafter\XINT_expr_iterry\romannumeral0#1(#5)\expandafter\relax
1900   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1901     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1902   #3#4#2%
1903 }%
```

10.49.2 `\XINT_expr_iterry`

`#1`=valeurs pour variable (locked), `#2`=initial values (reversed, one (braced) token each) `#3`=toutes les valeurs initiales (csv,locked), `#4`=variable, `#5`=expr, `#6`=`\xintbareeval` ou `\xintbarefloateval` ou `\xintbareiieval`

```

1904 \def\XINT_expr_iterry #1#2#3#4#5#6%
1905 {%
1906   \expandafter \XINT_expr_getop
1907   \csname .=%
1908   \expandafter\XINT_expr_iterr:_aa
1909   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1910 }%
1911 \def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
1912   \expandafter\XINT_expr_iterr:_a\fi #1}%

```

10.49.3 \XINT_expr_iterr:_a etc...

```

1913 \def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,%}
1914 \def\XINT_expr_iterr:_b #1#2#3#4,{%
1915   \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
1916   \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
1917   \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
1918   \csname.#3#4\endcsname {#1}{#2}%
1919 }%
1920 \def\XINT_expr_iterr:_noop\csname.=#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,%}
1921 \def\XINT_expr_iterr:_end \csname.^#\endcsname #1#2%
1922   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1923     {,\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1924 \def\XINT_expr_iterr:_c #1#2#3%
1925   {\expandafter\XINT_expr_iterr:_d\romannumeral`&&@#3#1~#2?{#3}}%
1926 \def\XINT_expr_iterr:_d #1{%
1927   \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
1928   \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
1929   \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
1930   \xint_orthat{\XINT_expr_iterr:_goon #1}%
1931 }%
1932 \def\XINT_expr_iterr:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
1933   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1934 \def\XINT_expr_iterr:_omit #1!#2#3~{\XINT_expr_iterr:_b }%
1935 \def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
1936   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1937     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1938 \def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
1939   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1940     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1941 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

10.49.4 \XINT_expr_iterr:_A etc...

n++ for *iterr*. ? is of *catcode* 3 here.

```

1942 \def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
1943 \def\XINT_expr_iterr:_D #1#2#3%
1944   {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
1945 \def\XINT_expr_iterr:_Goon #1!#2#3~#4?#5%
1946   {\expandafter\XINT_expr_iterr:_D
1947     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1948     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1949 \def\XINT_expr_iterr:_Omit #1!#2#3~#4?#5%

```

```

1950     {\expandafter\XINT_expr_iterr:_D
1951     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1952 \def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
1953     {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1954     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1955 \def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
1956     {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1957     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1958 \def\XINT_expr_iterr:_E #1{\if #1^\xint_dothis\XINT_expr_iterr:_Abort\fi
1959     \if #1?\xint_dothis\XINT_expr_iterr:_Break\fi
1960     \if #1!\xint_dothis\XINT_expr_iterr:_Omit\fi
1961     \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
1962 \catcode`? 11

```

10.50 Macros handling csv lists for functions with multiple comma separated arguments in expressions

10.50.1	<code>\xintANDof:csv</code>	350
10.50.2	<code>\xintORof:csv</code>	351
10.50.3	<code>\xintXORof:csv</code>	351
10.50.4	Generic csv routine	351
10.50.5	<code>\xintMaxof:csv, \xintiiMaxof:csv</code>	351
10.50.6	<code>\xintMinof:csv, \xintiiMinof:csv</code>	352
10.50.7	<code>\xintSum:csv, \xintiiSum:csv</code>	352
10.50.8	<code>\xintPrd:csv, \xintiiPrd:csv</code>	352
10.50.9	<code>\xintGCDof:csv, \xintLCMof:csv</code>	352
10.50.10	<code>\xintiiGCDof:csv, \xintiiLCMof:csv</code>	352
10.50.11	<code>\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits</code>	352
10.50.12	<code>\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv</code>	353
10.50.13	<code>\XINTinFloatSum:csv, \XINTinFloatPrd:csv</code>	353

These macros are used inside `\csname... \endcsname`. These things are not initiated by a `\romannumeral` in general, but in some cases they are, especially when involved in an `\xintNewExpr`. They will then be protected against expansion and expand only later in contexts governed by an initial `\romannumeral-`0`. There each new item may need to be expanded, which would not be the case in the use for the `_func_` things.

1.2g adds (to be continued)

10.50.1 `\xintANDof:csv`

1.09a. For use by `\xintexpr` inside `\csname. 1.1`, je remplace `ifTrueAelseB` par `iiNotZero` pour des raisons d'optimisations.

```

1963 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,^}%
1964 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1965     \else\expandafter\XINT_andof:_c\fi #1}%
1966 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1967 \def\XINT_andof:_no #1^{0}%
1968 \def\XINT_andof:_e #1^{1}% works with empty list

```

10.50.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```
1969 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
1970 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1971     \else\expandafter\XINT_orof:_c\fi #1}%
1972 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1973 \def\XINT_orof:_yes #1^{1}%
1974 \def\XINT_orof:_e #1^{0}% works with empty list
```

10.50.3 `\xintXORof:csv`

1.09a. For use by `\xintexpr` (inside a `\csname..\endcsname`).

```
1975 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
1976 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1977 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
1978     \else\expandafter\XINT_xorof:_c\fi #1}%
1979 \def\XINT_xorof:_c #1,#2%
1980     {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1981         \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1982         {\XINT_xorof:_a #2}%
1983     }%
1984 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)
```

10.50.4 Generic csv routine

1.1. generic routine. up to the loss of some efficiency, especially for `Sum:csv` and `Prod:csv`, where `\XINTinFloat` will be done twice for each argument.

```
1985 \def\XINT_oncsv:_empty #1,^,#2{#2}%
1986 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
1987 \def\XINT_oncsv:_a #1#2#3%
1988     {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
1989 \def\XINT_oncsv:_b #1#2#3,%
1990     {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
1991 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
1992 \def\XINT_oncsv:_d #1%
1993     {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
1994 \def\XINT_oncsv:_e #1,#2#3#4%
1995     {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%

```

10.50.5 `\xintMaxof:csv`, `\xintiiMaxof:csv`

1.09i. Rewritten for 1.1. Compatible avec liste vide donnant valeur par défaut. Pas compatible avec items manquants. ah je m'aperçois au dernier moment que je n'ai pas en effet de `\xintiiMax`. Je devrais le rajouter. En tout cas ici c'est uniquement pour `xintiexpr`, dans il faut bien sûr ne pas faire de `xintNum`, donc il faut un `iimax`.

```
1996 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1997     \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
1998 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
1999     \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.50.6 `\xintMinof:csv`, `\xintiiMinof:csv`

1.09i. Rewritten for 1.1. For use by `\xintiexpr`.

```
2000 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
2001         \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
2002 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimin
2003         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

10.50.7 `\xintSum:csv`, `\xintiiSum:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
2004 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
2005         \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
2006 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiaadd
2007         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

10.50.8 `\xintPrd:csv`, `\xintiiPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
2008 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
2009         \expandafter\xint_firstofone\romannumeral`&&@#1,^,{1/1[0]}}%
2010 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
2011         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
```

10.50.9 `\xintGCDof:csv`, `\xintLCMof:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`. Expansion réinstaurée pour besoins de `xintNewExpr` de version 1.1

```
2012 \def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
2013         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
2014 \def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
2015         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

10.50.10 `\xintiiGCDof:csv`, `\xintiiLCMof:csv`

1.1a pour `\xintiexpr`. Ces histoires de ii sont pénibles à la fin.

```
2016 \def\xintiiGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiigcd
2017         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
2018 \def\xintiiLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiilcm
2019         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

10.50.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`, `\XINTinFloatFacdigits`

for `\xintNewExpr` matters, mainly.

```
2020 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
2021 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]}%
2022 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]}%
```


10.50.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h

```
2023 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmax
2024 \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{0[0]}}%
2025 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmin
2026 \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{0[0]}}%
```

10.50.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```
2027 \def\XINTinFloatSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatadd
2028 \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{0[0]}}%
2029 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatmul
2030 \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{1[0]}}%
```

10.51 Auxiliary wrappers for function macros

```
2031 \def\XINT:expr:one:and:opt #1,#2,#3!#4#5%
2032 {%
2033 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2034 \expandafter\xint_secondoftwo\fi
2035 {#4}{#5[\xintNum{#2}]}{#1}%
2036 }%
2037 \def\XINT:expr:tacitzeroifonearg #1,#2,#3!#4#5%
2038 {%
2039 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2040 \expandafter\xint_secondoftwo\fi
2041 {#4{0}}{#5{\xintNum{#2}}}{#1}%
2042 }%
2043 \def\XINT:iiexpr:tacitzeroifonearg #1,#2,#3!#4%
2044 {%
2045 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2046 \expandafter\xint_secondoftwo\fi
2047 {#4{0}}{#4{#2}}{#1}%
2048 }%
2049 \def\XINT:expr:totwo #1#2{#1,#2}%
2050 \def\XINT:expr:two:to:two #1,#2,!#3%
2051 {%
2052 \expandafter\XINT:expr:totwo\romannumeral`&&@%
2053 #3{#1}{#2}%
2054 }%
2055 \def\XINT:expr:two:to:one #1,#2,!#3%
2056 {%
2057 #3{#1}{#2}%
2058 }%
2059 \def\XINT:flexpr:two:to:one #1,#2,!#3%
2060 {%
2061 #3{#1}{#2}%
2062 }%
2063 \let\XINT:flexpr:two:to:two\XINT:flexpr:two:to:one
```

10.52 The num, reduce, preduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, divmod, gcd, lcm, max, min, `+`, `*`, `?`, `!, not, all, any, xor, if, ifsgn, even, odd, first, last, len, reversed, factorial, binomial, and randrange functions

```

2064 \def\XINT_expr_func_num #1#2#3%
2065 {%
2066   \expandafter #1\expandafter #2\csname.=%
2067   \XINT:NEhook:one\xintNum{\XINT_expr_unlock #3}\endcsname
2068 }%
2069 \let\XINT_flexpr_func_num\XINT_expr_func_num
2070 \let\XINT_iiexpr_func_num\XINT_expr_func_num
2071 \def\XINT_expr_func_reduce #1#2#3%
2072 {%
2073   \expandafter #1\expandafter #2\csname.=%
2074   \XINT:NEhook:one\xintIrr{\XINT_expr_unlock #3}[0]\endcsname
2075 }%
2076 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
2077 \def\XINT_expr_func_preduce #1#2#3%
2078 {%
2079   \expandafter #1\expandafter #2\csname.=%
2080   \XINT:NEhook:one\xintPIrr{\XINT_expr_unlock #3}\endcsname
2081 }%
2082 \let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
2083 \def\XINT_expr_func_abs #1#2#3%
2084 {%
2085   \expandafter #1\expandafter #2\csname.=%
2086   \XINT:NEhook:one\xintAbs{\XINT_expr_unlock #3}\endcsname
2087 }%
2088 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
2089 \def\XINT_iiexpr_func_abs #1#2#3%
2090 {%
2091   \expandafter #1\expandafter #2\csname.=%
2092   \XINT:NEhook:one\xintiiAbs{\XINT_expr_unlock #3}\endcsname
2093 }%
2094 \def\XINT_expr_func_sgn #1#2#3%
2095 {%
2096   \expandafter #1\expandafter #2\csname.=%
2097   \XINT:NEhook:one\xintSgn{\XINT_expr_unlock #3}\endcsname
2098 }%
2099 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
2100 \def\XINT_iiexpr_func_sgn #1#2#3%
2101 {%
2102   \expandafter #1\expandafter #2\csname.=%
2103   \XINT:NEhook:one\xintiiSgn{\XINT_expr_unlock #3}\endcsname
2104 }%
2105 \def\XINT_expr_func_frac #1#2#3%
2106 {%
2107   \expandafter #1\expandafter #2\csname.=%
2108   \XINT:NEhook:one\xintTFrac{\XINT_expr_unlock #3}\endcsname
2109 }%
2110 \def\XINT_flexpr_func_frac #1#2#3%

```

```

2111 {%
2112   \expandafter #1\expandafter #2\csname.=%
2113   \XINT:NEhook:one\XINTinFloatFracdigits{\XINT_expr_unlock #3}\endcsname
2114 }%
```

no \XINT_iiexpr_func_frac

```

2115 \def\XINT_expr_func_floor #1#2#3%
2116 {%
2117   \expandafter #1\expandafter #2\csname.=%
2118   \XINT:NEhook:one\xintFloor{\XINT_expr_unlock #3}\endcsname
2119 }%
2120 \let\XINT_flexpr_func_floor\XINT_expr_func_floor
```

The floor and ceil functions in \xintiiexpr require protect(a/b) or, better, \qfrac(a/b); else the / will be executed first and do an integer rounded division.

```

2121 \def\XINT_iiexpr_func_floor #1#2#3%
2122 {%
2123   \expandafter #1\expandafter #2\csname.=%
2124   \XINT:NEhook:one\xintiFloor{\XINT_expr_unlock #3}\endcsname
2125 }%
2126 \def\XINT_expr_func_ceil #1#2#3%
2127 {%
2128   \expandafter #1\expandafter #2\csname.=%
2129   \XINT:NEhook:one\xintCeil{\XINT_expr_unlock #3}\endcsname
2130 }%
2131 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2132 \def\XINT_iiexpr_func_ceil #1#2#3%
2133 {%
2134   \expandafter #1\expandafter #2\csname.=%
2135   \XINT:NEhook:one\xintiCeil{\XINT_expr_unlock #3}\endcsname
2136 }%
2137 \def\XINT_expr_func_sqr #1#2#3%
2138 {%
2139   \expandafter #1\expandafter #2\csname.=%
2140   \XINT:NEhook:one\xintSqr{\XINT_expr_unlock #3}\endcsname
2141 }%
2142 \def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}% revoir après
2143 \def\XINT_flexpr_func_sqr #1#2#3%
2144 {%
2145   \expandafter #1\expandafter #2\csname.=%
2146   \XINT:NEhook:one\XINTinFloatSqr{\XINT_expr_unlock #3}\endcsname
2147 }%
2148 \def\XINT_iiexpr_func_sqr #1#2#3%
2149 {%
2150   \expandafter #1\expandafter #2\csname.=%
2151   \XINT:NEhook:one\xintiiSqr{\XINT_expr_unlock #3}\endcsname
2152 }%
2153 \def\XINT_expr_func_? #1#2#3%
2154 {%
2155   \expandafter #1\expandafter #2\csname.=%
2156   \XINT:NEhook:one\xintiiIsNotZero{\XINT_expr_unlock #3}\endcsname
2157 }%
2158 \let\XINT_flexpr_func_? \XINT_expr_func_?
```

```

2159 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2160 \def\XINT_expr_func_! #1#2#3%
2161 {%
2162   \expandafter #1\expandafter #2\csname.=%
2163   \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2164 }%
2165 \let\XINT_flexpr_func_! \XINT_expr_func_!
2166 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2167 \def\XINT_expr_func_not #1#2#3%
2168 {%
2169   \expandafter #1\expandafter #2\csname.=%
2170   \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2171 }%
2172 \let\XINT_flexpr_func_not \XINT_expr_func_not
2173 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2174 \def\XINT_expr_func_odd #1#2#3%
2175 {%
2176   \expandafter #1\expandafter #2\csname.=%
2177   \XINT:NEhook:one\xintOdd{\XINT_expr_unlock #3}\endcsname
2178 }%
2179 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2180 \def\XINT_iiexpr_func_odd #1#2#3%
2181 {%
2182   \expandafter #1\expandafter #2\csname.=%
2183   \XINT:NEhook:one\xintiiOdd{\XINT_expr_unlock #3}\endcsname
2184 }%
2185 \def\XINT_expr_func_even #1#2#3%
2186 {%
2187   \expandafter #1\expandafter #2\csname.=%
2188   \XINT:NEhook:one\xintEven{\XINT_expr_unlock #3}\endcsname
2189 }%
2190 \let\XINT_flexpr_func_even\XINT_expr_func_even
2191 \def\XINT_iiexpr_func_even #1#2#3%
2192 {%
2193   \expandafter #1\expandafter #2\csname.=%
2194   \XINT:NEhook:one\xintiiEven{\XINT_expr_unlock #3}\endcsname
2195 }%
2196 % REVOIR nuple
2197 \def\XINT_expr_func_nuple #1#2#3%
2198   {\expandafter #1\expandafter #2\csname.=\XINT_expr_unlock #3\endcsname }%
2199 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2200 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
2201 \def\XINT_expr_func_factorial #1#2#3%
2202 {%
2203   \expandafter #1\expandafter #2\csname.=%
2204   \expandafter\XINT:expr:one:and:opt
2205   \romannumeral`&&@\XINT_expr_unlock#3,,!\xintFac\XINTinFloatFac
2206   \endcsname
2207 }%
2208 \def\XINT_flexpr_func_factorial #1#2#3%
2209 {%
2210   \expandafter #1\expandafter #2\csname.=%

```

Package *xintexpr* implementation

```
2211 \expandafter\XINT:expr:one:and:opt
2212 \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatFacdigits\XINTinFloatFac
2213 \endcsname
2214 }%
2215 \def\XINT_iiexpr_func_factorial #1#2#3%
2216 {%
2217 \expandafter #1\expandafter #2\csname.=%
2218 \XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #3}\endcsname
2219 }%
2220 \def\XINT_expr_func_sqrt #1#2#3%
2221 {%
2222 \expandafter #1\expandafter #2\csname.=%
2223 \expandafter\XINT:expr:one:and:opt
2224 \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatSqrtdigits\XINTinFloatSqrt
2225 \endcsname
2226 }%
2227 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2228 \def\XINT_iiexpr_func_sqrt #1#2#3%
2229 {%
2230 \expandafter #1\expandafter #2\csname.=%
2231 \XINT:NEhook:one\xintiiSqrt{\XINT_expr_unlock #3}\endcsname
2232 }%
2233 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2234 {%
2235 \expandafter #1\expandafter #2\csname.=%
2236 \XINT:NEhook:one\xintiiSqrtr{\XINT_expr_unlock #3}\endcsname
2237 }%
2238 \def\XINT_expr_func_round #1#2#3%
2239 {%
2240 \expandafter #1\expandafter #2\csname.=%
2241 \expandafter\XINT:expr:tacitzeroifonearg
2242 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound\xintRound
2243 \endcsname
2244 }%
2245 \let\XINT_flexpr_func_round\XINT_expr_func_round
2246 \def\XINT_iiexpr_func_round #1#2#3%
2247 {%
2248 \expandafter #1\expandafter #2\csname.=%
2249 \expandafter\XINT:iiexpr:tacitzeroifonearg
2250 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound
2251 \endcsname
2252 }%
2253 \def\XINT_expr_func_trunc #1#2#3%
2254 {%
2255 \expandafter #1\expandafter #2\csname.=%
2256 \expandafter\XINT:expr:tacitzeroifonearg
2257 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc\xintTrunc
2258 \endcsname
2259 }%
2260 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2261 \def\XINT_iiexpr_func_trunc #1#2#3%
2262 {%
```

Package *xintexpr* implementation

```
2263 \expandafter #1\expandafter #2\csname.=%
2264 \expandafter\XINT:iiexpr:tacitzeroifonearg
2265 \romannumeral`&&\XINT_expr_unlock #3,!\xintiTrunc
2266 \endcsname
2267 }%
2268 \def\XINT_expr_func_float #1#2#3%
2269 {%
2270 \expandafter #1\expandafter #2\csname.=%
2271 \expandafter\XINT:expr:one:and:opt
2272 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatdigits\XINTinFloat
2273 \endcsname
2274 }%
2275 \let\XINT_flexpr_func_float\XINT_expr_func_float
2276 % \XINT_iiexpr_func_float not defined
2277 \def\XINT_expr_func_divmod #1#2#3%
2278 {%
2279 \expandafter #1\expandafter #2\csname.=%
2280 \expandafter\XINT:expr:two:to:two
2281 \romannumeral`&&\XINT_expr_unlock #3,!\xintDivMod
2282 \endcsname
2283 }%
2284 \def\XINT_flexpr_func_divmod #1#2#3%
2285 {%
2286 \expandafter #1\expandafter #2\csname.=%
2287 \expandafter\XINT:flexpr:two:to:two
2288 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatDivMod
2289 \endcsname
2290 }%
2291 \def\XINT_iiexpr_func_divmod #1#2#3%
2292 {%
2293 \expandafter #1\expandafter #2\csname.=%
2294 \expandafter\XINT:expr:two:to:two
2295 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiDivMod
2296 \endcsname
2297 }%
2298 \def\XINT_expr_func_mod #1#2#3%
2299 {%
2300 \expandafter #1\expandafter #2\csname.=%
2301 \expandafter\XINT:expr:two:to:one
2302 \romannumeral`&&\XINT_expr_unlock #3,!\xintMod
2303 \endcsname
2304 }%
2305 \def\XINT_flexpr_func_mod #1#2#3%
2306 {%
2307 \expandafter #1\expandafter #2\csname.=%
2308 \expandafter\XINT:flexpr:two:to:one
2309 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatMod
2310 \endcsname
2311 }%
2312 \def\XINT_iiexpr_func_mod #1#2#3%
2313 {%
2314 \expandafter #1\expandafter #2\csname.=%
```

Package *xintexpr* implementation

```
2315 \expandafter\XINT:expr:two:to:one
2316 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiMod
2317 \endcsname
2318 }%
2319 \def\XINT_expr_func_binomial #1#2#3%
2320 {%
2321 \expandafter #1\expandafter #2\csname.=%
2322 \expandafter\XINT:expr:two:to:one
2323 \romannumeral`&&\XINT_expr_unlock #3,!\xintBinomial
2324 \endcsname
2325 }%
2326 \def\XINT_flexpr_func_binomial #1#2#3%
2327 {%
2328 \expandafter #1\expandafter #2\csname.=%
2329 \expandafter\XINT:flexpr:two:to:one
2330 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatBinomial
2331 \endcsname
2332 }%
2333 \def\XINT_iiexpr_func_binomial #1#2#3%
2334 {%
2335 \expandafter #1\expandafter #2\csname.=%
2336 \expandafter\XINT:expr:two:to:one
2337 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiBinomial
2338 \endcsname
2339 }%
2340 \def\XINT_expr_func_pfactorial #1#2#3%
2341 {%
2342 \expandafter #1\expandafter #2\csname.=%
2343 \expandafter\XINT:expr:two:to:one
2344 \romannumeral`&&\XINT_expr_unlock #3,!\xintPFactorial
2345 \endcsname
2346 }%
2347 \def\XINT_flexpr_func_pfactorial #1#2#3%
2348 {%
2349 \expandafter #1\expandafter #2\csname.=%
2350 \expandafter\XINT:flexpr:two:to:one
2351 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatPFactorial
2352 \endcsname
2353 }%
2354 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2355 {%
2356 \expandafter #1\expandafter #2\csname.=%
2357 \expandafter\XINT:expr:two:to:one
2358 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiPFactorial
2359 \endcsname
2360 }%
2361 \def\XINT_expr_func_randrange #1#2#3%
2362 {%
2363 \expandafter #1\expandafter #2\csname.=%
2364 \expandafter\XINT:expr:randrange
2365 \romannumeral`&&\XINT_expr_unlock #3,,!%
2366 \endcsname
```

```

2367 }%
2368 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
2369 \def\XINT_iiexpr_func_randrange #1#2#3%
2370 {%
2371   \expandafter #1\expandafter #2\csname.=%
2372   \expandafter\XINT:iiexpr:randrange
2373   \romannumeral`&&\XINT_expr_unlock #3,,!%
2374   \endcsname
2375 }%
2376 \def\XINT:expr:randrange #1,#2,#3!%
2377 {%
2378   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2379     \expandafter\xint_secondoftwo\fi
2380   {\xintiiRandRange{\XINT:NEhook:one\xintNum{#1}}}%
2381   {\xintiiRandRangeAtoB{\XINT:NEhook:one\xintNum{#1}}%
2382     {\XINT:NEhook:one\xintNum{#2}}}%
2383 }%
2384 \def\XINT:iiexpr:randrange #1,#2,#3!%
2385 {%
2386   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2387     \expandafter\xint_secondoftwo\fi
2388   {\xintiiRandRange{#1}}{\xintiiRandRangeAtoB{#1}{#2}}%
2389 }%
2390 \def\XINT_expr_func_quo #1#2#3%
2391 {%
2392   \expandafter #1\expandafter #2\csname.=%
2393   \expandafter\XINT:expr:two:to:one
2394   \romannumeral`&&\XINT_expr_unlock #3,! \xintiQuo
2395   \endcsname
2396 }%
2397 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
2398 \def\XINT_iiexpr_func_quo #1#2#3%
2399 {%
2400   \expandafter #1\expandafter #2\csname.=%
2401   \expandafter\XINT:expr:two:to:one
2402   \romannumeral`&&\XINT_expr_unlock #3,! \xintiiQuo
2403   \endcsname
2404 }%
2405 \def\XINT_expr_func_rem #1#2#3%
2406 {%
2407   \expandafter #1\expandafter #2\csname.=%
2408   \expandafter\XINT:expr:two:to:one
2409   \romannumeral`&&\XINT_expr_unlock #3,! \xintiRem
2410   \endcsname
2411 }%
2412 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2413 \def\XINT_iiexpr_func_rem #1#2#3%
2414 {%
2415   \expandafter #1\expandafter #2\csname.=%
2416   \expandafter\XINT:expr:two:to:one
2417   \romannumeral`&&\XINT_expr_unlock #3,! \xintiiRem
2418   \endcsname

```


Package *xintexpr* implementation

```
2419 }%
2420 \def\XINT_expr_func_gcd #1#2#3%
2421 {%
2422   \expandafter #1\expandafter #2\csname.=%
2423   \XINT:NEhook:csv\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname
2424 }%
2425 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
2426 \def\XINT_iiexpr_func_gcd #1#2#3%
2427 {%
2428   \expandafter #1\expandafter #2\csname.=%
2429   \XINT:NEhook:csv\xintiigCDof:csv{\XINT_expr_unlock #3}\endcsname
2430 }%
2431 \def\XINT_expr_func_lcm #1#2#3%
2432 {%
2433   \expandafter #1\expandafter #2\csname.=%
2434   \XINT:NEhook:csv\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname
2435 }%
2436 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2437 \def\XINT_iiexpr_func_lcm #1#2#3%
2438 {%
2439   \expandafter #1\expandafter #2\csname.=%
2440   \XINT:NEhook:csv\xintiilCMof:csv{\XINT_expr_unlock #3}\endcsname
2441 }%
2442 \def\XINT_expr_func_max #1#2#3%
2443 {%
2444   \expandafter #1\expandafter #2\csname.=%
2445   \XINT:NEhook:csv\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname
2446 }%
2447 \def\XINT_iiexpr_func_max #1#2#3%
2448 {%
2449   \expandafter #1\expandafter #2\csname.=%
2450   \XINT:NEhook:csv\xintiimaxof:csv{\XINT_expr_unlock #3}\endcsname
2451 }%
2452 \def\XINT_flexpr_func_max #1#2#3%
2453 {%
2454   \expandafter #1\expandafter #2\csname.=%
2455   \XINT:NEhook:csv\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname
2456 }%
2457 \def\XINT_expr_func_min #1#2#3%
2458 {%
2459   \expandafter #1\expandafter #2\csname.=%
2460   \XINT:NEhook:csv\xintMinof:csv{\XINT_expr_unlock #3}\endcsname
2461 }%
2462 \def\XINT_iiexpr_func_min #1#2#3%
2463 {%
2464   \expandafter #1\expandafter #2\csname.=%
2465   \XINT:NEhook:csv\xintiiminof:csv{\XINT_expr_unlock #3}\endcsname
2466 }%
2467 \def\XINT_flexpr_func_min #1#2#3%
2468 {%
2469   \expandafter #1\expandafter #2\csname.=%
2470   \XINT:NEhook:csv\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname
```

```

2471 }%
2472 \expandafter
2473 \def\csname XINT_expr_func_+\endcsname #1#2#3%
2474 {%
2475     \expandafter #1\expandafter #2\csname.=%
2476     \XINT:NEhook:csv\xintSum:csv{\XINT_expr_unlock #3}\endcsname
2477 }%
2478 \expandafter
2479 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2480 {%
2481     \expandafter #1\expandafter #2\csname.=%
2482     \XINT:NEhook:csv\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname
2483 }%
2484 \expandafter
2485 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2486 {%
2487     \expandafter #1\expandafter #2\csname.=%
2488     \XINT:NEhook:csv\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname
2489 }%
2490 \expandafter
2491 \def\csname XINT_expr_func_*\endcsname #1#2#3%
2492 {%
2493     \expandafter #1\expandafter #2\csname.=%
2494     \XINT:NEhook:csv\xintPrd:csv{\XINT_expr_unlock #3}\endcsname
2495 }%
2496 \expandafter
2497 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2498 {%
2499     \expandafter #1\expandafter #2\csname.=%
2500     \XINT:NEhook:csv\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname
2501 }%
2502 \expandafter
2503 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2504 {%
2505     \expandafter #1\expandafter #2\csname.=%
2506     \XINT:NEhook:csv\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname
2507 }%
2508 \def\XINT_expr_func_all #1#2#3%
2509 {%
2510     \expandafter #1\expandafter #2\csname.=%
2511     \XINT:NEhook:csv\xintANDof:csv{\XINT_expr_unlock #3}\endcsname
2512 }%
2513 \let\XINT_flexpr_func_all\XINT_expr_func_all
2514 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2515 \def\XINT_expr_func_any #1#2#3%
2516 {%
2517     \expandafter #1\expandafter #2\csname.=%
2518     \XINT:NEhook:csv\xintORof:csv{\XINT_expr_unlock #3}\endcsname
2519 }%
2520 \let\XINT_flexpr_func_any\XINT_expr_func_any
2521 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2522 \def\XINT_expr_func_xor #1#2#3%

```

```

2523 {%
2524   \expandafter #1\expandafter #2\csname.=%
2525   \XINT:NEhook:csv\xintXORof:csv{\XINT_expr_unlock #3}\endcsname
2526 }%
2527 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2528 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2529 \def\XINT_expr_func_len #1#2#3%
2530 {%
2531   \expandafter#1\expandafter#2\csname.=%
2532   \XINT:NEhook:csv\xintLength:f:csv{\XINT_expr_unlock#3}\endcsname
2533 }%
2534 \let\XINT_flexpr_func_len \XINT_expr_func_len
2535 \let\XINT_iiexpr_func_len \XINT_expr_func_len

```

1.2k has `\xintFirstItem:f:csv` for improved `\xintNewExpr` compatibility.

```

2536 \def\XINT_expr_func_first #1#2#3%
2537 {%
2538   \expandafter #1\expandafter #2\csname.=%
2539   \XINT:NEhook:csv\xintFirstItem:f:csv{\XINT_expr_unlock #3}\endcsname
2540 }%
2541 \let\XINT_flexpr_func_first\XINT_expr_func_first
2542 \let\XINT_iiexpr_func_first\XINT_expr_func_first

```

1.2k has `\xintLastItem:f:csv` for efficiency and improved `\xintNewExpr` compatibility.

```

2543 \def\XINT_expr_func_last #1#2#3%
2544 {%
2545   \expandafter #1\expandafter #2\csname.=%
2546   \XINT:NEhook:csv\xintLastItem:f:csv{\XINT_expr_unlock #3}\endcsname
2547 }%
2548 \let\XINT_flexpr_func_last\XINT_expr_func_last
2549 \let\XINT_iiexpr_func_last\XINT_expr_func_last

```

1.2c I hesitated but left the function "reversed" from 1.1 with this name, not "reverse". But the inner not public macro got renamed into `\xintReverse::csv`. 1.2g opts for the name `\xintReverse:f:csv`, and rewrites it for direct handling of csv lists. 2016/03/17.

```

2550 \def\XINT_expr_func_reversed #1#2#3%
2551 {%
2552   \expandafter #1\expandafter #2\csname.=%
2553   \XINT:NEhook:csv\xintReverse:f:csv{\XINT_expr_unlock #3}\endcsname
2554 }%
2555 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2556 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2557 \def\xintiiifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
2558 \def\XINT_expr_func_if #1#2#3%
2559 {%
2560   \expandafter #1\expandafter #2\csname.=%
2561   \expandafter\xintiiifNotZero:%
2562   \romannumeral &&\XINT_expr_unlock #3,\endcsname
2563 }%
2564 \let\XINT_flexpr_func_if\XINT_expr_func_if
2565 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2566 \def\xintifInt: #1,#2,#3,{\xintifInt{#1}{#2}{#3}}%

```

```

2567 \def\XINT_expr_func_ifint #1#2#3%
2568 {%
2569   \expandafter #1\expandafter #2\csname.=%
2570   \expandafter\xintifInt:%
2571   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2572 }%
2573 \let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint
2574 \def\xintifFloatInt: #1,#2,#3,{\xintifFloatInt{#1}{#2}{#3}}%
2575 \def\XINT_flexpr_func_ifint #1#2#3%
2576 {%
2577   \expandafter #1\expandafter #2\csname.=%
2578   \expandafter\xintifFloatInt:%
2579   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2580 }%
2581 \def\xintifOne: #1,#2,#3,{\xintifOne{#1}{#2}{#3}}%
2582 \def\XINT_expr_func_ifone #1#2#3%
2583 {%
2584   \expandafter #1\expandafter #2\csname.=%
2585   \expandafter\xintifOne:%
2586   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2587 }%
2588 \let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
2589 \def\xintiifOne: #1,#2,#3,{\xintiifOne{#1}{#2}{#3}}%
2590 \def\XINT_iiexpr_func_ifone #1#2#3%
2591 {%
2592   \expandafter #1\expandafter #2\csname.=%
2593   \expandafter\xintiifOne:%
2594   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2595 }%
2596 \def\xintiiifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
2597 \def\XINT_expr_func_ifsgn #1#2#3%
2598 {%
2599   \expandafter #1\expandafter #2\csname.=%
2600   \expandafter\xintiiifSgn:%
2601   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2602 }%
2603 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2604 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn

```

10.53 f-expandable versions of the `\xintSeqB::csv` and alike routines, for `\xintNewExpr`

10.53.1	<code>\xintSeqB:f:csv</code>	364
10.53.2	<code>\xintiiSeqB:f:csv</code>	365
10.53.3	<code>\XINTinFloatSeqB:f:csv</code>	366

10.53.1 `\xintSeqB:f:csv`

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```

2605 \def\xintSeqB:f:csv #1#2%
2606   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2607 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%

```

```

2608 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2609   \expandafter\xint_gobble_i\romannumeral`&&@%
2610   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2611   #1{#3}{#4}{#2}}%
2612 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2613 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2614   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2615 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2616   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2617 \def\XINT_seqb:f:csv_p #1#2%
2618 {%
2619   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2620   {#1}{#2}}%
2621 }%
2622 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2623 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2624 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2625   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2626 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2627   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2628 \def\XINT_seqb:f:csv_n #1#2%
2629 {%
2630   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2631   {#1}{#2}}%
2632 }%
2633 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2634 \def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%

```

10.53.2 \xintiiSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

2015/11/11. I correct a typo dating back to release 1.1 (2014/10/29): the macro name had a "b" rather than "B", hence was not functional (causing \xintNewIIExpr to fail on inputs such as #1..[1]..#2).

```

2635 \def\xintiiSeqB:f:csv #1#2%
2636   {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral`&&#2}{#1}}%
2637 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&#2#1!}%
2638 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2639   \expandafter\xint_gobble_i\romannumeral`&&@%
2640   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2641   \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2642   #1{#3}{#4}{#2}}%
2643 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2644   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2645 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2646   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2647 \def\XINT_iiseqb:f:csv_p #1#2%
2648 {%
2649   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2650   \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}}%
2651 }%

```

```

2652 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2653 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2654 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2655     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2656 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2657     {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2658 \def\XINT_iiseqb:f:csv_n #1#2%
2659 {%
2660     \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2661     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2662 }%

```

10.53.3 \XINTinFloatSeqB:f:csv

Produces in *f*-expandable way. If the step is zero, gives empty result except if start and end coincide. This is all for `\xintNewExpr`.

```

2663 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2664     {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2665 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#!}%
2666 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2667     \expandafter\xint_gobble_i\romannumeral`&&@%
2668     \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_b1\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2669     #1{#3}{#4}}{#2}}%
2670 \def\XINT_flseqb:f:csv_b1 #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2671     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2672 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2673     {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2674 \def\XINT_flseqb:f:csv_p #1#2%
2675 {%
2676     \xintifCmp {#1}{#2}%
2677     \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2678 }%
2679 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2680 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2681 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2682     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2683 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2684     {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2685 \def\XINT_flseqb:f:csv_n #1#2%
2686 {%
2687     \xintifCmp {#1}{#2}%
2688     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2689 }%

```

10.54 User defined functions: `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`

1.2c (November 11-12, 2015). It is possible to overload a variable name with a function name (and conversely). The function interpretation will be used only if followed by an opening parenthesis, disabling the tacit multiplication usually applied to variables. Crazy things such as `add(f(f), f=1..10)` are possible if there is a function "f". Or we can use "e" both for an exponential function and the Euler constant.

Package *xintexpr* implementation

2015/11/13: function candidates names first completely expanded, then detokenized and cleaned of spaces.

2015/11/21: no more `\detokenize` on the function names. Also I use `#1(#2)#3:=#4` rather than `#1(#2):=#3`. Ah, rather `#1(#2)#3=#4`, then I don't have to worry about active `:`.

2016/02/22: 1.2f la macro associée à la fonction ne débute plus par un `\romannumeral`, de toute façon est pour emploi dans `\csname..\endcsname`.

2016/03/08: 1.2f allows comma separated expressions; until then the user had to use explicit parentheses `\xintdeffunc foo(x,..):=(.., .., ..)\relax`.

```
2690 \catcode`: 12
2691 \catcode`~ 12
2692 \def\xINT_tmpa #1#2#3#4%
2693 {%
2694   \def #1##1(##2)##3=##4;{%
2695     \edef\xINT_expr_tmpa {##1}%
2696     \edef\xINT_expr_tmpa {\xint_zapspace_o \xINT_expr_tmpa}%
2697     \def\xINT_expr_tmpb {0}%
2698     \def\xINT_expr_tmpc {(##4)}%
2699     \xintFor ###1 in {##2} \do
2700       {\edef\xINT_expr_tmpb {\the\numexpr\xINT_expr_tmpb+\xint_c_i}%
2701        \edef\xINT_expr_tmpc {subs(\unexpanded\expandafter{\xINT_expr_tmpc},%
2702          ###1=#####\xINT_expr_tmpb)}%
2703       }%
2704     \expandafter\xINT_expr_defuserfunc
2705       \csname XINT_#2_func_\xINT_expr_tmpa\expandafter\endcsname
2706       \expandafter{\xINT_expr_tmpa}{#2}%
2707     \expandafter#3\csname XINT_#2_userfunc_\xINT_expr_tmpa\endcsname
2708       [\xINT_expr_tmpb]{\xINT_expr_tmpc}%
2709     \ifxintverbose\xintMessage {xintexpr}{Info}
2710       {Function \xINT_expr_tmpa\space for \string\xint #4 parser
2711        associated to \string\xINT_#2_userfunc_\xINT_expr_tmpa\space
2712        with meaning \expandafter\meaning
2713        \csname XINT_#2_userfunc_\xINT_expr_tmpa\endcsname}%
2714     \fi
2715   }%
2716 }%
2717 \catcode`: 11
2718 \XINT_tmpa\xintdeffunc {expr} \XINT_NewFunc {expr}%
2719 \XINT_tmpa\xintdefiifunc {iiexpr}\XINT_NewIIFunc {iiexpr}%
2720 \XINT_tmpa\xintdeffloatfunc{fexpr}\XINT_NewFloatFunc{floatexpr}%
2721 \def\xINT_expr_defuserfunc #1#2#3%
2722 {%
2723   \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2724     \csname .=\XINT:expr:userfunc{#3}{#2}{\XINT_expr_unlock ##3}\endcsname
2725   }%
2726 }%
2727 \def\xINT:expr:userfunc #1#2#3%
2728   {\csname XINT_#1_userfunc_#2\expandafter\endcsname
2729   \romannumeral0\xintcsvtolistnonstripped{#3}}%
2730 \def\xINT:newexpr:userfunc #1#2#3%
2731   {\~xintExpandArgs{XINT_#1_userfunc_#2}{\xintCSVtoListNonStripped{#3}}}%
```

10.55 \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`.

```

2732 \def\xINT_expr_wrapit #1{\expandafter\xINT_expr_wrap\csname.#1\endcsname}%
2733 \def\xintNewFunction #1#2[#3]#4%
2734 {%
2735   \edef\xINT_expr_tmpa {#1}%
2736   \edef\xINT_expr_tmpa {\xint_zapspace_o \xINT_expr_tmpa}%
2737   \def\xINT_expr_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
2738   \begingroup
2739     \ifcase #3\relax
2740       \toks0{}%
2741     \or \toks0{##1}%
2742     \or \toks0{##1##2}%
2743     \or \toks0{##1##2##3}%
2744     \or \toks0{##1##2##3##4}%
2745     \or \toks0{##1##2##3##4##5}%
2746     \or \toks0{##1##2##3##4##5##6}%
2747     \or \toks0{##1##2##3##4##5##6##7}%
2748     \or \toks0{##1##2##3##4##5##6##7##8}%
2749     \else \toks0{##1##2##3##4##5##6##7##8##9}%
2750     \fi
2751     \expandafter
2752   \endgroup
2753   \expandafter
2754   \def\csname XINT_expr_macrofunc_\xINT_expr_tmpa\expandafter\endcsname
2755   \the\toks0\expandafter{\xINT_expr_tmpb
2756     {\xINT_expr_wrapit{##1}}{\xINT_expr_wrapit{##2}}{\xINT_expr_wrapit{##3}}%
2757     {\xINT_expr_wrapit{##4}}{\xINT_expr_wrapit{##5}}{\xINT_expr_wrapit{##6}}%
2758     {\xINT_expr_wrapit{##7}}{\xINT_expr_wrapit{##8}}{\xINT_expr_wrapit{##9}}}%
2759   \expandafter\xINT_expr_newfunction
2760   \csname XINT_expr_func_\xINT_expr_tmpa\expandafter\endcsname
2761   \expandafter{\xINT_expr_tmpa}{eval}\xintbareeval
2762   \expandafter\xINT_expr_newfunction
2763   \csname XINT_iiexpr_func_\xINT_expr_tmpa\expandafter\endcsname
2764   \expandafter{\xINT_expr_tmpa}{iieval}\xintbareiieval
2765   \expandafter\xINT_expr_newfunction
2766   \csname XINT_flexpr_func_\xINT_expr_tmpa\expandafter\endcsname
2767   \expandafter{\xINT_expr_tmpa}{floateval}\xintbarefloateval
2768   \ifxintverbose
2769     \xintMessage {xintexpr}{Info}
2770     {Function \xINT_expr_tmpa\space for the expression parsers is
2771     associated to \string\xINT_expr_macrofunc_\xINT_expr_tmpa\space
2772     with meaning \expandafter\meaning
2773     \csname XINT_expr_macrofunc_\xINT_expr_tmpa\endcsname}%
2774   \fi
2775 }%
2776 \def\xINT_expr_newfunction #1#2#3#4%
2777 {%

```



```

2778 \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral0%
2779 \XINT:expr:macrofunc{#4}{#3}{#2}{\XINT:expr_unlock##3}}%
2780 }%
2781 \def\XINT:expr:macrofunc #1#2#3#4%
2782 {%
2783 #1\csname XINT:expr_macrofunc_#3\expandafter\endcsname
2784 \romannumeral0\xintcsvtolistnonstripped{#4}\relax
2785 }%
2786 \def\XINT:newexpr:macrofunc #1{%
2787 \def\XINT:newexpr:macrofunc ##1##2##3##4%
2788 {%
2789 \expandafter#1\csname.=~XINT:newexpr:macrofunc:a{##2}{##3}%
2790 {\xintCSVtoListNonStripped{##4}}\endcsname
2791 }%
2792 }\XINT:newexpr:macrofunc { }%
2793 \def\XINT:newexpr:macrofunc:a #1#2#3%
2794 {%
2795 \expandafter\XINT:expr_unlock\romannumeral0\csname xintbare#1\endcsname
2796 \csname XINT:expr_macrofunc_#2\endcsname#3\relax
2797 }%

```

10.56 `\xintNewExpr`, `\xintNewIExpr`, `\xintNewFloatExpr`, `\xintNewIIExpr`

10.56.1	<code>\xintApply::csv</code> and <code>\xintApply:::csv</code>	370
10.56.2	Mysterious stuff	371
10.56.3	<code>\XINT:expr_redefinemacros</code>	373
10.56.4	<code>\XINT:expr_redefineprints</code>	374
10.56.5	<code>\xintNewExpr</code> , ..., at last.	375

There was an `\xintNewExpr` already in 1.07 from May 2013, which was modified in September 2013 to work with the `#` macro parameter character, and then refactored into a more powerful version in June 2014 for 1.1 release of 2014/10/28. List handling causes special challenges, addressed by `\xintApply::csv`, `\xintApply:::csv`, ... next.

Comments finally added 2015/12/11 (with later edits):

The whole point is to expand completely macros when they have only numerical arguments and to inhibit this expansion if not. This is done in a recursive way: the catcode `12~` is used to register a macro name whose expansion must be inhibited. Any argument itself starting with such a `~` will force use of `~` for the macro which receives it.

In this context the catcode `12$` is used to signal a "virtual list" argument. It triggers insertion of `\xintApply::csv` or `\xintApply:::csv` for delayed handling later. This succeeds into handling inputs such as `[#1..[#2]..#3][#4:#5]...`

A final `\scantokens` converts the `"~"` prefixed names into real control sequences.

For this whole mechanism we need to have everything expressed using exclusively f-expandable macros. We avoid `\csname... \endcsname` like construct, but if absolutely needed perhaps we will do it ultimately.

For the iterating loops `seq`, `iter`, etc..., and dummy variables, we have no macros to our disposal to handle the case where the list of indices is not explicit. Moreover `omit`, `abort`, `break` can not work with non numerical data. Thus the whole mechanism is currently not applicable to them. It does work when the macro parameters (or variables for `\xintdeffunc`) do not intervene in the list of values to iterate over. But we can not delay expansion of dummy variables.

Comments added 2018/02/28:

At 1.3 of February 2018, there was important refactoring. Earlier, `\XINT:expr_redefinemacros` was a very big macro which made aliases of the dozens of macros (most from `xintfrac` and some defined

especially by *xintexpr* for acting on csv lists primarily) involved in the expression rendering and then redefined them all to expand to their original selves only when applied to purely numeric arguments. At 1.3 only very few such re-definitions are made, as what is redefined are a limited number of core wrapper macros.

Only when the original macros have one or two arguments is it examined if they can expand immediately (this includes case of function having possibly only one, or possibly two arguments). For macros applying to three or more or an undefined number of arguments, we don't complicate matters into checking if expansion is possible, and we delay that expansion automatically (but `if()` and `ifsgn()` do check if first argument is numeric and expand to suitable branch in that case).

In particular any function defined by `\xintdeffunc` or `\xintNewFunction` (it is then basically only a macro abstraction) when used in new function definitions will never be expanded immediately, because the detection of whether they are applied to only numerical data has not yet been added. (this might be added in future).

Some aspects of the 1.3 refactoring have made recursive definition via `\xintdeffunc` possible (of course they always were via `\xintNewFunction` as the latter is but a wrapper of a standard TeX macro definition, where *xintexpr* parsing is not at all involved).

A somewhat complicated layer (not modified at 1.3) is devoted to making possible the parsing of constructs such as `[#1..[#2]..#3][#4:#5]` or `[#1..#2]*#3` and it seems to work. At 1.3, even esoteric construct such as `[divmod(#1,#2)]*#3` is parsable by `\xintNewExpr`. (In `\xintNewFloatExpr`, don't forget `\empty` token so that square brackets are not mistaken for optional argument of `\xintthefloatexpr`; same for `\xintdeffloatfunc`.)

Side note: I wonder if I really had a good idea to define these list operations `[..]*foo` or `foo^[...]` which do not seem to occur in other languages with the meanings I used. And they caused me lots of efforts for support at `\xintNewExpr` level...

The catcode 12 dollar sign is used to signal when a macro can not be expanded but would produce a csv list. Furthermore some cases require f-expandable macros as the original code expanding in *xintexpr* is in `\csname` context and did not need f-expandability.

As mentioned above, currently syntax with dummy variables can not go through where the values iterated over are not explicit; and `omit`, `abort`, `break` mechanisms are not parsable with non purely numerical data, in part because they are not implemented internally via pure f-expansion.

10.56.1 `\xintApply::csv` and `\xintApply:::csv`

```

2798 \def\xintApply::csv #1#2%
2799   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
2800 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},}%
2801 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
2802 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2803   \else\expandafter\XINT_applyon::_d\fi #1}%
2804 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
2805 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}%
2806 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%
2807 \def\xintApply:::csv #1#2#3%
2808   {\expandafter\XINT_applyon::_a\expandafter{\romannumeral`&&@#2}{#1}{#3}}%
2809 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},}%
2810 \def\XINT_applyon::_b #1#2#3#4,%
2811   {\expandafter\XINT_applyon::_c \romannumeral`&&@#4,{#1}{#2}{#3}}%
2812 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2813   \else\expandafter\XINT_applyon::_d\fi #1}%
2814 \def\XINT_applyon::_d #1,#2#3%
2815   {\expandafter\XINT_applyon::_e\expandafter
2816     {\romannumeral`&&\xintApply:::csv {#2{#1}}{#3}},{#2}{#3}}%

```

```
2817 \def\XINT_applyon:::_e #1,#2#3#4{\XINT_applyon:::_b {#2}{#3}{#4, #1}}%
2818 \def\XINT_applyon:::_end #1,#2#3#4{\xint_secondoftwo #4}%
```

10.56.2 Mysterious stuff

~ and \$ of catcode 12 in what follows.

```
2819 \catcode`~ 12
2820 \catcode`$ 12 % $
2821 \def\xint_ddfork #1$$#2#3\krof {#2}% $$
2822 \def\XINT:NE:RApply:::csv #1#2#3#4%
2823   {\xintApply:::csv{\~expandafter #2~\xint_exchangetwo_keepbraces{#4}}{#3}}%
2824 \def\XINT:NE:LApply:::csv #1#2#3{\~xintApply:::csv{#2}{#3}}%
2825 \def\XINT:NE:RLApply:::csv #1{\~xintApply:::csv}%
2826 \def\XINT:NE:two#1{\XINT:NE:two_{#1}{\detokenize{#1}}}%
2827 \def\XINT:NE:two_#1#2#3#4%
2828   {\expandafter\XINT:NE:two_a\romannumeral`&&@#4!{#3}{#1}{#2}}%
2829 \def\XINT:NE:two_a#1#2!#3#4#5%
2830   {\expandafter\XINT:NE:two_b\romannumeral`&&@#3!#1{#4}{#5}{#1#2}}%
2831 \def\XINT:NE:two_b#1#2!#3#4#5{\XINT:NE:two_fork_dd#1#3{#4}{#5}{#1#2}}%
2832 \def\XINT:NE:two_fork_dd #1#2{%
2833   \xint_ddfork
2834   #1#2\XINT:NE:RLApply:::csv
2835   #1$\XINT:NE:RApply:::csv% $
2836   $#2\XINT:NE:LApply:::csv% $
2837   $${\XINT:NE:two_fork_nn #1#2}% $$
2838   \krof
2839 }%
2840 \def\XINT:NE:two_fork_nn #1#2#3#4{%
2841   \if #1#\xint_dothis{#4}\fi
2842   \if #1~\xint_dothis{#4}\fi
2843   \if #2#\xint_dothis{#4}\fi
2844   \if #2~\xint_dothis{#4}\fi
2845   \xint_orthat{#3}%
2846 }%
2847 \def\XINT:NE:one#1#2{\expandafter\XINT:NE:one_a\romannumeral`&&@#2!#1}%
2848 \def\XINT:NE:one_a#1#2!#3{%
2849   \if ###1\xint_dothis {\detokenize{#3}}\fi
2850   \if ~#1\xint_dothis {\detokenize{#3}}\fi
2851   \if $#1\xint_dothis {\~xintApply:::csv{\detokenize{#3}}}\fi %$
2852   \xint_orthat #3{#1#2}%
2853 }%
2854 \def\XINT:NE:oneopt#1[#2]#3%
2855   {\expandafter\XINT:NE:oneopt_a\romannumeral`&&@#3!{#2}#1}%
2856 \def\XINT:NE:oneopt_a#1#2!#3#4%
2857   {\expandafter\XINT:NE:oneopt_b\romannumeral`&&@#3!#1#4{#1#2}}%
2858 \def\XINT:NE:oneopt_b#1#2!#3#4%
2859   {\expandafter\XINT:NE:oneopt_fork#1#3#4{#1#2}}%
2860 \def\XINT:NE:oneopt_fork#1#2#3#4{%
2861   \if1\if###11\else\if~#11\else\if###21\else\if~#21\else0\fi\fi\fi\fi
2862   \xint_dothis {\detokenize{#3}[#4]}\fi
2863   \if $#2\xint_dothis {\~xintApply:::csv{\detokenize{#3}[#4]}\fi %$
2864   \xint_orthat{#3[#4]}%
2865 }% pas complètement général, mais bon
```

Package *xintexpr* implementation

```

2866 \def\XINT:NE:csv #1{\detokenize{#1}}% radicalement fainéant
2867 \def\XINT:newexpr:one:and:opt #1,#2,#3!#4#5%
2868 {%
2869   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2870     \expandafter\xint_secondoftwo\fi
2871   {\XINT:NE:one#4}{\XINT:NE:oneopt#5[\XINT:NE:one\xintNum{#2}]}{#1}%
2872 }%
2873 \def\XINT:newexpr:tacitzeroifonearg #1,#2,#3!#4#5%
2874 {%
2875   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2876     \expandafter\xint_secondoftwo\fi
2877   {\XINT:NE:two#4{0}}{\XINT:NE:two#5{\XINT:NE:one\xintNum{#2}}}{#1}%
2878 }%
2879 \def\XINT:newiexpr:tacitzeroifonearg #1,#2,#3!#4%
2880 {%
2881   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2882     \expandafter\xint_secondoftwo\fi
2883   {\XINT:NE:two#4{0}}{\XINT:NE:two#4{#2}}{#1}%
2884 }%
2885 \def\XINT:newexpr:insertdollar~{\$noexpand$}%
2886 \def\XINT:newexpr:two:to:two #1,#2,!#3%
2887 {%
2888   \XINT:NE:two_
2889   {\expandafter\XINT:expr:totwo\romannumeral`&&@#3}%
2890   {\$noexpand$\expandafter~\XINT:expr:totwo~\romannumeral-`0\detokenize{#3}}%
2891   {#1}{#2}%
2892 }%
2893 \def\XINT:newflexpr:two:to:two #1,#2,!#3%
2894 {%
2895   \XINT:NE:two_
2896   {#3}%
2897   {\expandafter\XINT:newexpr:insertdollar\detokenize{#3}}%
2898   {#1}{#2}%
2899 }%
2900 \def\XINT:newexpr:two:to:one #1,#2,!#3%
2901 {%
2902   \XINT:NE:two#3{#1}{#2}%
2903 }%
2904 \let\XINT:newflexpr:two:to:one\XINT:newexpr:two:to:one
2905 \def\xintiiifNotZeroNE:#1#2,#3,#4,%
2906 {%
2907   \if1\if###11\else\if~#11\else\if$#11\else0%$
2908     \fi\fi\fi
2909   \xint_dothis{~xintiiifNotZero}\fi
2910   \xint_orthat\xintiiifNotZero
2911   {#1#2}{#3}{#4}%
2912 }%
2913 \def\xintifIntNE:#1#2,#3,#4,%
2914 {%
2915   \if1\if###11\else\if~#11\else\if$#11\else0%$
2916     \fi\fi\fi
2917   \xint_dothis{~xintifInt}\fi

```

```

2918 \xint_orthat\xintifInt
2919   {#1#2}{#3}{#4}%
2920 }%
2921 \def\xintifFloatIntNE:#1#2,#3,#4,%
2922 {%
2923   \if1\if###11\else\if~#11\else\if$#11\else0%$
2924     \fi\fi\fi
2925   \xint_dothis{~xintifFloatInt}\fi
2926   \xint_orthat\xintifFloatInt
2927   {#1#2}{#3}{#4}%
2928 }%
2929 \def\xintiiifOneNE:#1#2,#3,#4,%
2930 {%
2931   \if1\if###11\else\if~#11\else\if$#11\else0%$
2932     \fi\fi\fi
2933   \xint_dothis{~xintiiifOne}\fi
2934   \xint_orthat\xintiiifOne
2935   {#1#2}{#3}{#4}%
2936 }%
2937 \def\xintifOneNE:#1#2,#3,#4,%
2938 {%
2939   \if1\if###11\else\if~#11\else\if$#11\else0%$
2940     \fi\fi\fi
2941   \xint_dothis{~xintifOne}\fi
2942   \xint_orthat\xintifOne
2943   {#1#2}{#3}{#4}%
2944 }%
2945 \def\xintiiifSgnNE:#1#2,#3,#4,#5,%
2946 {%
2947   \if1\if###11\else\if~#11\else\if$#11\else0%$
2948     \fi\fi\fi
2949   \xint_dothis{~xintiiifSgn}\fi
2950   \xint_orthat\xintiiifSgn
2951   {#1#2}{#3}{#4}{#5}%
2952 }%

```

10.56.3 `\XINT_expr_redefinemacros`

Completely refactored at 1.3.

```

2953 \def\XINT_expr_redefinemacros {%
2954   \let\XINT:NEhook:one\XINT:NE:one
2955   \let\XINT:NEhook:two\XINT:NE:two
2956   \let\XINT:NEhook:csv\XINT:NE:csv
2957   \let\XINT:expr:one:and:opt      \XINT:newexpr:one:and:opt
2958   \let\XINT:expr:one:or:two:nums  \XINT:newexpr:one:or:two:nums
2959   \let\XINT:iiexpr:one:or:two:    \XINT:newiiexpr:one:or:two:
2960   \let\XINT:expr:tacitzeroifonearg \XINT:newexpr:tacitzeroifonearg
2961   \let\XINT:iiexpr:tacitzeroifonearg \XINT:newiiexpr:tacitzeroifonearg
2962   \let\XINT:expr:two:to:two      \XINT:newexpr:two:to:two
2963   \let\XINT:expr:two:to:one      \XINT:newexpr:two:to:one
2964   \let\XINT:flexpr:two:to:one     \XINT:newflexpr:two:to:one
2965   \let\XINT:expr:two:to:one      \XINT:newexpr:two:to:one

```

```

2966 \let\XINT:flexpr:two:to:two \XINT:newflexpr:two:to:two
2967 \let\XINT:flexpr:two:to:one \XINT:newflexpr:two:to:one
2968 \let\xintiiifNotZero: \xintiiifNotZeroNE:
2969 \let\xintifInt: \xintifIntNE:
2970 \let\xintifFloatInt: \xintifFloatIntNE:
2971 \let\xintiiifOne: \xintiiifOneNE:
2972 \let\xintifOne: \xintifOneNE:
2973 \let\xintiiifSgn: \xintiiifSgnNE:
2974 \let\xintSeqNumeric::csv \xintSeq::csv
2975 \let\xintiiSeqNumeric::csv \xintiiSeq::csv
2976 \let\XINTinFloatSeqNumeric::csv \XINTinFloatSeq::csv
2977 \let\xintSeqBNumeric::csv \xintSeqB::csv
2978 \let\xintiiSeqBNumeric::csv \xintiiSeqB::csv
2979 \let\XINTinFloatSeqBNumeric::csv\XINTinFloatSeqB::csv
2980 \def\xintSeq::csv
2981   {\XINT:NE:two_\xintSeqNumeric::csv{$noexpand\xintSeq::csv}}%
2982 \def\xintiiSeq::csv
2983   {\XINT:NE:two_\xintiiSeqNumeric::csv{$noexpand\xintiiSeq::csv}}%
2984 \def\XINTinFloatSeq::csv
2985   {\XINT:NE:two_\XINTinFloatSeqNumeric::csv{$noexpand\XINTinFloatSeq::csv}}%
2986 \def\xintSeqB::csv
2987   {\XINT:NE:two_\xintSeqBNumeric::csv{$noexpand\xintSeqB:f:csv}}%
2988 \def\xintiiSeqB::csv
2989   {\XINT:NE:two_\xintiiSeqBNumeric::csv{$noexpand\xintiiSeqB:f:csv}}%
2990 \def\XINTinFloatSeqB::csv
2991   {\XINT:NE:two_\XINTinFloatSeqBNumeric::csv{$noexpand\XINTinFloatSeqB:f:csv}}%
2992 \def\xintListSel:x:csv {\~xintListSel:f:csv }%
2993 \let\XINT:expr:userfunc \XINT:newexpr:userfunc
2994 \let\XINT:expr:macrofunc\XINT:newexpr:macrofunc
2995 \def\XINTinRandomFloatSdigits{\~XINTinRandomFloatSdigits }%
2996 \def\XINTinRandomFloatSixteen{\~XINTinRandomFloatSixteen }%
2997 \def\xintiiRandRange{\~xintiiRandRange }%
2998 \def\xintiiRandRangeAtoB{\~xintiiRandRangeAtoB }%
2999 }%

```

10.56.4 \XINT_expr_redefineprints

This is used by `\xintNewExpr` but not by `\xintdeffunc`.

```

3000 \def\XINT_expr_redefineprints
3001 {%
3002   \def\XINT_flexpr_noopt
3003     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
3004   \def\XINT_flexpr_withopt_b ##1##2%
3005     {\expandafter\XINT_flexpr_wrap\csize .;##1.=\XINT_expr_unlock ##2\endcsize }%
3006   \def\XINT_expr_unlock_sp ##1.;##2##3.##4!%
3007     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
3008     \XINTdigits{##2##3}{##4}}%
3009   \def\XINT_expr_print ##1%
3010     {\expandafter\xintSPRaw::csv\expandafter
3011     {\romannumeral`&&\XINT_expr_unlock ##1}}%
3012   \def\XINT_iiexpr_print ##1%
3013     {\expandafter\xintCSV::csv\expandafter

```

Package *xintexpr* implementation

```
3014         {\romannumeral`&&\XINT_expr_unlock ##1}}%
3015 \def\XINT_boolexpr_print ##1%
3016     {\expandafter\xintIsTrue::csv\expandafter
3017         {\romannumeral`&&\XINT_expr_unlock ##1}}%
3018 \def\xintCSV::csv    {\~xintCSV::csv    }%
3019 \def\xintSPRaw::csv  {\~xintSPRaw::csv  }%
3020 \def\xintPFloat::csv {\~xintPFloat::csv }%
3021 \def\xintIsTrue::csv {\~xintIsTrue::csv }%
3022 \def\xintRound::csv  {\~xintRound::csv  }%
3023 }%
```

10.56.5 `\xintNewExpr`, ..., at last.

1.2c modifications to accomodate `\XINT_expr_deffunc_newexpr` etc..

1.2f adds token `\XINT_newexpr_clean` to be able to have a different `\XINT_newfunc_clean`.

```
3024 \def\xintNewExpr      {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3025                       \xinttheexpr\XINT_newexpr_clean}%
3026 \def\xintNewFloatExpr{\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3027                       \xintthefloatexpr\XINT_newexpr_clean}%
3028 \def\xintNewIExpr     {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3029                       \xinttheiexpr\XINT_newexpr_clean}%
3030 \def\xintNewIIExpr    {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3031                       \xinttheiiexpr\XINT_newexpr_clean}%
3032 \def\xintNewBoolExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3033                       \xinttheboolexpr\XINT_newexpr_clean}%
3034 \def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&@}%
```

1.2c for `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`.

At 1.3, `NewFunc` does not use a comma delimited pattern anymore.

```
3035 \def\XINT_NewFunc
3036   {\XINT_NewExpr}\xint_gobble_i\xintthebareeval\XINT_newfunc_clean}%
3037 \def\XINT_NewFloatFunc
3038   {\XINT_NewExpr}\xint_gobble_i\xintthebarefloateval\XINT_newfunc_clean}%
3039 \def\XINT_NewIIFunc
3040   {\XINT_NewExpr}\xint_gobble_i\xintthebareiieval\XINT_newfunc_clean}%
3041 \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.

```
3042 \def\XINT_NewExpr #1#2#3#4#5#6[#7]%
3043 {%
3044   \begingroup
3045     \ifcase #7\relax
3046       \toks0 {\endgroup\def#5}%
3047     \or \toks0 {\endgroup\def#5##1}%
3048     \or \toks0 {\endgroup\def#5##1##2}%
3049     \or \toks0 {\endgroup\def#5##1##2##3}%
3050     \or \toks0 {\endgroup\def#5##1##2##3##4}%
3051     \or \toks0 {\endgroup\def#5##1##2##3##4##5}%
3052     \or \toks0 {\endgroup\def#5##1##2##3##4##5##6}%
3053     \or \toks0 {\endgroup\def#5##1##2##3##4##5##6##7}%
```

Package *xintexpr* implementation

```
3054 \or \toks0 {\endgroup\def#5##1##2##3##4##5##6##7##8}%
3055 \or \toks0 {\endgroup\def#5##1##2##3##4##5##6##7##8##9}%
3056 \fi
3057 \xintexprSafeCatcodes
3058 \XINT_expr_redefinemacros
3059 #1%
3060 \XINT_NewExpr_a #2#3#4#5%
3061 }%
```

1.2d's `\xintNewExpr` makes a local definition. In earlier releases, the definition was global.

```
3062 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
3063 \def\xintNewExpr_a #1%2%3%4%5@
3064 {@
3065 \def\xint_tmpa %1%2%3%4%5%6%7%8%9{%5}@
3066 \def~{$noexpand$}@
3067 \catcode` : 11 \catcode`_ 11
3068 \catcode`# 12 \catcode`~ 13 \escapechar 126
3069 \endlinechar -1 \everyeof {\noexpand }@
3070 \edef\xint_tmpb
3071 {\scantokens\expandafter{\romannumeral`&&@\expandafter
3072 %2\xint_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
3073 }@
3074 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
3075 \edef\xint_tmpa %1%2%3%4%5%6%7%8%9@
3076 {\scantokens\expandafter{\expandafter%3\meaning\xint_tmpb}}@
3077 \the\toks0\expandafter
3078 {\xint_tmpa{%%1}{%%2}{%%3}{%%4}{%%5}{%%6}{%%7}{%%8}{%%9}}@
3079 %1{\ifxintverbose
3080 \xintMessage{xintexpr}{Info}@
3081 {\string%4\space now with meaning \meaning%4}@
3082 \fi}@
3083 }@
3084 \catcode`% 14
3085 \let\xintexprRestoreCatcodes\empty
3086 \def\xintexprSafeCatcodes
3087 {%
3088 \edef\xintexprRestoreCatcodes {%
3089 \catcode59=\the\catcode59 % ;
3090 \catcode34=\the\catcode34 % "
3091 \catcode63=\the\catcode63 % ?
3092 \catcode124=\the\catcode124 % |
3093 \catcode38=\the\catcode38 % &
3094 \catcode33=\the\catcode33 % !
3095 \catcode93=\the\catcode93 % ]
3096 \catcode91=\the\catcode91 % [
3097 \catcode94=\the\catcode94 % ^
3098 \catcode95=\the\catcode95 % _
3099 \catcode47=\the\catcode47 % /
3100 \catcode41=\the\catcode41 % )
3101 \catcode40=\the\catcode40 % (
3102 \catcode42=\the\catcode42 % *
```


Package *xintexpr* implementation

```

3103     \catcode43=\the\catcode43 % +
3104     \catcode62=\the\catcode62 % >
3105     \catcode60=\the\catcode60 % <
3106     \catcode58=\the\catcode58 % :
3107     \catcode46=\the\catcode46 % .
3108     \catcode45=\the\catcode45 % -
3109     \catcode44=\the\catcode44 % ,
3110     \catcode61=\the\catcode61 % =
3111     \catcode96=\the\catcode96 % `
3112     \catcode32=\the\catcode32\relax % space
3113 }%
3114     \catcode59=12 % ;
3115     \catcode34=12 % "
3116     \catcode63=12 % ?
3117     \catcode124=12 % |
3118     \catcode38=4 % &
3119     \catcode33=12 % !
3120     \catcode93=12 % ]
3121     \catcode91=12 % [
3122     \catcode94=7 % ^
3123     \catcode95=8 % _
3124     \catcode47=12 % /
3125     \catcode41=12 % )
3126     \catcode40=12 % (
3127     \catcode42=12 % *
3128     \catcode43=12 % +
3129     \catcode62=12 % >
3130     \catcode60=12 % <
3131     \catcode58=12 % :
3132     \catcode46=12 % .
3133     \catcode45=12 % -
3134     \catcode44=12 % ,
3135     \catcode61=12 % =
3136     \catcode96=12 % `
3137     \catcode32=10 % space
3138 }%
3139 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
3140 \XINT_restorecatcodes_endinput%

xintkernel: 552. Total number of code lines: 14373. (but 3325 lines among them
xinttools:1454. start either with {% or with }%.)
xintcore:2183. Each package starts with circa 50 lines dealing with cat-
xint:1496. codes, package identification and reloading management,
xintbinhex: 472. also for Plain TEX. Version 1.3b of 2018/05/18.
xintgcd: 434.
xintfrac:3227.
xintseries: 386.
xintcfraction:1029.
xintexpr:3140.

```