

# The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.1a (2014/11/07); documentation date: 2015/03/07.

From source file `xint.dtx`. Time-stamp: <07-03-2015 at 20:06:50 CET>.

## Contents

1	Package <code>xintkernel</code> implementation . . . . .	1
2	Package <code>xinttools</code> implementation . . . . .	7
3	Package <code>xintcore</code> implementation . . . . .	32
4	Package <code>xint</code> implementation . . . . .	81
5	Package <code>xintbinhex</code> implementation . . . . .	119
6	Package <code>xintgcd</code> implementation . . . . .	131
7	Package <code>xintfrac</code> implementation . . . . .	143
8	Package <code>xintseries</code> implementation . . . . .	200
9	Package <code>xintcfrac</code> implementation . . . . .	210
10	Package <code>xintexpr</code> implementation . . . . .	232

This is 1.1a of 2014/11/07.

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.

`xinttools` is not loaded anymore by `xint`, nor by `xintfrac`. It is loaded by `xintexpr`.

## 1 Package `xintkernel` implementation

.1	Catcodes, $\varepsilon$ -TEX and reload detection . . . . .	2	.7	<code>\xint_dothis</code> , <code>\xint_orthat</code> . . . . .	5
.2	Package identification . . . . .	4	.8	<code>\xint_zapspace</code> . . . . .	5
.3	Token management utilities . . . . .	4	.9	Constants . . . . .	6
.4	gob til macros and UD style fork . . . . .	5	.10	<code>\odef</code> , <code>\oodef</code> , <code>\fdef</code> . . . . .	6
.5	<code>\xint_afterfi</code> . . . . .	5	.11	<code>\xintReverseOrder</code> . . . . .	6
.6	<code>\xint_bye</code> . . . . .	5	.12	<code>\xintLength</code> . . . . .	7

This package provides the common minimal code base for loading management and catcode control and also a few programming utilities. It is loaded by both `xintcore.sty` and `xinttools.sty` hence by all other packages.

First appeared as a separate package with release 1.1.

## 1.1 Catcodes, $\epsilon$ -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.

Starting with version 1.06 of the package, also ``` must be catcode-protected, because we replace everywhere in the code the twice-expansion done with `\expandafter` by the systematic use of `\roman\numeral-`0`.

Starting with version 1.06b I decide that I suffer from an indigestion of @ signs, so I replace them all with underscores `_`, à la  $\TeX$ 3.

Release 1.09b is more economical: some macros are defined already in `xint.sty` (now in `xintkernel.sty`) and re-used in other modules. All catcode changes have been unified and `\XINT_storecatcodes` will be used by each module to redefine `\XINT_restorecatcodes_endinput` in case catcodes have changed in-between the loading of `xint.sty` (now `xintkernel.sty`) and the module (not very probable but...).

```

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 \expandafter
19 \ifx\csname numexpr\endcsname\relax
20 \y{xintkernel}{\numexpr not available, aborting input}%
21 \aftergroup\endinput
22 \else
23 \expandafter
24 \ifx\csname XINTsetupcatcodes\endcsname\relax
25 \else
26 \y{xintkernel}{I was already loaded, aborting input}%
27 \aftergroup\endinput
28 \fi
29 \fi
30 \def\SetCatcodesIfInputNotAborted
31 {%
32 \endgroup
33 \def\XINT_restorecatcodes
34 {% takes care of all, to allow more economical code in modules
35 \catcode59=\the\catcode59 % ; xintexpr
36 \catcode126=\the\catcode126 % ~ xintexpr
37 \catcode39=\the\catcode39 % ' xintexpr
38 \catcode34=\the\catcode34 % " xintbinhex, and xintexpr
39 \catcode63=\the\catcode63 % ? xintexpr

```

## 1 Package *xintkernel* implementation

```

40     \catcode124=\the\catcode124 % | xintexpr
41     \catcode38=\the\catcode38  % & xintexpr
42     \catcode64=\the\catcode64  % @ xintexpr
43     \catcode33=\the\catcode33  % ! xintexpr
44     \catcode93=\the\catcode93  % ] -, xintfrac, xintseries, xintcfrac
45     \catcode91=\the\catcode91  % [ -, xintfrac, xintseries, xintcfrac
46     \catcode36=\the\catcode36  % $ xintgcd only
47     \catcode94=\the\catcode94  % ^
48     \catcode96=\the\catcode96  % `
49     \catcode47=\the\catcode47  % /
50     \catcode41=\the\catcode41  % )
51     \catcode40=\the\catcode40  % (
52     \catcode42=\the\catcode42  % *
53     \catcode43=\the\catcode43  % +
54     \catcode62=\the\catcode62  % >
55     \catcode60=\the\catcode60  % <
56     \catcode58=\the\catcode58  % :
57     \catcode46=\the\catcode46  % .
58     \catcode45=\the\catcode45  % -
59     \catcode44=\the\catcode44  % ,
60     \catcode35=\the\catcode35  % #
61     \catcode95=\the\catcode95  % _
62     \catcode125=\the\catcode125 % }
63     \catcode123=\the\catcode123 % {
64     \endlinechar=\the\endlinechar
65     \catcode13=\the\catcode13  % ^^M
66     \catcode32=\the\catcode32  %
67     \catcode61=\the\catcode61\relax % =
68 }%
69 \edef\XINT_restorecatcodes_endinput
70 {%
71     \XINT_restorecatcodes\noexpand\endinput %
72 }%
73 \def\XINT_setcatcodes
74 {%
75     \catcode61=12 % =
76     \catcode32=10 % space
77     \catcode13=5 % ^^M
78     \endlinechar=13 %
79     \catcode123=1 % {
80     \catcode125=2 % }
81     \catcode95=11 % _ LETTER
82     \catcode35=6 % #
83     \catcode44=12 % ,
84     \catcode45=12 % -
85     \catcode46=12 % .
86     \catcode58=11 % : LETTER
87     \catcode60=12 % <
88     \catcode62=12 % >
89     \catcode43=12 % +
90     \catcode42=12 % *
91     \catcode40=12 % (

```

## 1 Package `xintkernel` implementation

```
92     \catcode41=12 % )
93     \catcode47=12 % /
94     \catcode96=12 % `
95     \catcode94=11 % ^ LETTER
96     \catcode36=3  % $
97     \catcode91=12 % [
98     \catcode93=12 % ]
99     \catcode33=11 % ! LETTER
100    \catcode64=11 % @ LETTER
101    \catcode38=12 % &
102    \catcode124=12 % |
103    \catcode63=11 % ? LETTER
104    \catcode34=12 % "
105    \catcode39=12 % '
106    \catcode126=3  % ~
107    \catcode59=12 % ;
108    }%
109    \XINT_setcatcodes
110 }%
111 \SetCatcodesIfInputNotAborted
    Other modules could possibly be loaded under a different catcode regime.
112 \def\XINTsetupcatcodes {% for use by other modules
113     \edef\XINT_restorecatcodes_endinput
114     {%
115         \XINT_restorecatcodes\noexpand\endinput %
116     }%
117     \XINT_setcatcodes
118 }%
```

## 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 extra precautions.

1.09c uses e-TeX `\ifdefined`.

```
119 \ifdefined\ProvidesPackage
120   \let\XINT_providespackage\relax
121 \else
122   \def\XINT_providespackage #1#2[#3]%
123       {\immediate\write-1{Package: #2 #3}%
124       \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
125 \fi
126 \XINT_providespackage
127 \ProvidesPackage {xintkernel}%
128 [2014/11/07 v1.1a Paraphernalia for the xint packages (jfb)]%
```

## 1.3 Token management utilities

```
129 \long\def\xint_gobble_   {}%
130 \long\def\xint_gobble_i  #1{}%
131 \long\def\xint_gobble_ii #1#2{}%
```

## 1 Package `xintkernel` implementation

```
132 \long\def\xint_gobble_iii #1#2#3{%
133 \long\def\xint_gobble_iv #1#2#3#4{%
134 \long\def\xint_gobble_v #1#2#3#4#5{%
135 \long\def\xint_gobble_vi #1#2#3#4#5#6{%
136 \long\def\xint_gobble_vii #1#2#3#4#5#6#7{%
137 \long\def\xint_gobble_viii #1#2#3#4#5#6#7#8{%
138 \long\def\xint_firstofone #1{#1}%
139 \long\def\xint_firstoftwo #1#2{#1}%
140 \long\def\xint_secondoftwo #1#2{#2}%
141 \long\def\xint_firstofone_thenstop #1{ #1}%
142 \long\def\xint_firstoftwo_thenstop #1#2{ #1}%
143 \long\def\xint_secondoftwo_thenstop #1#2{ #2}%
```

### 1.4 gob til macros and UD style fork

```
144 \def\xint_gob_til_zero #10{%
145 \def\xint_UDzerominusfork #10-#2#3\krof {#2}%
146 \long\def\xint_gob_til_R #1\R {}%
147 \long\def\xint_gob_til_W #1\W {}%
148 \long\def\xint_gob_til_Z #1\Z {}%
149 \let\xint_relax\relax
150 \def\xint_brelax {\xint_relax }%
151 \long\def\xint_gob_til_xint_relax #1\xint_relax {}%
```

### 1.5 `\xint_afterfi`

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

### 1.6 `\xint_bye`

```
153 \long\def\xint_bye #1\xint_bye {}%
```

### 1.7 `\xint_dothis`, `\xint_orthat`

New with 1.1. Used as `\if..\xint_dothis{..}\fi` <multiple times> followed by `\xint_orthat{...}`. To be used with less probable things first.

```
154 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% v1.1
155 \let\xint_orthat \xint_firstofone
```

### 1.8 `\xint_zapspaces`

1.1. Zaps leading, intermediate, trailing, spaces in completely expanding context (`\edef`, `\csname...\endcsname`) To be used as

```
\xint_zapspaces foo \xint_gobble_i notice the mandatory space after foo
```

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

Explanation: if there are leading spaces, then the first #1 will be empty, and the first #2 being undelimited will be stripped from all the remaining leading spaces, if there was more than one to start with. Of course brace-stripping may occur. And this iterates: each time a #2 is removed, either we then have spaces and next #1 will be empty, or we have no spaces and #1 will end at the first space. Ultimately #2 will be `\xint_gobble_i`.

Code comment from 1.1 release said to do:

```
\xint_zapspaces foo \xint_bye\xint_bye
```

## 1 Package *xintkernel* implementation

perhaps because it was pretty. It works also, but `\xint_gobble_i` is one token less. Compatible with an empty `foo`.

```
156 \def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% v1.1
```

### 1.9 Constants

```
157 \chardef\xint_c_      0
158 \chardef\xint_c_i    1
159 \chardef\xint_c_ii   2
160 \chardef\xint_c_iii  3
161 \chardef\xint_c_iv   4
162 \chardef\xint_c_v    5
163 \chardef\xint_c_vi   6
164 \chardef\xint_c_vii  7
165 \chardef\xint_c_viii 8
```

### 1.10 `\odef`, `\oodef`, `\fdef`

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

```
166 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
167 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
168     \expandafter\expandafter\expandafter#1%
169     \expandafter\expandafter\expandafter }%
170 \def\xintfdef #1#2{\expandafter\def\expandafter#1\expandafter
171     {\romannumeral-`0#2}}%
172 \ifdefined\odef\else\let\odef\xintodef\fi
173 \ifdefined\oodef\else\let\oodef\xintoodef\fi
174 \ifdefined\fdef\else\let\fdef\xintfdef\fi
```

### 1.11 `\xintReverseOrder`

`\xintReverseOrder`: does NOT expand its argument.

```
175 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
176 \long\def\xintreverseorder #1%
177 {%
178     \XINT_rord_main }#1%
179     \xint_relax
180     \xint_bye\xint_bye\xint_bye\xint_bye
181     \xint_bye\xint_bye\xint_bye\xint_bye
182     \xint_relax
183 }%
184 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
185 {%
186     \xint_bye #9\XINT_rord_cleanup\xint_bye
187     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
188 }%
189 \long\edef\XINT_rord_cleanup\xint_bye\XINT_rord_main #1#2\xint_relax
190 {%
191     \noexpand\expandafter\space\noexpand\xint_gob_til_xint_relax #1%
192 }%
```

1.12 `\xintLength`

`\xintLength` does NOT expand its argument.

```

193 \def\xintLength {\romannumeral0\xintlength }%
194 \long\def\xintlength #1%
195 {%
196   \XINT_length_loop
197   0.#1\xint_relax\xint_relax\xint_relax\xint_relax
198     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
199 }%
200 \long\def\XINT_length_loop #1.#2#3#4#5#6#7#8#9%
201 {%
202   \xint_gob_til_xint_relax #9\XINT_length_finish_a\xint_relax
203   \expandafter\XINT_length_loop\the\numexpr #1+\xint_c_viii.%
204 }%
205 \def\XINT_length_finish_a\xint_relax\expandafter\XINT_length_loop
206   \the\numexpr #1+\xint_c_viii.#2\xint_bye
207 {%
208   \XINT_length_finish_b #2\W\W\W\W\W\W\W\Z {#1}%
209 }%
210 \def\XINT_length_finish_b #1#2#3#4#5#6#7#8\Z
211 {%
212   \xint_gob_til_W
213     #1\XINT_length_finish_c \xint_c_
214     #2\XINT_length_finish_c \xint_c_i
215     #3\XINT_length_finish_c \xint_c_ii
216     #4\XINT_length_finish_c \xint_c_iii
217     #5\XINT_length_finish_c \xint_c_iv
218     #6\XINT_length_finish_c \xint_c_v
219     #7\XINT_length_finish_c \xint_c_vi
220     \W\XINT_length_finish_c \xint_c_vii\Z
221 }%
222 \edef\XINT_length_finish_c #1#2\Z #3%
223   {\noexpand\expandafter\space\noexpand\the\numexpr #3+#1\relax}%
224 \XINT_restorecatcodes_endinput%

```

2 Package `xinttools` implementation

.1	Catcodes, $\varepsilon$ -TEX and reload detection . . .	8	.14	<code>\xintApply</code> . . . . .	18
.2	Package identification . . . . .	9	.15	<code>\xintApplyUnbraced</code> . . . . .	19
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgdef</code> . . .	9	.16	<code>\xintSeq</code> . . . . .	19
.4	<code>\xintRevWithBraces</code> . . . . .	9	.17	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> . . . . .	22
.5	<code>\xintZapFirstSpaces</code> . . . . .	10	.18	<code>\xintiloor</code> , <code>\xintloopindex</code> , <code>\xintouter-</code> <code>loopindex</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> , <code>\xin-</code> <code>tiloopskipandredo</code> . . . . .	22
.6	<code>\xintZapLastSpaces</code> . . . . .	10	.19	<code>\XINT_xflet</code> . . . . .	22
.7	<code>\xintZapSpaces</code> . . . . .	11	.20	<code>\xintApplyInline</code> . . . . .	23
.8	<code>\xintZapSpacesB</code> . . . . .	11	.21	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xint-</code> <code>BreakForAndDo</code> . . . . .	24
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNonStripped</code> 12				
.10	<code>\xintListWithSep</code> . . . . .	13			
.11	<code>\xintNthElt</code> . . . . .	14			
.12	<code>\xintKeep</code> . . . . .	15			
.13	<code>\xintTrim</code> . . . . .	17			

.22	<code>\XINT_forever, \xintintegers, \xintdimensions, \xinrationals</code>	26	.24	<code>\xintAssign, \xintAssignArray, \xintDigitsOf</code>	29
.23	<code>\xintForpair, \xintForthree, \xintForfour</code>	28			

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

```





## 2 Package *xinttools* implementation

```
78      #1\XINT_revwbr_finish_c 8%
79      #2\XINT_revwbr_finish_c 7%
80      #3\XINT_revwbr_finish_c 6%
81      #4\XINT_revwbr_finish_c 5%
82      #5\XINT_revwbr_finish_c 4%
83      #6\XINT_revwbr_finish_c 3%
84      #7\XINT_revwbr_finish_c 2%
85      \R\XINT_revwbr_finish_c 1\Z
86 }%
87 \def\XINT_revwbr_finish_c #1#2\Z
88 {%
89   \expandafter\expandafter\expandafter
90     \space
91   \csname xint_gobble_\romannumeral #1\endcsname
92 }%
```

### 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.

```
93 \def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%
```

defined via an `\edef` in order to inject space tokens inside.

```
94 \long\edef\xintzapfirstspaces #1%
95   {\noexpand\XINT_zapbsp_a \space #1\xint_relax \space\space\xint_relax }%
96 \xint_firstofone {\long\edef\XINT_zapbsp_a #1 } %<- space token here
97 {%
```

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_relax`. 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_relax<sp><sp>\xint_relax`

```
98   \noexpand\XINT_zapbsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapbsp_b #1\space\space
99 }%
100 \long\def\XINT_zapbsp_again? #1{\xint_bye #1\XINT_zapbsp_again }%
101 \xint_firstofone{\def\XINT_zapbsp_again\XINT_zapbsp_b} {\XINT_zapbsp_a }%
102 \long\def\XINT_zapbsp_b #1\xint_relax #2\xint_relax {#1}%
```

### 2.6 `\xintZapLastSpaces`

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

```
103 \def\xintZapLastSpaces {\romannumeral0\xintzaplastspaces }%
```

Next macro is defined via an `\edef` for the space tokens.

```
104 \long\edef\xintzaplastspaces #1{\noexpand\XINT_zapbsp_a {} \noexpand\empty#1%
105   \space\space\noexpand\xint_bye\xint_relax}%
```

## 2 Package *xinttools* implementation

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

```
106 \xint_firstofone {\long\def\XINT_zapesp_a #1#2 } %<- second space here
107   {\expandafter\XINT_zapesp_b\expandafter{#2}{#1}}%
```

Notice again an `\empty` added here. This is in preparation for possibly looping back to `\XINT_zapesp_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_relax`. 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 ?;-).

```
108 \long\def\XINT_zapesp_b #1#2#3\xint_relax
109   {\XINT_zapesp_end? #3\XINT_zapesp_e {#2#1}\empty #3\xint_relax }%
```

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

```
110 \long\def\XINT_zapesp_end? #1{\xint_bye #1\XINT_zapesp_end }%
```

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

```
111 \long\def\XINT_zapesp_end\XINT_zapesp_e #1#2\xint_relax { #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.

```
112 \long\edef\XINT_zapesp_e #1{\noexpand \XINT_zapesp_a {#1\space\space}}%
```

### 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\xintzapspaces }%
114 \long\edef\xintzapspaces #1% like \xintZapFirstSpaces.
115   {\noexpand\XINT_zapsp_a \space #1\xint_relax \space\space\xint_relax }%
116 \xint_firstofone {\long\edef\XINT_zapsp_a #1 } %
117   {\noexpand\XINT_zapsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapsp_b #1\space\space}%
118 \long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again }%
119 \xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a }%
120 \xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
121 \long\edef\XINT_zapsp_c #1\xint_relax #2\xint_relax {\noexpand\XINT_zapesp_a
122   {\noexpand \empty #1\space\space\noexpand\xint_bye\xint_relax }%
```

### 2.8 `\xintZapSpacesB`

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

```
123 \def\xintZapSpacesB {\romannumeral0\xintzapspacesb }%
124 \long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint_relax\xint_relax
```

## 2 Package *xinttools* implementation

```
125             \xint_bye\xintzapspaces {#1}}%
126 \long\def\xINT_zapspb_one? #1#2%
127   {\xint_gob_til_xint_relax #1\xINT_zapspb_onlyspaces\xint_relax
128     \xint_gob_til_xint_relax #2\xINT_zapspb_bracedorone\xint_relax
129     \xint_bye {#1}}%
130 \def\xINT_zapspb_onlyspaces\xint_relax
131   \xint_gob_til_xint_relax\xint_relax\xINT_zapspb_bracedorone\xint_relax
132   \xint_bye #1\xint_bye\xintzapspaces #2{ }%
133 \long\def\xINT_zapspb_bracedorone\xint_relax
134   \xint_bye #1\xint_relax\xint_bye\xintzapspaces #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.

```
135 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
136 \long\def\xintcsvtolist #1{\expandafter\xintApply
137   \expandafter\xintzapspacesb
138   \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%
139 \def\xintCSVtoListNoExpand {\romannumeral0\xintcsvtolistnoexpand }%
140 \long\def\xintcsvtolistnoexpand #1{\expandafter\xintApply
141   \expandafter\xintzapspacesb
142   \expandafter{\romannumeral0\xintcsvtolistnonstrippednoexpand{#1}}}%
143 \def\xintCSVtoListNonStripped {\romannumeral0\xintcsvtolistnonstripped }%
144 \def\xintCSVtoListNonStrippedNoExpand
145   {\romannumeral0\xintcsvtolistnonstrippednoexpand }%
146 \long\def\xintcsvtolistnonstripped #1%
147 {%
148   \expandafter\xINT_csvtol_loop_a\expandafter
149   {\expandafter}\romannumeral-`0#1%
150     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye
151     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
152 }%
153 \long\def\xintcsvtolistnonstrippednoexpand #1%
154 {%
155   \XINT_csvtol_loop_a
156   {#1,\xint_bye,\xint_bye,\xint_bye,\xint_bye
157     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
158 }%
159 \long\def\xINT_csvtol_loop_a #1#2,#3,#4,#5,#6,#7,#8,#9,%
160 {%
161   \xint_bye #9\xINT_csvtol_finish_a\xint_bye
162   \XINT_csvtol_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
163 }%
164 \long\def\xINT_csvtol_loop_b #1#2{\XINT_csvtol_loop_a {#1#2}}%
165 \long\def\xINT_csvtol_finish_a\xint_bye\xINT_csvtol_loop_b #1#2#3\Z
166 {%
167   \XINT_csvtol_finish_b #3\R,\R,\R,\R,\R,\R,\R,\Z #2{#1}}%
```

```

168 }%
169 \def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
170 {%
171   \xint_gob_til_R
172     #1\XINT_csvtol_finish_c 8%
173     #2\XINT_csvtol_finish_c 7%
174     #3\XINT_csvtol_finish_c 6%
175     #4\XINT_csvtol_finish_c 5%
176     #5\XINT_csvtol_finish_c 4%
177     #6\XINT_csvtol_finish_c 3%
178     #7\XINT_csvtol_finish_c 2%
179     \R\XINT_csvtol_finish_c 1\Z
180 }%
181 \def\XINT_csvtol_finish_c #1#2\Z
182 {%
183   \csname XINT_csvtol_finish_d\romannumeral #1\endcsname
184 }%
185 \long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
186 \long\def\XINT_csvtol_finish_dvii #1#2#3#4#5#6#7#8#9{ #9{#1}}%
187 \long\def\XINT_csvtol_finish_dvi #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
188 \long\def\XINT_csvtol_finish_dv #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
189 \long\def\XINT_csvtol_finish_div #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
190 \long\def\XINT_csvtol_finish_diii #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
191 \long\def\XINT_csvtol_finish_dii #1#2#3#4#5#6#7#8#9%
192     { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
193 \long\def\XINT_csvtol_finish_di #1#2#3#4#5#6#7#8#9%
194     { #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.

```

195 \def\xintListWithSep {\romannumeral0\xintlistwithsep }%
196 \def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand }%
197 \long\def\xintlistwithsep #1#2%
198   {\expandafter\XINT_lws\expandafter {\romannumeral-`0#2}{#1}}%
199 \long\def\XINT_lws #1#2{\XINT_lws_start {#2}#1\xint_bye }%
200 \long\def\xintlistwithsepnoexpand #1#2{\XINT_lws_start {#1}#2\xint_bye }%
201 \long\def\XINT_lws_start #1#2%
202 {%
203   \xint_bye #2\XINT_lws_dont\xint_bye
204   \XINT_lws_loop_a {#2}{#1}%
205 }%
206 \long\def\XINT_lws_dont\xint_bye\XINT_lws_loop_a #1#2{ }%
207 \long\def\XINT_lws_loop_a #1#2#3%
208 {%
209   \xint_bye #3\XINT_lws_end\xint_bye
210   \XINT_lws_loop_b {#1}{#2#3}{#2}%
211 }%
212 \long\def\XINT_lws_loop_b #1#2{\XINT_lws_loop_a {#1#2}}%
213 \long\def\XINT_lws_end\xint_bye\XINT_lws_loop_b #1#2#3{ #1}%

```

## 2.11 `\xintNthElt`

First included in release 1.06.

`\xintNthElt {i}{stuff f-expanding to {a}{b}...{z}}` (or `tokens' `abcd...z`) returns the *i*th element (one pair of braces removed). The list is first *f*-expanded. The `\xintNthEltNoExpand` does no expansion of its second argument. Both variants expand the first argument inside `\numexpr`.

With *i* = 0, the number of items is returned. This is different from `\xintLen` which is only for numbers (particularly, it checks the sign) and different from `\xintLength` which does not *f*-expand its argument.

Negative values return the *|i|*th element from the end. Release 1.09m rewrote the initial bits of the code (which checked the sign of #1 and expanded or not #2), some `improvements' made earlier in 1.09c were quite sub-efficient. Now uses `\xint_UDzerominusfork`, moved from `xint.sty`.

```

214 \def\xintNthElt          {\romannumeral0\xintnthelt }%
215 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand }%
216 \def\xintnthelt #1#2%
217 {%
218   \expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
219   \expandafter{\romannumeral-`0#2}%
220 }%
221 \def\xintntheltnoexpand #1%
222 {%
223   \expandafter\XINT_nthelt_a\the\numexpr #1.%
224 }%
225 \def\XINT_nthelt_a #1#2.%
226 {%
227   \xint_UDzerominusfork
228     #1-{\XINT_nthelt_bzero}%
229     0#1{\XINT_nthelt_bneg {#2}}%
230     0-{\XINT_nthelt_bpos {#1#2}}%
231   \krof
232 }%
233 \long\def\XINT_nthelt_bzero #1%
234 {%
235   \XINT_length_loop 0.#1\xint_relax\xint_relax\xint_relax\xint_relax
236     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
237 }%
238 \long\def\XINT_nthelt_bneg #1#2%
239 {%
240   \expandafter\XINT_nthelt_loop_a\expandafter {\the\numexpr #1\expandafter}%
241   \romannumeral0\xintrevwithbracesnoexpand {#2}%
242     \xint_relax\xint_relax\xint_relax\xint_relax
243     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
244 }%
245 \long\def\XINT_nthelt_bpos #1#2%
246 {%
247   \XINT_nthelt_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
248     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
249 }%
250 \def\XINT_nthelt_loop_a #1%
251 {%
252   \ifnum #1>\xint_c_viii
253     \expandafter\XINT_nthelt_loop_b

```

## 2 Package *xinttools* implementation

```
254 \else
255 \XINT_nthelt_getit
256 \fi
257 {#1}%
258 }%
259 \long\def\XINT_nthelt_loop_b #1#2#3#4#5#6#7#8#9%
260 {%
261 \xint_gob_til_xint_relax #9\XINT_nthelt_silentend\xint_relax
262 \expandafter\XINT_nthelt_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
263 }%
264 \def\XINT_nthelt_silentend #1\xint_bye { }%
265 \def\XINT_nthelt_getit\fi #1%
266 {%
267 \fi\expandafter\expandafter\expandafter\XINT_nthelt_finish
268 \csname xint_gobble_\romannumeral\numexpr#1-\xint_c_i\endcsname
269 }%
270 \long\edef\XINT_nthelt_finish #1#2\xint_bye
271 {\noexpand\xint_gob_til_xint_relax #1\noexpand\expandafter\space
272 \noexpand\xint_gobble_ii\xint_relax\space #1}%
```

### 2.12 `\xintKeep`

First included in release 1.09m.

`\xintKeep {i}{stuff f-expanding to {a}{b}...{z}}` (or 'tokens' `abcd...z`, but each naked token ends up braced in the output) returns (in two expansion steps) the first `i` elements from the list, which is first `f`-expanded. The `i` is expanded inside `\numexpr`. The variant `\xintKeepNoExpand` does not expand the list argument.

With `i = 0`, the empty sequence is returned.

With `i < 0`, the last `|i|` elements are returned (in the same order as in the original list).

With `|i|` equal to or bigger than the length of the (`f`-expanded) list, the full list is returned.

```
273 \def\xintKeep          {\romannumeral0\xintkeep }%
274 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
275 \def\xintkeep #1#2%
276 {%
277 \expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
278 \expandafter{\romannumeral-\`0#2}%
279 }%
280 \def\xintkeepnoexpand #1%
281 {%
282 \expandafter\XINT_keep_a\the\numexpr #1.%
283 }%
284 \def\XINT_keep_a #1#2.%
285 {%
286 \xint_UDzerominusfork
287 #1-{\expandafter\space\xint_gobble_i }%
288 0#1{\XINT_keep_bneg_a {#2}}%
289 0-{\XINT_keep_bpos {#1#2}}%
290 \krof
291 }%
292 \long\def\XINT_keep_bneg_a #1#2%
293 {%
294 \expandafter\XINT_keep_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
```

## 2 Package *xinttools* implementation

```

295 }%
296 \def\XINT_keep_bneg_b #1#2.%
297 {%
298   \xint_UDzerominusfork
299     #1-{\xint_firstofone_thenstop }%
300     0#1{\xint_firstofone_thenstop }%
301     0-{\XINT_trim_bpos {#1#2}}%
302   \krof
303 }%
304 \long\def\XINT_keep_bpos #1#2%
305 {%
306   \XINT_keep_loop_a {#1}{}#2\xint_relax\xint_relax\xint_relax\xint_relax
307     \xint_relax\xint_relax\xint_relax\xint_bye
308 }%
309 \def\XINT_keep_loop_a #1%
310 {%
311   \ifnum #1>\xint_c_vi
312     \expandafter\XINT_keep_loop_b
313   \else
314     \XINT_keep_finish
315   \fi
316   {#1}%
317 }%
318 \long\def\XINT_keep_loop_b #1#2#3#4#5#6#7#8#9%
319 {%
320   \xint_gob_til_xint_relax #9\XINT_keep_enda\xint_relax
321   \expandafter\XINT_keep_loop_c\expandafter{\the\numexpr #1-\xint_c_vii}%
322   {{#3}{#4}{#5}{#6}{#7}{#8}{#9}}{#2}%
323 }%
324 \long\def\XINT_keep_loop_c #1#2#3{\XINT_keep_loop_a {#1}{#3#2}}%
325 \long\def\XINT_keep_enda\xint_relax
326   \expandafter\XINT_keep_loop_c\expandafter #1#2#3#4\xint_bye
327 {%
328   \XINT_keep_endb #4\W\W\W\W\W\W\Z #2{#3}%
329 }%
330 \def\XINT_keep_endb #1#2#3#4#5#6#7\Z
331 {%
332   \xint_gob_til_W
333   #1\XINT_keep_endc_
334   #2\XINT_keep_endc_i
335   #3\XINT_keep_endc_ii
336   #4\XINT_keep_endc_iii
337   #5\XINT_keep_endc_iv
338   #6\XINT_keep_endc_v
339   \W\XINT_keep_endc_vi\Z
340 }%
341 \long\def\XINT_keep_endc_ #1\Z #2#3#4#5#6#7#8#9{ #9}%
342 \long\def\XINT_keep_endc_i #1\Z #2#3#4#5#6#7#8#9{ #9{#2}}%
343 \long\def\XINT_keep_endc_ii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}}%
344 \long\def\XINT_keep_endc_iii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}}%
345 \long\def\XINT_keep_endc_iv #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}}%
346 \long\def\XINT_keep_endc_v #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}{#6}}%

```



## 2 Package `xinttools` implementation

```
347 \long\def\XINT_keep_endc_vi\Z #1#2#3#4#5#6#7#8{ #8{#1}{#2}{#3}{#4}{#5}{#6}}%
348 \long\def\XINT_keep_finish\fi #1#2#3#4#5#6#7#8#9\xint_bye
349 {%
350   \fi\XINT_keep_finish_loop_a {#1}{#3}{#4}{#5}{#6}{#7}{#8}\Z {#2}%
351 }%
352 \def\XINT_keep_finish_loop_a #1%
353 {%
354   \xint_gob_til_zero #1\XINT_keep_finish_z0%
355   \expandafter\XINT_keep_finish_loop_b\expandafter
356     {\the\numexpr #1-\xint_c_i}%
357 }%
358 \long\def\XINT_keep_finish_z0%
359   \expandafter\XINT_keep_finish_loop_b\expandafter #1#2#3\Z #4{ #4#2}%
360 \long\def\XINT_keep_finish_loop_b #1#2#3%
361 {%
362   \xint_gob_til_xint_relax #3\XINT_keep_finish_exit\xint_relax
363   \XINT_keep_finish_loop_c {#1}{#2}{#3}%
364 }%
365 \long\def\XINT_keep_finish_exit\xint_relax
366   \XINT_keep_finish_loop_c #1#2#3\Z #4{ #4#2}%
367 \long\def\XINT_keep_finish_loop_c #1#2#3%
368   {\XINT_keep_finish_loop_a {#1}{#2}{#3}}%
```

### 2.13 `\xintTrim`

First included in release 1.09m.

`\xintTrim {i}{stuff f-expanding to {a}{b}...{z}}` (or 'tokens' `abcd...z`, but each naked token ends up braced in the output) returns (in two expansion steps) the sequence with the first `i` elements omitted. The list is first `f`-expanded. The `i` is expanded inside `\numexpr`. Variant `\xintTrimNoExpand` does not expand the list argument.

With `i = 0`, the original (expanded) list is returned.

With `i < 0`, the last `|i|` elements from the tail are suppressed.

With `|i|` equal to or bigger than the length of the (`f`-expanded) list, the empty list is returned.

```
369 \def\xintTrim          {\romannumeral0\xinttrim }%
370 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
371 \def\xinttrim #1#2%
372 {%
373   \expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
374   \expandafter{\romannumeral-`0#2}%
375 }%
376 \def\xinttrimnoexpand #1%
377 {%
378   \expandafter\XINT_trim_a\the\numexpr #1.%
379 }%
380 \def\XINT_trim_a #1#2.%
381 {%
382   \xint_UDzerominusfork
383     #1-{\xint_firstofone_thenstop }%
384     0#1{\XINT_trim_bneg_a {#2}}%
385     0-{\XINT_trim_bpos {#1#2}}%
386   \krof
387 }%
```

## 2 Package *xinttools* implementation

```
388 \long\def\XINT_trim_bneg_a #1#2%
389 {%
390   \expandafter\XINT_trim_bneg_b \the\numexpr \xintLength{#2}-#1.#2}%
391 }%
392 \def\XINT_trim_bneg_b #1#2.%
393 {%
394   \xint_UDzerominusfork
395     #1-{\expandafter\space\xint_gobble_i }%
396     0#1{\expandafter\space\xint_gobble_i }%
397     0-{\XINT_keep_bpos {#1#2}}%
398   \krof
399 }%
400 \long\def\XINT_trim_bpos #1#2%
401 {%
402   \XINT_trim_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
403     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
404 }%
405 \def\XINT_trim_loop_a #1%
406 {%
407   \ifnum #1>\xint_c_vii
408     \expandafter\XINT_trim_loop_b
409   \else
410     \XINT_trim_finish
411   \fi
412   {#1}%
413 }%
414 \long\def\XINT_trim_loop_b #1#2#3#4#5#6#7#8#9%
415 {%
416   \xint_gob_til_xint_relax #9\XINT_trim_silentend\xint_relax
417   \expandafter\XINT_trim_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
418 }%
419 \def\XINT_trim_silentend #1\xint_bye { }%
420 \def\XINT_trim_finish\fi #1%
421 {%
422   \fi\expandafter\expandafter\expandafter\XINT_trim_finish_a
423   \expandafter\expandafter\expandafter\space % avoids brace removal
424   \cename xint_gobble_\romannumeral\numexpr#1\endcsname
425 }%
426 \long\def\XINT_trim_finish_a #1\xint_relax #2\xint_bye {#1}%
```

### 2.14 `\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.

```
427 \def\xintApply          {\romannumeral0\xintapply }%
428 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
429 \long\def\xintapply #1#2%
430 {%
431   \expandafter\XINT_apply\expandafter {\romannumeral-`0#2}%
432   {#1}%
433 }%
```

## 2 Package *xinttools* implementation

```
434 \long\def\XINT_apply #1#2{\XINT_apply_loop_a }{#2}#1\xint_bye }%
435 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a }{#1}#2\xint_bye }%
436 \long\def\XINT_apply_loop_a #1#2#3%
437 {%
438   \xint_bye #3\XINT_apply_end\xint_bye
439   \expandafter
440   \XINT_apply_loop_b
441   \expandafter {\romannumeral-`0#2{#3}}{#1}{#2}%
442 }%
443 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a }{#2{#1}}}%
444 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
445   \expandafter #1#2#3{ #2}%
```

### 2.15 `\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.

```
446 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
447 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
448 \long\def\xintapplyunbraced #1#2%
449 {%
450   \expandafter\XINT_applyunbr\expandafter {\romannumeral-`0#2}%
451   {#1}%
452 }%
453 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a }{#2}#1\xint_bye }%
454 \long\def\xintapplyunbracednoexpand #1#2%
455   {\XINT_applyunbr_loop_a }{#1}#2\xint_bye }%
456 \long\def\XINT_applyunbr_loop_a #1#2#3%
457 {%
458   \xint_bye #3\XINT_applyunbr_end\xint_bye
459   \expandafter\XINT_applyunbr_loop_b
460   \expandafter {\romannumeral-`0#2{#3}}{#1}{#2}%
461 }%
462 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a }{#2#1}}%
463 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
464   \expandafter #1#2#3{ #2}%
```

### 2.16 `\xintSeq`

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

```
465 \def\xintSeq {\romannumeral0\xintseq }%
466 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
467 \def\XINT_seq_chkopt #1%
468 {%
469   \ifx [#1\expandafter\XINT_seq_opt
470     \else\expandafter\XINT_seq_noopt
471   \fi #1%
472 }%
```

## 2 Package *xinttools* implementation

```

473 \def\XINT_seq_noopt #1\xint_bye #2%
474 {%
475   \expandafter\XINT_seq\expandafter
476     {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
477 }%
478 \def\XINT_seq #1#2%
479 {%
480   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
481     \expandafter\xint_firstoftwo_thenstop
482   \or
483     \expandafter\XINT_seq_p
484   \else
485     \expandafter\XINT_seq_n
486   \fi
487   {#2}{#1}%
488 }%
489 \def\XINT_seq_p #1#2%
490 {%
491   \ifnum #1>#2
492     \expandafter\expandafter\expandafter\XINT_seq_p
493   \else
494     \expandafter\XINT_seq_e
495   \fi
496   \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
497 }%
498 \def\XINT_seq_n #1#2%
499 {%
500   \ifnum #1<#2
501     \expandafter\expandafter\expandafter\XINT_seq_n
502   \else
503     \expandafter\XINT_seq_e
504   \fi
505   \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
506 }%
507 \def\XINT_seq_e #1#2#3{ }%
508 \def\XINT_seq_opt [\xint_bye #1]#2#3%
509 {%
510   \expandafter\XINT_seqo\expandafter
511     {\the\numexpr #2\expandafter}\expandafter
512     {\the\numexpr #3\expandafter}\expandafter
513     {\the\numexpr #1}%
514 }%
515 \def\XINT_seqo #1#2%
516 {%
517   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
518     \expandafter\XINT_seqo_a
519   \or
520     \expandafter\XINT_seqo_pa
521   \else
522     \expandafter\XINT_seqo_na
523   \fi
524   {#1}{#2}%

```

## 2 Package *xinttools* implementation

```
525 }%
526 \def\XINT_seqo_a #1#2#3{ {#1}}%
527 \def\XINT_seqo_o #1#2#3#4{ #4}%
528 \def\XINT_seqo_pa #1#2#3%
529 {%
530   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
531     \expandafter\XINT_seqo_o
532   \or
533     \expandafter\XINT_seqo_pb
534   \else
535     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
536   \fi
537   {#1}{#2}{#3}{{#1}}%
538 }%
539 \def\XINT_seqo_pb #1#2#3%
540 {%
541   \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
542 }%
543 \def\XINT_seqo_pc #1#2%
544 {%
545   \ifnum #1>#2
546     \expandafter\XINT_seqo_o
547   \else
548     \expandafter\XINT_seqo_pd
549   \fi
550   {#1}{#2}%
551 }%
552 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
553 \def\XINT_seqo_na #1#2#3%
554 {%
555   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
556     \expandafter\XINT_seqo_o
557   \or
558     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
559   \else
560     \expandafter\XINT_seqo_nb
561   \fi
562   {#1}{#2}{#3}{{#1}}%
563 }%
564 \def\XINT_seqo_nb #1#2#3%
565 {%
566   \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
567 }%
568 \def\XINT_seqo_nc #1#2%
569 {%
570   \ifnum #1<#2
571     \expandafter\XINT_seqo_o
572   \else
573     \expandafter\XINT_seqo_nd
574   \fi
575   {#1}{#2}%
576 }%
```

```
577 \def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%
```

## 2.17 `\xintloop`, `\xintbreakloop`, `\xintbreakloopanddo`, `\xintloopskiptonext`

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

```
578 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
579 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
580     #1\xintloop_again\fi\xint_gobble_i {#1}}%
581 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%
582 \long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
583 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
584     #2\xintloop_again\fi\xint_gobble_i {#2}}%
```

## 2.18 `\xintilooop`, `\xintilooopindex`, `\xintouterilooopindex`, `\xintbreakilooop`, `\xintbreakilooopanddo`, `\xintilooopskiptonext`, `\xintilooopskipandredo`

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

```
585 \def\xintilooop [#1+#2]{%
586     \expandafter\xintilooop_a\the\numexpr #1\expandafter.\the\numexpr #2.}%
587 \long\def\xintilooop_a #1.#2.#3#4\repeat{%
588     #3#4\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
589 \def\xintilooop_again\fi\xint_gobble_iii #1#2{%
590     \fi\expandafter\xintilooop_again_b\the\numexpr#1+#2.#2.}%
591 \long\def\xintilooop_again_b #1.#2.#3{%
592     #3\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
593 \long\def\xintbreakilooop #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
594 \long\def\xintbreakilooopanddo
595     #1.#2\xintilooop_again\fi\xint_gobble_iii #3#4#5{#1}%
596 \long\def\xintilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%
597     {#2#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
598 \long\def\xintouterilooopindex #1\xintilooop_again
599     #2\xintilooop_again\fi\xint_gobble_iii #3%
600     {#3#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
601 \long\def\xintilooopskiptonext #1\xintilooop_again\fi\xint_gobble_iii #2#3{%
602     \expandafter\xintilooop_again_b \the\numexpr#2+#3.#3.}%
603 \long\def\xintilooopskipandredo #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
604     #4\xintilooop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%
```

## 2.19 `\XINT_xflet`

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

```
605 \def\XINT_xflet #1%
606 {%
607     \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
608 }%
609 \def\XINT_xflet_zapsp
610 {%
611     \expandafter\futurelet\expandafter\XINT_token
612     \expandafter\XINT_xflet_sp?\romannumeral-`0%
```

```

613 }%
614 \def\XINT_xflet_sp?
615 {%
616   \ifx\XINT_token\XINT_sptoken
617     \expandafter\XINT_xflet_zapsp
618   \else\expandafter\XINT_xflet_zapspB
619   \fi
620 }%
621 \def\XINT_xflet_zapspB
622 {%
623   \expandafter\futurelet\expandafter\XINT_tokenB
624   \expandafter\XINT_xflet_spB?\romannumeral-`0%
625 }%
626 \def\XINT_xflet_spB?
627 {%
628   \ifx\XINT_tokenB\XINT_sptoken
629     \expandafter\XINT_xflet_zapspB
630   \else\expandafter\XINT_xflet_eq?
631   \fi
632 }%
633 \def\XINT_xflet_eq?
634 {%
635   \ifx\XINT_token\XINT_tokenB
636     \expandafter\XINT_xflet_macro
637   \else\expandafter\XINT_xflet_zapsp
638   \fi
639 }%

```

## 2.20 `\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.

```

640 \catcode`Z 3
641 \long\def\xintApplyInline #1#2%
642 {%
643   \long\expandafter\def\expandafter\XINT_inline_macro
644   \expandafter ##\expandafter 1\expandafter {#1{##1}}%
645   \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
646 }%
647 \def\XINT_inline_b
648 {%
649   \ifx\XINT_token Z\expandafter\xint_gobble_i
650   \else\expandafter\XINT_inline_d\fi
651 }%
652 \long\def\XINT_inline_d #1%
653 {%
654   \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
655 }%
656 \def\XINT_inline_e

```

```

657 {%
658   \ifx\XINT_token Z\expandafter\XINT_inline_w
659   \else\expandafter\XINT_inline_f\fi
660 }%
661 \def\XINT_inline_f
662 {%
663   \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
664 }%
665 \long\def\XINT_inline_g #1%
666 {%
667   \expandafter\XINT_inline_macro\XINT_item
668   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
669 }%
670 \def\XINT_inline_w #1%
671 {%
672   \expandafter\XINT_inline_macro\XINT_item
673 }%

```

## 2.21 `\xintFor`, `\xintFor*`, `\xintBreakFor`, `\xintBreakForAndDo`

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

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

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

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

1.09f rewrites (2013/11/03) the code which now allows all macro parameters from #1 to #9 in `\xintFor`, `\xintFor*`, and `\XINT_forever`.

```

674 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{{#####2}}}\fi}%
675 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{{#####2}}}\fi}%
676 \def\XINT_tmpc #1%
677 {%
678   \expandafter\edef \csname XINT_for_left#1\endcsname
679     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
680   \expandafter\edef \csname XINT_for_right#1\endcsname
681     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
682 }%
683 \xintApplyInline \XINT_tmpc {123456789}%
684 \long\def\xintBreakFor #1Z{%
685 \long\def\xintBreakForAndDo #1#2Z{#1}%
686 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
687   \futurelet\XINT_token\XINT_for_ifstar }%
688 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
689   \else\expandafter\XINT_for \fi }%
690 \catcode`U 3 % with numexpr

```



## 2 Package *xinttools* implementation

```
691 \catcode`V 3 % with xintfrac.sty (xint.sty not enough)
692 \catcode`D 3 % with dimexpr
693 \def\XINT_flet_zapsp
694 {%
695   \futurelet\XINT_token\XINT_flet_sp?
696 }%
697 \def\XINT_flet_sp?
698 {%
699   \ifx\XINT_token\XINT_sptoken
700     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
701   \else\expandafter\XINT_flet_macro
702   \fi
703 }%
704 \long\def\XINT_for #1#2in#3#4#5%
705 {%
706   \expandafter\XINT_toks\expandafter
707   {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
708   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
709   \expandafter\XINT_flet_zapsp #3Z%
710 }%
711 \def\XINT_for_forever? #1Z%
712 {%
713   \ifx\XINT_token U\XINT_to_forever\fi
714   \ifx\XINT_token V\XINT_to_forever\fi
715   \ifx\XINT_token D\XINT_to_forever\fi
716   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtlist {#1}Z%
717 }%
718 \def\XINT_to_forever\fi #1\xintcsvtlist #2{\fi \XINT_forever #2}%
719 \long\def\XINT_forx *#1#2in#3#4#5%
720 {%
721   \expandafter\XINT_toks\expandafter
722   {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
723   \XINT_xflet\XINT_forx_forever? #3Z%
724 }%
725 \def\XINT_forx_forever?
726 {%
727   \ifx\XINT_token U\XINT_to_forxever\fi
728   \ifx\XINT_token V\XINT_to_forxever\fi
729   \ifx\XINT_token D\XINT_to_forxever\fi
730   \XINT_forx_empty?
731 }%
732 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
733 \catcode`U 11
734 \catcode`D 11
735 \catcode`V 11
736 \def\XINT_forx_empty?
737 {%
738   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
739   \the\XINT_toks
740 }%
741 \long\def\XINT_for_d #1#2#3%
742 {%
```

## 2 Package *xinttools* implementation

```
743 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
744 \XINT_toks {{#3}}%
745 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
746         \the\XINT_toks \csname XINT_for_right#1\endcsname }%
747 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_for_d #1{#2}}%
748 \futurelet\XINT_token\XINT_for_last?
749 }%
750 \long\def\XINT_forx_d #1#2#3%
751 {%
752 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
753 \XINT_toks {{#3}}%
754 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
755         \the\XINT_toks \csname XINT_for_right#1\endcsname }%
756 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_forx_d #1{#2}}%
757 \XINT_xflet\XINT_for_last?
758 }%
759 \def\XINT_for_last?
760 {%
761 \let\xintifForLast\xint_secondoftwo
762 \ifx\XINT_token Z\let\xintifForLast\xint_firstoftwo
763 \xint_afterfi{\xintBreakForAndDo{\XINT_x\xint_gobble_i Z}}\fi
764 \the\XINT_toks
765 }%
```

### 2.22 `\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`).

```
766 \catcode`U 3
767 \catcode`D 3
768 \catcode`V 3
769 \let\xintegers U%
770 \let\xintintegers U%
771 \let\xintdimensions D%
772 \let\xinrationals V%
773 \def\XINT_forever #1%
774 {%
775 \expandafter\XINT_forever_a
776 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
777 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
778 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
779 }%
780 \catcode`U 11
781 \catcode`D 11
782 \catcode`V 11
783 \def\XINT_?expr_Ua #1#2%
784 {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
785 \expandafter\relax\expandafter}%
```

## 2 Package *xinttools* implementation

```

786 \expandafter{\the\numexpr #2}}%
787 \def\XINT_?expr_Da #1#2%
788 {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
789 \expandafter s\expandafter p\expandafter\relax\expandafter}%
790 \expandafter{\number\dimexpr #2}}%
791 \catcode`Z 11
792 \def\XINT_?expr_Va #1#2%
793 {%
794 \expandafter\XINT_?expr_Vb\expandafter
795 {\romannumeral-`0\xintraawithzeros{\xintZapSpacesB{#2}}}%
796 {\romannumeral-`0\xintraawithzeros{\xintZapSpacesB{#1}}}%
797 }%
798 \catcode`Z 3
799 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1.}%
800 \def\XINT_?expr_Vc #1/#2.#3/#4.%
801 {%
802 \xintifEq {#2}{#4}%
803 {\XINT_?expr_Vf {#3}{#1}{#2}}%
804 {\expandafter\XINT_?expr_Vd\expandafter
805 {\romannumeral0\xintiimul {#2}{#4}}%
806 {\romannumeral0\xintiimul {#1}{#4}}%
807 {\romannumeral0\xintiimul {#2}{#3}}%
808 }%
809 }%
810 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
811 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
812 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
813 \def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
814 \def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
815 \def\XINT_?expr_Vi {{1/1}{0111}}%
816 \def\XINT_?expr_U #1#2%
817 {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
818 \def\XINT_?expr_D #1#2%
819 {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
820 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
821 \def\XINT_?expr_Vx #1#2%
822 {%
823 \expandafter\XINT_?expr_Vy\expandafter
824 {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
825 }%
826 \def\XINT_?expr_Vy #1#2#3#4%
827 {%
828 \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
829 }%
830 \def\XINT_forever_a #1#2#3#4%
831 {%
832 \ifx #4[\expandafter\XINT_forever_opt_a
833 \else\expandafter\XINT_forever_b
834 \fi #1#2#3#4%
835 }%
836 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
837 \long\def\XINT_forever_c #1#2#3#4#5%

```

## 2 Package `xinttools` implementation

```
838     {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
839 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
840 {%
841     \expandafter\expandafter\expandafter
842     \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
843     \romannumeral-`0#1{#4}{#5}#3%
844 }%
845 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
846 \long\def\XINT_forever_d #1#2#3#4#5%
847 {%
848     \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
849     \XINT_toks {#{2}}%
850     \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
851         \the\XINT_toks \csname XINT_for_right#1\endcsname }%
852     \XINT_x
853     \let\xintifForFirst\xint_secondoftwo
854     \expandafter\XINT_forever_d\expandafter #1\romannumeral-`0#4{#2}{#3}#4{#5}%
855 }%
```

### 2.23 `\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.

```
856 \catcode`j 3
857 \long\def\xintForpair #1#2#3in#4#5#6%
858 {%
859     \let\xintifForFirst\xint_firstoftwo
860     \XINT_toks {\XINT_forpair_d #2{#6}}%
861     \expandafter\the\expandafter\XINT_toks #4jZ%
862 }%
863 \long\def\XINT_forpair_d #1#2#3(#4)#5%
864 {%
865     \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
866     \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
867     \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
868         \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
869     \let\xintifForLast\xint_secondoftwo
870     \ifx #5j\expandafter\xint_firstoftwo
871     \else\expandafter\xint_secondoftwo
872     \fi
873     {\let\xintifForLast\xint_firstoftwo
874     \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
875     \XINT_x
876     \let\xintifForFirst\xint_secondoftwo\XINT_forpair_d #1{#2}%
877 }%
878 \long\def\xintForthree #1#2#3in#4#5#6%
879 {%
880     \let\xintifForFirst\xint_firstoftwo
881     \XINT_toks {\XINT_forthree_d #2{#6}}%
```

## 2 Package *xinttools* implementation

```
882 \expandafter\the\expandafter\XINT_toks #4jZ%
883 }%
884 \long\def\XINT_forthree_d #1#2#3(#4)#5%
885 {%
886 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
887 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
888 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
889 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
890 \let\xintifForLast\xint_secondoftwo
891 \ifx #5j\expandafter\xint_firstoftwo
892 \else\expandafter\xint_secondoftwo
893 \fi
894 {\let\xintifForLast\xint_firstoftwo
895 \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
896 \XINT_x
897 \let\xintifForFirst\xint_secondoftwo\XINT_forthree_d #1{#2}%
898 }%
899 \long\def\xintForfour #1#2#3in#4#5#6%
900 {%
901 \let\xintifForFirst\xint_firstoftwo
902 \XINT_toks {\XINT_forfour_d #2{#6}}%
903 \expandafter\the\expandafter\XINT_toks #4jZ%
904 }%
905 \long\def\XINT_forfour_d #1#2#3(#4)#5%
906 {%
907 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
908 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
909 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
910 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
911 \let\xintifForLast\xint_secondoftwo
912 \ifx #5j\expandafter\xint_firstoftwo
913 \else\expandafter\xint_secondoftwo
914 \fi
915 {\let\xintifForLast\xint_firstoftwo
916 \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
917 \XINT_x
918 \let\xintifForFirst\xint_secondoftwo\XINT_forfour_d #1{#2}%
919 }%
920 \catcode`Z 11
921 \catcode`j 11
```

### 2.24 `\xintAssign`, `\xintAssignArray`, `\xintDigitsOf`

`\xintAssign {a}{b}..{z}\to\A\B...Z` or `\xintAssignArray {a}{b}..{z}\to\U`

```
922 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
923 \def\XINT_assign_fork
924 {%
925 \let\XINT_assign_def\def
926 \ifx\XINT_token[\expandafter\XINT_assign_opt
927 \else\expandafter\XINT_assign_a
928 \fi
929 }%
```

## 2 Package *xinttools* implementation

```
930 \def\XINT_assign_opt [#1]%
931 {%
932   \ifcsname #1def\endcsname
933     \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
934   \else
935     \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
936   \fi
937   \XINT_assign_a
938 }%
939 \long\def\XINT_assign_a #1\to
940 {%
941   \expandafter\XINT_assign_b\romannumeral-`0#1{}\to
942 }%
943 \long\def\XINT_assign_b #1% attention to the # at the beginning of next line
944 #{%
945   \def\xint_temp {#1}%
946   \ifx\empty\xint_temp
947     \expandafter\XINT_assign_c
948   \else
949     \expandafter\XINT_assign_d
950   \fi
951 }%
952 \long\def\XINT_assign_c #1#2\to #3%
953 {%
954   \XINT_assign_def #3{#1}%
955   \def\xint_temp {#2}%
956   \unless\ifx\empty\xint_temp\xint_afterfi{\XINT_assign_b #2\to }\fi
957 }%
958 \def\XINT_assign_d #1\to #2% normally #1 is {} here.
959 {%
960   \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
961 }%
962 \def\xintRelaxArray #1%
963 {%
964   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
965   \escapechar -1
966   \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
967   \XINT_restoreescapechar
968   \xintilop [\csname\xint_arrayname 0\endcsname+-1]
969   \global
970   \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
971   \ifnum \xintilopindex > \xint_c_
972   \repeat
973   \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
974   \global\let #1\relax
975 }%
976 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
977   \XINT_flet_zapsp }%
978 \def\XINT_assignarray_fork
979 {%
980   \let\XINT_assignarray_def\def
981   \ifx\XINT_token[\expandafter\XINT_assignarray_opt
```

## 2 Package *xinttools* implementation

```

982         \else\expandafter\XINT_assignarray
983     \fi
984 }%
985 \def\XINT_assignarray_opt [#1]%
986 {%
987     \ifcsname #1def\endcsname
988         \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
989     \else
990         \expandafter\let\expandafter\XINT_assignarray_def
991                                 \csname xint#1def\endcsname
992     \fi
993     \XINT_assignarray
994 }%
995 \long\def\XINT_assignarray #1\to #2%
996 {%
997     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
998     \escapechar -1
999     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1000     \XINT_restoreescapechar
1001     \def\xint_itemcount {0}%
1002     \expandafter\XINT_assignarray_loop \romannumeral-`0#1\xint_relax
1003     \csname\xint_arrayname 00\expandafter\endcsname
1004     \csname\xint_arrayname 0\expandafter\endcsname
1005     \expandafter {\xint_arrayname}#2%
1006 }%
1007 \long\def\XINT_assignarray_loop #1%
1008 {%
1009     \def\xint_temp {#1}%
1010     \ifx\xint_brelax\xint_temp
1011         \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1012             \expandafter{\the\numexpr\xint_itemcount}%
1013         \expandafter\expandafter\expandafter\XINT_assignarray_end
1014     \else
1015         \expandafter\def\expandafter\xint_itemcount\expandafter
1016             {\the\numexpr\xint_itemcount+\xint_c_i}%
1017         \expandafter\XINT_assignarray_def
1018             \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1019         \expandafter{\xint_temp }%
1020         \expandafter\XINT_assignarray_loop
1021     \fi
1022 }%
1023 \def\XINT_assignarray_end #1#2#3#4%
1024 {%
1025     \def #4##1%
1026     {%
1027         \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1028     }%
1029     \def #1##1%
1030     {%
1031         \ifnum ##1<\xint_c_
1032             \xint_afterfi {\xintError:ArrayIndexIsNegative\space }%
1033         \else

```

```

1034     \xint_afterfi {%
1035         \ifnum ##1>#2
1036             \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
1037         \else\xint_afterfi
1038     {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1039     \fi}%
1040 \fi
1041 }%
1042 }%
1043 \let\xintDigitsOf\xintAssignArray
1044 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1045 \XINT_restorecatcodes_endinput%

```

### 3 Package `xintcore` implementation

.1	Catcodes, $\varepsilon$ - $\TeX$ and reload detection	32	.12	<code>\xintiSub</code> , <code>\xintiiSub</code>	47
.2	Package identification	33	.13	<code>\xintiMul</code> , <code>\xintiiMul</code>	52
.3	More token management, constants	33	.14	<code>\xintiSqr</code> , <code>\xintiiSqr</code>	61
.4	<code>\XINT_RQ</code>	34	.15	<code>\xintiPow</code> , <code>\xintiiPow</code>	61
.5	<code>\XINT_OQ</code>	34	.16	<code>\xintiDivision</code> , <code>\xintiQuo</code> , <code>\xintiRem</code> ,	
.6	<code>\XINT_cuz</code>	35		<code>\xintiiDivision</code> , <code>\xintiiQuo</code> , <code>\xintiiRem</code>	64
.7	<code>\xintNum</code>	36	.17	<code>\xintiDivRound</code> , <code>\xintiiDivRound</code>	77
.8	<code>\xintSgn</code> , <code>\xintiiSgn</code> , <code>\XINT_Sgn</code> ,		.18	<code>\xintiDivTrunc</code> , <code>\xintiiDivTrunc</code>	78
	<code>\XINT_cntSgn</code>	37	.19	<code>\xintiMod</code> , <code>\xintiiMod</code>	79
.9	<code>\xintiOpp</code>	38	.20	<code>\xintDec</code>	79
.10	<code>\xintiAbs</code> , <code>\xintiiAbs</code>	38	.21	<code>\xintInc</code>	80
.11	<code>\xintiAdd</code> , <code>\xintiiAdd</code>	46			

Got split off from `xint` with release 1.1. Adds `\xintiiDivRound`. Does not load `xinttools`.

Since release `xint 1.09a` these macros doing arithmetic operations apply systematically `\xintnum` to their arguments; this adds a little overhead but this is more convenient for using count registers even with infix notation; also this is what `xintfrac.sty` did all along. Simplifies the discussion in the documentation too.

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

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

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

```

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

```



### 3 Package *xintcore* 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{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 [2014/11/07 v1.1a Expandable arithmetic on big integers (jfb)]%
```

### 3.3 More token management, constants

```
47 \def\xint_minus_thenstop { -}%
48 \def\xint_gob_til_zeros_iii #1000{%
49 \def\xint_gob_til_zeros_iv #10000}%
50 \def\xint_gob_til_one #11{%
51 \def\xint_gob_til_G #1G}%
52 \def\xint_gob_til_minus #1-}%
53 \def\xint_gob_til_relax #1\relax {%
54 \def\xint_exchangetwo_keepbraces #1#2{{#2}{#1}}%
55 \def\xint_exchangetwo_keepbraces_thenstop #1#2{ {#2}{#1}}%
56 \def\xint_UDzerofork #10#2#3\krof {#2}%
57 \def\xint_UDsignfork #1-#2#3\krof {#2}%
58 \def\xint_UDwfork #1\W#2#3\krof {#2}%
59 \def\xint_UDzerofork #100#2#3\krof {#2}%
60 \def\xint_UDdonezerofork #110#2#3\krof {#2}%
61 \def\xint_UDsignsfork #1--#2#3\krof {#2}%
62 \chardef\xint_c_ix 9
```

### 3 Package *xintcore* implementation

```
63 \chardef\xint_c_x      10
64 \chardef\xint_c_ii^v  32 % not used in xint, common to xintfrac and xintbinhex
65 \chardef\xint_c_ii^vi  64
66 \mathchardef\xint_c_ixixixix 9999
67 \mathchardef\xint_c_x^iv  10000
68 \newcount\xint_c_x^viii  \xint_c_x^viii 100000000
```

#### 3.4 `\XINT_RQ`

Cette macro renverse et ajoute le nombre minimal de zéros à la fin pour que la longueur soit alors multiple de 4

```
\romannumeral0\XINT_RQ {}<le truc à renverser>\R\R\R\R\R\R\R\R\Z
```

Attention, ceci n'est utilisé que pour des chaînes de chiffres, et donc le comportement avec des `{.}` ou autres espaces n'a fait l'objet d'aucune attention.

```
69 \def\XINT_RQ #1#2#3#4#5#6#7#8#9%
70 {%
71   \xint_gob_til_R #9\XINT_RQ_end_a\R\XINT_RQ {#9#8#7#6#5#4#3#2#1}%
72 }%
73 \def\XINT_RQ_end_a\R\XINT_RQ #1#2\Z
74 {%
75   \XINT_RQ_end_b #1\Z
76 }%
77 \def\XINT_RQ_end_b #1#2#3#4#5#6#7#8%
78 {%
79   \xint_gob_til_R
80     #8\XINT_RQ_end_viii
81     #7\XINT_RQ_end_vii
82     #6\XINT_RQ_end_vi
83     #5\XINT_RQ_end_v
84     #4\XINT_RQ_end_iv
85     #3\XINT_RQ_end_iii
86     #2\XINT_RQ_end_ii
87     \R\XINT_RQ_end_i
88     \Z #2#3#4#5#6#7#8%
89 }%
90 \def\XINT_RQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
91 \def\XINT_RQ_end_vii  #1\Z #2#3#4#5#6#7#8#9\Z { #8#9000}%
92 \def\XINT_RQ_end_vi   #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#900}%
93 \def\XINT_RQ_end_v    #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90}%
94 \def\XINT_RQ_end_iv   #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#9}%
95 \def\XINT_RQ_end_iii  #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
96 \def\XINT_RQ_end_ii   #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
97 \def\XINT_RQ_end_i    \Z #1#2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%
```

#### 3.5 `\XINT_OQ`

```
98 \def\XINT_OQ #1#2#3#4#5#6#7#8#9%
99 {%
100   \xint_gob_til_R #9\XINT_OQ_end_a\R\XINT_OQ {#9#8#7#6#5#4#3#2#1}%
101 }%
102 \def\XINT_OQ_end_a\R\XINT_OQ #1#2\Z
103 {%
104   \XINT_OQ_end_b #1\Z
```

```

105 }%
106 \def\XINT_OQ_end_b #1#2#3#4#5#6#7#8%
107 {%
108   \xint_gob_til_R
109     #8\XINT_OQ_end_viii
110     #7\XINT_OQ_end_vii
111     #6\XINT_OQ_end_vi
112     #5\XINT_OQ_end_v
113     #4\XINT_OQ_end_iv
114     #3\XINT_OQ_end_iii
115     #2\XINT_OQ_end_ii
116     \R\XINT_OQ_end_i
117     \Z #2#3#4#5#6#7#8%
118 }%
119 \def\XINT_OQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
120 \def\XINT_OQ_end_vii #1\Z #2#3#4#5#6#7#8#9\Z { #8#90000000}%
121 \def\XINT_OQ_end_vi #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#90000000}%
122 \def\XINT_OQ_end_v #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90000000}%
123 \def\XINT_OQ_end_iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#90000000}%
124 \def\XINT_OQ_end_iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#90000000}%
125 \def\XINT_OQ_end_ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#90000000}%
126 \def\XINT_OQ_end_i #1\Z #2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

### 3.6 \XINT\_cuz

```

127 \edef\xint_cleanupzeros_andstop #1#2#3#4%
128 {%
129   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
130 }%
131 \def\xint_cleanupzeros_nostop #1#2#3#4%
132 {%
133   \the\numexpr #1#2#3#4\relax
134 }%
135 \def\XINT_rev_andcuz #1%
136 {%
137   \expandafter\xint_cleanupzeros_andstop
138   \romannumeral0\XINT_rord_main {}#1%
139   \xint_relax
140   \xint_bye\xint_bye\xint_bye\xint_bye
141   \xint_bye\xint_bye\xint_bye\xint_bye
142   \xint_relax
143 }%

```

routine `CleanUpZeros`. Utilisée en particulier par la soustraction.

INPUT: longueur **\*\*multiple de 4\*\*** (<-- ATTENTION)

OUTPUT: on a retiré tous les leading zéros, on n'est **\*\*plus\*\*** nécessairement de longueur 4n

Délimiteur pour `_main`: `\W\W\W\W\W\W\W\Z` avec SEPT `\W`

```

144 \def\XINT_cuz #1%
145 {%
146   \XINT_cuz_loop #1\W\W\W\W\W\W\W\Z%
147 }%
148 \def\XINT_cuz_loop #1#2#3#4#5#6#7#8%
149 {%
150   \xint_gob_til_W #8\xint_cuz_end_a\W

```

### 3 Package *xintcore* implementation

```
151 \xint_gob_til_Z #8\xint_cuz_end_A\Z
152 \XINT_cuz_check_a {#1#2#3#4#5#6#7#8}%
153 }%
154 \def\xint_cuz_end_a #1\XINT_cuz_check_a #2%
155 {%
156 \xint_cuz_end_b #2%
157 }%
158 \edef\xint_cuz_end_b #1#2#3#4#5\Z
159 {%
160 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
161 }%
162 \def\xint_cuz_end_A \Z\XINT_cuz_check_a #1{ 0}%
163 \def\XINT_cuz_check_a #1%
164 {%
165 \expandafter\XINT_cuz_check_b\the\numexpr #1\relax
166 }%
167 \def\XINT_cuz_check_b #1%
168 {%
169 \xint_gob_til_zero #1\xint_cuz_backtoloop 0\XINT_cuz_stop #1%
170 }%
171 \def\XINT_cuz_stop #1\W #2\Z{ #1}%
172 \def\xint_cuz_backtoloop 0\XINT_cuz_stop 0{\XINT_cuz_loop }%
```

#### 3.7 `\xintNum`

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

1.05 defines `\xintiNum`, which allows redefinition of `\xintNum` by `xintfrac.sty` Slightly modified in 1.06b (`\R->\xint_relax`) to avoid initial re-scan of input stack (while still allowing empty #1). In versions earlier than 1.09a it was entirely up to the user to apply `\xintnum`; starting with 1.09a arithmetic macros of `xint.sty` (like earlier already `xintfrac.sty` with its own `\xintnum`) make use of `\xintnum`. This allows arguments to be count registers, or even `\numexpr` arbitrary long expressions (with the trick of braces, see the user documentation).

Note (22/06/14): `\xintiNum` jamais utilisé sous ce nom, le supprimer? `\XINT_num` maintenant utilisé par le parseur de `xintexpr`.

```
173 \def\xintiNum {\romannumeral0\xintinum }%
174 \def\xintinum #1%
175 {%
176 \expandafter\XINT_num_loop
177 \romannumeral-`0#1\xint_relax\xint_relax\xint_relax\xint_relax
178 \xint_relax\xint_relax\xint_relax\xint_relax\Z
179 }%
180 \let\xintNum\xintiNum \let\xintnum\xintinum
181 \def\XINT_num #1%
182 {%
183 \XINT_num_loop #1\xint_relax\xint_relax\xint_relax\xint_relax
184 \xint_relax\xint_relax\xint_relax\xint_relax\Z
185 }%
186 \def\XINT_num_loop #1#2#3#4#5#6#7#8%
187 {%
188 \xint_gob_til_xint_relax #8\XINT_num_end\xint_relax
189 \XINT_num_NumEight #1#2#3#4#5#6#7#8%
190 }%
191 \edef\XINT_num_end\xint_relax\XINT_num_NumEight #1\xint_relax #2\Z
```

### 3 Package *xintcore* implementation

```
192 {%
193   \noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_\relax
194 }%
195 \def\xINT_num_NumEight #1#2#3#4#5#6#7#8%
196 {%
197   \ifnum \numexpr #1#2#3#4#5#6#7#8+\xint_c_= \xint_c_
198     \xint_afterfi {\expandafter\xINT_num_keepsign_a
199                   \the\numexpr #1#2#3#4#5#6#7#81\relax}%
200   \else
201     \xint_afterfi {\expandafter\xINT_num_finish
202                   \the\numexpr #1#2#3#4#5#6#7#8\relax}%
203   \fi
204 }%
205 \def\xINT_num_keepsign_a #1%
206 {%
207   \xint_gob_til_one#1\xINT_num_gobackto loop 1\xINT_num_keepsign_b
208 }%
209 \def\xINT_num_gobackto loop 1\xINT_num_keepsign_b {\xINT_num_loop }%
210 \def\xINT_num_keepsign_b #1{\xINT_num_loop -}%
211 \def\xINT_num_finish #1\xint_relax #2\Z { #1}%
```

#### 3.8 `\xintSgn`, `\xintiiSgn`, `\XINT_Sgn`, `\XINT_cntSgn`

Changed in 1.05. Earlier code was unnecessarily strange. 1.09a with `\xintnum`

1.09i defines `\XINT_Sgn` and `\XINT_cntSgn` (was `\XINT__Sgn` in 1.09i) for reasons of internal optimizations.

`xintfrac.sty` will overwrite `\xintsgn` with use of `\xintraw` rather than `\xintnum`, naturally.

```
212 \def\xintiiSgn {\romannumeral0\xintiisgn }%
213 \def\xintiisgn #1%
214 {%
215   \expandafter\xINT_sgn \romannumeral-`0#1\Z%
216 }%
217 \def\xintSgn {\romannumeral0\xintsgn }%
218 \def\xintsgn #1%
219 {%
220   \expandafter\xINT_sgn \romannumeral0\xintnum{#1}\Z%
221 }%
222 \def\xINT_sgn #1#2\Z
223 {%
224   \xint_UDzerominusfork
225   #1-{\ 0}%
226   0#1{ -1}%
227   0-{\ 1}%
228   \krof
229 }%
230 \def\xINT_Sgn #1#2\Z
231 {%
232   \xint_UDzerominusfork
233   #1-{\0}%
234   0#1{-1}%
235   0-{\1}%
236   \krof
```

```

237 }%
238 \def\XINT_cntSgn #1#2\Z
239 {%
240   \xint_UDzerominusfork
241   #1-\xint_c_
242   0#1\m@ne % I will not allocate a count only for -1?
243   0-\xint_c_i
244   \krof
245 }%

```

### 3.9 `\xintiOpp`

`\xintnum` added in 1.09a

```

246 \def\xintiOpp {\romannumeral0\xintiopp }%
247 \def\xintiopp #1%
248 {%
249   \expandafter\XINT_opp \romannumeral-\`0#1%
250 }%
251 \def\xintiOpp {\romannumeral0\xintiopp }%
252 \def\xintiopp #1%
253 {%
254   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
255 }%
256 \let\xintOpp\xintiOpp \let\xintopp\xintiopp
257 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
258 \def\XINT_opp #1%
259 {%
260   \xint_UDzerominusfork
261   #1-{ 0}%      zero
262   0#1{ }%      negative
263   0-{ -#1}%    positive
264   \krof
265 }%

```

### 3.10 `\xintiAbs`, `\xintiiAbs`

Release 1.09a has now `\xintiabs` which does `\xintnum` and this is inherited by `DecSplit`, by `Sqr`, and macros of `xintgcd.sty`. Attention, car ces macros de toute façon doivent passer à la valeur absolue et donc en profite pour faire le `\xintnum`, mais pour optimisation sans overhead il vaut mieux utiliser `\xintiiAbs` ou autre point d'accès.

```

266 \def\xintiiAbs {\romannumeral0\xintiiabs }%
267 \def\xintiiabs #1%
268 {%
269   \expandafter\XINT_abs \romannumeral-\`0#1%
270 }%
271 \def\xintiAbs {\romannumeral0\xintiabs }%
272 \def\xintiabs #1%
273 {%
274   \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
275 }%
276 \let\xintAbs\xintiAbs \let\xintabs\xintiabs

```

### 3 Package *xintcore* implementation

```
277 \def\XINT_Abs #1{\romannumeral0\XINT_abs #1}%
278 \def\XINT_abs #1%
279 {%
280   \xint_UDsignfork
281   #1{ }%
282   -{ #1}%
283   \krof
284 }%
```

-----  
ARITHMETIC OPERATIONS: ADDITION, SUBTRACTION, SUMS, MULTIPLICATION, PRODUCTS, FACTORIAL, POWERS, EUCLIDEAN DIVISION.

Release 1.03 re-organizes sub-routines to facilitate future developments: the diverse variants of addition, with diverse conditions on inputs and output are first listed; they will be used in multiplication, or in the summation, or in the power routines. I am aware that the commenting is close to non-existent, sorry about that.

ADDITION I: \XINT\_add\_A

INPUT:

\romannumeral0\XINT\_add\_A 0{<N1>\W\X\Y\Z <N2>\W\X\Y\Z

1. <N1> et <N2> renversés
2. de longueur 4n (avec des leading zéros éventuels)
3. l'un des deux ne doit pas se terminer par 0000

[Donc on peut avoir 0000 comme input si l'autre est >0 et ne se termine pas en 0000 bien sûr]. On peut avoir l'un des deux vides. Mais alors l'autre ne doit être ni vide ni 0000.

OUTPUT: la somme <N1>+<N2>, ordre normal, plus sur 4n, pas de leading zeros La procédure est plus rapide lorsque <N1> est le plus court des deux.

Nota bene: (30 avril 2013). J'ai une version qui est deux fois plus rapide sur des nombres d'environ 1000 chiffres chacun, et qui commence à être avantageuse pour des nombres d'au moins 200 chiffres. Cependant il serait vraiment compliqué d'en étendre l'utilisation aux emplois de l'addition dans les autres routines, comme celle de multiplication ou celle de division; et son implémentation ajouterait au minimum la mesure de la longueur des summands.

```
285 \def\XINT_add_A #1#2#3#4#5#6%
286 {%
287   \xint_gob_til_W #3\xint_add_az\W
288   \XINT_add_AB #1{#3#4#5#6}{#2}%
289 }%
290 \def\xint_add_az\W\XINT_add_AB #1#2%
291 {%
292   \XINT_add_AC_checkcarry #1%
293 }%
```

ici #2 est prévu pour l'addition, mais attention il devra être renversé pour \numexpr. #3 = résultat partiel. #4 = chiffres qui restent. On vérifie si le deuxième nombre s'arrête.

```
294 \def\XINT_add_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
295 {%
296   \xint_gob_til_W #5\xint_add_bz\W
297   \XINT_add_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
298 }%
299 \def\XINT_add_ABE #1#2#3#4#5#6%
300 {%
301   \expandafter\XINT_add_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
```

### 3 Package *xintcore* implementation

```
302 }%
303 \def\XINT_add_ABEA #1#2#3.#4%
304 {%
305   \XINT_add_A #2{#3#4}%
306 }%
```

ici le deuxième nombre est fini #6 part à la poubelle, #2#3#4#5 est le #2 dans \XINT\_add\_AB on ne vérifie pas la retenue cette fois, mais les fois suivantes

```
307 \def\xint_add_bz\W\XINT_add_ABE #1#2#3#4#5#6%
308 {%
309   \expandafter\XINT_add_CC\the\numexpr #1+10#5#4#3#2.%
310 }%
311 \def\XINT_add_CC #1#2#3.#4%
312 {%
313   \XINT_add_AC_checkcarry #2{#3#4}% on va examiner et \'eliminer #2
314 }%
```

retenue plus chiffres qui restent de l'un des deux nombres. #2 = résultat partiel #3#4#5#6 = summand, avec plus significatif à droite

```
315 \def\XINT_add_AC_checkcarry #1%
316 {%
317   \xint_gob_til_zero #1\xint_add_AC_nocarry 0\XINT_add_C
318 }%
319 \def\xint_add_AC_nocarry 0\XINT_add_C #1#2\W\X\Y\Z
320 {%
321   \expandafter
322   \xint_cleanupzeros_andstop
323   \romannumeral0%
324   \XINT_rord_main {}#2%
325   \xint_relax
326   \xint_bye\xint_bye\xint_bye\xint_bye
327   \xint_bye\xint_bye\xint_bye\xint_bye
328   \xint_relax
329   #1%
330 }%
331 \def\XINT_add_C #1#2#3#4#5%
332 {%
333   \xint_gob_til_W #2\xint_add_cz\W
334   \XINT_add_CD {#5#4#3#2}{#1}%
335 }%
336 \def\XINT_add_CD #1%
337 {%
338   \expandafter\XINT_add_CC\the\numexpr 1+10#1.%
339 }%
340 \def\xint_add_cz\W\XINT_add_CD #1#2{ 1#2}%
```

Addition II: \XINT\_addr\_A.

INPUT: \romannumeral0\XINT\_addr\_A 0{<N1>\W\X\Y\Z <N2>\W\X\Y\Z

Comme \XINT\_add\_A, la différence principale c'est qu'elle donne son résultat aussi \*sur 4n\*, renversé. De plus cette variante accepte que l'un ou même les deux inputs soient vides. Utilisé par la sommation et par la division (pour les quotients). Et aussi par la multiplication d'ailleurs.

INPUT: comme pour \XINT\_add\_A



### 3 Package *xintcore* implementation

1. <N1> et <N2> renversés
  2. de longueur 4n (avec des leading zéros éventuels)
  3. l'un des deux ne doit pas se terminer par 0000
- OUTPUT: la somme <N1>+<N2>, \*aussi renversée\* et \*sur 4n\*

```
341 \def\XINT_addr_A #1#2#3#4#5#6%
342 {%
343   \xint_gob_til_W #3\xint_addr_az\W
344   \XINT_addr_B #1{#3#4#5#6}{#2}%
345 }%
346 \def\xint_addr_az\W\XINT_addr_B #1#2%
347 {%
348   \XINT_addr_AC_checkcarry #1%
349 }%
350 \def\XINT_addr_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
351 {%
352   \xint_gob_til_W #5\xint_addr_bz\W
353   \XINT_addr_E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
354 }%
355 \def\XINT_addr_E #1#2#3#4#5#6%
356 {%
357   \expandafter\XINT_addr_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
358 }%
359 \def\XINT_addr_ABEA #1#2#3#4#5#6#7%
360 {%
361   \XINT_addr_A #2{#7#6#5#4#3}%
362 }%
363 \def\xint_addr_bz\W\XINT_addr_E #1#2#3#4#5#6%
364 {%
365   \expandafter\XINT_addr_CC\the\numexpr #1+10#5#4#3#2\relax
366 }%
367 \def\XINT_addr_CC #1#2#3#4#5#6#7%
368 {%
369   \XINT_addr_AC_checkcarry #2{#7#6#5#4#3}%
370 }%
371 \def\XINT_addr_AC_checkcarry #1%
372 {%
373   \xint_gob_til_zero #1\xint_addr_AC_nocarry 0\XINT_addr_C
374 }%
375 \def\xint_addr_AC_nocarry 0\XINT_addr_C #1#2\W\X\Y\Z { #1#2}%
376 \def\XINT_addr_C #1#2#3#4#5%
377 {%
378   \xint_gob_til_W #2\xint_addr_cz\W
379   \XINT_addr_D {#5#4#3#2}{#1}%
380 }%
381 \def\XINT_addr_D #1%
382 {%
383   \expandafter\XINT_addr_CC\the\numexpr 1+10#1\relax
384 }%
385 \def\xint_addr_cz\W\XINT_addr_D #1#2{ #21000}%
```

ADDITION III, \XINT\_adm\_A

INPUT:\romannumeral0\XINT\_adm\_A 0{<N1>\W\X\Y\Z <N2>\W\X\Y\Z

### 3 Package *xintcore* implementation

1. <N1> et <N2> renversés
  2. <N1> de longueur 4n ; <N2> non
  3. <N2> est \*garanti au moins aussi long\* que <N1>
- OUTPUT: la somme <N1>+<N2>, ordre normal, pas sur 4n, leading zeros retirés. Utilisé par la multiplication.

```
386 \def\XINT_addm_A #1#2#3#4#5#6%
387 {%
388   \xint_gob_til_W #3\xint_addm_az\W
389   \XINT_addm_AB #1#{#3#4#5#6}{#2}%
390 }%
391 \def\xint_addm_az\W\XINT_addm_AB #1#2%
392 {%
393   \XINT_addm_AC_checkcarry #1%
394 }%
395 \def\XINT_addm_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
396 {%
397   \XINT_addm_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
398 }%
399 \def\XINT_addm_ABE #1#2#3#4#5#6%
400 {%
401   \expandafter\XINT_addm_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
402 }%
403 \def\XINT_addm_ABEA #1#2#3.#4%
404 {%
405   \XINT_addm_A #2{#3#4}%
406 }%
407 \def\XINT_addm_AC_checkcarry #1%
408 {%
409   \xint_gob_til_zero #1\xint_addm_AC_nocarry 0\XINT_addm_C
410 }%
411 \def\xint_addm_AC_nocarry 0\XINT_addm_C #1#2\W\X\Y\Z
412 {%
413   \expandafter
414   \xint_cleanupzeros_andstop
415   \romannumeral0%
416   \XINT_rord_main {}#2%
417   \xint_relax
418   \xint_bye\xint_bye\xint_bye\xint_bye
419   \xint_bye\xint_bye\xint_bye\xint_bye
420   \xint_relax
421   #1%
422 }%
423 \def\XINT_addm_C #1#2#3#4#5%
424 {%
425   \xint_gob_til_W
426   #5\xint_addm_cw
427   #4\xint_addm_cx
428   #3\xint_addm_cy
429   #2\xint_addm_cz
430   \W\XINT_addm_CD {#5#4#3#2}{#1}%
431 }%
432 \def\XINT_addm_CD #1%
```

### 3 Package *xintcore* implementation

```

433 {%
434   \expandafter\XINT_addm_CC\the\numexpr 1+10#1.%
435 }%
436 \def\XINT_addm_CC #1#2#3.#4%
437 {%
438   \XINT_addm_AC_checkcarry #2{#3#4}%
439 }%
440 \def\xint_addm_cw
441   #1\xint_addm_cx
442   #2\xint_addm_cy
443   #3\xint_addm_cz
444   \W\XINT_addm_CD
445 {%
446   \expandafter\XINT_addm_CDw\the\numexpr 1+#1#2#3.%
447 }%
448 \def\XINT_addm_CDw #1.#2#3\X\Y\Z
449 {%
450   \XINT_addm_end #1#3%
451 }%
452 \def\xint_addm_cx
453   #1\xint_addm_cy
454   #2\xint_addm_cz
455   \W\XINT_addm_CD
456 {%
457   \expandafter\XINT_addm_CDx\the\numexpr 1+#1#2.%
458 }%
459 \def\XINT_addm_CDx #1.#2#3\Y\Z
460 {%
461   \XINT_addm_end #1#3%
462 }%
463 \def\xint_addm_cy
464   #1\xint_addm_cz
465   \W\XINT_addm_CD
466 {%
467   \expandafter\XINT_addm_CDy\the\numexpr 1+#1.%
468 }%
469 \def\XINT_addm_CDy #1.#2#3\Z
470 {%
471   \XINT_addm_end #1#3%
472 }%
473 \def\xint_addm_cz\W\XINT_addm_CD #1#2#3{\XINT_addm_end #1#3}%
474 \edef\XINT_addm_end #1#2#3#4#5%
475   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5\relax}%

ADDITION IV, variante \XINT_addp_A
INPUT: \romannumeral0\XINT_addp_A 0{<N1>\W\X\Y\Z <N2>\W\X\Y\Z
1. <N1> et <N2> renversés
2. <N1> de longueur 4n ; <N2> non
3. <N2> est *garanti au moins aussi long* que <N1>
OUTPUT: la somme <N1>+<N2>, dans l'ordre renversé, sur 4n, et en faisant attention de ne pas terminer en 0000. Utilisé par la multiplication servant pour le calcul des puissances.

476 \def\XINT_addp_A #1#2#3#4#5#6%

```

### 3 Package *xintcore* implementation

```
477 {%
478   \xint_gob_til_W #3\xint_addp_az\W
479   \XINT_addp_AB #1{#3#4#5#6}{#2}%
480 }%
481 \def\xint_addp_az\W\XINT_addp_AB #1#2%
482 {%
483   \XINT_addp_AC_checkcarry #1%
484 }%
485 \def\XINT_addp_AC_checkcarry #1%
486 {%
487   \xint_gob_til_zero #1\xint_addp_AC_nocarry 0\XINT_addp_C
488 }%
489 \def\xint_addp_AC_nocarry 0\XINT_addp_C
490 {%
491   \XINT_addp_F
492 }%
493 \def\XINT_addp_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
494 {%
495   \XINT_addp_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
496 }%
497 \def\XINT_addp_ABE #1#2#3#4#5#6%
498 {%
499   \expandafter\XINT_addp_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
500 }%
501 \def\XINT_addp_ABEA #1#2#3#4#5#6#7%
502 {%
503   \XINT_addp_A #2{#7#6#5#4#3}%<-- attention on met donc `a droite
504 }%
505 \def\XINT_addp_C #1#2#3#4#5%
506 {%
507   \xint_gob_til_W
508   #5\xint_addp_cw
509   #4\xint_addp_cx
510   #3\xint_addp_cy
511   #2\xint_addp_cz
512   \W\XINT_addp_CD {#5#4#3#2}{#1}%
513 }%
514 \def\XINT_addp_CD #1%
515 {%
516   \expandafter\XINT_addp_CC\the\numexpr 1+10#1\relax
517 }%
518 \def\XINT_addp_CC #1#2#3#4#5#6#7%
519 {%
520   \XINT_addp_AC_checkcarry #2{#7#6#5#4#3}%
521 }%
522 \def\xint_addp_cw
523   #1\xint_addp_cx
524   #2\xint_addp_cy
525   #3\xint_addp_cz
526   \W\XINT_addp_CD
527 {%
528   \expandafter\XINT_addp_CDw\the\numexpr \xint_c_i+10#1#2#3\relax
```

### 3 Package *xintcore* implementation

```
529 }%
530 \def\XINT_addp_CDw #1#2#3#4#5#6%
531 {%
532   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDw_zeros
533   0000\XINT_addp_endDw #2#3#4#5%
534 }%
535 \def\XINT_addp_endDw_zeros 0000\XINT_addp_endDw 0000#1\X\Y\Z{ #1}%
536 \def\XINT_addp_endDw #1#2#3#4#5\X\Y\Z{ #5#4#3#2#1}%
537 \def\xint_addp_cx
538   #1\xint_addp_cy
539   #2\xint_addp_cz
540   \W\XINT_addp_CD
541 {%
542   \expandafter\XINT_addp_CDx\the\numexpr \xint_c_i+100#1#2\relax
543 }%
544 \def\XINT_addp_CDx #1#2#3#4#5#6%
545 {%
546   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDx_zeros
547   0000\XINT_addp_endDx #2#3#4#5%
548 }%
549 \def\XINT_addp_endDx_zeros 0000\XINT_addp_endDx 0000#1\Y\Z{ #1}%
550 \def\XINT_addp_endDx #1#2#3#4#5\Y\Z{ #5#4#3#2#1}%
551 \def\xint_addp_cy #1\xint_addp_cz\W\XINT_addp_CD
552 {%
553   \expandafter\XINT_addp_CDy\the\numexpr \xint_c_i+1000#1\relax
554 }%
555 \def\XINT_addp_CDy #1#2#3#4#5#6%
556 {%
557   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDy_zeros
558   0000\XINT_addp_endDy #2#3#4#5%
559 }%
560 \def\XINT_addp_endDy_zeros 0000\XINT_addp_endDy 0000#1\Z{ #1}%
561 \def\XINT_addp_endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
562 \def\xint_addp_cz\W\XINT_addp_CD #1#2{ #21000}%
563 \def\XINT_addp_F #1#2#3#4#5%
564 {%
565   \xint_gob_til_W
566   #5\xint_addp_Gw
567   #4\xint_addp_Gx
568   #3\xint_addp_Gy
569   #2\xint_addp_Gz
570   \W\XINT_addp_G {#2#3#4#5}{#1}%
571 }%
572 \def\XINT_addp_G #1#2%
573 {%
574   \XINT_addp_F {#2#1}%
575 }%
576 \def\xint_addp_Gw
577   #1\xint_addp_Gx
578   #2\xint_addp_Gy
579   #3\xint_addp_Gz
580   \W\XINT_addp_G #4%
```

### 3 Package *xintcore* implementation

```
581 {%
582   \xint_gob_til_zeros_iv #3#2#10\XINT_addp_endGw_zeros
583   0000\XINT_addp_endGw #3#2#10%
584 }%
585 \def\xint_addp_endGw_zeros 0000\XINT_addp_endGw 0000#1\X\Y\Z{ #1}%
586 \def\xint_addp_endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
587 \def\xint_addp_Gx
588   #1\xint_addp_Gy
589   #2\xint_addp_Gz
590   \W\XINT_addp_G #3%
591 {%
592   \xint_gob_til_zeros_iv #2#100\XINT_addp_endGx_zeros
593   0000\XINT_addp_endGx #2#100%
594 }%
595 \def\xint_addp_endGx_zeros 0000\XINT_addp_endGx 0000#1\Y\Z{ #1}%
596 \def\xint_addp_endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
597 \def\xint_addp_Gy
598   #1\xint_addp_Gz
599   \W\XINT_addp_G #2%
600 {%
601   \xint_gob_til_zeros_iv #1000\XINT_addp_endGy_zeros
602   0000\XINT_addp_endGy #1000%
603 }%
604 \def\xint_addp_endGy_zeros 0000\XINT_addp_endGy 0000#1\Z{ #1}%
605 \def\xint_addp_endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
606 \def\xint_addp_Gz\W\XINT_addp_G #1#2{ #2}%
```

#### 3.11 *\xintiAdd*, *\xintiiAdd*

ADDITION [algo plus efficace lorsque le premier argument plus long que le second]

Note (octobre 2014, pendant la préparation de la sortie de 1.1)

Je n'aurais pas dû l'appeler *\xintAdd*, mais seulement *\xintiAdd*. Le format de sortie de *\xintAdd* est modifié par *xintfrac.sty*, celui de *\xintiAdd* ne bouge pas, et *\xintiiAdd* reste la version stricte.

```
607 \def\xintiiAdd {\romannumeral0\xintiiadd }%
608 \def\xintiiadd #1{\expandafter\xint_iiadd\romannumeral-`0#1\Z }%
609 \def\xint_iiadd #1#2\Z #3%
610 {%
611   \expandafter\XINT_add_fork\expandafter #1\romannumeral-`0#3\Z #2\Z
612 }%
613 \def\xintiAdd {\romannumeral0\xintiadd }%
614 \def\xintiadd #1%
615 {%
616   \expandafter\xint_add\romannumeral0\xintnum{#1}\Z
617 }%
618 \def\xint_add #1#2\Z #3%
619 {%
620   \expandafter\XINT_add_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
621 }%
622 \let\xintAdd\xintiAdd \let\xintadd\xintiadd
623 \def\XINT_add_fork #1#2%
624 {%
```

### 3 Package *xintcore* implementation

```
625 \xint_UDzerofork
626   #1\XINT_add_firstiszero
627   #2\XINT_add_secondiszero
628   0{ }%
629 \krof
630 \xint_UDsignsfork
631   #1#2\XINT_add_minusminus
632   #1-\XINT_add_minusplus
633   #2-\XINT_add_plusminus
634   --\XINT_add_plusplus
635 \krof #1#2%
636 }%
637 \def\xint_add_firstiszero #1\krof #2#3\Z #4\Z { #3}%
638 \def\xint_add_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
639 \def\xint_add_plusplus #1#2#3\Z #4\Z {\XINT_add_pre {#1#4}{#2#3}}%
640 \def\xint_add_minusminus #1#2#3\Z #4\Z
641   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#3}}%
642 \def\xint_add_minusplus #1#2#3\Z #4\Z {\XINT_sub_pre {#2#3}{#4}}%
643 \def\xint_add_plusminus #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#3}}%
```

#### positive summands

```
644 \def\xint_add_pre #1%
645 {%
646   \expandafter\xint_add_pre_b\expandafter
647   {\romannumeral0\XINT_RQ { }#1\R\R\R\R\R\R\R\R\Z }%
648 }%
649 \def\xint_add_pre_b #1#2%
650 {%
651   \expandafter\xint_add_A
652   \expandafter0\expandafter{\expandafter}%
653   \romannumeral0\XINT_RQ { }#2\R\R\R\R\R\R\R\R\Z
654   \W\X\Y\Z #1\W\X\Y\Z
655 }%
```

### 3.12 *\xintiSub*, *\xintiiSub*

Release 1.09a has *\xintnum* added into *\xintiSub*.

```
656 \def\xintiiSub {\romannumeral0\xintiisub }%
657 \def\xintiisub #1{\expandafter\xint_iisub\romannumeral-`0#1\Z }%
658 \def\xint_iisub #1#2\Z #3%
659 {%
660   \expandafter\xint_sub_fork\expandafter #1\romannumeral-`0#3\Z #2\Z
661 }%
662 \def\xintiSub {\romannumeral0\xintisub }%
663 \def\xintisub #1%
664 {%
665   \expandafter\xint_sub\romannumeral0\xintnum{#1}\Z
666 }%
667 \def\xint_sub #1#2\Z #3%
668 {%
669   \expandafter\xint_sub_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
```

### 3 Package *xintcore* implementation

```

670 }%
671 \let\xintSub\xintiSub \let\xintsub\xintisub
672 \def\XINT_sub_fork #1#2%
673 {%
674   \xint_UDzerofork
675   #1\XINT_sub_firstiszero
676   #2\XINT_sub_secondiszero
677   0}%
678 \krof
679 \xint_UDsignsfork
680   #1#2\XINT_sub_minusminus
681   #1-\XINT_sub_minusplus
682   #2-\XINT_sub_plusminus
683   --\XINT_sub_plusplus
684 \krof #1#2%
685 }%
686 \def\XINT_sub_firstiszero #1\krof #2#3\Z #4\Z {\XINT_opp #3}%
687 \def\XINT_sub_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
688 \def\XINT_sub_plusplus #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#2#3}}%
689 \def\XINT_sub_minusminus #1#2#3\Z #4\Z {\XINT_sub_pre {#3}{#4}}%
690 \def\XINT_sub_minusplus #1#2#3\Z #4\Z
691   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#2#3}}%
692 \def\XINT_sub_plusminus #1#2#3\Z #4\Z {\XINT_add_pre {#1#4}{#3}}%

```

SOUSTRACTION A-B avec A premier argument, B second argument de `\xintSub` et ensuite `\XINT_sub_pre` ici

```

693 \def\XINT_sub_pre #1%
694 {%
695   \expandafter\XINT_sub_pre_b\expandafter
696   {\romannumeral0\XINT_RQ { }#1\R\R\R\R\R\R\R\R\Z }%
697 }%
698 \def\XINT_sub_pre_b #1#2%
699 {%
700   \expandafter\XINT_sub_A
701   \expandafter1\expandafter{\expandafter}%
702   \romannumeral0\XINT_RQ { }#2\R\R\R\R\R\R\R\R\Z
703   \W\X\Y\Z #1 \W\X\Y\Z
704 }%

```

`\romannumeral0\XINT_sub_A 1{<N1>\W\X\Y\Z<N2>\W\X\Y\Z`

N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS POUR QUE LEURS LONGUEURS À CHACUN SOIENT MULTIPLES DE 4, MAIS AUCUN NE SE TERMINE EN 0000.

output: N2 - N1

Elle donne le résultat dans le **\*\*bon ordre\*\***, avec le bon signe, et sans zéros superflus.

```

705 \def\XINT_sub_A #1#2#3\W\X\Y\Z #4#5#6#7%
706 {%
707   \xint_gob_til_W
708   #4\xint_sub_az
709   \W\XINT_sub_B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
710 }%
711 \def\XINT_sub_B #1#2#3#4#5#6#7%
712 {%

```



### 3 Package *xintcore* implementation

```
713 \xint_gob_til_W
714 #4\xint_sub_bz
715 \W\XINT_sub_onestep #1#2{#7#6#5#4}{#3}%
716 }%
```

d'abord la branche principale #6 = 4 chiffres de N1, plus significatif en \*premier\*, #2#3#4#5 chiffres de N2, plus significatif en \*dernier\* On veut  $N2 - N1$ .

```
717 \def\XINT_sub_onestep #1#2#3#4#5#6%
718 {%
719 \expandafter\XINT_sub_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
720 }%
```

ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE

```
721 \def\XINT_sub_backtoA #1#2#3.#4%
722 {%
723 \XINT_sub_A #2{#3#4}%
724 }%
725 \def\xint_sub_bz
726 \W\XINT_sub_onestep #1#2#3#4#5#6#7%
727 {%
728 \xint_UDzerofork
729 #1\XINT_sub_C % une retenue
730 0\XINT_sub_D % pas de retenue
731 \krof
732 {#7}#2#3#4#5%
733 }%
734 \def\XINT_sub_D #1#2\W\X\Y\Z
735 {%
736 \expandafter
737 \xint_cleanupzeros_andstop
738 \romannumeral0%
739 \XINT_rord_main { }#2%
740 \xint_relax
741 \xint_bye\xint_bye\xint_bye\xint_bye
742 \xint_bye\xint_bye\xint_bye\xint_bye
743 \xint_relax
744 #1%
745 }%
746 \def\XINT_sub_C #1#2#3#4#5%
747 {%
748 \xint_gob_til_W
749 #2\xint_sub_cz
750 \W\XINT_sub_AC_onestep {#5#4#3#2}{#1}%
751 }%
752 \def\XINT_sub_AC_onestep #1%
753 {%
754 \expandafter\XINT_sub_backtoC\the\numexpr 11#1-\xint_c_i.%
755 }%
756 \def\XINT_sub_backtoC #1#2#3.#4%
757 {%
758 \XINT_sub_AC_checkcarry #2{#3#4}% la retenue va \^etre examin\`ee
759 }%
```

### 3 Package *xintcore* implementation

```
760 \def\XINT_sub_AC_checkcarry #1%
761 {%
762   \xint_gob_til_one #1\xint_sub_AC_nocarry 1\XINT_sub_C
763 }%
764 \def\xint_sub_AC_nocarry 1\XINT_sub_C #1#2\W\X\Y\Z
765 {%
766   \expandafter
767   \XINT_cuz_loop
768   \romannumeral0%
769   \XINT_rord_main {}#2%
770   \xint_relax
771   \xint_bye\xint_bye\xint_bye\xint_bye
772   \xint_bye\xint_bye\xint_bye\xint_bye
773   \xint_relax
774   #1\W\W\W\W\W\W\W\Z
775 }%
776 \def\xint_sub_cz\W\XINT_sub_AC_onestep #1%
777 {%
778   \XINT_cuz
779 }%
780 \def\xint_sub_az\W\XINT_sub_B #1#2#3#4#5#6#7%
781 {%
782   \xint_gob_til_W
783   #4\xint_sub_ez
784   \W\XINT_sub_Eenter #1{#3}#4#5#6#7%
785 }%
```

le premier nombre continue, le résultat sera  $< 0$ .

```
786 \def\XINT_sub_Eenter #1#2%
787 {%
788   \expandafter
789   \XINT_sub_E\expandafter1\expandafter{\expandafter}%
790   \romannumeral0%
791   \XINT_rord_main {}#2%
792   \xint_relax
793   \xint_bye\xint_bye\xint_bye\xint_bye
794   \xint_bye\xint_bye\xint_bye\xint_bye
795   \xint_relax
796   \W\X\Y\Z #1%
797 }%
798 \def\XINT_sub_E #1#2#3#4#5#6%
799 {%
800   \xint_gob_til_W #3\xint_sub_F\W
801   \XINT_sub_Eonestep #1{#6#5#4#3}{#2}%
802 }%
803 \def\XINT_sub_Eonestep #1#2%
804 {%
805   \expandafter\XINT_sub_backtoE\the\numexpr 109999-#2+#1.%
806 }%
807 \def\XINT_sub_backtoE #1#2#3.#4%
808 {%
809   \XINT_sub_E #2{#3#4}%
```

### 3 Package *xintcore* implementation

```
810 }%
811 \def\xint_sub_F\W\XINT_sub_Eonestep #1#2#3#4%
812 {%
813   \xint_UDonezerofork
814   #4#1{\XINT_sub_Fdec 0}% soustraire 1. Et faire signe -
815   #1#4{\XINT_sub_Finc 1}% additionner 1. Et faire signe -
816   10\XINT_sub_DD   % terminer. Mais avec signe -
817   \krof
818   {#3}%
819 }%
820 \def\XINT_sub_DD {\expandafter\xint_minus_thenstop\romannumeral0\XINT_sub_D }%
821 \def\XINT_sub_Fdec #1#2#3#4#5#6%
822 {%
823   \xint_gob_til_W #3\xint_sub_Fdec_finish\W
824   \XINT_sub_Fdec_onestep #1{#6#5#4#3}{#2}%
825 }%
826 \def\XINT_sub_Fdec_onestep #1#2%
827 {%
828   \expandafter\XINT_sub_backtoFdec\the\numexpr 11#2+#1-\xint_c_i.%
829 }%
830 \def\XINT_sub_backtoFdec #1#2#3.#4%
831 {%
832   \XINT_sub_Fdec #2{#3#4}%
833 }%
834 \def\xint_sub_Fdec_finish\W\XINT_sub_Fdec_onestep #1#2%
835 {%
836   \expandafter\xint_minus_thenstop\romannumeral0\XINT_cuz
837 }%
838 \def\XINT_sub_Finc #1#2#3#4#5#6%
839 {%
840   \xint_gob_til_W #3\xint_sub_Finc_finish\W
841   \XINT_sub_Finc_onestep #1{#6#5#4#3}{#2}%
842 }%
843 \def\XINT_sub_Finc_onestep #1#2%
844 {%
845   \expandafter\XINT_sub_backtoFinc\the\numexpr 10#2+#1.%
846 }%
847 \def\XINT_sub_backtoFinc #1#2#3.#4%
848 {%
849   \XINT_sub_Finc #2{#3#4}%
850 }%
851 \def\xint_sub_Finc_finish\W\XINT_sub_Finc_onestep #1#2#3%
852 {%
853   \xint_UDzerofork
854   #1{\expandafter\expandafter\expandafter
855     \xint_minus_thenstop\xint_cleanupzeros_nostop}%
856   0{ -1}%
857   \krof
858   #3%
859 }%
860 \def\xint_sub_ez\W\XINT_sub_Eenter #1%
861 {%
```

### 3 Package *xintcore* implementation

```
862 \xint_UDzerofork
863 #1\XINT_sub_K % il y a une retenue
864 0\XINT_sub_L % pas de retenue
865 \krof
866 }%
867 \def\XINT_sub_L #1\W\X\Y\Z {\XINT_cuz_loop #1\W\W\W\W\W\W\W\Z }%
868 \def\XINT_sub_K #1%
869 {%
870 \expandafter
871 \XINT_sub_KK\expandafter1\expandafter{\expandafter}%
872 \romannumeral0%
873 \XINT_rord_main {}#1%
874 \xint_relax
875 \xint_bye\xint_bye\xint_bye\xint_bye
876 \xint_bye\xint_bye\xint_bye\xint_bye
877 \xint_relax
878 }%
879 \def\XINT_sub_KK #1#2#3#4#5#6%
880 {%
881 \xint_gob_til_W #3\xint_sub_KK_finish\W
882 \XINT_sub_KK_onestep #1{#6#5#4#3}{#2}%
883 }%
884 \def\XINT_sub_KK_onestep #1#2%
885 {%
886 \expandafter\XINT_sub_backtoKK\the\numexpr 109999-#2+#1.%
887 }%
888 \def\XINT_sub_backtoKK #1#2#3.#4%
889 {%
890 \XINT_sub_KK #2{#3#4}%
891 }%
892 \def\xint_sub_KK_finish\W\XINT_sub_KK_onestep #1#2#3%
893 {%
894 \expandafter\xint_minus_thenstop
895 \romannumeral0\XINT_cuz_loop #3\W\W\W\W\W\W\W\Z
896 }%
```

#### 3.13 *\xintiMul*, *\xintiiMul*

##### 1.09a adds *\xintnum*

```
897 \def\xintiiMul {\romannumeral0\xintiimul }%
898 \def\xintiimul #1%
899 {%
900 \expandafter\xint_iimul\expandafter {\romannumeral-`0#1}%
901 }%
902 \def\xint_iimul #1#2%
903 {%
904 \expandafter\XINT_mul_fork \romannumeral-`0#2\Z #1\Z
905 }%
906 \def\xintiMul {\romannumeral0\xintimul }%
907 \def\xintimul #1%
908 {%
909 \expandafter\xint_mul\expandafter {\romannumeral0\xintnum{#1}}%
```

### 3 Package *xintcore* implementation

```

910 }%
911 \def\xint_mul #1#2%
912 {%
913   \expandafter\XINT_mul_fork \romannumeral0\xintnum{#2}\Z #1\Z
914 }%
915 \let\xintMul\xintiMul \let\xintmul\xintimul
916 \def\XINT_Mul #1#2{\romannumeral0\XINT_mul_fork #2\Z #1\Z }%

```

#### MULTIPLICATION

Ici #1#2 = 2e input et #3#4 = 1er input

Release 1.03 adds some overhead to first compute and compare the lengths of the two inputs. The algorithm is asymmetrical and whether the first input is the longest or the shortest sometimes has a strong impact. 50 digits times 1000 digits used to be 5 times faster than 1000 digits times 50 digits. With the new code, the user input order does not matter as it is decided by the routine what is best. This is important for the extension to fractions, as there is no way then to generally control or guess the most frequent sizes of the inputs besides actually computing their lengths.

```

917 \def\XINT_mul_fork #1#2\Z #3#4\Z
918 {%
919   \xint_UDzerofork
920     #1\XINT_mul_zero
921     #3\XINT_mul_zero
922     0}%
923   \krof
924   \xint_UDsignsfork
925     #1#3\XINT_mul_minusminus          % #1 = #3 = -
926     #1-\XINT_mul_minusplus #3}%      % #1 = -
927     #3-\XINT_mul_plusminus #1}%      % #3 = -
928     --\XINT_mul_plusplus #1#3}%
929   \krof
930   {#2}{#4}%
931 }%
932 \def\XINT_mul_zero #1\krof #2#3{ 0}%
933 \def\XINT_mul_minusminus #1#2%
934 {%
935   \expandafter\XINT_mul_choice_a
936   \expandafter{\romannumeral0\xintlength {#2}}%
937   {\romannumeral0\xintlength {#1}}{#1}{#2}%
938 }%
939 \def\XINT_mul_minusplus #1#2#3%
940 {%
941   \expandafter\xint_minus_thenstop\romannumeral0\expandafter
942   \XINT_mul_choice_a
943   \expandafter{\romannumeral0\xintlength {#1#3}}%
944   {\romannumeral0\xintlength {#2}}{#2}{#1#3}%
945 }%
946 \def\XINT_mul_plusminus #1#2#3%
947 {%
948   \expandafter\xint_minus_thenstop\romannumeral0\expandafter
949   \XINT_mul_choice_a
950   \expandafter{\romannumeral0\xintlength {#3}}%
951   {\romannumeral0\xintlength {#1#2}}{#1#2}{#3}%
952 }%

```

### 3 Package *xintcore* implementation

```

953 \def\XINT_mul_plusplus #1#2#3#4%
954 {%
955   \expandafter\XINT_mul_choice_a
956   \expandafter{\romannumeral0\xintlength {#2#4}}%
957   {\romannumeral0\xintlength {#1#3}}{#1#3}{#2#4}%
958 }%
959 \def\XINT_mul_choice_a #1#2%
960 {%
961   \expandafter\XINT_mul_choice_b\expandafter{#2}{#1}%
962 }%
963 \def\XINT_mul_choice_b #1#2%
964 {%
965   \ifnum #1<\xint_c_v
966     \expandafter\XINT_mul_choice_littlebyfirst
967   \else
968     \ifnum #2<\xint_c_v
969       \expandafter\expandafter\expandafter\XINT_mul_choice_littlebysecond
970     \else
971       \expandafter\expandafter\expandafter\XINT_mul_choice_compare
972     \fi
973   \fi
974   {#1}{#2}%
975 }%
976 \def\XINT_mul_choice_littlebyfirst #1#2#3#4%
977 {%
978   \expandafter\XINT_mul_M
979   \expandafter{\the\numexpr #3\expandafter}%
980   \romannumeral0\XINT_RQ { }#4\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
981 }%
982 \def\XINT_mul_choice_littlebysecond #1#2#3#4%
983 {%
984   \expandafter\XINT_mul_M
985   \expandafter{\the\numexpr #4\expandafter}%
986   \romannumeral0\XINT_RQ { }#3\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
987 }%
988 \def\XINT_mul_choice_compare #1#2%
989 {%
990   \ifnum #1>#2
991     \expandafter \XINT_mul_choice_i
992   \else
993     \expandafter \XINT_mul_choice_ii
994   \fi
995   {#1}{#2}%
996 }%
997 \def\XINT_mul_choice_i #1#2%
998 {%
999   \ifnum #1<\numexpr\ifcase \numexpr (#2-\xint_c_iii)/\xint_c_iv\relax
1000     \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1001     \expandafter\XINT_mul_choice_same
1002   \else
1003     \expandafter\XINT_mul_choice_permute
1004   \fi

```

### 3 Package *xintcore* implementation

```

1005 }%
1006 \def\XINT_mul_choice_ii #1#2%
1007 {%
1008   \ifnum #2<\numexpr\ifcase \numexpr (#1-\xint_c_iii)/\xint_c_iv\relax
1009     \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1010     \expandafter\XINT_mul_choice_permute
1011   \else
1012     \expandafter\XINT_mul_choice_same
1013   \fi
1014 }%
1015 \def\XINT_mul_choice_same #1#2%
1016 {%
1017   \expandafter\XINT_mul_enter
1018   \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
1019   \Z\Z\Z\Z #2\W\W\W\W
1020 }%
1021 \def\XINT_mul_choice_permute #1#2%
1022 {%
1023   \expandafter\XINT_mul_enter
1024   \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
1025   \Z\Z\Z\Z #1\W\W\W\W
1026 }%

```

Cette portion de routine d'addition se branche directement sur `_addr_` lorsque le premier nombre est épuisé, ce qui est garanti arriver avant le second nombre. Elle produit son résultat toujours sur  $4n$ , renversé. Ses deux inputs sont garantis sur  $4n$ .

```

1027 \def\XINT_mul_Ar #1#2#3#4#5#6%
1028 {%
1029   \xint_gob_til_Z #6\xint_mul_br\Z\XINT_mul_Br #1{#6#5#4#3}{#2}%
1030 }%
1031 \def\xint_mul_br\Z\XINT_mul_Br #1#2%
1032 {%
1033   \XINT_addr_AC_checkcarry #1%
1034 }%
1035 \def\XINT_mul_Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
1036 {%
1037   \expandafter\XINT_mul_ABEAr
1038   \the\numexpr #1+10#2+#8#7#6#5.{#3}#4\W\X\Y\Z
1039 }%
1040 \def\XINT_mul_ABEAr #1#2#3#4#5#6.#7%
1041 {%
1042   \XINT_mul_Ar #2{#7#6#5#4#3}%
1043 }%

```

<< Petite >> multiplication. `mul_Mr` renvoie le résultat \*à l'envers\*, sur  $4n$   
`\romannumeral0\XINT_mul_Mr {<n>}<N>\Z\Z\Z\Z`  
 Fait la multiplication de  $\langle N \rangle$  par  $\langle n \rangle$ , qui est  $< 10000$ .  $\langle N \rangle$  est présenté \*à l'envers\*, sur  $4n$ .  
 Lorsque  $\langle n \rangle$  vaut 0, donne 0000.

```

1044 \def\XINT_mul_Mr #1%
1045 {%
1046   \expandafter\XINT_mul_Mr_checkifzeroorone\expandafter{\the\numexpr #1}%
1047 }%

```

### 3 Package *xintcore* implementation

```

1048 \def\XINT_mul_Mr_checkifzeroorone #1%
1049 {%
1050   \ifcase #1
1051     \expandafter\XINT_mul_Mr_zero
1052   \or
1053     \expandafter\XINT_mul_Mr_one
1054   \else
1055     \expandafter\XINT_mul_Nr
1056   \fi
1057   {0000}{\#1}%
1058 }%
1059 \def\XINT_mul_Mr_zero #1\Z\Z\Z\Z { 0000}%
1060 \def\XINT_mul_Mr_one #1#2#3#4\Z\Z\Z\Z { #4}%
1061 \def\XINT_mul_Nr #1#2#3#4#5#6#7%
1062 {%
1063   \xint_gob_til_Z #4\xint_mul_pr\Z\XINT_mul_Pr {#1}{#3}{#7#6#5#4}{#2}{#3}%
1064 }%
1065 \def\XINT_mul_Pr #1#2#3%
1066 {%
1067   \expandafter\XINT_mul_Lr\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
1068 }%
1069 \def\XINT_mul_Lr 1#1#2#3#4#5#6#7#8#9%
1070 {%
1071   \XINT_mul_Nr {#1#2#3#4}{#9#8#7#6#5}%
1072 }%
1073 \def\xint_mul_pr\Z\XINT_mul_Pr #1#2#3#4#5%
1074 {%
1075   \xint_gob_til_zeros_iv #1\XINT_mul_Mr_end_nocarry 0000%
1076   \XINT_mul_Mr_end_carry #1{#4}%
1077 }%
1078 \def\XINT_mul_Mr_end_nocarry 0000\XINT_mul_Mr_end_carry 0000#1{ #1}%
1079 \def\XINT_mul_Mr_end_carry #1#2#3#4#5{ #5#4#3#2#1}%

```

<< Petite >> multiplication. renvoie le résultat \*à l'endroit\*, avec \*nettoyage des leading zéros\*.

\romannumeral0\XINT\_mul\_M {<n>}<N>\Z\Z\Z\Z

Fait la multiplication de <N> par <n>, qui est < 10000. <N> est présenté \*à l'envers\*, sur \*4n\*.

```

1080 \def\XINT_mul_M #1%
1081 {%
1082   \expandafter\XINT_mul_M_checkifzeroorone\expandafter{\the\numexpr #1}%
1083 }%
1084 \def\XINT_mul_M_checkifzeroorone #1%
1085 {%
1086   \ifcase #1
1087     \expandafter\XINT_mul_M_zero
1088   \or
1089     \expandafter\XINT_mul_M_one
1090   \else
1091     \expandafter\XINT_mul_N
1092   \fi
1093   {0000}{\#1}%
1094 }%

```



### 3 Package *xintcore* implementation

```

1095 \def\XINT_mul_M_zero #1\Z\Z\Z\Z { 0}%
1096 \def\XINT_mul_M_one #1#2#3#4\Z\Z\Z\Z
1097 {%
1098   \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#4}%
1099 }%
1100 \def\XINT_mul_N #1#2#3#4#5#6#7%
1101 {%
1102   \xint_gob_til_Z #4\xint_mul_p\Z\XINT_mul_P {#1}{#3}{#7#6#5#4}{#2}{#3}%
1103 }%
1104 \def\XINT_mul_P #1#2#3%
1105 {%
1106   \expandafter\XINT_mul_L\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
1107 }%
1108 \def\XINT_mul_L 1#1#2#3#4#5#6#7#8#9%
1109 {%
1110   \XINT_mul_N {#1#2#3#4}{#5#6#7#8#9}%
1111 }%
1112 \def\xint_mul_p\Z\XINT_mul_P #1#2#3#4#5%
1113 {%
1114   \XINT_mul_M_end #1#4%
1115 }%
1116 \edef\XINT_mul_M_end #1#2#3#4#5#6#7#8%
1117 {%
1118   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
1119 }%

```

Routine de multiplication principale (attention délimiteurs modifiés pour 1.08)

Le résultat partiel est toujours maintenu avec significatif à droite et il a un nombre multiple de 4 de chiffres

\romannumeral0\XINT\_mul\_enter <N1>\Z\Z\Z\Z <N2>\W\W\W\W

avec <N1> \*renversé\*, \*longueur 4n\* (zéros éventuellement ajoutés au-delà du chiffre le plus significatif) et <N2> dans l'ordre \*normal\*, et pas forcément longueur 4n. pas de signes.

Pour 1.08: dans \XINT\_mul\_enter et les modifs de 1.03 qui filtrent les courts, on pourrait croire que le second opérande a au moins quatre chiffres; mais le problème c'est que ceci est appelé par \XINT\_sqr. Et de plus \XINT\_sqr est utilisé dans la nouvelle routine d'extraction de racine carrée: je ne veux pas rajouter l'overhead à \XINT\_sqr de voir si sa longueur est au moins 4. Dilemme donc. Il ne semble pas y avoir d'autres accès directs (celui de big fac n'est pas un problème). J'ai presque été tenté de faire du 5x4, mais si on veut maintenir les résultats intermédiaires sur 4n, il y a des complications. Par ailleurs, je modifie aussi un petit peu la façon de coder la suite, compte tenu du style que j'ai développé ultérieurement. Attention terminaison modifiée pour le deuxième opérande.

```

1120 \def\XINT_mul_enter #1\Z\Z\Z\Z #2#3#4#5%
1121 {%
1122   \xint_gob_til_W #5\XINT_mul_exit_a\W
1123   \XINT_mul_start {#2#3#4#5}#1\Z\Z\Z\Z
1124 }%
1125 \def\XINT_mul_exit_a\W\XINT_mul_start #1%
1126 {%
1127   \XINT_mul_exit_b #1%
1128 }%
1129 \def\XINT_mul_exit_b #1#2#3#4%
1130 {%

```

### 3 Package *xintcore* implementation

```

1131 \xint_gob_til_W
1132 #2\XINT_mul_exit_ci
1133 #3\XINT_mul_exit_cii
1134 \W\XINT_mul_exit_ciii #1#2#3#4%
1135 }%
1136 \def\XINT_mul_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
1137 {%
1138 \XINT_mul_M {#1}#2\Z\Z\Z\Z
1139 }%
1140 \def\XINT_mul_exit_cii\W\XINT_mul_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
1141 {%
1142 \XINT_mul_M {#1}#2\Z\Z\Z\Z
1143 }%
1144 \def\XINT_mul_exit_ci\W\XINT_mul_exit_cii
1145 \W\XINT_mul_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
1146 {%
1147 \XINT_mul_M {#1}#2\Z\Z\Z\Z
1148 }%
1149 \def\XINT_mul_start #1#2\Z\Z\Z\Z
1150 {%
1151 \expandafter\XINT_mul_main\expandafter
1152 {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
1153 }%
1154 \def\XINT_mul_main #1#2\Z\Z\Z\Z #3#4#5#6%
1155 {%
1156 \xint_gob_til_W #6\XINT_mul_finish_a\W
1157 \XINT_mul_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
1158 }%
1159 \def\XINT_mul_compute #1#2#3\Z\Z\Z\Z
1160 {%
1161 \expandafter\XINT_mul_main\expandafter
1162 {\romannumeral0\expandafter
1163 \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
1164 \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
1165 \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
1166 }%

```

Ici, le deuxième nombre se termine. Fin du calcul. On utilise la variante `\XINT_addm_A` de l'addition car on sait que le deuxième terme est au moins aussi long que le premier. Lorsque le multiplicateur avait longueur  $4n$ , la dernière addition a fourni le résultat à l'envers, il faut donc encore le renverser.

```

1167 \def\XINT_mul_finish_a\W\XINT_mul_compute #1%
1168 {%
1169 \XINT_mul_finish_b #1%
1170 }%
1171 \def\XINT_mul_finish_b #1#2#3#4%
1172 {%
1173 \xint_gob_til_W
1174 #1\XINT_mul_finish_c
1175 #2\XINT_mul_finish_ci
1176 #3\XINT_mul_finish_cii
1177 \W\XINT_mul_finish_ciii #1#2#3#4%

```

### 3 Package *xintcore* implementation

```

1178 }%
1179 \def\XINT_mul_finish_ciii #1\W #2#3\Z\Z\Z \W\W\W
1180 {%
1181   \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1182   \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
1183 }%
1184 \def\XINT_mul_finish_cii
1185   \W\XINT_mul_finish_ciii #1\W\W #2#3\Z\Z\Z \W\W
1186 {%
1187   \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1188   \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
1189 }%
1190 \def\XINT_mul_finish_ci #1\XINT_mul_finish_ciii #2\W\W\W #3#4\Z\Z\Z \W
1191 {%
1192   \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1193   \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
1194 }%
1195 \def\XINT_mul_finish_c #1\XINT_mul_finish_ciii \W\W\W\W #2#3\Z\Z\Z
1196 {%
1197   \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#2}%
1198 }%

```

#### Variante de la Multiplication

`\romannumeral0\XINT_mulr_enter <N1>\Z\Z\Z\Z <N2>\W\W\W\W`

Ici <N1> est à l'envers sur 4n, et <N2> est à l'endroit, pas sur 4n, comme dans `\XINT_mul_enter`, mais le résultat est lui-même fourni \*à l'envers\*, sur \*4n\* (en faisant attention de ne pas avoir 0000 à la fin).

Utilisé par le calcul des puissances. J'ai modifié dans 1.08 sur le modèle de la nouvelle version de `\XINT_mul_enter`. Je pourrais économiser des macros et fusionner `\XINT_mul_enter` et `\XINT_mulr_enter`. Une autre fois.

```

1199 \def\XINT_mulr_enter #1\Z\Z\Z\Z #2#3#4#5%
1200 {%
1201   \xint_gob_til_W #5\XINT_mulr_exit_a\W
1202   \XINT_mulr_start {#2#3#4#5}#1\Z\Z\Z\Z
1203 }%
1204 \def\XINT_mulr_exit_a\W\XINT_mulr_start #1%
1205 {%
1206   \XINT_mulr_exit_b #1%
1207 }%
1208 \def\XINT_mulr_exit_b #1#2#3#4%
1209 {%
1210   \xint_gob_til_W
1211   #2\XINT_mulr_exit_ci
1212   #3\XINT_mulr_exit_cii
1213   \W\XINT_mulr_exit_ciii #1#2#3#4%
1214 }%
1215 \def\XINT_mulr_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W\W
1216 {%
1217   \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
1218 }%
1219 \def\XINT_mulr_exit_cii\W\XINT_mulr_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
1220 {%

```

### 3 Package *xintcore* implementation

```

1221 \XINT_mul_Mr {#1}#2\Z\Z\Z
1222 }%
1223 \def\XINT_mulr_exit_ci\W\XINT_mulr_exit_cii
1224 \W\XINT_mulr_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
1225 {%
1226 \XINT_mul_Mr {#1}#2\Z\Z\Z
1227 }%
1228 \def\XINT_mulr_start #1#2\Z\Z\Z
1229 {%
1230 \expandafter\XINT_mulr_main\expandafter
1231 {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
1232 }%
1233 \def\XINT_mulr_main #1#2\Z\Z\Z\Z #3#4#5#6%
1234 {%
1235 \xint_gob_til_W #6\XINT_mulr_finish_a\W
1236 \XINT_mulr_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
1237 }%
1238 \def\XINT_mulr_compute #1#2#3\Z\Z\Z\Z
1239 {%
1240 \expandafter\XINT_mulr_main\expandafter
1241 {\romannumeral0\expandafter
1242 \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
1243 \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
1244 \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
1245 }%
1246 \def\XINT_mulr_finish_a\W\XINT_mulr_compute #1%
1247 {%
1248 \XINT_mulr_finish_b #1%
1249 }%
1250 \def\XINT_mulr_finish_b #1#2#3#4%
1251 {%
1252 \xint_gob_til_W
1253 #1\XINT_mulr_finish_c
1254 #2\XINT_mulr_finish_ci
1255 #3\XINT_mulr_finish_cii
1256 \W\XINT_mulr_finish_ciii #1#2#3#4%
1257 }%
1258 \def\XINT_mulr_finish_ciii #1\W #2#3\Z\Z\Z\Z \W\W\W
1259 {%
1260 \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1261 \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
1262 }%
1263 \def\XINT_mulr_finish_cii
1264 \W\XINT_mulr_finish_ciii #1\W\W #2#3\Z\Z\Z\Z \W\W
1265 {%
1266 \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1267 \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
1268 }%
1269 \def\XINT_mulr_finish_ci #1\XINT_mulr_finish_ciii #2\W\W\W #3#4\Z\Z\Z\Z \W
1270 {%
1271 \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1272 \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z

```

```
1273 }%
1274 \def\XINT_mulr_finish_c #1\XINT_mulr_finish_ciii \W\W\W\W #2#3\Z\Z\Z { #2}%
```

### 3.14 `\xintiSqr`, `\xintiiSqr`

```
1275 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1276 \def\xintiisqr #1%
1277 {%
1278   \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiabs{#1}}%
1279 }%
1280 \def\xintiSqr {\romannumeral0\xintisqr }%
1281 \def\xintisqr #1%
1282 {%
1283   \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiabs{#1}}%
1284 }%
1285 \let\xintiSqr\xintiSqr \let\xintsqr\xintisqr
1286 \def\XINT_sqr #1%
1287 {%
1288   \expandafter\XINT_mul_enter
1289     \romannumeral0%
1290     \XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
1291     \Z\Z\Z\Z #1\W\W\W\W
1292 }%
```

### 3.15 `\xintiPow`, `\xintiiPow`

1.02 modified the `\XINT_posprod` routine, the was renamed `\XINT_pow_posprod` and moved here, as it was well adapted for computing powers. Then 1.03 moved the special variants of multiplication (hence of addition) which were needed to earlier in this style file.

Modified in 1.06, the exponent is given to a `\numexpr` rather than twice expanded. `\xintnum` added in 1.09a.

`\XINT_pow_posprod`: Routine de produit servant pour le calcul des puissances. Chaque nouveau terme est plus grand que ce qui a déjà été calculé. Par conséquent on a intérêt à le conserver en second dans la routine de multiplication, donc le précédent calcul a intérêt à avoir été donné sur  $4n$ , à l'envers. Il faut donc modifier la multiplication pour qu'elle fasse cela. Ce qui oblige à utiliser une version spéciale de l'addition également.

1.09j has reorganized the main loop, the described above `\XINT_pow_posprod` routine has been removed, intermediate multiplications are done immediately. Also, the maximal accepted exponent is now 100000 (no such restriction in `\xintFloatPow`, which accepts any exponent less than  $2^{31}$ , and in `\xintFloatPower` which accepts long integers as exponent).

$2^{100000}=9.990020930143845e30102$  and multiplication of two numbers with 30000 digits would take hours on my laptop (seconds for 1000 digits).

```
1293 \def\xintiiPow {\romannumeral0\xintiipow }%
1294 \def\xintiipow #1%
1295 {%
1296   \expandafter\xint_pow\romannumeral-`0#1\Z%
1297 }%
1298 \def\xintiPow {\romannumeral0\xintipow }%
1299 \def\xintipow #1%
1300 {%
1301   \expandafter\xint_pow\romannumeral0\xintnum{#1}\Z%
1302 }%
1303 \let\xintPow\xintiPow \let\xintpow\xintipow
```

### 3 Package *xintcore* implementation

```
1304 \def\xint_pow #1#2\Z
1305 {%
1306   \xint_UDsignfork
1307   #1\XINT_pow_Aneg
1308   -\XINT_pow_Anonneg
1309   \krof
1310   #1{#2}%
1311 }%
1312 \def\XINT_pow_Aneg #1#2#3%
1313 {%
1314   \expandafter\XINT_pow_Aneg_\expandafter{\the\numexpr #3}#2\Z
1315 }%
1316 \def\XINT_pow_Aneg_ #1%
1317 {%
1318   \ifodd #1
1319     \expandafter\XINT_pow_Aneg_Bodd
1320   \fi
1321   \XINT_pow_Anonneg_ {#1}%
1322 }%
1323 \def\XINT_pow_Aneg_Bodd #1%
1324 {%
1325   \expandafter\XINT_opp\romannumeral0\XINT_pow_Anonneg_
1326 }%
```

*B = #3, faire le xpxp. Modified with 1.06: use of \numexpr.*

```
1327 \def\XINT_pow_Anonneg #1#2#3%
1328 {%
1329   \expandafter\XINT_pow_Anonneg_\expandafter {\the\numexpr #3}#1#2\Z
1330 }%
```

*#1 = B, #2 = |A|. Modifié pour v1.1, car utilisait \XINT\_Cmp, ce qui d'ailleurs n'était sans doute pas super efficace, et m'obligeait à mettre \xintCmp dans xintcore. Donc ici A est déjà #2#3 et il y a un \Z après.*

```
1331 \def\XINT_pow_Anonneg_ #1#2#3\Z
1332 {%
1333   \if\relax #3\relax\xint_dothis
1334     {\ifcase #2 \expandafter\XINT_pow_AisZero
1335       \or\expandafter\XINT_pow_AisOne
1336       \else\expandafter\XINT_pow_AatleastTwo
1337     \fi }\fi
1338   \xint_orthat \XINT_pow_AatleastTwo {#1}{#2#3}%
1339 }%
1340 \def\XINT_pow_AisOne #1#2{ 1}%
```

*#1 = B*

```
1341 \def\XINT_pow_AisZero #1#2%
1342 {%
1343   \ifcase\XINT_cntSgn #1\Z
1344     \xint_afterfi { 1}%
1345   \or
1346     \xint_afterfi { 0}%
```

### 3 Package *xintcore* implementation

```

1347     \else
1348         \xint_afterfi {\xintError:DivisionByZero\space 0}%
1349     \fi
1350 }%
1351 \def\xINT_pow_AatleastTwo #1%
1352 {%
1353     \ifcase\xINT_cntSgn #1\Z
1354         \expandafter\xINT_pow_BisZero
1355     \or
1356         \expandafter\xINT_pow_checkBsize
1357     \else
1358         \expandafter\xINT_pow_BisNegative
1359     \fi
1360     {#1}%
1361 }%
1362 \edef\xINT_pow_BisNegative #1#2%
1363     {\noexpand\xintError:FractionRoundedToZero\space 0}%
1364 \def\xINT_pow_BisZero #1#2{ 1}%

1365 \def\xINT_pow_checkBsize #1%
1366 {%
1367     \ifnum #1>100000
1368         \expandafter\xINT_pow_BtooBig
1369     \else
1370         \expandafter\xINT_pow_loopI
1371     \fi
1372     {#1}%
1373 }%
1374 \edef\xINT_pow_BtooBig #1#2{\noexpand\xintError:ExponentTooBig\space 0}%
1375 \def\xINT_pow_loopI #1%
1376 {%
1377     \ifnum #1=\xint_c_i\xINT_pow_Iend\fi
1378     \ifodd #1
1379         \expandafter\xINT_pow_loopI_odd
1380     \else
1381         \expandafter\xINT_pow_loopI_even
1382     \fi
1383     {#1}%
1384 }%
1385 \edef\xINT_pow_Iend\fi #1\fi #2#3{\noexpand\fi\space #3}%
1386 \def\xINT_pow_loopI_even #1#2%
1387 {%
1388     \expandafter\xINT_pow_loopI\expandafter
1389     {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
1390     {\romannumeral0\xintiisqr {#2}}%
1391 }%
1392 \def\xINT_pow_loopI_odd #1#2%
1393 {%
1394     \expandafter\xINT_pow_loopI_odda\expandafter
1395     {\romannumeral0\xINT_RQ }#2\R\R\R\R\R\R\R\R\Z }{#1}{#2}%

```

B = #1 > 0, A = #2 > 1. With 1.05, I replace `\xintiLen{#1}>9` by direct use of `\numexpr` [to generate an error message if the exponent is too large] 1.06: `\numexpr` was already used above.

### 3 Package *xintcore* implementation

```
1396 }%
1397 \def\XINT_pow_loopI_odda #1#2#3%
1398 {%
1399   \expandafter\XINT_pow_loopII\expandafter
1400   {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
1401   {\romannumeral0\xintiisqr {#3}}{#1}%
1402 }%
1403 \def\XINT_pow_loopII #1%
1404 {%
1405   \ifnum #1 = \xint_c_i\XINT_pow_IIend\fi
1406   \ifodd #1
1407     \expandafter\XINT_pow_loopII_odd
1408   \else
1409     \expandafter\XINT_pow_loopII_even
1410   \fi
1411   {#1}%
1412 }%
1413 \def\XINT_pow_loopII_even #1#2%
1414 {%
1415   \expandafter\XINT_pow_loopII\expandafter
1416   {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
1417   {\romannumeral0\xintiisqr {#2}}%
1418 }%
1419 \def\XINT_pow_loopII_odd #1#2#3%
1420 {%
1421   \expandafter\XINT_pow_loopII_odda\expandafter
1422   {\romannumeral0\XINT_mulr_enter #3\Z\Z\Z #2\W\W\W\W}{#1}{#2}%
1423 }%
1424 \def\XINT_pow_loopII_odda #1#2#3%
1425 {%
1426   \expandafter\XINT_pow_loopII\expandafter
1427   {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
1428   {\romannumeral0\xintiisqr {#3}}{#1}%
1429 }%
1430 \def\XINT_pow_IIend\fi #1\fi #2#3#4%
1431 {%
1432   \fi\XINT_mul_enter #4\Z\Z\Z\Z #3\W\W\W\W
1433 }%
```

#### 3.16 `\xintiDivision`, `\xintiQuo`, `\xintiRem`, `\xintiiDivision`, `\xintiiQuo`, `\xintiiRem`

The 1.09a release inserted the use of `\xintnum`. The `\xintiiDivision` etc... are the ones which do only `\romannumeral-`0`.

January 5, 2014: Naturally, addition, subtraction, multiplication and division are the first things I did and since then I had left the division untouched. So in preparation of release 1.09j, I started revisiting the division, I did various minor improvements obtaining roughly 10% efficiency gain. Then I decided I should deliberately impact the input save stack, with the hope to gain more speed from removing tokens and leaving them upstream.

For this however I had to modify the underlying mathematical algorithm. The initial one is a bit unusual I guess, and, I trust, rather efficient, but it does not produce the quotient digits (in base 10000) one by one; at any given time it is possible that some correction will be made,



### 3 Package *xintcore* implementation

which means it is not an appropriate algorithm for a TeX implementation which will abandon the quotient upstream. Thus I now have with 1.09j a new underlying mathematical algorithm, presumably much more standard. It is a bit complicated to implement expandably these things, but in the end I had regained the already mentioned 10% efficiency and even more for small to medium sized inputs (up to 30% perhaps). And in passing I did a special routine for divisors < 10000, which is 5 to 10 times faster still.

But, I then tested a variant of my new implementation which again did not impact the input save stack and, for sizes of up to 200 digits, it is not much worse, indeed it is perhaps actually better than the one abandoning the quotient digits upstream (and in the end putting them in the correct order). So, finally, I re-incorporated the produced quotient digits within a tail recursion. Hence `\xintiDivision`, like all other routines in `xint` (except `\xintSeq` without optional parameter) does not impact the input save stack. One can have a produced quotient longer than  $4 \times 5000 = 20000$  digits, and no need to worry about consequences propagating to `\xintTrunc`, `\xintRound`, `\xintFloat`, `\xintFloatSqrt`, etc... and all other places using the division. See also `\xintXTrunc` in this context.

```

1434 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1435 \def\xintiiRem {\romannumeral0\xintiirem }%
1436 \def\xintiiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1437 \def\xintiirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
1438 \def\xintiQuo {\romannumeral0\xintiquo }%
1439 \def\xintiRem {\romannumeral0\xintirem }%
1440 \def\xintiQuo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision }%
1441 \def\xintirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision }%
1442 \let\xintQuo\xintiQuo\let\xintquo\xintiQuo % deprecated (1.1)
1443 \let\xintRem\xintiRem\let\xintrem\xintirem % deprecated (1.1)

```

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

```

1444 \def\xintiDivision {\romannumeral0\xintidivision }%
1445 \def\xintidivision #1{\expandafter\XINT_division\romannumeral0\xintnum{#1}\Z }%
1446 \let\xintDivision\xintiDivision \let\xintdivision\xintidivision % deprecated
1447 \def\XINT_division #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1448 \romannumeral0\xintnum{#3}\Z #2\Z }%
1449 \def\xintiDivision {\romannumeral0\xintiidivision }%
1450 \def\xintiidivision #1{\expandafter\XINT_iidivision \romannumeral-`0#1\Z }%
1451 \def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1452 \romannumeral-`0#3\Z #2\Z }%
1453 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1454 {%
1455 \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
1456 \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1457 \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1458 \romannumeral0\XINT_iidivision_bpos #1}\fi
1459 \xint_orthat{\XINT_iidivision_bpos #1#2}%
1460 }%
1461 \def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space {0}{0}}%
1462 \def\XINT_iidivision_aiszero #1\Z #2\Z { {0}{0}}%
1463 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1464 {\expandafter\space\expandafter{\romannumeral0\XINT_opp #1}}%
1465 \def\XINT_iidivision_bpos #1%
1466 {%

```

### 3 Package *xintcore* implementation

```

1467 \xint_UDsignfork
1468     #1\XINT_iidivision_aneg
1469     -{\XINT_iidivision_apos #1}%
1470 \krof
1471 }%
1472 \def\XINT_iidivision_apos #1#2\Z #3\Z{\XINT_div_prepare {#2}{#1#3}}%
1473 \def\XINT_iidivision_aneg #1\Z #2\Z
1474     {\expandafter
1475     \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1476 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\Z
1477     \expandafter\XINT_iidivision_aneg_rzero
1478     \else
1479     \expandafter\XINT_iidivision_aneg_rpos
1480     \fi {#1}{#2}}%
1481 \def\XINT_iidivision_aneg_rzero #1#2#3{ {-#1}{0}}% necessarily q was >0
1482 \def\XINT_iidivision_aneg_rpos #1%
1483 {%
1484     \expandafter\XINT_iidivision_aneg_end\expandafter
1485     {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1486 }%
1487 \def\XINT_iidivision_aneg_end #1#2#3%
1488 {%
1489     \expandafter\xint_exchangetwo_keepbraces_thenstop
1490     \expandafter{\romannumeral0\XINT_sub_pre {#3}{#2}}{#1}% r-> b-r
1491 }%

```

Pour la suite A et B sont  $> 0$ . #1 = B. Pour le moment à l'endroit. Calcul du plus petit  $K = 4n \geq$  longueur de B

```

1492 \def\XINT_div_prepare #1%
1493 {%
1494     \expandafter \XINT_div_prepareB_aa \expandafter
1495     {\romannumeral0\xintlength {#1}}{#1}% B > 0 ici
1496 }%
1497 \def\XINT_div_prepareB_aa #1%
1498 {%
1499     \ifnum #1=\xint_c_i
1500     \expandafter\XINT_div_prepareB_onedigit
1501     \else
1502     \expandafter\XINT_div_prepareB_a
1503     \fi
1504     {#1}%
1505 }%
1506 \def\XINT_div_prepareB_a #1%
1507 {%
1508     \expandafter\XINT_div_prepareB_c\expandafter
1509     {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
1510 }%

```

B=1 and B=2 treated specially.

```

1511 \def\XINT_div_prepareB_onedigit #1#2%
1512 {%
1513     \ifcase#2

```

### 3 Package *xintcore* implementation

```

1514 \or\expandafter\XINT_div_BisOne
1515 \or\expandafter\XINT_div_BisTwo
1516 \else\expandafter\XINT_div_prepareB_e
1517 \fi {000}{0}{4}{#2}%
1518 }%
1519 \def\XINT_div_BisOne #1#2#3#4#5{ {#5}{0}}%
1520 \def\XINT_div_BisTwo #1#2#3#4#5%
1521 {%
1522 \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1523 \ifodd\xintiiLDg{#5} \expandafter1\else \expandafter0\fi {#5}%
1524 }%
1525 \edef\XINT_div_BisTwo_a #1#2%
1526 {%
1527 \noexpand\expandafter\space\noexpand\expandafter
1528 {\noexpand\romannumeral0\noexpand\xinthalft {#2}}{#1}%
1529 }%

```

*#1 = K. 1.09j uses \csname, earlier versions did it with \ifcase.*

```

1530 \def\XINT_div_prepareB_c #1#2%
1531 {%
1532 \csname XINT_div_prepareB_d\romannumeral\numexpr#1-#2\endcsname
1533 {#1}%
1534 }%
1535 \def\XINT_div_prepareB_d {\XINT_div_prepareB_e }{0000}%
1536 \def\XINT_div_prepareB_di {\XINT_div_prepareB_e {0}{000}}%
1537 \def\XINT_div_prepareB_dii {\XINT_div_prepareB_e {00}{00}}%
1538 \def\XINT_div_prepareB_diii {\XINT_div_prepareB_e {000}{0}}%
1539 \def\XINT_div_cleanR #10000.{#1}%

```

*#1 = zéros à rajouter à B, #2=c [modifié dans 1.09j, ce sont maintenant des zéros explicites en nombre 4 - ancien c, et on utilisera \XINT\_div\_cleanR et non plus \XINT\_dsh\_checksignx pour nettoyer à la fin des zéros en excès dans le Reste; in all comments next, «c» stands now {0} or {00} or {000} or {0000} rather than a digit as in earlier versions], #3=K, #4 = B*

```

1540 \def\XINT_div_prepareB_e #1#2#3#4%
1541 {%
1542 \ifnum#3=\xint_c_iv\expandafter\XINT_div_prepareLittleB_f
1543 \else\expandafter\XINT_div_prepareB_f
1544 \fi
1545 #4#1{#3}{#2}{#1}%
1546 }%

```

*x = #1#2#3#4 = 4 premiers chiffres de B. #1 est non nul. B is reversed. With 1.09j or latter x+1 and (x+1)/2 are pre-computed. Si K=4 on ne renverse pas B, et donc B=x dans la suite. De plus pour K=4 on ne travaille pas avec x+1 et (x+1)/2 mais avec x et x/2.*

```

1547 \def\XINT_div_prepareB_f #1#2#3#4#5{%
1548 \expandafter\XINT_div_prepareB_g
1549 \the\numexpr #1#2#3#4+\xint_c_i\expandafter
1550 \the\numexpr (#1#2#3#4+\xint_c_i)/\xint_c_ii\expandafter
1551 \romannumeral0\xintreverseorder {#1#2#3#4#5}.{#1#2#3#4}%
1552 }%
1553 \def\XINT_div_prepareLittleB_f #1{%

```

### 3 Package *xintcore* implementation

```

1554 \expandafter\XINT_div_prepareB_g \the\numexpr #1/\xint_c_ii.{}.{}.{}#1%
1555 }%

#1 = x' = x+1= 1+quatre premiers chiffres de B, #2 = y = (x+1)/2 précalculé #3 = B préparé et
maintenant renversé, #4=x, #5 = K, #6 = «c», #7= {} ou {0} ou {00} ou {000}, #8 = A initial On
multiplie aussi A par 10^c. -> AK{x'yx}B«c». Par contre dans le cas little on a #1=y=(x/2), #2={},
#3={}, #4=x, donc cela donne ->AK{y{x}}{}«c», il n'y a pas de B.

1556 \def\XINT_div_prepareB_g #1.#2.#3.#4#5#6#7#8%
1557 {%
1558 \XINT_div_prepareA_a {#8#7}{#5}{{#1}{#2}{#4}}{#3}{#6}%
1559 }%

A, K, {x'yx}, B«c»

1560 \def\XINT_div_prepareA_a #1%
1561 {%
1562 \expandafter\XINT_div_prepareA_b\expandafter
1563 {\romannumeral0\xintlength {#1}}{#1}%
1564 }%

L0, A, K, {x'yx}, B«c»

1565 \def\XINT_div_prepareA_b #1%
1566 {%
1567 \expandafter\XINT_div_prepareA_c\expandafter
1568 {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
1569 }%

L, L0, A, K, {x'yx}, B, «c»

1570 \def\XINT_div_prepareA_c #1#2%
1571 {%
1572 \csname XINT_div_prepareA_d\romannumeral\numexpr #1-#2\endcsname
1573 {#1}%
1574 }%
1575 \def\XINT_div_prepareA_d {\XINT_div_prepareA_e {}}%
1576 \def\XINT_div_prepareA_di {\XINT_div_prepareA_e {0}}%
1577 \def\XINT_div_prepareA_dii {\XINT_div_prepareA_e {00}}%
1578 \def\XINT_div_prepareA_diii {\XINT_div_prepareA_e {000}}%

#1#3 = A préparé, #2 = longueur de ce A préparé, #4=K, #5={x'yx}-> LKAx'yxB«c»

1579 \def\XINT_div_prepareA_e #1#2#3#4#5%
1580 {%
1581 \XINT_div_start_a {#2}{#4}{#1#3}#5%
1582 }%

L, K, A, x',y,x, B, «c» (avec y{x} au lieu de x'yxB dans la variante little)

1583 \def\XINT_div_start_a #1#2%
1584 {%
1585 \ifnum #2=\xint_c_iv \expandafter\XINT_div_little_b
1586 \else
1587 \ifnum #1 < #2
1588 \expandafter\expandafter\expandafter\XINT_div_III_aa

```

### 3 Package *xintcore* implementation

```

1589     \else
1590     \expandafter\expandafter\expandafter\XINT_div_start_b
1591     \fi
1592     \fi
1593     {#1}{#2}%
1594 }%

```

*L, K, A, x', y, x, B, <<C>>.*

```

1595 \def\XINT_div_III_aa #1#2#3#4#5#6#7%
1596 {%
1597     \expandafter\expandafter\expandafter
1598     \XINT_div_III_b\xint_cleanupzeros_nostop #3.{0000}%
1599 }%

```

*R.Q<<C>>.*

```

1600 \def\XINT_div_III_b #1%
1601 {%
1602     \if0#1%
1603     \expandafter\XINT_div_III_bRzero
1604     \else
1605     \expandafter\XINT_div_III_bRpos
1606     \fi
1607     #1%
1608 }%
1609 \def\XINT_div_III_bRzero 0.#1#2%
1610 {%
1611     \expandafter\space\expandafter
1612     {\romannumeral0\XINT_cuz_loop #1\W\W\W\W\W\W\W\Z}{0}%
1613 }%
1614 \def\XINT_div_III_bRpos #1.#2#3%
1615 {%
1616     \expandafter\XINT_div_III_c \XINT_div_cleanR #1#3.{#2}%
1617 }%
1618 \def\XINT_div_III_c #1#2%
1619 {%
1620     \expandafter\space\expandafter
1621     {\romannumeral0\XINT_cuz_loop #2\W\W\W\W\W\W\W\Z}{#1}%
1622 }%

```

*L, K, A, x', y, x, B, <<C>>->K.A.x{LK{x'y}x}B<<C>>*

```

1623 \def\XINT_div_start_b #1#2#3#4#5#6%
1624 {%
1625     \XINT_div_start_c {#2}.#3.{#6}{#1}{#2}{#4}{#5}{#6}%
1626 }%

```

*Kalpha.A.x{LK{x'y}x}, B, <<C>>, au début #2=alpha est vide*

```

1627 \def\XINT_div_start_c #1#2.#3#4#5#6%
1628 {%
1629     \ifnum #1=\xint_c_iv\XINT_div_start_ca\fi
1630     \expandafter\XINT_div_start_c\expandafter
1631     {\the\numexpr #1-\xint_c_iv}#2#3#4#5#6.%

```

### 3 Package *xintcore* implementation

```

1632 }%
1633 \def\XINT_div_start_ca\fi\expandafter\XINT_div_start_c\expandafter
1634   #1#2#3#4#5{\fi\XINT_div_start_d {#2#3#4#5}#2#3#4#5}%

#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, {0000}, L, K, {x'y},x, alpha'=reste de A, B{«c»}. Pour K=4 on a en fait B=x, faudra revoir
après.

1635 \def\XINT_div_start_d #1#2.#3.#4#5#6%
1636 {%
1637   \XINT_div_I_a {#1}{#4}{#2}{#6}{0000}#5{#3}{#6}{}}%
1638 }%

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

1639 \def\XINT_div_I_a #1#2%
1640 {%
1641   \expandafter\XINT_div_I_b\the\numexpr #1/#2.{#1}{#2}%
1642 }%
1643 \def\XINT_div_I_b #1%
1644 {%
1645   \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1646 }%

On intercepte quotient nul: #1=a, x, alpha, B, #5=q0, L, K, {x'y}, x, alpha', BQ«c» -> q{alpha} L,
K, {x'y}, x, alpha', BQ«c»

1647 \def\XINT_div_I_czero 0%
1648   \XINT_div_I_c 0.#1#2#3#4#5{\XINT_div_I_g {#5}{#3}}%
1649 \def\XINT_div_I_c #1.#2#3%
1650 {%
1651   \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3.#1.%
1652 }%

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

1653 \def\XINT_div_I_da #1.%
1654 {%
1655   \ifnum #1>\xint_c_ix
1656     \expandafter\XINT_div_I_dp
1657   \else
1658     \ifnum #1<\xint_c_
1659       \expandafter\expandafter\expandafter\XINT_div_I_dN
1660     \else
1661       \expandafter\expandafter\expandafter\XINT_div_I_db
1662     \fi
1663   \fi
1664 }%
1665 \def\XINT_div_I_dN #1.%
1666 {%
1667   \expandafter\XINT_div_I_dp\the\numexpr #1-\xint_c.i.%
1668 }%
1669 \def\XINT_div_I_db #1.#2#3% #1=q=un chiffre, #2=alpha, #3=B
1670 {%

```

### 3 Package *xintcore* implementation

```

1671 \expandafter\XINT_div_I_dc\expandafter
1672 {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1673   {\romannumeral0\xintreverseorder{#2}}%
1674   {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z }}%
1675   #1{#2}{#3}%
1676 }%
1677 \def\XINT_div_I_dc #1#2%
1678 {%
1679   \if-#1% s'arranger pour que si n\'egatif on ait renvoy\'e alpha=-.
1680     \expandafter\xint_firstoftwo
1681   \else\expandafter\xint_secondoftwo\fi
1682   {\expandafter\XINT_div_I_dP\the\numexpr #2-\xint_c_i.}%
1683   {\XINT_div_I_e {#1}#2}%
1684 }%

```

*alpha,q,ancien alpha,B, q0->1nouveauq.alpha, L, K, {x'y},x, alpha', BQ<<c>*

```

1685 \def\XINT_div_I_e #1#2#3#4#5%
1686 {%
1687   \expandafter\XINT_div_I_f \the\numexpr \xint_c_x^iv+#2+#5{#1}%
1688 }%

```

*q.alpha, B, q0, L, K, {x'y},x, alpha'BQ<<c> (intercepter q=0?) -> 1nouveauq.nouvel alpha, L, K, {x'y}, x, alpha',BQ<<c>*

```

1689 \def\XINT_div_I_dP #1.#2#3#4%
1690 {%
1691   \expandafter \XINT_div_I_f \the\numexpr \xint_c_x^iv+#1+#4\expandafter
1692   {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1693     {\romannumeral0\xintreverseorder{#2}}%
1694     {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z }}%
1695 }%

```

*1#1#2#3#4=nouveau q, nouvel alpha, L, K, {x'y},x,alpha', BQ<<c>*

```

1696 \def\XINT_div_I_f 1#1#2#3#4{\XINT_div_I_g {#1#2#3#4}}%

```

*#1=q,#2=nouvel alpha,#3=L, #4=K, #5={x'y}, #6=x, #7= alpha',#8=B, #9=Q<<c> -> {x'y}alpha.alpha'.{{x'y}xKL}B{Qq}*

```

1697 \def\XINT_div_I_g #1#2#3#4#5#6#7#8#9%
1698 {%
1699   \ifnum#3=#4
1700     \expandafter\XINT_div_III_ab
1701   \else
1702     \expandafter\XINT_div_I_h
1703   \fi
1704   {#5}#2.#7.{#5}{#6}{#4}{#3}}{#8}{#9#1}%
1705 }%

```

*{x'y}alpha.alpha'.{{x'y}xKL}B{Qq}<<c> -> R sans leading zeros.{Qq}<<c>*

```

1706 \def\XINT_div_III_ab #1#2.#3.#4#5%
1707 {%
1708   \expandafter\XINT_div_III_b
1709   \romannumeral0\XINT_cuz_loop #2#3\W\W\W\W\W\W\W\Z.%
1710 }%

```

### 3 Package *xintcore* implementation

$\#1=\{x'y\}$ alpha. $\#2\#3\#4\#5\#6$ =reste de A.  $\#7=\{\{x'y\},x,K,L\}$ , $\#8=B$ ,nouveauQ«c» devient  $\{x'y\}$ ,alpha sur  $K+4$  chiffres.B,  $\{\{x'y\},x,K,L\}$ ,  $\#6$ = nouvel alpha',B,nouveauQ«c»

```
1711 \def\XINT_div_I_h #1.#2#3#4#5#6.#7#8%
1712 {%
1713   \XINT_div_II_b #1#2#3#4#5.{#8}{#7}{#6}{#8}%
1714 }%
```

$\{x'y\}$ alpha.B,  $\{\{x'y\},x,K,L\}$ , nouveau alpha',B, Q«c» On intercepte la situation avec alpha débutant par 0000 qui est la seule qui pourrait donner un q1 nul. Donc q1 est non nul et la soustraction spéciale recevra un  $q1*B$  de longueur K ou  $K+4$  et jamais 0000. Ensuite un q2 éventuel s'il est calculé est nécessairement non nul lui aussi. Comme dans la phase I on a aussi intercepté un q nul, la soustraction spéciale ne reçoit donc jamais un qB nul. Note: j'ai testé plusieurs fois que ma technique de gob\_til\_zeros est plus rapide que d'utiliser un \ifnum

```
1715 \def\XINT_div_II_b #1#2#3#4#5#6#7#8#9%
1716 {%
1717   \xint_gob_til_zeros_iv #2#3#4#5\XINT_div_II_skipc 0000%
1718   \XINT_div_II_c #1{#2#3#4#5}{#6#7#8#9}%
1719 }%
```

$x'y\{0000\}$ {4chiffres}reste de alpha. $\#6=B$ , $\#7=\{\{x'y\},x,K,L\}$ , alpha',B, Q«c» ->  $\{x'y\}x,K,L$  (à diminuer de 4), {alpha sur K}B{q1=0000}{alpha'}B,Q«c»

```
1720 \def\XINT_div_II_skipc 0000\XINT_div_II_c #1#2#3#4#5.#6#7%
1721 {%
1722   \XINT_div_II_k #7{#4#5}{#6}{0000}%
1723 }%
```

$x'ya->lqx'y$ alpha.B,  $\{\{x'y\},x,K,L\}$ , nouveau alpha',B, Q«c»

```
1724 \def\XINT_div_II_c #1#2#3#4%
1725 {%
1726   \expandafter\XINT_div_II_d\the\numexpr (#3#4+#2)/#1+\xint_c_ixixixix\relax
1727   {#1}{#2}#3#4%
1728 }%
```

1 suivi de q1 sur quatre chiffres,  $\#5=x'$ ,  $\#6=y$ ,  $\#7=alpha$ . $\#8=B$ ,  $\{\{x'y\},x,K,L\}$ , alpha', B, Q«c» --> nouvel alpha.x',y,B,q1, $\{\{x'y\},x,K,L\}$ , alpha', B, Q«c»

```
1729 \def\XINT_div_II_d 1#1#2#3#4#5#6#7.#8%
1730 {%
1731   \expandafter\XINT_div_II_e
1732   \romannumeral0\expandafter\XINT_div_sub_xp xp\expandafter
1733   {\romannumeral0\xintreverseorder{#7}}%
1734   {\romannumeral0\XINT_mul_Mr {#1#2#3#4}#8\Z\Z\Z }.%
1735   {#5}{#6}{#8}{#1#2#3#4}%
1736 }%
```

alpha.x',y,B,q1,  $\{\{x'y\},x,K,L\}$ , alpha', B, Q«c»

```
1737 \def\XINT_div_II_e #1#2#3#4%
1738 {%
1739   \xint_gob_til_zeros_iv #1#2#3#4\XINT_div_II_skipf 0000%
1740   \XINT_div_II_f #1#2#3#4%
1741 }%
```



### 3 Package *xintcore* implementation

0000alpha sur K chiffres. #2=x', #3=y, #4=B, #5=q1, #6={{x'y},x,K,L}, #7=alpha', BQ<< -> {x'y}x,K,L  
(à diminuer de 4), {alpha sur K}B{q1}{alpha'}BQ<<

```
1742 \def\XINT_div_II_skipf 0000\XINT_div_II_f 0000#1.#2#3#4#5#6%
1743 {%
1744   \XINT_div_II_k #6{#1}{#4}{#5}%
1745 }%
```

a1 (huit chiffres), alpha (sur K+4), x', y, B, q1, {{x'y},x,K,L}, alpha', B,Q<<

```
1746 \def\XINT_div_II_f #1#2#3#4#5#6#7#8#9.%
1747 {%
1748   \XINT_div_II_fa {#1#2#3#4#5#6#7#8}{#1#2#3#4#5#6#7#8#9}%
1749 }%
1750 \def\XINT_div_II_fa #1#2#3#4%
1751 {%
1752   \expandafter\XINT_div_II_g\expandafter
1753     {\the\numexpr (#1+#4)/#3-\xint_c_i}{#2}%
1754 }%
```

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

```
1755 \def\XINT_div_II_g #1#2#3#4%
1756 {%
1757   \expandafter \XINT_div_II_h
1758   \the\numexpr #4+#1+\xint_c_x^iv\expandafter\expandafter\expandafter
1759   {\expandafter\xint_gobble_iv
1760   \romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1761     {\romannumeral0\xintreverseorder{#2}}}%
1762   {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z }{#3}%
1763 }%
```

1 puis nouveau q sur 4 chiffres, #5=nouvel alpha sur K chiffres, #6=B, #7={{x'y},x,K,L} avec L à  
ajuster, alpha', BQ<< -> {x'y}x,K,L à diminuer de 4, {alpha}B{q}, alpha', BQ<<

```
1764 \def\XINT_div_II_h #1#2#3#4#5#6#7%
1765 {%
1766   \XINT_div_II_k #7{#5}{#6}{#1#2#3#4}%
1767 }%
```

{x'y}x,K,L à diminuer de 4, alpha, B{q}alpha', BQ<< -> nouveau L.K, x', y, x, alpha.B, q, alpha', B, Q<<  
->{LK{x'y}x}, x, a, alpha.B, q, alpha', B, Q<<

```
1768 \def\XINT_div_II_k #1#2#3#4#5%
1769 {%
1770   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_iv.{#3}#1{#2}#5.%
1771 }%
1772 \def\XINT_div_II_l #1.#2#3#4#5#6#7#8#9%
1773 {%
1774   \XINT_div_II_m {{#1}{#2}{{#3}{#4}}{#5}}{#5}{#6#7#8#9}#6#7#8#9%
1775 }%
```

### 3 Package *xintcore* implementation

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

```
1776 \def\XINT_div_II_m #1#2#3#4.#5#6%
1777 {%
1778   \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1779 }%
```

L, K, A, y, {}, x, {}, «c»->A.{yx}L{}«c» Comme ici K=4, dans la phase I on n'a pas besoin de alpha, car a = alpha. De plus on a maintenu B dans l'ordre qui est donc la même chose que x. Par ailleurs la phase I est simplifiée, il s'agit simplement de la division euclidienne de a par x, et de plus on n'a à la faire qu'une unique fois et ensuite la phase II peut boucler sur elle-même au lieu de revenir en phase I, par conséquent il n'y a pas non plus de q0 ici. Enfin, le y est (x/2) pas ((x+1)/2) il n'y a pas de x'=x+1

```
1780 \def\XINT_div_little_b #1#2#3#4#5#6#7%
1781 {%
1782   \XINT_div_little_c #3.{{#4}{#6}}{#1}%
1783 }%
```

#1#2#3#4=a, #5=alpha'=reste de A.#6={yx}, #7=L, «c» -> a, y, x, L, alpha'=reste de A, «c».

```
1784 \def\XINT_div_little_c #1#2#3#4#5.#6#7%
1785 {%
1786   \XINT_div_littleI_a {#1#2#3#4}#6{#7}{#5}%
1787 }%
```

a, y, x, L, alpha', «c» On calcule ici (contrairement à la phase I générale) le vrai quotient euclidien de a par x=B, c'est donc un chiffre de 0 à 9. De plus on n'a à faire cela qu'une unique fois.

```
1788 \def\XINT_div_littleI_a #1#2#3%
1789 {%
1790   \expandafter\XINT_div_littleI_b
1791   \the\numexpr (#1+#2)/#3-\xint_c_i{#1}{#2}{#3}%
1792 }%
```

On intercepte quotient nul: [est-ce vraiment utile? ou n'est-ce pas plutôt une perte de temps en moyenne? il faudrait tester] q=0#1=a, #2=y, x, L, alpha', «c» -> II\_a avec L{alpha}alpha'.{yx}{0000}«c». Et en cas de quotient non nul on procède avec littleI\_c avec #1=q, #2=a, #3=y, #4=x -> {nouvel alpha sur 4 chiffres}q{yx},L,alpha',«c».

```
1793 \def\XINT_div_littleI_b #1%
1794 {%
1795   \xint_gob_til_zero #1\XINT_div_littleI_skip 0\XINT_div_littleI_c #1%
1796 }%
1797 \def\XINT_div_littleI_skip 0\XINT_div_littleI_c 0#1#2#3#4#5%
1798   {\XINT_div_littleII_a {#4}{#1}#5.{{#2}{#3}}{0000}}%
1799 \def\XINT_div_littleI_c #1#2#3#4%
1800 {%
1801   \expandafter\expandafter\expandafter\XINT_div_littleI_e
1802   \expandafter\expandafter\expandafter
1803   {\expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4}#1{{#3}{#4}}%
1804 }%
```

### 3 Package *xintcore* implementation

*#1=nouvel alpha sur 4 chiffres#2=q,#3={yx}, #4=L, #5=alpha', <<c> -> L{alpha}alpha'.{yx}{000q}<<c>*  
*point d'entrée de la boucle principale*

```
1805 \def\XINT_div_littleI_e #1#2#3#4#5%
1806   {\XINT_div_littleII_a {#4}{#1}#5.{#3}{000#2}}%
```

*L{alpha}alpha'.{yx}Q<<c> et c'est là qu'on boucle*

```
1807 \def\XINT_div_littleII_a #1%
1808 {%
1809   \ifnum#1=\xint_c_iv
1810     \expandafter\XINT_div_littleIII_ab
1811   \else
1812     \expandafter\XINT_div_littleII_b
1813   \fi {#1}%
1814 }%
```

*L{alpha}alpha'.{yx}Q<<c> -> (en fait #3 est vide normalement ici) R sans leading zeros.Q<<c>*

```
1815 \def\XINT_div_littleIII_ab #1#2#3.#4%
1816 {%
1817   \expandafter\XINT_div_III_b\the\numexpr #2#3.%
1818 }%
```

*L{alpha}alpha'.{yx}Q<<c>. On diminue L de quatre, comme cela c'est fait.*

```
1819 \def\XINT_div_littleII_b #1%
1820 {%
1821   \expandafter\XINT_div_littleII_c\expandafter {\the\numexpr #1-\xint_c_iv}%
1822 }%
```

*{nouveauL}{alpha}alpha'.{yx}Q<<c>. On prélève 4 chiffres de alpha' -> {nouvel alpha sur huit chiffres}yx{nouveau L}{nouvel alpha'}Q<<c>. Regarder si l'ancien alpha était 0000 n'avancerait à rien car obligerait à refaire une chose comme la phase I, donc on ne perd pas de temps avec ça, on reste en permanence en phase II.*

```
1823 \def\XINT_div_littleII_c #1#2#3#4#5#6#7.#8%
1824 {%
1825   \XINT_div_littleII_d {#2#3#4#5#6}#8{#1}{#7}%
1826 }%
1827 \def\XINT_div_littleII_d #1#2#3%
1828 {%
1829   \expandafter\XINT_div_littleII_e\the\numexpr (#1+#2)/#3+\xint_c_ixixixix.%
1830   {#1}{#2}{#3}%
1831 }%
```

*1 suivi de #1=q1 sur quatre chiffres.#2=alpha, #3=y, #4=x, L, alpha', Q<<c> --> nouvel alpha sur 4.{q1}{yx},L,alpha', Q<<c>*

```
1832 \def\XINT_div_littleII_e 1#1.#2#3#4%
1833 {%
1834   \expandafter\expandafter\expandafter\XINT_div_littleII_f
1835   \expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4.%
1836   {#1}{#3}{#4}}%
1837 }%
```

### 3 Package *xintcore* implementation

$\alpha.q, \{yx\}, L, \alpha', Q \llcorner \rightarrow L\{\alpha\}\alpha'. \{yx\}\{Qq\} \llcorner$

```
1838 \def\XINT_div_littleII_f #1.#2#3#4#5#6%
1839 {%
1840   \XINT_div_littleII_a {#4}{#1}#5.{#3}{#6#2}%
1841 }%
```

La soustraction spéciale. Dans 1.09j, elle fait  $A - qB$ , pour  $A$  (en fait  $\alpha$  dans mes dénominations des commentaires du code) et  $qB$  chacun de longueur  $K$  ou  $K+4$ , avec  $K$  au moins huit multiple de quatre,  $qB$  a ses quatre chiffres significatifs (qui sont à droite) non nuls. Si  $A - qB < 0$  il suffit de renvoyer  $-$ , le résultat n'importe pas. On est sûr que  $qB$  est non nul. On le met dans cette version en premier pour tester plus facilement le cas avec  $qB$  de longueur  $K+4$  et  $A$  de longueur seulement  $K$ . Lorsque la longueur de  $qB$  est inférieure ou égale à celle de  $A$ , on va jusqu'à la fin de  $A$  et donc c'est la retenue finale qui décide du cas négatif éventuel. Le résultat non négatif est toujours donc renvoyé avec la même longueur que  $A$ , et il est dans l'ordre. J'ai fait une implémentation des phases I et II en maintenant  $\alpha$  toujours à l'envers afin d'éviter le reverse order systématique fait sur  $A$  (ou plutôt  $\alpha$ ), mais alors il fallait que la soustraction ici s'arrange pour repérer les huit chiffres les plus significatifs, au final ce n'était pas plus rapide, et même pénalisant pour de gros inputs. Dans les versions 1.09i et antérieures (en fait je pense qu'ici rien quasiment n'avait bougé depuis la première implémentation), la soustraction spéciale n'était pratiquée que dans des cas avec certainement  $A - qB$  positif ou nul. De plus on n'excluait pas  $q=0$ , donc il fallait aussi faire un éventuel reverseorder sur ce qui était encore non traité. Les cas avec  $q=0$  sont maintenant interceptés en amont et comme  $A$  et  $qB$  ont toujours quasiment la même longueur on ne s'embarrasse pas de complications pour la fin.

```
1842 \def\XINT_div_sub_xpxp #1#2% #1=alpha d\'ej\'a renvers\'e, #2 se d\'eveloppe en qB
1843 {%
1844   \expandafter\XINT_div_sub_xpxp_b #2\W\X\Y\Z #1\W\X\Y\Z
1845 }%
1846 \def\XINT_div_sub_xpxp_b
1847 {%
1848   \XINT_div_sub_A 1{%
1849 }%
1850 \def\XINT_div_sub_A #1#2#3#4#5#6%
1851 {%
1852   \xint_gob_til_W #3\xint_div_sub_az\W
1853   \XINT_div_sub_B #1{#3#4#5#6}{#2}%
1854 }%
1855 \def\XINT_div_sub_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
1856 {%
1857   \xint_gob_til_W #5\xint_div_sub_bz\W
1858   \XINT_div_sub_onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
1859 }%
1860 \def\XINT_div_sub_onestep #1#2#3#4#5#6%
1861 {%
1862   \expandafter\XINT_div_sub_backtoA
1863   \the\numexpr 11#6-#5#4#3#2+#1-\xint_c_i.%
1864 }%
1865 \def\XINT_div_sub_backtoA #1#2#3.#4%
1866 {%
1867   \XINT_div_sub_A #2{#3#4}%
1868 }%
```

### 3 Package *xintcore* implementation

si on arrive en sub\_bz c'est que qB était de longueur K+4 et A seulement de longueur K, le résultat est donc < 0, renvoyer juste -

```
1869 \def\xint_div_sub_bz\W\XINT_div_sub_onestep #1\Z { -}%
```

si on arrive en sub\_az c'est que qB était de longueur inférieure ou égale à celle de A, donc on continue jusqu'à la fin de A, et on vérifiera la retenue à la fin.

```
1870 \def\xint_div_sub_az\W\XINT_div_sub_B #1#2{\XINT_div_sub_C #1}%
```

```
1871 \def\XINT_div_sub_C #1#2#3#4#5#6%
```

```
1872 {%
```

```
1873   \xint_gob_til_W #3\xint_div_sub_cz\W
```

```
1874   \XINT_div_sub_C_onestep #1{#6#5#4#3}{#2}%
```

```
1875 }%
```

```
1876 \def\XINT_div_sub_C_onestep #1#2%
```

```
1877 {%
```

```
1878   \expandafter\XINT_div_sub_backtoC \the\numexpr 11#2+#1-\xint_c_i.%
```

```
1879 }%
```

```
1880 \def\XINT_div_sub_backtoC #1#2#3.#4%
```

```
1881 {%
```

```
1882   \XINT_div_sub_C #2{#3#4}%
```

```
1883 }%
```

une fois arrivé en sub\_cz on teste la retenue pour voir si le résultat final est en fait négatif, dans ce cas on renvoie seulement -

```
1884 \def\xint_div_sub_cz\W\XINT_div_sub_C_onestep #1#2%
```

```
1885 {%
```

```
1886   \if#10% retenue
```

```
1887     \expandafter\xint_div_sub_neg
```

```
1888   \else\expandafter\xint_div_sub_ok
```

```
1889   \fi
```

```
1890 }%
```

```
1891 \def\xint_div_sub_neg #1{ -}%
```

```
1892 \def\xint_div_sub_ok #1{ #1}%
```

#### 3.17 `\xintiDivRound`, `\xintiiDivRound`

v1.1, transferred from first release of `bnumexpr`.

```
1893 \def\xintiDivRound {\romannumeral0\xintidivround }%
```

```
1894 \def\xintidivround #1{\expandafter\XINT_iidivround\romannumeral0\xintnum{#1}\Z }%
```

```
1895 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
```

```
1896 \def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral-`0#1\Z }%
```

```
1897 \def\XINT_iidivround #1#2\Z #3{\expandafter\XINT_iidivround_a\expandafter #1%
```

```
1898   \romannumeral-`0#3\Z #2\Z }%
```

```
1899 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
```

```
1900 {%
```

```
1901   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
```

```
1902   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
```

```
1903   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
```

```
1904     \xint_orthat{\XINT_iidivround_bpos #1#2}%
```

```
1905 }%
```

```
1906 \def\XINT_iidivround_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space 0}%
```

### 3 Package *xintcore* implementation

```

1907 \def\XINT_iidivround_aiszero #1\Z #2\Z { 0}%
1908 \def\XINT_iidivround_bpos #1%
1909 {%
1910   \xint_UDsignfork
1911     #1{\xintiiopp\XINT_iidivround_pos {}}%
1912     -{\XINT_iidivround_pos #1}%
1913   \krof
1914 }%
1915 \def\XINT_iidivround_bneg #1%
1916 {%
1917   \xint_UDsignfork
1918     #1{\XINT_iidivround_pos {}}%
1919     -{\xintiiopp\XINT_iidivround_pos #1}%
1920   \krof
1921 }%
1922 \def\XINT_iidivround_pos #1#2\Z #3\Z{\expandafter\XINT_iidivround_pos_a
1923   \romannumeral0\XINT_div_prepare {#2}{#1#30}}%
1924 \def\XINT_iidivround_pos_a #1#2{\xintReverseOrder {#1\XINT_iidivround_pos_b}\Z }%
1925 \def\XINT_iidivround_pos_b #1#2{\xint_gob_til_Z #2\XINT_iidivround_pos_small\Z
1926   \XINT_iidivround_pos_c #1#2}%
1927 \def\XINT_iidivround_pos_c #1#2\Z {\ifnum #1>\xint_c_iv
1928   \expandafter\XINT_iidivround_pos_up
1929   \else \expandafter\xintreverseorder
1930   \fi {#2}}%
1931 \def\XINT_iidivround_pos_up #1{\xintinc {\xintReverseOrder{#1}}}%
1932 \def\XINT_iidivround_pos_small\Z\XINT_iidivround_pos_c #1#2%
1933   {\ifnum #1>\xint_c_iv\expandafter\xint_secondoftwo\else\expandafter
1934     \xint_firstoftwo\fi { 0}{ 1}}%

```

#### 3.18 *\xintiDivTrunc*, *\xintiiDivTrunc*

```

1935 \def\xintiDivTrunc {\romannumeral0\xintidivtrunc }%
1936 \def\xintidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
1937 \def\xintiiDivTrunc {\romannumeral0\xintiidivtrunc }%
1938 \def\xintiidivtrunc #1{\expandafter\XINT_iidivtrunc \romannumeral-`0#1\Z }%
1939 \def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1940   \romannumeral-`0#3\Z #2\Z }%
1941 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1942 {%
1943   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1944   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1945   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1946   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1947 }%
1948 \def\XINT_iidivtrunc_bpos #1%
1949 {%
1950   \xint_UDsignfork
1951     #1{\xintiiopp\XINT_iidivtrunc_pos {}}%
1952     -{\XINT_iidivtrunc_pos #1}%
1953   \krof
1954 }%
1955 \def\XINT_iidivtrunc_bneg #1%
1956 {%

```

### 3 Package *xintcore* implementation

```
1957 \xint_UDsignfork
1958     #1{\XINT_iidivtrunc_pos {}}%
1959     -{\xintiiopp\XINT_iidivtrunc_pos #1}%
1960 \krof
1961 }%
1962 \def\XINT_iidivtrunc_pos #1#2\Z #3\Z%
1963     {\expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_div_prepare {#2}{#1#3}}%
```

#### 3.19 *\xintiMod*, *\xintiiMod*

```
1964 \def\xintiMod     {\romannumeral0\xintimod }%
1965 \def\xintimod     #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
1966 \def\xintiiMod     {\romannumeral0\xintiiimod }%
1967 \def\xintiiimod   #1{\expandafter\XINT_iimod \romannumeral-`0#1\Z }%
1968 \def\XINT_iimod   #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
1969     \romannumeral-`0#3\Z #2\Z }%
1970 \def\XINT_iimod_a #1#2% #1 de A, #2 de B.
1971 {%
1972     \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1973     \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1974     \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
1975     \xint_orthat{\XINT_iimod_bpos #1#2}%
1976 }%
1977 \def\XINT_iimod_bpos #1%
1978 {%
1979     \xint_UDsignfork
1980     #1{\xintiiopp\XINT_iimod_pos {}}%
1981     -{\XINT_iimod_pos #1}%
1982     \krof
1983 }%
1984 \def\XINT_iimod_bneg #1%
1985 {%
1986     \xint_UDsignfork
1987     #1{\xintiiopp\XINT_iimod_pos {}}%
1988     -{\XINT_iimod_pos #1}%
1989     \krof
1990 }%
1991 \def\XINT_iimod_pos #1#2\Z #3\Z%
1992     {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare {#2}{#1#3}}%
```

#### 3.20 *\xintDec*

v1.08

```
1993 \def\xintDec     {\romannumeral0\xintdec }%
1994 \def\xintdec     #1%
1995 {%
1996     \expandafter\XINT_dec\romannumeral-`0#1%
1997     \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
1998 }%
1999 \def\XINT_dec     #1%
2000 {%
2001     \xint_UDzerominusfork
2002     #1-\XINT_dec_zero
```

### 3 Package *xintcore* implementation

```
2003     0#1\XINT_dec_neg
2004     0-{\XINT_dec_pos #1}%
2005     \krof
2006 }%
2007 \def\XINT_dec_zero #1\W\W\W\W\W\W\W\W { -1}%
2008 \def\XINT_dec_neg
2009     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_inc_pos }%
2010 \def\XINT_dec_pos
2011 {%
2012     \expandafter\XINT_dec_a \expandafter{\expandafter}%
2013     \romannumeral0\XINT_0Q {}}%
2014 }%
2015 \def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
2016 {%
2017     \expandafter\XINT_dec_b
2018     \the\numexpr 11#9#8#7#6#5#4#3#2-\xint_c_i\relax {#1}%
2019 }%
2020 \def\XINT_dec_b 1#1%
2021 {%
2022     \xint_gob_til_one #1\XINT_dec_A 1\XINT_dec_c
2023 }%
2024 \def\XINT_dec_c #1#2#3#4#5#6#7#8#9{\XINT_dec_a {#1#2#3#4#5#6#7#8#9}}%
2025 \def\XINT_dec_A 1\XINT_dec_c #1#2#3#4#5#6#7#8#9%
2026     {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
2027 \def\XINT_dec_B #1#2\W\W\W\W\W\W\W\W
2028 {%
2029     \expandafter\XINT_dec_cleanup
2030     \romannumeral0\XINT_rord_main { }#2%
2031     \xint_relax
2032     \xint_bye\xint_bye\xint_bye\xint_bye
2033     \xint_bye\xint_bye\xint_bye\xint_bye
2034     \xint_relax
2035     #1%
2036 }%
2037 \edef\XINT_dec_cleanup #1#2#3#4#5#6#7#8%
2038     {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax }%
```

#### 3.21 *\xintInc*

v1.08

```
2039 \def\xintInc {\romannumeral0\xintinc }%
2040 \def\xintinc #1%
2041 {%
2042     \expandafter\XINT_inc\romannumeral-\`0#1%
2043     \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
2044 }%
2045 \def\XINT_inc #1%
2046 {%
2047     \xint_UDzerominusfork
2048     #1-\XINT_inc_zero
2049     0#1\XINT_inc_neg
2050     0-{\XINT_inc_pos #1}%
```



## 4 Package `xint` implementation

```

2051 \krof
2052 }%
2053 \def\XINT_inc_zero #1\W\W\W\W\W\W\W { 1}%
2054 \def\XINT_inc_neg {\expandafter\XINT_opp\romannumeral0\XINT_dec_pos }%
2055 \def\XINT_inc_pos
2056 {%
2057 \expandafter\XINT_inc_a \expandafter{\expandafter}%
2058 \romannumeral0\XINT_0Q }%
2059 }%
2060 \def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
2061 {%
2062 \xint_gob_til_W #9\XINT_inc_end\W
2063 \expandafter\XINT_inc_b
2064 \the\numexpr 10#9#8#7#6#5#4#3#2+\xint_c_i\relax {#1}%
2065 }%
2066 \def\XINT_inc_b 1#1%
2067 {%
2068 \xint_gob_til_zero #1\XINT_inc_A 0\XINT_inc_c
2069 }%
2070 \def\XINT_inc_c #1#2#3#4#5#6#7#8#9{\XINT_inc_a {#1#2#3#4#5#6#7#8#9}}%
2071 \def\XINT_inc_A 0\XINT_inc_c #1#2#3#4#5#6#7#8#9%
2072 {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
2073 \def\XINT_inc_end\W #1\relax #2{ 1#2}%
2074 \XINT_restorecatcodes_endinput%

```

## 4 Package `xint` implementation

<ul style="list-style-type: none"> <li>.1 Package identification . . . . . 82</li> <li>.2 More token management . . . . . 83</li> <li>.3 <code>\xintSgnFork</code> . . . . . 83</li> <li>.4 <code>\xintIsOne, \xintiiIsOne</code> . . . . . 83</li> <li>.5 <code>\XINT_SQ</code> . . . . . 83</li> <li>.6 <code>\xintRev</code> . . . . . 84</li> <li>.7 <code>\xintLen</code> . . . . . 84</li> <li>.8 <code>\xintBool, \xintToggle</code> . . . . . 85</li> <li>.9 <code>\xintifSgn, \xintiiifSgn</code> . . . . . 85</li> <li>.10 <code>\xintifZero, \xintifNotZero, \xintii-</code> <code>ifZero, \xintiiifNotZero</code> . . . . . 86</li> <li>.11 <code>\xintifOne, \xintiiifOne</code> . . . . . 86</li> <li>.12 <code>\xintifTrueAelseB, \xintifFalseAelseB</code> 87</li> <li>.13 <code>\xintifCmp, \xintiiifCmp</code> . . . . . 87</li> <li>.14 <code>\xintifEq, \xintiiifEq</code> . . . . . 87</li> <li>.15 <code>\xintifGt, \xintiiifGt</code> . . . . . 88</li> <li>.16 <code>\xintifLt, \xintiiifLt</code> . . . . . 88</li> <li>.17 <code>\xintifOdd, \xintiiifOdd</code> . . . . . 89</li> <li>.18 <code>\xintCmp, \xintiiCmp</code> . . . . . 89</li> <li>.19 <code>\xintEq, \xintGt, \xintLt</code> . . . . . 92</li> <li>.20 <code>\xintNeq, \xintGtorEq, \xintLtorEq</code> . . 92</li> <li>.21 <code>\xintiiEq, \xintiiGt, \xintiiLt</code> . . . . 92</li> <li>.22 <code>\xintiiNeq, \xintiiGtorEq, \xintiiLtorEq</code> 92</li> <li>.23 <code>\xintIsZero, \xintIsNotZero, \xintii-</code> <code>IsZero, \xintiiIsNotZero</code> . . . . . 92</li> </ul>	<ul style="list-style-type: none"> <li>.24 <code>\xintIsTrue, \xintNot, \xintIsFalse</code> . . 92</li> <li>.25 <code>\xintAND, \xintOR, \xintXOR</code> . . . . . 93</li> <li>.26 <code>\xintANDof</code> . . . . . 93</li> <li>.27 <code>\xintORof</code> . . . . . 93</li> <li>.28 <code>\xintXORof</code> . . . . . 93</li> <li>.29 <code>\xintGeq</code> . . . . . 94</li> <li>.30 <code>\xintiMax, \xintiiMax</code> . . . . . 95</li> <li>.31 <code>\xintMaxof</code> . . . . . 97</li> <li>.32 <code>\xintiMin, \xintiiMin</code> . . . . . 97</li> <li>.33 <code>\xintiMinof</code> . . . . . 98</li> <li>.34 <code>\xintiiSum</code> . . . . . 99</li> <li>.35 <code>\xintiiPrd</code> . . . . . 100</li> <li>.36 <code>\xintFac</code> . . . . . 101</li> <li>.37 <code>\xintFDg, \xintiiFDg</code> . . . . . 102</li> <li>.38 <code>\xintLDg, \xintiiLDg</code> . . . . . 103</li> <li>.39 <code>\xintMON, \xintMMON, \xintiiMON, \xinti-</code> <code>imMON</code> . . . . . 103</li> <li>.40 <code>\xintOdd, \xintiiOdd, \xintEven, \xinti-</code> <code>iEven</code> . . . . . 104</li> <li>.41 <code>\xintDSL</code> . . . . . 105</li> <li>.42 <code>\xintDSR</code> . . . . . 105</li> <li>.43 <code>\xintDSH, \xintDSHr</code> . . . . . 106</li> <li>.44 <code>\xintDSx</code> . . . . . 107</li> <li>.45 <code>\xintDecSplit, \xintDecSplitL, \xintDec-</code> <code>SplitR</code> . . . . . 109</li> </ul>
---	--

.46	<code>\xintDouble</code>	113	<code>Root</code>	115
.47	<code>\xintHalf</code>	114	.49 <code>\xintiiE</code>	118
.48	<code>\xintiiSqrt, \xintiiSqrtR, \xintiiSquare-</code>			

The basic arithmetic routines `\xintiiAdd`, `\xintiiSub`, `\xintiiMul`, `\xintiiQuo` and `\xintiiPow` have been moved to new package `xintcore`.

```

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

```

## 4.1 Package identification

```
44 \XINT_providespackage
```

```
45 \ProvidesPackage{xint}%
46 [2014/11/07 v1.1a 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 `\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`.

```
53 \def\xintSgnFork {\romannumeral0\xintsgnfork }%
54 \def\xintsgnfork #1%
55 {%
56   \ifcase #1 \expandafter\xint_secondofthree_thenstop
57             \or\xexpandafter\xint_thirdofthree_thenstop
58             \else\xexpandafter\xint_firstofthree_thenstop
59   \fi
60 }%
```

## 4.4 `\xintIsOne`, `\xintiiIsOne`

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

```
61 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
62 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral-`0#1\W\Z }%
63 \def\xintIsOne {\romannumeral0\xintisone }%
64 \def\xintisone #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}\W\Z }%
65 \def\XINT_isOne #1{\romannumeral0\XINT_isone #1\W\Z }%
66 \def\XINT_isone #1#2%
67 {%
68   \xint_gob_til_one #1\XINT_isone_b 1%
69   \expandafter\space\expandafter 0\xint_gob_til_Z #2%
70 }%
71 \def\XINT_isone_b #1\xint_gob_til_Z #2%
72 {%
73   \xint_gob_til_W #2\XINT_isone_yes \W
74   \expandafter\space\expandafter 0\xint_gob_til_Z
75 }%
76 \def\XINT_isone_yes #1\Z { 1}%
```

## 4.5 `\XINT_SQ`

```
77 \def\XINT_SQ #1#2#3#4#5#6#7#8%
78 {%
79   \xint_gob_til_R #8\XINT_SQ_end_a\R\XINT_SQ {#8#7#6#5#4#3#2#1}%
80 }%
81 \def\XINT_SQ_end_a\R\XINT_SQ #1#2\Z
```

```

82 {%
83   \XINT_SQ_end_b #1\Z
84 }%
85 \def\XINT_SQ_end_b #1#2#3#4#5#6#7%
86 {%
87   \xint_gob_til_R
88     #7\XINT_SQ_end_vii
89     #6\XINT_SQ_end_vi
90     #5\XINT_SQ_end_v
91     #4\XINT_SQ_end_iv
92     #3\XINT_SQ_end_iii
93     #2\XINT_SQ_end_ii
94     \R\XINT_SQ_end_i
95     \Z #2#3#4#5#6#7%
96 }%
97 \def\XINT_SQ_end_vii #1\Z #2#3#4#5#6#7#8\Z { #8}%
98 \def\XINT_SQ_end_vi #1\Z #2#3#4#5#6#7#8\Z { #7#8000000}%
99 \def\XINT_SQ_end_v #1\Z #2#3#4#5#6#7#8\Z { #6#7#800000}%
100 \def\XINT_SQ_end_iv #1\Z #2#3#4#5#6#7#8\Z { #5#6#7#80000}%
101 \def\XINT_SQ_end_iii #1\Z #2#3#4#5#6#7#8\Z { #4#5#6#7#8000}%
102 \def\XINT_SQ_end_ii #1\Z #2#3#4#5#6#7#8\Z { #3#4#5#6#7#800}%
103 \def\XINT_SQ_end_i #1\Z #2#3#4#5#6#7\Z { #1#2#3#4#5#6#70}%

```

#### 4.6 `\xintRev`

`\xintRev`: expands fully its argument `\romannumeral-`0`, and checks the sign. However this last aspect does not appear like a very useful thing. And despite the fact that a special check is made for a sign, actually the input is not given to `\xintnum`, contrarily to `\xintLen`. This is all a bit incoherent. Should be fixed.

```

104 \def\xintRev {\romannumeral0\xintrev }%
105 \def\xintrev #1%
106 {%
107   \expandafter\XINT_rev_fork
108   \romannumeral-`0#1\xint_relax % empty #1 ok, \xint_relax stops expansion
109     \xint_bye\xint_bye\xint_bye\xint_bye
110     \xint_bye\xint_bye\xint_bye\xint_bye
111   \xint_relax
112 }%
113 \def\XINT_rev_fork #1%
114 {%
115   \xint_UDsignfork
116   #1{\expandafter\xint_minus_thenstop\romannumeral0\XINT_rord_main {}}%
117   -{\XINT_rord_main {}}#1}%
118   \krof
119 }%

```

#### 4.7 `\xintLen`

`\xintLen` is ONLY for (possibly long) integers. Gets extended to fractions by `xintfrac.sty`

```

120 \def\xintLen {\romannumeral0\xintlen }%
121 \def\xintlen #1%
122 {%

```

#### 4 Package *xint* implementation

```
123 \expandafter\XINT_len_fork
124 \romannumeral0\xintnum{#1}\xint_relax\xint_relax\xint_relax\xint_relax
125 \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
126 }%
127 \def\XINT_Len #1% variant which does not expand via \xintnum.
128 {%
129 \romannumeral0\XINT_len_fork
130 #1\xint_relax\xint_relax\xint_relax\xint_relax
131 \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
132 }%
133 \def\XINT_len_fork #1%
134 {%
135 \expandafter\XINT_length_loop
136 \xint_UDsignfork
137 #1{0.}%
138 -{0.#1}%
139 \krof
140 }%
```

### 4.8 `\xintBool`, `\xintToggle`

1.09c

```
141 \def\xintBool #1{\romannumeral-`0%
142 \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
143 \def\xintToggle #1{\romannumeral-`0\iftoggle{#1}{1}{0}}%
```

### 4.9 `\xintifSgn`, `\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`)

```
144 \def\xintifSgn {\romannumeral0\xintifsgn }%
145 \def\xintifsgn #1%
146 {%
147 \ifcase \xintSgn{#1}
148 \expandafter\xint_secondofthree_thenstop
149 \or\expandafter\xint_thirdofthree_thenstop
150 \else\expandafter\xint_firstofthree_thenstop
151 \fi
152 }%
153 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
154 \def\xintiiifsgn #1%
155 {%
156 \ifcase \xintiiSgn{#1}
157 \expandafter\xint_secondofthree_thenstop
158 \or\expandafter\xint_thirdofthree_thenstop
159 \else\expandafter\xint_firstofthree_thenstop
160 \fi
161 }%
```

**4.10 `\xintifZero`, `\xintifNotZero`, `\xintiiifZero`, `\xintiiifNotZero`**

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.

```

162 \def\xintifZero {\romannumeral0\xintifzero }%
163 \def\xintifzero #1%
164 {%
165   \if0\xintSgn{#1}%
166     \expandafter\xint_firstoftwo_thenstop
167   \else
168     \expandafter\xint_secondoftwo_thenstop
169   \fi
170 }%
171 \def\xintifNotZero {\romannumeral0\xintifnotzero }%
172 \def\xintifnotzero #1%
173 {%
174   \if0\xintSgn{#1}%
175     \expandafter\xint_secondoftwo_thenstop
176   \else
177     \expandafter\xint_firstoftwo_thenstop
178   \fi
179 }%
180 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
181 \def\xintiiifzero #1%
182 {%
183   \if0\xintiiSgn{#1}%
184     \expandafter\xint_firstoftwo_thenstop
185   \else
186     \expandafter\xint_secondoftwo_thenstop
187   \fi
188 }%
189 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
190 \def\xintiiifnotzero #1%
191 {%
192   \if0\xintiiSgn{#1}%
193     \expandafter\xint_secondoftwo_thenstop
194   \else
195     \expandafter\xint_firstoftwo_thenstop
196   \fi
197 }%

```

**4.11 `\xintifOne`, `\xintiiifOne`**

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

```

198 \def\xintiiifOne {\romannumeral0\xintiiifone }%
199 \def\xintiiifone #1%
200 {%
201   \if1\xintiiIsOne{#1}%
202     \expandafter\xint_firstoftwo_thenstop
203   \else

```

## 4 Package *xint* implementation

```
204     \expandafter\xint_secondoftwo_thenstop
205   \fi
206 }%
207 \def\xintifOne {\romannumeral0\xintifone }%
208 \def\xintifone #1%
209 {%
210   \if1\xintIsOne{#1}%
211     \expandafter\xint_firstoftwo_thenstop
212   \else
213     \expandafter\xint_secondoftwo_thenstop
214   \fi
215 }%
```

### 4.12 `\xintifTrueAelseB`, `\xintifFalseAelseB`

1.09i. Warning, `\xintifTrueFalse`, `\xintifTrue` deprecated, to be removed

```
216 \let\xintifTrueAelseB\xintifNotZero
217 \let\xintifFalseAelseB\xintifZero
218 \let\xintifTrue\xintifNotZero
219 \let\xintifTrueFalse\xintifNotZero
```

### 4.13 `\xintifCmp`, `\xintiifCmp`

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

```
220 \def\xintifCmp {\romannumeral0\xintifcmp }%
221 \def\xintifcmp #1#2%
222 {%
223   \ifcase\xintCmp {#1}{#2}
224     \expandafter\xint_secondofthree_thenstop
225     \or\expandafter\xint_thirdofthree_thenstop
226     \else\expandafter\xint_firstofthree_thenstop
227   \fi
228 }%
229 \def\xintiifCmp {\romannumeral0\xintiifcmp }%
230 \def\xintiifcmp #1#2%
231 {%
232   \ifcase\xintiiCmp {#1}{#2}
233     \expandafter\xint_secondofthree_thenstop
234     \or\expandafter\xint_thirdofthree_thenstop
235     \else\expandafter\xint_firstofthree_thenstop
236   \fi
237 }%
```

### 4.14 `\xintifEq`, `\xintiifEq`

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

```
238 \def\xintifEq {\romannumeral0\xintifeq }%
239 \def\xintifeq #1#2%
240 {%
241   \if0\xintCmp{#1}{#2}%
```

```

242         \expandafter\xint_firstoftwo_thenstop
243     \else\expandafter\xint_secondoftwo_thenstop
244 \fi
245 }%
246 \def\xintiiifEq {\romannumeral0\xintiiifeq }%
247 \def\xintiiifeq #1#2%
248 {%
249     \if0\xintiiCmp{#1}{#2}%
250         \expandafter\xint_firstoftwo_thenstop
251     \else\expandafter\xint_secondoftwo_thenstop
252 \fi
253 }%

```

#### 4.15 `\xintifGt`, `\xintiiifGt`

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

```

254 \def\xintifGt {\romannumeral0\xintifgt }%
255 \def\xintifgt #1#2%
256 {%
257     \if1\xintCmp{#1}{#2}%
258         \expandafter\xint_firstoftwo_thenstop
259     \else\expandafter\xint_secondoftwo_thenstop
260 \fi
261 }%
262 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
263 \def\xintiiifgt #1#2%
264 {%
265     \if1\xintiiCmp{#1}{#2}%
266         \expandafter\xint_firstoftwo_thenstop
267     \else\expandafter\xint_secondoftwo_thenstop
268 \fi
269 }%

```

#### 4.16 `\xintifLt`, `\xintiiifLt`

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

```

270 \def\xintifLt {\romannumeral0\xintiflt }%
271 \def\xintiflt #1#2%
272 {%
273     \ifnum\xintCmp{#1}{#2}<\xint_c_
274         \expandafter\xint_firstoftwo_thenstop
275     \else \expandafter\xint_secondoftwo_thenstop
276 \fi
277 }%
278 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
279 \def\xintiiiflt #1#2%
280 {%
281     \ifnum\xintiiCmp{#1}{#2}<\xint_c_
282         \expandafter\xint_firstoftwo_thenstop
283     \else \expandafter\xint_secondoftwo_thenstop
284 \fi

```



285 }%

#### 4.17 `\xintifOdd`, `\xintiiifOdd`

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

```
286 \def\xintiiifOdd {\romannumeral0\xintiiifodd }%
287 \def\xintiiifodd #1%
288 {%
289   \if\xintiiOdd{#1}1%
290   \expandafter\xint_firstoftwo_thenstop
291   \else
292   \expandafter\xint_secondoftwo_thenstop
293   \fi
294 }%
295 \def\xintifOdd {\romannumeral0\xintifodd }%
296 \def\xintifodd #1%
297 {%
298   \if\xintOdd{#1}1%
299   \expandafter\xint_firstoftwo_thenstop
300   \else
301   \expandafter\xint_secondoftwo_thenstop
302   \fi
303 }%
```

#### 4.18 `\xintCmp`, `\xintiiCmp`

Release 1.09a has `\xintnum` inserted into `\xintCmp`. Unnecessary `\xintiCmp` suppressed in 1.09f. And 1.1a does `\xintiiCmp`, for optimization in `\xintiexpr`. (not needed before, because `\XINT_cmp_fork` was directly used, or `\XINT_Cmp`)

```
304 \def\xintCmp {\romannumeral0\xintcmp }%
305 \def\xintcmp #1%
306 {%
307   \expandafter\xint_cmp\expandafter{\romannumeral0\xintnum{#1}}%
308 }%
309 \def\xint_cmp #1#2%
310 {%
311   \expandafter\XINT_cmp_fork \romannumeral0\xintnum{#2}\Z #1\Z
312 }%
313 \def\xintiiCmp {\romannumeral0\xintiicmp }%
314 \def\xintiicmp #1%
315 {%
316   \expandafter\xint_iicmp\expandafter{\romannumeral-`0#1}%
317 }%
318 \def\xint_iicmp #1#2%
319 {%
320   \expandafter\XINT_cmp_fork \romannumeral-`0#2\Z #1\Z
321 }%
322 \def\XINT_Cmp #1#2{\romannumeral0\XINT_cmp_fork #2\Z #1\Z }%
```

#### COMPARAISON

1 si  $\#3\#4 > \#1\#2$ , 0 si  $\#3\#4 = \#1\#2$ , -1 si  $\#3\#4 < \#1\#2$   
 $\#3\#4$  vient du \*premier\*,  $\#1\#2$  vient du \*second\*

#### 4 Package *xint* implementation

```

323 \def\XINT_cmp_fork #1#2\Z #3#4\Z
324 {%
325     \xint_UDsignsfork
326         #1#3\XINT_cmp_minusminus
327         #1-\XINT_cmp_minusplus
328         #3-\XINT_cmp_plusminus
329         --{\xint_UDzerosfork
330             #1#3\XINT_cmp_zerozero
331             #10\XINT_cmp_zeroplus
332             #30\XINT_cmp_pluszero
333             00\XINT_cmp_plusplus
334         \krof }%
335     \krof
336     {#2}{#4}#1#3%
337 }%
338 \def\XINT_cmp_minusplus #1#2#3#4{ 1}%
339 \def\XINT_cmp_plusminus #1#2#3#4{ -1}%
340 \def\XINT_cmp_zerozero #1#2#3#4{ 0}%
341 \def\XINT_cmp_zeroplus #1#2#3#4{ 1}%
342 \def\XINT_cmp_pluszero #1#2#3#4{ -1}%
343 \def\XINT_cmp_plusplus #1#2#3#4%
344 {%
345     \XINT_cmp_pre {#4#2}{#3#1}%
346 }%
347 \def\XINT_cmp_minusminus #1#2#3#4%
348 {%
349     \XINT_cmp_pre {#1}{#2}%
350 }%
351 \def\XINT_cmp_pre #1%
352 {%
353     \expandafter\XINT_cmp_pre_b\expandafter
354     {\romannumeral0\XINT_RQ { }#1\R\R\R\R\R\R\R\R\Z }%
355 }%
356 \def\XINT_cmp_pre_b #1#2%
357 {%
358     \expandafter\XINT_cmp_A
359     \expandafter1\expandafter{\expandafter}%
360     \romannumeral0\XINT_RQ { }#2\R\R\R\R\R\R\R\R\Z
361     \W\X\Y\Z #1\W\X\Y\Z
362 }%

```

#### COMPARAISON

N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS POUR QUE LEUR LONGUEURS À CHACUN SOIENT MULTIPLES DE 4, MAIS AUCUN NE SE TERMINE EN 0000. routine appelée via

`\XINT_cmp_A 1{<N1>\W\X\Y\Z<N2>\W\X\Y\Z`

ATTENTION RENVOIE 1 SI  $N1 < N2$ , 0 si  $N1 = N2$ , -1 si  $N1 > N2$

```

363 \def\XINT_cmp_A #1#2#3\W\X\Y\Z #4#5#6#7%
364 {%
365     \xint_gob_til_W #4\xint_cmp_az\W
366     \XINT_cmp_B #1{#4#5#6#7}{#2}{#3}\W\X\Y\Z
367 }%
368 \def\XINT_cmp_B #1#2#3#4#5#6#7%

```

#### 4 Package *xint* implementation

```

369 {%
370   \xint_gob_til_W#4\xint_cmp_bz\W
371   \XINT_cmp_onestep #1#2{#7#6#5#4}{#3}%
372 }%
373 \def\XINT_cmp_onestep #1#2#3#4#5#6%
374 {%
375   \expandafter\XINT_cmp_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
376 }%
377 \def\XINT_cmp_backtoA #1#2#3.#4%
378 {%
379   \XINT_cmp_A #2{#3#4}%
380 }%
381 \def\xint_cmp_bz\W\XINT_cmp_onestep #1\Z { 1}%
382 \def\xint_cmp_az\W\XINT_cmp_B #1#2#3#4#5#6#7%
383 {%
384   \xint_gob_til_W #4\xint_cmp_ez\W
385   \XINT_cmp_Eenter #1{#3}#4#5#6#7%
386 }%
387 \def\XINT_cmp_Eenter #1\Z { -1}%
388 \def\xint_cmp_ez\W\XINT_cmp_Eenter #1%
389 {%
390   \xint_UDzerofork
391   #1\XINT_cmp_K           %   il y a une retenue
392   0\XINT_cmp_L           %   pas de retenue
393   \krof
394 }%
395 \def\XINT_cmp_K #1\Z { -1}%
396 \def\XINT_cmp_L #1{\XINT_OneIfPositive_main #1}%
397 \def\XINT_OneIfPositive #1%
398 {%
399   \XINT_OneIfPositive_main #1\W\X\Y\Z%
400 }%
401 \def\XINT_OneIfPositive_main #1#2#3#4%
402 {%
403   \xint_gob_til_Z #4\xint_OneIfPositive_terminated\Z
404   \XINT_OneIfPositive_onestep #1#2#3#4%
405 }%
406 \def\xint_OneIfPositive_terminated\Z\XINT_OneIfPositive_onestep\W\X\Y\Z { 0}%
407 \def\XINT_OneIfPositive_onestep #1#2#3#4%
408 {%
409   \expandafter\XINT_OneIfPositive_check\the\numexpr #1#2#3#4\relax
410 }%
411 \def\XINT_OneIfPositive_check #1%
412 {%
413   \xint_gob_til_zero #1\xint_OneIfPositive_backtomain 0%
414   \XINT_OneIfPositive_finish #1%
415 }%
416 \def\XINT_OneIfPositive_finish #1\W\X\Y\Z{ 1}%
417 \def\xint_OneIfPositive_backtomain 0\XINT_OneIfPositive_finish 0%
418   {\XINT_OneIfPositive_main }%

```

#### 4.19 `\xintEq`, `\xintGt`, `\xintLt`

1.09a.

```

419 \def\xintEq {\romannumeral0\xinteq } \def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
420 \def\xintGt {\romannumeral0\xintgt } \def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
421 \def\xintLt {\romannumeral0\xintl } \def\xintl #1#2{\xintiflt{#1}{#2}{1}{0}}%

```

#### 4.20 `\xintNeq`, `\xintGtorEq`, `\xintLtorEq`

1.1. Pour `xintexpr`. No lowercase macros

```

422 \def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
423 \def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
424 \def\xintNeq #1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%

```

#### 4.21 `\xintiiEq`, `\xintiiGt`, `\xintiiLt`

1.1a Pour `\xintiiexpr`. No lowercase macros.

```

425 \def\xintiiEq #1#2{\romannumeral0\xintiiifeq{#1}{#2}{1}{0}}%
426 \def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%
427 \def\xintiiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%

```

#### 4.22 `\xintiiNeq`, `\xintiiGtorEq`, `\xintiiLtorEq`

1.1a. Pour `\xintiiexpr`. No lowercase macros.

```

428 \def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
429 \def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
430 \def\xintiiNeq #1#2{\romannumeral0\xintiiifeq {#1}{#2}{0}{1}}%

```

#### 4.23 `\xintIsZero`, `\xintIsNotZero`, `\xintiiIsZero`, `\xintiiIsNotZero`

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

```

431 \def\xintIsZero {\romannumeral0\xintiszero }%
432 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
433 \def\xintIsNotZero {\romannumeral0\xintisnotzero }%
434 \def\xintisnotzero
435 #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
436 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
437 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
438 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
439 \def\xintiiisnotzero
440 #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%

```

#### 4.24 `\xintIsTrue`, `\xintNot`, `\xintIsFalse`

1.09c

```

441 \let\xintIsTrue\xintIsNotZero
442 \let\xintNot\xintIsZero
443 \let\xintIsFalse\xintIsZero

```

## 4.25 `\xintAND`, `\xintOR`, `\xintXOR`

1.09a. Embarrassing bugs in `\xintAND` and `\xintOR` which inserted a space token corrected in 1.09i. `\xintxor` restyled with `\if` (faster) in 1.09i

```

444 \def\xintAND {\romannumeral0\xintand }%
445 \def\xintand #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
446             \else\expandafter\xint_secondoftwo\fi
447             { 0}{\xintisnotzero{#2}}}%
448 \def\xintOR {\romannumeral0\xintor }%
449 \def\xintor #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
450             \else\expandafter\xint_secondoftwo\fi
451             {\xintisnotzero{#2}}{ 1}}%
452 \def\xintXOR {\romannumeral0\xintxor }%
453 \def\xintxor #1#2{\if\xintIsZero{#1}\xintIsZero{#2}%
454             \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%

```

## 4.26 `\xintANDof`

New with 1.09a. `\xintANDof` works also with an empty list.

```

455 \def\xintANDof {\romannumeral0\xintandof }%
456 \def\xintandof #1{\expandafter\XINT_andof_a\romannumeral-`0#1\relax }%
457 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral-`0#1\Z }%
458 \def\XINT_andof_b #1%
459     {\xint_gob_til_relax #1\XINT_andof_e\relax\XINT_andof_c #1}%
460 \def\XINT_andof_c #1\Z
461     {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
462 \def\XINT_andof_no #1\relax { 0}%
463 \def\XINT_andof_e #1\Z { 1}%

```

## 4.27 `\xintORof`

New with 1.09a. Works also with an empty list.

```

464 \def\xintORof {\romannumeral0\xintorof }%
465 \def\xintorof #1{\expandafter\XINT_orof_a\romannumeral-`0#1\relax }%
466 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral-`0#1\Z }%
467 \def\XINT_orof_b #1%
468     {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
469 \def\XINT_orof_c #1\Z
470     {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
471 \def\XINT_orof_yes #1\relax { 1}%
472 \def\XINT_orof_e #1\Z { 0}%

```

## 4.28 `\xintXORof`

New with 1.09a. Works with an empty list, too. `\XINT_xorof_c` more efficient in 1.09i

```

473 \def\xintXORof {\romannumeral0\xintxorof }%
474 \def\xintxorof #1{\expandafter\XINT_xorof_a\expandafter
475     \romannumeral-`0#1\relax }%
476 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral-`0#2\Z #1}%

```

#### 4 Package `xint` implementation

```
477 \def\XINT_xorof_b #1%
478     {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
479 \def\XINT_xorof_c #1\Z #2%
480     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
481         \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
482         {\XINT_xorof_a #2}%
483     }%
484 \def\XINT_xorof_e #1\Z #2{ #2}%
```

### 4.29 `\xintGeq`

Release 1.09a has `\xintnum` added into `\xintGeq`. PLUS GRAND OU ÉGAL attention compare les **\*\*valeurs absolues\*\***

```
485 \def\xintGeq {\romannumeral0\xintgeq }%
486 \def\xintgeq #1%
487 {%
488     \expandafter\xint_geq\expandafter {\romannumeral0\xintnum{#1}}%
489 }%
490 \def\xint_geq #1#2%
491 {%
492     \expandafter\XINT_geq_fork \romannumeral0\xintnum{#2}\Z #1\Z
493 }%
494 \def\XINT_Geq #1#2{\romannumeral0\XINT_geq_fork #2\Z #1\Z }%
```

PLUS GRAND OU ÉGAL ATTENTION, TESTE les VALEURS ABSOLUES

```
495 \def\XINT_geq_fork #1#2\Z #3#4\Z
496 {%
497     \xint_UDzerofork
498     #1\XINT_geq_secondiszero %|#1#2|=0
499     #3\XINT_geq_firstiszero %|#1#2|>0
500     0{\xint_UDsignsfork
501         #1#3\XINT_geq_minusminus
502         #1-\XINT_geq_minusplus
503         #3-\XINT_geq_plusminus
504         --\XINT_geq_plusplus
505     \krof }%
506     \krof
507     {#2}{#4}#1#3%
508 }%
509 \def\XINT_geq_secondiszero #1#2#3#4{ 1}%
510 \def\XINT_geq_firstiszero #1#2#3#4{ 0}%
511 \def\XINT_geq_plusplus #1#2#3#4{\XINT_geq_pre {#4#2}{#3#1}}%
512 \def\XINT_geq_minusminus #1#2#3#4{\XINT_geq_pre {#2}{#1}}%
513 \def\XINT_geq_minusplus #1#2#3#4{\XINT_geq_pre {#4#2}{#1}}%
514 \def\XINT_geq_plusminus #1#2#3#4{\XINT_geq_pre {#2}{#3#1}}%
515 \def\XINT_geq_pre #1%
516 {%
517     \expandafter\XINT_geq_pre_b\expandafter
518     {\romannumeral0\XINT_RQ {#1}\R\R\R\R\R\R\R\R\Z }%
519 }%
520 \def\XINT_geq_pre_b #1#2%
```

## 4 Package *xint* implementation

```
521 {%
522   \expandafter\XINT_geq_A
523   \expandafter1\expandafter{\expandafter}%
524   \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
525     \W\X\Y\Z #1 \W\X\Y\Z
526 }%

PLUS GRAND OU ÉGAL
N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS POUR QUE LEURS LONGUEURS À CHACUN
SOIENT MULTIPLÉS DE 4, MAIS AUCUN NE SE TERMINE EN 0000
routine appelée via
\romannumeral0\XINT_geq_A 1{<N1>\W\X\Y\Z<N2>\W\X\Y\Z
ATTENTION RENVOIE 1 SI N1 < N2 ou N1 = N2 et 0 si N1 > N2

527 \def\XINT_geq_A #1#2#3\W\X\Y\Z #4#5#6#7%
528 {%
529   \xint_gob_til_W #4\xint_geq_az\W
530   \XINT_geq_B #1{#4#5#6#7}{#2}{#3}\W\X\Y\Z
531 }%
532 \def\XINT_geq_B #1#2#3#4#5#6#7%
533 {%
534   \xint_gob_til_W #4\xint_geq_bz\W
535   \XINT_geq_onestep #1#2{#7#6#5#4}{#3}%
536 }%
537 \def\XINT_geq_onestep #1#2#3#4#5#6%
538 {%
539   \expandafter\XINT_geq_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
540 }%
541 \def\XINT_geq_backtoA #1#2#3.#4%
542 {%
543   \XINT_geq_A #2{#3#4}%
544 }%
545 \def\xint_geq_bz\W\XINT_geq_onestep #1\W\X\Y\Z { 1}%
546 \def\xint_geq_az\W\XINT_geq_B #1#2#3#4#5#6#7%
547 {%
548   \xint_gob_til_W #4\xint_geq_ez\W
549   \XINT_geq_Enter #1%
550 }%
551 \def\XINT_geq_Enter #1\W\X\Y\Z { 0}%
552 \def\xint_geq_ez\W\XINT_geq_Enter #1%
553 {%
554   \xint_UDzerofork
555     #1{ 0}          %      il y a une retenue
556     0{ 1}          %      pas de retenue
557   \krof
558 }%
```

### 4.30 `\xintiMax`, `\xintiiMax`

The rationale is that it is more efficient than using `\xintCmp`. 1.03 makes the code a tiny bit slower but easier to re-use for fractions. Note: actually since 1.08a code for fractions does not all reduce to these entry points, so perhaps I should revert the changes made in 1.03. Release 1.09a has `\xintnum` added into `\xintiMax`.

## 4 Package *xint* implementation

1.1 adds the missing `\xintiiMax`. Using `\xintMax` and not `\xintiMax` in *xint* is deprecated.

```

559 \def\xintiMax {\romannumeral0\xintimax }%
560 \def\xintimax #1%
561 {%
562   \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
563 }%
564 \def\xint_max #1#2%
565 {%
566   \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
567 }%
568 \def\xintiiMax {\romannumeral0\xintiimax }%
569 \def\xintiimax #1%
570 {%
571   \expandafter\xint_iimax\expandafter {\romannumeral-`0#1}%
572 }%
573 \def\xint_iimax #1#2%
574 {%
575   \expandafter\XINT_max_pre\expandafter {\romannumeral-`0#2}{#1}%
576 }%
577 \let\xintMax\xintiMax \let\xintmax\xintimax % deprecated, should be only with xintfrac
578 \def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
579 \def\XINT_Max #1#2{\romannumeral0\XINT_max_fork #2\Z #1\Z {#1}{#2}}%

```

*#3#4 vient du \*premier\*, #1#2 vient du \*second\**

```

580 \def\XINT_max_fork #1#2\Z #3#4\Z
581 {%
582   \xint_UDsignsfork
583     #1#3\XINT_max_minusminus % A < 0, B < 0
584     #1-\XINT_max_minusplus % B < 0, A >= 0
585     #3-\XINT_max_plusminus % A < 0, B >= 0
586     --{\xint_UDzerosfork
587       #1#3\XINT_max_zerozero % A = B = 0
588       #10\XINT_max_zeroplus % B = 0, A > 0
589       #30\XINT_max_pluszero % A = 0, B > 0
590       00\XINT_max_plusplus % A, B > 0
591     \krof }%
592   \krof
593   {#2}{#4}#1#3%
594 }%

```

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

```

595 \def\XINT_max_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
596 \def\XINT_max_zeroplus #1#2#3#4{\xint_firstoftwo_thenstop }%
597 \def\XINT_max_pluszero #1#2#3#4{\xint_secondoftwo_thenstop }%
598 \def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
599 \def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
600 \def\XINT_max_plusplus #1#2#3#4%
601 {%
602   \ifodd\XINT_Geq {#4#2}{#3#1}
603   \expandafter\xint_firstoftwo_thenstop
604   \else

```



#### 4 Package `xint` implementation

```
605 \expandafter\xint_secondoftwo_thenstop
606 \fi
607 }%
```

```
#3=-, #4=-, #1 = |B| = -B, #2 = |A| = -A
```

```
608 \def\xINT_max_minusminus #1#2#3#4%
609 {%
610 \ifodd\xINT_Geq {#1}{#2}
611 \expandafter\xint_firstoftwo_thenstop
612 \else
613 \expandafter\xint_secondoftwo_thenstop
614 \fi
615 }%
```

#### 4.31 `\xintMaxof`

New with 1.09a.

```
616 \def\xintiMaxof {\romannumeral0\xintimaxof }%
617 \def\xintimaxof #1{\expandafter\xINT_imaxof_a\romannumeral-`0#1\relax }%
618 \def\xINT_imaxof_a #1{\expandafter\xINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
619 \def\xINT_imaxof_b #1\Z #2%
620 {\expandafter\xINT_imaxof_c\romannumeral-`0#2\Z {#1}\Z}%
621 \def\xINT_imaxof_c #1%
622 {\xint_gob_til_relax #1\xINT_imaxof_e\relax\xINT_imaxof_d #1}%
623 \def\xINT_imaxof_d #1\Z
624 {\expandafter\xINT_imaxof_b\romannumeral0\xintimax {#1}}%
625 \def\xINT_imaxof_e #1\Z #2\Z { #2}%
626 \let\xintMaxof\xintiMaxof \let\xintmaxof\xintimaxof
```

#### 4.32 `\xintiMin`, `\xintiiMin`

`\xintnum` added New with 1.09a. I add `\xintiiMin` in 1.1 and mark as deprecated `\xintMin`, renamed `\xintiMin`.

```
627 \def\xintiMin {\romannumeral0\xintimin }%
628 \def\xintimin #1%
629 {%
630 \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
631 }%
632 \def\xint_min #1#2%
633 {%
634 \expandafter\xINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
635 }%
636 \def\xintiiMin {\romannumeral0\xintiimin }%
637 \def\xintiimin #1%
638 {%
639 \expandafter\xint_iimin\expandafter {\romannumeral-`0#1}%
640 }%
641 \def\xint_iimin #1#2%
642 {%
643 \expandafter\xINT_min_pre\expandafter {\romannumeral-`0#2}{#1}%

```

## 4 Package `xint` implementation

```
644 }%
645 \let\xintMin\xintiMin \let\xintmin\xintimin % deprecated
646 \def\XINT_min_pre #1#2{\XINT_min_fork #1\Z #2\Z {#2}{#1}}%
647 \def\XINT_Min #1#2{\romannumeral0\XINT_min_fork #2\Z #1\Z {#1}{#2}}%
```

*#3#4 vient du \*premier\*, #1#2 vient du \*second\**

```
648 \def\XINT_min_fork #1#2\Z #3#4\Z
649 {%
650   \xint_UDsignsfork
651     #1#3\XINT_min_minusminus % A < 0, B < 0
652     #1-\XINT_min_minusplus % B < 0, A >= 0
653     #3-\XINT_min_plusminus % A < 0, B >= 0
654     --{\xint_UDzerosfork
655       #1#3\XINT_min_zerozero % A = B = 0
656       #10\XINT_min_zeroplus % B = 0, A > 0
657       #30\XINT_min_pluszero % A = 0, B > 0
658       00\XINT_min_plusplus % A, B > 0
659     \krof }%
660   \krof
661   {#2}{#4}#1#3%
662 }%
```

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

```
663 \def\XINT_min_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
664 \def\XINT_min_zeroplus #1#2#3#4{\xint_secondoftwo_thenstop }%
665 \def\XINT_min_pluszero #1#2#3#4{\xint_firstoftwo_thenstop }%
666 \def\XINT_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
667 \def\XINT_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
668 \def\XINT_min_plusplus #1#2#3#4%
669 {%
670   \ifodd\XINT_Geq {#4#2}{#3#1}
671     \expandafter\xint_secondoftwo_thenstop
672   \else
673     \expandafter\xint_firstoftwo_thenstop
674   \fi
675 }%
```

*#3=-, #4=-, #1 = |B| = -B, #2 = |A| = -A*

```
676 \def\XINT_min_minusminus #1#2#3#4%
677 {%
678   \ifodd\XINT_Geq {#1}{#2}
679     \expandafter\xint_secondoftwo_thenstop
680   \else
681     \expandafter\xint_firstoftwo_thenstop
682   \fi
683 }%
```

### 4.33 `\xintiMinof`

1.09a

#### 4 Package *xint* implementation

```
684 \def\xintiMinof      {\romannumeral0\xintiminof }%
685 \def\xintiminof     #1{\expandafter\XINT_iminof_a\romannumeral-`0#1\relax }%
686 \def\XINT_iminof_a #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
687 \def\XINT_iminof_b #1\Z #2%
688     {\expandafter\XINT_iminof_c\romannumeral-`0#2\Z {#1}\Z}%
689 \def\XINT_iminof_c #1%
690     {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
691 \def\XINT_iminof_d #1\Z
692     {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
693 \def\XINT_iminof_e #1\Z #2\Z { #2}%
694 \let\xintMinof\xintiMinof \let\xintminof\xintiminof
```

#### 4.34 `\xintiiSum`

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

```
\xintSumExpr {a}{b}...{z}\relax
```

1.03 (drastically) simplifies and makes the routines more efficient (for big computations). Also the way `\xintSum` and `\xintSumExpr ...\relax` are related. has been modified. Now `\xintSumExpr \z \relax` is accepted input when `\z` expands to a list of braced terms (prior only `\xintSum {z}` or `\xintSum \z` was possible).

1.09a does NOT add the `\xintnum` overhead. 1.09h renames `\xintiSum` to `\xintiiSum` to correctly reflect this.

```
695 \def\xintiiSum {\romannumeral0\xintiisum }%
696 \def\xintiisum #1{\xintiisumexpr #1\relax }%
697 \def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
698 \def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral-`0}%
699 \let\xintSum\xintiiSum \let\xintsum\xintiisum
700 \let\xintSumExpr\xintiiSumExpr \let\xintsumexpr\xintiisumexpr
701 \def\XINT_sumexpr {\XINT_sum_loop {0000}{0000}}%
702 \def\XINT_sum_loop #1#2#3%
703 {%
704     \expandafter\XINT_sum_checks\romannumeral-`0#3\Z {#1}{#2}%
705 }%
706 \def\XINT_sum_checks #1%
707 {%
708     \xint_gob_til_relax #1\XINT_sum_finished\relax
709     \xint_gob_til_zero #1\XINT_sum_skipzeroinput0%
710     \xint_UDsignfork
711     #1\XINT_sum_N
712     -{\XINT_sum_P #1}%
713     \krof
714 }%
715 \def\XINT_sum_finished #1\Z #2#3%
716 {%
717     \XINT_sub_A 1{#3}\W\X\Y\Z #2\W\X\Y\Z
718 }%
719 \def\XINT_sum_skipzeroinput #1\krof #2\Z {\XINT_sum_loop }%
720 \def\XINT_sum_P #1\Z #2%
721 {%
722     \expandafter\XINT_sum_loop\expandafter
723     {\romannumeral0\expandafter
724     \XINT_addr_A\expandafter0\expandafter{\expandafter}}%
```

#### 4 Package *xint* implementation

```
725 \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
726 \W\X\Y\Z #2\W\X\Y\Z }%
727 }%
728 \def\XINT_sum_N #1\Z #2#3%
729 {%
730 \expandafter\XINT_sum_NN\expandafter
731 {\romannumeral0\expandafter
732 \XINT_addr_A\expandafter0\expandafter{\expandafter}%
733 \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
734 \W\X\Y\Z #3\W\X\Y\Z }{#2}%
735 }%
736 \def\XINT_sum_NN #1#2{\XINT_sum_loop {#2}{#1}}%
```

### 4.35 `\xintiiPrd`

```
\xintPrd {a}...{z}}
```

```
\xintPrdExpr {a}...{z}\relax
```

Release 1.02 modified the product routine. The earlier version was faster in situations where each new term is bigger than the product of all previous terms, a situation which arises in the algorithm for computing powers. The 1.02 version was changed to be more efficient on big products, where the new term is small compared to what has been computed so far (the power algorithm now has its own product routine).

Finally, the 1.03 version just simplifies everything as the multiplication now decides what is best, with the price of a little overhead. So the code has been dramatically reduced here.

In 1.03 I also modify the way `\xintPrd` and `\xintPrdExpr ...\relax` are related. Now `\xintPrdExpr \z \relax` is accepted input when `\z` expands to a list of braced terms (prior only `\xintPrd {z}` or `\xintPrd \z` was possible).

In 1.06a I suddenly decide that `\xintProductExpr` was a silly name, and as the package is new and certainly not used, I decide I may just switch to `\xintPrdExpr` which I should have used from the beginning.

1.09a does NOT add the `\xintnum` overhead. 1.09h renames `\xintiPrd` to `\xintiiPrd` to correctly reflect this.

```
737 \def\xintiiPrd {\romannumeral0\xintiiprd }%
738 \def\xintiiprd #1{\xintiiprdexpr #1\relax }%
739 \let\xintPrd\xintiiPrd
740 \let\xintprd\xintiiprd
741 \def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
742 \def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral-`0}%
743 \let\xintPrdExpr\xintiiPrdExpr
744 \let\xintprdexpr\xintiiprdexpr
745 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
746 \def\XINT_prod_loop_a #1\Z #2%
747   {\expandafter\XINT_prod_loop_b \romannumeral-`0#2\Z #1\Z \Z}%
748 \def\XINT_prod_loop_b #1%
749   {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
750 \def\XINT_prod_loop_c
751   {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
752 \def\XINT_prod_finished #1\Z #2\Z \Z { #2}%
```

4.36 `\xintFac`

Modified with 1.02 and again in 1.03 for greater efficiency. I am tempted, here and elsewhere, to use `\ifcase\XINT_Geq {#1}{1000000000}` rather than `\ifnum\xintLength {#1}>9` but for the time being I leave things as they stand. With release 1.05, rather than using `\xintLength` I opt finally for direct use of `\numexpr` (which will throw a suitable number too big message), and to raise the `\xintError`:

`FactorialOfTooBigNumber` for argument larger than 1000000 (rather than 1000000000). With 1.09a, `\xintFac` uses `\xintnum`.

1.09j for no special reason, I lower the maximal number from 999999 to 100000. Any how this computation would need more memory than TL2013 standard allows to TeX. And I don't even mention time...

```

753 \def\xintiFac {\romannumeral0\xintifac }%
754 \def\xintifac #1%
755 {%
756   \expandafter\XINT_fac_fork\expandafter{\the\numexpr #1}%
757 }%
758 \let\xintFac\xintiFac \let\xintfac\xintifac
759 \def\XINT_fac_fork #1%
760 {%
761   \ifcase\XINT_cntSgn #1\Z
762     \xint_afterfi{\expandafter\space\expandafter 1\xint_gobble_i }%
763   \or
764     \expandafter\XINT_fac_checklength
765   \else
766     \xint_afterfi{\expandafter\xintError:FactorialOfNegativeNumber
767                 \expandafter\space\expandafter 1\xint_gobble_i }%
768   \fi
769   {#1}%
770 }%
771 \def\XINT_fac_checklength #1%
772 {%
773   \ifnum #1>100000
774     \xint_afterfi{\expandafter\xintError:FactorialOfTooBigNumber
775                 \expandafter\space\expandafter 1\xint_gobble_i }%
776   \else
777     \xint_afterfi{\ifnum #1>\xint_c_ixixixix
778                 \expandafter\XINT_fac_big_loop
779                 \else
780                 \expandafter\XINT_fac_loop
781                 \fi }%
782   \fi
783   {#1}%
784 }%
785 \def\XINT_fac_big_loop #1{\XINT_fac_big_loop_main {10000}{#1}{}}%
786 \def\XINT_fac_big_loop_main #1#2#3%
787 {%
788   \ifnum #1<#2
789     \expandafter
790     \XINT_fac_big_loop_main
791   \expandafter
792     {\the\numexpr #1+1\expandafter }%

```

#### 4 Package `xint` implementation

```
793 \else
794 \expandafter\XINT_fac_big_docomputation
795 \fi
796 {#2}{#3{#1}}%
797 }%
798 \def\XINT_fac_big_docomputation #1#2%
799 {%
800 \expandafter \XINT_fac_bigcompute_loop \expandafter
801 {\romannumeral0\XINT_fac_loop {9999}}#2\relax
802 }%
803 \def\XINT_fac_bigcompute_loop #1#2%
804 {%
805 \xint_gob_til_relax #2\XINT_fac_bigcompute_end\relax
806 \expandafter\XINT_fac_bigcompute_loop\expandafter
807 {\expandafter\XINT_mul_enter
808 \romannumeral0\XINT_RQ { }#2\R\R\R\R\R\R\R\Z
809 \Z\Z\Z\Z #1\W\W\W\W }%
810 }%
811 \def\XINT_fac_bigcompute_end #1#2#3#4#5%
812 {%
813 \XINT_fac_bigcompute_end_ #5%
814 }%
815 \def\XINT_fac_bigcompute_end_ #1\R #2\Z \W\X\Y\Z #3\W\X\Y\Z { #3}%
816 \def\XINT_fac_loop #1{\XINT_fac_loop_main 1{1000}{#1}}%
817 \def\XINT_fac_loop_main #1#2#3%
818 {%
819 \ifnum #3>#1
820 \else
821 \expandafter\XINT_fac_loop_exit
822 \fi
823 \expandafter\XINT_fac_loop_main\expandafter
824 {\the\numexpr #1+1\expandafter }\expandafter
825 {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z }%
826 {#3}%
827 }%
828 \def\XINT_fac_loop_exit #1#2#3#4#5#6#7%
829 {%
830 \XINT_fac_loop_exit_ #6%
831 }%
832 \def\XINT_fac_loop_exit_ #1#2#3%
833 {%
834 \XINT_mul_M
835 }%
```

-----  
-----  
DECIMAL OPERATIONS: FIRST DIGIT, LASTDIGIT, ODDNESS, MULTIPLICATION BY TEN, QUOTIENT BY TEN, QUOTIENT OR MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.

#### 4.37 `\xintFDg`, `\xintiiFDg`

FIRST DIGIT. Code simplified in 1.05. And prepared for redefinition by `xintfrac` to parse through `\xintNum`. Version 1.09a inserts the `\xintnum` already here.

#### 4 Package *xint* implementation

```
836 \def\xintiFDg {\romannumeral0\xintiifdg }%
837 \def\xintiifdg #1%
838 {%
839   \expandafter\XINT_fdg \romannumeral-\`0#1\W\Z
840 }%
841 \def\xintFDg {\romannumeral0\xintfdg }%
842 \def\xintfdg #1%
843 {%
844   \expandafter\XINT_fdg \romannumeral0\xintnum{#1}\W\Z
845 }%
846 \def\XINT_FDg #1{\romannumeral0\XINT_fdg #1\W\Z }%
847 \def\XINT_fdg #1#2#3\Z
848 {%
849   \xint_UDzerominusfork
850   #1-{\`0}% zero
851   0#1{ #2}% negative
852   0-{\`#1}% positive
853   \krof
854 }%
```

#### 4.38 `\xintLDg`, `\xintiLDg`

LAST DIGIT. Simplified in 1.05. And prepared for extension by `xintfrac` to parse through `\xintNum`. Release 1.09a adds the `\xintnum` already here, and this propagates to `\xintOdd`, etc... 1.09e The `\xintiLDg` is for defining `\xintiOdd` which is used once (currently) elsewhere .

```
855 \def\xintiLDg {\romannumeral0\xintiildg }%
856 \def\xintiildg #1%
857 {%
858   \expandafter\XINT_ldg\expandafter {\romannumeral-\`0#1}%
859 }%
860 \def\xintLDg {\romannumeral0\xintldg }%
861 \def\xintldg #1%
862 {%
863   \expandafter\XINT_ldg\expandafter {\romannumeral0\xintnum{#1}}%
864 }%
865 \def\XINT_LDg #1{\romannumeral0\XINT_ldg {#1}}%
866 \def\XINT_ldg #1%
867 {%
868   \expandafter\XINT_ldg_\romannumeral0\xintreverseorder {#1}\Z
869 }%
870 \def\XINT_ldg_ #1#2\Z{ #1}%
```

#### 4.39 `\xintMON`, `\xintMMON`, `\xintiMON`, `\xintiMMON`

MINUS ONE TO THE POWER N and  $(-1)^{N-1}$

```
871 \def\xintiMON {\romannumeral0\xintiimon }%
872 \def\xintiimon #1%
873 {%
874   \ifodd\xintiLDg {#1}
875     \xint_afterfi{-1}%
876   \else
```

#### 4 Package `xint` implementation

```
877     \xint_afterfi{ 1}%
878   \fi
879 }%
880 \def\xintiMMON {\romannumeral0\xintiimmon }%
881 \def\xintiimmon #1%
882 {%
883   \ifodd\xintiLDg {#1}
884     \xint_afterfi{ 1}%
885   \else
886     \xint_afterfi{ -1}%
887   \fi
888 }%
889 \def\xintMON {\romannumeral0\xintmon }%
890 \def\xintmon #1%
891 {%
892   \ifodd\xintLDg {#1}
893     \xint_afterfi{ -1}%
894   \else
895     \xint_afterfi{ 1}%
896   \fi
897 }%
898 \def\xintMMON {\romannumeral0\xintmmon }%
899 \def\xintmmon #1%
900 {%
901   \ifodd\xintLDg {#1}
902     \xint_afterfi{ 1}%
903   \else
904     \xint_afterfi{ -1}%
905   \fi
906 }%
```

#### 4.40 `\xintOdd`, `\xintiOdd`, `\xintEven`, `\xintiEven`

1.05 has `\xintiOdd`, whereas `\xintOdd` parses through `\xintNum`. Inadvertently, 1.09a redefined `\xintiLDg` hence `\xintiOdd` also parsed through `\xintNum`. Anyway, having a `\xintOdd` and a `\xintiOdd` was silly. Removed in 1.09f, now only `\xintOdd` and `\xintiOdd`. 1.1: `\xintEven` and `\xintiEven` added for `\xintiexpr`.

```
907 \def\xintiOdd {\romannumeral0\xintiiodd }%
908 \def\xintiiodd #1%
909 {%
910   \ifodd\xintiLDg{#1}
911     \xint_afterfi{ 1}%
912   \else
913     \xint_afterfi{ 0}%
914   \fi
915 }%
916 \def\xintiEven {\romannumeral0\xintiieven }%
917 \def\xintiieven #1%
918 {%
919   \ifodd\xintiLDg{#1}
920     \xint_afterfi{ 0}%
921   \else
```



#### 4 Package *xint* implementation

```
922     \xint_afterfi{ 1}%
923   \fi
924 }%
925 \def\xintOdd {\romannumeral0\xintodd }%
926 \def\xintodd #1%
927 {%
928   \ifodd\xintLDg{#1}
929     \xint_afterfi{ 1}%
930   \else
931     \xint_afterfi{ 0}%
932   \fi
933 }%
934 \def\xintEven {\romannumeral0\xinteven }%
935 \def\xinteven #1%
936 {%
937   \ifodd\xintLDg{#1}
938     \xint_afterfi{ 0}%
939   \else
940     \xint_afterfi{ 1}%
941   \fi
942 }%
```

#### 4.41 *\xintDSL*

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```
943 \def\xintDSL {\romannumeral0\xintdsl }%
944 \def\xintdsl #1%
945 {%
946   \expandafter\XINT_dsl \romannumeral-`0#1\Z
947 }%
948 \def\XINT_DSL #1{\romannumeral0\XINT_dsl #1\Z }%
949 \def\XINT_dsl #1%
950 {%
951   \xint_gob_til_zero #1\xint_dsl_zero 0\XINT_dsl_ #1%
952 }%
953 \def\xint_dsl_zero 0\XINT_dsl_ 0#1\Z { 0}%
954 \def\XINT_dsl_ #1\Z { #10}%
```

#### 4.42 *\xintDSR*

DECIMAL SHIFT RIGHT (=DIVISION PAR 10). Release 1.06b which replaced all @'s by underscores left undefined the *\xint\_minus* used in *\XINT\_dsr\_b*, and this bug was fixed only later in release 1.09b

```
955 \def\xintDSR {\romannumeral0\xintdsr }%
956 \def\xintdsr #1%
957 {%
958   \expandafter\XINT_dsr_a\expandafter {\romannumeral-`0#1}\W\Z
959 }%
960 \def\XINT_DSR #1{\romannumeral0\XINT_dsr_a {#1}\W\Z }%
961 \def\XINT_dsr_a
962 {%
963   \expandafter\XINT_dsr_b\romannumeral0\xintreverseorder
```

#### 4 Package *xint* implementation

```
964 }%
965 \def\XINT_dsr_b #1#2#3\Z
966 {%
967   \xint_gob_til_W #2\xint_dsr_onedigit\W
968   \xint_gob_til_minus #2\xint_dsr_onedigit-%
969   \expandafter\XINT_dsr_removev
970   \romannumeral0\xintreverseorder {#2#3}%
971 }%
972 \def\xint_dsr_onedigit #1\xintreverseorder #2{ 0}%
973 \def\XINT_dsr_removev #1\W { }%
```

#### 4.43 *\xintDSH*, *\xintDSHr*

DECIMAL SHIFTS *\xintDSH* {x}{A}

si  $x \leq 0$ , fait  $A \rightarrow A \cdot 10^{|x|}$ . v1.03 corrige l'oversight pour  $A=0$ .

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).

Release 1.06 now feeds  $x$  to a *\numexpr* first. I will have to revise this code at some point.

```
974 \def\xintDSHr {\romannumeral0\xintdshr }%
975 \def\xintdshr #1%
976 {%
977   \expandafter\XINT_dshr_checkxpositive \the\numexpr #1\relax\Z
978 }%
979 \def\XINT_dshr_checkxpositive #1%
980 {%
981   \xint_UDzerominusfork
982   0#1\XINT_dshr_xzeroorneg
983   #1-\XINT_dshr_xzeroorneg
984   0-\XINT_dshr_xpositive
985   \krof #1%
986 }%
987 \def\XINT_dshr_xzeroorneg #1\Z #2{ 0}%
988 \def\XINT_dshr_xpositive #1\Z
989 {%
990   \expandafter\xint_secondoftwo_thenstop\romannumeral0\xintdsx {#1}%
991 }%
992 \def\xintDSH {\romannumeral0\xintdsh }%
993 \def\xintdsh #1#2%
994 {%
995   \expandafter\xint_dsh\expandafter {\romannumeral-`0#2}{#1}%
996 }%
997 \def\xint_dsh #1#2%
998 {%
999   \expandafter\XINT_dsh_checksignx \the\numexpr #2\relax\Z {#1}%
1000 }%
1001 \def\XINT_dsh_checksignx #1%
1002 {%
1003   \xint_UDzerominusfork
1004   #1-\XINT_dsh_xiszero
1005   0#1\XINT_dsx_xisNeg_checkA      % on passe direct dans DSx
```

## 4 Package `xint` implementation

```
1006      0-{\XINT_dsh_xisPos #1}%
1007      \krof
1008 }%
1009 \def\XINT_dsh_xiszero #1\Z #2{ #2}%
1010 \def\XINT_dsh_xisPos #1\Z #2%
1011 {%
1012      \expandafter\xint_firstoftwo_thenstop
1013      \romannumeral0\XINT_dsx_checksiga #2\Z {#1}% via DSx
1014 }%
```

### 4.44 `\xintDSx`

Je fais cette routine pour la version 1.01, après modification de `\xintDecSplit`. Dorénavant `\xintDSx` fera appel à `\xintDecSplit` et de même `\xintDSH` fera appel à `\xintDSx`. J'ai donc supprimé entièrement l'ancien code de `\xintDSH` et re-écrit entièrement celui de `\xintDecSplit` pour `x` positif.

--> 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  $10^x Q \pmod R$  où il faut prendre le signe plus si  $Q$  est positif ou nul et le signe moins si  $Q$  est strictement négatif.

Release 1.06 has a faster and more compactly coded `\XINT_dsx_zeroloop`. Also, `x` is now given to a `\numexpr`. The earlier code should be then simplified, but I leave as is for the time being.

Release 1.07 modified the coding of `\XINT_dsx_zeroloop`, to avoid impacting the input stack. Indeed the truncating, rounding, and conversion to float routines all use internally `\XINT_dsx_zeroloop` (via `\XINT_dsx_addzerosnofuss`), and they were thus roughly limited to generating  $N = 8$  times the input save stack size digits. On TL2012 and TL2013, this means  $40000 = 8 \times 5000$  digits. Although generating more than 40000 digits is more like a one shot thing, I wanted to open the possibility of outputting tens of thousands of digits to fail, thus I re-organized `\XINT_dsx_zeroloop`.

January 5, 2014: but it is only with the new division implementation of 1.09j and also with its special `\xintXTrunc` routine that the possibility mentioned in the last paragraph has become a concrete one in terms of computation time.

```
1015 \def\xintDSx {\romannumeral0\xintdsx }%
1016 \def\xintdsx #1#2%
1017 {%
1018      \expandafter\xint_dsx\expandafter {\romannumeral-`0#2}{#1}%
1019 }%
1020 \def\xint_dsx #1#2%
1021 {%
1022      \expandafter\XINT_dsx_checksignx \the\numexpr #2\relax\Z {#1}%
1023 }%
1024 \def\XINT_DSx #1#2{\romannumeral0\XINT_dsx_checksignx #1\Z {#2}}%
1025 \def\XINT_dsx #1#2{\XINT_dsx_checksignx #1\Z {#2}}%
1026 \def\XINT_dsx_checksignx #1%
1027 {%
1028      \xint_UDzerominusfork
1029      #1-\XINT_dsx_xisZero
1030      0#1\XINT_dsx_xisNeg_checkA
1031      0-{\XINT_dsx_xisPos #1}%
```

#### 4 Package *xint* implementation

```

1032 \krof
1033 }%
1034 \def\XINT_dsx_xisZero #1\Z #2{ #2{0}}% attention comme x > 0
1035 \def\XINT_dsx_xisNeg_checkA #1\Z #2%
1036 {%
1037 \XINT_dsx_xisNeg_checkA_ #2\Z {#1}%
1038 }%
1039 \def\XINT_dsx_xisNeg_checkA_ #1#2\Z #3%
1040 {%
1041 \xint_gob_til_zero #1\XINT_dsx_xisNeg_Azero 0%
1042 \XINT_dsx_xisNeg_checkx {#3}{#3}{}\Z {#1#2}%
1043 }%
1044 \def\XINT_dsx_xisNeg_Azero #1\Z #2{ 0}%
1045 \def\XINT_dsx_xisNeg_checkx #1%
1046 {%
1047 \ifnum #1>1000000
1048 \xint_afterfi
1049 {\xintError:TooBigDecimalShift
1050 \expandafter\space\expandafter 0\xint_gobble_iv }%
1051 \else
1052 \expandafter \XINT_dsx_zeroloop
1053 \fi
1054 }%
1055 \def\XINT_dsx_addzerosnofuss #1{\XINT_dsx_zeroloop {#1}{}\Z }%
1056 \def\XINT_dsx_zeroloop #1#2%
1057 {%
1058 \ifnum #1<\xint_c_ix \XINT_dsx_exita\fi
1059 \expandafter\XINT_dsx_zeroloop\expandafter
1060 {\the\numexpr #1-\xint_c_viii}{#200000000}%
1061 }%
1062 \def\XINT_dsx_exita\fi\expandafter\XINT_dsx_zeroloop
1063 {%
1064 \fi\expandafter\XINT_dsx_exitb
1065 }%
1066 \def\XINT_dsx_exitb #1#2%
1067 {%
1068 \expandafter\expandafter\expandafter
1069 \XINT_dsx_addzeros\csname xint_gobble_\romannumeral -#1\endcsname #2%
1070 }%
1071 \def\XINT_dsx_addzeros #1\Z #2{ #2#1}%
1072 \def\XINT_dsx_xisPos #1\Z #2%
1073 {%
1074 \XINT_dsx_checksignA #2\Z {#1}%
1075 }%
1076 \def\XINT_dsx_checksignA #1%
1077 {%
1078 \xint_UDzerominusfork
1079 #1-\XINT_dsx_AisZero
1080 0#1\XINT_dsx_AisNeg
1081 0-{\XINT_dsx_AisPos #1}%
1082 \krof
1083 }%

```

#### 4 Package *xint* implementation

```
1084 \def\XINT_dsx_AisZero #1\Z #2{ {0}{0}}%
1085 \def\XINT_dsx_AisNeg #1\Z #2%
1086 {%
1087   \expandafter\XINT_dsx_AisNeg_dosplit_andcheckfirst
1088   \romannumeral0\XINT_split_checksizex {#2}{#1}%
1089 }%
1090 \def\XINT_dsx_AisNeg_dosplit_andcheckfirst #1%
1091 {%
1092   \XINT_dsx_AisNeg_checkiffirstempty #1\Z
1093 }%
1094 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
1095 {%
1096   \xint_gob_til_Z #1\XINT_dsx_AisNeg_finish_zero\Z
1097   \XINT_dsx_AisNeg_finish_notzero #1%
1098 }%
1099 \def\XINT_dsx_AisNeg_finish_zero\Z
1100   \XINT_dsx_AisNeg_finish_notzero\Z #1%
1101 {%
1102   \expandafter\XINT_dsx_end
1103   \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
1104 }%
1105 \def\XINT_dsx_AisNeg_finish_notzero #1\Z #2%
1106 {%
1107   \expandafter\XINT_dsx_end
1108   \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
1109 }%
1110 \def\XINT_dsx_AisPos #1\Z #2%
1111 {%
1112   \expandafter\XINT_dsx_AisPos_finish
1113   \romannumeral0\XINT_split_checksizex {#2}{#1}%
1114 }%
1115 \def\XINT_dsx_AisPos_finish #1#2%
1116 {%
1117   \expandafter\XINT_dsx_end
1118   \expandafter {\romannumeral0\XINT_num {#2}}%
1119   {\romannumeral0\XINT_num {#1}}%
1120 }%
1121 \edef\XINT_dsx_end #1#2%
1122 {%
1123   \noexpand\expandafter\space\noexpand\expandafter{#2}{#1}%
1124 }%
```

#### 4.45 `\xintDecSplit`, `\xintDecSplitL`, `\xintDecSplitR`

##### DECIMAL SPLIT

The macro `\xintDecSplit {x}{A}` first replaces A with |A| (\*) This macro cuts the number into two pieces L and R. The concatenation LR always reproduces |A|, and R may be empty or have leading zeros. 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.

(\*) warning: this may change in a future version. Only the behavior for A non-negative is guaranteed to remain the same.

#### 4 Package *xint* implementation

v1.05a: `\XINT_split_checksizex` does not compute the length anymore, rather the error will be from a `\numexpr`; but the limit of 999999999 does not make much sense.

v1.06: Improvements in `\XINT_split_fromleft_loop`, `\XINT_split_fromright_loop` and related macros. More readable coding, speed gains. Also, I now feed immediately a `\numexpr` with `x`. Some simplifications should probably be made to the code, which is kept as is for the time being.

1.09e pays attention to the use of `xintiabs` which acquired in 1.09a the `xintnum` overhead. So `xintiabs` rather without that overhead.

```
1125 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
1126 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
1127 \def\xintdecsplitl
1128 {%
1129   \expandafter\xint_firstoftwo_thenstop
1130   \romannumeral0\xintdecsplit
1131 }%
1132 \def\xintdecsplitr
1133 {%
1134   \expandafter\xint_secondoftwo_thenstop
1135   \romannumeral0\xintdecsplit
1136 }%
1137 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
1138 \def\xintdecsplit #1#2%
1139 {%
1140   \expandafter \xint_split \expandafter
1141   {\romannumeral0\xintiabs {#2}}{#1}% fait expansion de A
1142 }%
1143 \def\xint_split #1#2%
1144 {%
1145   \expandafter\XINT_split_checksizex\expandafter{\the\numexpr #2}{#1}%
1146 }%
1147 \def\XINT_split_checksizex #1% 999999999 is anyhow very big, could be reduced
1148 {%
1149   \ifnum\numexpr\XINT_Abs{#1}>999999999
1150     \xint_afterfi {\xintError:TooBigDecimalSplit\XINT_split_bigx }%
1151   \else
1152     \expandafter\XINT_split_xfork
1153   \fi
1154   #1\Z
1155 }%
1156 \def\XINT_split_bigx #1\Z #2%
1157 {%
1158   \ifcase\XINT_cntSgn #1\Z
1159   \or \xint_afterfi { }{#2}% positive big x
1160   \else
1161     \xint_afterfi { }{#2}% negative big x
1162   \fi
1163 }%
1164 \def\XINT_split_xfork #1%
1165 {%
1166   \xint_UDzerominusfork
1167   #1-\XINT_split_zerosplit
1168   0#1\XINT_split_fromleft
1169   0-\XINT_split_fromright #1}%
```

#### 4 Package `xint` implementation

```

1170 \krof
1171 }%
1172 \def\XINT_split_zerosplit #1\Z #2{ {#2}}}%
1173 \def\XINT_split_fromleft #1\Z #2%
1174 {%
1175 \XINT_split_fromleft_loop {#1}{#2\W\W\W\W\W\W\W\W\Z
1176 }%
1177 \def\XINT_split_fromleft_loop #1%
1178 {%
1179 \ifnum #1<\xint_c_viii\XINT_split_fromleft_exita\fi
1180 \expandafter\XINT_split_fromleft_loop_perhaps\expandafter
1181 {\the\numexpr #1-\xint_c_viii\expandafter}\XINT_split_fromleft_eight
1182 }%
1183 \def\XINT_split_fromleft_eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
1184 \def\XINT_split_fromleft_loop_perhaps #1#2%
1185 {%
1186 \xint_gob_til_W #2\XINT_split_fromleft_toofar\W
1187 \XINT_split_fromleft_loop {#1}%
1188 }%
1189 \def\XINT_split_fromleft_toofar\W\XINT_split_fromleft_loop #1#2#3\Z
1190 {%
1191 \XINT_split_fromleft_toofar_b #2\Z
1192 }%
1193 \def\XINT_split_fromleft_toofar_b #1\W #2\Z { {#1}}}%
1194 \def\XINT_split_fromleft_exita\fi
1195 \expandafter\XINT_split_fromleft_loop_perhaps\expandafter #1#2%
1196 {\fi \XINT_split_fromleft_exitb #1}%
1197 \def\XINT_split_fromleft_exitb\the\numexpr #1-\xint_c_viii\expandafter
1198 {%
1199 \csname XINT_split_fromleft_endsplit_\romannumeral #1\endcsname
1200 }%
1201 \def\XINT_split_fromleft_endsplit_ #1#2\W #3\Z { {#1}{#2}}}%
1202 \def\XINT_split_fromleft_endsplit_i #1#2%
1203 {\XINT_split_fromleft_checkiftoofar #2{#1#2}}%
1204 \def\XINT_split_fromleft_endsplit_ii #1#2#3%
1205 {\XINT_split_fromleft_checkiftoofar #3{#1#2#3}}%
1206 \def\XINT_split_fromleft_endsplit_iii #1#2#3#4%
1207 {\XINT_split_fromleft_checkiftoofar #4{#1#2#3#4}}%
1208 \def\XINT_split_fromleft_endsplit_iv #1#2#3#4#5%
1209 {\XINT_split_fromleft_checkiftoofar #5{#1#2#3#4#5}}%
1210 \def\XINT_split_fromleft_endsplit_v #1#2#3#4#5#6%
1211 {\XINT_split_fromleft_checkiftoofar #6{#1#2#3#4#5#6}}%
1212 \def\XINT_split_fromleft_endsplit_vi #1#2#3#4#5#6#7%
1213 {\XINT_split_fromleft_checkiftoofar #7{#1#2#3#4#5#6#7}}%
1214 \def\XINT_split_fromleft_endsplit_vii #1#2#3#4#5#6#7#8%
1215 {\XINT_split_fromleft_checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
1216 \def\XINT_split_fromleft_checkiftoofar #1#2#3\W #4\Z
1217 {%
1218 \xint_gob_til_W #1\XINT_split_fromleft_wenttoofar\W
1219 \space {#2}{#3}%
1220 }%
1221 \def\XINT_split_fromleft_wenttoofar\W\space #1%

```

#### 4 Package *xint* implementation

```

1222 {%
1223   \XINT_split_fromleft_wenttoofar_b #1\Z
1224 }%
1225 \def\XINT_split_fromleft_wenttoofar_b #1\W #2\Z { {#1}}%
1226 \def\XINT_split_fromright #1\Z #2%
1227 {%
1228   \expandafter \XINT_split_fromright_a \expandafter
1229   {\romannumeral0\xintreverseorder {#2}}{#1}{#2}%
1230 }%
1231 \def\XINT_split_fromright_a #1#2%
1232 {%
1233   \XINT_split_fromright_loop {#2}{#1\W\W\W\W\W\W\W\W\Z
1234 }%
1235 \def\XINT_split_fromright_loop #1%
1236 {%
1237   \ifnum #1<\xint_c_viii\XINT_split_fromright_exita\fi
1238   \expandafter\XINT_split_fromright_loop_perhaps\expandafter
1239   {\the\numexpr #1-\xint_c_viii\expandafter }\XINT_split_fromright_eight
1240 }%
1241 \def\XINT_split_fromright_eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
1242 \def\XINT_split_fromright_loop_perhaps #1#2%
1243 {%
1244   \xint_gob_til_W #2\XINT_split_fromright_toofar\W
1245   \XINT_split_fromright_loop {#1}%
1246 }%
1247 \def\XINT_split_fromright_toofar\W\XINT_split_fromright_loop #1#2#3\Z { {}}%
1248 \def\XINT_split_fromright_exita\fi
1249   \expandafter\XINT_split_fromright_loop_perhaps\expandafter #1#2%
1250   {\fi \XINT_split_fromright_exitb #1}%
1251 \def\XINT_split_fromright_exitb\the\numexpr #1-\xint_c_viii\expandafter
1252 {%
1253   \csname XINT_split_fromright_endsplit_\romannumeral #1\endcsname
1254 }%
1255 \edef\XINT_split_fromright_endsplit_ #1#2\W #3\Z #4%
1256 {%
1257   \noexpand\expandafter\space\noexpand\expandafter
1258   {\noexpand\romannumeral0\noexpand\xintreverseorder {#2}}{#1}%
1259 }%
1260 \def\XINT_split_fromright_endsplit_i #1#2%
1261   {\XINT_split_fromright_checkiftoofar #2{#2#1}}%
1262 \def\XINT_split_fromright_endsplit_ii #1#2#3%
1263   {\XINT_split_fromright_checkiftoofar #3{#3#2#1}}%
1264 \def\XINT_split_fromright_endsplit_iii #1#2#3#4%
1265   {\XINT_split_fromright_checkiftoofar #4{#4#3#2#1}}%
1266 \def\XINT_split_fromright_endsplit_iv #1#2#3#4#5%
1267   {\XINT_split_fromright_checkiftoofar #5{#5#4#3#2#1}}%
1268 \def\XINT_split_fromright_endsplit_v #1#2#3#4#5#6%
1269   {\XINT_split_fromright_checkiftoofar #6{#6#5#4#3#2#1}}%
1270 \def\XINT_split_fromright_endsplit_vi #1#2#3#4#5#6#7%
1271   {\XINT_split_fromright_checkiftoofar #7{#7#6#5#4#3#2#1}}%
1272 \def\XINT_split_fromright_endsplit_vii #1#2#3#4#5#6#7#8%
1273   {\XINT_split_fromright_checkiftoofar #8{#8#7#6#5#4#3#2#1}}%

```



#### 4 Package `xint` implementation

```
1274 \def\XINT_split_fromright_checkiftoofar #1%
1275 {%
1276   \xint_gob_til_W #1\XINT_split_fromright_wenttoofar\W
1277   \XINT_split_fromright_endsplit_
1278 }%
1279 \def\XINT_split_fromright_wenttoofar\W\XINT_split_fromright_endsplit_ #1\Z #2%
1280   { {}{#2}}%
```

### 4.46 `\xintDouble`

v1.08

```
1281 \def\xintDouble {\romannumeral0\xintdouble }%
1282 \def\xintdouble #1%
1283 {%
1284   \expandafter\XINT_dbl\romannumeral-`0#1%
1285   \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W
1286 }%
1287 \def\XINT_dbl #1%
1288 {%
1289   \xint_UDzerominusfork
1290   #1-\XINT_dbl_zero
1291   0#1\XINT_dbl_neg
1292   0-{\XINT_dbl_pos #1}%
1293   \krof
1294 }%
1295 \def\XINT_dbl_zero #1\Z \W\W\W\W\W\W\W { 0}%
1296 \def\XINT_dbl_neg
1297   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dbl_pos }%
1298 \def\XINT_dbl_pos
1299 {%
1300   \expandafter\XINT_dbl_a \expandafter{\expandafter}\expandafter 0%
1301   \romannumeral0\XINT_SQ }%
1302 }%
1303 \def\XINT_dbl_a #1#2#3#4#5#6#7#8#9%
1304 {%
1305   \xint_gob_til_W #9\XINT_dbl_end_a\W
1306   \expandafter\XINT_dbl_b
1307   \the\numexpr \xint_c_x^viii+#2+\xint_c_ii*#9#8#7#6#5#4#3\relax {#1}%
1308 }%
1309 \def\XINT_dbl_b 1#1#2#3#4#5#6#7#8#9%
1310 {%
1311   \XINT_dbl_a {#2#3#4#5#6#7#8#9}{#1}%
1312 }%
1313 \def\XINT_dbl_end_a #1+#2+#3\relax #4%
1314 {%
1315   \expandafter\XINT_dbl_end_b #2#4%
1316 }%
1317 \edef\XINT_dbl_end_b #1#2#3#4#5#6#7#8%
1318 {%
1319   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
1320 }%
```

4.47 `\xintHalf`

v1.08

```

1321 \def\xintHalf {\romannumeral0\xinthalf }%
1322 \def\xinthalf #1%
1323 {%
1324     \expandafter\XINT_half\romannumeral-`0#1%
1325     \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W
1326 }%
1327 \def\XINT_half #1%
1328 {%
1329     \xint_UDzerominusfork
1330     #1-\XINT_half_zero
1331     0#1\XINT_half_neg
1332     0-{\XINT_half_pos #1}%
1333     \krof
1334 }%
1335 \def\XINT_half_zero #1\Z \W\W\W\W\W\W\W { 0}%
1336 \def\XINT_half_neg {\expandafter\XINT_opp\romannumeral0\XINT_half_pos }%
1337 \def\XINT_half_pos {\expandafter\XINT_half_a\romannumeral0\XINT_SQ {}}%
1338 \def\XINT_half_a #1#2#3#4#5#6#7#8%
1339 {%
1340     \xint_gob_til_W #8\XINT_half_dont\W
1341     \expandafter\XINT_half_b
1342     \the\numexpr \xint_c_x^viii+\xint_c_v*#7#6#5#4#3#2#1\relax #8%
1343 }%
1344 \edef\XINT_half_dont\W\expandafter\XINT_half_b
1345     \the\numexpr \xint_c_x^viii+\xint_c_v*#1#2#3#4#5#6#7\relax \W\W\W\W\W\W\W
1346 {%
1347     \noexpand\expandafter\space
1348     \noexpand\the\numexpr (#1#2#3#4#5#6#7+\xint_c_i)/\xint_c_ii-\xint_c_i \relax
1349 }%
1350 \def\XINT_half_b 1#1#2#3#4#5#6#7#8%
1351 {%
1352     \XINT_half_c {#2#3#4#5#6#7}{#1}%
1353 }%
1354 \def\XINT_half_c #1#2#3#4#5#6#7#8#9%
1355 {%
1356     \xint_gob_til_W #3\XINT_half_end_a #2\W
1357     \expandafter\XINT_half_d
1358     \the\numexpr \xint_c_x^viii+\xint_c_v*#9#8#7#6#5#4#3+#2\relax {#1}%
1359 }%
1360 \def\XINT_half_d 1#1#2#3#4#5#6#7#8#9%
1361 {%
1362     \XINT_half_c {#2#3#4#5#6#7#8#9}{#1}%
1363 }%
1364 \def\XINT_half_end_a #1\W #2\relax #3%
1365 {%
1366     \xint_gob_til_zero #1\XINT_half_end_b 0\space #1#3%
1367 }%
1368 \edef\XINT_half_end_b 0\space 0#1#2#3#4#5#6#7%
1369 {%

```

```
1370 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7\relax
1371 }%
```

#### 4.48 `\xintiiSqrt`, `\xintiiSqrtR`, `\xintiiSquareRoot`

v1.08. 1.09a uses `\xintnum`.

Some overhead was added inadvertently in 1.09a to inner routines when `\xintiquo` and `\xintidivision` were also promoted to use `\xintnum`; release 1.09f thus uses `\xintiiquo` and `\xintiidivision` which avoid this `\xintnum` overhead.

1.09j replaced the previous long `\ifcase` from `\XINT_sqrt_c` by some nested `\ifnum`'s.

1.1 Ajout de `\xintiiSqrt` et `\xintiiSquareRoot`.

1.1a ajoute `\xintiiSqrtR`, which provides the rounded, not truncated square root.

```
1372 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
1373 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
1374 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
1375 \def\xintiSqrt {\romannumeral0\xintisqrt }%
1376 \def\xintiSquareRoot {\romannumeral0\xintisquareroot }%
1377 \def\xintisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot }%
1378 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1379 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
1380 \def\XINT_sqrt_post #1#2{\XINT_dec_pos #1\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }%
```

$N = (\#1)^2 - \#2$  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-> $\#1 - 1$ .

```
1381 \def\XINT_sqrtr_post #1#2{\xintiiiflt {#2}{#1}% Lt <-> a<b
1382 { #1}{\XINT_dec_pos #1\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }%
1383 \def\xintisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\Z }%
1384 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral-`0#1\Z }%
1385 \def\XINT_sqrt_checkin #1%
1386 {%
1387 \xint_UDzerominusfork
1388 #1-\XINT_sqrt_iszero
1389 0#1\XINT_sqrt_isneg
1390 0-{\XINT_sqrt #1}%
1391 \krof
1392 }%
1393 \def\XINT_sqrt_iszero #1\Z { 11}%
1394 \edef\XINT_sqrt_isneg #1\Z {\noexpand\xintError:RootOfNegative\space 11}%
1395 \def\XINT_sqrt #1\Z
1396 {%
1397 \expandafter\XINT_sqrt_start\expandafter {\romannumeral0\xintlength {#1}}{#1}%
1398 }%
1399 \def\XINT_sqrt_start #1%
1400 {%
1401 \ifnum #1<\xint_c_x
1402 \expandafter\XINT_sqrt_small_a
1403 \else
1404 \expandafter\XINT_sqrt_big_a
1405 \fi
1406 {#1}%
```

#### 4 Package *xint* implementation

```

1407 }%
1408 \def\XINT_sqrt_small_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_small_d }%
1409 \def\XINT_sqrt_big_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_big_d }%
1410 \def\XINT_sqrt_a #1%
1411 {%
1412   \ifodd #1
1413     \expandafter\XINT_sqrt_bB
1414   \else
1415     \expandafter\XINT_sqrt_bA
1416   \fi
1417   {#1}%
1418 }%
1419 \def\XINT_sqrt_bA #1#2#3%
1420 {%
1421   \XINT_sqrt_bA_b #3\Z #2{#1}{#3}%
1422 }%
1423 \def\XINT_sqrt_bA_b #1#2#3\Z
1424 {%
1425   \XINT_sqrt_c {#1#2}%
1426 }%
1427 \def\XINT_sqrt_bB #1#2#3%
1428 {%
1429   \XINT_sqrt_bB_b #3\Z #2{#1}{#3}%
1430 }%
1431 \def\XINT_sqrt_bB_b #1#2\Z
1432 {%
1433   \XINT_sqrt_c #1%
1434 }%
1435 \def\XINT_sqrt_c #1#2%
1436 {%
1437   \expandafter #2\expandafter
1438   {\the\numexpr\ifnum #1>\xint_c_iii
1439     \ifnum #1>\xint_c_viii
1440     \ifnum #1>15 \ifnum #1>24 \ifnum #1>35
1441     \ifnum #1>48 \ifnum #1>63 \ifnum #1>80
1442     10\else 9\fi \else 8\fi \else 7\fi \else 6\fi
1443     \else 5\fi \else 4\fi \else 3\fi \else 2\fi \relax }%
1444 }%
1445 \def\XINT_sqrt_small_d #1#2%
1446 {%
1447   \expandafter\XINT_sqrt_small_e\expandafter
1448   {\the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1449     \or 0\or 00\or 000\or 0000\fi }%
1450 }%
1451 \def\XINT_sqrt_small_e #1#2%
1452 {%
1453   \expandafter\XINT_sqrt_small_f\expandafter {\the\numexpr #1*#1-#2}{#1}%
1454 }%
1455 \def\XINT_sqrt_small_f #1#2%
1456 {%
1457   \expandafter\XINT_sqrt_small_g\expandafter
1458   {\the\numexpr ((#1+#2)/(\xint_c_ii*#2))-\xint_c_i}{#1}{#2}%

```

#### 4 Package *xint* implementation

```

1459 }%
1460 \def\XINT_sqrt_small_g #1%
1461 {%
1462   \ifnum #1>\xint_c_
1463     \expandafter\XINT_sqrt_small_h
1464   \else
1465     \expandafter\XINT_sqrt_small_end
1466   \fi
1467   {#1}%
1468 }%
1469 \def\XINT_sqrt_small_h #1#2#3%
1470 {%
1471   \expandafter\XINT_sqrt_small_f\expandafter
1472   {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
1473   {\the\numexpr #3-#1}%
1474 }%
1475 \def\XINT_sqrt_small_end #1#2#3{ {#3}{#2}}%
1476 \def\XINT_sqrt_big_d #1#2%
1477 {%
1478   \ifodd #2
1479     \expandafter\expandafter\expandafter\XINT_sqrt_big_eB
1480   \else
1481     \expandafter\expandafter\expandafter\XINT_sqrt_big_eA
1482   \fi
1483   \expandafter {\the\numexpr #2/\xint_c_ii }{#1}%
1484 }%
1485 \def\XINT_sqrt_big_eA #1#2#3%
1486 {%
1487   \XINT_sqrt_big_eA_a #3\Z {#2}{#1}{#3}%
1488 }%
1489 \def\XINT_sqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
1490 {%
1491   \XINT_sqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
1492 }%
1493 \def\XINT_sqrt_big_eA_b #1#2%
1494 {%
1495   \expandafter\XINT_sqrt_big_f
1496   \romannumeral0\XINT_sqrt_small_e {#2000}{#1}{#1}%
1497 }%
1498 \def\XINT_sqrt_big_eB #1#2#3%
1499 {%
1500   \XINT_sqrt_big_eB_a #3\Z {#2}{#1}{#3}%
1501 }%
1502 \def\XINT_sqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
1503 {%
1504   \XINT_sqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
1505 }%
1506 \def\XINT_sqrt_big_eB_b #1#2\Z #3%
1507 {%
1508   \expandafter\XINT_sqrt_big_f
1509   \romannumeral0\XINT_sqrt_small_e {#30000}{#1}{#1}%
1510 }%

```

#### 4 Package `xint` implementation

```

1511 \def\XINT_sqrt_big_f #1#2#3#4%
1512 {%
1513   \expandafter\XINT_sqrt_big_f_a\expandafter
1514   {\the\numexpr #2+#3\expandafter}\expandafter
1515   {\romannumeral0\XINT_dsx_addzerosnofuss
1516     {\numexpr #4-\xint_c_iv\relax}{#1}}{#4}%
1517 }%
1518 \def\XINT_sqrt_big_f_a #1#2#3#4%
1519 {%
1520   \expandafter\XINT_sqrt_big_g\expandafter
1521   {\romannumeral0\xintiisub
1522     {\XINT_dsx_addzerosnofuss
1523       {\numexpr \xint_c_ii*#3-\xint_c_viii\relax}{#1}}{#4}}%
1524   {#2}{#3}%
1525 }%
1526 \def\XINT_sqrt_big_g #1#2%
1527 {%
1528   \expandafter\XINT_sqrt_big_j
1529   \romannumeral0\xintiidivision{#1}%
1530   {\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\R\Z \W\W\W\W\W\W }{#2}%
1531 }%
1532 \def\XINT_sqrt_big_j #1%
1533 {%
1534   \if0\XINT_Sgn #1\Z
1535     \expandafter \XINT_sqrt_big_end
1536   \else \expandafter \XINT_sqrt_big_k
1537   \fi {#1}%
1538 }%
1539 \def\XINT_sqrt_big_k #1#2#3%
1540 {%
1541   \expandafter\XINT_sqrt_big_l\expandafter
1542   {\romannumeral0\xintiisub {#3}{#1}}%
1543   {\romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}}%
1544 }%
1545 \def\XINT_sqrt_big_l #1#2%
1546 {%
1547   \expandafter\XINT_sqrt_big_g\expandafter
1548   {#2}{#1}%
1549 }%
1550 \def\XINT_sqrt_big_end #1#2#3#4{ {#3}{#2}}%

```

#### 4.49 `\xintiiE`

Originally was used in `\xintiiexpr`. Transferred from `xintfrac` for 1.1.

```

1551 \def\xintiiE {\romannumeral0\xintiiE }% used in \xintMod.
1552 \def\xintiiE #1#2%
1553   {\expandafter\XINT_iiE\the\numexpr #2\expandafter.\expandafter{\romannumeral-`0#1}}%
1554 \def\XINT_iiE #1.#2{\ifnum#1>\xint_c_ \xint_dothis{\xint_dsh {#2}{-#1}}\fi
1555   \xint_orthat{ #2}}%
1556 \XINT_restorecatcodes_endinput%

```

## 5 Package `xintbinhex` implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection	119	.6	<code>\xintBinToDec</code>	126
.2	Package identification	120	.7	<code>\xintBinToHex</code>	129
.3	Constants, etc...	120	.8	<code>\xintHexToBin</code>	129
.4	<code>\xintDecToHex</code> , <code>\xintDecToBin</code>	122	.9	<code>\xintCHexToBin</code>	130
.5	<code>\xintHexToDec</code>	125			

The commenting is currently (2015/03/07) very sparse.

### 5.1 Catcodes, $\varepsilon$ -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@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{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 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 % 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 [2014/11/07 v1.1a Expandable binary and hexadecimal conversions (jfb)]%

```

## 5.3 Constants, etc...

v1.08

```

47 \chardef\xint_c_xvi          16
48 % \chardef\xint_c_ii^v       32 % already in xint.sty
49 % \chardef\xint_c_ii^vi      64 % already in xint.sty
50 \chardef\xint_c_ii^vii       128
51 \mathchardef\xint_c_ii^viii  256
52 \mathchardef\xint_c_ii^xii  4096
53 \newcount\xint_c_ii^xv      \xint_c_ii^xv  32768
54 \newcount\xint_c_ii^xvi     \xint_c_ii^xvi  65536
55 \newcount\xint_c_x^v        \xint_c_x^v    100000
56 \newcount\xint_c_x^ix       \xint_c_x^ix   1000000000
57 \def\xINT_tmpa #1{\ifx\relax#1\else
58   \expandafter\edef\csname XINT_sdth_#1\endcsname
59   {\ifcase #1 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
60     8\or 9\or A\or B\or C\or D\or E\or F\fi}%
61   \expandafter\xINT_tmpa\fi }%
62 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
63 \def\xINT_tmpa #1{\ifx\relax#1\else
64   \expandafter\edef\csname XINT_sdtb_#1\endcsname
65   {\ifcase #1
66     0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
67     1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\fi}%
68   \expandafter\xINT_tmpa\fi }%
69 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
70 \let\xINT_tmpa\relax
71 \expandafter\def\csname XINT_sbtd_0000\endcsname {0}%
72 \expandafter\def\csname XINT_sbtd_0001\endcsname {1}%
73 \expandafter\def\csname XINT_sbtd_0010\endcsname {2}%
74 \expandafter\def\csname XINT_sbtd_0011\endcsname {3}%
75 \expandafter\def\csname XINT_sbtd_0100\endcsname {4}%
76 \expandafter\def\csname XINT_sbtd_0101\endcsname {5}%
77 \expandafter\def\csname XINT_sbtd_0110\endcsname {6}%
78 \expandafter\def\csname XINT_sbtd_0111\endcsname {7}%
79 \expandafter\def\csname XINT_sbtd_1000\endcsname {8}%
80 \expandafter\def\csname XINT_sbtd_1001\endcsname {9}%
81 \expandafter\def\csname XINT_sbtd_1010\endcsname {10}%
82 \expandafter\def\csname XINT_sbtd_1011\endcsname {11}%
83 \expandafter\def\csname XINT_sbtd_1100\endcsname {12}%
84 \expandafter\def\csname XINT_sbtd_1101\endcsname {13}%

```



## 5 Package *xintbinhex* implementation

```

85 \expandafter\def\csname XINT_sbt_d_1110\endcsname {14}%
86 \expandafter\def\csname XINT_sbt_d_1111\endcsname {15}%
87 \expandafter\let\csname XINT_sbt_h_0000\expandafter\endcsname
88     \csname XINT_sbt_d_0000\endcsname
89 \expandafter\let\csname XINT_sbt_h_0001\expandafter\endcsname
90     \csname XINT_sbt_d_0001\endcsname
91 \expandafter\let\csname XINT_sbt_h_0010\expandafter\endcsname
92     \csname XINT_sbt_d_0010\endcsname
93 \expandafter\let\csname XINT_sbt_h_0011\expandafter\endcsname
94     \csname XINT_sbt_d_0011\endcsname
95 \expandafter\let\csname XINT_sbt_h_0100\expandafter\endcsname
96     \csname XINT_sbt_d_0100\endcsname
97 \expandafter\let\csname XINT_sbt_h_0101\expandafter\endcsname
98     \csname XINT_sbt_d_0101\endcsname
99 \expandafter\let\csname XINT_sbt_h_0110\expandafter\endcsname
100     \csname XINT_sbt_d_0110\endcsname
101 \expandafter\let\csname XINT_sbt_h_0111\expandafter\endcsname
102     \csname XINT_sbt_d_0111\endcsname
103 \expandafter\let\csname XINT_sbt_h_1000\expandafter\endcsname
104     \csname XINT_sbt_d_1000\endcsname
105 \expandafter\let\csname XINT_sbt_h_1001\expandafter\endcsname
106     \csname XINT_sbt_d_1001\endcsname
107 \expandafter\def\csname XINT_sbt_h_1010\endcsname {A}%
108 \expandafter\def\csname XINT_sbt_h_1011\endcsname {B}%
109 \expandafter\def\csname XINT_sbt_h_1100\endcsname {C}%
110 \expandafter\def\csname XINT_sbt_h_1101\endcsname {D}%
111 \expandafter\def\csname XINT_sbt_h_1110\endcsname {E}%
112 \expandafter\def\csname XINT_sbt_h_1111\endcsname {F}%
113 \expandafter\def\csname XINT_sht_b_0\endcsname {0000}%
114 \expandafter\def\csname XINT_sht_b_1\endcsname {0001}%
115 \expandafter\def\csname XINT_sht_b_2\endcsname {0010}%
116 \expandafter\def\csname XINT_sht_b_3\endcsname {0011}%
117 \expandafter\def\csname XINT_sht_b_4\endcsname {0100}%
118 \expandafter\def\csname XINT_sht_b_5\endcsname {0101}%
119 \expandafter\def\csname XINT_sht_b_6\endcsname {0110}%
120 \expandafter\def\csname XINT_sht_b_7\endcsname {0111}%
121 \expandafter\def\csname XINT_sht_b_8\endcsname {1000}%
122 \expandafter\def\csname XINT_sht_b_9\endcsname {1001}%
123 \def\XINT_sht_b_A {1010}%
124 \def\XINT_sht_b_B {1011}%
125 \def\XINT_sht_b_C {1100}%
126 \def\XINT_sht_b_D {1101}%
127 \def\XINT_sht_b_E {1110}%
128 \def\XINT_sht_b_F {1111}%
129 \def\XINT_sht_b_G {}%
130 \def\XINT_smallhex #1%
131 {%
132     \expandafter\XINT_smallhex_a\expandafter
133     {\the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i}{#1}%
134 }%
135 \def\XINT_smallhex_a #1#2%
136 {%

```

## 5 Package *xintbinhex* implementation

```
137 \csname XINT_sdth_#1\expandafter\expandafter\expandafter\endcsname
138 \csname XINT_sdth_\the\numexpr #2-\xint_c_xvi*#1\endcsname
139 }%
140 \def\XINT_smallbin #1%
141 {%
142 \expandafter\XINT_smallbin_a\expandafter
143 {\the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i}{#1}%
144 }%
145 \def\XINT_smallbin_a #1#2%
146 {%
147 \csname XINT_sdtb_#1\expandafter\expandafter\expandafter\endcsname
148 \csname XINT_sdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
149 }%
```

### 5.4 `\xintDecToHex`, `\xintDecToBin`

v1.08

```
150 \def\xintDecToHex {\romannumeral0\xintdectohex }%
151 \def\xintdectohex #1%
152 {\expandafter\XINT_dth_checkin\romannumeral-`0#1\W\W\W\W \T}%
153 \def\XINT_dth_checkin #1%
154 {%
155 \xint_UDsignfork
156 #1\XINT_dth_N
157 -{\XINT_dth_P #1}%
158 \krof
159 }%
160 \def\XINT_dth_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dth_P }%
161 \def\XINT_dth_P {\expandafter\XINT_dth_III\romannumeral-`0\XINT_dtbh_I {0.}}%
162 \def\xintDecToBin {\romannumeral0\xintdectobin }%
163 \def\xintdectobin #1%
164 {\expandafter\XINT_dtb_checkin\romannumeral-`0#1\W\W\W\W \T }%
165 \def\XINT_dtb_checkin #1%
166 {%
167 \xint_UDsignfork
168 #1\XINT_dtb_N
169 -{\XINT_dtb_P #1}%
170 \krof
171 }%
172 \def\XINT_dtb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dtb_P }%
173 \def\XINT_dtb_P {\expandafter\XINT_dtb_III\romannumeral-`0\XINT_dtbh_I {0.}}%
174 \def\XINT_dtbh_I #1#2#3#4#5%
175 {%
176 \xint_gob_til_W #5\XINT_dtbh_II_a\W\XINT_dtbh_I_a {}{#2#3#4#5}#1\Z.%
177 }%
178 \def\XINT_dtbh_II_a\W\XINT_dtbh_I_a #1#2{\XINT_dtbh_II_b #2}%
179 \def\XINT_dtbh_II_b #1#2#3#4%
180 {%
181 \xint_gob_til_W
182 #1\XINT_dtbh_II_c
183 #2\XINT_dtbh_II_ci
184 #3\XINT_dtbh_II_cii
```

## 5 Package *xintbinhex* implementation

```

185     \W\XINT_dtbh_II_ciii #1#2#3#4%
186 }%
187 \def\XINT_dtbh_II_c \W\XINT_dtbh_II_ci
188     \W\XINT_dtbh_II_cii
189     \W\XINT_dtbh_II_ciii \W\W\W\W {{{}}%
190 \def\XINT_dtbh_II_ci #1\XINT_dtbh_II_ciii #2\W\W\W
191     {\XINT_dtbh_II_d }{{#2}{0}}%
192 \def\XINT_dtbh_II_cii\W\XINT_dtbh_II_ciii #1#2\W\W
193     {\XINT_dtbh_II_d }{{#1#2}{00}}%
194 \def\XINT_dtbh_II_ciii #1#2#3\W
195     {\XINT_dtbh_II_d }{{#1#2#3}{000}}%
196 \def\XINT_dtbh_I_a #1#2#3.%
197 {%
198     \xint_gob_til_Z #3\XINT_dtbh_I_z\Z
199     \expandafter\XINT_dtbh_I_b\the\numexpr #2+#30000.{#1}%
200 }%
201 \def\XINT_dtbh_I_b #1.%
202 {%
203     \expandafter\XINT_dtbh_I_c\the\numexpr
204     (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
205 }%
206 \def\XINT_dtbh_I_c #1.#2.%
207 {%
208     \expandafter\XINT_dtbh_I_d\expandafter
209     {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
210 }%
211 \def\XINT_dtbh_I_d #1#2#3{\XINT_dtbh_I_a {#3#1.}{#2}}%
212 \def\XINT_dtbh_I_z\Z\expandafter\XINT_dtbh_I_b\the\numexpr #1+#2.%
213 {%
214     \ifnum #1=\xint_c_ \expandafter\XINT_dtbh_I_end_zb\fi
215     \XINT_dtbh_I_end_za {#1}%
216 }%
217 \def\XINT_dtbh_I_end_za #1#2{\XINT_dtbh_I {#2#1.}}%
218 \def\XINT_dtbh_I_end_zb\XINT_dtbh_I_end_za #1#2{\XINT_dtbh_I {#2}}%
219 \def\XINT_dtbh_II_d #1#2#3#4.%
220 {%
221     \xint_gob_til_Z #4\XINT_dtbh_II_z\Z
222     \expandafter\XINT_dtbh_II_e\the\numexpr #2+#4#3.{#1}{#3}%
223 }%
224 \def\XINT_dtbh_II_e #1.%
225 {%
226     \expandafter\XINT_dtbh_II_f\the\numexpr
227     (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
228 }%
229 \def\XINT_dtbh_II_f #1.#2.%
230 {%
231     \expandafter\XINT_dtbh_II_g\expandafter
232     {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
233 }%
234 \def\XINT_dtbh_II_g #1#2#3{\XINT_dtbh_II_d {#3#1.}{#2}}%
235 \def\XINT_dtbh_II_z\Z\expandafter\XINT_dtbh_II_e\the\numexpr #1+#2.%
236 {%

```

## 5 Package *xintbinhex* implementation

```

237 \ifnum #1=\xint_c_ \expandafter\XINT_dtbh_II_end_zb\fi
238 \XINT_dtbh_II_end_za {#1}%
239 }%
240 \def\XINT_dtbh_II_end_za #1#2#3{{#2#1.\Z.}%
241 \def\XINT_dtbh_II_end_zb\XINT_dtbh_II_end_za #1#2#3{{#2\Z.}%
242 \def\XINT_dth_III #1#2.%
243 {%
244 \xint_gob_til_Z #2\XINT_dth_end\Z
245 \expandafter\XINT_dth_III\expandafter
246 {\romannumeral-`0\XINT_dth_small #2.#1}%
247 }%
248 \def\XINT_dth_small #1.%
249 {%
250 \expandafter\XINT_smallhex\expandafter
251 {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
252 \romannumeral-`0\expandafter\XINT_smallhex\expandafter
253 {\the\numexpr
254 #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
255 }%
256 \def\XINT_dth_end\Z\expandafter\XINT_dth_III\expandafter #1#2\T
257 {%
258 \XINT_dth_end_b #1%
259 }%
260 \def\XINT_dth_end_b #1.{\XINT_dth_end_c }%
261 \def\XINT_dth_end_c #1{\xint_gob_til_zero #1\XINT_dth_end_d 0\space #1}%
262 \def\XINT_dth_end_d 0\space 0#1%
263 {%
264 \xint_gob_til_zero #1\XINT_dth_end_e 0\space #1%
265 }%
266 \def\XINT_dth_end_e 0\space 0#1%
267 {%
268 \xint_gob_til_zero #1\XINT_dth_end_f 0\space #1%
269 }%
270 \def\XINT_dth_end_f 0\space 0{ }%
271 \def\XINT_dtb_III #1#2.%
272 {%
273 \xint_gob_til_Z #2\XINT_dtb_end\Z
274 \expandafter\XINT_dtb_III\expandafter
275 {\romannumeral-`0\XINT_dtb_small #2.#1}%
276 }%
277 \def\XINT_dtb_small #1.%
278 {%
279 \expandafter\XINT_smallbin\expandafter
280 {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
281 \romannumeral-`0\expandafter\XINT_smallbin\expandafter
282 {\the\numexpr
283 #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
284 }%
285 \def\XINT_dtb_end\Z\expandafter\XINT_dtb_III\expandafter #1#2\T
286 {%
287 \XINT_dtb_end_b #1%
288 }%

```

## 5 Package *xintbinhex* implementation

```
289 \def\XINT_dtb_end_b #1.{\XINT_dtb_end_c }%
290 \def\XINT_dtb_end_c #1#2#3#4#5#6#7#8%
291 {%
292   \expandafter\XINT_dtb_end_d\the\numexpr #1#2#3#4#5#6#7#8\relax
293 }%
294 \edef\XINT_dtb_end_d #1#2#3#4#5#6#7#8#9%
295 {%
296   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
297 }%
```

### 5.5 *\xintHexToDec*

v1.08

```
298 \def\xintHexToDec {\romannumeral0\xinthextodec }%
299 \def\xinthextodec #1%
300   {\expandafter\XINT_htd_checkin\romannumeral-`0#1\W\W\W\W \T }%
301 \def\XINT_htd_checkin #1%
302 {%
303   \xint_UDsignfork
304   #1\XINT_htd_neg
305   -{\XINT_htd_I {0000}#1}%
306   \krof
307 }%
308 \def\XINT_htd_neg {\expandafter\xint_minus_thenstop
309   \romannumeral0\XINT_htd_I {0000}}%
310 \def\XINT_htd_I #1#2#3#4#5%
311 {%
312   \xint_gob_til_W #5\XINT_htd_II_a\W
313   \XINT_htd_I_a {}{"#2#3#4#5}#1\Z\Z\Z\Z
314 }%
315 \def\XINT_htd_II_a \W\XINT_htd_I_a #1#2{\XINT_htd_II_b #2}%
316 \def\XINT_htd_II_b "#1#2#3#4%
317 {%
318   \xint_gob_til_W
319   #1\XINT_htd_II_c
320   #2\XINT_htd_II_ci
321   #3\XINT_htd_II_cii
322   \W\XINT_htd_II_ciii #1#2#3#4%
323 }%
324 \def\XINT_htd_II_c \W\XINT_htd_II_ci
325   \W\XINT_htd_II_cii
326   \W\XINT_htd_II_ciii \W\W\W\W #1\Z\Z\Z\Z\T
327 {%
328   \expandafter\xint_cleanupzeros_andstop
329   \romannumeral0\XINT_rord_main }#1%
330   \xint_relax
331   \xint_bye\xint_bye\xint_bye\xint_bye
332   \xint_bye\xint_bye\xint_bye\xint_bye
333   \xint_relax
334 }%
335 \def\XINT_htd_II_ci #1\XINT_htd_II_ciii
336   #2\W\W\W\W {\XINT_htd_II_d }{"#2}{\xint_c_xvi}}%
```

## 5 Package *xintbinhex* implementation

```

337 \def\XINT_htd_II_cii\W\XINT_htd_II_ciii
338         #1#2\W\W {\XINT_htd_II_d }{"#1#2}{\xint_c_ii^viii}}%
339 \def\XINT_htd_II_ciii #1#2#3\W {\XINT_htd_II_d }{"#1#2#3}{\xint_c_ii^xii}}%
340 \def\XINT_htd_I_a #1#2#3#4#5#6%
341 {%
342     \xint_gob_til_Z #3\XINT_htd_I_end_a\Z
343     \expandafter\XINT_htd_I_b\the\numexpr
344     #2+\xint_c_ii^xvi*#6#5#4#3+\xint_c_x^ix\relax {#1}%
345 }%
346 \def\XINT_htd_I_b 1#1#2#3#4#5#6#7#8#9{\XINT_htd_I_c {#1#2#3#4#5}{#9#8#7#6}}%
347 \def\XINT_htd_I_c #1#2#3{\XINT_htd_I_a {#3#2}{#1}}%
348 \def\XINT_htd_I_end_a\Z\expandafter\XINT_htd_I_b\the\numexpr #1+#2\relax
349 {%
350     \expandafter\XINT_htd_I_end_b\the\numexpr \xint_c_x^v+#1\relax
351 }%
352 \def\XINT_htd_I_end_b 1#1#2#3#4#5%
353 {%
354     \xint_gob_til_zero #1\XINT_htd_I_end_bz0%
355     \XINT_htd_I_end_c #1#2#3#4#5%
356 }%
357 \def\XINT_htd_I_end_c #1#2#3#4#5#6{\XINT_htd_I {#6#5#4#3#2#1000}}%
358 \def\XINT_htd_I_end_bz0\XINT_htd_I_end_c 0#1#2#3#4%
359 {%
360     \xint_gob_til_zeros_iv #1#2#3#4\XINT_htd_I_end_bzz 0000%
361     \XINT_htd_I_end_D {#4#3#2#1}}%
362 }%
363 \def\XINT_htd_I_end_D #1#2{\XINT_htd_I {#2#1}}%
364 \def\XINT_htd_I_end_bzz 0000\XINT_htd_I_end_D #1{\XINT_htd_I }%
365 \def\XINT_htd_II_d #1#2#3#4#5#6#7%
366 {%
367     \xint_gob_til_Z #4\XINT_htd_II_end_a\Z
368     \expandafter\XINT_htd_II_e\the\numexpr
369     #2+#3*#7#6#5#4+\xint_c_x^viii\relax {#1}{#3}}%
370 }%
371 \def\XINT_htd_II_e 1#1#2#3#4#5#6#7#8{\XINT_htd_II_f {#1#2#3#4}{#5#6#7#8}}%
372 \def\XINT_htd_II_f #1#2#3{\XINT_htd_II_d {#2#3}{#1}}%
373 \def\XINT_htd_II_end_a\Z\expandafter\XINT_htd_II_e
374     \the\numexpr #1+#2\relax #3#4\T
375 {%
376     \XINT_htd_II_end_b #1#3%
377 }%
378 \edef\XINT_htd_II_end_b #1#2#3#4#5#6#7#8%
379 {%
380     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
381 }%

```

### 5.6 *\xintBinToDec*

v1.08

```

382 \def\xintBinToDec {\romannumeral0\xintbintodec }%
383 \def\xintbintodec #1{\expandafter\XINT_btd_checkin
384     \romannumeral-`0#1\W\W\W\W\W\W\W\W \T }%

```

## 5 Package *xintbinhex* implementation

```

385 \def\XINT_btd_checkin #1%
386 {%
387   \xint_UDsignfork
388     #1\XINT_btd_neg
389     -{\XINT_btd_I {000000}}#1}%
390   \krof
391 }%
392 \def\XINT_btd_neg {\expandafter\xint_minus_thenstop
393                   \romannumeral0\XINT_btd_I {000000}}%
394 \def\XINT_btd_I #1#2#3#4#5#6#7#8#9%
395 {%
396   \xint_gob_til_W #9\XINT_btd_II_a {#2#3#4#5#6#7#8#9}\W
397   \XINT_btd_I_a {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_xvi+%
398             \csname XINT_sbtd_#6#7#8#9\endcsname}%
399   #1\Z\Z\Z\Z\Z\Z
400 }%
401 \def\XINT_btd_II_a #1\W\XINT_btd_I_a #2#3{\XINT_btd_II_b #1}%
402 \def\XINT_btd_II_b #1#2#3#4#5#6#7#8%
403 {%
404   \xint_gob_til_W
405     #1\XINT_btd_II_c
406     #2\XINT_btd_II_ci
407     #3\XINT_btd_II_cii
408     #4\XINT_btd_II_ciii
409     #5\XINT_btd_II_civ
410     #6\XINT_btd_II_cv
411     #7\XINT_btd_II_cvi
412     \W\XINT_btd_II_cvii #1#2#3#4#5#6#7#8%
413 }%
414 \def\XINT_btd_II_c #1\XINT_btd_II_cvii \W\W\W\W\W\W\W #2\Z\Z\Z\Z\Z\Z\T
415 {%
416   \expandafter\XINT_btd_II_c_end
417   \romannumeral0\XINT_rord_main {}#2%
418   \xint_relax
419     \xint_bye\xint_bye\xint_bye\xint_bye
420     \xint_bye\xint_bye\xint_bye\xint_bye
421   \xint_relax
422 }%
423 \edef\XINT_btd_II_c_end #1#2#3#4#5#6%
424 {%
425   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6\relax
426 }%
427 \def\XINT_btd_II_ci #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
428   {\XINT_btd_II_d {}{#2}{\xint_c_ii }}%
429 \def\XINT_btd_II_cii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
430   {\XINT_btd_II_d {}{\csname XINT_sbtd_00#2\endcsname }{\xint_c_iv }}%
431 \def\XINT_btd_II_ciii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
432   {\XINT_btd_II_d {}{\csname XINT_sbtd_0#2\endcsname }{\xint_c_viii }}%
433 \def\XINT_btd_II_civ #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
434   {\XINT_btd_II_d {}{\csname XINT_sbtd_#2\endcsname}{\xint_c_xvi }}%
435 \def\XINT_btd_II_cv #1\XINT_btd_II_cvii #2#3#4#5#6\W\W\W\W
436 {%

```

## 5 Package *xintbinhex* implementation

```

437 \XINT_btd_II_d {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_ii+%
438 #6}{\xint_c_ii^v }%
439 }%
440 \def\XINT_btd_II_cvi #1\XINT_btd_II_cvii #2#3#4#5#6#7\W\W
441 {%
442 \XINT_btd_II_d {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_iv+%
443 \csname XINT_sbtd_00#6#7\endcsname}{\xint_c_ii^vi }%
444 }%
445 \def\XINT_btd_II_cvii #1#2#3#4#5#6#7\W
446 {%
447 \XINT_btd_II_d {}{\csname XINT_sbtd_#1#2#3#4\endcsname*\xint_c_viii+%
448 \csname XINT_sbtd_0#5#6#7\endcsname}{\xint_c_ii^vii }%
449 }%
450 \def\XINT_btd_II_d #1#2#3#4#5#6#7#8#9%
451 {%
452 \xint_gob_til_Z #4\XINT_btd_II_end_a\Z
453 \expandafter\XINT_btd_II_e\the\numexpr
454 #2+(\xint_c_x^ix+#3*#9#8#7#6#5#4)\relax {#1}{#3}%
455 }%
456 \def\XINT_btd_II_e 1#1#2#3#4#5#6#7#8#9{\XINT_btd_II_f {#1#2#3}{#4#5#6#7#8#9}}%
457 \def\XINT_btd_II_f #1#2#3{\XINT_btd_II_d {#2#3}{#1}}%
458 \def\XINT_btd_II_end_a\Z\expandafter\XINT_btd_II_e
459 \the\numexpr #1+(#2\relax #3#4\T
460 {%
461 \XINT_btd_II_end_b #1#3%
462 }%
463 \edef\XINT_btd_II_end_b #1#2#3#4#5#6#7#8#9%
464 {%
465 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
466 }%
467 \def\XINT_btd_I_a #1#2#3#4#5#6#7#8%
468 {%
469 \xint_gob_til_Z #3\XINT_btd_I_end_a\Z
470 \expandafter\XINT_btd_I_b\the\numexpr
471 #2+\xint_c_ii^viii*#8#7#6#5#4#3+\xint_c_x^ix\relax {#1}%
472 }%
473 \def\XINT_btd_I_b 1#1#2#3#4#5#6#7#8#9{\XINT_btd_I_c {#1#2#3}{#9#8#7#6#5#4}}%
474 \def\XINT_btd_I_c #1#2#3{\XINT_btd_I_a {#3#2}{#1}}%
475 \def\XINT_btd_I_end_a\Z\expandafter\XINT_btd_I_b
476 \the\numexpr #1+\xint_c_ii^viii #2\relax
477 {%
478 \expandafter\XINT_btd_I_end_b\the\numexpr 1000+#1\relax
479 }%
480 \def\XINT_btd_I_end_b 1#1#2#3%
481 {%
482 \xint_gob_til_zeros_iii #1#2#3\XINT_btd_I_end_bz 000%
483 \XINT_btd_I_end_c #1#2#3%
484 }%
485 \def\XINT_btd_I_end_c #1#2#3#4{\XINT_btd_I {#4#3#2#1000}}%
486 \def\XINT_btd_I_end_bz 000\XINT_btd_I_end_c 000{\XINT_btd_I }%

```



## 5.7 `\xintBinToHex`

v1.08

```

487 \def\xintBinToHex {\romannumeral0\xintbinto hex }%
488 \def\xintbinto hex #1%
489 {%
490   \expandafter\XINT_bth_checkin
491     \romannumeral0\expandafter\XINT_num_loop
492     \romannumeral-`0#1\xint_relax\xint_relax
493       \xint_relax\xint_relax
494     \xint_relax\xint_relax\xint_relax\xint_relax\Z
495   \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
496 }%
497 \def\XINT_bth_checkin #1%
498 {%
499   \xint_UDsignfork
500     #1\XINT_bth_N
501     -{\XINT_bth_P #1}%
502   \krof
503 }%
504 \def\XINT_bth_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_bth_P }%
505 \def\XINT_bth_P {\expandafter\XINT_bth_I\expandafter{\expandafter}%
506   \romannumeral0\XINT_OQ {}}%
507 \def\XINT_bth_I #1#2#3#4#5#6#7#8#9%
508 {%
509   \xint_gob_til_W #9\XINT_bth_end_a\W
510   \expandafter\expandafter\expandafter
511   \XINT_bth_I
512   \expandafter\expandafter\expandafter
513   {\csname XINT_sbth_#9#8#7#6\expandafter\expandafter\expandafter\endcsname
514   \csname XINT_sbth_#5#4#3#2\endcsname #1}%
515 }%
516 \def\XINT_bth_end_a\W \expandafter\expandafter\expandafter
517   \XINT_bth_I \expandafter\expandafter\expandafter #1%
518 {%
519   \XINT_bth_end_b #1%
520 }%
521 \def\XINT_bth_end_b #1\endcsname #2\endcsname #3%
522 {%
523   \xint_gob_til_zero #3\XINT_bth_end_z 0\space #3%
524 }%
525 \def\XINT_bth_end_z0\space 0{ }%

```

## 5.8 `\xintHexToBin`

v1.08

```

526 \def\xintHexToBin {\romannumeral0\xinthextobin }%
527 \def\xinthextobin #1%
528 {%
529   \expandafter\XINT_htb_checkin\romannumeral-`0#1GGGGGGG\T
530 }%

```

## 5 Package *xintbinhex* implementation

```
531 \def\XINT_htb_checkin #1%
532 {%
533   \xint_UDsignfork
534     #1\XINT_htb_N
535     -{\XINT_htb_P #1}%
536   \krof
537 }%
538 \def\XINT_htb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_htb_P }%
539 \def\XINT_htb_P {\XINT_htb_I_a {}}%
540 \def\XINT_htb_I_a #1#2#3#4#5#6#7#8#9%
541 {%
542   \xint_gob_til_G #9\XINT_htb_II_a G%
543   \expandafter\expandafter\expandafter
544   \XINT_htb_I_b
545   \expandafter\expandafter\expandafter
546   {\csname XINT_shtb_#2\expandafter\expandafter\expandafter\endcsname
547   \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
548   \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
549   \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
550   \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
551   \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
552   \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
553   \csname XINT_shtb_#9\endcsname }{#1}%
554 }%
555 \def\XINT_htb_I_b #1#2{\XINT_htb_I_a {#2#1}}%
556 \def\XINT_htb_II_a G\expandafter\expandafter\expandafter\XINT_htb_I_b
557 {%
558   \expandafter\expandafter\expandafter \XINT_htb_II_b
559 }%
560 \def\XINT_htb_II_b #1#2#3\T
561 {%
562   \XINT_num_loop #2#1%
563   \xint_relax\xint_relax\xint_relax\xint_relax
564   \xint_relax\xint_relax\xint_relax\xint_relax\Z
565 }%
```

### 5.9 *\xintCHexToBin*

v1.08

```
566 \def\xintCHexToBin {\romannumeral0\xintchextobin }%
567 \def\xintchextobin #1%
568 {%
569   \expandafter\XINT_chtb_checkin\romannumeral-`0#1%
570   \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
571 }%
572 \def\XINT_chtb_checkin #1%
573 {%
574   \xint_UDsignfork
575     #1\XINT_chtb_N
576     -{\XINT_chtb_P #1}%
577   \krof
578 }%
```

```

579 \def\XINT_chtb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_chtb_P }%
580 \def\XINT_chtb_P {\expandafter\XINT_chtb_I\expandafter{\expandafter}%
581         \romannumeral0\XINT_OQ {}}%
582 \def\XINT_chtb_I #1#2#3#4#5#6#7#8#9%
583 {%
584     \xint_gob_til_W #9\XINT_chtb_end_a\W
585     \expandafter\expandafter\expandafter
586     \XINT_chtb_I
587     \expandafter\expandafter\expandafter
588     {\csname XINT_shtb_#9\expandafter\expandafter\expandafter\endcsname
589     \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
590     \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
591     \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
592     \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
593     \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
594     \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
595     \csname XINT_shtb_#2\endcsname
596     #1}%
597 }%
598 \def\XINT_chtb_end_a\W\expandafter\expandafter\expandafter
599     \XINT_chtb_I\expandafter\expandafter\expandafter #1%
600 {%
601     \XINT_chtb_end_b #1%
602     \xint_relax\xint_relax\xint_relax\xint_relax
603     \xint_relax\xint_relax\xint_relax\xint_relax\Z
604 }%
605 \def\XINT_chtb_end_b #1\W#2\W#3\W#4\W#5\W#6\W#7\W#8\W\endcsname
606 {%
607     \XINT_num_loop
608 }%
609 \XINT_restorecatcodes_endinput%

```

## 6 Package `xintgcd` implementation

.1	Catcodes, $\varepsilon$ - $\TeX$ and reload detection	. . . 131	.7	<code>\xintBezoutAlgorithm</code>	. . . . . 139
.2	Package identification	. . . . . 132	.8	<code>\xintGCDof</code>	. . . . . 141
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	. . . . . 132	.9	<code>\xintLCMof</code>	. . . . . 141
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	. . . . . 133	.10	<code>\xintTypesetEuclideanAlgorithm</code>	. . . . . 142
.5	<code>\xintBezout</code>	. . . . . 134	.11	<code>\xintTypesetBezoutAlgorithm</code>	. . . . . 142
.6	<code>\xintEuclideanAlgorithm</code>	. . . . . 138			

The commenting is currently (2015/03/07) 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  $\LaTeX$ 's `\loop`.

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %

```

## 6 Package *xintgcd* implementation

```
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 [2014/11/07 v1.1a Euclidean algorithm with xint package (jfb)]%
```

### 6.3 *\xintGCD*, *\xintiGCD*

The macros of 1.09a benefits from the `\xintnum` which has been inserted inside `\xintiabs` in `\xintnameimp`; this is a little overhead but is more convenient for the user and also makes it easier to use into `\xintexpr`-essions. 1.1a adds `\xintiGCD` mainly for `\xintiexpr` benefit. Perhaps one should always have had ONLY `ii` versions from the beginning. And perhaps for sake of consistency, `\xintGCD` should be named `\xintiGCD`? too late.

## 6 Package `xintgcd` implementation

```
47 \def\xintGCD {\romannumeral0\xintgcd }%
48 \def\xintgcd #1%
49 {%
50   \expandafter\XINT_gcd\expandafter{\romannumeral0\xintiabs {#1}}%
51 }%
52 \def\XINT_gcd #1#2%
53 {%
54   \expandafter\XINT_gcd_fork\romannumeral0\xintiabs {#2}\Z #1\Z
55 }%
56 \def\xintiGCD {\romannumeral0\xintiigcd }%
57 \def\xintiigcd #1%
58 {%
59   \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiiabs {#1}}%
60 }%
61 \def\XINT_iigcd #1#2%
62 {%
63   \expandafter\XINT_gcd_fork\romannumeral0\xintiiabs {#2}\Z #1\Z
64 }%
```

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

```
65 \def\XINT_gcd_fork #1#2\Z #3#4\Z
66 {%
67   \xint_UDzerofork
68     #1\XINT_gcd_BisZero
69     #3\XINT_gcd_AisZero
70     0\XINT_gcd_loop
71   \krof
72   {#1#2}{#3#4}%
73 }%
74 \def\XINT_gcd_AisZero #1#2{ #1}%
75 \def\XINT_gcd_BisZero #1#2{ #2}%
76 \def\XINT_gcd_CheckRem #1#2\Z
77 {%
78   \xint_gob_til_zero #1\xint_gcd_end0\XINT_gcd_loop {#1#2}%
79 }%
80 \def\xint_gcd_end0\XINT_gcd_loop #1#2{ #2}%
```

`#1=B, #2=A`

```
81 \def\XINT_gcd_loop #1#2%
82 {%
83   \expandafter\expandafter\expandafter
84     \XINT_gcd_CheckRem
85   \expandafter\xint_secondoftwo
86   \romannumeral0\XINT_div_prepare {#1}{#2}\Z
87   {#1}%
88 }%
```

### 6.4 `\xintLCM`, `\xintiiLCM`

New with 1.09a. Inadvertent use of `\xintiQuo` which was promoted at the same time to add the `\xintnum` overhead. So with 1.09f `\xintiiQuo` without the overhead. However `\xintiabs` has the `\xintnum` thing.

## 6 Package *xintgcd* implementation

The advantage is that we can thus use `lcm` in `\xintexpr`. The disadvantage is that this has overhead in `\xintiexpr`. Thus 1.1a has `\xintiLCM`.

```
89 \def\xintLCM {\romannumeral0\xintlcm}%
90 \def\xintlcm #1%
91 {%
92   \expandafter\XINT_lcm\expandafter{\romannumeral0\xintiabs {#1}}%
93 }%
94 \def\XINT_lcm #1#2%
95 {%
96   \expandafter\XINT_lcm_fork\romannumeral0\xintiabs {#2}\Z #1\Z
97 }%
98 \def\xintiLCM {\romannumeral0\xintiilcm}%
99 \def\xintiilcm #1%
100 {%
101   \expandafter\XINT_iilcm\expandafter{\romannumeral0\xintiabs {#1}}%
102 }%
103 \def\XINT_iilcm #1#2%
104 {%
105   \expandafter\XINT_lcm_fork\romannumeral0\xintiabs {#2}\Z #1\Z
106 }%
107 \def\XINT_lcm_fork #1#2\Z #3#4\Z
108 {%
109   \xint_UDzerofork
110     #1\XINT_lcm_BisZero
111     #3\XINT_lcm_AisZero
112     0\expandafter
113   \krof
114   \XINT_lcm_notzero\expandafter{\romannumeral0\XINT_gcd_loop {#1#2}{#3#4}}%
115   {#1#2}{#3#4}%
116 }%
117 \def\XINT_lcm_AisZero #1#2#3#4#5{ 0}%
118 \def\XINT_lcm_BisZero #1#2#3#4#5{ 0}%
119 \def\XINT_lcm_notzero #1#2#3{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

### 6.5 `\xintBezout`

#### 1.09a inserts use of `\xintnum`

```
120 \def\xintBezout {\romannumeral0\xintbezout }%
121 \def\xintbezout #1%
122 {%
123   \expandafter\xint_bezout\expandafter {\romannumeral0\xintnum{#1}}%
124 }%
125 \def\xint_bezout #1#2%
126 {%
127   \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
128 }%

```

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

```
129 \def\XINT_bezout_fork #1#2\Z #3#4\Z
130 {%

```

## 6 Package *xintgcd* implementation

```

131 \xint_UDzerosfork
132 #1#3\XINT_bezout_botharezero
133 #10\XINT_bezout_secondiszero
134 #30\XINT_bezout_firstiszero
135 00{\xint_UDsignsfork
136 #1#3\XINT_bezout_minusminus % A < 0, B < 0
137 #1-\XINT_bezout_minusplus % A > 0, B < 0
138 #3-\XINT_bezout_plusminus % A < 0, B > 0
139 --\XINT_bezout_plusplus % A > 0, B > 0
140 \krof }%
141 \krof
142 {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
143 }%
144 \edef\XINT_bezout_botharezero #1#2#3#4#5#6%
145 {%
146 \noexpand\xintError:NoBezoutForZeros\space {0}{0}{0}{0}{0}%
147 }%

attention première entrée doit être ici (-1)^n donc 1
#4#2 = 0 = A, B = #3#1

148 \def\XINT_bezout_firstiszero #1#2#3#4#5#6%
149 {%
150 \xint_UDsignfork
151 #3{ {0}{#3#1}{0}{1}{#1}}%
152 -{ {0}{#3#1}{0}{-1}{#1}}%
153 \krof
154 }%

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

155 \def\XINT_bezout_secondiszero #1#2#3#4#5#6%
156 {%
157 \xint_UDsignfork
158 #4{ {#4#2}{0}{-1}{0}{#2}}%
159 -{ {#4#2}{0}{1}{0}{#2}}%
160 \krof
161 }%

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

162 \def\XINT_bezout_minusminus #1#2#3#4%
163 {%
164 \expandafter\XINT_bezout_mm_post
165 \romannumeral0\XINT_bezout_loop_a 1{#1}{#2}1001%
166 }%
167 \def\XINT_bezout_mm_post #1#2%
168 {%
169 \expandafter\XINT_bezout_mm_postb\expandafter
170 {\romannumeral0\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
171 }%
172 \def\XINT_bezout_mm_postb #1#2%
173 {%
174 \expandafter\XINT_bezout_mm_postc\expandafter {#2}{#1}%

```

```

175 }%
176 \edef\XINT_bezout_mmm_postc #1#2#3#4#5%
177 {%
178   \space {#4}{#5}{#1}{#2}{#3}%
179 }%

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

180 \def\XINT_bezout_minusplus #1#2#3#4%
181 {%
182   \expandafter\XINT_bezout_mp_post
183   \romannumeral0\XINT_bezout_loop_a 1{#1}{#4#2}1001%
184 }%
185 \def\XINT_bezout_mp_post #1#2%
186 {%
187   \expandafter\XINT_bezout_mp_postb\expandafter
188   {\romannumeral0\xintiiopp {#2}}{#1}%
189 }%
190 \edef\XINT_bezout_mp_postb #1#2#3#4#5%
191 {%
192   \space {#4}{#5}{#2}{#1}{#3}%
193 }%

plusminus A < 0, B > 0

194 \def\XINT_bezout_plusminus #1#2#3#4%
195 {%
196   \expandafter\XINT_bezout_pm_post
197   \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#2}1001%
198 }%
199 \def\XINT_bezout_pm_post #1%
200 {%
201   \expandafter \XINT_bezout_pm_postb \expandafter
202   {\romannumeral0\xintiiopp{#1}}%
203 }%
204 \edef\XINT_bezout_pm_postb #1#2#3#4#5%
205 {%
206   \space {#4}{#5}{#1}{#2}{#3}%
207 }%

plusplus

208 \def\XINT_bezout_plusplus #1#2#3#4%
209 {%
210   \expandafter\XINT_bezout_pp_post
211   \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#4#2}1001%
212 }%

la parité  $(-1)^N$  est en #1, et on la jette ici.

213 \edef\XINT_bezout_pp_post #1#2#3#4#5%
214 {%
215   \space {#4}{#5}{#1}{#2}{#3}%
216 }%

```



## 6 Package *xintgcd* implementation

$n = 0: 1B\alpha(0)\beta(0)\alpha(-1)\beta(-1)$   
 $n$  général:  $\{(-1)^n\{r(n)\}\{r(n-2)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{\alpha(n-2)\}\{\beta(n-2)\}\}$   
 $\#2 = B, \#3 = A$

```

217 \def\XINT_bezout_loop_a #1#2#3%
218 {%
219   \expandafter\XINT_bezout_loop_b
220   \expandafter{\the\numexpr -#1\expandafter }%
221   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
222 }%

```

Le  $q(n)$  a ici une existence éphémère, dans le version Bezout Algorithm il faudra le conserver. On voudra à la fin  $\{q(n)\{r(n)\}\{\alpha(n)\}\{\beta(n)\}\}$ . De plus ce n'est plus  $(-1)^n$  que l'on veut mais  $n$ . (ou dans un autre ordre)

$\{(-1)^n\{q(n)\}\{r(n)\}\{r(n-1)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{\alpha(n-2)\}\{\beta(n-2)\}\}$

```

223 \def\XINT_bezout_loop_b #1#2#3#4#5#6#7#8%
224 {%
225   \expandafter \XINT_bezout_loop_c \expandafter
226     {\romannumeral0\xintiadd{\XINT_Mul{#5}{#2}}{#7}}%
227     {\romannumeral0\xintiadd{\XINT_Mul{#6}{#2}}{#8}}%
228     {#1}{#3}{#4}{#5}{#6}%
229 }%

```

$\{\alpha(n)\}\{-\beta(n)\}\{-(-1)^n\}\{r(n)\}\{r(n-1)\}\{\alpha(n-1)\}\{\beta(n-1)\}$

```

230 \def\XINT_bezout_loop_c #1#2%
231 {%
232   \expandafter \XINT_bezout_loop_d \expandafter
233     {#2}{#1}%
234 }%

```

$\{\beta(n)\}\{\alpha(n)\}\{(-1)^{(n+1)}\}\{r(n)\}\{r(n-1)\}\{\alpha(n-1)\}\{\beta(n-1)\}$

```

235 \def\XINT_bezout_loop_d #1#2#3#4#5%
236 {%
237   \XINT_bezout_loop_e #4\Z {#3}{#5}{#2}{#1}%
238 }%

```

$r(n)\Z \{(-1)^{(n+1)}\}\{r(n-1)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}$

```

239 \def\XINT_bezout_loop_e #1#2\Z
240 {%
241   \xint_gob_til_zero #1\xint_bezout_loop_exit0\XINT_bezout_loop_f
242   {#1#2}%
243 }%

```

$\{r(n)\}\{(-1)^{(n+1)}\}\{r(n-1)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}$

```

244 \def\XINT_bezout_loop_f #1#2%
245 {%
246   \XINT_bezout_loop_a {#2}{#1}%
247 }%

```

$\{(-1)^{(n+1)}\{r(n)\}\{r(n-1)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}$  et itération

```

248 \def\xint_bezout_loop_exit0\XINT_bezout_loop_f #1#2%
249 {%
250   \ifcase #2
251   \or \expandafter\XINT_bezout_exiteven
252   \else\expandafter\XINT_bezout_exitodd
253   \fi
254 }%
255 \edef\XINT_bezout_exiteven #1#2#3#4#5%
256 {%
257   \space {#5}{#4}{#1}%
258 }%
259 \edef\XINT_bezout_exitodd #1#2#3#4#5%
260 {%
261   \space {-#5}{-#4}{#1}%
262 }%

```

## 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

```

263 \def\xintEuclideanAlgorithm {\romannumeral0\xinteuclideanalgorithm }%
264 \def\xinteuclideanalgorithm #1%
265 {%
266   \expandafter \XINT_euc \expandafter{\romannumeral0\xintiabs {#1}}%
267 }%
268 \def\XINT_euc #1#2%
269 {%
270   \expandafter\XINT_euc_fork \romannumeral0\xintiabs {#2}\Z #1\Z
271 }%

```

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

```

272 \def\XINT_euc_fork #1#2\Z #3#4\Z
273 {%
274   \xint_UDzerofork
275   #1\XINT_euc_BisZero
276   #3\XINT_euc_AisZero
277   0\XINT_euc_a
278   \krof
279   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
280 }%

```

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

```

281 \def\XINT_euc_AisZero #1#2#3#4#5#6{ {1}{0}{#2}{#2}{0}{0}}%
282 \def\XINT_euc_BisZero #1#2#3#4#5#6{ {1}{0}{#3}{#3}{0}{0}}%

```

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

```

283 \def\XINT_euc_a #1#2#3%
284 {%
285   \expandafter\XINT_euc_b
286   \expandafter {\the\numexpr #1+1\expandafter }%
287   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
288 }%

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

289 \def\XINT_euc_b #1#2#3#4%
290 {%
291   \XINT_euc_c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
292 }%

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.

293 \def\XINT_euc_c #1#2\Z
294 {%
295   \xint_gob_til_zero #1\xint_euc_end0\XINT_euc_a
296 }%

{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}}

297 \def\xint_euc_end0\XINT_euc_a #1#2#3#4\Z%
298 {%
299   \expandafter\xint_euc_end_
300   \romannumeral0%
301   \XINT_rord_main {}#4{{#1}{#3}}%
302   \xint_relax
303   \xint_bye\xint_bye\xint_bye\xint_bye
304   \xint_bye\xint_bye\xint_bye\xint_bye
305   \xint_relax
306 }%
307 \edef\xint_euc_end_ #1#2#3%
308 {%
309   \space {#1}{#3}{#2}%
310 }%

```

## 6.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

```

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

```

```

311 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
312 \def\xintbezoutalgorithm #1%
313 {%
314   \expandafter \XINT_bezalg \expandafter{\romannumeral0\xintiabs {#1}}%
315 }%
316 \def\XINT_bezalg #1#2%

```

## 6 Package *xintgcd* implementation

```

317 {%
318   \expandafter\XINT_bezalg_fork \romannumeral0\xintiabs {#2}\Z #1\Z
319 }%
```

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

```

320 \def\XINT_bezalg_fork #1#2\Z #3#4\Z
321 {%
322   \xint_UDzerofork
323   #1\XINT_bezalg_BisZero
324   #3\XINT_bezalg_AisZero
325   0\XINT_bezalg_a
326   \krof
327   0{#1#2}{#3#4}1001{#3#4}{#1#2}}{\Z
328 }%
329 \def\XINT_bezalg_AisZero #1#2#3\Z{ {1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
330 \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  $\{n\}\{r(n)\}\{r(n-1)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{q(n)\}\{r(n)\}$  division de #3 par #2

```

331 \def\XINT_bezalg_a #1#2#3%
332 {%
333   \expandafter\XINT_bezalg_b
334   \expandafter {\the\numexpr #1+1\expandafter }%
335   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
336 }%
```

$\{n+1\}\{q(n+1)\}\{r(n+1)\}\{r(n)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}\dots$

```

337 \def\XINT_bezalg_b #1#2#3#4#5#6#7#8%
338 {%
339   \expandafter\XINT_bezalg_c\expandafter
340   {\romannumeral0\xintiadd {\xintiiMul {#6}{#2}}{#8}}%
341   {\romannumeral0\xintiadd {\xintiiMul {#5}{#2}}{#7}}%
342   {#1}{#2}{#3}{#4}{#5}{#6}%
343 }%
```

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

```

344 \def\XINT_bezalg_c #1#2#3#4#5#6%
345 {%
346   \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
347 }%
```

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

```

348 \def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
349 {%
350   \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{#3}{#4}{#1}{#6}}%
351 }%
```

$r(n+1)\Z \{n+1\}\{r(n+1)\}\{r(n)\}\{\alpha(n+1)\}\{\beta(n+1)\}$   
 $\{\alpha(n)\}\{\beta(n)\}\{q,r,\alpha,\beta(n+1)\}$   
 Test si  $r(n+1)$  est nul.

```

352 \def\XINT_bezalg_e #1#2\Z
353 {%
354   \xint_gob_til_zero #1\xint_bezalg_end0\XINT_bezalg_a
355 }%

Ici  $r(n+1) = 0$ . On arrête on se prépare à inverser.
 $\{n+1\}\{r(n+1)\}\{r(n)\}\{\alpha(n+1)\}\{\beta(n+1)\}\{\alpha(n)\}\{\beta(n)\}$ 
 $\{q,r,\alpha,\beta(n+1)\} \dots \{\{A\}\{B\}\}\{ \}$ \Z
On veut renvoyer
 $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q1\}\{r1\}\{\alpha1=q1\}\{\beta1=1\}$ 
 $\{q2\}\{r2\}\{\alpha2\}\{\beta2\} \dots \{qN\}\{rN=0\}\{\alphaN=A/D\}\{\betaN=B/D\}$ 

356 \def\xint_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
357 {%
358   \expandafter\xint_bezalg_end_
359   \romannumeral0%
360   \XINT_rord_main {}#8{\{#1\}\{#3\}}%
361   \xint_relax
362   \xint_bye\xint_bye\xint_bye\xint_bye
363   \xint_bye\xint_bye\xint_bye\xint_bye
364   \xint_relax
365 }%

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

366 \edef\xint_bezalg_end_ #1#2#3#4%
367 {%
368   \space {\#1}\{#3\}\{0\}\{1\}\{#2\}\{#4\}\{1\}\{0\}%
369 }%

```

## 6.8 \xintGCDof

New with 1.09a. I also tried an optimization (not working two by two) which I thought was clever but it seemed to be less efficient ...

```

370 \def\xintGCDof      {\romannumeral0\xintgcdof }%
371 \def\xintgcdof     #1{\expandafter\XINT_gcdof_a\romannumeral-`0#1\relax }%
372 \def\XINT_gcdof_a #1{\expandafter\XINT_gcdof_b\romannumeral-`0#1\Z }%
373 \def\XINT_gcdof_b #1\Z #2{\expandafter\XINT_gcdof_c\romannumeral-`0#2\Z {\#1}\Z}%
374 \def\XINT_gcdof_c #1{\xint_gob_til_relax #1\XINT_gcdof_e\relax\XINT_gcdof_d #1}%
375 \def\XINT_gcdof_d #1\Z {\expandafter\XINT_gcdof_b\romannumeral0\xintgcd {\#1}}%
376 \def\XINT_gcdof_e #1\Z #2\Z { #2}%

```

## 6.9 \xintLCMof

New with 1.09a

```

377 \def\xintLCMof      {\romannumeral0\xintlcmof }%
378 \def\xintlcmof     #1{\expandafter\XINT_lcmof_a\romannumeral-`0#1\relax }%
379 \def\XINT_lcmof_a #1{\expandafter\XINT_lcmof_b\romannumeral-`0#1\Z }%

```

```

380 \def\XINT_lcmof_b #1\Z #2{\expandafter\XINT_lcmof_c\romannumeral-\`0#2\Z {#1}\Z}%
381 \def\XINT_lcmof_c #1{\xint_gob_til_relax #1\XINT_lcmof_e\relax\XINT_lcmof_d #1}%
382 \def\XINT_lcmof_d #1\Z {\expandafter\XINT_lcmof_b\romannumeral0\xintlcm {#1}}%
383 \def\XINT_lcmof_e #1\Z #2\Z { #2}%

```

## 6.10 \xintTypesetEuclideanAlgorithm

### TYPESETTING

Organisation:

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

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

$r(n-2) = q(n)r(n-1) + r(n)$  (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`

```

384 \def\xintTypesetEuclideanAlgorithm {%
385   \unless\ifdefined\xintAssignArray
386     \errmessage
387       {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
388     \expandafter\xint_gobble_iii
389   \fi
390   \XINT_TypesetEuclideanAlgorithm
391 }%
392 \def\XINT_TypesetEuclideanAlgorithm #1#2%
393 {% l'algo remplace #1 et #2 par |#1| et |#2|
394   \par
395   \begingroup
396     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
397     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
398     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
399     \count 255 1
400     \xintloop
401       \indent\hbox to \wd 0 {\hfil$\U{\numexpr 2*\count255\relax}$}%
402       ${} = \U{\numexpr 2*\count255 + 3\relax}
403       \times \U{\numexpr 2*\count255 + 2\relax}
404         + \U{\numexpr 2*\count255 + 4\relax}$%
405     \ifnum \count255 < \N
406       \par
407       \advance \count255 1
408     \repeat
409   \endgroup
410 }%

```

## 6.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a:  $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$

$\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$  Donc  $4N+8$  termes:  $U_1 = N$ ,  $U_2 = A$ ,  $U_5 = D$ ,  $U_6 = B$ ,  $q_1 = U_9$ ,  $q_n = U_{4n+5}$ , n au moins 1

$rn = U_{4n+6}$ , 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`

## 7 Package *xintfrac* implementation

```

411 \def\xintTypesetBezoutAlgorithm {%
412   \unless\ifdefined\xintAssignArray
413     \errmessage
414       {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
415     \expandafter\xint_gobble_iii
416   \fi
417   \XINT_TypesetBezoutAlgorithm
418 }%
419 \def\XINT_TypesetBezoutAlgorithm #1#2%
420 {%
421   \par
422   \begingroup
423     \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
424     \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
425     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
426     \count255 1
427     \xintloop
428       \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
429       ${} = \BEZ{4*\count255 + 5}
430       \times \BEZ{4*\count255 + 2}
431       + \BEZ{4*\count255 + 6}$\hfill\break
432       \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 7}$}%
433       ${} = \BEZ{4*\count255 + 5}
434       \times \BEZ{4*\count255 + 3}
435       + \BEZ{4*\count255 - 1}$\hfill\break
436       \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 8}$}%
437       ${} = \BEZ{4*\count255 + 5}
438       \times \BEZ{4*\count255 + 4}
439       + \BEZ{4*\count255 }$
440     \par
441     \ifnum \count255 < \N
442       \advance \count255 1
443     \repeat
444     \edef\U{\BEZ{4*\N + 4}}%
445     \edef\V{\BEZ{4*\N + 3}}%
446     \edef\D{\BEZ5}%
447     \ifodd\N
448       $\U\times\A - \V\times \B = -\D$%
449     \else
450       $\U\times\A - \V\times\B = \D$%
451     \fi
452   \par
453 \endgroup
454 }%
455 \XINT_restorecatcodes_endinput%

```

## 7 Package *xintfrac* implementation

<ul style="list-style-type: none"> <li>.1 Catcodes, <math>\varepsilon</math>-TeX and reload detection . . . 144</li> <li>.2 Package identification . . . . . 145</li> <li>.3 <code>\XINT_cntSgnFork</code> . . . . . 145</li> <li>.4 <code>\xintLen</code> . . . . . 145</li> </ul>		<ul style="list-style-type: none"> <li>.5 <code>\XINT_lenrord_loop</code> . . . . . 146</li> <li>.6 <code>\XINT_outfrac</code> . . . . . 146</li> <li>.7 <code>\XINT_inFrac</code> . . . . . 147</li> <li>.8 <code>\XINT_frac</code> . . . . . 148</li> </ul>
---	--	---

## 7 Package *xintfrac* implementation

.9	<code>\XINT_factortens, \XINT_cuz_cnt</code>	150	.39	<code>\xintSum</code>	179
.10	<code>\xintRaw</code>	152	.40	<code>\xintMul</code>	180
.11	<code>\xintPraw</code>	152	.41	<code>\xintSqr</code>	180
.12	<code>\xintRawWithZeros</code>	152	.42	<code>\xintPow</code>	181
.13	<code>\xintFloor, \xintiFloor</code>	153	.43	<code>\xintFac</code>	181
.14	<code>\xintCeil, \xintiCeil</code>	153	.44	<code>\xintPrd</code>	182
.15	<code>\xintNumerator</code>	153	.45	<code>\xintDiv</code>	182
.16	<code>\xintDenominator</code>	154	.46	<code>\xintDivFloor</code>	183
.17	<code>\xintFrac</code>	154	.47	<code>\xintDivTrunc</code>	183
.18	<code>\xintSignedFrac</code>	154	.48	<code>\xintDivRound</code>	183
.19	<code>\xintFwOver</code>	155	.49	<code>\xintMod</code>	183
.20	<code>\xintSignedFwOver</code>	155	.50	<code>\XINTinFloatMod</code>	184
.21	<code>\xintREZ</code>	156	.51	<code>\xintIsOne</code>	184
.22	<code>\xintE, \xintFloatE, \XINTinFloatE</code>	157	.52	<code>\xintGeq</code>	184
.23	<code>\xintIrr</code>	158	.53	<code>\xintMax</code>	185
.24	<code>\xintifInt</code>	159	.54	<code>\xintMaxof</code>	186
.25	<code>\xintJrr</code>	159	.55	<code>\xintMin</code>	186
.26	<code>\xintTFrac</code>	161	.56	<code>\xintMinof</code>	187
.27	<code>\XINTinFloatFracdigits</code>	161	.57	<code>\xintCmp</code>	187
.28	<code>\xintTrunc, \xintiTrunc</code>	161	.58	<code>\xintAbs</code>	189
.29	<code>\xintTTrunc</code>	163	.59	<code>\xintOpp</code>	189
.30	<code>\xintNum</code>	164	.60	<code>\xintSgn</code>	189
.31	<code>\xintRound, \xintiRound</code>	164	.61	<code>\xintFloatAdd, \XINTinFloatAdd</code>	189
.32	<code>\xintXTrunc</code>	165	.62	<code>\xintFloatSub, \XINTinFloatSub</code>	190
.33	<code>\xintDigits</code>	170	.63	<code>\xintFloatMul, \XINTinFloatMul</code>	191
.34	<code>\xintFloat</code>	171	.64	<code>\xintFloatDiv, \XINTinFloatDiv</code>	191
.35	<code>\xintPFloat</code>	174	.65	<code>\xintFloatPow, \XINTinFloatPow</code>	192
.36	<code>\XINTinFloat</code>	175	.66	<code>\xintFloatPower, \XINTinFloatPower</code>	195
.37	<code>\xintAdd</code>	177	.67	<code>\xintFloatSqrt, \XINTinFloatSqrt</code>	197
.38	<code>\xintSub</code>	179			

The commenting is currently (2015/03/07) very sparse.

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

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

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

```

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

```



## 7 Package *xintfrac* implementation

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

### 7.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2014/11/07 v1.1a Expandable operations on fractions (jfb)]%
47 \chardef\xint_c_xviii 18
```

### 7.3 `\XINT_cntSgnFork`

1.09i. Used internally, #1 must expand to `\m@ne`, `\z@`, or `\@ne` or equivalent. Does not insert a space token to stop a `romannumeral0` expansion.

```
48 \def\XINT_cntSgnFork #1%
49 {%
50   \ifcase #1\expandafter\xint_secondofthree
51     \or\expandafter\xint_thirdofthree
52     \else\expandafter\xint_firstofthree
53   \fi
54 }%
```

### 7.4 `\xintLen`

```
55 \def\xintLen {\romannumeral0\xintlen }%
56 \def\xintlen #1%
57 {%
```

## 7 Package *xintfrac* implementation

```
58 \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
59 }%
60 \def\XINT_flen #1#2#3%
61 {%
62 \expandafter\space
63 \the\numexpr -1+\XINT_Abs {#1}+\XINT_Len {#2}+\XINT_Len {#3}\relax
64 }%
```

### 7.5 `\XINT_lenrord_loop`

```
65 \def\XINT_lenrord_loop #1#2#3#4#5#6#7#8#9%
66 {% faire \romannumeral-`0\XINT_lenrord_loop 0{#1#2#3#4#5#6#7#8#9}
67 \xint_gob_til_W #9\XINT_lenrord_W\W
68 \expandafter\XINT_lenrord_loop\expandafter
69 {\the\numexpr #1+7}{#9#8#7#6#5#4#3#2}%
70 }%
71 \def\XINT_lenrord_W\W\expandafter\XINT_lenrord_loop\expandafter #1#2#3\Z
72 {%
73 \expandafter\XINT_lenrord_X\expandafter {#1#2}\Z
74 }%
75 \def\XINT_lenrord_X #1#2\Z
76 {%
77 \XINT_lenrord_Y #2\R\R\R\R\R\R\T {#1}%
78 }%
79 \def\XINT_lenrord_Y #1#2#3#4#5#6#7#8\T
80 {%
81 \xint_gob_til_W
82 #7\XINT_lenrord_Z \xint_c_viii
83 #6\XINT_lenrord_Z \xint_c_vii
84 #5\XINT_lenrord_Z \xint_c_vi
85 #4\XINT_lenrord_Z \xint_c_v
86 #3\XINT_lenrord_Z \xint_c_iv
87 #2\XINT_lenrord_Z \xint_c_iii
88 \W\XINT_lenrord_Z \xint_c_ii \Z
89 }%
90 \def\XINT_lenrord_Z #1#2\Z #3% retourne: {longueur}renverse\Z
91 {%
92 \expandafter{\the\numexpr #3-#1\relax}%
93 }%
```

### 7.6 `\XINT_outfrac`

1.06a version now outputs  $0/1[0]$  and not  $0[0]$  in case of zero. More generally all macros have been checked in `xintfrac`, `xintseries`, `xintcfrac`, to make sure the output format for fractions was always  $A/B[n]$ . (except `\xintIrr`, `\xintJrr`, `\xintRawWithZeros`)

The problem with statements like those in the previous paragraph is that it is hard to maintain consistencies across releases.

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.

```
94 \def\XINT_outfrac #1#2#3%
95 {%
```

## 7 Package *xintfrac* implementation

```

96 \ifcase\XINT_cntSgn #3\Z
97   \expandafter \XINT_outfrac_divisionbyzero
98 \or
99   \expandafter \XINT_outfrac_P
100 \else
101   \expandafter \XINT_outfrac_N
102 \fi
103 {#2}{#3}[#1]%
104 }%
105 \def\XINT_outfrac_divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
106 \edef\XINT_outfrac_P #1#2%
107 {%
108   \noexpand\if0\noexpand\XINT_Sgn #1\noexpand\Z
109   \noexpand\expandafter\noexpand\XINT_outfrac_Zero
110   \noexpand\fi
111   \space #1/#2%
112 }%
113 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
114 \def\XINT_outfrac_N #1#2%
115 {%
116   \expandafter\XINT_outfrac_N_a\expandafter
117   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
118 }%
119 \def\XINT_outfrac_N_a #1#2%
120 {%
121   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
122 }%

```

### 7.7 `\XINT_inFrac`

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.

```

123 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
124 \def\XINT_infrac #1%
125 {%
126   \expandafter\XINT_infrac_ \romannumeral-`0#1[\W]\Z\T
127 }%
128 \def\XINT_infrac_ #1[#2#3]#4\Z
129 {%
130   \xint_UDwfork
131   #2\XINT_infrac_A
132   \W\XINT_infrac_B
133   \krof
134   #1[#2#3]#4%
135 }%
136 \def\XINT_infrac_A #1[\W]\T
137 {%
138   \XINT_frac #1/\W\Z
139 }%
140 \def\XINT_infrac_B #1%

```

```

141 {%
142   \xint_gob_til_zero #1\XINT_infrac_Zero0\XINT_infrac_BB #1%
143 }%
144 \def\XINT_infrac_BB #1[\W]\T {\XINT_infrac_BC #1/\W\Z }%
145 \def\XINT_infrac_BC #1/#2#3\Z
146 {%
147   \xint_UDwfork
148   #2\XINT_infrac_BCa
149   \W{\expandafter\XINT_infrac_BCb \romannumeral-`0#2}%
150   \krof
151   #3\Z #1\Z
152 }%
153 \def\XINT_infrac_BCa \Z #1[#2]#3\Z { {#2}{#1}{1}}%
154 \def\XINT_infrac_BCb #1[#2]/\W\Z #3\Z { {#2}{#3}{#1}}%
155 \def\XINT_infrac_Zero #1\T { {0}{0}{1}}%

```

## 7.8 \XINT\_frac

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 `\xint-expr..\relax`

```

156 \def\XINT_frac #1/#2#3\Z
157 {%
158   \xint_UDwfork
159   #2\XINT_frac_A
160   \W{\expandafter\XINT_frac_U \romannumeral-`0#2}%
161   \krof
162   #3e\W\Z #1e\W\Z
163 }%
164 \def\XINT_frac_U #1e#2#3\Z
165 {%
166   \xint_UDwfork
167   #2\XINT_frac_Ua
168   \W{\XINT_frac_Ub #2}%
169   \krof
170   #3\Z #1\Z
171 }%
172 \def\XINT_frac_Ua \Z #1/\W\Z {\XINT_frac_B #1.\W\Z {0}}%
173 \def\XINT_frac_Ub #1/\W e\W\Z #2\Z {\XINT_frac_B #2.\W\Z {#1}}%
174 \def\XINT_frac_B #1.#2#3\Z
175 {%
176   \xint_UDwfork
177   #2\XINT_frac_Ba
178   \W{\XINT_frac_Bb #2}%
179   \krof
180   #3\Z #1\Z
181 }%
182 \def\XINT_frac_Ba \Z #1\Z {\XINT_frac_T {0}{#1}}%
183 \def\XINT_frac_Bb #1.\W\Z #2\Z
184 {%
185   \expandafter \XINT_frac_T \expandafter
186   {\romannumeral0\xintlength {#1}}{#2#1}%

```

## 7 Package *xintfrac* implementation

```

187 }%
188 \def\XINT_frac_A e\W\Z {\XINT_frac_T {0}{1}{0}}%
189 \def\XINT_frac_T #1#2#3#4e#5#6\Z
190 {%
191   \xint_UDwfork
192     #5\XINT_frac-Ta
193     \W{\XINT_frac-Tb #5}%
194   \krof
195   #6\Z #4\Z {#1}{#2}{#3}%
196 }%
197 \def\XINT_frac-Ta \Z #1\Z      {\XINT_frac_C #1.\W\Z {0}}%
198 \def\XINT_frac-Tb #1e\W\Z #2\Z {\XINT_frac_C #2.\W\Z {#1}}%
199 \def\XINT_frac_C #1.#2#3\Z
200 {%
201   \xint_UDwfork
202     #2\XINT_frac-Ca
203     \W{\XINT_frac-Cb #2}%
204   \krof
205   #3\Z #1\Z
206 }%
207 \def\XINT_frac-Ca \Z #1\Z {\XINT_frac_D {0}{#1}}%
208 \def\XINT_frac-Cb #1.\W\Z #2\Z
209 {%
210   \expandafter\XINT_frac_D\expandafter
211   {\romannumeral0\xintlength {#1}}{#2#1}%
212 }%
213 \def\XINT_frac_D #1#2#3#4#5#6%
214 {%
215   \expandafter \XINT_frac_E \expandafter
216   {\the\numexpr -#1+#3+#4-#6\expandafter}\expandafter
217   {\romannumeral0\XINT_num_loop #2%
218     \xint_relax\xint_relax\xint_relax\xint_relax
219     \xint_relax\xint_relax\xint_relax\xint_relax\Z }%
220   {\romannumeral0\XINT_num_loop #5%
221     \xint_relax\xint_relax\xint_relax\xint_relax
222     \xint_relax\xint_relax\xint_relax\xint_relax\Z }%
223 }%
224 \def\XINT_frac_E #1#2#3%
225 {%
226   \expandafter \XINT_frac_F #3\Z {#2}{#1}%
227 }%
228 \def\XINT_frac_F #1%
229 {%
230   \xint_UDzerominusfork
231     #1-\XINT_frac_Gdivisionbyzero
232     0#1\XINT_frac_Gneg
233     0-{\XINT_frac_Gpos #1}%
234   \krof
235 }%
236 \edef\XINT_frac_Gdivisionbyzero #1\Z #2#3%
237 {%
238   \noexpand\xintError:DivisionByZero\space {0}{#2}{0}%

```

```

239 }%
240 \def\XINT_frac_Gneg #1\Z #2#3%
241 {%
242   \expandafter\XINT_frac_H \expandafter{\romannumeral0\XINT_opp #2}{#3}{#1}%
243 }%
244 \def\XINT_frac_H #1#2{ #{2}{#1}}%
245 \def\XINT_frac_Gpos #1\Z #2#3{ #{3}{#2}{#1}}%

```

## 7.9 \XINT\_factortens, \XINT\_cuz\_cnt

```

246 \def\XINT_factortens #1%
247 {%
248   \expandafter\XINT_cuz_cnt_loop\expandafter
249   {\expandafter}\romannumeral0\XINT_rord_main { }#1%
250   \xint_relax
251   \xint_bye\xint_bye\xint_bye\xint_bye
252   \xint_bye\xint_bye\xint_bye\xint_bye
253   \xint_relax
254   \R\R\R\R\R\R\R\Z
255 }%
256 \def\XINT_cuz_cnt #1%
257 {%
258   \XINT_cuz_cnt_loop { }#1\R\R\R\R\R\R\R\Z
259 }%
260 \def\XINT_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
261 {%
262   \xint_gob_til_R #9\XINT_cuz_cnt_toofara \R
263   \expandafter\XINT_cuz_cnt_checka\expandafter
264   {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
265 }%
266 \def\XINT_cuz_cnt_toofara\R
267   \expandafter\XINT_cuz_cnt_checka\expandafter #1#2%
268 {%
269   \XINT_cuz_cnt_toofarb {#1}#2%
270 }%
271 \def\XINT_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {#1}}%
272 \def\XINT_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
273 {%
274   \xint_gob_til_R #2\XINT_cuz_cnt_toofard 7%
275   #3\XINT_cuz_cnt_toofard 6%
276   #4\XINT_cuz_cnt_toofard 5%
277   #5\XINT_cuz_cnt_toofard 4%
278   #6\XINT_cuz_cnt_toofard 3%
279   #7\XINT_cuz_cnt_toofard 2%
280   #8\XINT_cuz_cnt_toofard 1%
281   \Z #1#2#3#4#5#6#7#8%
282 }%
283 \def\XINT_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
284 {%
285   \expandafter\XINT_cuz_cnt_toofare
286   \the\numexpr #3\relax \R\R\R\R\R\R\R\Z
287   {\the\numexpr #5-#1\relax}\R\Z
288 }%

```

## 7 Package *xintfrac* implementation

```

289 \def\XINT_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
290 {%
291     \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
292         #3\XINT_cuz_cnt_stopc 2%
293         #4\XINT_cuz_cnt_stopc 3%
294         #5\XINT_cuz_cnt_stopc 4%
295         #6\XINT_cuz_cnt_stopc 5%
296         #7\XINT_cuz_cnt_stopc 6%
297         #8\XINT_cuz_cnt_stopc 7%
298         \Z #1#2#3#4#5#6#7#8%
299 }%
300 \def\XINT_cuz_cnt_checka #1#2%
301 {%
302     \expandafter\XINT_cuz_cnt_checkb\the\numexpr #2\relax \Z {#1}%
303 }%
304 \def\XINT_cuz_cnt_checkb #1%
305 {%
306     \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
307     0\XINT_cuz_cnt_stopa #1%
308 }%
309 \def\XINT_cuz_cnt_stopa #1\Z
310 {%
311     \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\Z %
312 }%
313 \def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
314 {%
315     \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
316         #3\XINT_cuz_cnt_stopc 2%
317         #4\XINT_cuz_cnt_stopc 3%
318         #5\XINT_cuz_cnt_stopc 4%
319         #6\XINT_cuz_cnt_stopc 5%
320         #7\XINT_cuz_cnt_stopc 6%
321         #8\XINT_cuz_cnt_stopc 7%
322         #9\XINT_cuz_cnt_stopc 8%
323         \Z #1#2#3#4#5#6#7#8#9%
324 }%
325 \def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
326 {%
327     \expandafter\XINT_cuz_cnt_stopd\expandafter
328     {\the\numexpr #5-#1}#3%
329 }%
330 \def\XINT_cuz_cnt_stopd #1#2\R #3\Z
331 {%
332     \expandafter\space\expandafter
333     {\romannumeral0\XINT_rord_main {}}#2%
334     \xint_relax
335     \xint_bye\xint_bye\xint_bye\xint_bye
336     \xint_bye\xint_bye\xint_bye\xint_bye
337     \xint_relax }{#1}%
338 }%

```

## 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.

```
339 \def\xintRaw {\romannumeral0\xintraw }%
340 \def\xintraw
341 {%
342   \expandafter\XINT_raw\romannumeral0\XINT_infrac
343 }%
344 \def\XINT_raw #1#2#3{ #2/#3[#1]}%
```

## 7.11 `\xintPraw`

1.09b

```
345 \def\xintPraw {\romannumeral0\xintpraw }%
346 \def\xintpraw
347 {%
348   \expandafter\XINT_praw\romannumeral0\XINT_infrac
349 }%
350 \def\XINT_praw #1%
351 {%
352   \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
353 }%
354 \def\XINT_praw_A #1#2#3%
355 {%
356   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
357   \else\expandafter\xint_secondoftwo
358   \fi { #2[#1]}{ #2/#3[#1]}%
359 }%
360 \def\XINT_praw_a\XINT_praw_A #1#2#3%
361 {%
362   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
363   \else\expandafter\xint_secondoftwo
364   \fi { #2}{ #2/#3}%
365 }%
```

## 7.12 `\xintRawWithZeros`

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

```
366 \def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
367 \def\xintrawwithzeros
368 {%
369   \expandafter\XINT_rawz\romannumeral0\XINT_infrac
370 }%
371 \def\XINT_rawz #1%
372 {%
373   \ifcase\XINT_cntSgn #1\Z
374   \expandafter\XINT_rawz_Ba
375   \or
376   \expandafter\XINT_rawz_A
```



```

377   \else
378     \expandafter\XINT_rawz_Ba
379   \fi
380   {#1}%
381 }%
382 \def\XINT_rawz_A #1#2#3{\xint_dsh {#2}{-#1}/#3}%
383 \def\XINT_rawz_Ba #1#2#3{\expandafter\XINT_rawz_Bb
384   \expandafter{\romannumeral0\xint_dsh {#3}{#1}}{#2}}%
385 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

### 7.13 `\xintFloor`, `\xintiFloor`

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

```

386 \def\xintFloor {\romannumeral0\xintfloor }%
387 \def\xintfloor #1% devrais-je faire \xintREZ?
388   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}./1[0]}%
389 \def\xintiFloor {\romannumeral0\xintifloor }%
390 \def\xintifloor #1%
391   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}.)}%
392 \def\XINT_ifloor #1/#2.{\xintiiquo {#1}{#2}}%

```

### 7.14 `\xintCeil`, `\xintiCeil`

1.09a

```

393 \def\xintCeil {\romannumeral0\xintceil }%
394 \def\xintceil #1{\xintiioopp {\xintFloor {\xintOpp{#1}}}}%
395 \def\xintiCeil {\romannumeral0\xinticeil }%
396 \def\xinticeil #1{\xintiioopp {\xintiFloor {\xintOpp{#1}}}}%

```

### 7.15 `\xintNumerator`

```

397 \def\xintNumerator {\romannumeral0\xintnumerator }%
398 \def\xintnumerator
399 {%
400   \expandafter\XINT_numer\romannumeral0\XINT_infrac
401 }%
402 \def\XINT_numer #1%
403 {%
404   \ifcase\XINT_cntSgn #1\Z
405     \expandafter\XINT_numer_B
406   \or
407     \expandafter\XINT_numer_A
408   \else
409     \expandafter\XINT_numer_B
410   \fi
411   {#1}%
412 }%
413 \def\XINT_numer_A #1#2#3{\xint_dsh {#2}{-#1}}%
414 \def\XINT_numer_B #1#2#3{ #2}%

```

**7.16 `\xintDenominator`**

```

415 \def\xintDenominator {\romannumeral0\xintdenominator }%
416 \def\xintdenominator
417 {%
418   \expandafter\XINT_denom\romannumeral0\XINT_infrac
419 }%
420 \def\XINT_denom #1%
421 {%
422   \ifcase\XINT_cntSgn #1\Z
423     \expandafter\XINT_denom_B
424   \or
425     \expandafter\XINT_denom_A
426   \else
427     \expandafter\XINT_denom_B
428   \fi
429   {#1}%
430 }%
431 \def\XINT_denom_A #1#2#3{ #3}%
432 \def\XINT_denom_B #1#2#3{\xint_dsh {#3}{#1}}%

```

**7.17 `\xintFrac`**

```

433 \def\xintFrac {\romannumeral0\xintfrac }%
434 \def\xintfrac #1%
435 {%
436   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
437 }%
438 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
439 \catcode\^=7
440 \def\XINT_fracfrac_B #1#2\Z
441 {%
442   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
443 }%
444 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
445 {%
446   \if1\XINT_isOne {#3}%
447     \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
448   \fi
449   \space
450   \frac {#2}{#3}%
451 }%
452 \def\XINT_fracfrac_D #1#2#3%
453 {%
454   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
455   \space
456   \frac {#2}{#3}#1%
457 }%
458 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

**7.18 `\xintSignedFrac`**

```

459 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
460 \def\xintsignedfrac #1%
461 {%

```

## 7 Package *xintfrac* implementation

```

462 \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
463 }%
464 \def\XINT_sgnfrac_a #1#2%
465 {%
466 \XINT_sgnfrac_b #2\Z {#1}%
467 }%
468 \def\XINT_sgnfrac_b #1%
469 {%
470 \xint_UDsignfork
471 #1\XINT_sgnfrac_N
472 -{\XINT_sgnfrac_P #1}%
473 \krof
474 }%
475 \def\XINT_sgnfrac_P #1\Z #2%
476 {%
477 \XINT_fracfrac_A {#2}{#1}%
478 }%
479 \def\XINT_sgnfrac_N
480 {%
481 \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfrac_P
482 }%

```

### 7.19 *\xintFwOver*

```

483 \def\xintFwOver {\romannumeral0\xintfwover }%
484 \def\xintfwover #1%
485 {%
486 \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
487 }%
488 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
489 \def\XINT_fwover_B #1#2\Z
490 {%
491 \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
492 }%
493 \catcode`^=11
494 \def\XINT_fwover_C #1#2#3#4#5%
495 {%
496 \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
497 \else\xint_afterfi { #4}%
498 \fi
499 }%
500 \def\XINT_fwover_D #1#2#3%
501 {%
502 \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
503 \else\xint_afterfi { #2\cdot }%
504 \fi
505 #1%
506 }%

```

### 7.20 *\xintSignedFwOver*

```

507 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
508 \def\xintsignedfwover #1%
509 {%
510 \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%

```

```

511 }%
512 \def\XINT_sgnfwover_a #1#2%
513 {%
514   \XINT_sgnfwover_b #2\Z {#1}%
515 }%
516 \def\XINT_sgnfwover_b #1%
517 {%
518   \xint_UDsignfork
519   #1\XINT_sgnfwover_N
520   -{\XINT_sgnfwover_P #1}%
521   \krof
522 }%
523 \def\XINT_sgnfwover_P #1\Z #2%
524 {%
525   \XINT_fwover_A {#2}{#1}%
526 }%
527 \def\XINT_sgnfwover_N
528 {%
529   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfwover_P
530 }%

```

### 7.21 \xintREZ

```

531 \def\xintREZ {\romannumeral0\xintrez }%
532 \def\xintrez
533 {%
534   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
535 }%
536 \def\XINT_rez_A #1#2%
537 {%
538   \XINT_rez_AB #2\Z {#1}%
539 }%
540 \def\XINT_rez_AB #1%
541 {%
542   \xint_UDzerominusfork
543   #1-\XINT_rez_zero
544   0#1\XINT_rez_neg
545   0-{\XINT_rez_B #1}%
546   \krof
547 }%
548 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
549 \def\XINT_rez_neg {\expandafter\xint_minus_thenstop\romannumeral0\XINT_rez_B }%
550 \def\XINT_rez_B #1\Z
551 {%
552   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
553 }%
554 \def\XINT_rez_C #1#2#3#4%
555 {%
556   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}{#3}{#2}{#1}%
557 }%
558 \def\XINT_rez_D #1#2#3#4#5%
559 {%
560   \expandafter\XINT_rez_E\expandafter
561   {\the\numexpr #3+#4-#2}{#1}{#5}%

```

```
562 }%
563 \def\XINT_rez_E #1#2#3{ #3/#2[#1]}%
```

## 7.22 `\xintE`, `\xintFloatE`, `\XINTinFloatE`

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

`\xintfE` (1.07) and `\xintiE` (1.09i) are for `\xintexpr` and cousins. It is quite annoying that `\numexpr` does not know how to deal correctly with a minus sign - as prefix: `\numexpr -(1)\relax` is illegal! (one can do `\numexpr 0-(1)\relax`).

the 1.07 `\xintE` puts directly its second argument in a `\numexpr`. The `\xintfE` first uses `\xintNum` on it, this is necessary for use in `\xintexpr`. (but one cannot use directly infix notation in the second argument of `\xintfE`)

1.09i also adds `\xintFloatE` and modifies `\XINTinFloatE`, although currently the latter is only used from `\xintfloatexpr` hence always with `\XINTdigits`, it comes equipped with its first argument within brackets as the other `\XINTinFloat...` macros.

1.09m ceases here and elsewhere, also in `\xintcfracname`, to use `\Z` as delimiter in the code for the optional argument, as this is unsafe (it makes impossible to the user to employ `\Z` as argument to the macro). Replaced by `\xint_relax`. 1.09e had already done that in `\xintSeq`, but this should have been systematic.

1.1 modifies and moves `\xintiiE` to `xint.sty`, and cleans up some unneeded stuff, now that expressions implement scientific notation directly at the number parsing level.

```
564 \def\xintE {\romannumeral0\xinte }%
565 \def\xinte #1%
566 {%
567   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
568 }%
569 \def\XINT_e #1#2#3#4%
570 {%
571   \expandafter\XINT_e_end\expandafter{\the\numexpr #1+#4}{#2}{#3}%
572 }%
573 \def\XINT_e_end #1#2#3{ #2/#3[#1]}%
574 \def\xintFloatE {\romannumeral0\xintfloate }%
575 \def\xintfloate #1{\XINT_floate_chkopt #1\xint_relax }%
576 \def\XINT_floate_chkopt #1%
577 {%
578   \ifx [#1\expandafter\XINT_floate_opt
579     \else\expandafter\XINT_floate_noopt
580   \fi #1%
581 }%
582 \def\XINT_floate_noopt #1\xint_relax
583 {%
584   \expandafter\XINT_floate_a\expandafter\XINTdigits
585     \romannumeral0\XINT_infrac {#1}%
586 }%
587 \def\XINT_floate_opt [\xint_relax #1]#2%
588 {%
589   \expandafter\XINT_floate_a\expandafter
590     {\the\numexpr #1\expandafter}\romannumeral0\XINT_infrac {#2}%
591 }%
592 \def\XINT_floate_a #1#2#3#4#5%
593 {%
594   \expandafter\expandafter\expandafter\XINT_float_a
```

## 7 Package *xintfrac* implementation

```

595 \expandafter\xint_exchangetwo_keepbraces\expandafter
596     {\the\numexpr #2+#5}{#1}{#3}{#4}\XINT_float_Q
597 }%
598 \def\xINTinFloatE {\romannumeral0\xINTinfloate }%
599 \def\xINTinfloate {\expandafter\xINT_infloate\romannumeral0\xINTinfloat [\XINTdigits]}%
600 \def\xINT_infloate #1[#2]#3%
601     {\expandafter\xINT_infloate_end\expandafter {\the\numexpr #3+#2}{#1}}%
602 \def\xINT_infloate_end #1#2{ #2[#1]}%

```

### 7.23 `\xintIrr`

1.04 fixes a buggy `\xintIrr {0}`. 1.05 modifies the initial parsing and post-processing to use `\xintrawwithzeros` and to more quickly deal with an input denominator equal to 1. 1.08 version does not remove a /1 denominator.

```

603 \def\xintIrr {\romannumeral0\xintirr }%
604 \def\xintirr #1%
605 {%
606     \expandafter\xINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z
607 }%
608 \def\xINT_irr_start #1#2/#3\Z
609 {%
610     \if0\xINT_isOne {#3}%
611     \xint_afterfi
612     {\xint_UDsignfork
613         #1\xINT_irr_negative
614         -{\XINT_irr_nonneg #1}%
615     \krof}%
616     \else
617     \xint_afterfi{\XINT_irr_denomiseone #1}%
618     \fi
619     #2\Z {#3}%
620 }%
621 \def\xINT_irr_denomiseone #1\Z #2{ #1/1}% changed in 1.08
622 \def\xINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z \xint_minus_thenstop}%
623 \def\xINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
624 \def\xINT_irr_D #1#2\Z #3#4\Z
625 {%
626     \xint_UDzerosfork
627     #3#1\xINT_irr_indeterminate
628     #30\xINT_irr_divisionbyzero
629     #10\xINT_irr_zero
630     00\xINT_irr_loop_a
631     \krof
632     {#3#4}{#1#2}{#3#4}{#1#2}%
633 }%
634 \def\xINT_irr_indeterminate #1#2#3#4#5{\xintError:NaN\space 0/0}%
635 \def\xINT_irr_divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
636 \def\xINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
637 \def\xINT_irr_loop_a #1#2%
638 {%
639     \expandafter\xINT_irr_loop_d
640     \romannumeral0\xINT_div_prepare {#1}{#2}{#1}%

```

```

641 }%
642 \def\XINT_irr_loop_d #1#2%
643 {%
644   \XINT_irr_loop_e #2\Z
645 }%
646 \def\XINT_irr_loop_e #1#2\Z
647 {%
648   \xint_gob_til_zero #1\xint_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
649 }%
650 \def\xint_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
651 {%
652   \expandafter\XINT_irr_loop_exitb\expandafter
653   {\romannumeral0\xintiiquo {#3}{#2}}%
654   {\romannumeral0\xintiiquo {#4}{#2}}%
655 }%
656 \def\XINT_irr_loop_exitb #1#2%
657 {%
658   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
659 }%
660 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

## 7.24 `\xintifInt`

1.09e. *xintfrac.sty* only. Fixed in 1.1 to not use `\xintIrr` anymore as it was really stupid overhead.

```

661 \def\xintifInt {\romannumeral0\xintifint }%
662 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawwithzeros {#1}.}%
663 \def\XINT_ifint #1/#2.%
664 {%
665   \if 0\xintiiRem {#1}{#2}%
666   \expandafter\xint_firstoftwo_thenstop
667   \else
668   \expandafter\xint_secondoftwo_thenstop
669   \fi
670 }%

```

## 7.25 `\xintJrr`

Modified similarly as `\xintIrr` in release 1.05. 1.08 version does not remove a /1 denominator.

```

671 \def\xintJrr {\romannumeral0\xintjrr }%
672 \def\xintjrr #1%
673 {%
674   \expandafter\XINT_jrr_start\romannumeral0\xintraawwithzeros {#1}\Z
675 }%
676 \def\XINT_jrr_start #1#2/#3\Z
677 {%
678   \if0\XINT_isOne {#3}\xint_afterfi
679     {\xint_UDsignfork
680       #1\XINT_jrr_negative
681       -{\XINT_jrr_nonneg #1}%
682       \krof}%

```

## 7 Package *xintfrac* implementation

```

683 \else
684 \xint_afterfi{\XINT_jrr_denomisine #1}%
685 \fi
686 #2\Z {#3}%
687 }%
688 \def\XINT_jrr_denomisine #1\Z #2{ #1/1}% changed in 1.08
689 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z \xint_minus_thenstop }%
690 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
691 \def\XINT_jrr_D #1#2\Z #3#4\Z
692 {%
693 \xint_UDzerosfork
694 #3#1\XINT_jrr_indeterminate
695 #30\XINT_jrr_divisionbyzero
696 #10\XINT_jrr_zero
697 00\XINT_jrr_loop_a
698 \krof
699 {#3#4}{#1#2}1001%
700 }%
701 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7{\xintError:NaN\space 0/0}%
702 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
703 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
704 \def\XINT_jrr_loop_a #1#2%
705 {%
706 \expandafter\XINT_jrr_loop_b
707 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
708 }%
709 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
710 {%
711 \expandafter \XINT_jrr_loop_c \expandafter
712 {\romannumeral0\xintiiadd{\XINT_Mul{#4}{#1}}{#6}}%
713 {\romannumeral0\xintiiadd{\XINT_Mul{#5}{#1}}{#7}}%
714 {#2}{#3}{#4}{#5}%
715 }%
716 \def\XINT_jrr_loop_c #1#2%
717 {%
718 \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
719 }%
720 \def\XINT_jrr_loop_d #1#2#3#4%
721 {%
722 \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
723 }%
724 \def\XINT_jrr_loop_e #1#2\Z
725 {%
726 \xint_gob_til_zero #1\xint_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
727 }%
728 \def\xint_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
729 {%
730 \XINT_irr_finish {#3}{#4}%
731 }%

```



## 7.26 `\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`.

```

732 \def\xintTFrac {\romannumeral0\xinttfrac }%
733 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintraawithzeros {#1}\Z }%
734 \def\XINT_tfrac_fork #1%
735 {%
736   \xint_UDzerominusfork
737     #1-\XINT_tfrac_zero
738     0#1{\xintiopp\XINT_tfrac_P }%
739     0-{\XINT_tfrac_P #1}%
740   \krof
741 }%
742 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
743 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
744   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

## 7.27 `\XINTinFloatFracdigits`

1.09i, for `frac` in `\xintfloatexpr`. This version computes exactly from the input the fractional part and then only converts it into a float with the asked-for number of digits. I will have to think it again some day, certainly.

1.1 removes optional argument for which there was anyhow no interface, for technical reasons having to do with `\xintNewExpr`.

1.1a renames the macro as `\XINTinFloatFracdigits` (from `\XINTinFloatFrac`) to be synchronous with the `\XINTinFloatSqrt` and `\XINTinFloat` habits related to `\xintNewExpr` problems.

Note to myself: I still have to rethink the whole thing about what is the best to do, the initial way of going through `\xinttfrac` was just a first implementation.

```

745 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
746 \def\XINTinfloatfracdigits #1%
747 {%
748   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xinttfrac{#1}}%
749 }%
750 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

## 7.28 `\xintTrunc`, `\xintiTrunc`

Modified in 1.06 to give the first argument to a `\numexpr`.

1.09f fixes the overhead added in 1.09a to some inner routines when `\xintiquo` was redefined to use `\xintnum`. Now uses `\xintiiquo`, rather.

1.09j: minor improvements, `\XINT_trunc_E` was very strange and defined two never occurring branches; also, optimizes the call to the division routine, and the zero loops.

1.1 adds `\xintTTrunc` as a shortcut to what `\xintiTrunc 0` does, and maps `\xintNum` to it.

```

751 \def\xintTrunc {\romannumeral0\xinttrunc }%
752 \def\xintiTrunc {\romannumeral0\xintitrunc }%

```

## 7 Package *xintfrac* implementation

```

753 \def\xinttrunc #1%
754 {%
755   \expandafter\XINT_trunc\expandafter {\the\numexpr #1}%
756 }%
757 \def\XINT_trunc #1#2%
758 {%
759   \expandafter\XINT_trunc_G
760   \romannumeral0\expandafter\XINT_trunc_A
761   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
762 }%
763 \def\xintitrunc #1%
764 {%
765   \expandafter\XINT_itrunc\expandafter {\the\numexpr #1}%
766 }%
767 \def\XINT_itrunc #1#2%
768 {%
769   \expandafter\XINT_itrunc_G
770   \romannumeral0\expandafter\XINT_trunc_A
771   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
772 }%
773 \def\XINT_trunc_A #1#2#3#4%
774 {%
775   \expandafter\XINT_trunc_checkifzero
776   \expandafter{\the\numexpr #1+#4}#2\Z {#3}%
777 }%
778 \def\XINT_trunc_checkifzero #1#2#3\Z
779 {%
780   \xint_gob_til_zero #2\XINT_trunc_iszero0\XINT_trunc_B {#1}{#2#3}%
781 }%
782 \def\XINT_trunc_iszero0\XINT_trunc_B #1#2#3{ 0\Z 0}%
783 \def\XINT_trunc_B #1%
784 {%
785   \ifcase\XINT_cntSgn #1\Z
786     \expandafter\XINT_trunc_D
787   \or
788     \expandafter\XINT_trunc_D
789   \else
790     \expandafter\XINT_trunc_C
791   \fi
792   {#1}%
793 }%
794 \def\XINT_trunc_C #1#2#3%
795 {%
796   \expandafter\XINT_trunc_CE\expandafter
797   {\romannumeral0\XINT_dsx_zero loop {-#1}{}\Z {#3}}{#2}%
798 }%
799 \def\XINT_trunc_CE #1#2{\XINT_trunc_E #2.{#1}}%
800 \def\XINT_trunc_D #1#2%
801 {%
802   \expandafter\XINT_trunc_E
803   \romannumeral0\XINT_dsx_zero loop {#1}{}\Z {#2}.%
804 }%

```

## 7 Package *xintfrac* implementation

```
805 \def\XINT_trunc_E #1%
806 {%
807   \xint_UDsignfork
808     #1\XINT_trunc_Fneg
809     -{\XINT_trunc_Fpos #1}%
810   \krof
811 }%
812 \def\XINT_trunc_Fneg #1.#2{\expandafter\xint_firstoftwo_thenstop
813   \romannumeral0\XINT_div_prepare {#2}{#1}\Z \xint_minus_thenstop}%
814 \def\XINT_trunc_Fpos #1.#2{\expandafter\xint_firstoftwo_thenstop
815   \romannumeral0\XINT_div_prepare {#2}{#1}\Z \space }%
816 \def\XINT_itrunc_G #1#2\Z #3#4%
817 {%
818   \xint_gob_til_zero #1\XINT_trunc_zero 0#3#1#2%
819 }%
820 \def\XINT_trunc_zero 0#1#20{ 0}%
821 \def\XINT_trunc_G #1\Z #2#3%
822 {%
823   \xint_gob_til_zero #2\XINT_trunc_zero 0%
824   \expandafter\XINT_trunc_H\expandafter
825   {\the\numexpr\romannumeral0\xintlength {#1}-#3}{#3}{#1}#2%
826 }%
827 \def\XINT_trunc_H #1#2%
828 {%
829   \ifnum #1 > \xint_c_
830     \xint_afterfi {\XINT_trunc_Ha {#2}}%
831   \else
832     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
833   \fi
834 }%
835 \def\XINT_trunc_Ha
836 {%
837   \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
838 }%
839 \def\XINT_trunc_Haa #1#2#3%
840 {%
841   #3#1.#2%
842 }%
843 \def\XINT_trunc_Hb #1#2#3%
844 {%
845   \expandafter #3\expandafter0\expandafter.%
846   \romannumeral0\XINT_dsx_zero loop {#1}{}\Z {}#2% #1=-0 autoris'e !
847 }%
```

### 7.29 *\xintTTrunc*

1.1, a tiny bit more efficient than doing `\xintiTrunc0`. I map `\xintNum` to it, and I use it in `\xintexpr` for various things. Faster I guess than the `\xintiFloor`.

```
848 \def\xintTTrunc {\romannumeral0\xintttrunc }%
849 \def\xintttrunc #1%
850 {%
851   \expandafter\XINT_itrunc_G
```

```

852 \romannumeral0\expandafter\XINT_ttrunc_A
853 \romannumeral0\XINT_infrac {#1}0% this last 0 to let \XINT_itrunc_G be happy
854 }%
855 \def\XINT_ttrunc_A #1#2#3{\XINT_trunc_checkifzero {#1}#2\Z {#3}}%

```

### 7.30 `\xintNum`

This extension of the xint original `xintNum` is added in 1.05, as a synonym to `\xintIrr`, but raising an error when the input does not evaluate to an integer. Usable with not too much overhead on integer input as `\xintIrr` checks quickly for a denominator equal to 1 (which will be put there by the `\XINT_infrac` called by `\xintrawwithzeros`). This way, macros such as `\xintQuo` can be modified with minimal overhead to accept fractional input as long as it evaluates to an integer.

22 june 2014 (dev 1.1) I just don't understand what was the point of going through `\xintIrr` if to raise an error afterwards... and raising errors is silly, so let's do it sanely at last. In between I added `\xintiFloor`, thus, let's just let it to it.

24 october 2014 (final 1.1) (I left it taking dust since June...), I did `\xintTTrunc`, and will thus map `\xintNum` to it

```

856 \let\xintNum \xintTTrunc
857 \let\xintnum \xintttrunc

```

### 7.31 `\xintRound`, `\xintiRound`

Modified in 1.06 to give the first argument to a `\numexpr`.

```

858 \def\xintRound {\romannumeral0\xintround }%
859 \def\xintiRound {\romannumeral0\xintiround }%
860 \def\xintround #1%
861 {%
862   \expandafter\XINT_round\expandafter {\the\numexpr #1}%
863 }%
864 \def\XINT_round
865 {%
866   \expandafter\XINT_trunc_G\romannumeral0\XINT_round_A
867 }%
868 \def\xintiround #1%
869 {%
870   \expandafter\XINT_iround\expandafter {\the\numexpr #1}%
871 }%
872 \def\XINT_iround
873 {%
874   \expandafter\XINT_itrunc_G\romannumeral0\XINT_round_A
875 }%
876 \def\XINT_round_A #1#2%
877 {%
878   \expandafter\XINT_round_B
879   \romannumeral0\expandafter\XINT_trunc_A
880   \romannumeral0\XINT_infrac {#2}{\the\numexpr #1+1\relax}{#1}%
881 }%
882 \def\XINT_round_B #1\Z
883 {%
884   \expandafter\XINT_round_C
885   \romannumeral0\XINT_rord_main {}#1%

```

```

886     \xint_relax
887     \xint_bye\xint_bye\xint_bye\xint_bye
888     \xint_bye\xint_bye\xint_bye\xint_bye
889     \xint_relax
890     \Z
891 }%
892 \def\xINT_round_C #1%
893 {%
894     \ifnum #1<5
895         \expandafter\xINT_round_Daa
896     \else
897         \expandafter\xINT_round_Dba
898     \fi
899 }%
900 \def\xINT_round_Daa #1%
901 {%
902     \xint_gob_til_Z #1\xINT_round_Daz\Z \XINT_round_Da #1%
903 }%
904 \def\xINT_round_Daz\Z \XINT_round_Da \Z { 0\Z }%
905 \def\xINT_round_Da #1\Z
906 {%
907     \XINT_rord_main {}#1%
908     \xint_relax
909     \xint_bye\xint_bye\xint_bye\xint_bye
910     \xint_bye\xint_bye\xint_bye\xint_bye
911     \xint_relax \Z
912 }%
913 \def\xINT_round_Dba #1%
914 {%
915     \xint_gob_til_Z #1\xINT_round_Dbz\Z \XINT_round_Db #1%
916 }%
917 \def\xINT_round_Dbz\Z \XINT_round_Db \Z { 1\Z }%
918 \def\xINT_round_Db #1\Z
919 {%
920     \XINT_addm_A 0{}1000\W\X\Y\Z #1000\W\X\Y\Z \Z
921 }%

```

### 7.32 `\xintXTrunc`

1.09j [2014/01/06] This is completely expandable but not f-expandable. Designed be used inside an `\edef` or a `\write`, if one is interested in getting tens of thousands of digits from the decimal expansion of some fraction... it is not worth using it rather than `\xintTrunc` if for less than \*hundreds\* of digits. For efficiency it clones part of the preparatory division macros, as the same denominator will be used again and again. The D parameter which says how many digits to keep after decimal mark must be at least 1 (and it is forcefully set to such a value if found negative or zero, to avoid an eternal loop).

For reasons of efficiency I try to use the shortest possible denominator, so if the fraction is  $A/B[N]$ , I want to use B. For N at least zero, just immediately replace A by  $A \cdot 10^N$ . The first division then may be a little longish but the next ones will be fast (if B is not too big). For  $N < 0$ , this is a bit more complicated. I thought somewhat about this, and I would need a rather complicated approach going through a long division algorithm, forcing me to essentially clone the actual division with some differences; a side thing is that as this would use blocks of four digits I would have a hard time allowing a non-multiple of four number of post decimal mark digits.

7 Package *xintfrac* implementation

Thus, for  $N < 0$ , another method is followed. First the euclidean division  $A/B = Q + R/B$  is done. The number of digits of  $Q$  is  $M$ . If  $|N| \leq D$ , we launch inside a `\csname` the routine for obtaining  $D - |N|$  next digits (this may impact TeX's memory if  $D$  is very big), call them  $T$ . We then need to position the decimal mark  $D$  slots from the right of  $QT$ , which has length  $M + D - |N|$ , hence  $|N|$  slots from the right of  $Q$ . We thus avoid having to work with the  $T$ , as  $D$  may be very very big (`\xintXTrunc`'s only goal is to make it possible to learn by hearts decimal expansions with thousands of digits). We can use the `\xintDecSplit` for that on  $Q$ . Computing the length  $M$  of  $Q$  was a more or less unavoidable step. If  $|N| > D$ , the `\csname` step is skipped we need to remove the  $D - |N|$  last digits from  $Q$ , etc.. we compare  $D - |N|$  with the length  $M$  of  $Q$  etc... (well in this last, very uncommon, branch, I stopped trying to optimize things and I even do an `\xintnum` to ensure a 0 if something comes out empty from `\xintDecSplit`).

```

922 \def\xintXTrunc #1#2%
923 {%
924   \expandafter\XINT_xtrunc_a\expandafter
925   {\the\numexpr #1\expandafter}\romannumeral0\xintra {#2}%
926 }%
927 \def\XINT_xtrunc_a #1%
928 {%
929   \expandafter\XINT_xtrunc_b\expandafter
930   {\the\numexpr \ifnum#1<\xint_c_i \xint_c_i-\fi #1}%
931 }%
932 \def\XINT_xtrunc_b #1%
933 {%
934   \expandafter\XINT_xtrunc_c\expandafter
935   {\the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i}{#1}%
936 }%
937 \def\XINT_xtrunc_c #1#2%
938 {%
939   \expandafter\XINT_xtrunc_d\expandafter
940   {\the\numexpr #2-\xint_c_ii^vi*#1}{#1}{#2}%
941 }%
942 \def\XINT_xtrunc_d #1#2#3#4/#5[#6]%
943 {%
944   \XINT_xtrunc_e #4.{#6}{#5}{#3}{#2}{#1}%
945 }%
946 % #1=numerator.#2=N,#3=B,#4=D,#5=Blocs,#6=extra
947 \def\XINT_xtrunc_e #1%
948 {%
949   \xint_UDzerominusfork
950     #1-\XINT_xtrunc_zero
951     0#1\XINT_xtrunc_N
952     0-\{XINT_xtrunc_P #1}%
953   \krof
954 }%
955 \def\XINT_xtrunc_zero .#1#2#3#4#5%
956 {%
957   0.\romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
958     {\the\numexpr #5}{Z }%
959   \xintiloop [#4+-1]
960   \ifnum \xintiloopindex>\xint_c_
961     0000000000000000000000000000000000000000000000000000000000000000%
962   \repeat

```

## 7 Package *xintfrac* implementation

```

963 }%
964 \def\XINT_xtrunc_N {-\XINT_xtrunc_P }%
965 \def\XINT_xtrunc_P #1.#2%
966 {%
967   \ifnum #2<\xint_c_
968     \expandafter\XINT_xtrunc_negN_Q
969   \else
970     \expandafter\XINT_xtrunc_Q
971   \fi {#2}{#1}.%
972 }%
973 \def\XINT_xtrunc_negN_Q #1#2.#3#4#5#6%
974 {%
975   \expandafter\XINT_xtrunc_negN_R
976   \romannumeral0\XINT_div_prepare {#3}{#2}{#3}{#1}{#4}%
977 }%
978 % #1=Q, #2=R, #3=B, #4=N<0, #5=D
979 \def\XINT_xtrunc_negN_R #1#2#3#4#5%
980 {%
981   \expandafter\XINT_xtrunc_negN_S\expandafter
982   {\the\numexpr -#4}{#5}{#2}{#3}{#1}%
983 }%
984 \def\XINT_xtrunc_negN_S #1#2%
985 {%
986   \expandafter\XINT_xtrunc_negN_T\expandafter
987   {\the\numexpr #2-#1}{#1}{#2}%
988 }%
989 \def\XINT_xtrunc_negN_T #1%
990 {%
991   \ifnum \xint_c_<#1
992     \expandafter\XINT_xtrunc_negNA
993   \else
994     \expandafter\XINT_xtrunc_negNW
995   \fi {#1}%
996 }%
997 % #1=D-|N|>0, #2=|N|, #3=D, #4=R, #5=B, #6=Q
998 \def\XINT_xtrunc_unlock #10.{ }%
999 \def\XINT_xtrunc_negNA #1#2#3#4#5#6%
1000 {%
1001   \expandafter\XINT_xtrunc_negNB\expandafter
1002   {\romannumeral0\expandafter\expandafter\expandafter
1003   \XINT_xtrunc_unlock\expandafter\string
1004   \cscname\XINT_xtrunc_b {#1}#4/#5[0]\expandafter\endcscname
1005   \expandafter}\expandafter
1006   {\the\numexpr\xintLength{#6}-#2}{#6}%
1007 }%
1008 \def\XINT_xtrunc_negNB #1#2#3{\XINT_xtrunc_negNC {#2}{#3}{#1}%
1009 \def\XINT_xtrunc_negNC #1%
1010 {%
1011   \ifnum \xint_c_ < #1
1012     \expandafter\XINT_xtrunc_negNDa
1013   \else
1014     \expandafter\XINT_xtrunc_negNE

```

## 7 Package *xintfrac* implementation

```

1015   \fi {#1}%
1016 }%
1017 \def\XINT_xtrunc_negNDa #1#2%
1018 {%
1019   \expandafter\XINT_xtrunc_negNDb%
1020   \romannumeral0\XINT_split_fromleft_loop {#1}{#2\W\W\W\W\W\W\W\W\Z
1021 }%
1022 \def\XINT_xtrunc_negNDb #1#2{#1.#2}%
1023 \def\XINT_xtrunc_negNE #1#2%
1024 {%
1025   0.\romannumeral0\XINT_dsx_zeroloop {-#1}{\Z }{#2}%
1026 }%
1027 % #1=D-|N|<=0, #2=|N|, #3=D, #4=R, #5=B, #6=Q
1028 \def\XINT_xtrunc_negNW #1#2#3#4#5#6%
1029 {%
1030   \expandafter\XINT_xtrunc_negNX\expandafter
1031   {\romannumeral0\xintnum{\xintDecSplitL {-#1}{#6}}}{#3}%
1032 }%
1033 \def\XINT_xtrunc_negNX #1#2%
1034 {%
1035   \expandafter\XINT_xtrunc_negNC\expandafter
1036   {\the\numexpr\xintLength {#1}-#2}{#1}%
1037 }%
1038 \def\XINT_xtrunc_Q #1%
1039 {%
1040   \expandafter\XINT_xtrunc_prepare_I
1041   \romannumeral0\XINT_dsx_zeroloop {#1}{\Z
1042 }%
1043 \def\XINT_xtrunc_prepare_I #1.#2#3%
1044 {%
1045   \expandafter\XINT_xtrunc_prepareB_aa\expandafter
1046   {\romannumeral0\xintlength {#2}}{#2}{#1}%
1047 }%
1048 \def\XINT_xtrunc_prepareB_aa #1%
1049 {%
1050   \ifnum #1=\xint_c_i
1051     \expandafter\XINT_xtrunc_prepareB_onedigit
1052   \else
1053     \expandafter\XINT_xtrunc_prepareB_PaBa
1054   \fi
1055   {#1}%
1056 }%
1057 \def\XINT_xtrunc_prepareB_onedigit #1#2%
1058 {%
1059   \ifcase#2
1060   \or\expandafter\XINT_xtrunc_BisOne
1061   \or\expandafter\XINT_xtrunc_BisTwo
1062   \else\expandafter\XINT_xtrunc_prepareB_PaBe
1063   \fi {000}{0}{4}{#2}%
1064 }%
1065 \def\XINT_xtrunc_BisOne #1#2#3#4#5#6#7%
1066 {%

```







```

1166 \mathchardef\XINTdigits 16
1167 \def\xintDigits #1#2%
1168   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}
1169 \def\xinttheDigits {\number\XINTdigits }

```

### 7.34 `\xintFloat`

1.07. Completely re-written in 1.08a, with spectacular speed gains. The earlier version was seriously silly when dealing with inputs having a big power of ten. Again some modifications in 1.08b for a better treatment of cases with long explicit numerators or denominators.

Here again some inner macros used the `\xintiquo` with extra `\xintnum` overhead in 1.09a, 1.09f reinstalled use of `\xintiquo` without this overhead.

```

1170 \def\xintFloat   {\romannumeral0\xintfloat }
1171 \def\xintfloat #1{\XINT_float_chkopt #1\xint_relax }
1172 \def\XINT_float_chkopt #1%
1173 {%
1174   \ifx [#1\expandafter\XINT_float_opt
1175     \else\expandafter\XINT_float_noopt
1176   \fi #1%
1177 }%
1178 \def\XINT_float_noopt #1\xint_relax
1179 {%
1180   \expandafter\XINT_float_a\expandafter\XINTdigits
1181   \romannumeral0\XINT_infrac {#1}\XINT_float_Q
1182 }%
1183 \def\XINT_float_opt [\xint_relax #1]#2%
1184 {%
1185   \expandafter\XINT_float_a\expandafter
1186   {\the\numexpr #1\expandafter}%
1187   \romannumeral0\XINT_infrac {#2}\XINT_float_Q
1188 }%
1189 \def\XINT_float_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1190 {%
1191   \XINT_float_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1192 }%
1193 \def\XINT_float_fork #1%
1194 {%
1195   \xint_UDzerominusfork
1196   #1-\XINT_float_zero
1197   0#1\XINT_float_J
1198   0-{\XINT_float_K #1}%
1199   \krof
1200 }%
1201 \def\XINT_float_zero #1\Z #2#3#4#5{ 0.e0}%
1202 \def\XINT_float_J {\expandafter\xint_minus_thenstop\romannumeral0\XINT_float_K }
1203 \def\XINT_float_K #1\Z #2% #1=A, #2=P, #3=n, #4=B
1204 {%
1205   \expandafter\XINT_float_L\expandafter
1206   {\the\numexpr\xintLength{#1}\expandafter}\expandafter
1207   {\the\numexpr #2+\xint_c_ii}{#1}{#2}%
1208 }%
1209 \def\XINT_float_L #1#2%

```

7 Package *xintfrac* implementation

```

1210 {%
1211   \ifnum #1>#2
1212     \expandafter\XINT_float_Ma
1213   \else
1214     \expandafter\XINT_float_Mc
1215   \fi {#1}{#2}%
1216 }%
1217 \def\XINT_float_Ma #1#2#3%
1218 {%
1219   \expandafter\XINT_float_Mb\expandafter
1220   {\the\numexpr #1-#2\expandafter\expandafter\expandafter}%
1221   \expandafter\expandafter\expandafter
1222   {\expandafter\xint_firstoftwo
1223     \romannumeral0\XINT_split_fromleft_loop {#2}{#3}\W\W\W\W\W\W\W\W\Z
1224     }{#2}%
1225 }%
1226 \def\XINT_float_Mb #1#2#3#4#5#6% #2=A', #3=P+2, #4=P, #5=n, #6=B
1227 {%
1228   \expandafter\XINT_float_N\expandafter
1229   {\the\numexpr\xintLength{#6}\expandafter}\expandafter
1230   {\the\numexpr #3\expandafter}\expandafter
1231   {\the\numexpr #1+#5}%
1232   {#6}{#3}{#2}{#4}%
1233 }% long de B, P+2, n', B, |A'|=P+2, A', P
1234 \def\XINT_float_Mc #1#2#3#4#5#6%
1235 {%
1236   \expandafter\XINT_float_N\expandafter
1237   {\romannumeral0\xintlength{#6}}{#2}{#5}{#6}{#1}{#3}{#4}%
1238 }% long de B, P+2, n, B, |A|, A, P
1239 \def\XINT_float_N #1#2%
1240 {%
1241   \ifnum #1>#2
1242     \expandafter\XINT_float_O
1243   \else
1244     \expandafter\XINT_float_P
1245   \fi {#1}{#2}%
1246 }%
1247 \def\XINT_float_O #1#2#3#4%
1248 {%
1249   \expandafter\XINT_float_P\expandafter
1250   {\the\numexpr #2\expandafter}\expandafter
1251   {\the\numexpr #2\expandafter}\expandafter
1252   {\the\numexpr #3-#1+#2\expandafter\expandafter\expandafter}%
1253   \expandafter\expandafter\expandafter
1254   {\expandafter\xint_firstoftwo
1255     \romannumeral0\XINT_split_fromleft_loop {#2}{#4}\W\W\W\W\W\W\W\W\Z
1256     }%
1257 }% |B|,P+2,n,B,|A|,A,P
1258 \def\XINT_float_P #1#2#3#4#5#6#7#8%
1259 {%
1260   \expandafter #8\expandafter {\the\numexpr #1-#5+#2-\xint_c_i}%
1261   {#6}{#4}{#7}{#3}%

```

## 7 Package *xintfrac* implementation

```

1262 }% |B|-|A|+P+1,A,B,P,n
1263 \def\XINT_float_Q #1%
1264 {%
1265   \ifnum #1<\xint_c_
1266     \expandafter\XINT_float_Ri
1267   \else
1268     \expandafter\XINT_float_Rii
1269   \fi #1}%
1270 }%
1271 \def\XINT_float_Ri #1#2#3%
1272 {%
1273   \expandafter\XINT_float_Sa
1274   \romannumeral0\xintiigo {#2}%
1275   {\XINT_dsx_addzerosnofuss {-#1}{#3}}\Z {#1}%
1276 }%
1277 \def\XINT_float_Rii #1#2#3%
1278 {%
1279   \expandafter\XINT_float_Sa
1280   \romannumeral0\xintiigo
1281   {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
1282 }%
1283 \def\XINT_float_Sa #1%
1284 {%
1285   \if #19%
1286     \xint_afterfi {\XINT_float_Sb\XINT_float_Wb }%
1287   \else
1288     \xint_afterfi {\XINT_float_Sb\XINT_float_Wa }%
1289   \fi #1%
1290 }%
1291 \def\XINT_float_Sb #1#2\Z #3#4%
1292 {%
1293   \expandafter\XINT_float_T\expandafter
1294   {\the\numexpr #4+\xint_c_i\expandafter}%
1295   \romannumeral-\`0\XINT_lenrord_loop 0{#2}\Z\W\W\W\W\W\W\W\W\Z #1{#3}{#4}%
1296 }%
1297 \def\XINT_float_T #1#2#3%
1298 {%
1299   \ifnum #2>#1
1300     \xint_afterfi{\XINT_float_U\XINT_float_Xb}%
1301   \else
1302     \xint_afterfi{\XINT_float_U\XINT_float_Xa #3}%
1303   \fi
1304 }%
1305 \def\XINT_float_U #1#2%
1306 {%
1307   \ifnum #2<\xint_c_v
1308     \expandafter\XINT_float_Va
1309   \else
1310     \expandafter\XINT_float_Vb
1311   \fi #1%
1312 }%
1313 \def\XINT_float_Va #1#2\Z #3%

```

```

1314 {%
1315   \expandafter#1%
1316   \romannumeral0\expandafter\XINT_float_Wa
1317   \romannumeral0\XINT_rord_main {}#2%
1318   \xint_relax
1319   \xint_bye\xint_bye\xint_bye\xint_bye
1320   \xint_bye\xint_bye\xint_bye\xint_bye
1321   \xint_relax \Z
1322 }%
1323 \def\XINT_float_Vb #1#2\Z #3%
1324 {%
1325   \expandafter #1%
1326   \romannumeral0\expandafter #3%
1327   \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1328 }%
1329 \def\XINT_float_Wa #1{ #1.}%
1330 \def\XINT_float_Wb #1#2%
1331   {\if #11\xint_afterfi{ 10.}\else\xint_afterfi{ #1.#2}\fi }%
1332 \def\XINT_float_Xa #1\Z #2#3#4%
1333 {%
1334   \expandafter\XINT_float_Y\expandafter
1335   {\the\numexpr #3+#4-#2}{#1}%
1336 }%
1337 \def\XINT_float_Xb #1\Z #2#3#4%
1338 {%
1339   \expandafter\XINT_float_Y\expandafter
1340   {\the\numexpr #3+#4+\xint_c_i-#2}{#1}%
1341 }%
1342 \def\XINT_float_Y #1#2{ #2e#1}%

```

## 7.35 *\xintPFloat*

### 1.1

```

1343 \def\xintPFloat {\romannumeral0\xintpfloat }%
1344 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
1345 \def\XINT_pfloat_chkopt #1%
1346 {%
1347   \ifx [#1\expandafter\XINT_pfloat_opt
1348     \else\expandafter\XINT_pfloat_noopt
1349   \fi #1%
1350 }%
1351 \def\XINT_pfloat_noopt #1\xint_relax
1352 {%
1353   \expandafter\XINT_pfloat_a\expandafter\XINTdigits
1354   \romannumeral0\XINTinfloat [\XINTdigits]{#1}%
1355 }%
1356 \def\XINT_pfloat_opt [\xint_relax #1]#2%
1357 {%
1358   \expandafter\XINT_pfloat_a\expandafter {\the\numexpr #1\expandafter}%
1359   \romannumeral0\XINTinfloat [\numexpr #1\relax]#2}%
1360 }%
1361 \def\XINT_pfloat_a #1#2%

```

```

1362 {%
1363   \xint_UDzerominusfork
1364           #2-\XINT_pfloat_zero
1365           0#2\XINT_pfloat_neg
1366           0-\XINT_pfloat_pos #2}%
1367   \krof {#1}%
1368 }%
1369 \def\XINT_pfloat_zero #1[#2]{ 0}%
1370 \def\XINT_pfloat_neg
1371   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_pfloat_pos {}}%
1372 \def\XINT_pfloat_pos #1#2#3[#4]%
1373 {%
1374   \ifnum#4>0 \xint_dothis\XINT_pfloat_no\fi
1375   \ifnum#4>\numexpr-#2\relax \xint_dothis\XINT_pfloat_b\fi
1376   \ifnum#4>\numexpr-#2-\xint_c_v\relax \xint_dothis\XINT_pfloat_B\fi
1377   \xint_orthat\XINT_pfloat_no {#2}{#4}{#1#3}%
1378 }%
1379 \def\XINT_pfloat_no #1#2%
1380 {%
1381   \expandafter\XINT_pfloat_no_b\expandafter{\the\numexpr #2+#1-\xint_c_i\relax}%
1382 }%
1383 \def\XINT_pfloat_no_b #1#2{\XINT_pfloat_no_c #2e#1}%
1384 \def\XINT_pfloat_no_c #1{ #1.}%
1385 \def\XINT_pfloat_b #1#2#3%
1386   {\expandafter\XINT_pfloat_c
1387     \romannumeral0\expandafter\XINT_split_fromleft_loop
1388     \expandafter {\the\numexpr #1+#2-\xint_c_i}#3\W\W\W\W\W\W\W\W\Z }%
1389 \def\XINT_pfloat_c #1#2{ #1.#2}% #2 peut \^etre vide
1390 \def\XINT_pfloat_B #1#2#3%
1391   {\expandafter\XINT_pfloat_C
1392     \romannumeral0\XINT_dsx_zeroloop {\numexpr -#1-#2}{}\Z { }#3}%
1393 \def\XINT_pfloat_C { 0.}%

```

### 7.36 \XINTinFloat

1.07. Completely rewritten in 1.08a for immensely greater efficiency when the power of ten is big: previous version had some very serious bottlenecks arising from the creation of long strings of zeros, which made things such as  $2^{999999}$  completely impossible, but now even  $2^{999999999}$  with 24 significant digits is no problem! Again (slightly) improved in 1.08b.

I decide in 1.09a not to use anymore `\romannumeral`-0` mais `\romannumeral0` also in the float routines, for consistency of style.

Here again some inner macros used the `\xintiquo` with extra `\xintnum` overhead in 1.09a, 1.09f fixed that to use `\xintiiquo` for example.

1.09i added a stupid bug to `\XINT_infloat_zero` when it changed `0[0]` to a silly `0/1[0]`, breaking in particular `\xintFloatAdd` when one of the argument is zero :(((

1.09j fixes this. Besides, for notational coherence `\XINT_inFloat` and `\XINT_infloat` have been renamed respectively `\XINTinFloat` and `\XINTinfloat` in release 1.09j.

```

1394 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1395 \def\XINTinfloat [#1]#2%
1396 {%
1397   \expandafter\XINT_infloat_a\expandafter
1398   {\the\numexpr #1\expandafter}%

```

## 7 Package *xintfrac* implementation

```

1399 \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
1400 }%
1401 \def\XINT_infloat_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1402 {%
1403 \XINT_infloat_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1404 }%
1405 \def\XINT_infloat_fork #1%
1406 {%
1407 \xint_UDzerominusfork
1408 #1-\XINT_infloat_zero
1409 0#1\XINT_infloat_J
1410 0-\XINT_float_K #1}%
1411 \krof
1412 }%
1413 \def\XINT_infloat_zero #1\Z #2#3#4#5{ 0[0]}%
1414 % the 0[0] was stupidly changed to 0/1[0] in 1.09i, with the result that the
1415 % Float addition would crash when an operand was zero
1416 \def\XINT_infloat_J {\expandafter-\romannumeral0\XINT_float_K }%
1417 \def\XINT_infloat_Q #1%
1418 {%
1419 \ifnum #1<\xint_c_
1420 \expandafter\XINT_infloat_Ri
1421 \else
1422 \expandafter\XINT_infloat_Rii
1423 \fi {#1}%
1424 }%
1425 \def\XINT_infloat_Ri #1#2#3%
1426 {%
1427 \expandafter\XINT_infloat_S\expandafter
1428 {\romannumeral0\xintiiquo {#2}%
1429 {\XINT_dsx_addzerosnofuss {-#1}{#3}}{#1}%
1430 }%
1431 \def\XINT_infloat_Rii #1#2#3%
1432 {%
1433 \expandafter\XINT_infloat_S\expandafter
1434 {\romannumeral0\xintiiquo
1435 {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}}{#1}%
1436 }%
1437 \def\XINT_infloat_S #1#2#3%
1438 {%
1439 \expandafter\XINT_infloat_T\expandafter
1440 {\the\numexpr #3+\xint_c_i\expandafter}%
1441 \romannumeral-\`0\XINT_lenrord_loop 0{#1\Z\W\W\W\W\W\W\W\Z
1442 {#2}%
1443 }%
1444 \def\XINT_infloat_T #1#2#3%
1445 {%
1446 \ifnum #2>#1
1447 \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wb}%
1448 \else
1449 \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wa #3}%
1450 \fi

```



```

1451 }%
1452 \def\XINT_infloat_U #1#2%
1453 {%
1454   \ifnum #2<\xint_c_v
1455     \expandafter\XINT_infloat_Va
1456   \else
1457     \expandafter\XINT_infloat_Vb
1458   \fi #1%
1459 }%
1460 \def\XINT_infloat_Va #1#2\Z
1461 {%
1462   \expandafter#1%
1463   \romannumeral0\XINT_rord_main {}#2%
1464   \xint_relax
1465   \xint_bye\xint_bye\xint_bye\xint_bye
1466   \xint_bye\xint_bye\xint_bye\xint_bye
1467   \xint_relax \Z
1468 }%
1469 \def\XINT_infloat_Vb #1#2\Z
1470 {%
1471   \expandafter #1%
1472   \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1473 }%
1474 \def\XINT_infloat_Wa #1\Z #2#3%
1475 {%
1476   \expandafter\XINT_infloat_X\expandafter
1477   {\the\numexpr #3+\xint_c_i-#2}{#1}%
1478 }%
1479 \def\XINT_infloat_Wb #1\Z #2#3%
1480 {%
1481   \expandafter\XINT_infloat_X\expandafter
1482   {\the\numexpr #3+\xint_c_ii-#2}{#1}%
1483 }%
1484 \def\XINT_infloat_X #1#2{ #2[#1]}%

```

### 7.37 `\xintAdd`

modified in v1.1. Et aussi 25 juin pour intercepter `summand nul`.

```

1485 \def\xintAdd {\romannumeral0\xintadd }%
1486 \def\xintadd #1{\expandafter\xint_fadd\romannumeral0\xintraw {#1}}%
1487 \def\xint_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1488 \def\XINT_fadd_Azero #1]{\xintraw }%
1489 \def\XINT_fadd_a #1/#2[#3]#4%
1490   {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1491 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1492 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1493 \def\XINT_fadd_c #1/#2[#3]#4%
1494 {%
1495   \expandafter\XINT_fadd_Aa\expandafter{\the\numexpr #4-#3}{#3}{#4}{#1}{#2}%
1496 }%
1497 \def\XINT_fadd_Aa #1%
1498 {%

```

## 7 Package *xintfrac* implementation

```

1499 \ifcase\XINT_cntSgn #1\Z
1500 \expandafter\XINT_fadd_B
1501 \or
1502 \expandafter \XINT_fadd_Ba
1503 \else
1504 \expandafter \XINT_fadd_Bb
1505 \fi {#1}%
1506 }%
1507 \def\XINT_fadd_B #1#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1508 \def\XINT_fadd_Ba #1#2#3#4#5#6#7%
1509 {%
1510 \expandafter\XINT_fadd_C\expandafter
1511 {\romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {#6}}%
1512 {#7}{#5}{#4}[#2]}%
1513 }%
1514 \def\XINT_fadd_Bb #1#2#3#4#5#6#7%
1515 {%
1516 \expandafter\XINT_fadd_C\expandafter
1517 {\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {#4}}%
1518 {#5}{#7}{#6}[#3]}%
1519 }%
1520 \def\XINT_fadd_C #1#2#3%
1521 {%
1522 \ifcase\romannumeral0\XINT_cmp_pre {#2}{#3} %<- intentional space here.
1523 \expandafter\XINT_fadd_eq
1524 \or\expandafter\XINT_fadd_D
1525 \else\expandafter\XINT_fadd_Da
1526 \fi {#2}{#3}{#1}%
1527 }%
1528 \def\XINT_fadd_eq #1#2#3#4#5%
1529 {%
1530 \expandafter\XINT_fadd_G
1531 \romannumeral0\xintiiadd {#3}{#4}/#1[#5]}%
1532 }%
1533 \def\XINT_fadd_D #1#2%
1534 {%
1535 \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
1536 }%
1537 \def\XINT_fadd_E #1#2%
1538 {%
1539 \if0\XINT_Sgn #2\Z
1540 \expandafter\XINT_fadd_F
1541 \else\expandafter\XINT_fadd_K
1542 \fi {#1}%
1543 }%
1544 \def\XINT_fadd_F #1#2#3#4#5#6%
1545 {%
1546 \expandafter\XINT_fadd_G
1547 \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2[#6]}%
1548 }%
1549 \def\XINT_fadd_Da #1#2%
1550 {%

```

## 7 Package *xintfrac* implementation

```
1551 \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
1552 }%
1553 \def\XINT_fadd_Ea #1#2%
1554 {%
1555 \if0\XINT_Sgn #2\Z
1556 \expandafter\XINT_fadd_Fa
1557 \else\expandafter\XINT_fadd_K
1558 \fi {#1}%
1559 }%
1560 \def\XINT_fadd_Fa #1#2#3#4#5#6%
1561 {%
1562 \expandafter\XINT_fadd_G
1563 \romannumeral0\xintiiaadd {\xintiiMul {#4}{#1}}{#5}/#3%[#6]%
1564 }%
1565 \def\XINT_fadd_G #1{\if0#1\XINT_fadd_iszero\fi\space #1}%
1566 \def\XINT_fadd_K #1#2#3#4#5%
1567 {%
1568 \expandafter\XINT_fadd_L
1569 \romannumeral0\xintiiaadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1570 {{#2}{#3}}%
1571 }%
1572 \def\XINT_fadd_L #1{\if0#1\XINT_fadd_iszero\fi \XINT_fadd_M #1}%
1573 \def\XINT_fadd_M #1.#2{\expandafter\XINT_fadd_N \expandafter
1574 \romannumeral0\xintiimul #2}{#1}%
1575 \def\XINT_fadd_N #1#2{ #2/#1}%
1576 \edef\XINT_fadd_iszero\fi #1[#2]{\noexpand\fi\space 0/1[0]}% ou [2] originel?
```

### 7.38 *\xintSub*

refait dans 1.1 pour vérifier si summands nuls.

```
1577 \def\xintSub {\romannumeral0\xintsub }%
1578 \def\xintsub #1{\expandafter\xint_fsub\romannumeral0\xintraff {#1}}%
1579 \def\xint_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1580 \def\XINT_fsub_Azero #1]{\xintopp }%
1581 \def\XINT_fsub_a #1/#2[#3]#4%
1582 {\expandafter\XINT_fsub_b\romannumeral0\xintraff {#4}{#3}{#1}{#2}}%
1583 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1584 #1-\XINT_fadd_Bzero
1585 0#1\XINT_fadd_c
1586 0-{\XINT_fadd_c -#1}%
1587 \krof }%
```

### 7.39 *\xintSum*

```
1588 \def\xintSum {\romannumeral0\xintsum }%
1589 \def\xintsum #1{\xintsumexpr #1\relax }%
1590 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1591 \def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral-`0}%
1592 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1593 \def\XINT_fsum_loop_a #1#2%
1594 {%
1595 \expandafter\XINT_fsum_loop_b \romannumeral-`0#2\Z {#1}%

```

```

1596 }%
1597 \def\XINT_fsum_loop_b #1%
1598 {%
1599     \xint_gob_til_relax #1\XINT_fsum_finished\relax
1600     \XINT_fsum_loop_c #1%
1601 }%
1602 \def\XINT_fsum_loop_c #1\Z #2%
1603 {%
1604     \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1605 }%
1606 \def\XINT_fsum_finished #1\Z #2{ #2}%

```

## 7.40 \xintMul

modif 1.1 25-juin-14 pour vérifier plus tôt si nul

```

1607 \def\xintMul {\romannumeral0\xintmul }%
1608 \def\xintmul #1{\expandafter\xint_fmulo\romannumeral0\xintraw {#1}.}%
1609 \def\xint_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_a #1}%
1610 \def\XINT_fmulo_a #1[#2].#3%
1611     {\expandafter\XINT_fmulo_b\romannumeral0\xintraw {#3}#1[#2.]}%
1612 \def\XINT_fmulo_b #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1613 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]%
1614 {%
1615     \expandafter\XINT_fmulo_d
1616     \expandafter{\the\numexpr #3+#6\expandafter}%
1617     \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1618     {\romannumeral0\xintiimul {#4}{#1}}%
1619 }%
1620 \def\XINT_fmulo_d #1#2#3%
1621 {%
1622     \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1623 }%
1624 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1625 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

## 7.41 \xintSqr

1.1 modifs comme xintMul

```

1626 \def\xintSqr {\romannumeral0\xintsqr }%
1627 \def\xintsqr #1{\expandafter\xint_fsqr\romannumeral0\xintraw {#1}}%
1628 \def\xint_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1629 \def\xint_fsqr_a #1/#2[#3]%
1630 {%
1631     \expandafter\XINT_fsqr_b
1632     \expandafter{\the\numexpr #3+#3\expandafter}%
1633     \expandafter{\romannumeral0\xintiisqr {#2}}%
1634     {\romannumeral0\xintiisqr {#1}}%
1635 }%
1636 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1637 \def\XINT_fsqr_zero #1{ 0/1[0]}%

```

## 7.42 `\xintPow`

Modified in 1.06 to give the exponent to a `\numexpr`.

With 1.07 and for use within the `\xintexpr` parser, we must allow fractions (which are integers in disguise) as input to the exponent, so we must have a variant which uses `\xintNum` and not only `\numexpr` for normalizing the input. Hence the `\xintfPow` here.

1.08b: well actually I think that with `xintfrac.sty` loaded the exponent should always be allowed to be a fraction giving an integer. So I do as for `\xintFac`, and remove here the duplicated. Then `\xintexpr` can use the `\xintPow` as defined here.

```

1638 \def\xintPow {\romannumeral0\xintpow }%
1639 \def\xintpow #1%
1640 {%
1641   \expandafter\xint_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1642 }%
1643 \def\xint_fpow #1#2%
1644 {%
1645   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1646 }%
1647 \def\XINT_fpow_fork #1#2\Z
1648 {%
1649   \xint_UDzerominusfork
1650   #1-\XINT_fpow_zero
1651   0#1\XINT_fpow_neg
1652   0-{\XINT_fpow_pos #1}%
1653   \krof
1654   {#2}%
1655 }%
1656 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1657 \def\XINT_fpow_pos #1#2#3#4#5%
1658 {%
1659   \expandafter\XINT_fpow_pos_A\expandafter
1660   {\the\numexpr #1#2*#3\expandafter}\expandafter
1661   {\romannumeral0\xintiipow {#5}{#1#2}}%
1662   {\romannumeral0\xintiipow {#4}{#1#2}}%
1663 }%
1664 \def\XINT_fpow_neg #1#2#3#4%
1665 {%
1666   \expandafter\XINT_fpow_pos_A\expandafter
1667   {\the\numexpr -#1*#2\expandafter}\expandafter
1668   {\romannumeral0\xintiipow {#3}{#1}}%
1669   {\romannumeral0\xintiipow {#4}{#1}}%
1670 }%
1671 \def\XINT_fpow_pos_A #1#2#3%
1672 {%
1673   \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1674 }%
1675 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

## 7.43 `\xintFac`

1.07: to be used by the `\xintexpr` scanner which needs to be able to apply `\xintFac` to a fraction which is an integer in disguise; so we use `\xintNum` and not only `\numexpr`. Je modifie cela dans

## 7 Package *xintfrac* implementation

1.08b, au lieu d'avoir un `\xintfFac` spécialement pour `\xintexpr`, tout simplement j'étends `\xintFac` comme les autres macros, pour qu'elle utilise `\xintNum`.

```
1676 \def\xintFac {\romannumeral0\xintfac }%
1677 \def\xintfac #1%
1678 {%
1679   \expandafter\XINT_fac_fork\expandafter{\the\numexpr \xintNum{#1}}%
1680 }%
```

### 7.44 `\xintPrd`

```
1681 \def\xintPrd {\romannumeral0\xintprd }%
1682 \def\xintprd #1{\xintprdexpr #1\relax }%
1683 \def\xintPrdExpr {\romannumeral0\xintprdexpr }%
1684 \def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral-`0}%
1685 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1686 \def\XINT_fprod_loop_a #1#2%
1687 {%
1688   \expandafter\XINT_fprod_loop_b \romannumeral-`0#2\Z {#1}%
1689 }%
1690 \def\XINT_fprod_loop_b #1%
1691 {%
1692   \xint_gob_til_relax #1\XINT_fprod_finished\relax
1693   \XINT_fprod_loop_c #1%
1694 }%
1695 \def\XINT_fprod_loop_c #1\Z #2%
1696 {%
1697   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1698 }%
1699 \def\XINT_fprod_finished #1\Z #2{ #2}%
```

### 7.45 `\xintDiv`

```
1700 \def\xintDiv {\romannumeral0\xintdiv }%
1701 \def\xintdiv #1%
1702 {%
1703   \expandafter\xint_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1704 }%
1705 \def\xint_fdiv #1#2%
1706   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}{#1}}%
1707 \def\XINT_fdiv_A #1#2#3#4#5#6%
1708 {%
1709   \expandafter\XINT_fdiv_B
1710   \expandafter{\the\numexpr #4-#1\expandafter}%
1711   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1712   {\romannumeral0\xintiimul {#3}{#5}}%
1713 }%
1714 \def\XINT_fdiv_B #1#2#3%
1715 {%
1716   \expandafter\XINT_fdiv_C
1717   \expandafter{#3}{#1}{#2}%
1718 }%
1719 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%
```

## 7.46 `\xintDivFloor`

### 1.1

```
1720 \def\xintDivFloor    {\romannumeral0\xintdivfloor }%
1721 \def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%
```

## 7.47 `\xintDivTrunc`

### 1.1. `\xintttrunc` rather than `\xintitrunc0` in 1.1a

```
1722 \def\xintDivTrunc    {\romannumeral0\xintdivtrunc }%
1723 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

## 7.48 `\xintDivRound`

### 1.1

```
1724 \def\xintDivRound    {\romannumeral0\xintdivround }%
1725 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

## 7.49 `\xintMod`

1.1. `\xintMod {q1}{q2}` computes  $q_2 * t(q_1/q_2)$  with  $t(q_1/q_2)$  equal to the truncated division of two arbitrary fractions  $q_1$  and  $q_2$ . We put some efforts into minimizing the amount of computations.

```
1726 \def\xintMod {\romannumeral0\xintmod }%
1727 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1728 \def\XINT_mod_a #1#2.#3%
1729   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1730 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1731 {%
1732   \if0#2\xint_dothis\XINT_mod_divbyzero\fi
1733   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1734   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1735   \xint_orthat{\XINT_mod_bpos #1#2}%
1736 }%
1737 \def\XINT_mod_bpos #1%
1738 {%
1739   \xint_UDsignfork
1740     #1{\xintiiopp\XINT_mod_pos {}}%
1741     -{\XINT_mod_pos #1}%
1742   \krof
1743 }%
1744 \def\XINT_mod_bneg #1%
1745 {%
1746   \xint_UDsignfork
1747     #1{\xintiiopp\XINT_mod_pos {}}%
1748     -{\XINT_mod_pos #1}%
1749   \krof
1750 }%
1751 \def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
1752 \def\XINT_mod_aiszero #1.{ 0/1[0]}%
```

## 7 Package *xintfrac* implementation

```
1753 \def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
1754 {%
1755   \expandafter\XINT_mod_pos_a
1756   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1757   {\romannumeral0\xintiimul {#6}{#3}}%      n fois u
1758   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}% m fois u
1759   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%   t fois n
1760 }%
1761 \def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%
```

### 7.50 *\XINTinFloatMod*

Pour emploi dans *xintexpr* 1.1

```
1762 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
1763 \def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
1764   {\romannumeral0\XINTinfloat[#1]{#2}}%
1765   {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}%
1766 \def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
1767 \def\XINT_infloatmod_a #1#2#3{\XINTinfloat [#3]{\xintMod {#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.

```
1768 \def\xintIsOne {\romannumeral0\xintisone }%
1769 \def\xintisone #1{\expandafter\XINT_fracisone
1770   \romannumeral0\xintraawwithzeros{#1}\Z }%
1771 \def\XINT_fracisone #1/#2\Z
1772   {\if0\XINT_Cmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

### 7.52 *\xintGeq*

Rewritten completely in 1.08a to be less dumb when comparing fractions having big powers of tens.

```
1773 \def\xintGeq {\romannumeral0\xintgeq }%
1774 \def\xintgeq #1%
1775 {%
1776   \expandafter\xint_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1777 }%
1778 \def\xint_fgeq #1#2%
1779 {%
1780   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1781 }%
1782 \def\XINT_fgeq_A #1%
1783 {%
1784   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1785   \XINT_fgeq_B #1%
1786 }%
1787 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1788 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1789 {%

```



## 7 Package *xintfrac* implementation

```
1790 \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1791 \expandafter\XINT_fgeq_C\expandafter
1792 {\the\numexpr #7-#3\expandafter}\expandafter
1793 {\romannumeral0\xintiimul {#4#5}{#2}}%
1794 {\romannumeral0\xintiimul {#6}{#1}}%
1795 }%
1796 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1797 \def\XINT_fgeq_C #1#2#3%
1798 {%
1799 \expandafter\XINT_fgeq_D\expandafter
1800 {#3}{#1}{#2}%
1801 }%
1802 \def\XINT_fgeq_D #1#2#3%
1803 {%
1804 \expandafter\XINT_cntSgnFork\romannumeral-`0\expandafter\XINT_cntSgn
1805 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1806 { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1807 }%
1808 \def\XINT_fgeq_E #1%
1809 {%
1810 \xint_UDsignfork
1811 #1\XINT_fgeq_Fd
1812 -{\XINT_fgeq_Fn #1}%
1813 \krof
1814 }%
1815 \def\XINT_fgeq_Fd #1\Z #2#3%
1816 {%
1817 \expandafter\XINT_fgeq_Fe\expandafter
1818 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1819 }%
1820 \def\XINT_fgeq_Fe #1#2{\XINT_geq_pre {#2}{#1}}%
1821 \def\XINT_fgeq_Fn #1\Z #2#3%
1822 {%
1823 \expandafter\XINT_geq_pre\expandafter
1824 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1825 }%
```

### 7.53 *\xintMax*

Rewritten completely in 1.08a.

```
1826 \def\xintMax {\romannumeral0\xintmax }%
1827 \def\xintmax #1%
1828 {%
1829 \expandafter\xint_fmax\expandafter {\romannumeral0\xintraff {#1}}%
1830 }%
1831 \def\xint_fmax #1#2%
1832 {%
1833 \expandafter\XINT_fmax_A\romannumeral0\xintraff {#2}#1%
1834 }%
1835 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1836 {%
1837 \xint_UDsignsfork
```

## 7 Package *xintfrac* implementation

```
1838     #1#5\XINT_fmax_minusminus
1839     -#5\XINT_fmax_firstneg
1840     #1-\XINT_fmax_secondneg
1841     --\XINT_fmax_nonneg_a
1842     \krof
1843     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1844 }%
1845 \def\XINT_fmax_minusminus --%
1846     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmin_nonneg_b }%
1847 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1848 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1849 \def\XINT_fmax_nonneg_a #1#2#3#4%
1850 {%
1851     \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1852 }%
1853 \def\XINT_fmax_nonneg_b #1#2%
1854 {%
1855     \if0\romannumeral0\XINT_fgeq_A #1#2%
1856         \xint_afterfi{ #1}%
1857     \else \xint_afterfi{ #2}%
1858     \fi
1859 }%
```

### 7.54 *\xintMaxof*

```
1860 \def\xintMaxof     {\romannumeral0\xintmaxof }%
1861 \def\xintmaxof     #1{\expandafter\XINT_maxof_a\romannumeral-`0#1\relax }%
1862 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}\Z }%
1863 \def\XINT_maxof_b #1\Z #2%
1864     {\expandafter\XINT_maxof_c\romannumeral-`0#2\Z {#1}\Z}%
1865 \def\XINT_maxof_c #1%
1866     {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%
1867 \def\XINT_maxof_d #1\Z
1868     {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1869 \def\XINT_maxof_e #1\Z #2\Z { #2}%
```

### 7.55 *\xintMin*

Rewritten completely in 1.08a.

```
1870 \def\xintMin {\romannumeral0\xintmin }%
1871 \def\xintmin #1%
1872 {%
1873     \expandafter\xint_fmin\expandafter {\romannumeral0\xintra{#1}}%
1874 }%
1875 \def\xint_fmin #1#2%
1876 {%
1877     \expandafter\XINT_fmin_A\romannumeral0\xintra{#2}#1%
1878 }%
1879 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1880 {%
1881     \xint_UDsignsfork
1882     #1#5\XINT_fmin_minusminus
1883     -#5\XINT_fmin_firstneg
```

## 7 Package *xintfrac* implementation

```
1884      #1-\XINT_fmin_secondneg
1885      --\XINT_fmin_nonneg_a
1886      \krof
1887      #1#5{#2/#3[#4]}{#6/#7[#8]}%
1888 }%
1889 \def\XINT_fmin_minusminus --%
1890   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmax_nonneg_b }%
1891 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1892 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1893 \def\XINT_fmin_nonneg_a #1#2#3#4%
1894 {%
1895   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1896 }%
1897 \def\XINT_fmin_nonneg_b #1#2%
1898 {%
1899   \if0\romannumeral0\XINT_fgeq_A #1#2%
1900     \xint_afterfi{ #2}%
1901   \else \xint_afterfi{ #1}%
1902   \fi
1903 }%
```

### 7.56 *\xintMinof*

```
1904 \def\xintMinof      {\romannumeral0\xintminof }%
1905 \def\xintminof      #1{\expandafter\XINT_minof_a\romannumeral-`0#1\relax }%
1906 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}\Z }%
1907 \def\XINT_minof_b #1\Z #2%
1908     {\expandafter\XINT_minof_c\romannumeral-`0#2\Z {#1}\Z}%
1909 \def\XINT_minof_c #1%
1910     {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
1911 \def\XINT_minof_d #1\Z
1912     {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1913 \def\XINT_minof_e #1\Z #2\Z { #2}%
```

### 7.57 *\xintCmp*

Rewritten completely in 1.08a to be less dumb when comparing fractions having big powers of tens.

```
1914 %\def\xintCmp {\romannumeral0\xintcmp }%
1915 \def\xintcmp #1%
1916 {%
1917   \expandafter\xint_fcmp\expandafter {\romannumeral0\xintra{#1}}%
1918 }%
1919 \def\xint_fcmp #1#2%
1920 {%
1921   \expandafter\XINT_fcmp_A\romannumeral0\xintra{#2}#1%
1922 }%
1923 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1924 {%
1925   \xint_UDsignsfork
1926     #1#5\XINT_fcmp_minusminus
1927     -#5\XINT_fcmp_firstneg
1928     #1-\XINT_fcmp_secondneg
1929     --\XINT_fcmp_nonneg_a
```

## 7 Package *xintfrac* implementation

```

1930 \krof
1931 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1932 }%
1933 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1934 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1935 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1936 \def\XINT_fcmp_nonneg_a #1#2%
1937 {%
1938 \xint_UDzerosfork
1939 #1#2\XINT_fcmp_zerozero
1940 0#2\XINT_fcmp_firstzero
1941 #10\XINT_fcmp_secondzero
1942 00\XINT_fcmp_pos
1943 \krof
1944 #1#2%
1945 }%
1946 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}% 1.08b had some [ and ] here!!!
1947 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}% incredibly I never saw that until
1948 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}% preparing 1.09a.
1949 \def\XINT_fcmp_pos #1#2#3#4%
1950 {%
1951 \XINT_fcmp_B #1#3#2#4%
1952 }%
1953 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1954 {%
1955 \expandafter\XINT_fcmp_C\expandafter
1956 {\the\numexpr #6-#3\expandafter}\expandafter
1957 {\romannumeral0\xintiimul {#4}{#2}}%
1958 {\romannumeral0\xintiimul {#5}{#1}}%
1959 }%
1960 \def\XINT_fcmp_C #1#2#3%
1961 {%
1962 \expandafter\XINT_fcmp_D\expandafter
1963 {#3}{#1}{#2}%
1964 }%
1965 \def\XINT_fcmp_D #1#2#3%
1966 {%
1967 \expandafter\XINT_cntSgnFork\romannumeral-`0\expandafter\XINT_cntSgn
1968 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1969 { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1970 }%
1971 \def\XINT_fcmp_E #1%
1972 {%
1973 \xint_UDsignfork
1974 #1\XINT_fcmp_Fd
1975 -{\XINT_fcmp_Fn #1}%
1976 \krof
1977 }%
1978 \def\XINT_fcmp_Fd #1\Z #2#3%
1979 {%
1980 \expandafter\XINT_fcmp_Fe\expandafter
1981 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%

```

```

1982 }%
1983 \def\XINT_fcmp_Fe #1#2{\XINT_cmp_pre {#2}{#1}}%
1984 \def\XINT_fcmp_Fn #1\Z #2#3%
1985 {%
1986   \expandafter\XINT_cmp_pre\expandafter
1987   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1988 }%

```

### 7.58 `\xintAbs`

Simplified in 1.09i. (original macro had been written before `\xintRaw`)

```

1989 \def\xintAbs   {\romannumeral0\xintabs }%
1990 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

### 7.59 `\xintOpp`

caution that `-#1` would not be ok if `#1` has `[n]` stuff. Simplified in 1.09i. (original macro had been written before `\xintRaw`)

```

1991 \def\xintOpp   {\romannumeral0\xintopp }%
1992 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

### 7.60 `\xintSgn`

Simplified in 1.09i. (original macro had been written before `\xintRaw`)

```

1993 \def\xintSgn   {\romannumeral0\xintsgn }%
1994 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\Z }%

```

### 7.61 `\xintFloatAdd`, `\XINTinFloatAdd`

1.07; 1.09ka improves a bit the efficiency of the coding of `\XINT_FL_Add_d`.

```

1995 \def\xintFloatAdd   {\romannumeral0\xintfloatadd }%
1996 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax }%
1997 \def\XINTinFloatAdd   {\romannumeral0\XINTinfloatadd }%
1998 \def\XINTinfloatadd #1{\XINT_fladd_chkopt \XINTinfloat #1\xint_relax }%
1999 \def\XINT_fladd_chkopt #1#2%
2000 {%
2001   \ifx [#2\expandafter\XINT_fladd_opt
2002     \else\expandafter\XINT_fladd_noopt
2003   \fi #1#2%
2004 }%
2005 \def\XINT_fladd_noopt #1#2\xint_relax #3%
2006 {%
2007   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2008 }%
2009 \def\XINT_fladd_opt #1[\xint_relax #2]#3#4%
2010 {%
2011   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{#4}}%
2012 }%
2013 \def\XINT_FL_Add #1#2%

```

## 7 Package *xintfrac* implementation

```

2014 {%
2015   \expandafter\XINT_FL_Add_a\expandafter{\the\numexpr #1\expandafter}%
2016   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2017 }%
2018 \def\XINT_FL_Add_a #1#2#3%
2019 {%
2020   \expandafter\XINT_FL_Add_b\romannumeral0\XINTinfloat [#1]{#3}#2{#1}%
2021 }%
2022 \def\XINT_FL_Add_b #1%
2023 {%
2024   \xint_gob_til_zero #1\XINT_FL_Add_zero 0\XINT_FL_Add_c #1%
2025 }%
2026 \def\XINT_FL_Add_c #1[#2]#3%
2027 {%
2028   \xint_gob_til_zero #3\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]#3%
2029 }%
2030 \def\XINT_FL_Add_d #1[#2]#3[#4]#5%
2031 {%
2032   \ifnum \numexpr #2-#4-#5>\xint_c_i
2033     \expandafter \xint_secondofthree_thenstop
2034   \else
2035     \ifnum \numexpr #4-#2-#5>\xint_c_i
2036       \expandafter\expandafter\expandafter\xint_thirdofthree_thenstop
2037     \fi
2038   \fi
2039   \xintadd {#1[#2]}{#3[#4]}%
2040 }%
2041 \def\XINT_FL_Add_zero 0\XINT_FL_Add_c 0[0]#1[#2]#3{#1[#2]}%
2042 \def\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]0[0]#3{#1[#2]}%

```

### 7.62 *\xintFloatSub*, *\XINTinFloatSub*

#### 1.07

```

2043 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2044 \def\xintfloatsub #1{\XINT_flsub_chkopt \xintfloat #1\xint_relax }%
2045 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2046 \def\XINTinfloatsub #1{\XINT_flsub_chkopt \XINTinfloat #1\xint_relax }%
2047 \def\XINT_flsub_chkopt #1#2%
2048 {%
2049   \ifx [#2\expandafter\XINT_flsub_opt
2050     \else\expandafter\XINT_flsub_noopt
2051   \fi #1#2%
2052 }%
2053 \def\XINT_flsub_noopt #1#2\xint_relax #3%
2054 {%
2055   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{\xintOpp{#3}}}%
2056 }%
2057 \def\XINT_flsub_opt #1[\xint_relax #2]#3#4%
2058 {%
2059   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{\xintOpp{#4}}}%
2060 }%

```

**7.63 `\xintFloatMul`, `\XINTinFloatMul`**

## 1.07

```

2061 \def\xintFloatMul    {\romannumeral0\xintfloatmul}%
2062 \def\xintfloatmul    #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
2063 \def\XINTinFloatMul  {\romannumeral0\XINTinfloatmul }%
2064 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloat #1\xint_relax }%
2065 \def\XINT_flmul_chkopt #1#2%
2066 {%
2067   \ifx [#2\expandafter\XINT_flmul_opt
2068     \else\expandafter\XINT_flmul_noopt
2069   \fi #1#2%
2070 }%
2071 \def\XINT_flmul_noopt #1#2\xint_relax #3%
2072 {%
2073   #1[\XINTdigits]{\XINT_FL_Mul {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2074 }%
2075 \def\XINT_flmul_opt #1[\xint_relax #2]#3#4%
2076 {%
2077   #1[#2]{\XINT_FL_Mul {#2+\xint_c_ii}{#3}{#4}}%
2078 }%
2079 \def\XINT_FL_Mul #1#2%
2080 {%
2081   \expandafter\XINT_FL_Mul_a\expandafter{\the\numexpr #1\expandafter}%
2082   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2083 }%
2084 \def\XINT_FL_Mul_a #1#2#3%
2085 {%
2086   \expandafter\XINT_FL_Mul_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2087 }%
2088 \def\XINT_FL_Mul_b #1[#2]#3[#4]{\xintE{\xintiiMul {#1}{#3}}{#2+#4}}%

```

**7.64 `\xintFloatDiv`, `\XINTinFloatDiv`**

## 1.07

```

2089 \def\xintFloatDiv    {\romannumeral0\xintfloatdiv}%
2090 \def\xintfloatdiv    #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
2091 \def\XINTinFloatDiv  {\romannumeral0\XINTinfloatdiv }%
2092 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloat #1\xint_relax }%
2093 \def\XINT_fldiv_chkopt #1#2%
2094 {%
2095   \ifx [#2\expandafter\XINT_fldiv_opt
2096     \else\expandafter\XINT_fldiv_noopt
2097   \fi #1#2%
2098 }%
2099 \def\XINT_fldiv_noopt #1#2\xint_relax #3%
2100 {%
2101   #1[\XINTdigits]{\XINT_FL_Div {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2102 }%
2103 \def\XINT_fldiv_opt #1[\xint_relax #2]#3#4%
2104 {%

```

```

2105     #1[#2]{\XINT_FL_Div {#2+\xint_c_ii}{#3}{#4}}%
2106 }%
2107 \def\XINT_FL_Div #1#2%
2108 {%
2109     \expandafter\XINT_FL_Div_a\expandafter{\the\numexpr #1\expandafter}%
2110     \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2111 }%
2112 \def\XINT_FL_Div_a #1#2#3%
2113 {%
2114     \expandafter\XINT_FL_Div_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2115 }%
2116 \def\XINT_FL_Div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%

```

## 7.65 `\xintFloatPow`, `\XINTinFloatPow`

1.07. Release 1.09j has re-organized the core loop, and `\XINT_flpow_prd` sub-routine has been removed.

```

2117 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2118 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax }%
2119 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
2120 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloat #1\xint_relax }%
2121 \def\XINT_flpow_chkopt #1#2%
2122 {%
2123     \ifx [#2\expandafter\XINT_flpow_opt
2124         \else\expandafter\XINT_flpow_noopt
2125     \fi
2126     #1#2%
2127 }%
2128 \def\XINT_flpow_noopt #1#2\xint_relax #3%
2129 {%
2130     \expandafter\XINT_flpow_checkB_start\expandafter
2131         {\the\numexpr #3\expandafter}\expandafter
2132         {\the\numexpr \XINTdigits}{#2}{#1[\XINTdigits]}%
2133 }%
2134 \def\XINT_flpow_opt #1[\xint_relax #2]#3#4%
2135 {%
2136     \expandafter\XINT_flpow_checkB_start\expandafter
2137         {\the\numexpr #4\expandafter}\expandafter
2138         {\the\numexpr #2}{#3}{#1[#2]}%
2139 }%
2140 \def\XINT_flpow_checkB_start #1{\XINT_flpow_checkB_a #1\Z }%
2141 \def\XINT_flpow_checkB_a #1%
2142 {%
2143     \xint_UDzerominusfork
2144     #1-\XINT_flpow_BisZero
2145     0#1{\XINT_flpow_checkB_b 1}%
2146     0-{\XINT_flpow_checkB_b 0#1}%
2147     \krof
2148 }%
2149 \def\XINT_flpow_BisZero \Z #1#2#3{#3{1/1[0]}}%
2150 \def\XINT_flpow_checkB_b #1#2\Z #3%
2151 {%

```



## 7 Package *xintfrac* implementation

```

2152 \expandafter\XINT_flpow_checkB_c \expandafter
2153 {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
2154 }%
2155 \def\XINT_flpow_checkB_c #1#2%
2156 {%
2157 \expandafter\XINT_flpow_checkB_d \expandafter
2158 {\the\numexpr \expandafter\xintlength\expandafter
2159 {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2160 }%
2161 \def\XINT_flpow_checkB_d #1#2#3#4%
2162 {%
2163 \expandafter \XINT_flpow_a
2164 \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2165 }%
2166 \def\XINT_flpow_a #1%
2167 {%
2168 \xint_UDzerominusfork
2169 #1-\XINT_flpow_zero
2170 0#1{\XINT_flpow_b 1}%
2171 0-{\XINT_flpow_b 0#1}%
2172 \krof
2173 }%
2174 \def\XINT_flpow_b #1#2[#3]#4#5%
2175 {%
2176 \XINT_flpow_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2177 {#1*\ifodd #5 1\else 0\fi}%
2178 }%
2179 \def\XINT_flpow_zero [#1]#2#3#4#5%
2180 % xint is not equipped to signal infinity, the 2^31 will provoke
2181 % deliberately a number too big and arithmetic overflow in \XINT_float_Xb
2182 {%
2183 \if #41\xint_afterfi {\xintError:DivisionByZero #5{1[2147483648]}}%
2184 \else \xint_afterfi {#5{0[0]}}\fi
2185 }%
2186 \def\XINT_flpow_loopI #1%
2187 {%
2188 \ifnum #1=\xint_c_i\XINT_flpow_ItoIII\fi
2189 \ifodd #1
2190 \expandafter\XINT_flpow_loopI_odd
2191 \else
2192 \expandafter\XINT_flpow_loopI_even
2193 \fi
2194 {#1}%
2195 }%
2196 \def\XINT_flpow_ItoIII\fi #1\fi #2#3#4#5%
2197 {%
2198 \fi\expandafter\XINT_flpow_III\the\numexpr #5\relax #3%
2199 }%
2200 \def\XINT_flpow_loopI_even #1#2#3%
2201 {%
2202 \expandafter\XINT_flpow_loopI\expandafter
2203 {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter

```

## 7 Package *xintfrac* implementation

```

2204     {#3{#2}{#2}}{#3}%
2205 }%
2206 \def\XINT_flpow_loopI_odd #1#2#3%
2207 {%
2208     \expandafter\XINT_flpow_loopII\expandafter
2209     {\the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2210     {#3{#2}{#2}}{#3}{#2}%
2211 }%
2212 \def\XINT_flpow_loopII #1%
2213 {%
2214     \ifnum #1 = \xint_c_i\XINT_flpow_IItoIII\fi
2215     \ifodd #1
2216         \expandafter\XINT_flpow_loopII_odd
2217     \else
2218         \expandafter\XINT_flpow_loopII_even
2219     \fi
2220     {#1}%
2221 }%
2222 \def\XINT_flpow_loopII_even #1#2#3%
2223 {%
2224     \expandafter\XINT_flpow_loopII\expandafter
2225     {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2226     {#3{#2}{#2}}{#3}%
2227 }%
2228 \def\XINT_flpow_loopII_odd #1#2#3#4%
2229 {%
2230     \expandafter\XINT_flpow_loopII_odda\expandafter
2231     {#3{#2}{#4}}{#1}{#2}{#3}%
2232 }%
2233 \def\XINT_flpow_loopII_odda #1#2#3#4%
2234 {%
2235     \expandafter\XINT_flpow_loopII\expandafter
2236     {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2237     {#4{#3}{#3}}{#4}{#1}%
2238 }%
2239 \def\XINT_flpow_IItoIII\fi #1\fi #2#3#4#5#6%
2240 {%
2241     \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2242     #4{#3}{#5}%
2243 }%
2244 \def\XINT_flpow_III #1#2[#3]#4%
2245 {%
2246     \expandafter\XINT_flpow_IIIend\expandafter
2247     {\the\numexpr\if #41-\fi#3\expandafter}%
2248     \xint_UDzerofork
2249     #4{#2}%
2250     0{1/#2}%
2251     \krof #1%
2252 }%
2253 \def\XINT_flpow_IIIend #1#2#3#4%
2254 {%
2255     \xint_UDzerofork

```

```

2256   #3{#4{#2[#1]}}%
2257   0{#4{-#2[#1]}}%
2258   \krof
2259 }%

```

## 7.66 `\xintFloatPower`, `\XINTinFloatPower`

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain.

```

2260 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2261 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
2262 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower}%
2263 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloat #1\xint_relax }%
2264 \def\XINT_flpower_chkopt #1#2%
2265 {%
2266   \ifx [#2\expandafter\XINT_flpower_opt
2267     \else\expandafter\XINT_flpower_noopt
2268   \fi
2269   #1#2%
2270 }%
2271 \def\XINT_flpower_noopt #1#2\xint_relax #3%
2272 {%
2273   \expandafter\XINT_flpower_checkB_start\expandafter
2274     {\the\numexpr \XINTdigits\expandafter}\expandafter
2275     {\romannumeral0\xintnum{#3}}{#2}{#1[\XINTdigits]}%
2276 }%
2277 \def\XINT_flpower_opt #1[\xint_relax #2]#3#4%
2278 {%
2279   \expandafter\XINT_flpower_checkB_start\expandafter
2280     {\the\numexpr #2\expandafter}\expandafter
2281     {\romannumeral0\xintnum{#4}}{#3}{#1[#2]}%
2282 }%
2283 \def\XINT_flpower_checkB_start #1#2{\XINT_flpower_checkB_a #2\Z {#1}}%
2284 \def\XINT_flpower_checkB_a #1%
2285 {%
2286   \xint_UDzerominusfork
2287   #1-\XINT_flpower_BisZero
2288   0#1{\XINT_flpower_checkB_b 1}%
2289   0-{\XINT_flpower_checkB_b 0#1}%
2290   \krof
2291 }%
2292 \def\XINT_flpower_BisZero \Z #1#2#3{#3{1/1[0]}}%
2293 \def\XINT_flpower_checkB_b #1#2\Z #3%
2294 {%
2295   \expandafter\XINT_flpower_checkB_c \expandafter
2296   {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
2297 }%
2298 \def\XINT_flpower_checkB_c #1#2%
2299 {%
2300   \expandafter\XINT_flpower_checkB_d \expandafter
2301   {\the\numexpr \expandafter\xintlength\expandafter
2302     {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2303 }%

```

## 7 Package *xintfrac* implementation

```

2304 \def\XINT_flpower_checkB_d #1#2#3#4%
2305 {%
2306   \expandafter \XINT_flpower_a
2307   \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2308 }%
2309 \def\XINT_flpower_a #1%
2310 {%
2311   \xint_UDzerominusfork
2312   #1-\XINT_flpow_zero
2313   0#1{\XINT_flpower_b 1}%
2314   0-{\XINT_flpower_b 0#1}%
2315   \krof
2316 }%
2317 \def\XINT_flpower_b #1#2[#3]#4#5%
2318 {%
2319   \XINT_flpower_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2320   {#1*\xintiiOdd {#5}}%
2321 }%
2322 \def\XINT_flpower_loopI #1%
2323 {%
2324   \if1\XINT_isOne {#1}\XINT_flpower_ItoIII\fi
2325   \if1\xintiiOdd{#1}%
2326     \expandafter\expandafter\expandafter\XINT_flpower_loopI_odd
2327   \else
2328     \expandafter\expandafter\expandafter\XINT_flpower_loopI_even
2329   \fi
2330   \expandafter {\romannumeral0\xinthalft{#1}}%
2331 }%
2332 \def\XINT_flpower_ItoIII\fi #1\fi\expandafter #2#3#4#5%
2333 {%
2334   \fi\expandafter\XINT_flpow_III \the\numexpr #5\relax #3%
2335 }%
2336 \def\XINT_flpower_loopI_even #1#2#3%
2337 {%
2338   \expandafter\XINT_flpower_toI\expandafter {#3{#2}{#2}}{#1}{#3}%
2339 }%
2340 \def\XINT_flpower_loopI_odd #1#2#3%
2341 {%
2342   \expandafter\XINT_flpower_toII\expandafter {#3{#2}{#2}}{#1}{#3}{#2}%
2343 }%
2344 \def\XINT_flpower_toI #1#2{\XINT_flpower_loopI {#2}{#1}}%
2345 \def\XINT_flpower_toII #1#2{\XINT_flpower_loopII {#2}{#1}}%
2346 \def\XINT_flpower_loopII #1%
2347 {%
2348   \if1\XINT_isOne {#1}\XINT_flpower_IItoIII\fi
2349   \if1\xintiiOdd{#1}%
2350     \expandafter\expandafter\expandafter\XINT_flpower_loopII_odd
2351   \else
2352     \expandafter\expandafter\expandafter\XINT_flpower_loopII_even
2353   \fi
2354   \expandafter {\romannumeral0\xinthalft{#1}}%
2355 }%

```

```

2356 \def\XINT_flpower_loopII_even #1#2#3%
2357 {%
2358   \expandafter\XINT_flpower_toII\expandafter
2359   {#3{#2}{#2}}{#1}{#3}%
2360 }%
2361 \def\XINT_flpower_loopII_odd #1#2#3#4%
2362 {%
2363   \expandafter\XINT_flpower_loopII_odda\expandafter
2364   {#3{#2}{#4}}{#2}{#3}{#1}%
2365 }%
2366 \def\XINT_flpower_loopII_odda #1#2#3#4%
2367 {%
2368   \expandafter\XINT_flpower_toII\expandafter
2369   {#3{#2}{#2}}{#4}{#3}{#1}%
2370 }%
2371 \def\XINT_flpower_IItoIII\fi #1\fi\expandafter #2#3#4#5#6%
2372 {%
2373   \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2374   #4{#3}{#5}%
2375 }%

```

## 7.67 `\xintFloatSqrt`, `\XINTinFloatSqrt`

### 1.08

```

2376 \def\xintFloatSqrt {\romannumeral0\xintfloatsqr }%
2377 \def\xintfloatsqr #1{\XINT_flsqrt_chkopt \xintfloat #1\xint_relax }%
2378 \def\XINTinFloatSqrt {\romannumeral0\XINTinfloatsqr }%
2379 \def\XINTinfloatsqr #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
2380 \def\XINT_flsqrt_chkopt #1#2%
2381 {%
2382   \ifx [#2\expandafter\XINT_flsqrt_opt
2383     \else\expandafter\XINT_flsqrt_noopt
2384   \fi #1#2%
2385 }%
2386 \def\XINT_flsqrt_noopt #1#2\xint_relax
2387 {%
2388   #1[\XINTdigits]{\XINT_FL_sqrt \XINTdigits {#2}}%
2389 }%
2390 \def\XINT_flsqrt_opt #1[\xint_relax #2]#3%
2391 {%
2392   #1[#2]{\XINT_FL_sqrt {#2}{#3}}%
2393 }%
2394 \def\XINT_FL_sqrt #1%
2395 {%
2396   \ifnum\numexpr #1<\xint_c_xviii
2397     \xint_afterfi {\XINT_FL_sqrt_a\xint_c_xviii}%
2398   \else
2399     \xint_afterfi {\XINT_FL_sqrt_a {#1+\xint_c_i}}%
2400   \fi
2401 }%
2402 \def\XINT_FL_sqrt_a #1#2%
2403 {%

```

7 Package *xintfrac* implementation

```

2404 \expandafter\XINT_FL_sqrt_checkifzeroorneg
2405 \romannumeral0\XINTinfloat [#1]{#2}%
2406 }%
2407 \def\XINT_FL_sqrt_checkifzeroorneg #1%
2408 {%
2409 \xint_UDzerominusfork
2410 #1-\XINT_FL_sqrt_iszero
2411 0#1\XINT_FL_sqrt_isneg
2412 0-\XINT_FL_sqrt_b #1}%
2413 \krof
2414 }%
2415 \def\XINT_FL_sqrt_iszero #1[#2]{0[0]}%
2416 \def\XINT_FL_sqrt_isneg #1[#2]{\xintError:RootOfNegative 0[0]}%
2417 \def\XINT_FL_sqrt_b #1[#2]%
2418 {%
2419 \ifodd #2
2420 \xint_afterfi{\XINT_FL_sqrt_c 01}%
2421 \else
2422 \xint_afterfi{\XINT_FL_sqrt_c {}0}%
2423 \fi
2424 {#1}{#2}%
2425 }%
2426 \def\XINT_FL_sqrt_c #1#2#3#4%
2427 {%
2428 \expandafter\XINT_flsqrt\expandafter {\the\numexpr #4-#2}{#3#1}%
2429 }%
2430 \def\XINT_flsqrt #1#2%
2431 {%
2432 \expandafter\XINT_sqrt_a
2433 \expandafter{\romannumeral0\xintlength {#2}}\XINT_flsqrt_big_d {#2}{#1}%
2434 }%
2435 \def\XINT_flsqrt_big_d #1#2%
2436 {%
2437 \ifodd #2
2438 \expandafter\expandafter\expandafter\XINT_flsqrt_big_eB
2439 \else
2440 \expandafter\expandafter\expandafter\XINT_flsqrt_big_eA
2441 \fi
2442 \expandafter {\the\numexpr (#2-\xint_c_i)/\xint_c_ii }{#1}%
2443 }%
2444 \def\XINT_flsqrt_big_eA #1#2#3%
2445 {%
2446 \XINT_flsqrt_big_eA_a #3\Z {#2}{#1}{#3}%
2447 }%
2448 \def\XINT_flsqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
2449 {%
2450 \XINT_flsqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
2451 }%
2452 \def\XINT_flsqrt_big_eA_b #1#2%
2453 {%
2454 \expandafter\XINT_flsqrt_big_f
2455 \romannumeral0\XINT_flsqrt_small_e {#2001}{#1}%

```

## 7 Package *xintfrac* implementation

```

2456 }%
2457 \def\XINT_flsqrt_big_eB #1#2#3%
2458 {%
2459   \XINT_flsqrt_big_eB_a #3\Z {#2}{#1}{#3}%
2460 }%
2461 \def\XINT_flsqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
2462 {%
2463   \XINT_flsqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
2464 }%
2465 \def\XINT_flsqrt_big_eB_b #1#2\Z #3%
2466 {%
2467   \expandafter\XINT_flsqrt_big_f
2468   \romannumeral0\XINT_flsqrt_small_e {#30001}{#1}%
2469 }%
2470 \def\XINT_flsqrt_small_e #1#2%
2471 {%
2472   \expandafter\XINT_flsqrt_small_f\expandafter
2473   {\the\numexpr #1*#1-#2-\xint_c_i}{#1}%
2474 }%
2475 \def\XINT_flsqrt_small_f #1#2%
2476 {%
2477   \expandafter\XINT_flsqrt_small_g\expandafter
2478   {\the\numexpr (#1+#2)/(2*#2)-\xint_c_i }{#1}{#2}%
2479 }%
2480 \def\XINT_flsqrt_small_g #1%
2481 {%
2482   \ifnum #1>\xint_c_
2483     \expandafter\XINT_flsqrt_small_h
2484   \else
2485     \expandafter\XINT_flsqrt_small_end
2486   \fi
2487   {#1}%
2488 }%
2489 \def\XINT_flsqrt_small_h #1#2#3%
2490 {%
2491   \expandafter\XINT_flsqrt_small_f\expandafter
2492   {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
2493   {\the\numexpr #3-#1}%
2494 }%
2495 \def\XINT_flsqrt_small_end #1#2#3%
2496 {%
2497   \expandafter\space\expandafter
2498   {\the\numexpr \xint_c_i+#3*\xint_c_x^iv-
2499     (#2*\xint_c_x^iv+#3)/(\xint_c_ii*#3)}%
2500 }%
2501 \def\XINT_flsqrt_big_f #1%
2502 {%
2503   \expandafter\XINT_flsqrt_big_fa\expandafter
2504   {\romannumeral0\xintiisqr {#1}}{#1}%
2505 }%
2506 \def\XINT_flsqrt_big_fa #1#2#3#4%
2507 {%

```

## 8 Package *xintseries* implementation

```

2508 \expandafter\XINT_flsqrt_big_fb\expandafter
2509 {\romannumeral0\XINT_dsx_addzerosnofuss
2510     {\numexpr #3-\xint_c_viii\relax}{#2}}%
2511 {\romannumeral0\xintiisub
2512     {\XINT_dsx_addzerosnofuss
2513     {\numexpr \xint_c_ii*(#3-\xint_c_viii)\relax}{#1}}{#4}}%
2514     {#3}}%
2515 }%
2516 \def\XINT_flsqrt_big_fb #1#2%
2517 {%
2518     \expandafter\XINT_flsqrt_big_g\expandafter {#2}{#1}%
2519 }%
2520 \def\XINT_flsqrt_big_g #1#2%
2521 {%
2522     \expandafter\XINT_flsqrt_big_j
2523     \romannumeral0\xintiidivision
2524     {#1}{\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\R\Z \W\W\W\W\W\W }{#2}}%
2525 }%
2526 \def\XINT_flsqrt_big_j #1%
2527 {%
2528     \if0\XINT_Sgn #1\Z
2529         \expandafter \XINT_flsqrt_big_end_a
2530     \else \expandafter \XINT_flsqrt_big_k
2531     \fi {#1}%
2532 }%
2533 \def\XINT_flsqrt_big_k #1#2#3%
2534 {%
2535     \expandafter\XINT_flsqrt_big_l\expandafter
2536     {\romannumeral0\XINT_sub_pre {#3}{#1}}%
2537     {\romannumeral0\xintiiaadd {#2}{\romannumeral0\XINT_sqr {#1}}}%
2538 }%
2539 \def\XINT_flsqrt_big_l #1#2%
2540 {%
2541     \expandafter\XINT_flsqrt_big_g\expandafter
2542     {#2}{#1}%
2543 }%
2544 \def\XINT_flsqrt_big_end_a #1#2#3#4#5%
2545 {%
2546     \expandafter\XINT_flsqrt_big_end_b\expandafter
2547     {\the\numexpr -#4+#5/\xint_c_ii\expandafter}\expandafter
2548     {\romannumeral0\xintiisub
2549     {\XINT_dsx_addzerosnofuss {#4}{#3}}%
2550     {\xintHalf{\xintiiaQuo{\XINT_dsx_addzerosnofuss {#4}{#2}}{#3}}}}%
2551 }%
2552 \def\XINT_flsqrt_big_end_b #1#2{#2[#1]}%
2553 \XINT_restorecatcodes_endinput%

```

## 8 Package *xintseries* implementation

<ul style="list-style-type: none"> <li>.1 Catcodes, <math>\varepsilon</math>-TeX and reload detection . . . 201</li> <li>.2 Package identification . . . . . 202</li> <li>.3 <i>\xintSeries</i> . . . . . 202</li> </ul>		<ul style="list-style-type: none"> <li>.4 <i>\xintiSeries</i> . . . . . 202</li> <li>.5 <i>\xintPowerSeries</i> . . . . . 203</li> <li>.6 <i>\xintPowerSeriesX</i> . . . . . 204</li> </ul>
--	--	---



.7	<code>\xintRationalSeries</code>	204	.10	<code>\xintFxFtPowerSeriesX</code>	207
.8	<code>\xintRationalSeriesX</code>	205	.11	<code>\xintFloatPowerSeries</code>	207
.9	<code>\xintFxFtPowerSeries</code>	206	.12	<code>\xintFloatPowerSeriesX</code>	209

The commenting is currently (2015/03/07) very sparse.

## 8.1 Catcodes, $\varepsilon$ -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 [2014/11/07 v1.1a Expandable partial sums with xint package (jfb)]%

```

## 8.3 `\xintSeries`

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.

```

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`

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.

```

73 \def\xintiSeries {\romannumeral0\xintiseries }%
74 \def\xintiseries #1#2%
75 {%
76   \expandafter\XINT_iseries\expandafter
77   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
78 }%
79 \def\XINT_iseries #1#2#3%
80 {%
81   \ifnum #2<#1

```

```

82     \xint_afterfi { 0}%
83   \else
84     \xint_afterfi {\XINT_ieseries_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%
87 \def\XINT_ieseries_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_ieseries_exit \fi
90   \expandafter\XINT_ieseries_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\xintiiadd {#2}{#4{#1}}}%
93   {#3}{#4}%
94 }%
95 \def\XINT_ieseries_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-`0#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-`0#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\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}%
251   {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
254   {\fi \expandafter\XINT_fppowseries_exit_ii }%
255 \def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257   \xinttrunc {#7}
258   {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

### 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-`0#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_relax }%

```

## 8 Package *xintseries* implementation

```

285 \def\XINT_flpowseries_chkopt #1%
286 {%
287   \ifx [#1\expandafter\XINT_flpowseries_opt
288     \else\expandafter\XINT_flpowseries_noopt
289     \fi
290   #1%
291 }%
292 \def\XINT_flpowseries_noopt #1\xint_relax #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_relax #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_relax }%
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_relax #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_relax #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-`0#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, $\varepsilon$ - $\TeX$ and reload detection	210	.16	<code>\xintiGctoF</code>	222
.2	Package identification	211	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	223
.3	<code>\xintCFrac</code>	211	.18	<code>\xintiCstoCv</code>	224
.4	<code>\xintGCFrac</code>	212	.19	<code>\xintGctoCv</code>	224
.5	<code>\xintGGCFrac</code>	214	.20	<code>\xintiGctoCv</code>	226
.6	<code>\xintGctoGCx</code>	215	.21	<code>\xintFtoCv</code>	227
.7	<code>\xintFtoCs</code>	215	.22	<code>\xintFtoCCv</code>	227
.8	<code>\xintFtoCx</code>	216	.23	<code>\xintCntoF</code>	227
.9	<code>\xintFtoC</code>	216	.24	<code>\xintGCntoF</code>	228
.10	<code>\xintFtoGC</code>	217	.25	<code>\xintCntoCs</code>	229
.11	<code>\xintFGtoC</code>	217	.26	<code>\xintCntoGC</code>	229
.12	<code>\xintFtoCC</code>	218	.27	<code>\xintGCntoGC</code>	230
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	219	.28	<code>\xintCstoGC</code>	231
.14	<code>\xintiCstoF</code>	220	.29	<code>\xintGctoGC</code>	231
.15	<code>\xintGctoF</code>	220			

The commenting is currently (2015/03/07) very sparse. Release 1.09m (2014/02/26) has modified a few things: `\xintFtoCs` and `\xintCntoCs` insert spaces after the commas, `\xintCstoF` and `\xintCstoCv` authorize spaces in the input also before the commas, `\xintCntoCs` does not brace the produced coefficients, new macros `\xintFtoC`, `\xintCtoF`, `\xintCtoCv`, `\xintFGtoC`, and `\xintGGCFrac`.

### 9.1 Catcodes, $\varepsilon$ - $\TeX$ 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 [2014/11/07 v1.1a 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_relax
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_relax
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_relax #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\xintraewithzeros {#1}\Z
73   \relax\relax
74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77   \expandafter\XINT_cfrac_A\romannumeral0\xintraewithzeros {#1}\Z
78   \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82   \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86   \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90   \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}%
94 \def\XINT_cfrac_loop_a
95 {%
96   \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100   \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104   \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108   \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111   {\XINT_cfrac_T #5#6{#2}#4\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114   \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#11#2}{#3}}%
115 }%
116 \def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
117 {%
118   \XINT_cfrac_end_b #3%
119 }%
120 \def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%
9.4 \xintGCFrac
121 \def\xintGCFrac {\romannumeral0\xintgcfrac }%

```

## 9 Package *xintcfrac* implementation

```

122 \def\xintgfrac #1{\XINT_gcfrac_opt_a #1\xint_relax }%
123 \def\XINT_gcfrac_opt_a #1%
124 {%
125   \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
126 }%
127 \def\XINT_gcfrac_noopt #1\xint_relax
128 {%
129   \XINT_gcfrac #1+\xint_relax/\relax\relax
130 }%
131 \def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint_relax #1]%
132 {%
133   \fi\csname XINT_gcfrac_opt#1\endcsname
134 }%
135 \def\XINT_gcfrac_optl #1%
136 {%
137   \XINT_gcfrac #1+\xint_relax/\relax\hfill
138 }%
139 \def\XINT_gcfrac_optc #1%
140 {%
141   \XINT_gcfrac #1+\xint_relax/\relax\relax
142 }%
143 \def\XINT_gcfrac_optr #1%
144 {%
145   \XINT_gcfrac #1+\xint_relax/\hfill\relax
146 }%
147 \def\XINT_gcfrac
148 {%
149   \expandafter\XINT_gcfrac_enter\romannumeral-`0%
150 }%
151 \def\XINT_gcfrac_enter {\XINT_gcfrac_loop {}}%
152 \def\XINT_gcfrac_loop #1#2+#3/%
153 {%
154   \xint_gob_til_xint_relax #3\XINT_gcfrac_endloop\xint_relax
155   \XINT_gcfrac_loop {{#3}{#2}#1}%
156 }%
157 \def\XINT_gcfrac_endloop\xint_relax\XINT_gcfrac_loop #1#2#3%
158 {%
159   \XINT_gcfrac_T #2#3#1\xint_relax\xint_relax
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_xint_relax #5\XINT_gcfrac_end\xint_relax\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_relax\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_relax }%
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_relax
182 {%
183   \XINT_ggcfrac #1+\xint_relax/\relax\relax
184 }%
185 \def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint_relax #1]%
186 {%
187   \fi\csname XINT_ggcfrac_opt#1\endcsname
188 }%
189 \def\XINT_ggcfrac_optl #1%
190 {%
191   \XINT_ggcfrac #1+\xint_relax/\relax\hfill
192 }%
193 \def\XINT_ggcfrac_optc #1%
194 {%
195   \XINT_ggcfrac #1+\xint_relax/\relax\relax
196 }%
197 \def\XINT_ggcfrac_optr #1%
198 {%
199   \XINT_ggcfrac #1+\xint_relax/\hfill\relax
200 }%
201 \def\XINT_ggcfrac
202 {%
203   \expandafter\XINT_ggcfrac_enter\romannumeral-`0%
204 }%
205 \def\XINT_ggcfrac_enter {\XINT_ggcfrac_loop {}}%
206 \def\XINT_ggcfrac_loop #1#2+#3/%
207 {%
208   \xint_gob_til_xint_relax #3\XINT_ggcfrac_endloop\xint_relax
209   \XINT_ggcfrac_loop {#3}{#2}#1}%
210 }%
211 \def\XINT_ggcfrac_endloop\xint_relax\XINT_ggcfrac_loop #1#2#3%
212 {%
213   \XINT_ggcfrac_T #2#3#1\xint_relax\xint_relax
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_xint_relax #5\XINT_ggcfrac_end\xint_relax\XINT_ggcfrac_U
219   #1#2{#5+\cfrac{#1#4#2}{#3}}%
220 }%
221 \def\XINT_ggcfrac_end\xint_relax\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-`0#3}{#1}{#2}%
230 }%
231 \def\XINT_gctgcx_start #1#2#3{\XINT_gctgcx_loop_a {}{#2}{#3}#1+\xint_relax/}%
232 \def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
233 {%
234 \xint_gob_til_xint_relax #5\XINT_gctgcx_end\xint_relax
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_relax\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   \xintifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
348     {\XINT_fgtc_f {#1}}%
349 }%
350 \def\XINT_fgtc_f #1#2%
351 {%
352   \xintifZero {#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}\xint_relax
438 }%
439 \def\xintCtoF {\romannumeral0\xintctof }%
440 \def\xintctof #1%
441 {%
442   \expandafter\XINT_ctf_prep \romannumeral-`0#1\xint_relax
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_xint_relax #5\XINT_ctf_end\xint_relax
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\Z #4\Z }%
458 {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
459 {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
460 {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
461 }%
462 \def\XINT_ctf_loop_c #1#2%
463 {%
464 \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
465 }%
466 \def\XINT_ctf_loop_d #1#2%
467 {%
468 \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}#1}%
469 }%
470 \def\XINT_ctf_loop_e #1#2%
471 {%
472 \expandafter\XINT_ctf_loop_a\expandafter{#2}#1%
473 }%
474 \def\XINT_ctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]
```

### 9.14 *\xintiCstoF*

```
475 \def\xintiCstoF {\romannumeral0\xinticstof }%
476 \def\xinticstof #1%
477 {%
478 \expandafter\XINT_icstf_prep \romannumeral-`0#1,\xint_relax,%
479 }%
480 \def\XINT_icstf_prep
481 {%
482 \XINT_icstf_loop_a 1001%
483 }%
484 \def\XINT_icstf_loop_a #1#2#3#4#5,%
485 {%
486 \xint_gob_til_xint_relax #5\XINT_icstf_end\xint_relax
487 \expandafter
488 \XINT_icstf_loop_b \romannumeral-`0#5.{#1}{#2}{#3}{#4}%
489 }%
490 \def\XINT_icstf_loop_b #1.#2#3#4#5%
491 {%
492 \expandafter\XINT_icstf_loop_c\expandafter
493 {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
494 {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
495 {#2}{#3}}%
496 }%
497 \def\XINT_icstf_loop_c #1#2%
498 {%
499 \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
500 }%
501 \def\XINT_icstf_end#1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]
```

### 9.15 *\xintGctoF*

```
502 \def\xintGctoF {\romannumeral0\xintgctof }%
503 \def\xintgctof #1%
```

## 9 Package *xintcfrac* implementation

```

504 {%
505   \expandafter\XINT_gctf_prep \romannumeral-`0#1+\xint_relax/%
506 }%
507 \def\XINT_gctf_prep
508 {%
509   \XINT_gctf_loop_a 1001%
510 }%
511 \def\XINT_gctf_loop_a #1#2#3#4#5+%
512 {%
513   \expandafter\XINT_gctf_loop_b
514   \romannumeral0\xintraewithzeros {#5}.{#1}{#2}{#3}{#4}%
515 }%
516 \def\XINT_gctf_loop_b #1/#2.#3#4#5#6%
517 {%
518   \expandafter\XINT_gctf_loop_c\expandafter
519   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
520   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
521   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
522   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
523 }%
524 \def\XINT_gctf_loop_c #1#2%
525 {%
526   \expandafter\XINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
527 }%
528 \def\XINT_gctf_loop_d #1#2%
529 {%
530   \expandafter\XINT_gctf_loop_e\expandafter {\expandafter{#2}#1}%
531 }%
532 \def\XINT_gctf_loop_e #1#2%
533 {%
534   \expandafter\XINT_gctf_loop_f\expandafter {\expandafter{#2}#1}%
535 }%
536 \def\XINT_gctf_loop_f #1#2/%
537 {%
538   \xint_gob_til_xint_relax #2\XINT_gctf_end\xint_relax
539   \expandafter\XINT_gctf_loop_g
540   \romannumeral0\xintraewithzeros {#2}.#1%
541 }%
542 \def\XINT_gctf_loop_g #1/#2.#3#4#5#6%
543 {%
544   \expandafter\XINT_gctf_loop_h\expandafter
545   {\romannumeral0\XINT_mul_fork #1\Z #6\Z }%
546   {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
547   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
548   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
549 }%
550 \def\XINT_gctf_loop_h #1#2%
551 {%
552   \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
553 }%
554 \def\XINT_gctf_loop_i #1#2%
555 {%

```

```

556 \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}#1}%
557 }%
558 \def\XINT_gctf_loop_j #1#2%
559 {%
560 \expandafter\XINT_gctf_loop_a\expandafter {#2}#1%
561 }%
562 \def\XINT_gctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

### 9.16 `\xintiGtoF`

```

563 \def\xintiGtoF {\romannumeral0\xintigctof }%
564 \def\xintigctof #1%
565 {%
566 \expandafter\XINT_igctf_prep \romannumeral-`0#1+\xint_relax/%
567 }%
568 \def\XINT_igctf_prep
569 {%
570 \XINT_igctf_loop_a 1001%
571 }%
572 \def\XINT_igctf_loop_a #1#2#3#4#5+%
573 {%
574 \expandafter\XINT_igctf_loop_b
575 \romannumeral-`0#5.{#1}{#2}{#3}{#4}%
576 }%
577 \def\XINT_igctf_loop_b #1.#2#3#4#5%
578 {%
579 \expandafter\XINT_igctf_loop_c\expandafter
580 {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
581 {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
582 {#2}{#3}%
583 }%
584 \def\XINT_igctf_loop_c #1#2%
585 {%
586 \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
587 }%
588 \def\XINT_igctf_loop_f #1#2#3#4/%
589 {%
590 \xint_gob_til_xint_relax #4\XINT_igctf_end\xint_relax
591 \expandafter\XINT_igctf_loop_g
592 \romannumeral-`0#4.{#2}{#3}#1%
593 }%
594 \def\XINT_igctf_loop_g #1.#2#3%
595 {%
596 \expandafter\XINT_igctf_loop_h\expandafter
597 {\romannumeral0\XINT_mul_fork #1\Z #3\Z }%
598 {\romannumeral0\XINT_mul_fork #1\Z #2\Z }%
599 }%
600 \def\XINT_igctf_loop_h #1#2%
601 {%
602 \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%
603 }%
604 \def\XINT_igctf_loop_i #1#2#3#4%
605 {%
606 \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%

```

```
607 }%
608 \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` became the one of the new `\xintCtoF` dealing with a braced rather than comma separated list.

```
609 \def\xintCstoCv {\romannumeral0\xintcstocv }%
610 \def\xintcstocv #1%
611 {%
612   \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}\xint_relax
613 }%
614 \def\xintCtoCv {\romannumeral0\xintctocv }%
615 \def\xintctocv #1%
616 {%
617   \expandafter\XINT_ctcv_prep\romannumeral-`0#1\xint_relax
618 }%
619 \def\XINT_ctcv_prep
620 {%
621   \XINT_ctcv_loop_a {}1001%
622 }%
623 \def\XINT_ctcv_loop_a #1#2#3#4#5#6%
624 {%
625   \xint_gob_til_xint_relax #6\XINT_ctcv_end\xint_relax
626   \expandafter\XINT_ctcv_loop_b
627   \romannumeral0\xintrawwithzeros {#6}.#2#{3}{#4}{#5}{#1}%
628 }%
629 \def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
630 {%
631   \expandafter\XINT_ctcv_loop_c\expandafter
632   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
633   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
634   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
635   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
636 }%
637 \def\XINT_ctcv_loop_c #1#2%
638 {%
639   \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
640 }%
641 \def\XINT_ctcv_loop_d #1#2%
642 {%
643   \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}#1}%
644 }%
645 \def\XINT_ctcv_loop_e #1#2%
646 {%
647   \expandafter\XINT_ctcv_loop_f\expandafter{#2}#1%
648 }%
649 \def\XINT_ctcv_loop_f #1#2#3#4#5%
650 {%
651   \expandafter\XINT_ctcv_loop_g\expandafter
652   {\romannumeral0\xintrawwithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
653 }%
```

```
654 \def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
655 \def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%
```

### 9.18 `\xintiCstoCv`

```
656 \def\xintiCstoCv {\romannumeral0\xinticstocv }%
657 \def\xinticstocv #1%
658 {%
659   \expandafter\XINT_icstcv_prep \romannumeral-`0#1,\xint_relax,%
660 }%
661 \def\XINT_icstcv_prep
662 {%
663   \XINT_icstcv_loop_a {}1001%
664 }%
665 \def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
666 {%
667   \xint_gob_til_xint_relax #6\XINT_icstcv_end\xint_relax
668   \expandafter
669   \XINT_icstcv_loop_b \romannumeral-`0#6.{#2}{#3}{#4}{#5}{#1}%
670 }%
671 \def\XINT_icstcv_loop_b #1.#2#3#4#5%
672 {%
673   \expandafter\XINT_icstcv_loop_c\expandafter
674   {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
675   {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
676   {{#2}{#3}}%
677 }%
678 \def\XINT_icstcv_loop_c #1#2%
679 {%
680   \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
681 }%
682 \def\XINT_icstcv_loop_d #1#2%
683 {%
684   \expandafter\XINT_icstcv_loop_e\expandafter
685   {\romannumeral0\xintraewithzeros {#1/#2}}{#1}{#2}}%
686 }%
687 \def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
688 \def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}% 1.09b removes [0]
```

### 9.19 `\xintGctoCv`

```
689 \def\xintGctoCv {\romannumeral0\xintgctocv }%
690 \def\xintgctocv #1%
691 {%
692   \expandafter\XINT_gctcv_prep \romannumeral-`0#1+\xint_relax/%
693 }%
694 \def\XINT_gctcv_prep
695 {%
696   \XINT_gctcv_loop_a {}1001%
697 }%
698 \def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
699 {%
700   \expandafter\XINT_gctcv_loop_b
701   \romannumeral0\xintraewithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
```



## 9 Package *xintcfrac* implementation

```

702 }%
703 \def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
704 {%
705   \expandafter\XINT_gctcv_loop_c\expandafter
706   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
707   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
708   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
709   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
710 }%
711 \def\XINT_gctcv_loop_c #1#2%
712 {%
713   \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
714 }%
715 \def\XINT_gctcv_loop_d #1#2%
716 {%
717   \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
718 }%
719 \def\XINT_gctcv_loop_e #1#2%
720 {%
721   \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
722 }%
723 \def\XINT_gctcv_loop_f #1#2%
724 {%
725   \expandafter\XINT_gctcv_loop_g\expandafter
726   {\romannumeral0\xintraawithzeros {#1/#2}}{#1}{#2}}%
727 }%
728 \def\XINT_gctcv_loop_g #1#2#3#4%
729 {%
730   \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
731 }%
732 \def\XINT_gctcv_loop_h #1#2#3/%
733 {%
734   \xint_gob_til_xint_relax #3\XINT_gctcv_end\xint_relax
735   \expandafter\XINT_gctcv_loop_i
736   \romannumeral0\xintraawithzeros {#3}.#2{#1}%
737 }%
738 \def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
739 {%
740   \expandafter\XINT_gctcv_loop_j\expandafter
741   {\romannumeral0\XINT_mul_fork #1\Z #6\Z }%
742   {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
743   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
744   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
745 }%
746 \def\XINT_gctcv_loop_j #1#2%
747 {%
748   \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
749 }%
750 \def\XINT_gctcv_loop_k #1#2%
751 {%
752   \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}#1}%
753 }%

```

```

754 \def\XINT_gctcv_loop_l #1#2%
755 {%
756   \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}#1}%
757 }%
758 \def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}#1}%
759 \def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%

```

## 9.20 \xintiGctoCv

```

760 \def\xintiGctoCv {\romannumeral0\xintigctocv }%
761 \def\xintigctocv #1%
762 {%
763   \expandafter\XINT_igctcv_prep \romannumeral-`0#1+\xint_relax/%
764 }%
765 \def\XINT_igctcv_prep
766 {%
767   \XINT_igctcv_loop_a {}1001%
768 }%
769 \def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
770 {%
771   \expandafter\XINT_igctcv_loop_b
772   \romannumeral-`0#6.{#2}{#3}{#4}{#5}{#1}%
773 }%
774 \def\XINT_igctcv_loop_b #1.#2#3#4#5%
775 {%
776   \expandafter\XINT_igctcv_loop_c\expandafter
777   {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
778   {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
779   {{#2}{#3}}%
780 }%
781 \def\XINT_igctcv_loop_c #1#2%
782 {%
783   \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
784 }%
785 \def\XINT_igctcv_loop_f #1#2#3#4/%
786 {%
787   \xint_gob_til_xint_relax #4\XINT_igctcv_end_a\xint_relax
788   \expandafter\XINT_igctcv_loop_g
789   \romannumeral-`0#4.#1#2{#3}%
790 }%
791 \def\XINT_igctcv_loop_g #1.#2#3#4#5%
792 {%
793   \expandafter\XINT_igctcv_loop_h\expandafter
794   {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
795   {\romannumeral0\XINT_mul_fork #1\Z #4\Z }%
796   {{#2}{#3}}%
797 }%
798 \def\XINT_igctcv_loop_h #1#2%
799 {%
800   \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
801 }%
802 \def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
803 \def\XINT_igctcv_loop_k #1#2%
804 {%

```

```

805 \expandafter\XINT_igctcv_loop_1\expandafter
806 {\romannumeral0\xintraewithzeros {#1/#2}}%
807 }%
808 \def\XINT_igctcv_loop_1 #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
809 \def\XINT_igctcv_end_a #1.#2#3#4#5%
810 {%
811 \expandafter\XINT_igctcv_end_b\expandafter
812 {\romannumeral0\xintraewithzeros {#2/#3}}%
813 }%
814 \def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]

```

### 9.21 `\xintFtoCv`

Still uses `\xinticstocv` `\xintFtoCs` rather than `\xintctocv` `\xintFtoC`.

```

815 \def\xintFtoCv {\romannumeral0\xintftocv }%
816 \def\xintftocv #1%
817 {%
818 \xinticstocv {\xintFtoCs {#1}}%
819 }%

```

### 9.22 `\xintFtoCCv`

```

820 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
821 \def\xintftoccv #1%
822 {%
823 \xintigctocv {\xintFtoCC {#1}}%
824 }%

```

### 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.

```

825 \def\xintCntoF {\romannumeral0\xintcntof }%
826 \def\xintcntof #1%
827 {%
828 \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
829 }%
830 \def\XINT_cntf #1#2%
831 {%
832 \ifnum #1>\xint_c_
833 \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
834 {\the\numexpr #1-1\expandafter}\expandafter
835 {\romannumeral-`0#2{#1}}{#2}}%
836 \else
837 \xint_afterfi
838 {\ifnum #1=\xint_c_
839 \xint_afterfi {\expandafter\space \romannumeral-`0#2{0}}%
840 \else \xint_afterfi { }% 1.09m now returns nothing.
841 \fi}%
842 \fi
843 }%
844 \def\XINT_cntf_loop #1#2#3%

```

```

845 {%
846   \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
847   \expandafter\XINT_cntf_loop\expandafter
848   {\the\numexpr #1-1\expandafter }\expandafter
849   {\romannumeral0\xintadd {\xintDiv {1[0]}\{#2}\{#3\{#1}\}}%
850   {#3}%
851 }%
852 \def\XINT_cntf_exit \fi
853   \expandafter\XINT_cntf_loop\expandafter
854   #1\expandafter #2#3%
855 {%
856   \fi\xint_gobble_ii #2%
857 }%

```

## 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.

```

858 \def\xintGCntoF {\romannumeral0\xintgcntof }%
859 \def\xintgcntof #1%
860 {%
861   \expandafter\XINT_gcntf\expandafter {\the\numexpr #1}%
862 }%
863 \def\XINT_gcntf #1#2#3%
864 {%
865   \ifnum #1>\xint_c_
866     \xint_afterfi {\expandafter\XINT_gcntf_loop\expandafter
867                   {\the\numexpr #1-1\expandafter}\expandafter
868                   {\romannumeral-`0#2\{#1}\{#2}\{#3}\}}%
869   \else
870     \xint_afterfi
871     {\ifnum #1=\xint_c_
872       \xint_afterfi {\expandafter\space\romannumeral-`0#2\{0}\}}%
873     \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
874     \fi}%
875   \fi
876 }%
877 \def\XINT_gcntf_loop #1#2#3#4%
878 {%
879   \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
880   \expandafter\XINT_gcntf_loop\expandafter
881   {\the\numexpr #1-1\expandafter }\expandafter
882   {\romannumeral0\xintadd {\xintDiv {\#4\{#1}\{#2}\{#3\{#1}\}}%
883   {#3}\{#4}%
884 }%
885 \def\XINT_gcntf_exit \fi
886   \expandafter\XINT_gcntf_loop\expandafter
887   #1\expandafter #2#3#4%
888 {%
889   \fi\xint_gobble_ii #2%
890 }%

```

## 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.

```

891 \def\xintCntoCs {\romannumeral0\xintcntocs }%
892 \def\xintcntocs #1%
893 {%
894   \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
895 }%
896 \def\XINT_cntcs #1#2%
897 {%
898   \ifnum #1<0
899     \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
900   \else
901     \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
902                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
903                   {\romannumeral-`0#2{#1}}{#2}}% produced coeff not braced
904   \fi
905 }%
906 \def\XINT_cntcs_loop #1#2#3%
907 {%
908   \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
909   \expandafter\XINT_cntcs_loop\expandafter
910   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
911   {\romannumeral-`0#3{#1}, #2}{#3}% space added, 1.09m
912 }%
913 \def\XINT_cntcs_exit \fi
914   \expandafter\XINT_cntcs_loop\expandafter
915   #1\expandafter #2#3%
916 {%
917   \fi\XINT_cntcs_exit_b #2%
918 }%
919 \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.

```

920 \def\xintCntoGC {\romannumeral0\xintcntogc }%
921 \def\xintcntogc #1%
922 {%
923   \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
924 }%
925 \def\XINT_cntgc #1#2%
926 {%
927   \ifnum #1<0

```

```

928     \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
929   \else
930     \xint_afterfi {\expandafter\XINT_cntgc_loop\expandafter
931                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
932                   {\expandafter{\romannumeral-`0#2{#1}}{#2}}}%
933   \fi
934 }%
935 \def\XINT_cntgc_loop #1#2#3%
936 {%
937   \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
938   \expandafter\XINT_cntgc_loop\expandafter
939   {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
940   {\expandafter{\romannumeral-`0#3{#1}}+1/#2}{#3}%
941 }%
942 \def\XINT_cntgc_exit \fi
943   \expandafter\XINT_cntgc_loop\expandafter
944   #1\expandafter #2#3%
945 {%
946   \fi\XINT_cntgc_exit_b #2%
947 }%
948 \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.

```

949 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
950 \def\xintgcntogc #1%
951 {%
952   \expandafter\XINT_gcntgc\expandafter {\the\numexpr #1}%
953 }%
954 \def\XINT_gcntgc #1#2#3%
955 {%
956   \ifnum #1<0
957     \xint_afterfi { }% 1.09i now returns nothing
958   \else
959     \xint_afterfi {\expandafter\XINT_gcntgc_loop\expandafter
960                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
961                   {\expandafter{\romannumeral-`0#2{#1}}{#2}{#3}}}%
962   \fi
963 }%
964 \def\XINT_gcntgc_loop #1#2#3#4%
965 {%
966   \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi
967   \expandafter\XINT_gcntgc_loop_b\expandafter
968   {\expandafter{\romannumeral-`0#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
969 }%
970 \def\XINT_gcntgc_loop_b #1#2#3%
971 {%
972   \expandafter\XINT_gcntgc_loop\expandafter
973   {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
974   {\expandafter{\romannumeral-`0#2}{#1}}%

```

```

975 }%
976 \def\XINT_gcntgc_exit \fi
977   \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
978 {%
979   \fi\XINT_gcntgc_exit_b #1%
980 }%
981 \def\XINT_gcntgc_exit_b #1/{ }%

```

## 9.28 \xintCstoGC

```

982 \def\xintCstoGC {\romannumeral0\xintcstogc }%
983 \def\xintcstogc #1%
984 {%
985   \expandafter\XINT_cstc_prep \romannumeral-`0#1,\xint_relax,%
986 }%
987 \def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
988 \def\XINT_cstc_loop_a #1#2,%
989 {%
990   \xint_gob_til_xint_relax #2\XINT_cstc_end\xint_relax
991   \XINT_cstc_loop_b {#1}{#2}%
992 }%
993 \def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
994 \def\XINT_cstc_end\xint_relax\XINT_cstc_loop_b #1#2{ #1}%

```

## 9.29 \xintGctoGC

```

995 \def\xintGctoGC {\romannumeral0\xintgctogc }%
996 \def\xintgctogc #1%
997 {%
998   \expandafter\XINT_gctgc_start \romannumeral-`0#1+\xint_relax/%
999 }%
1000 \def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
1001 \def\XINT_gctgc_loop_a #1#2+#3/%
1002 {%
1003   \xint_gob_til_xint_relax #3\XINT_gctgc_end\xint_relax
1004   \expandafter\XINT_gctgc_loop_b\expandafter
1005   {\romannumeral-`0#2}{#3}{#1}%
1006 }%
1007 \def\XINT_gctgc_loop_b #1#2%
1008 {%
1009   \expandafter\XINT_gctgc_loop_c\expandafter
1010   {\romannumeral-`0#2}{#1}%
1011 }%
1012 \def\XINT_gctgc_loop_c #1#2#3%
1013 {%
1014   \XINT_gctgc_loop_a {#3{#2}+{#1}}/%
1015 }%
1016 \def\XINT_gctgc_end\xint_relax\expandafter\XINT_gctgc_loop_b
1017 {%
1018   \expandafter\XINT_gctgc_end_b
1019 }%
1020 \def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
1021 \XINT_restorecatcodes_endinput%

```

## 10 Package `xintexpr` implementation

### Contents

10.1	Catcodes, $\varepsilon$ -TeX and reload detection	237
10.2	Package identification	238
10.3	Locking and unlocking	238
10.4	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	238
10.5	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	238
10.6	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	238
10.7	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiieexpr</code> , <code>\xinttheexpr</code> , etc...	238
10.8	<code>\xintthe</code>	239
10.9	<code>\xintthecoords</code>	239
10.10	<code>\xintbareval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	239
10.11	<code>\xinteval</code> , <code>\xintiieval</code>	239
10.12	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	239
10.13	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	240
10.14	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code>	240
10.15	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiexpr</code>	241
10.16	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	241
10.16.1	<code>\XINT::_end</code>	241
10.16.2	<code>\xintCSV::csv</code>	241
10.16.3	<code>\xintSPRaw</code> , <code>\xintSPRaw::csv</code>	241
10.16.4	<code>\xintIsTrue::csv</code>	242
10.16.5	<code>\xintRound::csv</code>	242
10.16.6	<code>\XINTinFloat::csv</code>	242
10.16.7	<code>\xintPFloat::csv</code>	242
10.17	<code>\XINT_expr_getnext</code> : fetching some number then an operator	243
10.18	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	244
10.18.1	Integral part	244
10.18.2	Fractional part	245
10.18.3	Scientific notation	245
10.18.4	Hexadecimal numbers	246
10.18.5	Function and variable names	247
10.19	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	248
10.20	Opening and closing parentheses, square brackets for lists, the <code>^C</code> for omit and abort within <code>seq</code> or <code>rseq</code>	249
10.21	<code> </code> , <code>  </code> , <code>&amp;</code> , <code>&amp;&amp;</code> , <code>&lt;</code> , <code>&gt;</code> , <code>=</code> , <code>==</code> , <code>&lt;=</code> , <code>&gt;=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>]</code> , <code>[:</code> , <code>:</code> , <code>]</code> , <code>^C</code> , and <code>++</code> operators	251
10.21.1	The <code> </code> , <code>&amp;</code> , <code>xor</code> , <code>&lt;</code> , <code>&gt;</code> , <code>=</code> , <code>&lt;=</code> , <code>&gt;=</code> , <code>!=</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , and <code>]</code> operators	251
10.21.2	The <code>]+</code> , <code>]-</code> , <code>]*</code> , <code>] /</code> , <code>] ^</code> , <code>+ [</code> , <code>- [</code> , <code>* [</code> , <code>/ [</code> , and <code>^ [</code> list operators	253
10.21.3	The <code>'and'</code> , <code>'or'</code> , <code>'xor'</code> , and <code>'mod'</code> as infix operator words	255
10.21.4	The <code>  </code> , <code>&amp;&amp;</code> , <code>**</code> , <code>** [</code> , <code>] **</code> operators as synonyms	255
10.21.5	List selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	256
10.22	Macros for a..b list generation	258
10.22.1	<code>\xintSeq::csv</code>	258
10.22.2	<code>\xintiiSeq::csv</code>	259
10.23	Macros for a..[d]..b list generation	260
10.23.1	<code>\xintSeqA::csv</code> , <code>\xintiiSeqA::csv</code> , <code>\XINTinFloatSeqA::csv</code>	260
10.23.2	<code>\xintSeqB::csv</code>	260



## Contents

10.23.3	<code>\xintiiSeqB::csv</code>	261
10.23.4	<code>\XINTinFloatSeqB::csv</code>	261
10.24	The comma as binary operator	262
10.25	The minus as prefix operator of variable precedence level	262
10.26	? as two-way and ?? as three-way conditionals with braced branches	263
10.27	! as postfix factorial operator	263
10.28	The A/B[N] mechanism	264
10.29	For variables	264
10.29.1	Defining variables	264
10.29.2	Letters as dummy variables; the nil list	265
10.29.3	The omit and abort constructs	265
10.29.4	The @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	265
10.30	For functions	266
10.31	The bool, tog!, protect, unknown, and break "functions"	266
10.32	seq and the implementation of dummy variables	267
10.32.1	<code>\XINT_expr_onlitteral_seq</code>	267
10.32.2	<code>\XINT_expr_onlitteral_seq_a</code>	267
10.32.3	<code>\XINT_isbalanced_a</code> for <code>\XINT_expr_onlitteral_seq_a</code>	268
10.32.4	<code>\XINT_allexpr_func_seqx</code> , <code>\XINT_allexpr_func_subx</code>	268
10.32.5	break, abort, omit within seq	268
10.32.6	<code>\XINT_expr_seq:_A</code>	269
10.32.7	add and mul, <code>\XINT_expr_onlitteral_add</code> , <code>\XINT_expr_onlitteral_mul</code>	269
10.32.8	<code>\XINT_expr_func_opx</code> , <code>\XINT_flexpr_func_opx</code> , <code>\XINT_iiexpr_func_opx</code>	270
10.32.9	<code>\XINT_expr_op:_a</code> , ...	270
10.32.10	subs, <code>\XINT_expr_onlitteral_subs</code>	270
10.33	rseq	270
10.33.1	<code>\XINT_expr_rseqx</code>	271
10.33.2	<code>\XINT_expr_rseqy</code>	271
10.33.3	<code>\XINT_expr_rseq:_a</code> etc...	271
10.33.4	<code>\XINT_expr_rseq:_A</code> etc...	272
10.34	rrseq	272
10.34.1	<code>\XINT_expr_rrseqx</code>	272
10.34.2	<code>\XINT_expr_rrseqy</code>	272
10.34.3	<code>\XINT_expr_rrseq:_a</code> etc...	273
10.34.4	<code>\XINT_expr_rrseq:_A</code> etc...	273
10.35	iter	274
10.35.1	<code>\XINT_expr_iterx</code>	274
10.35.2	<code>\XINT_expr_itory</code>	274
10.35.3	<code>\XINT_expr_iter:_a</code> etc...	274
10.35.4	<code>\XINT_expr_iter:_A</code> etc...	275
10.36	Macros handling csv lists for functions with multiple comma separated arguments in expressions	275
10.36.1	<code>\xintANDof:csv</code>	276
10.36.2	<code>\xintORof:csv</code>	276
10.36.3	<code>\xintXORof:csv</code>	276
10.36.4	Generic csv routine	276
10.36.5	<code>\xintMaxof:csv</code> , <code>\xintiiMaxof:csv</code>	277
10.36.6	<code>\xintMinof:csv</code> , <code>\xintiiMinof:csv</code>	277
10.36.7	<code>\xintSum:csv</code> , <code>\xintiiSum:csv</code>	277
10.36.8	<code>\xintPrd:csv</code> , <code>\xintiiPrd:csv</code>	277
10.36.9	<code>\xintGCDof:csv</code> , <code>\xintLCMof:csv</code>	277
10.36.10	<code>\xintiiGCDof:csv</code> , <code>\xintiiLCMof:csv</code>	278
10.36.11	<code>\XINTinFloatdigits</code> , <code>\XINTinFloatSqrtdigits</code>	278

10.36.12	<code>\XINTinFloatMaxof:csv</code> , <code>\XINTinFloatMinof:csv</code> . . . . .	278
10.36.13	<code>\XINTinFloatSum:csv</code> , <code>\XINTinFloatPrd:csv</code> . . . . .	278
10.37	The num, reduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, gcd, lcm, max, min, <code>`+`</code> , <code>`*`</code> , <code>?</code> , <code>!</code> , not, all, any, xor, if, ifsgn, first, last, even, odd, and reversed functions . . . . .	278
10.38	f-expandable versions of the <code>SeqB::csv</code> routines, for <code>\xintNewExpr</code> . . . . .	284
10.38.1	<code>\xintSeqB:f:csv</code> . . . . .	284
10.38.2	<code>\xintiiSeqB:f:csv</code> . . . . .	285
10.38.3	<code>\XINTinFloatSeqB:f:csv</code> . . . . .	285
10.39	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code> . . . . .	286
10.39.1	<code>\xintApply::csv</code> . . . . .	286
10.39.2	<code>\xintApply:::csv</code> . . . . .	286
10.39.3	<code>\XINT_expr_RApply::csv</code> , <code>\XINT_expr_LApply:::csv</code> , <code>\XINT_expr_RLApply:::csv</code> . . . . .	286
10.39.4	Mysterious stuff . . . . .	287

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]`.

Release [1.1 \[2014/10/28\]](#) has made many extensions, some bug fixes, and some breaking changes:

**bug fixes** • `\xintiexpr` did not strip leading zeroes,

- `\xinttheexpr \xintiexpr 1.23\relax\relax` should have produced `1`, but it produced `1.23`
- the catcode of `;` was not set at package launching time.

**breaking changes** • in `\xintiexpr`, `/` does *rounded* division, rather than the Euclidean division (for positive arguments, this is truncated division). The new `//` operator does truncated division,

- the `:` operator for three-way branching is gone, replaced with `??`,
- `1e(3+5)` is now illegal. The number parser identifies `e` and `E` in the same way it does for the decimal mark, earlier versions treated `e` as `E` rather as postfix operators,
- the `add` and `mul` have a new syntax, old syntax is with ``+`` and ``*`` (quotes mandatory), `sum` and `prd` are gone,
- no more special treatment for encountered brace pairs `{..}` by the number scanner, `a/b[N]` notation can be used without use of braces (the `N` will end up as is in a `\numexpr`, it is not parsed by the `\xintexpr`-ession scanner).
- although `&` and `|` are still available as Boolean operators the use of `&&` and `||` is strongly recommended. The single letter operators might be assigned some other meaning in later releases (bitwise operations, perhaps). Do not use them.
- place holders for `\xintNewExpr` could be denoted `#1`, `#2`, ... or also, for special purposes `$1`, `$2`, ... Only the first form is now accepted and the special cases previously treated via the second form are now managed via a `protect(...)` function.

**novelties** They are quite a few.

- `\xintiexpr`, `\xinttheexpr` admit an optional argument within brackets `[d]`, they round the computation result (or results, if comma separated) to `d` digits after decimal mark, (the whole computation is done exactly, as in `xintexpr`),

- `\xintfloatexpr`, `\xintthefloatexpr` similarly admit an optional argument which serves to keep only `d` digits of precision, getting rid of cumulated uncertainties in the last digits (the whole computation is done according to the precision set via `\xintDigits`),
- `\xinttheexpr` and `\xintthefloatexpr` 'pretty-print' if possible, the former removing unit denominator or `[0]` brackets, the latter avoiding scientific notation if decimal notation is practical,
- the `//` does truncated division and `/:` is the associated modulo,
- multi-character operators `&&`, `||`, `==`, `<=`, `>=`, `!=`, `**`,
- multi-letter infix binary words 'and', 'or', 'xor', 'mod' (quotes mandatory),
- functions `even`, `odd`,
- `\xintdefvar A3:=3.1415`; for variable definitions (non expandable, naturally), usable in subsequent expressions; variable names may contain letters, digits, underscores. They should not start with a digit, the `@` is reserved, and single lowercase and uppercase Latin letters are predefined to work as dummy variables (see next),
- generation of comma separated lists `a..b`, `a..[d]..b`,
- Python syntax-like list extractors `[list][n:]`, `[list][:n]`, `[list][a:b]` allowing negative indices, but no optional step argument, and `[list][n]` (`n=0` for the number of items in the list),
- functions `first`, `last`, `reversed`,
- itemwise operations on comma separated lists `a*[list]`, etc.., possible on both sides `a*[list]^b`, an obeying the same precedence rules as with numbers,
- `add` and `mul` must use a dummy variable: `add(x(x+1)(x-1), x=-10..10)`,
- variable substitutions with `subs`: `subs(subs(add(x^2+y^2,x=1..y),y=t),t=20)`,
- sequence generation using `seq` with a dummy variable: `seq(x^3, x=-10..10)`,
- simple recursive lists with `rseq`, with `@` given the last value, `rseq(1;2@+1,i=1..10)`,
- higher recursion with `rrseq`, `@1`, `@2`, `@3`, `@4`, and `@@n` for earlier values, up to `n=K` where `K` is the number of terms of the initial stretch `rrseq(0,1;@1+@2,i=2..100)`,
- iteration with `iter` which is like `rrseq` but outputs only the last `K` terms, where `K` was the number of initial terms,
- inside `seq`, `rseq`, `rrseq`, `iter`, possibility to use `omit`, `abort` and `break` to control termination,
- `n++` potentially infinite index generation for `seq`, `rseq`, `rrseq`, and `iter`, it is advised to use `abort` or `break(..)` at some point,
- the `add`, `mul`, `seq`, ... are nestable,
- `\xintthecoords` converts a comma separated list of an even number of items to the format as expected by the `TikZ coordinates` syntax,
- completely rewritten `\xintNewExpr`, `protect` function to handle external macros. However not all constructs are compatible with `\xintNewExpr`.

Comments dating back to earlier releases:

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.

**1.08b [2013/06/14]** corrected a problem originating in the attempt to attribute a special rôle to braces: expansion could be stopped by space tokens, as various macros tried to expand without grabbing what came next. They now have a doubled `\romannumeral-`0`.

**1.09a [2013/09/24]** has a better mechanism regarding `\xintthe`, more commenting and better organization of the code, and most importantly it implements functions, comparison operators, logic operators, conditionals. The code was reorganized and expansion proceeds a bit differently in order to have the `_getnext` and `_getop` codes entirely shared by `\xintexpr` and `\xintfloatexpr`. `\xintNewExpr` was rewritten in order to work with the standard macro parameter character `#`, to be catcode protected and to also allow comma separated expressions.

**1.09c [2013/10/09]** added the `bool` and `togl` operators, `\xintboolexpr`, and `\xintNewNumExpr`, `\xintNewBoolExpr`. The code for `\xintNewExpr` is shared with `float`, `num`, and `bool`-expressions. Also the precedence level of the postfix operators `!`, `?` and `:` has been made lower than the one of functions.

**1.09i [2013/12/18]** unpacks count and dimen registers and control sequences, with tacit multiplication. It has also made small improvements. (speed gains in macro expansions in quite a few places.)

Also, 1.09i implements `\xintiexpr`, `\xinttheiexpr`. New function `frac`. And encapsulation in `\csname..\endcsname` is done with `.=` as first tokens, so unpacking with `\string` can be done in a completely escape char agnostic way.

**1.09j [2014/01/09]** extends the tacit multiplication to the case of a sub `\xintexpr`-essions. Also, it now `\xintprotects` the result of the `\xintexpr` full expansions, thus, an `\xintexpr` without `\xintthe` prefix can be used not only as the first item within an ```\fdef'` as previously but also now anywhere within an `\edef`. Five tokens are used to pack the computation result rather than the possibly hundreds or thousands of digits of an `\xintthe` unlocked result. I deliberately omit a second `\xintprotect` which, however would be necessary if some macro `\.=digits/digits[digits]` had acquired some expandable meaning elsewhere. But this seems not that probable, and adding the protection would mean impacting everything only to allow some crazy user which has loaded something else than `xint` to do an `\edef...` the `\xintexpr` computations are otherwise in no way affected if such control sequences have a meaning.

**1.09k [2014/01/21]** does tacit multiplication also for an opening parenthesis encountered during the scanning of a number, or at a time when the parser expects an infix operator.

And it adds to the syntax recognition of hexadecimal numbers starting with a `"`, and having possibly a fractional part (except in `\xintiexpr`, naturally).

1.09kb [2014/02/13] fixes the bug introduced in `\xintNewExpr` in 1.09i of December 2013: an `\endlinechar -1` was removed, but without it there is a spurious trailing space token in the outputs of the created macros, and nesting is then impossible.

This is release 1.1a of [2014/11/07].

## 10.1 Catcodes, $\varepsilon$ -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 \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%

```

## 10.2 Package identification

```

55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2014/11/07 v1.1a Expandable expression parser (jfb)]%

```

## 10.3 Locking and unlocking

je dois réfléchir si je dois bloquer expansion après `unlock_a`, à cause de nil.

```

58 \def\xint_gob_til_! #1!{% this ! has catcode 11
59 \edef\XINT_expr_lockscan#1!{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
60 \edef\XINT_expr_lockit #1{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
61 \def\XINT_expr_inintpart #1!\XINT_num{#1}}%
62 \def\XINT_expr_infracpart #1e#2!{#1![\the\numexpr#2-\xintLength{#1}]!}%
63 \def\XINT_expr_inexppart e#1!{!\the\numexpr #1!}%
64 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string }%
65 \def\XINT_expr_unlock_a #1.={}%
66 \def\XINT_expr_unexpectedtoken {\xintError:ignored }%
67 \let\XINT_expr_done\space

```

## 10.4 `\XINT_expr_wrap`, `\XINT_iiexpr_wrap`

```

68 \def\XINT_expr_wrap {!\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
69 \def\XINT_iiexpr_wrap {!\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%

```

## 10.5 `\XINT_protectii`, `\XINT_expr_usethe`

```

70 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
71 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xinthe!}%

```

## 10.6 `\XINT_expr_print`, `\XINT_iiexpr_print`, `\XINT_boolexpr_print`

See also the `\XINT_flexpr_print` which is special, below.

```

72 \def\XINT_expr_print #1{\xintSPRaw::csv {\XINT_expr_unlock #1}}%
73 \def\XINT_iiexpr_print #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
74 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%

```

## 10.7 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiexpr`, `\xinttheexpr`, etc...

```

75 \def\xintexpr {\romannumeral0\xinteval }%
76 \def\xintiexpr {\romannumeral0\xintieval }%
77 \def\xintfloatexpr {\romannumeral0\xintfloateval }%

```

```

78 \def\xintiexpr    {\romannumeral0\xintieval    }%
79 \def\xinttheexpr
80   {\romannumeral-`0\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
81 \def\xinttheiexpr  {\romannumeral-`0\xintthe\xintiexpr }%
82 \def\xintthefloatexpr {\romannumeral-`0\xintthe\xintfloatexpr }%
83 \def\xinttheiiepr
84   {\romannumeral-`0\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%
85 % \let\xintnumexpr  \xintiexpr    % was deprecated, now obsolete with 1.1
86 % \let\xintthenumexpr\xinttheiexpr % was deprecated, now obsolete with 1.1

```

## 10.8 \xintthe

```

87 \def\xintthe #1{\romannumeral-`0\expandafter\xint_gobble_iii\romannumeral-`0#1}%

```

## 10.9 \xintthecoords

1.1 Wraps up an even number of comma separated items into pairs of TikZ coordinates; for use in the following way:

```
coordinates {\xintthecoords\xintfloatexpr ... \relax}
```

The crazyness with the \csname and unlock is due to TikZ somewhat STRANGE control of the TOTAL number of expansions which should not exceed the very low value of 100 !! As we implemented \XINT\_thecoords\_b in an "inline" style for efficiency, we need to hide its expansions.

Not to be used as \xintthecoords\xintthefloatexpr, only as \xintthecoords\xintfloatexpr (or \xintiexpr etc...). Perhaps \xintthecoords could make an extra check, but one should not accustome users to too loose requirements!

```

88 \def\xintthecoords #1{\romannumeral-`0\expandafter\expandafter\expandafter
89   \XINT_thecoords_a
90   \expandafter\xint_gobble_iii\romannumeral0#1}%
91 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
92   {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
93     \romannumeral-`0#1#2,!,!,^ \endcsname }%
94 \def\XINT_thecoords_b #1#2,#3#4,%
95   {\xint_gob_til! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
96 \def\XINT_thecoords_c #1^{}%

```

## 10.10 \xintbareeval, \xintbarefloateval, \xintbareiieval

```

97 \def\xintbareeval
98   {\expandafter\XINT_expr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
99 \def\xintbarefloateval
100  {\expandafter\XINT_flexpr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
101 \def\xintbareiieval
102  {\expandafter\XINT_iiexpr_until_end_a\romannumeral-`0\XINT_expr_getnext }%

```

## 10.11 \xinteval, \xintiieval

```

103 \def\xinteval    {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
104 \def\xintiieval {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareiieval }%

```

## 10.12 \xintieval, \XINT\_iexpr\_wrap

Optional argument since 1.1

```

105 \def\xintieval #1%
106   {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%

```

```

107 \def\XINT_iexpr_noopt
108   {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
109 \def\XINT_iexpr_withopt [#1]%
110 {%
111   \expandafter\XINT_iexpr_wrap\expandafter
112   {\the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter}%
113   \romannumeral0\xintbareeval
114 }%
115 \def\XINT_iexpr_wrap #1#2%
116 {%
117   \expandafter\XINT_expr_wrap
118   \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
119 }%

```

### 10.13 `\xintfloateval`, `\XINT_flexpr_wrap`, `\XINT_flexpr_print`

Optional argument since 1.1

```

120 \def\xintfloateval #1%
121 {%
122   \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
123   \fi #1%
124 }%
125 \def\XINT_flexpr_noopt
126 {%
127   \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
128   \romannumeral0\xintbarefloateval
129 }%
130 \def\XINT_flexpr_withopt_a [#1]%
131 {%
132   \expandafter\XINT_flexpr_withopt_b\expandafter
133   {\the\numexpr\xint_zapspace #1 \xint_gobble_i\expandafter}%
134   \romannumeral0\xintbarefloateval
135 }%
136 \def\XINT_flexpr_withopt_b #1#2%
137 {%
138   \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
139   \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
140 }%
141 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
142 \def\XINT_flexpr_print #1%
143 {%
144   \expandafter\xintPFloat::csv
145   \romannumeral-\`0\expandafter\XINT_expr_unlock_sp\string #1!%
146 }%
147 \catcode`: 12
148   \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
149 \catcode`: 11

```

### 10.14 `\xintboolexpr`, `\xinttheboolexpr`

```

150 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
151   \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%

```



```
152 \def\xinttheboolexpr {\romannumeral-`0\expandafter\expandafter\expandafter
153   \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
154 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%
```

### 10.15 `\xintifboolexpr`, `\xintifboolfloatexpr`, `\xintifbooliiexpr`

Do not work with comma separated expressions.

```
155 \def\xintifboolexpr #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
156 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
157 \def\xintifbooliiexpr #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%
```

### 10.16 Macros handling csv lists on output (for `\XINT_expr_print` et al. routines)

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

#### 10.16.1 `\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`

```
158 \def\XINT::_end #1,#2{\xint_gobble_i #2}%
```

#### 10.16.2 `\xintCSV::csv`

pour `\xinttheiiexpr`. 1.1a adds the `\romannumeral-`0` for each item, which have no use for `\xintiexpr` etc..., but are necessary for `\xintNewExpr` to be able to handle comma separated inputs. I am not sure but I think I had them just prior to releasing 1.1 but removed them foolishly.

```
159 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral-`0#1,^,%}
160 \def\XINT_csv::_a {\XINT_csv::_b {}}%
161 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral-`0#2,{#1}}%
162 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
163 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

#### 10.16.3 `\xintSPRaw`, `\xintSPRaw::csv`

Pour `\xinttheexpr`. J'avais voulu optimiser en testant si présence ou non de [N], cependant `reduce()` produit résultat sans, et du coup, le /1 peut ne pas être retiré. Bon je rajoute un [0] dans `reduce`. 14/10/25 au moment de boucler.

Same added `\romannumeral-`0` in 1.1a for `\xintNewExpr` purposes.

```
164 \def\xintSPRaw {\romannumeral0\xintspraw }%
165 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral-`0#1[\W]}%
166 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
167 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
168 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
169 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral-`0#1,^,%}
170 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
171 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral-`0#2,{#1}}%
172 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
173   \if ^#1\xint_dothis\XINT::_end\fi
174   \xint_orthat\XINT_spraw::_d #1}%
175 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
176 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

#### 10.16.4 `\xintIsTrue::csv`

```

177 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral-`0#1,^,%}
178 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
179 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral-`0#2,{#1}}%
180 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
181         \if ^#1\xint_dothis\XINT_::_end\fi
182         \xint_orthat\XINT_istrue::_d #1}%
183 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
184 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%

```

#### 10.16.5 `\xintRound::csv`

Pour `\xintiexpr` avec argument optionnel (finalement, malgré un certain overhead lors de l'exécution, pour économiser du code je ne distingue plus les deux cas). Reason for annoying expansion bridge is related to `\xintNewExpr`. Attention utilise `\XINT_::_end`.

```

185 \def\XINT_::_end #1,#2#3{\xint_gobble_i #3}%
186 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
187     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,%}
188 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral-`0#3,{#1}{#2}}%
189 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
190         \if ^#1\xint_dothis\XINT_::_end\fi
191         \xint_orthat\XINT_round::_d #1}%
192 \def\XINT_round::_d #1,#2{%
193     \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
194     \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
195 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%

```

#### 10.16.6 `\XINTinFloat::csv`

Pour `\xintfloatexpr`. Attention, prépare sous la forme `digits[N]` pour traitement par les macros. Pas utilisé en sortie. Utilise `\XINT_::_end`.

1.1a I believe this is not needed for `\xintNewExpr`, as it is removed by re-defined by `\XINT_flexpr_wrap` code, hence no need to add the extra `\romannumeral-`0`. Sub-expressions in `\xintNewExpr` are not supported.

I didn't start and don't want now to think about it at all.

```

196 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
197     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,%}
198 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
199 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
200         \if ^#1\xint_dothis\XINT_::_end\fi
201         \xint_orthat\XINT_infloat::_d #1}%
202 \def\XINT_infloat::_d #1,#2%
203     {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
204 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%

```

#### 10.16.7 `\xintPFloat::csv`

Expansion à cause de `\xintNewExpr`. Attention à l'ordre, pas le même que pour `\XINTinFloat::csv`. Donc c'est cette routine qui imprime. Utilise `\XINT_::_end`

```

205 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter

```

```

206   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,}%
207 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral-`0#3,{#1}{#2}}%
208 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
209     \if ^#1\xint_dothis\XINT::_end\fi
210     \xint_orthat\XINT_pfloat::_d #1}%
211 \def\XINT_pfloat::_d #1,#2%
212 {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint_relax #2]{#1},{#2}}%
213 \def\XINT_pfloat::_e #1,#2#3{\XINT_pfloat::_b {#2}{#3, #1}}%

```

## 10.17 `\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.

```

214 \def\XINT_expr_getnext #1%
215 {%
216   \expandafter\XINT_expr_getnext_a\romannumeral-`0#1%
217 }%
218 \def\XINT_expr_getnext_a #1%
219 {% screens out sub-expressions and \count or \dimen registers/variables
220   \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
221   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
222     \expandafter\XINT_expr_countetc
223   \else
224     \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
225   \fi
226   #1%
227 }%
228 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%
229 \def\XINT_expr_countetc #1%
230 {%
231   \ifx\count#1\else\ifx#1\dimen\else\ifx#1\numexpr\else\ifx#1\dimexpr\else
232   \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else
233     \XINT_expr_unpackvar
234   \fi\fi\fi\fi\fi\fi\fi
235   \expandafter\XINT_expr_getnext\number #1%
236 }%
237 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getnext\number #1%
238   {\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
239 \begingroup
240 \lccode`*=`#
241 \lowercase{\endgroup
242 \def\XINT_expr_getnextfork #1{%
243   \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
244   \if#1[\xint_dothis {\xint_c_xviii ({})}\fi
245   \if#1+\xint_dothis \XINT_expr_getnext \fi
246   \if#1.\xint_dothis {\XINT_expr_scandec_II\XINT_expr_infracpart}\fi
247   \if#1-\xint_dothis -\fi
248   \if#1(\xint_dothis {\xint_c_xviii ({})}\fi
249   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
250 }}%
251 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

## 10.18 The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser

```

252 \catcode96 11 % `
253 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
254 {%
255   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
256   \if `#1\xint_dothis {\XINT_expr_onlitteral_`}\fi
257   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_scandec_I\fi
258   \xint_orthat \XINT_expr_scanfunc #1%
259 }%
260 \catcode96 12 % `
261 \def\XINT_expr_scandec_I
262 {%
263   \expandafter\XINT_expr_gettop\romannumeral-`0\expandafter
264   \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
265   \romannumeral-`0\XINT_expr_scanintpart_b
266 }%
267 \def\XINT_expr_scandec_II
268 {%
269   \expandafter\XINT_expr_gettop\romannumeral-`0\expandafter
270   \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
271   \romannumeral-`0\XINT_expr_scanfracpart_b
272 }%

```

### 10.18.1 Integral part

```

273 \def\XINT_expr_scanintpart_a #1%
274 {% careful that ! has catcode letter here
275   \ifcat \relax #1\xint_dothis{!!#1}\fi % stops the scan
276   \if e#1\xint_dothis{\expandafter\XINT_expr_inexppart
277     \romannumeral-`0\XINT_expr_scanexppart_a e}\fi
278   \if E#1\xint_dothis{\expandafter\XINT_expr_inexppart
279     \romannumeral-`0\XINT_expr_scanexppart_a e}\fi
280   % \if @#1\xint_dothis{!*#1}\fi % tacit multiplication later
281   % \if _#1\xint_dothis{!*#1}\fi % tacit multiplication for variables
282   \ifcat a#1\xint_dothis{!!*#1}\fi % includes subexpressions (#1=! letter)
283   \xint_orthat {\expandafter\XINT_expr_scanintpart_aa\string #1}%
284 }%
285 \def\XINT_expr_scanintpart_aa #1%
286 {%
287   \if .#1\xint_dothis\XINT_expr_scandec_transition\fi
288   \ifnum \xint_c_ix<1#1 \xint_dothis\XINT_expr_scanintpart_b\fi
289   \xint_orthat {!!}#1%
290 }%
291 \def\XINT_expr_scanintpart_b #1#2%
292 {%
293   \expandafter #1\romannumeral-`0\expandafter
294   \XINT_expr_scanintpart_a\romannumeral-`0#2%
295 }%
296 \def\XINT_expr_scandec_transition .#1%
297 {%
298   \expandafter\XINT_expr_scandec_trans_a\romannumeral-`0#1%
299 }%

```

```

300 \def\XINT_expr_scandec_trans_a #1%
301 {%
302   \if .#1\xint_dothis{!!..}\fi
303   \xint_orthat {\expandafter\XINT_expr_infracpart
304               \romannumeral-`0\XINT_expr_scanfracpart_a #1}%
305 }%

```

### 10.18.2 Fractional part

```

306 \def\XINT_expr_scanfracpart_a #1%
307 {%
308   \ifcat \relax #1\xint_dothis{e!#1}\fi % stops the scan
309   \if e#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
310   \if E#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
311   \ifcat a#1\xint_dothis{e!*#1}\fi % and also the case of subexpressions (!)
312   \xint_orthat {\expandafter\XINT_expr_scanfracpart_aa\string #1}%
313 }%
314 \def\XINT_expr_scanfracpart_aa #1%
315 {%
316   \ifnum \xint_c_ix<1#1
317     \expandafter\XINT_expr_scanfracpart_b
318   \else
319     \xint_afterfi {e!}%
320   \fi
321   #1%
322 }%
323 \def\XINT_expr_scanfracpart_b #1#2%
324 {%
325   \expandafter #1\romannumeral-`0\expandafter
326   \XINT_expr_scanfracpart_a\romannumeral-`0#2%
327 }%

```

### 10.18.3 Scientific notation

```

328 \def\XINT_expr_scanexppart_a #1#2%
329 {%
330   \expandafter #1\romannumeral-`0\expandafter
331   \XINT_expr_scanexppart_b\romannumeral-`0#2%
332 }%
333 \def\XINT_expr_scanexppart_b #1%
334 {%
335   \ifcat \relax #1\xint_dothis{0!#1}\fi % stops the scan (incorrect syntax)
336   \ifcat a#1\xint_dothis{0!*#1}\fi % idem
337   \if +#1\xint_dothis {\XINT_expr_scanexppart_a +}\fi
338   \if -#1\xint_dothis {\XINT_expr_scanexppart_a -}\fi
339   \xint_orthat {\expandafter\XINT_expr_scanexppart_c\string #1}%
340 }%
341 \def\XINT_expr_scanexppart_c #1%
342 {%
343   \ifnum \xint_c_ix<1#1
344     \expandafter\XINT_expr_scanexppart_d
345   \else
346     \expandafter !%
347   \fi
348   #1%

```

```

349 }%
350 \def\XINT_expr_scanexppart_d #1#2%
351 {%
352   \expandafter #1\romannumeral-\`0\expandafter
353   \XINT_expr_scanexppart_e\romannumeral-\`0#2%
354 }%
355 \def\XINT_expr_scanexppart_e #1%
356 {%
357   \ifcat \relax #1\xint_dothis{!#1}\fi % stops the scan
358   \ifcat a#1\xint_dothis{!*#1}\fi % idem
359   \xint_orthat {\expandafter\XINT_expr_scanexppart_f\string #1}%
360 }%
361 \def\XINT_expr_scanexppart_f #1%
362 {%
363   \ifnum \xint_c_ix<1#1
364     \expandafter\XINT_expr_scanexppart_d
365   \else
366     \expandafter !%
367   \fi
368   #1%
369 }%

```

#### 10.18.4 Hexadecimal numbers

```

370 \def\XINT_expr_scanhex_I #1%
371 {%
372   \expandafter\XINT_expr_getop\romannumeral-\`0\expandafter
373   \XINT_expr_lockscan\expandafter\XINT_expr_inhex
374   \romannumeral-\`0\XINT_expr_scanhexI_a
375 }%
376 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
377 {%
378   \if#2I\xintHexToDec{#1}%
379   \else
380     \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
381     [\the\numexpr-4*\xintLength{#3}]%
382   \fi
383 }%
384 \def\XINT_expr_scanhexI_a #1%
385 {%
386   \ifcat #1\relax\xint_dothis{.I;!#1}\fi
387   \ifx      !#1\xint_dothis{.I;!#1}\fi % tacit multiplication
388   \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
389 }%
390 \def\XINT_expr_scanhexI_aa #1%
391 {%
392   \if\ifnum`#1>`/
393     \ifnum`#1>`9
394     \ifnum`#1>`@
395     \ifnum`#1>`F
396     0\else\fi\else0\fi\else1\fi\else0\fi 1%
397     \expandafter\XINT_expr_scanhexI_b
398   \else
399     \if .#1%

```

```

400     \expandafter\xint_firstoftwo
401     \else % gather what we got so far, leave catcode 12 #1 in stream
402     \expandafter\xint_secondoftwo
403     \fi
404     {\expandafter\XINT_expr_scanhex_transition}%
405     {\xint_afterfi {.I;!}}%
406     \fi
407     #1%
408 }%
409 \def\XINT_expr_scanhexI_b #1#2%
410 {%
411     \expandafter #1\romannumeral-`0\expandafter
412     \XINT_expr_scanhexI_a\romannumeral-`0#2%
413 }%
414 \def\XINT_expr_scanhex_transition .#1%
415 {%
416     \expandafter.\expandafter.\romannumeral-`0\expandafter
417     \XINT_expr_scanhexII_a\romannumeral-`0#1%
418 }%
419 \def\XINT_expr_scanhexII_a #1%
420 {%
421     \ifcat #1\relax\xint_dothis{;!#1}\fi
422     \ifx      !#1\xint_dothis{!*!}\fi % tacit multiplication
423     \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
424 }%
425 \def\XINT_expr_scanhexII_aa #1%
426 {%
427     \if\ifnum`#1>` /
428     \ifnum`#1>` 9
429     \ifnum`#1>` @
430     \ifnum`#1>` F
431     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
432     \expandafter\XINT_expr_scanhexII_b
433     \else
434     \xint_afterfi {;!}%
435     \fi
436     #1%
437 }%
438 \def\XINT_expr_scanhexII_b #1#2%
439 {%
440     \expandafter #1\romannumeral-`0\expandafter
441     \XINT_expr_scanhexII_a\romannumeral-`0#2%
442 }%

```

### 10.18.5 Function and variable names

```

443 \def\XINT_expr_scanfunc
444 {%
445     \expandafter\XINT_expr_func\romannumeral-`0\XINT_expr_scanfunc_a
446 }%
447 \def\XINT_expr_scanfunc_a #1#2%
448 {%
449     \expandafter #1\romannumeral-`0\expandafter\XINT_expr_scanfunc_b\romannumeral-`0#2%
450 }%

```

```

451 \def\XINT_expr_scanfunc_b #1%
452 {%
453   \ifx !#1\xint_dothis{\xint_firstoftwo{(_*!)}\fi
454   \ifcat \relax#1\xint_dothis{(_)\fi
455   \if (#1\xint_dothis{\xint_firstoftwo{(`)}\fi
456   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
457   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
458   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
459   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
460   \xint_orthat {(_}%
461   #1%
462 }%
463 \def\XINT_expr_func #1(#2%
464 {% #2=` pour une fonction, #2=_ pour une variable
465   \if #2`\ifcsname XINT_expr_var_#1\endcsname
466     \expandafter\expandafter\expandafter\xint_thirdofthree
467   \fi\fi
468   \xint_firstoftwo {\xint_c_xviii #2{#1}}{\xint_c_xviii _{#1}*}%
469 }%

```

## 10.19 `\XINT_expr_getop`: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

```

470 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
471 {%
472   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral-`0#2%
473 }%
474 \catcode`* 11
475 \def\XINT_expr_getop_a #1#2%
476 {%
477   \ifx \relax #2\xint_dothis\xint_firstofthree\fi
478   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
479   \if _#2\xint_dothis \xint_secondofthree\fi
480   \if @#2\xint_dothis \xint_secondofthree\fi
481   \if (#2\xint_dothis \xint_secondofthree\fi
482   \ifx !#2\xint_dothis \xint_secondofthree\fi
483   \xint_orthat \xint_thirdofthree
484   {\XINT_expr_founded #1}%
485   {\XINT_expr_precedence_* *#1#2}% tacit multiplication
486   {\XINT_expr_getop_b #2#1}%
487 }%
488 \catcode`* 12
489 \def\XINT_expr_founded {\xint_c_ \relax }% \relax is a place holder here.
490 \def\XINT_expr_getop_b #1%
491 {% ? and : a special syntax in \xintexpr as they are
492 % followed by braced arguments, and thus we must intercept them here.
493 % I wanted to change this but now I don't have time to think about it.
494 % 1.1 removes : as logic operator. Replaced by ??.
495   \if '#1\xint_dothis{\XINT_expr_binopwrld }\fi
496   \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
497   \xint_orthat {\XINT_expr_scanop_a #1}%

```



Package *xintexpr* implementation

```
498 }%
499 \def\XINT_expr_binopwrđ #1#2'\expandafter\XINT_expr_foundop_a
500   \csname XINT_expr_itself_\xint_zapspaceš #2 \xint_gobble_i\endcsname #1}%
501 \def\XINT_expr_scanop_a #1#2#3%
502   {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral-`0#3}%
503 \def\XINT_expr_scanop_b #1#2#3%
504 {%
505   \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
506   \ifcsname XINT_expr_itself_#1#3\endcsname
507     \xint_dothis
508       {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
509   \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
510 }%
511 \def\XINT_expr_scanop_c #1#2#3%
512 {%
513   \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral-`0#3%
514 }%
515 \def\XINT_expr_scanop_d #1#2#3%
516 {%
517   \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
518   \ifcsname XINT_expr_itself_#1#3\endcsname
519     \xint_dothis
520       {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
521   \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
522 }%
523 \def\XINT_expr_foundop_a #1%
524 {%
525   \ifcsname XINT_expr_precedence_#1\endcsname
526     \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
527     \expandafter #1%
528   \else
529     \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
530   \fi
531 }%
532 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
533 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%
```

## 10.20 Opening and closing parentheses, square brackets for lists, the ^C for omit and abort within seq or rseq

```
534 \catcode`) 11
535 \def\XINT_tmpa #1#2#3#4% (avant #4#5)
536 {%
537   \def#1##1%
538   {%
539     \xint_UDsignfork
540       ##1{\expandafter#1\romannumeral-`0#3}%
541       -{##1}%
542     \krof
543   }%
544   \def#2##1##2%
545   {%
546     \ifcase ##1\xint_afterfi
```

Package *xintexpr* implementation

```

547     {\ifx\XINT_expr_itself_ ^C ##2\xint_dothis
548     {\expandafter#1\romannumeral-`0\expandafter\XINT_expr_getnext\xint_gobble_i}\fi
549     \xint_orthat \XINT_expr_done }%
550     \or\xint_afterfi{\XINT_expr_extra_)
551         \expandafter #1\romannumeral-`0\XINT_expr_gettop }%
552     \else
553     \xint_afterfi{\expandafter#1\romannumeral-`0\csname XINT_#4_op_##2\endcsname }%
554     \fi
555 }%
556 }%
557 \def\XINT_expr_extra_) {\xintError:removed }%
558 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
559     \expandafter\XINT_tmpa
560     \csname XINT_#1_until_end_a\expandafter\endcsname
561     \csname XINT_#1_until_end_b\expandafter\endcsname
562     \csname XINT_#1_op_-vi\endcsname
563     {#1}%
564 }%
565 \def\XINT_tmpa #1#2#3#4#5#6%
566 {%
567     \def #1##1{\expandafter #3\romannumeral-`0\XINT_expr_getnext }%
568     \def #2{\expandafter #3\romannumeral-`0\XINT_expr_getnext }%
569     \def #3##1{\xint_UDsignfork
570         ##1{\expandafter #3\romannumeral-`0#5}%
571         -{#4##1}%
572         \krof }%
573     \def #4##1##2{\ifcase ##1%
574         \xint_afterfi{\ifx\XINT_expr_itself_ ^C ##2\xint_dothis{\xint_c_ ##2}\fi
575         \xint_orthat\XINT_expr_missing_) }%
576         \or \csname XINT_#6_op_##2\expandafter\endcsname
577         \else
578         \xint_afterfi{\expandafter #3\romannumeral-`0\csname XINT_#6_op_##2\endcsname }%
579         \fi
580     }%
581 }%
582 \def\XINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%
583 \catcode` ) 12
584 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
585     \expandafter\XINT_tmpa
586     \csname XINT_#1_op_(\expandafter\endcsname
587     \csname XINT_#1_oparen\expandafter\endcsname
588     \csname XINT_#1_until_)_a\expandafter\endcsname
589     \csname XINT_#1_until_)_b\expandafter\endcsname
590     \csname XINT_#1_op_-vi\endcsname
591     {#1}%
592 }%
593 \expandafter\let\csname XINT_expr_precedence_)\endcsname\xint_c_i
594 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
595 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
596 \let\XINT_expr_precedence_a \xint_c_xviii
597 \expandafter\let\csname XINT_expr_precedence_^C\endcsname \xint_c_
598 \expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i

```

```

599 \catcode`. 11 \catcode`= 11 \catcode`+ 11
600 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
601   \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
602   \expandafter\let\csname XINT_#1_op_;\endcsname \space
603   \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_getop ##1a}%
604   \expandafter\let\csname XINT_#1_op_a)\endcsname \XINT_expr_getop
605   \expandafter\def\csname XINT_#1_op_+)\endcsname ##1##2\relax
606   {\expandafter\XINT_expr_foundend \expandafter
607     {\expandafter\.=+\xintiCeil{\XINT_expr_unlock ##1}}}%
608 }%
609 \catcode`. 12 \catcode`= 12 \catcode`+ 12

```

## 10.21 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, -, \*, /, ^, \*\*, //, /:, ..., ..[, ].., ][, ][:, :], ^C, and ++ operators

```

610 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{.}{.}{.}{.}{.}{.}%
611   {+}{-}{*}{/}{^}{a+}{a-}{a*}{a/}{a**}{a^}%
612   {]}{[}{:]}{:^C}{++}{++}}
613   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%

```

### 10.21.1 The |, &, xor, <, >, =, <=, >=, !=, //, /:, ..., ..[, and ].. operators

```

614 \def\XINT_tmprc #1#2#3#4#5#6#7#8%
615 {%
616   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
617   {% keep value, get next number and operator, then do until
618     \expandafter #2\expandafter ##1%
619     \romannumeral-`0\expandafter\XINT_expr_getnext }%
620   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
621   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
622     -{#3##1##2}%
623     \krof }%
624   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
625   {% either execute next operation now, or first do next (possibly unary)
626     \ifnum ##2>#5%
627     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
628       \csname XINT_#8_op_##3\endcsname {##4}}%
629     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
630       \csname .=#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
631     \fi }%
632   \let #7#5%
633 }%
634 \def\XINT_tmprb #1#2#3#4#5#6%
635 {%
636   \expandafter\XINT_tmprc
637   \csname XINT_#1_op_#3\endcsname\expandafter\endcsname
638   \csname XINT_#1_until_#3_a\endcsname\expandafter\endcsname
639   \csname XINT_#1_until_#3_b\endcsname\expandafter\endcsname
640   \csname XINT_#1_op_-#5\endcsname\expandafter\endcsname
641   \csname xint_c_#4\endcsname\expandafter\endcsname
642   \csname #2#6\endcsname\expandafter\endcsname
643   \csname XINT_expr_precedence_#3\endcsname {#1}%
644 }%
645 \xintFor #1 in {expr, flexpr} \do {%
646   \def\XINT_tmpra ##1{\XINT_tmprb {#1}{xint}##1}%

```

```

647 \xintApplyInline {\XINT_tmpa }{%
648   {\{iii\}{vi}{OR}}%
649   {\&\{iv\}{vi}{AND}}%
650   {\{xor\}\{iii\}{vi}{XOR}}%
651   {\<\{v\}{vi}{Lt}}%
652   {\>\{v\}{vi}{Gt}}%
653   {\={v\}{vi}{Eq}}%
654   {\{<=\}\{v\}{vi}{LtorEq}}%
655   {\{>=\}\{v\}{vi}{GtorEq}}%
656   {\{!=\}\{v\}{vi}{Neq}}%
657   {\{.\}\{iii\}{vi}{Seq::csv}}%
658   {\{/\/\}\{vii\}{vii}{DivTrunc}}%
659   {\{/:\}\{vii\}{vii}{Mod}}%
660 }%
661 }%
662 \def\XINT_tmpa #1{\XINT_tmpb {expr}{xint}#1}%
663 \xintApplyInline {\XINT_tmpa }{%
664   {\+{\vi}{vi}{Add}}%
665   {\-{\vi}{vi}{Sub}}%
666   {\*{\vii}{vii}{Mul}}%
667   {\/{vii}{vii}{Div}}%
668   {\^{viii}{viii}{Pow}}%
669   {\{.\}[\{iii\}{vi}{SeqA::csv}}%
670   {\{[\{iii\}{vi}{SeqB::csv}}%
671 }%
672 \def\XINT_tmpa #1{\XINT_tmpb {flexpr}{XINTinFloat}#1}%
673 \xintApplyInline {\XINT_tmpa }{%
674   {\+{\vi}{vi}{Add}}%
675   {\-{\vi}{vi}{Sub}}%
676   {\*{\vii}{vii}{Mul}}%
677   {\/{vii}{vii}{Div}}%
678   {\^{viii}{viii}{Power}}%
679   {\{.\}[\{iii\}{vi}{SeqA::csv}}%
680   {\{[\{iii\}{vi}{SeqB::csv}}%
681 }%
682 \def\XINT_tmpa #1{\XINT_tmpb {iiexpr}{xint}#1}%
683 \xintApplyInline {\XINT_tmpa }{%
684   {\{iii\}{vi}{OR}}%
685   {\&\{iv\}{vi}{AND}}%
686   {\{xor\}\{iii\}{vi}{XOR}}%
687   {\<\{v\}{vi}{iiLt}}%
688   {\>\{v\}{vi}{iiGt}}%
689   {\={v\}{vi}{iiEq}}%
690   {\{<=\}\{v\}{vi}{iiLtorEq}}%
691   {\{>=\}\{v\}{vi}{iiGtorEq}}%
692   {\{!=\}\{v\}{vi}{iiNeq}}%
693   {\+{\vi}{vi}{iiAdd}}%
694   {\-{\vi}{vi}{iiSub}}%
695   {\*{\vii}{vii}{iiMul}}%
696   {\/{vii}{vii}{iiDivRound}}% CHANGED IN 1.1! PREVIOUSLY DID EUCLIDEAN QUOTIENT
697   {\^{viii}{viii}{iiPow}}%
698   {\{.\}[\{iii\}{vi}{iiSeqA::csv}}%

```

```

699  {{}}{iii}{vi}{iiSeqB::csv}}%
700  {{}}{iii}{vi}{iiSeq::csv}}%
701  {{/}}{vii}{vii}{iiDivTrunc}}%
702  {{/:}}{vii}{vii}{iiMod}}%
703 }%

```

### 10.21.2 The $+$ , $-$ , $*$ , $/$ , $^$ , $+$ , $-$ , $*$ , $/$ , and $^$ list operators

#### $\backslash$ XINT\_expr\_binop\_inline\_b

```

704 \def\XINT_expr_binop_inline_a
705   {\expandafter\xint_gobble_i\romannumeral-\`0\XINT_expr_binop_inline_b }%
706 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
707 \def\XINT_expr_binop_inline_c #1{%
708   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
709   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
710   \xint_orthat\XINT_expr_binop_inline_d #1}%
711 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
712 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
713 \def\XINT_expr_binop_inline_end #1,#2{}%
714 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
715 {%
716   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iiexpr
717   {% keep value, get next number and operator, then do until
718     \expandafter #2\expandafter ##1%
719     \romannumeral-\`0\expandafter\XINT_expr_getnext }%
720   \def #2#1#2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
721   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral-\`0#4}%
722     -{#3##1#2}}%
723   \krof }%
724   \def #3#1#2#3#4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
725   {% either execute next operation now, or first do next (possibly unary)
726     \ifnum ##2>#5%
727     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-\`0%
728       \csname XINT_#8_op_##3\endcsname {##4}}%
729     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
730       \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
731         {\expandafter\expandafter\expandafter#6\expandafter
732           \xint_exchangetwo_keepbraces\expandafter
733           {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
734         \romannumeral-\`0\XINT_expr_unlock ##1,^,\endcsname }%
735     \fi }%
736   \let #7#5%
737 }%
738 \def\XINT_tmpb #1#2#3#4%
739 {%
740   \expandafter\XINT_tmpc
741   \csname XINT_#1_op_#2\expandafter\endcsname
742   \csname XINT_#1_until_#2_a\expandafter\endcsname
743   \csname XINT_#1_until_#2_b\expandafter\endcsname
744   \csname XINT_#1_op_#3\expandafter\endcsname
745   \csname xint_c_#3\expandafter\endcsname
746   \csname #4\expandafter\endcsname
747   \csname XINT_expr_precedence_#2\endcsname {#1}%
748 }%

```

Package *xintexpr* implementation

```

749 \xintApplyInline {\expandafter\XINT_tmpb \xint_firstofone}{%
750  {{expr}{+}{vi}{xintAdd}}}%
751  {{expr}{-}{vi}{xintSub}}}%
752  {{expr}{*}{vii}{xintMul}}}%
753  {{expr}{/}{vii}{xintDiv}}}%
754  {{expr}{a^}{viii}{xintPow}}}%
755  {{iiexpr}{+}{vi}{xintiiAdd}}}%
756  {{iiexpr}{-}{vi}{xintiiSub}}}%
757  {{iiexpr}{*}{vii}{xintiiMul}}}%
758  {{iiexpr}{/}{vii}{xintiiDivRound}}}%
759  {{iiexpr}{a^}{viii}{xintiiPow}}}%
760  {{flexpr}{+}{vi}{XINTinFloatAdd}}}%
761  {{flexpr}{-}{vi}{XINTinFloatSub}}}%
762  {{flexpr}{*}{vii}{XINTinFloatMul}}}%
763  {{flexpr}{/}{vii}{XINTinFloatDiv}}}%
764  {{flexpr}{a^}{viii}{XINTinFloatPower}}}%
765 }%
766 \def\XINT_tmpc #1#2#3#4#5#6#7%
767 {%
768  \def #1#1{\expandafter#2\expandafter##1\romannumeral-`0%
769    \expandafter #3\romannumeral-`0\XINT_expr_getnext }%
770  \def #2#1##2##3##4%
771  {% either execute next operation now, or first do next (possibly unary)
772    \ifnum ##2>#4%
773    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
774      \csname XINT_#7_op_##3\endcsname {##4}}}%
775    \else \xint_afterfi {\expandafter ##2\expandafter ##3%
776      \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
777        {\expandafter#5\expandafter
778          {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
779        \romannumeral-`0\XINT_expr_unlock ##4,^,\endcsname }%
780    \fi }%
781  \let #6#4%
782 }%
783 \def\XINT_tmpb #1#2#3#4%
784 {%
785  \expandafter\XINT_tmpc
786  \csname XINT_#1_op_#2\expandafter\endcsname
787  \csname XINT_#1_until_#2\expandafter\endcsname
788  \csname XINT_#1_until_)_a\expandafter\endcsname
789  \csname xint_c_#3\expandafter\endcsname
790  \csname #4\expandafter\endcsname
791  \csname XINT_expr_precedence_#2\endcsname {#1}%
792 }%
793 \xintApplyInline {\expandafter\XINT_tmpb\xint_firstofone }{%
794  {{expr}{+}{vi}{xintAdd}}}%
795  {{expr}{-}{vi}{xintSub}}}%
796  {{expr}{*}{vii}{xintMul}}}%
797  {{expr}{/}{vii}{xintDiv}}}%
798  {{expr}{^}{viii}{xintPow}}}%
799  {{iiexpr}{+}{vi}{xintiiAdd}}}%
800  {{iiexpr}{-}{vi}{xintiiSub}}}%

```

```

801  {{{iexpr}{*[]{}vii}{xintiiMul}}}%
802  {{{iexpr}{/[]{}vii}{xintiiDivRound}}}%
803  {{{iexpr}{^[]{}viii}{xintiiPow}}}%
804  {{flexpr}{+[]{}vi}{XINTinFloatAdd}}}%
805  {{flexpr}{-[]{}vi}{XINTinFloatSub}}}%
806  {{flexpr}{*[]{}vii}{XINTinFloatMul}}}%
807  {{flexpr}{/[]{}vii}{XINTinFloatDiv}}}%
808  {{flexpr}{^[]{}viii}{XINTinFloatPower}}}%
809 }%

```

### 10.21.3 The 'and', 'or', 'xor', and 'mod' as infix operator words

```

810 \xintFor #1 in {and,or,xor,mod} \do {%
811   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
812 \expandafter\let\csname XINT_expr_precedence_and\endcsname
813   \csname XINT_expr_precedence_&\endcsname
814 \expandafter\let\csname XINT_expr_precedence_or\endcsname
815   \csname XINT_expr_precedence_|\endcsname
816 \expandafter\let\csname XINT_expr_precedence_mod\endcsname
817   \csname XINT_expr_precedence_/\endcsname
818 \xintFor #1 in {expr, flexpr, iexpr} \do {%
819   \expandafter\let\csname XINT_#1_op_and\endcsname
820     \csname XINT_#1_op_&\endcsname
821   \expandafter\let\csname XINT_#1_op_or\endcsname
822     \csname XINT_#1_op_|\endcsname
823   \expandafter\let\csname XINT_#1_op_mod\endcsname
824     \csname XINT_#1_op_/\endcsname
825 }%

```

### 10.21.4 The ||, &&, \*\*, \*\*[, ]\*\* operators as synonyms

```

826 \expandafter\let\csname XINT_expr_precedence_==\endcsname
827   \csname XINT_expr_precedence_=\endcsname
828 \expandafter\let\csname XINT_expr_precedence_&&\endcsname
829   \csname XINT_expr_precedence_&\endcsname
830 \expandafter\let\csname XINT_expr_precedence_||\endcsname
831   \csname XINT_expr_precedence_|\endcsname
832 \expandafter\let\csname XINT_expr_precedence_**\endcsname
833   \csname XINT_expr_precedence_^\endcsname
834 \expandafter\let\csname XINT_expr_precedence_a**\endcsname
835   \csname XINT_expr_precedence_a^\endcsname
836 \expandafter\let\csname XINT_expr_precedence_**[\endcsname
837   \csname XINT_expr_precedence_^\endcsname
838 \xintFor #1 in {expr, flexpr, iexpr} \do {%
839   \expandafter\let\csname XINT_#1_op_==\endcsname
840     \csname XINT_#1_op_=\endcsname
841   \expandafter\let\csname XINT_#1_op_&&\endcsname
842     \csname XINT_#1_op_&\endcsname
843   \expandafter\let\csname XINT_#1_op_||\endcsname
844     \csname XINT_#1_op_|\endcsname
845   \expandafter\let\csname XINT_#1_op_**\endcsname
846     \csname XINT_#1_op_^\endcsname
847   \expandafter\let\csname XINT_#1_op_a**\endcsname
848     \csname XINT_#1_op_a^\endcsname

```

```
849 \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
850 \csname XINT_#1_op_^\endcsname
851 }%
```

### 10.21.5 List selectors: [list][N], [list][:b], [list][a:], [list][a:b]

1.1 (27 octobre 2014) I implement Python syntax, see <http://stackoverflow.com/a/13005464/4184837>. Do not implement third argument giving the step. Also, I gather that [5:2] selector returns empty and not, as I could have been tempted to do, (list[5], list[4], list[3]). Anyway, it is simpler not to go that way. For reversing I could implement [::-1] but this would get confusing, better to do function "reversed".

This gets the job done, but I would definitely need \xintTrim::csv, \xintKeep::csv, \xintNthElt::csv for better efficiency. Not for 1.1.

```
852 \def\xINT_tmpa #1#2#3#4#5#6%
853 {%
854 \def #1##1% \XINT_expr_op_][
855 {%
856 \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
857 }%
858 \def #2##1##2% \XINT_expr_until_][_a
859 {\xint_UDsignfork
860 ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
861 -{#3##1##2}%
862 \krof }%
863 \def #3##1##2##3##4% \XINT_expr_until_][_b
864 {%
865 \ifnum ##2>\xint_c_ii
866 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
867 \csname XINT_#6_op_##3\endcsname {##4}}%
868 \else
869 \xint_afterfi
870 {\expandafter ##2\expandafter ##3\csname
871 .=\expandafter\xintListSel:csv \romannumeral-`0\XINT_expr_unlock ##4;%
872 \XINT_expr_unlock ##1;\endcsname % unlock for \xintNewExpr
873 }%
874 \fi
875 }%
876 \let #5\xint_c_ii
877 }%
878 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
879 \expandafter\xINT_tmpa
880 \csname XINT_#1_op_][\expandafter\endcsname
881 \csname XINT_#1_until_][_a\expandafter\endcsname
882 \csname XINT_#1_until_][_b\expandafter\endcsname
883 \csname XINT_#1_op_-vi\expandafter\endcsname
884 \csname XINT_expr_precedence_][\endcsname {#1}%
885 }%
886 \def\xINT_tmpa #1#2#3#4#5#6%
887 {%
888 \def #1##1% \XINT_expr_op_:
889 {%
890 \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
891 }%
```



Package *xintexpr* implementation

```

892 \def #2##1##2% \XINT_expr_until_:_a
893 {\xint_UDsignfork
894   ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
895   -{#3##1##2}%
896   \krof }%
897 \def #3##1##2##3##4% \XINT_expr_until_:_b
898 {%
899   \ifnum ##2>\xint_c_iii
900     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
901                   \csname XINT_#6_op_##3\endcsname {##4}}%
902   \else
903     \xint_afterfi
904     {\expandafter ##2\expandafter ##3\csname
905      .:=\xintiiifSgn{\XINT_expr_unlock ##1}NPP.% : and dots for expansion
906      \xintiiifSgn{\XINT_expr_unlock ##4}NPP.% in \xintNewExpr context
907 % reason for \xintNum is a/1[x] format, but -0.5 will not work, seen <0, but 0 after
908      \xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}\endcsname
909     }%
910   \fi
911 }%
912 \let #5\xint_c_iii
913 }%
914 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
915 \expandafter\XINT_tmpa
916   \csname XINT_#1_op_:\expandafter\endcsname
917   \csname XINT_#1_until_:_a\expandafter\endcsname
918   \csname XINT_#1_until_:_b\expandafter\endcsname
919   \csname XINT_#1_op_-vi\expandafter\endcsname
920   \csname XINT_expr_precedence_:\endcsname {#1}%
921 }%
922 \catcode`[ 11 \catcode`] 11
923 \let\XINT_expr_precedence_:] \xint_c_iii
924 \def\XINT_expr_op_:] #1{\expandafter\xint_c_i\expandafter ]}%
925   \csname .:=\xintiiifSgn{\XINT_expr_unlock #1}npp\XINT_expr_unlock #1\endcsname }%
926 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
927 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
928 \let\XINT_expr_precedence_:[: \xint_c_iii
929 \edef\XINT_expr_op_:[: #1{\xint_c_ii \expandafter\noexpand
930   \csname XINT_expr_itself_][\endcsname #10\string :}%
931 % : must be catcode 12, else will be mistaken for start of variable by expression parser
932 \let\XINT_flexpr_op_:[: \XINT_expr_op_:[:
933 \let\XINT_iiexpr_op_:[: \XINT_expr_op_:[:
934 \catcode`[ 12 \catcode`] 12
935 \def\xintListSel:csv #1{% these complications are due to \xintNewExpr matters
936   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral-`0}\fi
937   \if : \noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
938   \xint_orthat {\XINT_listsel:_nth #1}%
939 }%
940 \def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
941   \expandafter\XINT_listsel:_keep\fi }%
942 \def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname }%
943 \def\XINT_listsel:_trim #1;#2;%

```

```

944   {\xintListWithSep,{\xintTrim {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
945 \def\xINT_listsel:_keep #1;#2;%
946   {\xintListWithSep,{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
947 \def\xINT_listsel:_nth#1;#2;%
948   {\xintNthElt {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}%
949 \def\xINT_listsel:_PP #1;#2;#3;%
950   {\xintListWithSep,%
951     {\xintTrim {\xintNum{#1}}%
952       {\xintKeep {\xintNum{#2}}%
953         {\xintCSVtoListNonStripped{#3}}}%
954       }%
955     }%
956   }%
957 \def\xINT_listsel:_NN #1;#2;#3;%
958   {\xintListWithSep,%
959     {\xintTrim {\xintNum{#2}}%
960       {\xintKeep {\xintNum{#1}}%
961         {\xintCSVtoListNonStripped{#3}}}%
962       }%
963     }%
964   }%
965 \def\xINT_listsel:_NP #1;#2;#3;%
966   {\expandafter\xINT_listsel:_NP_a \the\numexpr #1+%
967     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#2;#3;%
968 \def\xINT_listsel:_NP_a #1#2;{\if -#1\expandafter\xINT_listsel:_OP\fi
969   \XINT_listsel:_PP #1#2;}%
970 \def\xINT_listsel:_OP\xINT_listsel:_PP #1;{\XINT_listsel:_PP 0;}%
971 \def\xINT_listsel:_PN #1;#2;#3;%
972   {\expandafter\xINT_listsel:_PN_a \the\numexpr #2+%
973     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#1;#3;%
974 \def\xINT_listsel:_PN_a #1#2;#3;{\if -#1\expandafter\xINT_listsel:_PO\fi
975   \XINT_listsel:_PP #3;#1#2;}%
976 \def\xINT_listsel:_PO\xINT_listsel:_PP #1;#2;{\XINT_listsel:_PP #1;0;}%

```

## 10.22 Macros for a..b list generation

Attention, ne produit que des listes de petits entiers!

### 10.22.1 `\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.

```

977 \def\xintSeq::csv {\romannumeral0\xintseq::csv }%
978 \def\xintseq::csv #1#2%
979 {%
980   \expandafter\xINT_seq::csv\expandafter
981     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
982     {\the\numexpr \xintiFloor{#2}}%
983 }%
984 \def\xINT_seq::csv #1#2%

```

```

985 {%
986   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
987     \expandafter\XINT_seq::csv_z
988   \or
989     \expandafter\XINT_seq::csv_p
990   \else
991     \expandafter\XINT_seq::csv_n
992   \fi
993   {#2}{#1}%
994 }%
995 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
996 \def\XINT_seq::csv_p #1#2%
997 {%
998   \ifnum #1>#2
999     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1000   \else
1001     \expandafter\XINT_seq::csv_e
1002   \fi
1003   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1004 }%
1005 \def\XINT_seq::csv_n #1#2%
1006 {%
1007   \ifnum #1<#2
1008     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1009   \else
1010     \expandafter\XINT_seq::csv_e
1011   \fi
1012   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1013 }%
1014 \def\XINT_seq::csv_e #1,{ }%

```

### 10.22.2 \xintiiseq::csv

```

1015 \def\xintiiseq::csv {\romannumeral0\xintiiseq::csv }%
1016 \def\xintiiseq::csv #1#2%
1017 {%
1018   \expandafter\XINT_iiseq::csv\expandafter
1019     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1020 }%
1021 \def\XINT_iiseq::csv #1#2%
1022 {%
1023   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1024     \expandafter\XINT_iiseq::csv_z
1025   \or
1026     \expandafter\XINT_iiseq::csv_p
1027   \else
1028     \expandafter\XINT_iiseq::csv_n
1029   \fi
1030   {#2}{#1}%
1031 }%
1032 \def\XINT_iiseq::csv_z #1#2{ #1}%
1033 \def\XINT_iiseq::csv_p #1#2%
1034 {%

```

```

1035 \ifnum #1>#2
1036 \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1037 \else
1038 \expandafter\XINT_seq::csv_e
1039 \fi
1040 \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1041 }%
1042 \def\XINT_iiseq::csv_n #1#2%
1043 {%
1044 \ifnum #1<#2
1045 \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1046 \else
1047 \expandafter\XINT_seq::csv_e
1048 \fi
1049 \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1050 }%
1051 \def\XINT_seq::csv_e #1,{ }%

```

## 10.23 Macros for a..[d].b list generation

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  $d>0$ .

### 10.23.1 `\xintSeqA::csv`, `\xintiiSeqA::csv`, `\XINTinFloatSeqA::csv`

```

1052 \def\xintSeqA::csv #1%
1053 {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%
1054 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1;}%
1055 \def\xintiiSeqA::csv #1#2{\XINT_iiseqA::csv #1#2}%
1056 \def\XINT_iiseqA::csv #1#2#3#4{\expandafter\XINT_seqa::csv_a
1057 \romannumeral-`0\expandafter \XINT_expr_unlock\expandafter#4%
1058 \expandafter;\romannumeral-`0\XINT_expr_unlock #2;}%
1059 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1060 {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1061 \def\XINT_flseqa::csv #1#2%
1062 {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1063 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1064 #1-{z}%
1065 0#1{n}%
1066 0-{p}%
1067 \krof #1}%

```

### 10.23.2 `\xintSeqB::csv`

```

1068 \def\xintSeqB::csv #1#2%
1069 {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1070 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral-`0#2#1!}%
1071 \def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1072 \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1073 \def\XINT_seqb::csv_p #1#2#3%
1074 {%
1075 \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1076 {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1077 % \romannumeral0 stopped by \endcsname, XINT_expr_seq_empty? constructs "nil".

```

```

1078   {\romannumeral0\xintadd {#3}{#1}{#2}{#3}%
1079 }%
1080 \def\xINT_seqb::csv_n #1#2#3%
1081 {%
1082   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1083   {, #1\expandafter\xINT_seqb::csv_n\expandafter}%
1084   {\romannumeral0\xintadd {#3}{#1}{#2}{#3}%
1085 }%
1086 \def\xINT_seqb::csv_z #1#2#3{, #1}%

```

### 10.23.3 \xintiiSeqB::csv

```

1087 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1088 \def\xINT_iiseqb::csv #1#2#3#4%
1089   {\expandafter\xINT_iiseqb::csv_a
1090   \romannumeral-`0\expandafter \XINT_expr_unlock\expandafter#2%
1091   \romannumeral-`0\XINT_expr_unlock #4!}%
1092 \def\xINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\xINT_expr_seq_empty?
1093   \romannumeral-`0\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1094 \def\xINT_iiseqb::csv_p #1#2#3%
1095 {%
1096   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\xINT_iiseqb::csv_p\expandafter}%
1097   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1098   {\romannumeral0\xintiiadd {#3}{#1}{#2}{#3}%
1099 }%
1100 \def\xINT_iiseqb::csv_n #1#2#3%
1101 {%
1102   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1103   {, #1\expandafter\xINT_iiseqb::csv_n\expandafter}%
1104   {\romannumeral0\xintiiadd {#3}{#1}{#2}{#3}%
1105 }%
1106 \def\xINT_iiseqb::csv_z #1#2#3{, #1}%

```

### 10.23.4 \XINTinFloatSeqB::csv

```

1107 \def\XINTinFloatSeqB::csv #1#2{\expandafter\xINT_flseqb::csv \expandafter
1108   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}{#1}}%
1109 \def\xINT_flseqb::csv #1#2{\expandafter\xINT_flseqb::csv_a\romannumeral-`0#2#1!}%
1110 \def\xINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\xINT_expr_seq_empty?
1111   \romannumeral-`0\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1112 \def\xINT_flseqb::csv_p #1#2#3%
1113 {%
1114   \xintifCmp {#1}{#2}{, #1\expandafter\xINT_flseqb::csv_p\expandafter}%
1115   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1116   {\romannumeral0\XINTinfloatadd {#3}{#1}{#2}{#3}%
1117 }%
1118 \def\xINT_flseqb::csv_n #1#2#3%
1119 {%
1120   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1121   {, #1\expandafter\xINT_flseqb::csv_n\expandafter}%
1122   {\romannumeral0\XINTinfloatadd {#3}{#1}{#2}{#3}%
1123 }%
1124 \def\xINT_flseqb::csv_z #1#2#3{, #1}%

```

## 10.24 The comma as binary operator

New with 1.09a.

```

1125 \def\XINT_tmpa #1#2#3#4#5#6%
1126 {%
1127   \def #1##1% \XINT_expr_op_,
1128   {%
1129     \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
1130   }%
1131   \def #2##1##2% \XINT_expr_until_,_a
1132   {\xint_UDsignfork
1133     ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
1134     -{##1##2}%
1135     \krof }%
1136   \def #3##1##2##3##4% \XINT_expr_until_,_b
1137   {%
1138     \ifnum ##2>\xint_c_ii
1139       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
1140         \csname XINT_#6_op_##3\endcsname {##4}}%
1141     \else
1142       \xint_afterfi
1143       {\expandafter ##2\expandafter ##3%
1144         \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1145     \fi
1146   }%
1147   \let #5\xint_c_ii
1148 }%
1149 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1150 \expandafter\XINT_tmpa
1151   \csname XINT_#1_op_,\expandafter\endcsname
1152   \csname XINT_#1_until_,_a\expandafter\endcsname
1153   \csname XINT_#1_until_,_b\expandafter\endcsname
1154   \csname XINT_#1_op_-vi\expandafter\endcsname
1155   \csname XINT_expr_precedence_,\endcsname {#1}%
1156 }%

```

## 10.25 The minus as prefix operator of variable precedence level

```

1157 \def\XINT_tmpa #1#2#3%
1158 {%
1159   \expandafter\XINT_tmpb
1160   \csname XINT_#1_op_-#3\expandafter\endcsname
1161   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1162   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1163   \csname xint_c_#3\endcsname {#1}#2%
1164 }%
1165 \def\XINT_tmpb #1#2#3#4#5#6%
1166 {%
1167   \def #1% \XINT_expr_op_-<level>
1168   {% get next number+operator then switch to _until macro
1169     \expandafter #2\romannumeral-`0\XINT_expr_getnext
1170   }%

```

```

1171 \def #2##1% \XINT_expr_until_<l>_a
1172 {\xint_UDsignfork
1173   ##1{\expandafter #2\romannumeral-`0#1}%
1174   -{#3##1}%
1175   \krof }%
1176 \def #3##1##2##3% \XINT_expr_until_<l>_b
1177 {% _until tests precedence level with next op, executes now or postpones
1178   \ifnum ##1>#4%
1179     \xint_afterfi {\expandafter #2\romannumeral-`0%
1180                   \csname XINT_#5_op_##2\endcsname {##3}}%
1181   \else
1182     \xint_afterfi {\expandafter ##1\expandafter ##2%
1183                   \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1184   \fi
1185 }%
1186 }%
1187 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1188 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1189 \xintApplyInline{\XINT_tmpa {iexpr}\xintiiOpp}{\vi}{vii}{viii}{ix}}%

```

## 10.26 ? 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. ?{yes}{no} and ??{<0}{=0}{>0}

```

1190 \let\XINT_expr_precedence_? \xint_c_x
1191 \def\XINT_expr_op_? #1#2{\if ?#2\expandafter \XINT_expr_op_??\fi
1192   \XINT_expr_op_?a #1{#2}}%
1193 \def\XINT_expr_op_?a #1#2#3%
1194 {%
1195   \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1196 }%
1197 \let\XINT_flexpr_op_?\XINT_expr_op_?
1198 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1199 \def\XINT_expr_op_?? #1#2#3#4#5#6%
1200 {%
1201   \xintiiifSgn {\XINT_expr_unlock #2}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1202   {\XINT_expr_getnext #6}%
1203 }%

```

## 10.27 ! as postfix factorial operator

As of 2014/11/07, not yet a float version of factorial. I must do it!

```

1204 \let\XINT_expr_precedence_! \xint_c_x
1205 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1206   \csname .=\xintFac{\XINT_expr_unlock #1}\endcsname }%
1207 \let\XINT_flexpr_op_!\XINT_expr_op_!
1208 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1209   \csname .=\xintiFac{\XINT_expr_unlock #1}\endcsname }%

```

## 10.28 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\* 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 CRAZINESS OF NUMEXPR: \the\numexpr 3 + 7 9 \relax !! Hence we have to do the job ourselves.

```

1210 \catcode`[ 11
1211 \catcode`* 11
1212 \let\XINT_expr_precedence_ \xint_c_vii
1213 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1214     \csname .=\xintE{\XINT_expr_unlock #1}%
1215     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1216 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1217     \csname .=\xintiiE{\XINT_expr_unlock #1}%
1218     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1219 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1220     \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1221     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1222 \catcode`[ 12
1223 \catcode`* 12

```

## 10.29 For variables

```

1224 \def\XINT_expr_op__ #1% op__ with two _'s
1225     {%
1226     \ifcsname XINT_expr_var_#1\endcsname
1227     \expandafter\xint_firstoftwo
1228     \else
1229     \expandafter\xint_secondoftwo
1230     \fi
1231     {\expandafter\expandafter\expandafter\expandafter
1232     \expandafter\expandafter\expandafter
1233     \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1234     {\XINT_expr_unknown_variable {#1}%
1235     \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1236     }%
1237 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1238 \let\XINT_flexpr_op__ \XINT_expr_op__
1239 \let\XINT_iiexpr_op__ \XINT_expr_op__

```

### 10.29.1 Defining variables

1.1 An active : character will be a pain and I almost decided not to use := but rather = as affectation operator, but this is the same problem inside expressions with the modulo operator /:, or with babel+frenchb with all high punctuation ?, !, :, ;.

It is not recommended to overwrite single Latin letters which are pre-defined to serve as dummy variables. Variable names may contain letters, digits, underscores, and must not start with a digit.

```

1240 \catcode`: 12
1241 \def\xintdefvar #1:=#2;{\expandafter\odef
1242     \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
1243     {\expandafter\empty\romannumeral0\xintbareeval #2\relax }}%

```



```

1244 \def\xintdefiivar #1:=#2;{\expandafter\odef
1245   \csname XINT_expr_var_\xint_zapspace #1 \xint_gobble_i\endcsname
1246   {\expandafter\empty\romannumeral0\xintbareiieval #2\relax }%
1247 }%
1248 \def\xintdeffloatvar #1:=#2;{\expandafter\odef
1249   \csname XINT_expr_var_\xint_zapspace #1 \xint_gobble_i\endcsname
1250   {\expandafter\empty\romannumeral0\xintbarefloateval #2\relax }%
1251 }%
1252 \catcode`: 11

```

### 10.29.2 Letters as dummy variables; the nil list

```

1253 \def\XINT_tmpa #1%
1254 {%
1255   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1256   {\romannumeral0\XINT_expr_lockscan ##2!##1\relax !#1{##2}}%
1257 }%
1258 \xintApplyUnbraced \XINT_tmpa {abcdefghijklmnopqrstuvwxyz}%
1259 \xintApplyUnbraced \XINT_tmpa {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1260 \expandafter\def\expandafter\XINT_expr_var_nil\expandafter
1261   {\expandafter\empty\csname .= \endcsname}%

```

### 10.29.3 The omit and abort constructs

```

1262 \catcode`. 11 \catcode`= 11
1263 \def\XINT_expr_var_omit #1\relax !{1^C!{}{}{}.\.=!\relax !}% 24 juin
1264 \def\XINT_expr_var_abort #1\relax !{1^C!{}{}{}.\.=^\relax !}% 25 juin
1265 \catcode`. 12 \catcode`= 12

```

### 10.29.4 The @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

I had completely forgotten what the @@@ etc... stuff were supposed to do: this is for nesting recursions! (I was mad back in June 2014). @@(N) gives the Nth back, @@@(N) gives the Nth back of the higher recursion!

```

1266 \catcode`? 3
1267 \def\XINT_expr_var_@ #1~#2{ #2#1~#2}%
1268 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1269 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{ #3#1~#2#3}%
1270 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{ #4#1~#2#3#4}%
1271 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{ #5#1~#2#3#4#5}%
1272 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1273 {%
1274   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1275     {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1276 }%
1277 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1278 {%
1279   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1280     {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1281 }%
1282 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1283 {%
1284   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1285     {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%

```

```

1286 }%
1287 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1288 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1289 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
1290 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1291 {%
1292   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1293   {\XINT_expr_unlock#3}{#5}#4~#5?%
1294 }%
1295 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1296 {%
1297   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1298   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1299 }%
1300 \def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1301 {%
1302   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1303   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1304 }%
1305 \catcode`? 11

```

### 10.30 For functions

```

1306 \def\XINT_tmpa #1#2#3{%
1307   \def #1##1% \XINT_expr_op_`, #2=\XINT_expr_oparen
1308   {%
1309     \ifcsname XINT_expr_onlitteral_##1\endcsname
1310     \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi
1311     \ifcsname XINT_#3_func_##1\endcsname
1312     \xint_dothis{\expandafter\expandafter
1313       \csname XINT_#3_func_##1\endcsname\romannumeral-`0#2}\fi
1314     \xint_orthat{\XINT_expr_unknown_function {##1}%
1315       \expandafter\XINT_expr_func_unknown\romannumeral-`0#2}%
1316   }%
1317 }%
1318 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1319 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1320   \expandafter\XINT_tmpa
1321     \csname XINT_#1_op_`\expandafter\endcsname
1322     \csname XINT_#1_oparen\endcsname
1323     {#1}%
1324 }%
1325 \expandafter\def\csname XINT_expr_onlitteral_`\endcsname #1#2#3({\xint_c_xviii `{#2}}%

```

### 10.31 The bool, togl, protect, unknown, and break "functions"

*bool*, *togl* and *protect* use delimited macros. Only *unknown* and *break* are true functions with a more flexible parsing of the opening and closing parentheses, which may possibly arise from expansion itself.

```

1326 \def\XINT_expr_onlitteral_bool #1)%
1327   {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1328 \def\XINT_expr_onlitteral_togl #1)%
1329   {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%

```

```

1330 \def\XINT_expr_onlitteral_protect #1)%
1331     {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%
1332 \def\XINT_expr_func_unknown #1#2#3{\expandafter #1\expandafter #2\csname .=#3\endcsname }%
1333 \def\XINT_expr_func_break #1#2#3%
1334 {\expandafter #1\expandafter #2\csname.=?\romannumeral-\`0\XINT_expr_unlock #3\endcsname }%
1335 \let\XINT_flexpr_func_break \XINT_expr_func_break
1336 \let\XINT_iiexpr_func_break \XINT_expr_func_break

```

## 10.32 seq and the implementation of dummy variables

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_onlitteral_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 `iiexpr`.

### 10.32.1 \XINT\_expr\_onlitteral\_seq

```

1337 \def\XINT_expr_onlitteral_seq
1338 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral-\`0\XINT_expr_onlitteral_seq_a {}}%
1339 \def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

### 10.32.2 \XINT\_expr\_onlitteral\_seq\_a

```

1340 \def\XINT_expr_onlitteral_seq_a #1#2,%
1341 {% checks balancing of parentheses
1342     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1343         \expandafter\XINT_expr_onlitteral_seq_c
1344         \or\expandafter\XINT_expr_onlitteral_seq_b
1345         \else\expandafter\xintError:we_are_doomed
1346     \fi {#1#2},%
1347 }%
1348 \def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1},}%
1349 \def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
1350 {%
1351     \XINT_expr_onlitteral_seq_d {#2{#1}}{}}%
1352 }%
1353 \def\XINT_expr_onlitteral_seq_d #1#2#3)%
1354 {%
1355     \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1356         \or\expandafter\XINT_expr_onlitteral_seq_e
1357         \else\expandafter\xintError:we_are_doomed

```

```

1358   \fi
1359   {#1}{#2#3}%
1360 }%
1361 \def\XINT_expr_onlitteral_seq_e #1#2{\XINT_expr_onlitteral_seq_d {#1}{#2}}%

```

### 10.32.3 \XINT\_isbalanced\_a for \XINT\_expr\_onlitteral\_seq\_a

Expands to `\m@ne` in case a closing `)` had no opening `(` matching it, to `\@ne` if opening `(` had no closing `)` matching it, to `\z@` if expression was balanced.

```

1362 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1363 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1364 \def\XINT_isbalanced_b #1)#2%
1365   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%

   if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1

1366 \def\XINT_isbalanced_error #1)\xint_bye {\m@ne}%

```

`#2` was `\xint_bye`, was there a `)` in original `#1`?

```

1367 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1368   {\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.

```

1369 \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.

```

```

1370 \def\XINT_isbalanced_d #1)#2%
1371   {\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.

```

1372 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

### 10.32.4 \XINT\_allexpr\_func\_seqx, \XINT\_allexpr\_func\_subx

```

1373 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareeval }%
1374 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbarefloateval}%
1375 \def\XINT_iiexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareiieval }%
1376 \def\XINT_allexpr_seqx #1#2#3#4% #2 is the index list, fully evaluated and encapsulated
1377 {% #3 is the Latin letter serving as dummy variable, #4 is the expression to evaluate
1378   \expandafter \XINT_expr_getop
1379   \csname .=\expandafter\XINT_expr_seq:_aa
1380     \romannumeral-`0\XINT_expr_unlock #2!{#1#4\relax !#3},^,\endcsname
1381 }%
1382 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1383   \expandafter\XINT_expr_seq:_a\fi #1}%

```

### 10.32.5 break, abort, omit within seq

when evaluation is done in `seq:_d`, after the `!` we find: the Latin letter, the braced evaluated value to which it will be assigned, a saved copy of the the `\xintexpr` stuff, the braced accumulated

comma separated list of previous computations, and the rest of the list of comma separated values to assign to the dummy letter and at the very end there is ^ and the final comma.

```

1384 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1385         \romannumeral0\XINT_expr_seq:_b {#2}#1}%
1386 \def\XINT_expr_seq:_b #1#2,{\XINT_expr_seq:_c #2,{#1}}%
1387 \def\XINT_expr_seq:_c #1{\if ,#1\xint_dothis\XINT_expr_seq:_noop\fi
1388         \if ^#1\xint_dothis\XINT_expr_seq:_end\fi
1389         \xint_orthat\XINT_expr_seq:_d #1}%
1390 \def\XINT_expr_seq:_d #1,#2{\expandafter\XINT_expr_seq:_e
1391     \romannumeral-\`0\expandafter\XINT_expr_unlock\romannumeral0#2{#1}{#2}}%
1392 \def\XINT_expr_seq:_e #1{\if #1^\xint_dothis\XINT_expr_seq:_abort\fi
1393     \if #1?\xint_dothis\XINT_expr_seq:_break\fi
1394     \if #1!\xint_dothis\XINT_expr_seq:_omit\fi
1395     \xint_orthat{\XINT_expr_seq:_goon #1}}%
1396 \def\XINT_expr_seq:_goon #1!#2#3#4{,#1\XINT_expr_seq:_b {#4}}%
1397 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1398 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1399 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1400 \def\XINT_expr_seq:_noop ,#1{\XINT_expr_seq:_b {#1}}%
1401 \def\XINT_expr_seq:_end ^,#1{% if all is omit, _empty? constructs "nil"
1402 \def\XINT_expr_seq_empty? #1{%
1403 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1404 \XINT_expr_seq_empty? { }%

```

### 10.32.6 \XINT\_expr\_seq:\_A

This is for index lists generated by ++. The starting point will have been replaced by its ceil. For efficiency I use \numexpr rather than \xintInc, hence the indexing is limited to small integers.

```

1405 \def\XINT_expr_seq:_A +#1!#2,^,%
1406     {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D {#1}{#2}}%
1407 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E
1408     \romannumeral-\`0\expandafter\XINT_expr_unlock\romannumeral0#2{#1}{#2}}%
1409 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1410     \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1411     \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1412     \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1413 \def\XINT_expr_seq:_Goon #1!#2#3#4%
1414     {,#1\expandafter\XINT_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
1415 \def\XINT_expr_seq:_Omit #1!#2#3#4%
1416     {\expandafter\XINT_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
1417 \def\XINT_expr_seq:_Abort #1!#2#3#4{%
1418 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%

```

### 10.32.7 add and mul, \XINT\_expr\_onlitteral\_add, \XINT\_expr\_onlitteral\_mul

```

1419 \def\XINT_expr_onlitteral_add
1420     {\expandafter\XINT_expr_onlitteral_add_f\romannumeral-\`0\XINT_expr_onlitteral_seq_a {}}%
1421 \def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opx}#2)\relax #1+}%
1422 \def\XINT_expr_onlitteral_mul
1423     {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral-\`0\XINT_expr_onlitteral_seq_a {}}%
1424 \def\XINT_expr_onlitteral_mul_f #1#2{\xint_c_xviii `{opx}#2)\relax #1*}%

```

### 10.32.8 `\XINT_expr_func_opx`, `\XINT_flexpr_func_opx`, `\XINT_iiexpr_func_opx`

```

1425 \expandafter\edef\csname XINT_expr_op:_{+}\endcsname
1426     {\noexpand\xint_gobble_v {}}{\noexpand\csname .:=0\endcsname}%
1427 \expandafter\edef\csname XINT_expr_op:_{*}\endcsname
1428     {\noexpand\xint_gobble_v {}}{\noexpand\csname .:=1\endcsname}%
1429 \def\XINT_expr_func_opx #1#2{\XINT_allexpr_opx \xintexpr }%
1430 \def\XINT_flexpr_func_opx #1#2{\XINT_allexpr_opx \xintfloatexpr }%
1431 \def\XINT_iiexpr_func_opx #1#2{\XINT_allexpr_opx \xintiiepr }%
1432 \def\XINT_allexpr_opx #1#2#3#4#5%
1433 {% au d\'e part on avait op(#4,#3=#2 (\'evalu\'e ici)) #3=la variable, #4=expression, #5=+ ou*.
1434     \expandafter\XINT_expr_getop\romannumeral0\expandafter\XINT_expr_op:_a
1435     \csname XINT_expr_op:_{#5}\expandafter\endcsname
1436     \romannumeral-`0\XINT_expr_unlock #2!#5#1#3{#4}%
1437 }%

```

### 10.32.9 `\XINT_expr_op:_a`, ...

```

1438 \def\XINT_expr_op:_a #1#2!#3#4#5#6{\XINT_expr_op:_b {#1#4#3{#6\relax\relax !#5}}#2,^,%
1439 % #1=op_+ ou op_*, #2=liste, #3=+ou*,#4=\xintexpr, etc, #5=la var,#6=expression
1440 \def\XINT_expr_op:_b #1#2,{\XINT_expr_op:_c #2,#1}%
1441 \def\XINT_expr_op:_c #1{\if ,#1\xint_dothis\XINT_expr_op:_noop\fi
1442     \if ^#1\xint_dothis\XINT_expr_op:_end\fi
1443     \xint_orthat\XINT_expr_op:_d #1}%
1444 \def\XINT_expr_op:_noop #1,#2#3#4#5{\XINT_expr_op:_b {#2}#3#4{#5}}%
1445 \def\XINT_expr_op:_d #1,#2#3#4#5%
1446 % #1=valeur, #2=partiel, #3=\xintexpr #4=+ ou *, #5 = expression
1447 {\expandafter\expandafter\expandafter\XINT_expr_op:_e #3#2#4#3#5{#1}{#3#4{#5}}}%
1448 % #2=nom de la variable, #3=ancienne valeur variable,
1449 \def\XINT_expr_op:_e !#1!#2#3#4{\XINT_expr_op:_b {!#1}#4}%
1450 \def\XINT_expr_op:_end ^,#1#2#3#4{\expandafter\expandafter\expandafter\space
1451     \expandafter\xint_gobble_iv #1}%

```

### 10.32.10 `subs`, `\XINT_expr_onlitteral_subs`

```

1452 \def\XINT_expr_onlitteral_subs
1453 {\expandafter\XINT_expr_onlitteral_subs_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
1454 \def\XINT_expr_onlitteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1455 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1456 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1457 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval}%
1458 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1459 {% #3 is the dummy variable, #4 is the expression to evaluate
1460     \expandafter \XINT_expr_getop
1461     \csname .:=\expandafter\XINT_expr_subx:_a
1462     \romannumeral-`0\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1463 }%
1464 \def\XINT_expr_subx:_a #1!#2% 10/25 that was a quick addition!
1465 {\expandafter\XINT_expr_subx:_end \romannumeral0#2{#1}}%
1466 % attention, if one day I add a space in unlock, will need \romannumeral-`0
1467 \def\XINT_expr_subx:_end #1!#2#3{\XINT_expr_unlock #1}%

```

## 10.33 `rseq`

When `func_rseq` has its turn, initial segment has been scanned by `oparen`, the `;` mimicking the rôle of a closing parenthesis, and stopping further expansion.

```

1468 \def\XINT_expr_func_rseq  {\XINT_allexpr_rseq \xintbareeval    }%
1469 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
1470 \def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval  }%
1471 \def\XINT_allexpr_rseq #1#2%
1472 {%
1473   \expandafter\XINT_expr_rseq\expandafter #1\expandafter
1474   #2\romannumeral-\`0\XINT_expr_onlitteral_seq_a }%
1475 }%

```

### 10.33.1 \XINT\_expr\_rseqx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```

1476 \def\XINT_expr_rseqx #1#2#3#4%
1477 {%
1478   \expandafter\XINT_expr_rseq\romannumeral0#1(#4)\relax
1479   #2#3#1%
1480 }%

```

### 10.33.2 \XINT\_expr\_rseqy

```

1481 \def\XINT_expr_rseqy #1#2#3#4#5% #1=valeurs pour variable (locked),
1482           % #2=toutes les valeurs initiales (csv,locked),
1483           % #3=variable, #4=expr,
1484           % #5=\xintbareeval ou \xintbarefloateval ou \xintbareiieval
1485 {%
1486   \expandafter \XINT_expr_getop
1487   \csname .=\XINT_expr_unlock #2%
1488   \expandafter\XINT_expr_rseq:_aa
1489           \romannumeral-\`0\XINT_expr_unlock #1!{#5#4\relax !#3}#2,^,\endcsname
1490 }%
1491 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1492           \expandafter\XINT_expr_rseq:_a\fi #1}%

```

### 10.33.3 \XINT\_expr\_rseq:\_a etc...

```

1493 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b #3{#2}#1}%
1494 \def\XINT_expr_rseq:_b #1#2#3,{\XINT_expr_rseq:_c #3,~#1{#2}}%
1495 \def\XINT_expr_rseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rseq:_noop\fi
1496           \if ^#1\xint_dothis\XINT_expr_rseq:_end\fi
1497           \xint_orthat\XINT_expr_rseq:_d #1}%
1498 \def\XINT_expr_rseq:_d #1,~#2#3{\expandafter\XINT_expr_rseq:_e
1499           \romannumeral-\`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2{#3}}%
1500 \def\XINT_expr_rseq:_e #1{%
1501   \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1502   \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1503   \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1504   \xint_orthat{\XINT_expr_rseq:_goon #1}}%
1505 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1506           \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1507 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1508 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{ }%
1509 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%
1510 \def\XINT_expr_rseq:_noop ,~#1#2{\XINT_expr_rseq:_b #1{#2}}%

```

1511 \def\XINT\_expr\_rseq:\_end ^,~#1#2{}% no nil for rseq

### 10.33.4 \XINT\_expr\_rseq:\_A etc...

n++ for rseq

```
1512 \def\XINT_expr_rseq:_A +#1!#2#3,^,{\XINT_expr_rseq:_D {#1}#3{#2}}%
1513 \def\XINT_expr_rseq:_D #1#2#3{\expandafter\XINT_expr_rseq:_E
1514   \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2{#3}}%
1515 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1516   \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1517   \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1518   \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1519 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1520   {,#1\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1521   \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1522 \def\XINT_expr_rseq:_Omit #1!#2#3~#4#5%
1523   {\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1524 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{}%
1525 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%
```

## 10.34 rrseq

When `func_rrseq` has its turn, initial segment has been scanned by `oparen`, the `;` mimicking the rôle of a closing parenthesis, and stopping further expansion.

```
1526 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval }%
1527 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
1528 \def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval }%
1529 \def\XINT_allexpr_rrseq #1#2%
1530 {%
1531   \expandafter\XINT_expr_rrseq\expandafter #1\expandafter
1532   #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
1533 }%
```

### 10.34.1 \XINT\_expr\_rrseqx

The `(#4)` is for `++` mechanism which must have its closing parenthesis.

```
1534 \def\XINT_expr_rrseqx #1#2#3#4%
1535 {%
1536   \expandafter\XINT_expr_rrseq\romannumeral0#1(#4)\expandafter\relax
1537   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1538     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
1539   #2#3#1%
1540 }%
```

### 10.34.2 \XINT\_expr\_rrseqy

```
1541 \def\XINT_expr_rrseqy #1#2#3#4#5#6% #1=valeurs pour variable (locked),
1542   % #2=initial values (reversed, one (braced) token each)
1543   % #3=toutes les valeurs initiales (csv,locked),
1544   % #4=variable, #5=expr,
1545   % #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval
```



```

1546 {%
1547   \expandafter \XINT_expr_getop
1548   \csname .=\XINT_expr_unlock #3%
1549   \expandafter\XINT_expr_rrseq:_aa
1550       \romannumeral-\`0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
1551 }%
1552 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1553     \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

### 10.34.3 \XINT\_expr\_rrseq:\_a etc...

```

1554 \catcode`? 3
1555 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1}%
1556 \def\XINT_expr_rrseq:_b #1#2#3,{\XINT_expr_rrseq:_c #3,~#1?{#2}}%
1557 \def\XINT_expr_rrseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rrseq:_noop\fi
1558     \if ^#1\xint_dothis\XINT_expr_rrseq:_end\fi
1559     \xint_orthat\XINT_expr_rrseq:_d #1}%
1560 \def\XINT_expr_rrseq:_d #1,~#2?#3{\expandafter\XINT_expr_rrseq:_e
1561     \romannumeral-\`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1562 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1563     {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1564 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1565 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{ }%
1566 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%
1567 \def\XINT_expr_rrseq:_noop ,~#1?#2{\XINT_expr_rrseq:_b {#1}{#2}}%
1568 \def\XINT_expr_rrseq:_end ^,~#1?#2{}% No nil for rrseq.
1569 \catcode`? 11
1570 \def\XINT_expr_rrseq:_e #1{%
1571     \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1572     \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1573     \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1574     \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1575 }%

```

### 10.34.4 \XINT\_expr\_rrseq:\_A etc...

#### n++ for rrseq

```

1576 \catcode`? 3
1577 \def\XINT_expr_rrseq:_A +#1!#2#3,^,{\XINT_expr_rrseq:_D {#1}{#3}{#2}}%
1578 \def\XINT_expr_rrseq:_D #1#2#3{\expandafter\XINT_expr_rrseq:_E
1579     \romannumeral-\`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1580 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5
1581     {,#1\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1582     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1583 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1584     {\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1585 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{}%
1586 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1587 \catcode`? 11
1588 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1589     \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1590     \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1591     \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

### 10.35 iter

```

1592 \def\XINT_expr_func_iter  {\XINT_allexpr_iter \xintbareeval    }%
1593 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval}%
1594 \def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval  }%
1595 \def\XINT_allexpr_iter #1#2%
1596 {%
1597   \expandafter\XINT_expr_iterx\expandafter #1\expandafter
1598   #2\romannumeral-\`0\XINT_expr_onlitteral_seq_a }%
1599 }%

```

#### 10.35.1 \XINT\_expr\_iterx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```

1600 \def\XINT_expr_iterx #1#2#3#4%
1601 {%
1602   \expandafter\XINT_expr_itery\romannumeral0#1(#4)\expandafter\relax
1603   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1604     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
1605   #2#3#1%
1606 }%

```

#### 10.35.2 \XINT\_expr\_itery

```

1607 \def\XINT_expr_itery #1#2#3#4#5#6% #1=valeurs pour variable (locked),
1608   % #2=initial values (reversed, one (braced) token each)
1609   % #3=toutes les valeurs initiales (csv,locked),
1610   % #4=variable, #5=expr,
1611   % #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval
1612 {%
1613   \expandafter \XINT_expr_getop
1614   \csname .=%
1615   \expandafter\XINT_expr_iter:_aa
1616     \romannumeral-\`0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
1617 }%
1618 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1619   \expandafter\XINT_expr_iter:_a\fi #1}%

```

#### 10.35.3 \XINT\_expr\_iter:\_a etc...

```

1620 \catcode`? 3
1621 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1}%
1622 \def\XINT_expr_iter:_b #1#2#3,{\XINT_expr_iter:_c #3,~#1?{#2}}%
1623 \def\XINT_expr_iter:_c #1{\if ,#1\xint_dothis\XINT_expr_iter:_noop\fi
1624   \if ^#1\xint_dothis\XINT_expr_iter:_end\fi
1625   \xint_orthat\XINT_expr_iter:_d #1}%
1626 \def\XINT_expr_iter:_d #1,~#2?#3{\expandafter\XINT_expr_iter:_e
1627   \romannumeral-\`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1628 \def\XINT_expr_iter:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iter:_b\expandafter
1629   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1630 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1631 \def\XINT_expr_iter:_abort #1!#2#3~#4?#5#6^,%
1632   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1633     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%

```

```

1634 \def\XINT_expr_iter:_break #1!#2#3~#4?#5#6^,%
1635   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1636     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1637 \def\XINT_expr_iter:_noop ,~#1?#2{\XINT_expr_iter:_b {#1}{#2}}%
1638 \def\XINT_expr_iter:_end ^,~#1?#2%
1639   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1640     {\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1641 \catcode`? 11
1642 \def\XINT_expr_iter:_e #1{%
1643   \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1644   \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1645   \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1646   \xint_orthat{\XINT_expr_iter:_goon #1}%
1647 }%
1648 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

#### 10.35.4 `\XINT_expr_iter:_A` etc...

##### n++ for iter

```

1649 \catcode`? 3
1650 \def\XINT_expr_iter:_A +#1!#2#3^,{\XINT_expr_iter:_D {#1}{#3}{#2}}%
1651 \def\XINT_expr_iter:_D #1#2#3{\expandafter\XINT_expr_iter:_E
1652   \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1653 \def\XINT_expr_iter:_Goon #1!#2#3~#4?#5%
1654   {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1655   \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1656 \def\XINT_expr_iter:_Omit #1!#2#3~%#4?#5%
1657   {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1658 \def\XINT_expr_iter:_Abort #1!#2#3~#4?#5%
1659   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1660     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1661 \def\XINT_expr_iter:_Break #1!#2#3~#4?#5%
1662   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1663     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1664 \catcode`? 11
1665 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1666   \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1667   \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1668   \xint_orthat{\XINT_expr_iter:_Goon #1}}%

```

## 10.36 Macros handling csv lists for functions with multiple comma separated arguments in expressions

These 17 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.

### 10.36.1 `\xintANDof:csv`

1.09a. For use by `\xintexpr` inside `\csname. 1.1`, je remplace `ifTrueAelseB` par `iiNotZero` pour des raisons d'optimisations.

```
1669 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral-`0#1,,^}%
1670 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1671     \else\expandafter\XINT_andof:_c\fi #1}%
1672 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1673 \def\XINT_andof:_no #1^{0}%
1674 \def\XINT_andof:_e #1^{1}% works with empty list
```

### 10.36.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```
1675 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral-`0#1,,^}%
1676 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1677     \else\expandafter\XINT_orof:_c\fi #1}%
1678 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1679 \def\XINT_orof:_yes #1^{1}%
1680 \def\XINT_orof:_e #1^{0}% works with empty list
```

### 10.36.3 `\xintXORof:csv`

1.09a. For use by `\xintexpr` (inside a `\csname..\endcsname`).

```
1681 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral-`0#1,,^}%
1682 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1683 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
1684     \else\expandafter\XINT_xorof:_c\fi #1}%
1685 \def\XINT_xorof:_c #1,#2%
1686     {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1687         \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1688         {\XINT_xorof:_a #2}%
1689     }%
1690 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)
```

### 10.36.4 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.

```
1691 \def\XINT_oncsv:_empty #1,^,#2{#2}%
1692 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
1693 \def\XINT_oncsv:_a #1#2#3%
1694     {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
1695 \def\XINT_oncsv:_b #1#2#3,%
1696     {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral-`0#2{#3}}#1#2}%
1697 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral-`0#4,{#1}#2#3}%
1698 \def\XINT_oncsv:_d #1%
1699     {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
1700 \def\XINT_oncsv:_e #1,#2#3#4%
1701     {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral-`0#3{#4{#1}}{#2}}#3#4}%

```

### 10.36.5 `\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 `xintiiexpr`, dans il faut bien sûr ne pas faire de `xintNum`, donc il faut un `iimax`.

```
1702 \def\xintMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmax
1703         \expandafter\xint_firstofone\romannumeral-`0#1,^{0/1[0]}}%
1704 \def\xintiiMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiiimax
1705         \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

### 10.36.6 `\xintMinof:csv`, `\xintiiMinof:csv`

1.09i. Rewritten for 1.1. For use by `\xintiiexpr`.

```
1706 \def\xintMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmin
1707         \expandafter\xint_firstofone\romannumeral-`0#1,^{0/1[0]}}%
1708 \def\xintiiMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiiimin
1709         \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

### 10.36.7 `\xintSum:csv`, `\xintiiSum:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
1710 \def\xintSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintadd
1711         \expandafter\xint_firstofone\romannumeral-`0#1,^{0/1[0]}}%
1712 \def\xintiiSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiiadd
1713         \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

### 10.36.8 `\xintPrd:csv`, `\xintiiPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
1714 \def\xintPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmul
1715         \expandafter\xint_firstofone\romannumeral-`0#1,^{1/1[0]}}%
1716 \def\xintiiPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiiimul
1717         \expandafter\xint_firstofone\romannumeral-`0#1,^,1}%
```

### 10.36.9 `\xintGCDof:csv`, `\xintLCMof:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`. Expansion réinstaurée pour besoins de `xintNewExpr` de version 1.1

```
1718 \def\xintGCDof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintgcd
1719         \expandafter\xint_firstofone\romannumeral-`0#1,^,1}%
1720 \def\xintLCMof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintlcm
1721         \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

### 10.36.10 `\xintiigCDof:csv`, `\xintiilCMof:csv`

1.1a pour `\xintiexpr`. Ces histoires de `ii` sont pénibles à la fin.

```
1722 \def\xintiigCDof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintiigcd
1723         \expandafter\xint_firstofone\romannumeral-`0#1,^,1}%
1724 \def\xintiilCMof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintiilcm
1725         \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

### 10.36.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`

for `\xintNewExpr` matters, mainly.

```
1726 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
1727 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt [\XINTdigits]}%
```

### 10.36.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h

```
1728 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintmax
1729         \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{0[0]}}%
1730 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintmin
1731         \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{0[0]}}%
```

### 10.36.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```
1732 \def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\XINTinfloatadd
1733         \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{0[0]}}%
1734 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\XINTinfloatmul
1735         \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{1[0]}}%
```

## 10.37 The `num`, `reduce`, `abs`, `sgn`, `frac`, `floor`, `ceil`, `sqr`, `sqrt`, `sqrt`, `float`, `round`, `trunc`, `mod`, `quo`, `rem`, `gcd`, `lcm`, `max`, `min`, ``+``, ``*``, `?`, `!`, `not`, `all`, `any`, `xor`, `if`, `ifsgn`, `first`, `last`, `even`, `odd`, and `reversed` functions

```
1736 \def\XINT_expr_twoargs #1,#2,{{#1}{#2}}%
1737 \def\XINT_expr_argandopt #1,#2,#3.#4#5%
1738 {%
1739     \if\relax#3\relax\expandafter\xint_firstoftwo\else
1740         \expandafter\xint_secondoftwo\fi
1741     {#4}{#5[\xintNum {#2}]}{#1}%
1742 }%
1743 \def\XINT_expr_oneortwo #1#2#3,#4,#5.%
1744 {%
1745     \if\relax#5\relax\expandafter\xint_firstoftwo\else
1746         \expandafter\xint_secondoftwo\fi
1747     {#1{0}}{#2{\xintNum {#4}}}{#3}%
1748 }%
1749 \def\XINT_iiexpr_oneortwo #1#2,#3,#4.%
1750 {%
```

Package *xintexpr* implementation

```

1751 \if\relax#4\relax\expandafter\xint_firstoftwo\else
1752 \expandafter\xint_secondoftwo\fi
1753 {#1{0}}{#1{#3}}{#2}%
1754 }%
1755 \def\xINT_expr_func_num #1#2#3%
1756 {\expandafter #1\expandafter #2\csname.=\xintNum {\xINT_expr_unlock #3}\endcsname }%
1757 \let\xINT_flexpr_func_num\xINT_expr_func_num
1758 \let\xINT_iiexpr_func_num\xINT_expr_func_num
1759 % [0] added Oct 25. For interaction with SPRaw::csv
1760 \def\xINT_expr_func_reduce #1#2#3%
1761 {\expandafter #1\expandafter #2\csname.=\xintIrr {\xINT_expr_unlock #3}[0]\endcsname }%
1762 \let\xINT_flexpr_func_reduce\xINT_expr_func_reduce
1763 % no \XINT_iiexpr_func_reduce
1764 \def\xINT_expr_func_abs #1#2#3%
1765 {\expandafter #1\expandafter #2\csname.=\xintAbs {\xINT_expr_unlock #3}\endcsname }%
1766 \let\xINT_flexpr_func_abs\xINT_expr_func_abs
1767 \def\xINT_iiexpr_func_abs #1#2#3%
1768 {\expandafter #1\expandafter #2\csname.=\xintiAbs {\xINT_expr_unlock #3}\endcsname }%
1769 \def\xINT_expr_func_sgn #1#2#3%
1770 {\expandafter #1\expandafter #2\csname.=\xintSgn {\xINT_expr_unlock #3}\endcsname }%
1771 \let\xINT_flexpr_func_sgn\xINT_expr_func_sgn
1772 \def\xINT_iiexpr_func_sgn #1#2#3%
1773 {\expandafter #1\expandafter #2\csname.=\xintiiSgn {\xINT_expr_unlock #3}\endcsname }%
1774 \def\xINT_expr_func_frac #1#2#3%
1775 {\expandafter #1\expandafter #2\csname.=\xintTFrac {\xINT_expr_unlock #3}\endcsname }%
1776 \def\xINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
1777 .=\XINTinFloatFracdigits {\xINT_expr_unlock #3}\endcsname }%
1778 % no \XINT_iiexpr_func_frac
1779 \def\xINT_expr_func_floor #1#2#3%
1780 {\expandafter #1\expandafter #2\csname .=\xintFloor {\xINT_expr_unlock #3}\endcsname }%
1781 \let\xINT_flexpr_func_floor\xINT_expr_func_floor
1782 \def\xINT_iiexpr_func_floor #1#2#3%
1783 {% mais absurde si on ne peut pas avoir quotient comme input
1784 \expandafter #1\expandafter #2\csname.=\xintiFloor {\xINT_expr_unlock #3}\endcsname }%
1785 \def\xINT_expr_func_ceil #1#2#3%
1786 {\expandafter #1\expandafter #2\csname .=\xintCeil {\xINT_expr_unlock #3}\endcsname }%
1787 \let\xINT_flexpr_func_ceil\xINT_expr_func_ceil
1788 \def\xINT_iiexpr_func_ceil #1#2#3%
1789 {% mais absurde si on ne peut pas avoir quotient comme input
1790 \expandafter #1\expandafter #2\csname.=\xintiCeil {\xINT_expr_unlock #3}\endcsname }%
1791 \def\xINT_expr_func_sqr #1#2#3%
1792 {\expandafter #1\expandafter #2\csname.=\xintSqr {\xINT_expr_unlock #3}\endcsname }%
1793 \def\xINT_flexpr_func_sqr #1#2#3%
1794 {%
1795 \expandafter #1\expandafter #2\csname
1796 .=\XINTinFloatMul % [\XINTdigits]% pour simplifier mes affaires avec \xintNewExpr
1797 {\xINT_expr_unlock #3}{\xINT_expr_unlock #3}\endcsname
1798 }%
1799 \def\xINT_iiexpr_func_sqr #1#2#3%
1800 {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\xINT_expr_unlock #3}\endcsname }%
1801 \def\xINT_expr_func_sqrt #1#2#3%
1802 {%

```

Package *xintexpr* implementation

```

1803 \expandafter #1\expandafter #2\csname .=%
1804 \expandafter\XINT_expr_argandopt
1805 \romannumeral-`0\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\XINTinFloatSqrt
1806 \endcsname
1807 }%
1808 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
1809 \def\XINT_iiexpr_func_sqrt #1#2#3%
1810 {\expandafter #1\expandafter #2\csname.=\xintiSqrt {\XINT_expr_unlock #3}\endcsname }%
1811 \def\XINT_iiexpr_func_sqrtr #1#2#3%
1812 {\expandafter #1\expandafter #2\csname.=\xintiSqrtr {\XINT_expr_unlock #3}\endcsname }%
1813 \def\XINT_expr_func_round #1#2#3%
1814 {%
1815 \expandafter #1\expandafter #2\csname .=%
1816 \expandafter\XINT_expr_oneortwo
1817 \expandafter\xintiRound\expandafter\xintRound
1818 \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1819 }%
1820 \let\XINT_flexpr_func_round\XINT_expr_func_round
1821 \def\XINT_iiexpr_func_round #1#2#3%
1822 {%
1823 \expandafter #1\expandafter #2\csname .=%
1824 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiRound
1825 \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1826 }%
1827 \def\XINT_expr_func_trunc #1#2#3%
1828 {%
1829 \expandafter #1\expandafter #2\csname .=%
1830 \expandafter\XINT_expr_oneortwo
1831 \expandafter\xintiTrunc\expandafter\xintTrunc
1832 \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1833 }%
1834 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
1835 \def\XINT_iiexpr_func_trunc #1#2#3%
1836 {%
1837 \expandafter #1\expandafter #2\csname .=%
1838 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
1839 \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1840 }%
1841 \def\XINT_expr_func_float #1#2#3%
1842 {%
1843 \expandafter #1\expandafter #2\csname .=%
1844 \expandafter\XINT_expr_argandopt
1845 \romannumeral-`0\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat
1846 \endcsname
1847 }%
1848 \let\XINT_flexpr_func_float\XINT_expr_func_float
1849 % \XINT_iiexpr_func_float not defined
1850 \def\XINT_expr_func_mod #1#2#3%
1851 {%
1852 \expandafter #1\expandafter #2\csname .=%
1853 \expandafter\expandafter\expandafter\xintMod
1854 \expandafter\XINT_expr_twoargs

```



```

1855 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1856 }%
1857 \def\XINT_flexpr_func_mod #1#2#3%
1858 {%
1859 \expandafter #1\expandafter #2\csname .=%
1860 \expandafter\XINTinFloatMod
1861 \romannumeral-`0\expandafter\XINT_expr_twoargs
1862 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1863 }%
1864 \def\XINT_iiexpr_func_mod #1#2#3%
1865 {%
1866 \expandafter #1\expandafter #2\csname .=%
1867 \expandafter\expandafter\expandafter\xintiMod
1868 \expandafter\XINT_expr_twoargs
1869 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1870 }%
1871 \def\XINT_expr_func_quo #1#2#3%
1872 {%
1873 \expandafter #1\expandafter #2\csname .=%
1874 \expandafter\expandafter\expandafter\xintiQuo
1875 \expandafter\XINT_expr_twoargs
1876 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1877 }%
1878 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
1879 \def\XINT_iiexpr_func_quo #1#2#3%
1880 {%
1881 \expandafter #1\expandafter #2\csname .=%
1882 \expandafter\expandafter\expandafter\xintiQuo
1883 \expandafter\XINT_expr_twoargs
1884 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1885 }%
1886 \def\XINT_expr_func_rem #1#2#3%
1887 {%
1888 \expandafter #1\expandafter #2\csname .=%
1889 \expandafter\expandafter\expandafter\xintiRem
1890 \expandafter\XINT_expr_twoargs
1891 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1892 }%
1893 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
1894 \def\XINT_iiexpr_func_rem #1#2#3%
1895 {%
1896 \expandafter #1\expandafter #2\csname .=%
1897 \expandafter\expandafter\expandafter\xintiRem
1898 \expandafter\XINT_expr_twoargs
1899 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1900 }%
1901 \def\XINT_expr_func_gcd #1#2#3%
1902 {\expandafter #1\expandafter #2\csname
1903 .=\xintGCDoF:csv{\XINT_expr_unlock #3}\endcsname }%
1904 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
1905 \def\XINT_iiexpr_func_gcd #1#2#3%
1906 {\expandafter #1\expandafter #2\csname

```

Package *xintexpr* implementation

```

1907                                     .=\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
1908 \def\XINT_expr_func_lcm #1#2#3%
1909   {\expandafter #1\expandafter #2\csname
1910                                     .=\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
1911 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
1912 \def\XINT_iiexpr_func_lcm #1#2#3%
1913   {\expandafter #1\expandafter #2\csname
1914                                     .=\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
1915 \def\XINT_expr_func_max #1#2#3%
1916   {\expandafter #1\expandafter #2\csname
1917                                     .=\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1918 \def\XINT_iiexpr_func_max #1#2#3%
1919   {\expandafter #1\expandafter #2\csname
1920                                     .=\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1921 \def\XINT_flexpr_func_max #1#2#3%
1922   {\expandafter #1\expandafter #2\csname
1923                                     .=\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1924 \def\XINT_expr_func_min #1#2#3%
1925   {\expandafter #1\expandafter #2\csname
1926                                     .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1927 \def\XINT_iiexpr_func_min #1#2#3%
1928   {\expandafter #1\expandafter #2\csname
1929                                     .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1930 \def\XINT_flexpr_func_min #1#2#3%
1931   {\expandafter #1\expandafter #2\csname
1932                                     .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1933 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
1934   {\expandafter #1\expandafter #2\csname
1935                                     .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
1936 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
1937   {\expandafter #1\expandafter #2\csname
1938                                     .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
1939 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
1940   {\expandafter #1\expandafter #2\csname
1941                                     .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
1942 \expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
1943   {\expandafter #1\expandafter #2\csname
1944                                     .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1945 \expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
1946   {\expandafter #1\expandafter #2\csname
1947                                     .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1948 \expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
1949   {\expandafter #1\expandafter #2\csname
1950                                     .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1951 \def\XINT_expr_func_? #1#2#3%
1952   {\expandafter #1\expandafter #2\csname
1953                                     .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
1954 \let\XINT_flexpr_func_? \XINT_expr_func_?
1955 \let\XINT_iiexpr_func_? \XINT_expr_func_?
1956 \def\XINT_expr_func_! #1#2#3%
1957   {\expandafter #1\expandafter #2\csname.\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
1958 \let\XINT_flexpr_func_! \XINT_expr_func_!

```

Package *xintexpr* implementation

```

1959 \let\XINT_iiexpr_func_! \XINT_expr_func_!
1960 \def\XINT_expr_func_not #1#2#3%
1961 {\expandafter #1\expandafter #2\csname.=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
1962 \let\XINT_flexpr_func_not \XINT_expr_func_not
1963 \let\XINT_iiexpr_func_not \XINT_expr_func_not
1964 \def\XINT_expr_func_all #1#2#3%
1965   {\expandafter #1\expandafter #2\csname
1966     .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
1967 \let\XINT_flexpr_func_all\XINT_expr_func_all
1968 \let\XINT_iiexpr_func_all\XINT_expr_func_all
1969 \def\XINT_expr_func_any #1#2#3%
1970   {\expandafter #1\expandafter #2\csname
1971     .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
1972 \let\XINT_flexpr_func_any\XINT_expr_func_any
1973 \let\XINT_iiexpr_func_any\XINT_expr_func_any
1974 \def\XINT_expr_func_xor #1#2#3%
1975   {\expandafter #1\expandafter #2\csname
1976     .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
1977 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
1978 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
1979 \def\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
1980 \def\XINT_expr_func_if #1#2#3%
1981   {\expandafter #1\expandafter #2\csname
1982     .=\expandafter\xintifNotZero:\romannumeral-`0\XINT_expr_unlock #3,\endcsname }%
1983 \let\XINT_flexpr_func_if\XINT_expr_func_if
1984 \let\XINT_iiexpr_func_if\XINT_expr_func_if
1985 \def\xintifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
1986 \def\XINT_expr_func_ifsgn #1#2#3%
1987 {%
1988   \expandafter #1\expandafter #2\csname
1989     .=\expandafter\xintifSgn:\romannumeral-`0\XINT_expr_unlock #3,\endcsname
1990 }%
1991 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
1992 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
1993 \def\XINT_expr_func_first #1#2#3%
1994   {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
1995     \romannumeral-`0\XINT_expr_unlock #3,\endcsname }%
1996 \def\XINT_expr_func_firsta #1,#2^{#1}%
1997 \let\XINT_flexpr_func_first\XINT_expr_func_first
1998 \let\XINT_iiexpr_func_first\XINT_expr_func_first
1999 \def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
2000   {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
2001     \romannumeral-`0\XINT_expr_unlock #3,\endcsname }%
2002 \def\XINT_expr_func_lasta #1,#2%
2003   {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
2004 \let\XINT_flexpr_func_last\XINT_expr_func_last
2005 \let\XINT_iiexpr_func_last\XINT_expr_func_last
2006 \def\XINT_expr_func_odd #1#2#3%
2007   {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
2008 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2009 \def\XINT_iiexpr_func_odd #1#2#3%
2010   {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%

```

```

2011 \def\XINT_expr_func_even #1#2#3%
2012   {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
2013 \let\XINT_flexpr_func_even\XINT_expr_func_even
2014 \def\XINT_iiexpr_func_even #1#2#3%
2015   {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
2016 \def\XINT_expr_func_nuple #1#2#3%
2017   {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname }%
2018 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2019 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
2020 \def\XINT_expr_func_reversed #1#2#3%
2021   {\expandafter #1\expandafter #2\csname .=\xintReversed::csv
2022                                     {\XINT_expr_unlock #3}\endcsname }%
2023 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2024 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2025 \def\xintReversed::csv #1% should be done directly, of course
2026   {\xintListWithSep,{\xintRevWithBraces {\xintCSVtoListNonStripped{#1}}}}%

```

## 10.38 f-expandable versions of the SeqB::csv routines, for `\xintNewExpr`

### 10.38.1 `\xintSeqB:f:csv`

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```

2027 \def\xintSeqB:f:csv #1#2%
2028   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintraw{#2}}{#1}}%
2029 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral-`0#2#1!}%
2030 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2031   \expandafter\xint_gobble_i\romannumeral-`0%
2032   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2033   #1{#3}{#4}}{#2}}%
2034 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2035 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2036                                     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2037 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2038                                     {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2039 \def\XINT_seqb:f:csv_p #1#2%
2040 {%
2041   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2042   {#1}{#2}}%
2043 }%
2044 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2045 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2046 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2047                                     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2048 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2049                                     {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2050 \def\XINT_seqb:f:csv_n #1#2%
2051 {%
2052   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2053   {#1}{#2}}%
2054 }%
2055 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2056 \def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%

```

### 10.38.2 `\xintiiSeqB:f:csv`

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```

2057 \def\xintiiSeqb:f:csv #1#2%
2058   {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral-`0#2}{#1}}%
2059 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral-`0#2#1!}%
2060 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2061   \expandafter\xint_gobble_i\romannumeral-`0%
2062   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2063   \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2064   #1{#3}{#4}{#2}}%
2065 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2066   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2067 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2068   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2069 \def\XINT_iiseqb:f:csv_p #1#2%
2070 {%
2071   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2072   \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2073 }%
2074 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2075 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2076 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2077   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2078 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2079   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2080 \def\XINT_iiseqb:f:csv_n #1#2%
2081 {%
2082   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2083   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2084 }%

```

### 10.38.3 `\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`.

```

2085 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2086   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}{#1}}%
2087 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral-`0#2#1!}%
2088 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2089   \expandafter\xint_gobble_i\romannumeral-`0%
2090   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2091   #1{#3}{#4}{#2}}%
2092 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2093   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2094 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2095   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2096 \def\XINT_flseqb:f:csv_p #1#2%
2097 {%
2098   \xintifCmp {#1}{#2}%

```

```

2099 \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2100 }%
2101 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2102 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2103 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2104 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2105 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2106 {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2107 \def\XINT_flseqb:f:csv_n #1#2%
2108 {%
2109 \xintifCmp {#1}{#2}%
2110 \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2111 }%

```

## 10.39 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

### 10.39.1 \xintApply::csv

Don't ask me what this is for. I wrote it in June, and we are now late October.

```

2112 \def\xintApply::csv #1#2%
2113 {\expandafter\XINT_applyon::_a\expandafter {\romannumeral-`0#2}{#1}}%
2114 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},}%
2115 \def\XINT_applyon::_b #1#2#3{\expandafter\XINT_applyon::_c \romannumeral-`0#3,{#1}{#2}}%
2116 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2117 \else\expandafter\XINT_applyon::_d\fi #1}%
2118 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral-`0#2{#1},{#2}}%
2119 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}%
2120 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%

```

### 10.39.2 \xintApply::csv

```

2121 \def\xintApply::csv #1#2#3%
2122 {\expandafter\XINT_applyon::_a\expandafter{\romannumeral-`0#2}{#1}{#3}}%
2123 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},}%
2124 \def\XINT_applyon::_b #1#2#3#4,%
2125 {\expandafter\XINT_applyon::_c \romannumeral-`0#4,{#1}{#2}{#3}}%
2126 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2127 \else\expandafter\XINT_applyon::_d\fi #1}%
2128 \def\XINT_applyon::_d #1,#2#3%
2129 {\expandafter\XINT_applyon::_e\expandafter
2130 {\romannumeral-`0\xintApply::csv {#2}{#1}{#3}},{#2}{#3}}%
2131 \def\XINT_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4}, #1}%
2132 \def\XINT_applyon::_end #1,#2#3#4{\xint_secondoftwo #4}%

```

### 10.39.3 \XINT\_expr\_RApply::csv, \XINT\_expr\_LApply::csv, \XINT\_expr\_RLApply::csv

The #1 in `_RApply` will start with a `~`. No risk of glueing to previous `~expandafter` during the `\scantokens`.

```

2133 \def\XINT_expr_RApply::csv #1#2#3#4%
2134 {~\xintApply::csv{~\expandafter#1~\xint_exchangetwo_keepbraces{#4}}{#3}}%
2135 \def\XINT_expr_LApply::csv #1#2#3#4{~\xintApply::csv{#1}{#3}}{#4}}%
2136 \def\XINT_expr_RLApply::csv #1#2{~\xintApply::csv{#1}}%

```

### 10.39.4 Mysterious stuff

actually I dimly remember that the whole point is to allow maximal evaluation as long as macro parameters not encountered. Else it would be easier. `\xintNewIExpr \f [2]{[12] #1+#2+3*6*1}` will correctly compute the 18.

1.1a re-establishes the trick with `\toks0\expandafter{\the\toks0\expandafter etc...}` with `*no*` space after 0. I don't know why it was removed at some point before releasing 1.1, but `\XINT_expr_redefinemacros` is even bigger without the trick.

```

2137 \catcode`~ 12 % by the way, catcode is set to 3 in \XINTsetupcatcodes
2138 \catcode`$ 12 % $
2139 \def\xint_xptwo_getab_b #1#2!#3%
2140   {\expandafter\xint_xptwo_getab_c\romannumeral-`0#3!#1{#1#2}}%
2141 \def\xint_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2142 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2143 \def\xint_NEfork #1#2{\xint_ddfork
2144   #1#2\xint_expr_RLApply::csv
2145   #1$\xint_expr_RApply::csv% $
2146   #2\xint_expr_LApply::csv% $
2147   $$\xint_NEfork_nn #1#2}% $$
2148   \krof }%
2149 \def\xint_NEfork_nn #1#2#3#4{%
2150   \if #1#\xint_dothis{#3}\fi
2151   \if #1~\xint_dothis{#3}\fi
2152   \if #2#\xint_dothis{#3}\fi
2153   \if #2~\xint_dothis{#3}\fi
2154   \xint_orthat {\csname #4NE\endcsname }%
2155   }%
2156 \def\xint_NEfork_one #1#2!#3#4#5#6{%
2157   \if ###\xint_dothis {#3}\fi
2158   \if ~#\xint_dothis {#3}\fi
2159   \if $#1\xint_dothis {\xintApply::csv{#3#5}}\fi %$
2160   \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2161 }%
2162 \toks0 {}%
2163 \xintFor #1 in {DivTrunc,iiDivTrunc,iiDivRound,Mod,iiMod,iRound,Round,iTrunc,Trunc,%
2164   Lt,Gt,Eq,LtorEq,GtorEq,Neq,AND,OR,XOR,iQuo,iRem,Add,Sub,Mul,Div,Pow,E,%
2165   iiAdd,iiSub,iiMul,iiPow,iiQuo,iiRem,iiE,SeqA::csv,iiSeqA::csv}\do
2166 {\toks0
2167   \expandafter{\the\toks0% no space! (makes shorter macro in the end)
2168   \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2169   \endcsname\expandafter\def\csname xint#1\endcsname ###1###2{%
2170     \expandafter\xint_NEfork
2171     \romannumeral-`0\expandafter\xint_xptwo_getab_b
2172     \romannumeral-`0###2!{###1}{~xint#1}{xint#1}}%
2173   }%
2174 }%
2175 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2176   Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%
2177   Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,Fac,iFac,Bool,Toggle}\do
2178 {\toks0
2179   \expandafter{\the\toks0%
2180   \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter

```

Package *xintexpr* implementation

```

2181 \endcsname\expandafter\def\csname xint#1\endcsname ####1{%
2182   \expandafter\XINT_NEfork_one\romannumeral-`0####1!{~xint#1}{xint#1}{}}}%
2183 }%
2184 }%
2185 \xintFor #1 in {Add,Sub,Mul,Div,Power,E,Mod,SeqA::csv}\do
2186 {\toks0
2187   \expandafter{\the\toks0}
2188   \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2189     \csname XINTinFloat#1\expandafter\endcsname
2190   \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
2191     \expandafter\XINT_NEfork
2192     \romannumeral-`0\expandafter\XINT_xptwo_getab_b
2193     \romannumeral-`0####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}}%
2194 }%
2195 }%
2196 \xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits}\do
2197 {\toks0
2198   \expandafter{\the\toks0}
2199   \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2200 \endcsname\expandafter\def\csname #1\endcsname ####1{\expandafter
2201   \XINT_NEfork_one\romannumeral-`0####1!{~#1}{#1}{}}}%
2202 }%
2203 }%
2204 \xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
2205 {\toks0
2206   \expandafter{\the\toks0} no space
2207   \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2208 \endcsname\expandafter\def\csname #1\endcsname ####1####2{%
2209     \expandafter\XINT_NEfork
2210     \romannumeral-`0\expandafter\XINT_xptwo_getab_b
2211     \romannumeral-`0####2!{####1}{$noexpand$#1}{#1}}}%
2212 }%
2213 }%
2214 \xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
2215 {\toks0
2216   \expandafter{\the\toks0} no space
2217   \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
2218 \endcsname\expandafter\def\csname #1::csv\endcsname ####1####2{%
2219     \expandafter\XINT_NEfork
2220     \romannumeral-`0\expandafter\XINT_xptwo_getab_b
2221     \romannumeral-`0####2!{####1}{$noexpand$#1:f:csv}{#1::csv}}}%
2222 }%
2223 }%
2224 \toks0
2225 \expandafter{\the\toks0}
2226 \let\XINTinFloatNE\XINTinFloat
2227 \def\XINTinFloat [#1]##2{% not ultimately general, but got tired
2228   \expandafter\XINT_NEfork_one
2229   \romannumeral-`0##2!{~XINTinFloat[#1]}{XINTinFloat}{}{[#1]}}}%
2230 \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2231 \def\XINTinFloatSqrt [#1]##2{%
2232   \expandafter\XINT_NEfork_one

```



Package *xintexpr* implementation

```

2233     \romannumeral-`0##2!{~XINTinFloatSqrt[#1]}{XINTinFloatSqrt}{#1}}%
2234 }%
2235 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2236             GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2237 {\toks0
2238 \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {~xint#1:csv}}%
2239 }%
2240 \xintFor #1 in {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2241 {\toks0
2242 \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {~#1:csv}}%
2243 }%
2244 \expandafter\def\expandafter\XINT_expr_redefinmacros\expandafter
2245 {\the\toks0
2246 \def\XINT_flexpr_noopt {\expandafter\XINT_flexpr_withopt_b\expandafter-%
2247             \romannumeral0\xintbarefloateval }%
2248 \def\XINT_flexpr_withopt_b ##1##2%
2249     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2250 \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!{\if -##2\expandafter\xint_firstoftwo
2251     \else\expandafter\xint_secondoftwo\fi \XINTdigits{##2##3}{##4}}%
2252 \def\XINT_expr_print ##1{\expandafter\xintSPRaw::csv\expandafter
2253     {\romannumeral-`0\XINT_expr_unlock ##1}}%
2254 \def\XINT_iiexpr_print ##1{\expandafter\xintCSV::csv\expandafter
2255     {\romannumeral-`0\XINT_expr_unlock ##1}}%
2256 \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2257     {\romannumeral-`0\XINT_expr_unlock ##1}}%
2258 \def\xintCSV::csv {~xintCSV::csv }% spaces to separate from possible catcode 11
2259 \def\xintSPRaw::csv {~xintSPRaw::csv }% stuff after
2260 \def\xintPFloat::csv {~xintPFloat::csv }%
2261 \def\xintIsTrue::csv {~xintIsTrue::csv }%
2262 \def\xintRound::csv {~xintRound::csv }%
2263 % \def\XINTinFloat::csv {~XINTinFloat::csv }% should not be needed.
2264 \def\xintReversed::csv {~xintReversed::csv }%
2265 \def\xintListSel:csv {~xintListSel:csv }%
2266 }%
2267 \toks0 {}%
2268 \def\xintNewExpr {\xint_NewExpr\xinttheexpr }%
2269 \def\xintNewFloatExpr {\xint_NewExpr\xintthefloatexpr }%
2270 \def\xintNewIExpr {\xint_NewExpr\xinttheiexpr }%
2271 % \let\xintNewNumExpr\xintNewIExpr % made obsolete for 1.1 release
2272 \def\xintNewIIExpr {\xint_NewExpr\xinttheiiexpr }%
2273 \def\xintNewBoolExpr {\xint_NewExpr\xinttheboolexpr }%
2274 \def\XINT_newexpr_finish #1>{\noexpand\romannumeral-`0}%
2275 \def\xint_NewExpr #1#2[#3]%
2276 {%
2277 \begingroup
2278 \ifcase #3\relax
2279     \toks0 {\xdef #2}%
2280 \or \toks0 {\xdef #2##1}%
2281 \or \toks0 {\xdef #2##1##2}%
2282 \or \toks0 {\xdef #2##1##2##3}%
2283 \or \toks0 {\xdef #2##1##2##3##4}%
2284 \or \toks0 {\xdef #2##1##2##3##4##5}%

```

Package *xintexpr* implementation

```

2285 \or \toks0 {\xdef #2##1##2##3##4##5##6}%
2286 \or \toks0 {\xdef #2##1##2##3##4##5##6##7}%
2287 \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8}%
2288 \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8##9}%
2289 \fi
2290 \xintexprSafeCatcodes
2291 \XINT_NewExpr #1%
2292 }%
2293 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
2294 \def\XINT_NewExpr %1%2@
2295 {@
2296 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%2}@
2297 \XINT_expr_redefinmacros
2298 \def~{ $noexpand$}@
2299 \catcode` : 11 \catcode`_ 11
2300 \catcode`# 12 \catcode`~ 13 \escapechar 126
2301 \endlinechar -1 \everyeof {\noexpand }@
2302 \edef\XINT_tmpb
2303 {\scantokens\expandafter
2304 {\romannumeral-`0\expandafter%1\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2305 }@
2306 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
2307 \the\toks0
2308 {\scantokens\expandafter{\expandafter\XINT_newexpr_finish\meaning\XINT_tmpb}}@
2309 \endgroup
2310 }@
2311 \catcode`% 14
2312 \let\xintexprRestoreCatcodes\empty
2313 \def\xintexprSafeCatcodes
2314 {%
2315 \edef\xintexprRestoreCatcodes {%
2316 \catcode59=\the\catcode59 % ;
2317 \catcode34=\the\catcode34 % "
2318 \catcode63=\the\catcode63 % ?
2319 \catcode124=\the\catcode124 % |
2320 \catcode38=\the\catcode38 % &
2321 \catcode33=\the\catcode33 % !
2322 \catcode93=\the\catcode93 % ]
2323 \catcode91=\the\catcode91 % [
2324 \catcode94=\the\catcode94 % ^
2325 \catcode95=\the\catcode95 % _
2326 \catcode47=\the\catcode47 % /
2327 \catcode41=\the\catcode41 % )
2328 \catcode40=\the\catcode40 % (
2329 \catcode42=\the\catcode42 % *
2330 \catcode43=\the\catcode43 % +
2331 \catcode62=\the\catcode62 % >
2332 \catcode60=\the\catcode60 % <
2333 \catcode58=\the\catcode58 % :
2334 \catcode46=\the\catcode46 % .
2335 \catcode45=\the\catcode45 % -
2336 \catcode44=\the\catcode44 % ,

```

Package *xintexpr* implementation

```
2337     \catcode61=\the\catcode61 % =
2338     \catcode32=\the\catcode32\relax % space
2339 }%
2340     \catcode59=12 % ;
2341     \catcode34=12 % "
2342     \catcode63=12 % ?
2343     \catcode124=12 % |
2344     \catcode38=4 % &
2345     \catcode33=12 % !
2346     \catcode93=12 % ]
2347     \catcode91=12 % [
2348     \catcode94=7 % ^
2349     \catcode95=8 % _
2350     \catcode47=12 % /
2351     \catcode41=12 % )
2352     \catcode40=12 % (
2353     \catcode42=12 % *
2354     \catcode43=12 % +
2355     \catcode62=12 % >
2356     \catcode60=12 % <
2357     \catcode58=12 % :
2358     \catcode46=12 % .
2359     \catcode45=12 % -
2360     \catcode44=12 % ,
2361     \catcode61=12 % =
2362     \catcode32=10 % space
2363 }%
2364 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
2365 \XINT_restorecatcodes_endinput%

xintkernel: 224. Total number of code lines: 12288. Each package starts with
xinttools:1045. circa 50 lines dealing with catcodes, package identification
xintcore:2074. and reloading management, also for Plain TeX. Version 1.1a of
xint:1556. 2014/11/07.
xintbinhex: 609.
xintgcd: 455.
xintfrac:2553.
xintseries: 386.
xintcfraction:1021.
xintexpr:2365.
```