

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.1c (2015/09/12); documentation date: 2015/09/17.

From source file xint.dtx. Time-stamp: <17-09-2015 at 11:09:20 CEST>.

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	85
5	Package <code>xintbinhex</code> implementation	119
6	Package <code>xintgcd</code> implementation	132
7	Package <code>xintfrac</code> implementation	144
8	Package <code>xintseries</code> implementation	201
9	Package <code>xintcfrac</code> implementation	211
10	Package <code>xintexpr</code> implementation	233

This is 1.1c of 2015/09/12.

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, ε -TeX and reload detection	2	.7	<code>\xint_dothis</code> , <code>\xint_orthat</code>	5
.2	Package identification	4	.8	<code>\xint_zapspaces</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, ϵ -TeX and reload detection

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

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

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 [2015/09/12 v1.1c 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{}}
```

```

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_zapspace

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

```
\xint_zapspace 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_zapspace`.

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_zapspace foo \xint_bye\xint_bye
```

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, ε -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>\xintgfdef</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>\xintilloop</code> , <code>\xintilloopindex</code> , <code>\xintouter-</code> <code>illoopindex</code> , <code>\xintbreakilloop</code> , <code>\xintbreak-</code> <code>illoopanddo</code> , <code>\xintilloopskiptonext</code> , <code>\xin-</code> <code>tilloopskipandredo</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, ϵ -TeX and reload detection

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

```

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

```



```

42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

2.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2015/09/12 v1.1c Expandable and non-expandable utilities (jfb)]%
```

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

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

2.3 \xintgodef, \xintgoodef, \xintgfdef

1.09i. For use in \xintAssign.

```

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

```

2.4 \xintRevWithBraces

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

```

52 \def\xintRevWithBraces {\romannumeral0\xintrevwithbraces }%
53 \def\xintRevWithBracesNoExpand {\romannumeral0\xintrevwithbracesnoexpand }%
54 \long\def\xintrevwithbraces #1%
55 {%
56     \expandafter\XINT_revwbr_loop\expandafter{\expandafter}%
57     \romannumeral-`0#1\xint_relax\xint_relax\xint_relax\xint_relax
58         \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
59 }%
60 \long\def\xintrevwithbracesnoexpand #1%
61 {%
62     \XINT_revwbr_loop }%
63     #1\xint_relax\xint_relax\xint_relax\xint_relax
64     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
65 }%
66 \long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
67 {%
68     \xint_gob_til_xint_relax #9\XINT_revwbr_finish_a\xint_relax
69     \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}#1}%
70 }%
71 \long\def\XINT_revwbr_finish_a\xint_relax\XINT_revwbr_loop #1#2\xint_bye
72 {%
73     \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\R\Z #1%
74 }%
75 \def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
76 {%

```

```

77 \xint_gob_til_R
78 #1\XINT_revwbr_finish_c \xint_gobble_viii
79 #2\XINT_revwbr_finish_c \xint_gobble_vii
80 #3\XINT_revwbr_finish_c \xint_gobble_vi
81 #4\XINT_revwbr_finish_c \xint_gobble_v
82 #5\XINT_revwbr_finish_c \xint_gobble_iv
83 #6\XINT_revwbr_finish_c \xint_gobble_iii
84 #7\XINT_revwbr_finish_c \xint_gobble_ii
85 \R\XINT_revwbr_finish_c \xint_gobble_i\Z
86 }%

```

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

```

87 \edef\XINT_revwbr_finish_c #1#2\Z {\noexpand\expandafter\space #1}%

```

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.

```

88 \def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%

```

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

```

89 \long\edef\xintzapfirstspaces #1%
90 {\noexpand\XINT_zapbsp_a \space #1\xint_relax \space\space\xint_relax }%
91 \xint_firstofone {\long\edef\XINT_zapbsp_a #1 } %<- space token here
92 {%

```

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

```

93 \noexpand\XINT_zapbsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapbsp_b #1\space\space
94 }%
95 \long\def\XINT_zapbsp_again? #1{\xint_bye #1\XINT_zapbsp_again }%
96 \xint_firstofone{\def\XINT_zapbsp_again\XINT_zapbsp_b} {\XINT_zapbsp_a }%
97 \long\def\XINT_zapbsp_b #1\xint_relax #2\xint_relax {#1}%

```

2.6 \xintZapLastSpaces

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

```

98 \def\xintZapLastSpaces {\romannumeral0\xintzaplastspaces }%

```

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

```

99 \long\edef\xintzaplastspaces #1{\noexpand\XINT_zapbsp_a {} \noexpand\empty#1%
100 \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.

```
101 \xint_firstofone {\long\def\XINT_zapesp_a #1#2 } %<- second space here
102     {\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 ?;-).

```
103 \long\def\XINT_zapesp_b #1#2#3\xint_relax
104     {\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`.

```
105 \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.

```
106 \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.

```
107 \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`.

```
108 \def\xintZapSpaces {\romannumeral0\xintzapspaces }%
109 \long\edef\xintzapspaces #1% like \xintZapFirstSpaces.
110     {\noexpand\XINT_zapsp_a \space #1\xint_relax \space\space\xint_relax }%
111 \xint_firstofone {\long\edef\XINT_zapsp_a #1 } %
112     {\noexpand\XINT_zapsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapsp_b #1\space\space}%
113 \long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again }%
114 \xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a }%
115 \xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
116 \long\edef\XINT_zapsp_c #1\xint_relax #2\xint_relax {\noexpand\XINT_zapesp_a
117     {} \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).

```
118 \def\xintZapSpacesB {\romannumeral0\xintzapspacesb }%
119 \long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint_relax\xint_relax
```

```

120             \xint_bye\xintzapspaces {#1}}%
121 \long\def\XINT_zapspb_one? #1#2%
122   {\xint_gob_til_xint_relax #1\XINT_zapspb_onlyspaces\xint_relax
123     \xint_gob_til_xint_relax #2\XINT_zapspb_bracedorone\xint_relax
124     \xint_bye {#1}}%
125 \def\XINT_zapspb_onlyspaces\xint_relax
126   \xint_gob_til_xint_relax\xint_relax\XINT_zapspb_bracedorone\xint_relax
127   \xint_bye #1\xint_bye\xintzapspaces #2{ }%
128 \long\def\XINT_zapspb_bracedorone\xint_relax
129   \xint_bye #1\xint_relax\xint_bye\xintzapspaces #2{ #1}%

```

2.9 \xintCSVtoList, \xintCSVtoListNonStripped

\xintCSVtoList transforms a,b,\dots,z into $\{a\}\{b\}\dots\{z\}$. 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.

```

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

```

163 }%

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

```

164 \def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
165 {%
166   \xint_gob_til_R
167     #1\expandafter\XINT_csvtol_finish_dviii\xint_gob_til_Z
168     #2\expandafter\XINT_csvtol_finish_dvii \xint_gob_til_Z
169     #3\expandafter\XINT_csvtol_finish_dvi  \xint_gob_til_Z
170     #4\expandafter\XINT_csvtol_finish_dv   \xint_gob_til_Z
171     #5\expandafter\XINT_csvtol_finish_div  \xint_gob_til_Z
172     #6\expandafter\XINT_csvtol_finish_diii \xint_gob_til_Z
173     #7\expandafter\XINT_csvtol_finish_dii  \xint_gob_til_Z
174     \R\XINT_csvtol_finish_di \Z
175 }%
176 \long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
177 \long\def\XINT_csvtol_finish_dvii  #1#2#3#4#5#6#7#8#9{ #9{#1}}%
178 \long\def\XINT_csvtol_finish_dvi   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
179 \long\def\XINT_csvtol_finish_dv    #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
180 \long\def\XINT_csvtol_finish_div   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
181 \long\def\XINT_csvtol_finish_diii  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
182 \long\def\XINT_csvtol_finish_dii   #1#2#3#4#5#6#7#8#9%
183                                     { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
184 \long\def\XINT_csvtol_finish_di\Z  #1#2#3#4#5#6#7#8#9%
185                                     { #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.

```

186 \def\xintListWithSep {\romannumeral0\xintlistwithsep}%
187 \def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand}%
188 \long\def\xintlistwithsep #1#2%
189   {\expandafter\XINT_lws\expandafter {\romannumeral-`0#2}{#1}}%
190 \long\def\XINT_lws #1#2{\XINT_lws_start {#2}#1\xint_bye}%
191 \long\def\xintlistwithsepnoexpand #1#2{\XINT_lws_start {#1}#2\xint_bye}%
192 \long\def\XINT_lws_start #1#2%
193 {%
194   \xint_bye #2\XINT_lws_dont\xint_bye
195   \XINT_lws_loop_a {#2}{#1}%
196 }%
197 \long\def\XINT_lws_dont\xint_bye\XINT_lws_loop_a #1#2{ }%
198 \long\def\XINT_lws_loop_a #1#2#3%
199 {%
200   \xint_bye #3\XINT_lws_end\xint_bye
201   \XINT_lws_loop_b {#1}{#2#3}{#2}%
202 }%
203 \long\def\XINT_lws_loop_b #1#2{\XINT_lws_loop_a {#1#2}}%
204 \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.

```

205 \def\xintNthElt          {\romannumeral0\xintnthelt }%
206 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand}%
207 \def\xintnthelt #1#2%
208 {%
209   \expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
210   \expandafter{\romannumeral-`0#2}%
211 }%
212 \def\xintntheltnoexpand #1%
213 {%
214   \expandafter\XINT_nthelt_a\the\numexpr #1.%
215 }%
216 \def\XINT_nthelt_a #1#2.%
217 {%
218   \xint_UDzerominusfork
219   #1-{\XINT_nthelt_bzero}%
220   0#1{\XINT_nthelt_bneg {#2}}%
221   0-{\XINT_nthelt_bpos {#1#2}}%
222   \krof
223 }%
224 \long\def\XINT_nthelt_bzero #1%
225 {%
226   \XINT_length_loop 0.#1\xint_relax\xint_relax\xint_relax\xint_relax
227   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
228 }%
229 \long\def\XINT_nthelt_bneg #1#2%
230 {%
231   \expandafter\XINT_nthelt_loop_a\expandafter {\the\numexpr #1\expandafter}%
232   \romannumeral0\xintrevwithbracesnoexpand {#2}%
233   \xint_relax\xint_relax\xint_relax\xint_relax
234   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
235 }%
236 \long\def\XINT_nthelt_bpos #1#2%
237 {%
238   \XINT_nthelt_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
239   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
240 }%
241 \def\XINT_nthelt_loop_a #1%
242 {%
243   \ifnum #1>\xint_c_viii
244     \expandafter\XINT_nthelt_loop_b

```

```

245 \else
246 \XINT_nthelt_getit
247 \fi
248 {#1}%
249 }%
250 \long\def\XINT_nthelt_loop_b #1#2#3#4#5#6#7#8#9%
251 {%
252 \xint_gob_til_xint_relax #9\XINT_nthelt_silentend\xint_relax
253 \expandafter\XINT_nthelt_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
254 }%
255 \def\XINT_nthelt_silentend #1\xint_bye { }%
256 \def\XINT_nthelt_getit\fi #1%
257 {%
258 \fi\expandafter\expandafter\expandafter\XINT_nthelt_finish
259 \csname xint_gobble_\romannumeral\numexpr#1-\xint_c_i\endcsname
260 }%
261 \long\edef\XINT_nthelt_finish #1#2\xint_bye
262 {\noexpand\xint_gob_til_xint_relax #1\noexpand\expandafter\space
263 \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.

```

264 \def\xintKeep {\romannumeral0\xintkeep }%
265 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
266 \def\xintkeep #1#2%
267 {%
268 \expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
269 \expandafter{\romannumeral-\`0#2}%
270 }%
271 \def\xintkeepnoexpand #1%
272 {%
273 \expandafter\XINT_keep_a\the\numexpr #1.%
274 }%
275 \def\XINT_keep_a #1#2.%
276 {%
277 \xint_UDzerominusfork
278 #1-{\expandafter\space\xint_gobble_i }%
279 0#1{\XINT_keep_bneg_a {#2}}%
280 0-{\XINT_keep_bpos {#1#2}}%
281 \krof
282 }%
283 \long\def\XINT_keep_bneg_a #1#2%
284 {%
285 \expandafter\XINT_keep_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%

```

2 Package `xinttools` implementation

```

286 }%
287 \def\XINT_keep_bneg_b #1#2.%
288 {%
289     \xint_UDzerominusfork
290     #1-{\xint_firstofone_thenstop }%
291     0#1{\xint_firstofone_thenstop }%
292     0-{\XINT_trim_bpos {#1#2}}%
293     \krof
294 }%
295 \long\def\XINT_keep_bpos #1#2%
296 {%
297     \XINT_keep_loop_a {#1}{}#2\xint_relax\xint_relax\xint_relax\xint_relax
298     \xint_relax\xint_relax\xint_relax\xint_bye
299 }%
300 \def\XINT_keep_loop_a #1%
301 {%
302     \ifnum #1>\xint_c_vi
303         \expandafter\XINT_keep_loop_b
304     \else
305         \XINT_keep_finish
306     \fi
307     {#1}%
308 }%
309 \long\def\XINT_keep_loop_b #1#2#3#4#5#6#7#8#9%
310 {%
311     \xint_gob_til_xint_relax #9\XINT_keep_enda\xint_relax
312     \expandafter\XINT_keep_loop_c\expandafter{\the\numexpr #1-\xint_c_vii}%
313     {{#3}{#4}{#5}{#6}{#7}{#8}{#9}}{#2}%
314 }%
315 \long\def\XINT_keep_loop_c #1#2#3{\XINT_keep_loop_a {#1}{#3#2}}%
316 \long\def\XINT_keep_enda\xint_relax
317     \expandafter\XINT_keep_loop_c\expandafter #1#2#3#4\xint_bye
318 {%
319     \XINT_keep_endb #4\W\W\W\W\W\W\W\Z #2{#3}%
320 }%
321 \def\XINT_keep_endb #1#2#3#4#5#6#7\Z
322 {%
323     \xint_gob_til_W
324     #1\XINT_keep_endc_
325     #2\XINT_keep_endc_i
326     #3\XINT_keep_endc_ii
327     #4\XINT_keep_endc_iii
328     #5\XINT_keep_endc_iv
329     #6\XINT_keep_endc_v
330     \W\XINT_keep_endc_vi\Z
331 }%
332 \long\def\XINT_keep_endc_ #1\Z #2#3#4#5#6#7#8#9{ #9}%
333 \long\def\XINT_keep_endc_i #1\Z #2#3#4#5#6#7#8#9{ #9{#2}}%
334 \long\def\XINT_keep_endc_ii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}}%
335 \long\def\XINT_keep_endc_iii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}}%
336 \long\def\XINT_keep_endc_iv #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}}%
337 \long\def\XINT_keep_endc_v #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}{#6}}%

```



```

338 \long\def\XINT_keep_endc_vi\Z #1#2#3#4#5#6#7#8{ #8{#1}{#2}{#3}{#4}{#5}{#6}}%
339 \long\def\XINT_keep_finish\fi #1#2#3#4#5#6#7#8#9\xint_bye
340 {%
341   \fi\XINT_keep_finish_loop_a {#1}{#3}{#4}{#5}{#6}{#7}{#8}\Z {#2}%
342 }%
343 \def\XINT_keep_finish_loop_a #1%
344 {%
345   \xint_gob_til_zero #1\XINT_keep_finish_z0%
346   \expandafter\XINT_keep_finish_loop_b\expandafter
347     {\the\numexpr #1-\xint_c_i}%
348 }%
349 \long\def\XINT_keep_finish_z0%
350   \expandafter\XINT_keep_finish_loop_b\expandafter #1#2#3\Z #4{ #4#2}%
351 \long\def\XINT_keep_finish_loop_b #1#2#3%
352 {%
353   \xint_gob_til_xint_relax #3\XINT_keep_finish_exit\xint_relax
354   \XINT_keep_finish_loop_c {#1}{#2}{#3}%
355 }%
356 \long\def\XINT_keep_finish_exit\xint_relax
357   \XINT_keep_finish_loop_c #1#2#3\Z #4{ #4#2}%
358 \long\def\XINT_keep_finish_loop_c #1#2#3%
359   {\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.

```

360 \def\xintTrim      {\romannumeral0\xinttrim }%
361 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
362 \def\xinttrim #1#2%
363 {%
364   \expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
365   \expandafter{\romannumeral-\`0#2}%
366 }%
367 \def\xinttrimnoexpand #1%
368 {%
369   \expandafter\XINT_trim_a\the\numexpr #1.%
370 }%
371 \def\XINT_trim_a #1#2.%
372 {%
373   \xint_UDzerominusfork
374     #1-{\xint_firstofone_thenstop }%
375     0#1{\XINT_trim_bneg_a {#2}}%
376     0-{\XINT_trim_bpos {#1#2}}%
377   \krof
378 }%

```

```

379 \long\def\XINT_trim_bneg_a #1#2%
380 {%
381   \expandafter\XINT_trim_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
382 }%
383 \def\XINT_trim_bneg_b #1#2.%
384 {%
385   \xint_UDzerominusfork
386     #1-{\expandafter\space\xint_gobble_i }%
387     0#1{\expandafter\space\xint_gobble_i }%
388     0-{\XINT_keep_bpos {#1#2}}%
389   \krof
390 }%
391 \long\def\XINT_trim_bpos #1#2%
392 {%
393   \XINT_trim_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
394     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
395 }%
396 \def\XINT_trim_loop_a #1%
397 {%
398   \ifnum #1>\xint_c_vii
399     \expandafter\XINT_trim_loop_b
400   \else
401     \XINT_trim_finish
402   \fi
403   {#1}%
404 }%
405 \long\def\XINT_trim_loop_b #1#2#3#4#5#6#7#8#9%
406 {%
407   \xint_gob_til_xint_relax #9\XINT_trim_silentend\xint_relax
408   \expandafter\XINT_trim_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
409 }%
410 \def\XINT_trim_silentend #1\xint_bye { }%
411 \def\XINT_trim_finish\fi #1%
412 {%
413   \fi\expandafter\expandafter\expandafter\XINT_trim_finish_a
414   \expandafter\expandafter\expandafter\space % avoids brace removal
415   \csname xint_gobble_\romannumeral\numexpr#1\endcsname
416 }%
417 \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.

```

418 \def\xintApply      {\romannumeral0\xintapply }%
419 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
420 \long\def\xintapply #1#2%
421 {%
422   \expandafter\XINT_apply\expandafter {\romannumeral-`0#2}%
423   {#1}%
424 }%

```

```

425 \long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
426 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
427 \long\def\XINT_apply_loop_a #1#2#3%
428 {%
429   \xint_bye #3\XINT_apply_end\xint_bye
430   \expandafter
431   \XINT_apply_loop_b
432   \expandafter {\romannumeral-`0#2{#3}}{#1}{#2}%
433 }%
434 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2}#1}%
435 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
436   \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.

```

437 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
438 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
439 \long\def\xintapplyunbraced #1#2%
440 {%
441   \expandafter\XINT_applyunbr\expandafter {\romannumeral-`0#2}%
442   {#1}%
443 }%
444 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
445 \long\def\xintapplyunbracednoexpand #1#2%
446   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
447 \long\def\XINT_applyunbr_loop_a #1#2#3%
448 {%
449   \xint_bye #3\XINT_applyunbr_end\xint_bye
450   \expandafter\XINT_applyunbr_loop_b
451   \expandafter {\romannumeral-`0#2{#3}}{#1}{#2}%
452 }%
453 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2}#1}%
454 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
455   \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 generate thousands of terms then.

```

456 \def\xintSeq {\romannumeral0\xintseq }%
457 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
458 \def\XINT_seq_chkopt #1%
459 {%
460   \ifx [#1\expandafter\XINT_seq_opt
461     \else\expandafter\XINT_seq_noopt
462   \fi #1%
463 }%

```

2 Package *xinttools* implementation

```
464 \def\XINT_seq_noopt #1\xint_bye #2%
465 {%
466   \expandafter\XINT_seq\expandafter
467     {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
468 }%
469 \def\XINT_seq #1#2%
470 {%
471   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
472     \expandafter\xint_firstoftwo_thenstop
473   \or
474     \expandafter\XINT_seq_p
475   \else
476     \expandafter\XINT_seq_n
477   \fi
478   {#2}{#1}%
479 }%
480 \def\XINT_seq_p #1#2%
481 {%
482   \ifnum #1>#2
483     \expandafter\expandafter\expandafter\XINT_seq_p
484   \else
485     \expandafter\XINT_seq_e
486   \fi
487   \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
488 }%
489 \def\XINT_seq_n #1#2%
490 {%
491   \ifnum #1<#2
492     \expandafter\expandafter\expandafter\XINT_seq_n
493   \else
494     \expandafter\XINT_seq_e
495   \fi
496   \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
497 }%
498 \def\XINT_seq_e #1#2#3{ }%
499 \def\XINT_seq_opt [\xint_bye #1]#2#3%
500 {%
501   \expandafter\XINT_seqo\expandafter
502     {\the\numexpr #2\expandafter}\expandafter
503     {\the\numexpr #3\expandafter}\expandafter
504     {\the\numexpr #1}%
505 }%
506 \def\XINT_seqo #1#2%
507 {%
508   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
509     \expandafter\XINT_seqo_a
510   \or
511     \expandafter\XINT_seqo_pa
512   \else
513     \expandafter\XINT_seqo_na
514   \fi
515   {#1}{#2}%

```

2 Package *xinttools* implementation

```

516 }%
517 \def\XINT_seqo_a #1#2#3{ {#1}}%
518 \def\XINT_seqo_o #1#2#3#4{ #4}%
519 \def\XINT_seqo_pa #1#2#3%
520 {%
521     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
522         \expandafter\XINT_seqo_o
523     \or
524         \expandafter\XINT_seqo_pb
525     \else
526         \xint_afterfi{\expandafter\space\xint_gobble_iv}%
527     \fi
528     {#1}{#2}{#3}{{#1}}%
529 }%
530 \def\XINT_seqo_pb #1#2#3%
531 {%
532     \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
533 }%
534 \def\XINT_seqo_pc #1#2%
535 {%
536     \ifnum #1>#2
537         \expandafter\XINT_seqo_o
538     \else
539         \expandafter\XINT_seqo_pd
540     \fi
541     {#1}{#2}%
542 }%
543 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
544 \def\XINT_seqo_na #1#2#3%
545 {%
546     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
547         \expandafter\XINT_seqo_o
548     \or
549         \xint_afterfi{\expandafter\space\xint_gobble_iv}%
550     \else
551         \expandafter\XINT_seqo_nb
552     \fi
553     {#1}{#2}{#3}{{#1}}%
554 }%
555 \def\XINT_seqo_nb #1#2#3%
556 {%
557     \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
558 }%
559 \def\XINT_seqo_nc #1#2%
560 {%
561     \ifnum #1<#2
562         \expandafter\XINT_seqo_o
563     \else
564         \expandafter\XINT_seqo_nd
565     \fi
566     {#1}{#2}%
567 }%

```

```
568 \def\XINT_sequo_nd #1#2#3#4{\XINT_sequo_nb {#1}{#2}{#3}{#4{#1}}}%
```

2.17 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

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

```
569 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
570 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
571      #1\xintloop_again\fi\xint_gobble_i {#1}}%
572 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%
573 \long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
574 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
575      #2\xintloop_again\fi\xint_gobble_i {#2}}%
```

2.18 \xintiloop, \xintiloopindex, \xintouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintilopskiptonext, \xintilopskipandredo

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

```
576 \def\xintiloop [#1+#2]{%
577   \expandafter\xintiloop_a\the\numexpr #1\expandafter.\the\numexpr #2.%
578 \long\def\xintiloop_a #1.#2.#3#4\repeat{%
579   #3#4\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
580 \def\xintiloop_again\fi\xint_gobble_iii #1#2{%
581   \fi\expandafter\xintiloop_again_b\the\numexpr#1+#2.#2.%
582 \long\def\xintiloop_again_b #1.#2.#3{%
583   #3\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
584 \long\def\xintbreakiloop #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
585 \long\def\xintbreakiloopanddo
586   #1.#2\xintiloop_again\fi\xint_gobble_iii #3#4#5{#1}%
587 \long\def\xintiloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
588   {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
589 \long\def\xintouteriloopindex #1\xintiloop_again
590   #2\xintiloop_again\fi\xint_gobble_iii #3%
591   {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
592 \long\def\xintilopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
593   \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.%
594 \long\def\xintilopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
595   #4\xintiloop_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.

```
596 \def\XINT_xflet #1%
597 {%
598   \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
599 }%
600 \def\XINT_xflet_zapsp
601 {%
602   \expandafter\futurelet\expandafter\XINT_token
```

```

603 \expandafter\XINT_xflet_sp?\romannumeral-\`0%
604 }%
605 \def\XINT_xflet_sp?
606 {%
607 \ifx\XINT_token\XINT_sptoken
608 \expandafter\XINT_xflet_zapsp
609 \else\expandafter\XINT_xflet_zapspB
610 \fi
611 }%
612 \def\XINT_xflet_zapspB
613 {%
614 \expandafter\futurelet\expandafter\XINT_tokenB
615 \expandafter\XINT_xflet_spB?\romannumeral-\`0%
616 }%
617 \def\XINT_xflet_spB?
618 {%
619 \ifx\XINT_tokenB\XINT_sptoken
620 \expandafter\XINT_xflet_zapspB
621 \else\expandafter\XINT_xflet_eq?
622 \fi
623 }%
624 \def\XINT_xflet_eq?
625 {%
626 \ifx\XINT_token\XINT_tokenB
627 \expandafter\XINT_xflet_macro
628 \else\expandafter\XINT_xflet_zapsp
629 \fi
630 }%

```

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.

```

631 \catcode\Z 3
632 \long\def\xintApplyInline #1#2%
633 {%
634 \long\expandafter\def\expandafter\XINT_inline_macro
635 \expandafter ##\expandafter 1\expandafter {#1{##1}}%
636 \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
637 }%
638 \def\XINT_inline_b
639 {%
640 \ifx\XINT_token Z\expandafter\xint_gobble_i
641 \else\expandafter\XINT_inline_d\fi
642 }%
643 \long\def\XINT_inline_d #1%
644 {%
645 \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
646 }%

```

```

647 \def\XINT_inline_e
648 {%
649   \ifx\XINT_token Z\expandafter\XINT_inline_w
650   \else\expandafter\XINT_inline_f\fi
651 }%
652 \def\XINT_inline_f
653 {%
654   \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
655 }%
656 \long\def\XINT_inline_g #1%
657 {%
658   \expandafter\XINT_inline_macro\XINT_item
659   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
660 }%
661 \def\XINT_inline_w #1%
662 {%
663   \expandafter\XINT_inline_macro\XINT_item
664 }%

```

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.

```

665 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#####2}}\fi}%
666 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#####2}}\fi}%
667 \def\XINT_tmpc #1%
668 {%
669   \expandafter\edef \csname XINT_for_left#1\endcsname
670     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
671   \expandafter\edef \csname XINT_for_right#1\endcsname
672     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
673 }%
674 \xintApplyInline \XINT_tmpc {123456789}%
675 \long\def\xintBreakFor #1Z{%
676 \long\def\xintBreakForAndDo #1#2Z{#1}%
677 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
678   \futurelet\XINT_token\XINT_for_ifstar}%
679 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
680   \else\expandafter\XINT_for \fi}%

```


2 Package *xinttools* implementation

```
681 \catcode`U 3 % with numexpr
682 \catcode`V 3 % with xintfrac.sty (xint.sty not enough)
683 \catcode`D 3 % with dimexpr
684 \def\XINT_flet_zapsp
685 {%
686   \futurelet\XINT_token\XINT_flet_sp?
687 }%
688 \def\XINT_flet_sp?
689 {%
690   \ifx\XINT_token\XINT_sptoken
691     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
692   \else\expandafter\XINT_flet_macro
693   \fi
694 }%
695 \long\def\XINT_for #1#2in#3#4#5%
696 {%
697   \expandafter\XINT_toks\expandafter
698   {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
699   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
700   \expandafter\XINT_flet_zapsp #3Z%
701 }%
702 \def\XINT_for_forever? #1Z%
703 {%
704   \ifx\XINT_token U\XINT_to_forever\fi
705   \ifx\XINT_token V\XINT_to_forever\fi
706   \ifx\XINT_token D\XINT_to_forever\fi
707   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
708 }%
709 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
710 \long\def\XINT_forx *#1#2in#3#4#5%
711 {%
712   \expandafter\XINT_toks\expandafter
713   {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
714   \XINT_xflet\XINT_forx_forever? #3Z%
715 }%
716 \def\XINT_forx_forever?
717 {%
718   \ifx\XINT_token U\XINT_to_forxever\fi
719   \ifx\XINT_token V\XINT_to_forxever\fi
720   \ifx\XINT_token D\XINT_to_forxever\fi
721   \XINT_forx_empty?
722 }%
723 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
724 \catcode`U 11
725 \catcode`D 11
726 \catcode`V 11
727 \def\XINT_forx_empty?
728 {%
729   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
730   \the\XINT_toks
731 }%
732 \long\def\XINT_for_d #1#2#3%
```

```

733 {%
734 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
735 \XINT_toks {{#3}}%
736 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
737 \the\XINT_toks \csname XINT_for_right#1\endcsname}%
738 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_for_d #1{#2}}%
739 \futurelet\XINT_token\XINT_for_last?
740}%
741 \long\def\XINT_forx_d #1#2#3%
742 {%
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_forx_d #1{#2}}%
748 \XINT_xflet\XINT_for_last?
749}%
750 \def\XINT_for_last?
751 {%
752 \let\xintifForLast\xint_secondoftwo
753 \ifx\XINT_token Z\let\xintifForLast\xint_firstoftwo
754 \xint_afterfi{\xintBreakForAndDo{\XINT_x\xint_gobble_i Z}}\fi
755 \the\XINT_toks
756}%

```

2.22 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

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

```

757 \catcode`U 3
758 \catcode`D 3
759 \catcode`V 3
760 \let\xintegers U%
761 \let\xintintegers U%
762 \let\xintdimensions D%
763 \let\xinrationals V%
764 \def\XINT_forever #1%
765 {%
766 \expandafter\XINT_forever_a
767 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
768 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
769 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
770}%
771 \catcode`U 11
772 \catcode`D 11
773 \catcode`V 11
774 \def\XINT_?expr_Ua #1#2%
775 {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax

```

2 Package *xinttools* implementation

```

776             \expandafter\relax\expandafter}%
777     \expandafter{\the\numexpr #2}}%
778 \def\XINT_?expr_Da #1#2%
779     {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
780         \expandafter s\expandafter p\expandafter\relax\expandafter}%
781     \expandafter{\number\dimexpr #2}}%
782 \catcode`Z 11
783 \def\XINT_?expr_Va #1#2%
784 {%
785     \expandafter\XINT_?expr_Vb\expandafter
786         {\romannumeral-`0\xintraawithzeros{\xintZapSpacesB{#2}}}%
787         {\romannumeral-`0\xintraawithzeros{\xintZapSpacesB{#1}}}%
788 }%
789 \catcode`Z 3
790 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1.%}
791 \def\XINT_?expr_Vc #1/#2.#3/#4.%
792 {%
793     \xintifEq {#2}{#4}%
794         {\XINT_?expr_Vf {#3}{#1}{#2}}%
795         {\expandafter\XINT_?expr_Vd\expandafter
796             {\romannumeral0\xintiimul {#2}{#4}}%
797             {\romannumeral0\xintiimul {#1}{#4}}%
798             {\romannumeral0\xintiimul {#2}{#3}}%
799         }%
800 }%
801 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
802 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
803 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
804 \def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
805 \def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
806 \def\XINT_?expr_Vi {{1/1}{0111}}%
807 \def\XINT_?expr_U #1#2%
808     {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
809 \def\XINT_?expr_D #1#2%
810     {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
811 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
812 \def\XINT_?expr_Vx #1#2%
813 {%
814     \expandafter\XINT_?expr_Vy\expandafter
815         {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
816 }%
817 \def\XINT_?expr_Vy #1#2#3#4%
818 {%
819     \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
820 }%
821 \def\XINT_forever_a #1#2#3#4%
822 {%
823     \ifx #4[\expandafter\XINT_forever_opt_a
824         \else\expandafter\XINT_forever_b
825     \fi #1#2#3#4%
826 }%
827 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%

```

```

828 \long\def\XINT_forever_c #1#2#3#4#5%
829   {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
830 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
831 {%
832   \expandafter\expandafter\expandafter
833   \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
834   \romannumeral-`0#1{#4}{#5}#3%
835 }%
836 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
837 \long\def\XINT_forever_d #1#2#3#4#5%
838 {%
839   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
840   \XINT_toks {{#2}}%
841   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
842     \the\XINT_toks \csname XINT_for_right#1\endcsname}%
843   \XINT_x
844   \let\xintifForFirst\xint_seconddoftwo
845   \expandafter\XINT_forever_d\expandafter #1\romannumeral-`0#4{#2}{#3}#4{#5}%
846 }%

```

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.

```

847 \catcode`j 3
848 \long\def\xintForpair #1#2#3in#4#5#6%
849 {%
850   \let\xintifForFirst\xint_firstoftwo
851   \XINT_toks {\XINT_forpair_d #2{#6}}%
852   \expandafter\the\expandafter\XINT_toks #4jZ%
853 }%
854 \long\def\XINT_forpair_d #1#2#3(#4)#5%
855 {%
856   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
857   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
858   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
859     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
860   \let\xintifForLast\xint_seconddoftwo
861   \ifx #5j\expandafter\xint_firstoftwo
862     \else\expandafter\xint_seconddoftwo
863   \fi
864   {\let\xintifForLast\xint_firstoftwo
865     \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
866   \XINT_x
867   \let\xintifForFirst\xint_seconddoftwo\XINT_forpair_d #1{#2}%
868 }%
869 \long\def\xintForthree #1#2#3in#4#5#6%
870 {%
871   \let\xintifForFirst\xint_firstoftwo

```

```

872 \XINT_toks {\XINT_forthree_d #2{#6}}%
873 \expandafter\the\expandafter\XINT_toks #4jZ%
874 }%
875 \long\def\XINT_forthree_d #1#2#3(#4)#5%
876 {%
877 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
878 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
879 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
880 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
881 \let\xintifForLast\xint_secondoftwo
882 \ifx #5j\expandafter\xint_firstoftwo
883 \else\expandafter\xint_secondoftwo
884 \fi
885 {\let\xintifForLast\xint_firstoftwo
886 \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
887 \XINT_x
888 \let\xintifForFirst\xint_secondoftwo\XINT_forthree_d #1{#2}%
889 }%
890 \long\def\xintForfour #1#2#3in#4#5#6%
891 {%
892 \let\xintifForFirst\xint_firstoftwo
893 \XINT_toks {\XINT_forfour_d #2{#6}}%
894 \expandafter\the\expandafter\XINT_toks #4jZ%
895 }%
896 \long\def\XINT_forfour_d #1#2#3(#4)#5%
897 {%
898 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
899 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
900 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
901 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
902 \let\xintifForLast\xint_secondoftwo
903 \ifx #5j\expandafter\xint_firstoftwo
904 \else\expandafter\xint_secondoftwo
905 \fi
906 {\let\xintifForLast\xint_firstoftwo
907 \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
908 \XINT_x
909 \let\xintifForFirst\xint_secondoftwo\XINT_forfour_d #1{#2}%
910 }%
911 \catcode`Z 11
912 \catcode`j 11

```

2.24 \xintAssign, \xintAssignArray, \xintDigitsOf

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

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

```

913 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
914 \def\XINT_assign_fork
915 {%

```

2 Package *xinttools* implementation

```
916 \let\XINT_assign_def\def
917 \ifx\XINT_token[\expandafter\XINT_assign_opt
918 \else\expandafter\XINT_assign_a
919 \fi
920 }%
921 \def\XINT_assign_opt [#1]%
922 {%
923 \ifcsname #1def\endcsname
924 \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
925 \else
926 \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
927 \fi
928 \XINT_assign_a
929 }%
930 \long\def\XINT_assign_a #1\to
931 {%
932 \def\XINT_flet_macro{\XINT_assign_b}%
933 \expandafter\XINT_flet_zapsp\romannumeral-`0#1\xint_relax\to
934 }%
935 \long\def\XINT_assign_b
936 {%
937 \ifx\XINT_token\bgroup
938 \expandafter\XINT_assign_c
939 \else\expandafter\XINT_assign_f
940 \fi
941 }%
942 \long\def\XINT_assign_f #1\xint_relax\to #2%
943 {%
944 \XINT_assign_def #2{#1}%
945 }%
946 \long\def\XINT_assign_c #1%
947 {%
948 \def\xint_temp {#1}%
949 \ifx\xint_temp\xint_brelax
950 \expandafter\XINT_assign_e
951 \else
952 \expandafter\XINT_assign_d
953 \fi
954 }%
955 \long\def\XINT_assign_d #1\to #2%
956 {%
957 \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
958 \XINT_assign_c #1\to
959 }%
960 \def\XINT_assign_e #1\to {}%
961 \def\xintRelaxArray #1%
962 {%
963 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
964 \escapechar -1
965 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
966 \XINT_restoreescapechar
967 \xintilooop [\csname\xint_arrayname 0\endcsname+-1]
```

2 Package *xinttools* implementation

```

968 \global
969 \expandafter\let\csname\xint_arrayname\xintloopindex\endcsname\relax
970 \ifnum \xintloopindex > \xint_c_
971 \repeat
972 \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
973 \global\let #1\relax
974 }%
975 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
976 \XINT_flet_zapsp }%
977 \def\XINT_assignarray_fork
978 {%
979 \let\XINT_assignarray_def\def
980 \ifx\XINT_token[\expandafter\XINT_assignarray_opt
981 \else\expandafter\XINT_assignarray
982 \fi
983 }%
984 \def\XINT_assignarray_opt [#1]%
985 {%
986 \ifcsname #1def\endcsname
987 \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
988 \else
989 \expandafter\let\expandafter\XINT_assignarray_def
990 \csname xint#1def\endcsname
991 \fi
992 \XINT_assignarray
993 }%
994 \long\def\XINT_assignarray #1\to #2%
995 {%
996 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
997 \escapechar -1
998 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
999 \XINT_restoreescapechar
1000 \def\xint_itemcount {0}%
1001 \expandafter\XINT_assignarray_loop \romannumeral-`0#1\xint_relax
1002 \csname\xint_arrayname 00\expandafter\endcsname
1003 \csname\xint_arrayname 0\expandafter\endcsname
1004 \expandafter {\xint_arrayname}#2%
1005 }%
1006 \long\def\XINT_assignarray_loop #1%
1007 {%
1008 \def\xint_temp {#1}%
1009 \ifx\xint_brelax\xint_temp
1010 \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1011 \expandafter{\the\numexpr\xint_itemcount}%
1012 \expandafter\expandafter\expandafter\XINT_assignarray_end
1013 \else
1014 \expandafter\def\expandafter\xint_itemcount\expandafter
1015 {\the\numexpr\xint_itemcount+\xint_c_i}%
1016 \expandafter\XINT_assignarray_def
1017 \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1018 \expandafter{\xint_temp }%
1019 \expandafter\XINT_assignarray_loop

```

3 Package `xintcore` implementation

```

1020 \fi
1021 }%
1022 \def\XINT_assignarray_end #1#2#3#4%
1023 {%
1024 \def #4##1%
1025 {%
1026 \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1027 }%
1028 \def #1##1%
1029 {%
1030 \ifnum ##1<\xint_c_
1031 \xint_afterfi {\xintError:ArrayIndexIsNegative\space }%
1032 \else
1033 \xint_afterfi {%
1034 \ifnum ##1>#2
1035 \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
1036 \else\xint_afterfi
1037 {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1038 \fi}%
1039 \fi
1040 }%
1041 }%
1042 \let\xintDigitsOf\xintAssignArray
1043 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1044 \XINT_restorecatcodes_endinput%

```

3 Package `xintcore` implementation

.1	Catcodes, ε -TeX and reload detection	33	.18	Variants for addition sub-routines	44
.2	Package identification	34	.18.1	Addition vI: <code>\XINT_add_A</code>	45
.3	More token management, constants	34	.18.2	Addition vII: <code>\XINT_addr_A</code>	46
.4	<code>\XINT_RQ</code>	34	.18.3	Addition vIII: <code>\XINT_addm_A</code>	47
.5	<code>\XINT_OQ</code>	35	.18.4	Addition vIV: <code>\XINT_addp_A</code>	49
.6	<code>\XINT_SQ</code>	35	.19	<code>\xintiAdd</code> , <code>\xintiiAdd</code>	52
.7	<code>\XINT_cuz</code>	36	.20	<code>\xintiSub</code> , <code>\xintiiSub</code>	53
.8	<code>\xintNum</code>	37	.21	<code>\xintiMul</code> , <code>\xintiiMul</code>	58
.9	<code>\xintSgn</code> , <code>\xintiiSgn</code> , <code>\XINT_Sgn</code> , <code>\XINT_cntSgn</code>	38	.21.1	"Small" multiplication: <code>\XINT_mul_Mr</code>	61
.10	<code>\xintiOpp</code>	39	.21.2	"Small" multiplication variant: <code>\XINT_mul_M</code>	62
.11	<code>\xintiAbs</code> , <code>\xintiiAbs</code>	39	.21.3	Main routine: <code>\XINT_mul_enter</code>	63
.12	<code>\xintFDg</code> , <code>\xintiiFDg</code>	40	.21.4	Variant: <code>\XINT_mulr_enter</code>	65
.13	<code>\xintLDg</code> , <code>\xintiiLDg</code>	40	.22	<code>\xintiSqr</code> , <code>\xintiiSqr</code>	67
.14	<code>\xintDouble</code>	41	.23	<code>\xintiPow</code> , <code>\xintiiPow</code>	67
.15	<code>\xintHalf</code>	42	.24	<code>\xintiDivision</code> , <code>\xintiQuo</code> , <code>\xintiRem</code> , <code>\xintiiDivision</code> , <code>\xintiiQuo</code> , <code>\xintiiRem</code>	70
.16	<code>\xintDec</code>	43	.25	<code>\xintiDivRound</code> , <code>\xintiiDivRound</code>	83
.17	<code>\xintInc</code>	44	.26	<code>\xintiDivTrunc</code> , <code>\xintiiDivTrunc</code>	84
			.27	<code>\xintiMod</code> , <code>\xintiiMod</code>	85

Got split off from `xint` with release 1.1 (macros `\XINT_SQ`, `\xintLDg`, `\xintHalf` which are dependencies of `\XINT_div_prepare` were forgotten and they were added to the package only later with 1.1b). Release 1.1 also added the new macro `\xintiiDivRound`. The package does not load `xinttools`.

Since release `xint` 1.09a the 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. It simplifies the discussion in the documentation too.

3.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2015/09/12 v1.1c 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_UDzerosfork #100#2#3\krof {#2}%
60 \def\xint_UDonezerofork #110#2#3\krof {#2}%
61 \def\xint_UDsignsfork #1--#2#3\krof {#2}%
62 \chardef\xint_c_ix 9
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

```

3 Package *xintcore* implementation

```
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 #1\Z #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#900000000}%
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#9000000}%
123 \def\XINT_OQ_end_iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#900000}%
124 \def\XINT_OQ_end_iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#90000}%
125 \def\XINT_OQ_end_ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#9000}%
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_SQ

```
127 \def\XINT_SQ #1#2#3#4#5#6#7#8%
128 {%
129   \xint_gob_til_R #8\XINT_SQ_end_a\R\XINT_SQ {#8#7#6#5#4#3#2#1}%
130 }%
131 \def\XINT_SQ_end_a\R\XINT_SQ #1#2\Z
132 {%
```

```

133 \XINT_SQ_end_b #1\Z
134 }%
135 \def\XINT_SQ_end_b #1#2#3#4#5#6#7%
136 {%
137 \xint_gob_til_R
138 #7\XINT_SQ_end_vii
139 #6\XINT_SQ_end_vi
140 #5\XINT_SQ_end_v
141 #4\XINT_SQ_end_iv
142 #3\XINT_SQ_end_iii
143 #2\XINT_SQ_end_ii
144 \R\XINT_SQ_end_i
145 \Z #2#3#4#5#6#7%
146 }%
147 \def\XINT_SQ_end_vii #1\Z #2#3#4#5#6#7#8\Z { #8}%
148 \def\XINT_SQ_end_vi #1\Z #2#3#4#5#6#7#8\Z { #7#8000000}%
149 \def\XINT_SQ_end_v #1\Z #2#3#4#5#6#7#8\Z { #6#7#800000}%
150 \def\XINT_SQ_end_iv #1\Z #2#3#4#5#6#7#8\Z { #5#6#7#80000}%
151 \def\XINT_SQ_end_iii #1\Z #2#3#4#5#6#7#8\Z { #4#5#6#7#8000}%
152 \def\XINT_SQ_end_ii #1\Z #2#3#4#5#6#7#8\Z { #3#4#5#6#7#800}%
153 \def\XINT_SQ_end_i \Z #1#2#3#4#5#6#7\Z { #1#2#3#4#5#6#70}%

```

3.7 \XINT_cuz

```

154 \edef\xint_cleanupzeros_andstop #1#2#3#4%
155 {%
156 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
157 }%
158 \def\xint_cleanupzeros_nostop #1#2#3#4%
159 {%
160 \the\numexpr #1#2#3#4\relax
161 }%
162 \def\XINT_rev_andcuz #1%
163 {%
164 \expandafter\xint_cleanupzeros_andstop
165 \romannumeral0\XINT_rord_main {}#1%
166 \xint_relax
167 \xint_bye\xint_bye\xint_bye\xint_bye
168 \xint_bye\xint_bye\xint_bye\xint_bye
169 \xint_relax
170 }%

```

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

```

171 \def\XINT_cuz #1%
172 {%
173 \XINT_cuz_loop #1\W\W\W\W\W\W\W\Z%
174 }%
175 \def\XINT_cuz_loop #1#2#3#4#5#6#7#8%
176 {%
177 \xint_gob_til_W #8\xint_cuz_end_a\W
178 \xint_gob_til_Z #8\xint_cuz_end_A\Z

```

3 Package *xintcore* implementation

```

179 \XINT_cuz_check_a {#1#2#3#4#5#6#7#8}%
180 }%
181 \def\xint_cuz_end_a #1\XINT_cuz_check_a #2%
182 {%
183 \xint_cuz_end_b #2%
184 }%
185 \edef\xint_cuz_end_b #1#2#3#4#5\Z
186 {%
187 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
188 }%
189 \def\xint_cuz_end_A \Z\XINT_cuz_check_a #1{ 0}%
190 \def\XINT_cuz_check_a #1%
191 {%
192 \expandafter\XINT_cuz_check_b\the\numexpr #1\relax
193 }%
194 \def\XINT_cuz_check_b #1%
195 {%
196 \xint_gob_til_zero #1\xint_cuz_backtoloop 0\XINT_cuz_stop #1%
197 }%
198 \def\XINT_cuz_stop #1\W #2\Z{ #1}%
199 \def\xint_cuz_backtoloop 0\XINT_cuz_stop 0{\XINT_cuz_loop }%

```

3.8 *\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`.

```

200 \def\xintiNum {\romannumeral0\xintinum }%
201 \def\xintinum #1%
202 {%
203 \expandafter\XINT_num_loop
204 \romannumeral-`0#1\xint_relax\xint_relax\xint_relax\xint_relax
205 \xint_relax\xint_relax\xint_relax\xint_relax\Z
206 }%
207 \let\xintNum\xintiNum \let\xintnum\xintinum
208 \def\XINT_num #1%
209 {%
210 \XINT_num_loop #1\xint_relax\xint_relax\xint_relax\xint_relax
211 \xint_relax\xint_relax\xint_relax\xint_relax\Z
212 }%
213 \def\XINT_num_loop #1#2#3#4#5#6#7#8%
214 {%
215 \xint_gob_til_xint_relax #8\XINT_num_end\xint_relax
216 \XINT_num_NumEight #1#2#3#4#5#6#7#8%
217 }%
218 \edef\XINT_num_end\xint_relax\XINT_num_NumEight #1\xint_relax #2\Z
219 {%

```

3 Package *xintcore* implementation

```
220 \noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_\relax
221 }%
222 \def\XINT_num_NumEight #1#2#3#4#5#6#7#8%
223 {%
224 \ifnum \numexpr #1#2#3#4#5#6#7#8+\xint_c_= \xint_c_
225 \xint_afterfi {\expandafter\XINT_num_keepsign_a
226 \the\numexpr #1#2#3#4#5#6#7#81\relax}%
227 \else
228 \xint_afterfi {\expandafter\XINT_num_finish
229 \the\numexpr #1#2#3#4#5#6#7#8\relax}%
230 \fi
231 }%
232 \def\XINT_num_keepsign_a #1%
233 {%
234 \xint_gob_til_one#1\XINT_num_gobackto loop 1\XINT_num_keepsign_b
235 }%
236 \def\XINT_num_gobackto loop 1\XINT_num_keepsign_b {\XINT_num_loop }%
237 \def\XINT_num_keepsign_b #1{\XINT_num_loop -}%
238 \def\XINT_num_finish #1\xint_relax #2\Z { #1}%
```

3.9 `\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.

```
239 \def\xintiiSgn {\romannumeral0\xintiisgn }%
240 \def\xintiisgn #1%
241 {%
242 \expandafter\XINT_sgn \romannumeral-`0#1\Z%
243 }%
244 \def\xintSgn {\romannumeral0\xintsgn }%
245 \def\xintsgn #1%
246 {%
247 \expandafter\XINT_sgn \romannumeral0\xintnum{#1}\Z%
248 }%
249 \def\XINT_sgn #1#2\Z
250 {%
251 \xint_UDzerominusfork
252 #1-{ 0}%
253 0#1{ -1}%
254 0-{ 1}%
255 \krof
256 }%
257 \def\XINT_Sgn #1#2\Z
258 {%
259 \xint_UDzerominusfork
260 #1-{0}%
261 0#1{-1}%
262 0-{1}%
263 \krof
264 }%
```

```

265 \def\XINT_cntSgn #1#2\Z
266 {%
267   \xint_UDzerominusfork
268   #1-\xint_c_
269   0#1\m@ne % I will not allocate a count only for -1?
270   0-\xint_c_i
271   \krof
272 }%

```

3.10 \xintiOpp

\xintnum added in 1.09a

```

273 \def\xintiiOpp {\romannumeral0\xintiiopp }%
274 \def\xintiiopp #1%
275 {%
276   \expandafter\XINT_opp \romannumeral-\`0#1%
277 }%
278 \def\xintiOpp {\romannumeral0\xintiopp }%
279 \def\xintiopp #1%
280 {%
281   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
282 }%
283 \let\xintOpp\xintiOpp \let\xintopp\xintiopp
284 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
285 \def\XINT_opp #1%
286 {%
287   \xint_UDzerominusfork
288   #1-{ 0}%      zero
289   0#1{ }%      negative
290   0-{ -#1}%    positive
291   \krof
292 }%

```

3.11 \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.

```

293 \def\xintiiAbs {\romannumeral0\xintiiabs }%
294 \def\xintiiabs #1%
295 {%
296   \expandafter\XINT_abs \romannumeral-\`0#1%
297 }%
298 \def\xintiAbs {\romannumeral0\xintiabs }%
299 \def\xintiabs #1%
300 {%
301   \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
302 }%
303 \let\xintAbs\xintiAbs \let\xintabs\xintiabs
304 \def\XINT_Abs #1{\romannumeral0\XINT_abs #1}%

```

```

305 \def\XINT_abs #1%
306 {%
307   \xint_UDsignfork
308   #1{ }%
309   -{ #1}%
310   \krof
311 }%

```

3.12 \xintFDg, \xintiFDg

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.

```

312 \def\xintiFDg {\romannumeral0\xintiifdg }%
313 \def\xintiifdg #1%
314 {%
315   \expandafter\XINT_fdg \romannumeral-`0#1\W\Z
316 }%
317 \def\xintFDg {\romannumeral0\xintfdg }%
318 \def\xintfdg #1%
319 {%
320   \expandafter\XINT_fdg \romannumeral0\xintnum{#1}\W\Z
321 }%
322 \def\XINT_FDg #1{\romannumeral0\XINT_fdg #1\W\Z }%
323 \def\XINT_fdg #1#2#3\Z
324 {%
325   \xint_UDzerominusfork
326   #1-{ 0}%   zero
327   0#1{ #2}%  negative
328   0-{ #1}%   positive
329   \krof
330 }%

```

3.13 \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 \xintiiOdd which is used once (currently) elsewhere .

bug fix (1.1b): \xintiLDg is needed by the division macros next, thus it needs to be in the xintcore.sty

```

331 \def\xintiLDg {\romannumeral0\xintiildg }%
332 \def\xintiildg #1%
333 {%
334   \expandafter\XINT_ldg\expandafter {\romannumeral-`0#1}%
335 }%
336 \def\xintLDg {\romannumeral0\xintldg }%
337 \def\xintldg #1%
338 {%
339   \expandafter\XINT_ldg\expandafter {\romannumeral0\xintnum{#1}}%
340 }%
341 \def\XINT_LDg #1{\romannumeral0\XINT_ldg {#1}}%
342 \def\XINT_ldg #1%

```



```

343 {%
344   \expandafter\XINT_ldg_\romannumeral0\xintreverseorder {#1}\Z
345 }%
346 \def\XINT_ldg_ #1#2\Z{ #1}%

```

3.14 \xintDouble

v1.08

```

347 \def\xintDouble {\romannumeral0\xintdouble }%
348 \def\xintdouble #1%
349 {%
350   \expandafter\XINT_dbl_\romannumeral-\`0#1%
351   \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W
352 }%
353 \def\XINT_dbl #1%
354 {%
355   \xint_UDzerominusfork
356   #1-\XINT_dbl_zero
357   0#1\XINT_dbl_neg
358   0-{\XINT_dbl_pos #1}%
359   \krof
360 }%
361 \def\XINT_dbl_zero #1\Z \W\W\W\W\W\W\W { 0}%
362 \def\XINT_dbl_neg
363   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dbl_pos }%
364 \def\XINT_dbl_pos
365 {%
366   \expandafter\XINT_dbl_a \expandafter{\expandafter}\expandafter 0%
367   \romannumeral0\XINT_SQ }%
368 }%
369 \def\XINT_dbl_a #1#2#3#4#5#6#7#8#9%
370 {%
371   \xint_gob_til_W #9\XINT_dbl_end_a\W
372   \expandafter\XINT_dbl_b
373   \the\numexpr \xint_c_x^viii+#2+\xint_c_ii*#9#8#7#6#5#4#3\relax {#1}%
374 }%
375 \def\XINT_dbl_b 1#1#2#3#4#5#6#7#8#9%
376 {%
377   \XINT_dbl_a {#2#3#4#5#6#7#8#9}{#1}%
378 }%
379 \def\XINT_dbl_end_a #1+#2+#3\relax #4%
380 {%
381   \expandafter\XINT_dbl_end_b #2#4%
382 }%
383 \edef\XINT_dbl_end_b #1#2#3#4#5#6#7#8%
384 {%
385   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
386 }%

```

3.15 \xintHalf

v1.08. Release 1.1 left it in `xint.sty`, but it is needed by the division routines included in `xintcore.sty`. Thus moved here for bugfix release 1.1c. Also `\XINT_SQ` which it uses. Moved here `\xintDouble` as well by sympathy.

```

387 \def\xintHalf {\romannumeral0\xinthalf }%
388 \def\xinthalf #1%
389 {%
390     \expandafter\XINT_half\romannumeral-`0#1%
391     \R\R\R\R\R\R\R\Z \W\W\W\W\W\W
392 }%
393 \def\XINT_half #1%
394 {%
395     \xint_UDzerominusfork
396     #1-\XINT_half_zero
397     0#1\XINT_half_neg
398     0-{\XINT_half_pos #1}%
399     \krof
400 }%
401 \def\XINT_half_zero #1\Z \W\W\W\W\W\W { 0}%
402 \def\XINT_half_neg {\expandafter\XINT_opp\romannumeral0\XINT_half_pos }%
403 \def\XINT_half_pos {\expandafter\XINT_half_a\romannumeral0\XINT_SQ {}}%
404 \def\XINT_half_a #1#2#3#4#5#6#7#8%
405 {%
406     \xint_gob_til_W #8\XINT_half_dont\W
407     \expandafter\XINT_half_b
408     \the\numexpr \xint_c_x^viii+\xint_c_v*#7#6#5#4#3#2#1\relax #8%
409 }%
410 \edef\XINT_half_dont\W\expandafter\XINT_half_b
411     \the\numexpr \xint_c_x^viii+\xint_c_v*#1#2#3#4#5#6#7\relax \W\W\W\W\W\W
412 {%
413     \noexpand\expandafter\space
414     \noexpand\the\numexpr (#1#2#3#4#5#6#7+\xint_c_i)/\xint_c_ii-\xint_c_i \relax
415 }%
416 \def\XINT_half_b 1#1#2#3#4#5#6#7#8%
417 {%
418     \XINT_half_c {#2#3#4#5#6#7}{#1}%
419 }%
420 \def\XINT_half_c #1#2#3#4#5#6#7#8#9%
421 {%
422     \xint_gob_til_W #3\XINT_half_end_a #2\W
423     \expandafter\XINT_half_d
424     \the\numexpr \xint_c_x^viii+\xint_c_v*#9#8#7#6#5#4#3+#2\relax {#1}%
425 }%
426 \def\XINT_half_d 1#1#2#3#4#5#6#7#8#9%
427 {%
428     \XINT_half_c {#2#3#4#5#6#7#8#9}{#1}%
429 }%
430 \def\XINT_half_end_a #1\W #2\relax #3%
431 {%
432     \xint_gob_til_zero #1\XINT_half_end_b 0\space #1#3%
433 }%

```

```

434 \edef\XINT_half_end_b 0\space 0#1#2#3#4#5#6#7%
435 {%
436     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7\relax
437 }%

```

3.16 \xintDec

v1.08

```

438 \def\xintDec {\romannumeral0\xintdec }%
439 \def\xintdec #1%
440 {%
441     \expandafter\XINT_dec\romannumeral-`0#1%
442     \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
443 }%
444 \def\XINT_dec #1%
445 {%
446     \xint_UDzerominusfork
447     #1-\XINT_dec_zero
448     0#1\XINT_dec_neg
449     0-{\XINT_dec_pos #1}%
450     \krof
451 }%
452 \def\XINT_dec_zero #1\W\W\W\W\W\W\W\W { -1}%
453 \def\XINT_dec_neg
454     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_inc_pos }%
455 \def\XINT_dec_pos
456 {%
457     \expandafter\XINT_dec_a \expandafter{\expandafter}%
458     \romannumeral0\XINT_OQ }%
459 }%
460 \def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
461 {%
462     \expandafter\XINT_dec_b
463     \the\numexpr 11#9#8#7#6#5#4#3#2-\xint_c_i\relax {#1}%
464 }%
465 \def\XINT_dec_b 1#1%
466 {%
467     \xint_gob_til_one #1\XINT_dec_A 1\XINT_dec_c
468 }%
469 \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}}%
470 \def\XINT_dec_A 1\XINT_dec_c #1#2#3#4#5#6#7#8#9%
471     {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
472 \def\XINT_dec_B #1#2\W\W\W\W\W\W\W\W
473 {%
474     \expandafter\XINT_dec_cleanup
475     \romannumeral0\XINT_rord_main {}#2%
476     \xint_relax
477     \xint_bye\xint_bye\xint_bye\xint_bye
478     \xint_bye\xint_bye\xint_bye\xint_bye
479     \xint_relax
480     #1%
481 }%

```

```

482 \edef\XINT_dec_cleanup #1#2#3#4#5#6#7#8%
483   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax }%

```

3.17 *\xintInc*

v1.08

```

484 \def\xintInc {\romannumeral0\xintinc }%
485 \def\xintinc #1%
486 {%
487   \expandafter\XINT_inc\romannumeral-`0#1%
488   \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
489 }%
490 \def\XINT_inc #1%
491 {%
492   \xint_UDzerominusfork
493   #1-\XINT_inc_zero
494   0#1\XINT_inc_neg
495   0-{\XINT_inc_pos #1}%
496   \krof
497 }%
498 \def\XINT_inc_zero #1\W\W\W\W\W\W\W\W { 1}%
499 \def\XINT_inc_neg {\expandafter\XINT_opp\romannumeral0\XINT_dec_pos }%
500 \def\XINT_inc_pos
501 {%
502   \expandafter\XINT_inc_a \expandafter{\expandafter}%
503   \romannumeral0\XINT_OQ }%
504 }%
505 \def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
506 {%
507   \xint_gob_til_W #9\XINT_inc_end\W
508   \expandafter\XINT_inc_b
509   \the\numexpr 10#9#8#7#6#5#4#3#2+\xint_c_i\relax {#1}%
510 }%
511 \def\XINT_inc_b 1#1%
512 {%
513   \xint_gob_til_zero #1\XINT_inc_A 0\XINT_inc_c
514 }%
515 \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}}%
516 \def\XINT_inc_A 0\XINT_inc_c #1#2#3#4#5#6#7#8#9%
517   {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
518 \def\XINT_inc_end\W #1\relax #2{ 1#2}%

```

3.18 Variants for addition sub-routines

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 and multiplication each have multiple implementations corresponding to slightly differing formats on input and on output.

3.18.1 Addition vl: `\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.

```

519 \def\XINT_add_A #1#2#3#4#5#6%
520 {%
521   \xint_gob_til_W #3\xint_add_az\W
522   \XINT_add_AB #1{#3#4#5#6}{#2}%
523 }%
524 \def\xint_add_az\W\XINT_add_AB #1#2%
525 {%
526   \XINT_add_AC_checkcarry #1%
527 }%

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.

528 \def\XINT_add_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
529 {%
530   \xint_gob_til_W #5\xint_add_bz\W
531   \XINT_add_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
532 }%
533 \def\XINT_add_ABE #1#2#3#4#5#6%
534 {%
535   \expandafter\XINT_add_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
536 }%
537 \def\XINT_add_ABEA #1#2#3.#4%
538 {%
539   \XINT_add_A #2{#3#4}%
540 }%
```

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

```

541 \def\xint_add_bz\W\XINT_add_ABE #1#2#3#4#5#6%
542 {%
543   \expandafter\XINT_add_CC\the\numexpr #1+10#5#4#3#2.%
544 }%
545 \def\XINT_add_CC #1#2#3.#4%
546 {%
547   \XINT_add_AC_checkcarry #2{#3#4}% on va examiner et \'eliminer #2
548 }%
```

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

```

549 \def\XINT_add_AC_checkcarry #1%
550 {%
551   \xint_gob_til_zero #1\xint_add_AC_nocarry 0\XINT_add_C
552 }%
553 \def\xint_add_AC_nocarry 0\XINT_add_C #1#2\W\X\Y\Z
554 {%
555   \expandafter
556   \xint_cleanupzeros_andstop
557   \romannumeral0%
558   \XINT_rord_main {}#2%
559   \xint_relax
560   \xint_bye\xint_bye\xint_bye\xint_bye
561   \xint_bye\xint_bye\xint_bye\xint_bye
562   \xint_relax
563   #1%
564 }%
565 \def\XINT_add_C #1#2#3#4#5%
566 {%
567   \xint_gob_til_W #2\xint_add_cz\W
568   \XINT_add_CD {#5#4#3#2}{#1}%
569 }%
570 \def\XINT_add_CD #1%
571 {%
572   \expandafter\XINT_add_CC\the\numexpr 1+10#1.%
573 }%
574 \def\xint_add_cz\W\XINT_add_CD #1#2{ 1#2}%

```

3.18.2 Addition vII: *\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*

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**

```

575 \def\XINT_addr_A #1#2#3#4#5#6%
576 {%
577   \xint_gob_til_W #3\xint_addr_az\W
578   \XINT_addr_B #1{#3#4#5#6}{#2}%
579 }%
580 \def\xint_addr_az\W\XINT_addr_B #1#2%
581 {%
582   \XINT_addr_AC_checkcarry #1%
583 }%
584 \def\XINT_addr_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
585 {%

```

3 Package *xintcore* implementation

```
586 \xint_gob_til_W #5\xint_addr_bz\W
587 \XINT_addr_E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
588 }%
589 \def\XINT_addr_E #1#2#3#4#5#6%
590 {%
591 \expandafter\XINT_addr_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
592 }%
593 \def\XINT_addr_ABEA #1#2#3#4#5#6#7%
594 {%
595 \XINT_addr_A #2{#7#6#5#4#3}%
596 }%
597 \def\xint_addr_bz\W\XINT_addr_E #1#2#3#4#5#6%
598 {%
599 \expandafter\XINT_addr_CC\the\numexpr #1+10#5#4#3#2\relax
600 }%
601 \def\XINT_addr_CC #1#2#3#4#5#6#7%
602 {%
603 \XINT_addr_AC_checkcarry #2{#7#6#5#4#3}%
604 }%
605 \def\XINT_addr_AC_checkcarry #1%
606 {%
607 \xint_gob_til_zero #1\xint_addr_AC_nocarry 0\XINT_addr_C
608 }%
609 \def\xint_addr_AC_nocarry 0\XINT_addr_C #1#2\W\X\Y\Z { #1#2}%
610 \def\XINT_addr_C #1#2#3#4#5%
611 {%
612 \xint_gob_til_W #2\xint_addr_cz\W
613 \XINT_addr_D {#5#4#3#2}{#1}%
614 }%
615 \def\XINT_addr_D #1%
616 {%
617 \expandafter\XINT_addr_CC\the\numexpr 1+10#1\relax
618 }%
619 \def\xint_addr_cz\W\XINT_addr_D #1#2{ #21000}%
```

3.18.3 Addition vIII: `\XINT_addm_A`

INPUT: `\romannumeral0\XINT_addm_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>`, ordre normal, pas sur `4n`, leading zeros retirés. Utilisé par la multiplication.

```
620 \def\XINT_addm_A #1#2#3#4#5#6%
621 {%
622 \xint_gob_til_W #3\xint_addm_az\W
623 \XINT_addm_AB #1{#3#4#5#6}{#2}%
624 }%
625 \def\xint_addm_az\W\XINT_addm_AB #1#2%
626 {%
627 \XINT_addm_AC_checkcarry #1%
628 }%
```

3 Package *xintcore* implementation

```
629 \def\XINT_addm_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
630 {%
631   \XINT_addm_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
632 }%
633 \def\XINT_addm_ABE #1#2#3#4#5#6%
634 {%
635   \expandafter\XINT_addm_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
636 }%
637 \def\XINT_addm_ABEA #1#2#3.#4%
638 {%
639   \XINT_addm_A #2{#3#4}%
640 }%
641 \def\XINT_addm_AC_checkcarry #1%
642 {%
643   \xint_gob_til_zero #1\xint_addm_AC_nocarry 0\XINT_addm_C
644 }%
645 \def\xint_addm_AC_nocarry 0\XINT_addm_C #1#2\W\X\Y\Z
646 {%
647   \expandafter
648   \xint_cleanupzeros_andstop
649   \romannumeral0%
650   \XINT_rord_main {}#2%
651   \xint_relax
652   \xint_bye\xint_bye\xint_bye\xint_bye
653   \xint_bye\xint_bye\xint_bye\xint_bye
654   \xint_relax
655   #1%
656 }%
657 \def\XINT_addm_C #1#2#3#4#5%
658 {%
659   \xint_gob_til_W
660   #5\xint_addm_cw
661   #4\xint_addm_cx
662   #3\xint_addm_cy
663   #2\xint_addm_cz
664   \W\XINT_addm_CD {#5#4#3#2}{#1}%
665 }%
666 \def\XINT_addm_CD #1%
667 {%
668   \expandafter\XINT_addm_CC\the\numexpr 1+10#1.%
669 }%
670 \def\XINT_addm_CC #1#2#3.#4%
671 {%
672   \XINT_addm_AC_checkcarry #2{#3#4}%
673 }%
674 \def\xint_addm_cw
675   #1\xint_addm_cx
676   #2\xint_addm_cy
677   #3\xint_addm_cz
678   \W\XINT_addm_CD
679 {%
680   \expandafter\XINT_addm_CDw\the\numexpr 1+#1#2#3.%
```



```

681 }%
682 \def\XINT_addm_CDw #1.#2#3\X\Y\Z
683 {%
684   \XINT_addm_end #1#3%
685 }%
686 \def\xint_addm_cx
687   #1\xint_addm_cy
688   #2\xint_addm_cz
689   \W\XINT_addm_CD
690 {%
691   \expandafter\XINT_addm_CDx\the\numexpr 1+#1#2.%
692 }%
693 \def\XINT_addm_CDx #1.#2#3\Y\Z
694 {%
695   \XINT_addm_end #1#3%
696 }%
697 \def\xint_addm_cy
698   #1\xint_addm_cz
699   \W\XINT_addm_CD
700 {%
701   \expandafter\XINT_addm_CDy\the\numexpr 1+#1.%
702 }%
703 \def\XINT_addm_CDy #1.#2#3\Z
704 {%
705   \XINT_addm_end #1#3%
706 }%
707 \def\xint_addm_cz\W\XINT_addm_CD #1#2#3{\XINT_addm_end #1#3}%
708 \edef\XINT_addm_end #1#2#3#4#5%
709   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5\relax}%

```

3.18.4 Addition vIV: *\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.

```

710 \def\XINT_addp_A #1#2#3#4#5#6%
711 {%
712   \xint_gob_til_W #3\xint_addp_az\W
713   \XINT_addp_AB #1{#3#4#5#6}{#2}%
714 }%
715 \def\xint_addp_az\W\XINT_addp_AB #1#2%
716 {%
717   \XINT_addp_AC_checkcarry #1%
718 }%
719 \def\XINT_addp_AC_checkcarry #1%
720 {%
721   \xint_gob_til_zero #1\xint_addp_AC_nocarry 0\XINT_addp_C
722 }%
723 \def\xint_addp_AC_nocarry 0\XINT_addp_C

```

3 Package *xintcore* implementation

```
724 {%
725   \XINT_addp_F
726 }%
727 \def\XINT_addp_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
728 {%
729   \XINT_addp_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
730 }%
731 \def\XINT_addp_ABE #1#2#3#4#5#6%
732 {%
733   \expandafter\XINT_addp_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
734 }%
735 \def\XINT_addp_ABEA #1#2#3#4#5#6#7%
736 {%
737   \XINT_addp_A #2{#7#6#5#4#3}%<-- attention on met donc \`a droite
738 }%
739 \def\XINT_addp_C #1#2#3#4#5%
740 {%
741   \xint_gob_til_W
742   #5\xint_addp_cw
743   #4\xint_addp_cx
744   #3\xint_addp_cy
745   #2\xint_addp_cz
746   \W\XINT_addp_CD {#5#4#3#2}{#1}%
747 }%
748 \def\XINT_addp_CD #1%
749 {%
750   \expandafter\XINT_addp_CC\the\numexpr 1+10#1\relax
751 }%
752 \def\XINT_addp_CC #1#2#3#4#5#6#7%
753 {%
754   \XINT_addp_AC_checkcarry #2{#7#6#5#4#3}%
755 }%
756 \def\xint_addp_cw
757   #1\xint_addp_cx
758   #2\xint_addp_cy
759   #3\xint_addp_cz
760   \W\XINT_addp_CD
761 {%
762   \expandafter\XINT_addp_CDw\the\numexpr \xint_c_i+10#1#2#3\relax
763 }%
764 \def\XINT_addp_CDw #1#2#3#4#5#6%
765 {%
766   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDw_zeros
767   0000\XINT_addp_endDw #2#3#4#5%
768 }%
769 \def\XINT_addp_endDw_zeros 0000\XINT_addp_endDw 0000#1\X\Y\Z{ #1}%
770 \def\XINT_addp_endDw #1#2#3#4#5\X\Y\Z{ #5#4#3#2#1}%
771 \def\xint_addp_cx
772   #1\xint_addp_cy
773   #2\xint_addp_cz
774   \W\XINT_addp_CD
775 }
```

3 Package *xintcore* implementation

```
776 \expandafter\XINT_addp_CDx\the\numexpr \xint_c_i+100#1#2\relax
777 }%
778 \def\XINT_addp_CDx #1#2#3#4#5#6%
779 {%
780 \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDx_zeros
781 0000\XINT_addp_endDx #2#3#4#5%
782 }%
783 \def\XINT_addp_endDx_zeros 0000\XINT_addp_endDx 0000#1\Y\Z{ #1}%
784 \def\XINT_addp_endDx #1#2#3#4#5\Y\Z{ #5#4#3#2#1}%
785 \def\xint_addp_cy #1\xint_addp_cz\W\XINT_addp_CD
786 {%
787 \expandafter\XINT_addp_CDy\the\numexpr \xint_c_i+1000#1\relax
788 }%
789 \def\XINT_addp_CDy #1#2#3#4#5#6%
790 {%
791 \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDy_zeros
792 0000\XINT_addp_endDy #2#3#4#5%
793 }%
794 \def\XINT_addp_endDy_zeros 0000\XINT_addp_endDy 0000#1\Z{ #1}%
795 \def\XINT_addp_endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
796 \def\xint_addp_cz\W\XINT_addp_CD #1#2{ #21000}%
797 \def\XINT_addp_F #1#2#3#4#5%
798 {%
799 \xint_gob_til_W
800 #5\xint_addp_Gw
801 #4\xint_addp_Gx
802 #3\xint_addp_Gy
803 #2\xint_addp_Gz
804 \W\XINT_addp_G {#2#3#4#5}{#1}%
805 }%
806 \def\XINT_addp_G #1#2%
807 {%
808 \XINT_addp_F {#2#1}%
809 }%
810 \def\xint_addp_Gw
811 #1\xint_addp_Gx
812 #2\xint_addp_Gy
813 #3\xint_addp_Gz
814 \W\XINT_addp_G #4%
815 {%
816 \xint_gob_til_zeros_iv #3#2#10\XINT_addp_endGw_zeros
817 0000\XINT_addp_endGw #3#2#10%
818 }%
819 \def\XINT_addp_endGw_zeros 0000\XINT_addp_endGw 0000#1\X\Y\Z{ #1}%
820 \def\XINT_addp_endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
821 \def\xint_addp_Gx
822 #1\xint_addp_Gy
823 #2\xint_addp_Gz
824 \W\XINT_addp_G #3%
825 {%
826 \xint_gob_til_zeros_iv #2#100\XINT_addp_endGx_zeros
827 0000\XINT_addp_endGx #2#100%
```

```

828 }%
829 \def\XINT_addp_endGx_zeros 0000\XINT_addp_endGx 0000#1\Y\Z{ #1}%
830 \def\XINT_addp_endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
831 \def\xint_addp_Gy
832   #1\xint_addp_Gz
833   \W\XINT_addp_G #2%
834 {%
835   \xint_gob_til_zeros_iv #1000\XINT_addp_endGy_zeros
836   0000\XINT_addp_endGy #1000%
837 }%
838 \def\XINT_addp_endGy_zeros 0000\XINT_addp_endGy 0000#1\Z{ #1}%
839 \def\XINT_addp_endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
840 \def\xint_addp_Gz\W\XINT_addp_G #1#2{ #2}%

```

3.19 \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.

```

841 \def\xintiiAdd {\romannumeral0\xintiiadd }%
842 \def\xintiiadd #1{\expandafter\xint_iiadd\romannumeral-`0#1\Z }%
843 \def\xint_iiadd #1#2\Z #3%
844 {%
845   \expandafter\XINT_add_fork\expandafter #1\romannumeral-`0#3\Z #2\Z
846 }%
847 \def\xintiAdd {\romannumeral0\xintiadd }%
848 \def\xintiadd #1%
849 {%
850   \expandafter\xint_add\romannumeral0\xintnum{#1}\Z
851 }%
852 \def\xint_add #1#2\Z #3%
853 {%
854   \expandafter\XINT_add_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
855 }%
856 \let\xintAdd\xintiAdd \let\xintadd\xintiadd
857 \def\XINT_add_fork #1#2%
858 {%
859   \xint_UDzerofork
860   #1\XINT_add_firstiszero
861   #2\XINT_add_secondiszero
862   0{ }%
863   \krof
864   \xint_UDsignsfork
865   #1#2\XINT_add_minusminus
866   #1-\XINT_add_minusplus
867   #2-\XINT_add_plusminus
868   --\XINT_add_plusplus
869   \krof #1#2%
870 }%
871 \def\XINT_add_firstiszero #1\krof #2#3\Z #4\Z { #3}%

```

3 Package *xintcore* implementation

```
872 \def\XINT_add_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
873 \def\XINT_add_plusplus #1#2#3\Z #4\Z {\XINT_add_pre {#1#4}{#2#3}}%
874 \def\XINT_add_minusminus #1#2#3\Z #4\Z
875   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#3}}%
876 \def\XINT_add_minusplus #1#2#3\Z #4\Z {\XINT_sub_pre {#2#3}{#4}}%
877 \def\XINT_add_plusminus #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#3}}%
```

positive summands

```
878 \def\XINT_add_pre #1%
879 {%
880   \expandafter\XINT_add_pre_b\expandafter
881   {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z }%
882 }%
883 \def\XINT_add_pre_b #1#2%
884 {%
885   \expandafter\XINT_add_A
886   \expandafter0\expandafter{\expandafter}%
887   \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
888   \W\X\Y\Z #1\W\X\Y\Z
889 }%
```

3.20 \xintiSub, \xintiiSub

Release 1.09a has \xintnum added into \xintiSub.

```
890 \def\xintiiSub {\romannumeral0\xintiisub }%
891 \def\xintiisub #1{\expandafter\xint_iisub\romannumeral-`0#1\Z }%
892 \def\xint_iisub #1#2\Z #3%
893 {%
894   \expandafter\XINT_sub_fork\expandafter #1\romannumeral-`0#3\Z #2\Z
895 }%
896 \def\xintiSub {\romannumeral0\xintisub }%
897 \def\xintisub #1%
898 {%
899   \expandafter\xint_sub\romannumeral0\xintnum{#1}\Z
900 }%
901 \def\xint_sub #1#2\Z #3%
902 {%
903   \expandafter\XINT_sub_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
904 }%
905 \let\xintSub\xintiSub \let\xintsub\xintisub
906 \def\XINT_sub_fork #1#2%
907 {%
908   \xint_UDzerofork
909   #1\XINT_sub_firstiszero
910   #2\XINT_sub_secondiszero
911   0}%
912 \krof
913 \xint_UDsignsfork
914   #1#2\XINT_sub_minusminus
915   #1-\XINT_sub_minusplus
916   #2-\XINT_sub_plusminus
```

3 Package *xintcore* implementation

```

917      --\XINT_sub_plusplus
918      \krof #1#2%
919 }%
920 \def\XINT_sub_firstiszero #1\krof #2#3\Z #4\Z {\XINT_opp #3}%
921 \def\XINT_sub_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
922 \def\XINT_sub_plusplus #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#2#3}}%
923 \def\XINT_sub_minusminus #1#2#3\Z #4\Z {\XINT_sub_pre {#3}{#4}}%
924 \def\XINT_sub_minusplus #1#2#3\Z #4\Z
925   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#2#3}}%
926 \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

```

927 \def\XINT_sub_pre #1%
928 {%
929   \expandafter\XINT_sub_pre_b\expandafter
930   {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z }%
931 }%
932 \def\XINT_sub_pre_b #1#2%
933 {%
934   \expandafter\XINT_sub_A
935   \expandafter1\expandafter{\expandafter}%
936   \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
937   \W\X\Y\Z #1 \W\X\Y\Z
938 }%

```

\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.

```

939 \def\XINT_sub_A #1#2#3\W\X\Y\Z #4#5#6#7%
940 {%
941   \xint_gob_til_W
942   #4\xint_sub_az
943   \W\XINT_sub_B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
944 }%
945 \def\XINT_sub_B #1#2#3#4#5#6#7%
946 {%
947   \xint_gob_til_W
948   #4\xint_sub_bz
949   \W\XINT_sub_onestep #1#2{#7#6#5#4}{#3}%
950 }%

```

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.

```

951 \def\XINT_sub_onestep #1#2#3#4#5#6%
952 {%
953   \expandafter\XINT_sub_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
954 }%

```

ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE

```

955 \def\XINT_sub_backtoA #1#2#3.#4%
956 {%
957     \XINT_sub_A #2{#3#4}%
958 }%
959 \def\xint_sub_bz
960     \W\XINT_sub_onestep #1#2#3#4#5#6#7%
961 {%
962     \xint_UDzerofork
963     #1\XINT_sub_C    % une retenue
964     0\XINT_sub_D    % pas de retenue
965     \krof
966     {#7}#2#3#4#5%
967 }%
968 \def\XINT_sub_D #1#2\W\X\Y\Z
969 {%
970     \expandafter
971     \xint_cleanupzeros_andstop
972     \romannumeral0%
973     \XINT_rord_main {}#2%
974     \xint_relax
975     \xint_bye\xint_bye\xint_bye\xint_bye
976     \xint_bye\xint_bye\xint_bye\xint_bye
977     \xint_relax
978     #1%
979 }%
980 \def\XINT_sub_C #1#2#3#4#5%
981 {%
982     \xint_gob_til_W
983     #2\xint_sub_cz
984     \W\XINT_sub_AC_onestep {#5#4#3#2}{#1}%
985 }%
986 \def\XINT_sub_AC_onestep #1%
987 {%
988     \expandafter\XINT_sub_backtoC\the\numexpr 11#1-\xint_c_i.%
989 }%
990 \def\XINT_sub_backtoC #1#2#3.#4%
991 {%
992     \XINT_sub_AC_checkcarry #2{#3#4}% la retenue va \^etre examin\^ee
993 }%
994 \def\XINT_sub_AC_checkcarry #1%
995 {%
996     \xint_gob_til_one #1\xint_sub_AC_nocarry 1\XINT_sub_C
997 }%
998 \def\xint_sub_AC_nocarry 1\XINT_sub_C #1#2\W\X\Y\Z
999 {%
1000     \expandafter
1001     \XINT_cuz_loop
1002     \romannumeral0%
1003     \XINT_rord_main {}#2%
1004     \xint_relax
1005     \xint_bye\xint_bye\xint_bye\xint_bye

```

3 Package *xintcore* implementation

```

1006      \xint_bye\xint_bye\xint_bye\xint_bye
1007      \xint_relax
1008      #1\W\W\W\W\W\W\W\Z
1009 }%
1010 \def\xint_sub_cz\W\XINT_sub_AC_onestep #1%
1011 {%
1012      \XINT_cuz
1013 }%
1014 \def\xint_sub_az\W\XINT_sub_B #1#2#3#4#5#6#7%
1015 {%
1016      \xint_gob_til_W
1017      #4\xint_sub_ez
1018      \W\XINT_sub_Eenter #1{#3}#4#5#6#7%
1019 }%

```

le premier nombre continue, le résultat sera < 0 .

```

1020 \def\XINT_sub_Eenter #1#2%
1021 {%
1022      \expandafter
1023      \XINT_sub_E\expandafter1\expandafter{\expandafter}%
1024      \romannumeral0%
1025      \XINT_rord_main {}#2%
1026      \xint_relax
1027      \xint_bye\xint_bye\xint_bye\xint_bye
1028      \xint_bye\xint_bye\xint_bye\xint_bye
1029      \xint_relax
1030      \W\X\Y\Z #1%
1031 }%
1032 \def\XINT_sub_E #1#2#3#4#5#6%
1033 {%
1034      \xint_gob_til_W #3\xint_sub_F\W
1035      \XINT_sub_Eonestep #1{#6#5#4#3}{#2}%
1036 }%
1037 \def\XINT_sub_Eonestep #1#2%
1038 {%
1039      \expandafter\XINT_sub_backtoE\the\numexpr 109999-#2+#1.%
1040 }%
1041 \def\XINT_sub_backtoE #1#2#3.#4%
1042 {%
1043      \XINT_sub_E #2{#3#4}%
1044 }%
1045 \def\xint_sub_F\W\XINT_sub_Eonestep #1#2#3#4%
1046 {%
1047      \xint_UDonezerofork
1048      #4#1{\XINT_sub_Fdec 0}% soustraire 1. Et faire signe -
1049      #1#4{\XINT_sub_Finc 1}% additionner 1. Et faire signe -
1050      10\XINT_sub_DD      % terminer. Mais avec signe -
1051      \krof
1052      {#3}%
1053 }%
1054 \def\XINT_sub_DD {\expandafter\xint_minus_thenstop\romannumeral0\XINT_sub_D }%
1055 \def\XINT_sub_Fdec #1#2#3#4#5#6%

```


3 Package *xintcore* implementation

```

1056 {%
1057   \xint_gob_til_W #3\xint_sub_Fdec_finish\W
1058   \XINT_sub_Fdec_onestep #1{#6#5#4#3}{#2}%
1059 }%
1060 \def\XINT_sub_Fdec_onestep #1#2%
1061 {%
1062   \expandafter\XINT_sub_backtoFdec\the\numexpr 11#2+#1-\xint_c_i.%
1063 }%
1064 \def\XINT_sub_backtoFdec #1#2#3.#4%
1065 {%
1066   \XINT_sub_Fdec #2{#3#4}%
1067 }%
1068 \def\xint_sub_Fdec_finish\W\XINT_sub_Fdec_onestep #1#2%
1069 {%
1070   \expandafter\xint_minus_thenstop\romannumeral0\XINT_cuz
1071 }%
1072 \def\XINT_sub_Finc #1#2#3#4#5#6%
1073 {%
1074   \xint_gob_til_W #3\xint_sub_Finc_finish\W
1075   \XINT_sub_Finc_onestep #1{#6#5#4#3}{#2}%
1076 }%
1077 \def\XINT_sub_Finc_onestep #1#2%
1078 {%
1079   \expandafter\XINT_sub_backtoFinc\the\numexpr 10#2+#1.%
1080 }%
1081 \def\XINT_sub_backtoFinc #1#2#3.#4%
1082 {%
1083   \XINT_sub_Finc #2{#3#4}%
1084 }%
1085 \def\xint_sub_Finc_finish\W\XINT_sub_Finc_onestep #1#2#3%
1086 {%
1087   \xint_UDzerofork
1088   #1{\expandafter\expandafter\expandafter
1089     \xint_minus_thenstop\xint_cleanupzeros_nostop}%
1090   0{ -1}%
1091   \krof
1092   #3%
1093 }%
1094 \def\xint_sub_ez\W\XINT_sub_Eenter #1%
1095 {%
1096   \xint_UDzerofork
1097   #1\XINT_sub_K % il y a une retenue
1098   0\XINT_sub_L % pas de retenue
1099   \krof
1100 }%
1101 \def\XINT_sub_L #1\W\X\Y\Z {\XINT_cuz_loop #1\W\W\W\W\W\W\Z }%
1102 \def\XINT_sub_K #1%
1103 {%
1104   \expandafter
1105   \XINT_sub_KK\expandafter1\expandafter{\expandafter}%
1106   \romannumeral0%
1107   \XINT_rord_main {}#1%

```

3 Package *xintcore* implementation

```
1108 \xint_relax
1109 \xint_bye\xint_bye\xint_bye\xint_bye
1110 \xint_bye\xint_bye\xint_bye\xint_bye
1111 \xint_relax
1112 }%
1113 \def\XINT_sub_KK #1#2#3#4#5#6%
1114 {%
1115 \xint_gob_til_W #3\xint_sub_KK_finish\W
1116 \XINT_sub_KK_onestep #1{#6#5#4#3}{#2}%
1117 }%
1118 \def\XINT_sub_KK_onestep #1#2%
1119 {%
1120 \expandafter\XINT_sub_backtoKK\the\numexpr 109999-#2+#1.%
1121 }%
1122 \def\XINT_sub_backtoKK #1#2#3.#4%
1123 {%
1124 \XINT_sub_KK #2{#3#4}%
1125 }%
1126 \def\xint_sub_KK_finish\W\XINT_sub_KK_onestep #1#2#3%
1127 {%
1128 \expandafter\xint_minus_thenstop
1129 \romannumeral0\XINT_cuz_loop #3\W\W\W\W\W\W\W\Z
1130 }%
```

3.21 *\xintiMul*, *\xintiiMul*

1.09a adds *\xintnum*

```
1131 \def\xintiiMul {\romannumeral0\xintiimul }%
1132 \def\xintiimul #1%
1133 {%
1134 \expandafter\xint_iimul\expandafter {\romannumeral-`0#1}%
1135 }%
1136 \def\xint_iimul #1#2%
1137 {%
1138 \expandafter\XINT_mul_fork \romannumeral-`0#2\Z #1\Z
1139 }%
1140 \def\xintiMul {\romannumeral0\xintimul }%
1141 \def\xintimul #1%
1142 {%
1143 \expandafter\xint_mul\expandafter {\romannumeral0\xintnum{#1}}%
1144 }%
1145 \def\xint_mul #1#2%
1146 {%
1147 \expandafter\XINT_mul_fork \romannumeral0\xintnum{#2}\Z #1\Z
1148 }%
1149 \let\xintMul\xintiMul \let\xintmul\xintimul
1150 \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

3 Package *xintcore* implementation

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.

```

1151 \def\XINT_mul_fork #1#2\Z #3#4\Z
1152 {%
1153   \xint_UDzerofork
1154     #1\XINT_mul_zero
1155     #3\XINT_mul_zero
1156     0{}}%
1157   \krof
1158   \xint_UDsignsfork
1159     #1#3\XINT_mul_minusminus      % #1 = #3 = -
1160     #1-\XINT_mul_minusplus #3}%   % #1 = -
1161     #3-\XINT_mul_plusminus #1}%   % #3 = -
1162     --\XINT_mul_plusplus #1#3}%
1163   \krof
1164   {#2}{#4}%
1165 }%
1166 \def\XINT_mul_zero #1\krof #2#3{ 0}%
1167 \def\XINT_mul_minusminus #1#2%
1168 {%
1169   \expandafter\XINT_mul_choice_a
1170   \expandafter{\romannumeral0\xintlength {#2}}%
1171   {\romannumeral0\xintlength {#1}}{#1}{#2}%
1172 }%
1173 \def\XINT_mul_minusplus #1#2#3%
1174 {%
1175   \expandafter\xint_minus_thenstop\romannumeral0\expandafter
1176   \XINT_mul_choice_a
1177   \expandafter{\romannumeral0\xintlength {#1#3}}%
1178   {\romannumeral0\xintlength {#2}}{#2}{#1#3}%
1179 }%
1180 \def\XINT_mul_plusminus #1#2#3%
1181 {%
1182   \expandafter\xint_minus_thenstop\romannumeral0\expandafter
1183   \XINT_mul_choice_a
1184   \expandafter{\romannumeral0\xintlength {#3}}%
1185   {\romannumeral0\xintlength {#1#2}}{#1#2}{#3}%
1186 }%
1187 \def\XINT_mul_plusplus #1#2#3#4%
1188 {%
1189   \expandafter\XINT_mul_choice_a
1190   \expandafter{\romannumeral0\xintlength {#2#4}}%
1191   {\romannumeral0\xintlength {#1#3}}{#1#3}{#2#4}%
1192 }%
1193 \def\XINT_mul_choice_a #1#2%
1194 {%
1195   \expandafter\XINT_mul_choice_b\expandafter{#2}{#1}%
1196 }%
1197 \def\XINT_mul_choice_b #1#2%
1198 {%

```

3 Package *xintcore* implementation

```

1199 \ifnum #1<\xint_c_v
1200 \expandafter\XINT_mul_choice_littlebyfirst
1201 \else
1202 \ifnum #2<\xint_c_v
1203 \expandafter\expandafter\expandafter\XINT_mul_choice_littlebysecond
1204 \else
1205 \expandafter\expandafter\expandafter\XINT_mul_choice_compare
1206 \fi
1207 \fi
1208 {#1}{#2}%
1209 }%
1210 \def\XINT_mul_choice_littlebyfirst #1#2#3#4%
1211 {%
1212 \expandafter\XINT_mul_M
1213 \expandafter{\the\numexpr #3\expandafter}%
1214 \romannumeral0\XINT_RQ {}#4\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
1215 }%
1216 \def\XINT_mul_choice_littlebysecond #1#2#3#4%
1217 {%
1218 \expandafter\XINT_mul_M
1219 \expandafter{\the\numexpr #4\expandafter}%
1220 \romannumeral0\XINT_RQ {}#3\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
1221 }%
1222 \def\XINT_mul_choice_compare #1#2%
1223 {%
1224 \ifnum #1>#2
1225 \expandafter \XINT_mul_choice_i
1226 \else
1227 \expandafter \XINT_mul_choice_ii
1228 \fi
1229 {#1}{#2}%
1230 }%
1231 \def\XINT_mul_choice_i #1#2%
1232 {%
1233 \ifnum #1<\numexpr\ifcase \numexpr (#2-\xint_c_iii)/\xint_c_iv\relax
1234 \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1235 \expandafter\XINT_mul_choice_same
1236 \else
1237 \expandafter\XINT_mul_choice_permute
1238 \fi
1239 }%
1240 \def\XINT_mul_choice_ii #1#2%
1241 {%
1242 \ifnum #2<\numexpr\ifcase \numexpr (#1-\xint_c_iii)/\xint_c_iv\relax
1243 \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1244 \expandafter\XINT_mul_choice_permute
1245 \else
1246 \expandafter\XINT_mul_choice_same
1247 \fi
1248 }%
1249 \def\XINT_mul_choice_same #1#2%
1250 {%

```

3 Package *xintcore* implementation

```
1251 \expandafter\XINT_mul_enter
1252 \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
1253 \Z\Z\Z\Z #2\W\W\W\W
1254 }%
1255 \def\XINT_mul_choice_permute #1#2%
1256 {%
1257 \expandafter\XINT_mul_enter
1258 \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
1259 \Z\Z\Z\Z #1\W\W\W\W
1260 }%
```

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 $4n$.

```
1261 \def\XINT_mul_Ar #1#2#3#4#5#6%
1262 {%
1263 \xint_gob_til_Z #6\xint_mul_br\Z\XINT_mul_Br #1{#6#5#4#3}{#2}%
1264 }%
1265 \def\xint_mul_br\Z\XINT_mul_Br #1#2%
1266 {%
1267 \XINT_addr_AC_checkcarry #1%
1268 }%
1269 \def\XINT_mul_Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
1270 {%
1271 \expandafter\XINT_mul_ABEAr
1272 \the\numexpr #1+10#2+#8#7#6#5.{#3}#4\W\X\Y\Z
1273 }%
1274 \def\XINT_mul_ABEAr #1#2#3#4#5#6.#7%
1275 {%
1276 \XINT_mul_Ar #2{#7#6#5#4#3}%
1277 }%
```

3.21.1 “Small” multiplication: `\XINT_mul_Mr`

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

```
1278 \def\XINT_mul_Mr #1%
1279 {%
1280 \expandafter\XINT_mul_Mr_checkifzeroorone\expandafter{\the\numexpr #1}%
1281 }%
1282 \def\XINT_mul_Mr_checkifzeroorone #1%
1283 {%
1284 \ifcase #1
1285 \expandafter\XINT_mul_Mr_zero
1286 \or
1287 \expandafter\XINT_mul_Mr_one
1288 \else
1289 \expandafter\XINT_mul_Nr
1290 \fi
```

3 Package *xintcore* implementation

```

1291 {0000}{#1}%
1292 }%
1293 \def\XINT_mul_Mr_zero #1\Z\Z\Z\Z { 0000}%
1294 \def\XINT_mul_Mr_one #1#2#3#4\Z\Z\Z\Z { #4}%
1295 \def\XINT_mul_Nr #1#2#3#4#5#6#7%
1296 {%
1297 \xint_gob_til_Z #4\xint_mul_pr\Z\XINT_mul_Pr {#1}{#3}{#7#6#5#4}{#2}{#3}%
1298 }%
1299 \def\XINT_mul_Pr #1#2#3%
1300 {%
1301 \expandafter\XINT_mul_Lr\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
1302 }%
1303 \def\XINT_mul_Lr 1#1#2#3#4#5#6#7#8#9%
1304 {%
1305 \XINT_mul_Nr {#1#2#3#4}{#9#8#7#6#5}%
1306 }%
1307 \def\xint_mul_pr\Z\XINT_mul_Pr #1#2#3#4#5%
1308 {%
1309 \xint_gob_til_zeros_iv #1\XINT_mul_Mr_end_nocarry 0000%
1310 \XINT_mul_Mr_end_carry #1{#4}%
1311 }%
1312 \def\XINT_mul_Mr_end_nocarry 0000\XINT_mul_Mr_end_carry 0000#1{ #1}%
1313 \def\XINT_mul_Mr_end_carry #1#2#3#4#5{ #5#4#3#2#1}%

```

3.21.2 “Small” multiplication variant: *\XINT_mul_M*

<< 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*.

```

1314 \def\XINT_mul_M #1%
1315 {%
1316 \expandafter\XINT_mul_M_checkifzeroorone\expandafter{\the\numexpr #1}%
1317 }%
1318 \def\XINT_mul_M_checkifzeroorone #1%
1319 {%
1320 \ifcase #1
1321 \expandafter\XINT_mul_M_zero
1322 \or
1323 \expandafter\XINT_mul_M_one
1324 \else
1325 \expandafter\XINT_mul_N
1326 \fi
1327 {0000}{#1}%
1328 }%
1329 \def\XINT_mul_M_zero #1\Z\Z\Z\Z { 0}%
1330 \def\XINT_mul_M_one #1#2#3#4\Z\Z\Z\Z
1331 {%
1332 \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#4}%
1333 }%
1334 \def\XINT_mul_N #1#2#3#4#5#6#7%
1335 {%

```

3 Package *xintcore* implementation

```
1336 \xint_gob_til_Z #4\xint_mul_p\Z\XINT_mul_P {#1}{#3}{#7#6#5#4}{#2}{#3}%
1337 }%
1338 \def\XINT_mul_P #1#2#3%
1339 {%
1340 \expandafter\XINT_mul_L\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
1341 }%
1342 \def\XINT_mul_L 1#1#2#3#4#5#6#7#8#9%
1343 {%
1344 \XINT_mul_N {#1#2#3#4}{#5#6#7#8#9}%
1345 }%
1346 \def\xint_mul_p\Z\XINT_mul_P #1#2#3#4#5%
1347 {%
1348 \XINT_mul_M_end #1#4%
1349 }%
1350 \edef\XINT_mul_M_end #1#2#3#4#5#6#7#8%
1351 {%
1352 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
1353 }%
```

3.21.3 Main routine: *\XINT_mul_enter*

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 la 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.

```
1354 \def\XINT_mul_enter #1\Z\Z\Z\Z #2#3#4#5%
1355 {%
1356 \xint_gob_til_W #5\XINT_mul_exit_a\W
1357 \XINT_mul_start {#2#3#4#5}#1\Z\Z\Z\Z
1358 }%
1359 \def\XINT_mul_exit_a\W\XINT_mul_start #1%
1360 {%
1361 \XINT_mul_exit_b #1%
1362 }%
1363 \def\XINT_mul_exit_b #1#2#3#4%
1364 {%
1365 \xint_gob_til_W
1366 #2\XINT_mul_exit_ci
1367 #3\XINT_mul_exit_cii
1368 \W\XINT_mul_exit_ciii #1#2#3#4%
1369 }%
```

3 Package *xintcore* implementation

```

1370 \def\XINT_mul_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
1371 {%
1372     \XINT_mul_M {#1}#2\Z\Z\Z\Z
1373 }%
1374 \def\XINT_mul_exit_cii\W\XINT_mul_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
1375 {%
1376     \XINT_mul_M {#1}#2\Z\Z\Z\Z
1377 }%
1378 \def\XINT_mul_exit_ci\W\XINT_mul_exit_cii
1379     \W\XINT_mul_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
1380 {%
1381     \XINT_mul_M {#1}#2\Z\Z\Z\Z
1382 }%
1383 \def\XINT_mul_start #1#2\Z\Z\Z\Z
1384 {%
1385     \expandafter\XINT_mul_main\expandafter
1386     {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
1387 }%
1388 \def\XINT_mul_main #1#2\Z\Z\Z\Z #3#4#5#6%
1389 {%
1390     \xint_gob_til_W #6\XINT_mul_finish_a\W
1391     \XINT_mul_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
1392 }%
1393 \def\XINT_mul_compute #1#2#3\Z\Z\Z\Z
1394 {%
1395     \expandafter\XINT_mul_main\expandafter
1396     {\romannumeral0\expandafter
1397     \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
1398     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
1399     \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
1400 }%

```

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.

```

1401 \def\XINT_mul_finish_a\W\XINT_mul_compute #1%
1402 {%
1403     \XINT_mul_finish_b #1%
1404 }%
1405 \def\XINT_mul_finish_b #1#2#3#4%
1406 {%
1407     \xint_gob_til_W
1408     #1\XINT_mul_finish_c
1409     #2\XINT_mul_finish_ci
1410     #3\XINT_mul_finish_cii
1411     \W\XINT_mul_finish_ciii #1#2#3#4%
1412 }%
1413 \def\XINT_mul_finish_ciii #1\W #2#3\Z\Z\Z\Z \W\W\W
1414 {%
1415     \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1416     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 0000#2\W\X\Y\Z

```



```

1417 }%
1418 \def\XINT_mul_finish_cii
1419   \W\XINT_mul_finish_ciii #1\W\W #2#3\Z\Z\Z\Z \W\W
1420 {%
1421   \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1422   \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
1423 }%
1424 \def\XINT_mul_finish_ci #1\XINT_mul_finish_ciii #2\W\W\W #3#4\Z\Z\Z\Z \W
1425 {%
1426   \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1427   \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
1428 }%
1429 \def\XINT_mul_finish_c #1\XINT_mul_finish_ciii \W\W\W\W #2#3\Z\Z\Z\Z
1430 {%
1431   \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#2}%
1432 }%

```

3.21.4 Variant: `\XINT_mulr_enter`

`\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.

```

1433 \def\XINT_mulr_enter #1\Z\Z\Z\Z #2#3#4#5%
1434 {%
1435   \xint_gob_til_W #5\XINT_mulr_exit_a\W
1436   \XINT_mulr_start {#2#3#4#5}#1\Z\Z\Z\Z
1437 }%
1438 \def\XINT_mulr_exit_a\W\XINT_mulr_start #1%
1439 {%
1440   \XINT_mulr_exit_b #1%
1441 }%
1442 \def\XINT_mulr_exit_b #1#2#3#4%
1443 {%
1444   \xint_gob_til_W
1445   #2\XINT_mulr_exit_ci
1446   #3\XINT_mulr_exit_cii
1447   \W\XINT_mulr_exit_ciii #1#2#3#4%
1448 }%
1449 \def\XINT_mulr_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
1450 {%
1451   \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
1452 }%
1453 \def\XINT_mulr_exit_cii\W\XINT_mulr_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
1454 {%
1455   \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
1456 }%
1457 \def\XINT_mulr_exit_ci\W\XINT_mulr_exit_cii
1458   \W\XINT_mulr_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W

```

3 Package *xintcore* implementation

```

1459 {%
1460   \XINT_mul_Mr {#1}#2\Z\Z\Z
1461 }%
1462 \def\XINT_mulr_start #1#2\Z\Z\Z
1463 {%
1464   \expandafter\XINT_mulr_main\expandafter
1465   {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z}#2\Z\Z\Z
1466 }%
1467 \def\XINT_mulr_main #1#2\Z\Z\Z #3#4#5#6%
1468 {%
1469   \xint_gob_til_W #6\XINT_mulr_finish_a\W
1470   \XINT_mulr_compute {#3#4#5#6}{#1}#2\Z\Z\Z
1471 }%
1472 \def\XINT_mulr_compute #1#2#3\Z\Z\Z
1473 {%
1474   \expandafter\XINT_mulr_main\expandafter
1475   {\romannumeral0\expandafter
1476   \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
1477   \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z
1478   \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z
1479 }%
1480 \def\XINT_mulr_finish_a\W\XINT_mulr_compute #1%
1481 {%
1482   \XINT_mulr_finish_b #1%
1483 }%
1484 \def\XINT_mulr_finish_b #1#2#3#4%
1485 {%
1486   \xint_gob_til_W
1487   #1\XINT_mulr_finish_c
1488   #2\XINT_mulr_finish_ci
1489   #3\XINT_mulr_finish_cii
1490   \W\XINT_mulr_finish_ciii #1#2#3#4%
1491 }%
1492 \def\XINT_mulr_finish_ciii #1\W #2#3\Z\Z\Z \W\W\W
1493 {%
1494   \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1495   \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
1496 }%
1497 \def\XINT_mulr_finish_cii
1498   \W\XINT_mulr_finish_ciii #1\W\W #2#3\Z\Z\Z \W\W
1499 {%
1500   \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1501   \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
1502 }%
1503 \def\XINT_mulr_finish_ci #1\XINT_mulr_finish_ciii #2\W\W\W #3#4\Z\Z\Z \W
1504 {%
1505   \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1506   \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
1507 }%
1508 \def\XINT_mulr_finish_c #1\XINT_mulr_finish_ciii \W\W\W\W #2#3\Z\Z\Z { #2}%

```

3.22 `\xintiSqr`, `\xintiiSqr`

```

1509 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1510 \def\xintiisqr #1%
1511 {%
1512   \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiabs{#1}}%
1513 }%
1514 \def\xintiSqr {\romannumeral0\xintisqr }%
1515 \def\xintisqr #1%
1516 {%
1517   \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiabs{#1}}%
1518 }%
1519 \let\xintSqr\xintiSqr \let\xintsqr\xintisqr
1520 \def\XINT_sqr #1%
1521 {%
1522   \expandafter\XINT_mul_enter
1523     \romannumeral0%
1524     \XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
1525     \Z\Z\Z\Z #1\W\W\W\W
1526 }%

```

3.23 `\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).

```

1527 \def\xintiiPow {\romannumeral0\xintiipow }%
1528 \def\xintiipow #1%
1529 {%
1530   \expandafter\xint_pow\romannumeral-`0#1\Z%
1531 }%
1532 \def\xintiPow {\romannumeral0\xintipow }%
1533 \def\xintipow #1%
1534 {%
1535   \expandafter\xint_pow\romannumeral0\xintnum{#1}\Z%
1536 }%
1537 \let\xintPow\xintiPow \let\xintpow\xintipow
1538 \def\xint_pow #1#2\Z
1539 {%
1540   \xint_UDsignfork

```

3 Package *xintcore* implementation

```

1541      #1\XINT_pow_Aneg
1542      -\XINT_pow_Anonneg
1543      \krof
1544      #1{#2}%
1545 }%
1546 \def\XINT_pow_Aneg #1#2#3%
1547 {%
1548   \expandafter\XINT_pow_Aneg_\expandafter{\the\numexpr #3}#2\Z
1549 }%
1550 \def\XINT_pow_Aneg_ #1%
1551 {%
1552   \ifodd #1
1553     \expandafter\XINT_pow_Aneg_Bodd
1554   \fi
1555   \XINT_pow_Anonneg_ {#1}%
1556 }%
1557 \def\XINT_pow_Aneg_Bodd #1%
1558 {%
1559   \expandafter\XINT_opp\romannumeral0\XINT_pow_Anonneg_
1560 }%

```

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

```

1561 \def\XINT_pow_Anonneg #1#2#3%
1562 {%
1563   \expandafter\XINT_pow_Anonneg_\expandafter {\the\numexpr #3}#1#2\Z
1564 }%

```

#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.

```

1565 \def\XINT_pow_Anonneg_ #1#2#3\Z
1566 {%
1567   \if\relax #3\relax\xint_dothis
1568     {\ifcase #2 \expandafter\XINT_pow_AisZero
1569       \or\expandafter\XINT_pow_AisOne
1570       \else\expandafter\XINT_pow_AatleastTwo
1571       \fi }\fi
1572   \xint_orthat \XINT_pow_AatleastTwo {#1}{#2#3}%
1573 }%
1574 \def\XINT_pow_AisOne #1#2{ 1}%

```

#1 = B

```

1575 \def\XINT_pow_AisZero #1#2%
1576 {%
1577   \ifcase\XINT_cntSgn #1\Z
1578     \xint_afterfi { 1}%
1579   \or
1580     \xint_afterfi { 0}%
1581   \else
1582     \xint_afterfi {\xintError:DivisionByZero\space 0}%
1583   \fi

```

3 Package *xintcore* implementation

```

1584 }%
1585 \def\XINT_pow_AatleastTwo #1%
1586 {%
1587   \ifcase\XINT_cntSgn #1\Z
1588     \expandafter\XINT_pow_BisZero
1589   \or
1590     \expandafter\XINT_pow_checkBsize
1591   \else
1592     \expandafter\XINT_pow_BisNegative
1593   \fi
1594   {#1}%
1595 }%
1596 \edef\XINT_pow_BisNegative #1#2%
1597   {\noexpand\xintError:FractionRoundedToZero\space 0}%
1598 \def\XINT_pow_BisZero #1#2{ 1}%

  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.

1599 \def\XINT_pow_checkBsize #1%
1600 {%
1601   \ifnum #1>1000000
1602     \expandafter\XINT_pow_BtooBig
1603   \else
1604     \expandafter\XINT_pow_loopI
1605   \fi
1606   {#1}%
1607 }%
1608 \edef\XINT_pow_BtooBig #1#2{\noexpand\xintError:ExponentTooBig\space 0}%
1609 \def\XINT_pow_loopI #1%
1610 {%
1611   \ifnum #1=\xint_c_i\XINT_pow_Iend\fi
1612   \ifodd #1
1613     \expandafter\XINT_pow_loopI_odd
1614   \else
1615     \expandafter\XINT_pow_loopI_even
1616   \fi
1617   {#1}%
1618 }%
1619 \edef\XINT_pow_Iend\fi #1\fi #2#3{\noexpand\fi\space #3}%
1620 \def\XINT_pow_loopI_even #1#2%
1621 {%
1622   \expandafter\XINT_pow_loopI\expandafter
1623   {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
1624   {\romannumeral0\xintiisqr {#2}}%
1625 }%
1626 \def\XINT_pow_loopI_odd #1#2%
1627 {%
1628   \expandafter\XINT_pow_loopI_odda\expandafter
1629   {\romannumeral0\XINT_RQ }{#2\R\R\R\R\R\R\R\R\Z }{#1}{#2}%
1630 }%
1631 \def\XINT_pow_loopI_odda #1#2#3%
1632 {%

```

```

1633 \expandafter\XINT_pow_loopII\expandafter
1634 {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
1635 {\romannumeral0\xintiisqr {#3}}{#1}%
1636 }%
1637 \def\XINT_pow_loopII #1%
1638 {%
1639 \ifnum #1 = \xint_c_i\XINT_pow_IIend\fi
1640 \ifodd #1
1641 \expandafter\XINT_pow_loopII_odd
1642 \else
1643 \expandafter\XINT_pow_loopII_even
1644 \fi
1645 {#1}%
1646 }%
1647 \def\XINT_pow_loopII_even #1#2%
1648 {%
1649 \expandafter\XINT_pow_loopII\expandafter
1650 {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
1651 {\romannumeral0\xintiisqr {#2}}%
1652 }%
1653 \def\XINT_pow_loopII_odd #1#2#3%
1654 {%
1655 \expandafter\XINT_pow_loopII_odda\expandafter
1656 {\romannumeral0\XINT_mulr_enter #3\Z\Z\Z\Z #2\W\W\W\W}{#1}{#2}%
1657 }%
1658 \def\XINT_pow_loopII_odda #1#2#3%
1659 {%
1660 \expandafter\XINT_pow_loopII\expandafter
1661 {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
1662 {\romannumeral0\xintiisqr {#3}}{#1}%
1663 }%
1664 \def\XINT_pow_IIend\fi #1\fi #2#3#4%
1665 {%
1666 \fi\XINT_mul_enter #4\Z\Z\Z\Z #3\W\W\W\W
1667 }%

```

3.24 `\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, 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

3 Package *xintcore* implementation

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.

```

1668 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1669 \def\xintiiRem {\romannumeral0\xintiirem }%
1670 \def\xintiiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1671 \def\xintiirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
1672 \def\xintiQuo {\romannumeral0\xintiquo }%
1673 \def\xintiRem {\romannumeral0\xintirem }%
1674 \def\xintiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision }%
1675 \def\xintirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision }%
1676 \let\xintQuo\xintiQuo\let\xintquo\xintiquo % deprecated (1.1)
1677 \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: $A=BQ+R$, $0 \leq R < |B|$.

```

1678 \def\xintiDivision {\romannumeral0\xintidivision }%
1679 \def\xintidivision #1{\expandafter\XINT_division\romannumeral0\xintnum{#1}\Z }%
1680 \let\xintDivision\xintiDivision \let\xintdivision\xintidivision % deprecated
1681 \def\XINT_division #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1682 \romannumeral0\xintnum{#3}\Z #2\Z }%
1683 \def\xintiiDivision {\romannumeral0\xintiidivision }%
1684 \def\xintiidivision #1{\expandafter\XINT_iidivision \romannumeral-`0#1\Z }%
1685 \def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1686 \romannumeral-`0#3\Z #2\Z }%
1687 \def\XINT_iidivision_a #1#2# #1 de A, #2 de B.
1688 {%
1689 \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
1690 \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1691 \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1692 \romannumeral0\XINT_iidivision_bpos #1}\fi
1693 \xint_orthat{\XINT_iidivision_bpos #1#2}%
1694 }%
1695 \def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space {0}{0}}%
1696 \def\XINT_iidivision_aiszero #1\Z #2\Z { {0}{0}}%
1697 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1698 {\expandafter\space\expandafter{\romannumeral0\XINT_opp #1}}%
1699 \def\XINT_iidivision_bpos #1%
1700 {%
1701 \xint_UDsignfork
1702 #1\XINT_iidivision_aneg
1703 -{\XINT_iidivision_apos #1}%

```

3 Package *xintcore* implementation

```

1704 \krof
1705 }%
1706 \def\XINT_iidivision_apos #1#2\Z #3\Z{\XINT_div_prepare {#2}{#1#3}}%
1707 \def\XINT_iidivision_aneg #1\Z #2\Z
1708 {\expandafter
1709 \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1710 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\Z
1711 \expandafter\XINT_iidivision_aneg_rzero
1712 \else
1713 \expandafter\XINT_iidivision_aneg_rpos
1714 \fi {#1}{#2}}%
1715 \def\XINT_iidivision_aneg_rzero #1#2#3{ {-#1}{0}}% necessarily q was >0
1716 \def\XINT_iidivision_aneg_rpos #1%
1717 {%
1718 \expandafter\XINT_iidivision_aneg_end\expandafter
1719 {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1720 }%
1721 \def\XINT_iidivision_aneg_end #1#2#3%
1722 {%
1723 \expandafter\xint_exchangetwo_keepbraces_thenstop
1724 \expandafter{\romannumeral0\XINT_sub_pre {#3}{#2}}{#1}% r-> b-r
1725 }%

```

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

```

1726 \def\XINT_div_prepare #1%
1727 {%
1728 \expandafter \XINT_div_prepareB_aa \expandafter
1729 {\romannumeral0\xintlength {#1}}{#1}% B > 0 ici
1730 }%
1731 \def\XINT_div_prepareB_aa #1%
1732 {%
1733 \ifnum #1=\xint_c_i
1734 \expandafter\XINT_div_prepareB_onedigit
1735 \else
1736 \expandafter\XINT_div_prepareB_a
1737 \fi
1738 {#1}%
1739 }%
1740 \def\XINT_div_prepareB_a #1%
1741 {%
1742 \expandafter\XINT_div_prepareB_c\expandafter
1743 {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
1744 }%

```

B=1 and B=2 treated specially.

```

1745 \def\XINT_div_prepareB_onedigit #1#2%
1746 {%
1747 \ifcase#2
1748 \or\expandafter\XINT_div_BisOne
1749 \or\expandafter\XINT_div_BisTwo
1750 \else\expandafter\XINT_div_prepareB_e

```


3 Package *xintcore* implementation

```

1751 \fi {000}{0}{4}{#2}%
1752 }%
1753 \def\XINT_div_BisOne #1#2#3#4#5{ {#5}{0}}%
1754 \def\XINT_div_BisTwo #1#2#3#4#5%
1755 {%
1756 \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1757 \ifodd\xintiilDg{#5} \expandafter1\else \expandafter0\fi {#5}%
1758 }%
1759 \edef\XINT_div_BisTwo_a #1#2%
1760 {%
1761 \noexpand\expandafter\space\noexpand\expandafter
1762 { \noexpand\romannumeral0\noexpand\xinthalft {#2}}{#1}%
1763 }%

```

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

```

1764 \def\XINT_div_prepareB_c #1#2%
1765 {%
1766 \csname XINT_div_prepareB_d\romannumeral\numexpr#1-#2\endcsname
1767 {#1}%
1768 }%
1769 \def\XINT_div_prepareB_d {\XINT_div_prepareB_e }{0000}%
1770 \def\XINT_div_prepareB_di {\XINT_div_prepareB_e {0}{000}}%
1771 \def\XINT_div_prepareB_dii {\XINT_div_prepareB_e {00}{00}}%
1772 \def\XINT_div_prepareB_diii {\XINT_div_prepareB_e {000}{0}}%
1773 \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_checksigrx 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

```

1774 \def\XINT_div_prepareB_e #1#2#3#4%
1775 {%
1776 \ifnum#3=\xint_c_iv\expandafter\XINT_div_prepareLittleB_f
1777 \else\expandafter\XINT_div_prepareB_f
1778 \fi
1779 #4#1{#3}{#2}{#1}%
1780 }%

```

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.

```

1781 \def\XINT_div_prepareB_f #1#2#3#4#5{%
1782 \expandafter\XINT_div_prepareB_g
1783 \the\numexpr #1#2#3#4+\xint_c_i\expandafter
1784 .\the\numexpr (#1#2#3#4+\xint_c_i)/\xint_c_ii\expandafter
1785 .\romannumeral0\xintreverseorder {#1#2#3#4#5}.{#1#2#3#4}%
1786 }%
1787 \def\XINT_div_prepareLittleB_f #1{%
1788 \expandafter\XINT_div_prepareB_g \the\numexpr #1/\xint_c_ii.{}.{#1}%
1789 }%

```

3 Package *xintcore* implementation

#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.

```

1790 \def\XINT_div_prepareB_g #1.#2.#3.#4#5#6#7#8%
1791 {%
1792   \XINT_div_prepareA_a {#8#7}{#5}{#1}{#2}{#4}{#3}{#6}%
1793 }%

  A, K, {x'yx}, B«c»

1794 \def\XINT_div_prepareA_a #1%
1795 {%
1796   \expandafter\XINT_div_prepareA_b\expandafter
1797   {\romannumeral0\xintlength {#1}}{#1}%
1798 }%

  L0, A, K, {x'yx}, B«c»

1799 \def\XINT_div_prepareA_b #1%
1800 {%
1801   \expandafter\XINT_div_prepareA_c\expandafter
1802   {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
1803 }%

  L, L0, A, K, {x'yx}, B, «c»

1804 \def\XINT_div_prepareA_c #1#2%
1805 {%
1806   \csname XINT_div_prepareA_d\romannumeral\numexpr #1-#2\endcsname
1807   {#1}%
1808 }%
1809 \def\XINT_div_prepareA_d {\XINT_div_prepareA_e {}}%
1810 \def\XINT_div_prepareA_di {\XINT_div_prepareA_e {0}}%
1811 \def\XINT_div_prepareA_dii {\XINT_div_prepareA_e {00}}%
1812 \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»

1813 \def\XINT_div_prepareA_e #1#2#3#4#5%
1814 {%
1815   \XINT_div_start_a {#2}{#4}{#1#3}%5%
1816 }%

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

1817 \def\XINT_div_start_a #1#2%
1818 {%
1819   \ifnum #2=\xint_c_iv \expandafter\XINT_div_little_b
1820   \else
1821     \ifnum #1 < #2
1822       \expandafter\expandafter\expandafter\XINT_div_III_aa
1823     \else
1824       \expandafter\expandafter\expandafter\XINT_div_start_b

```

3 Package *xintcore* implementation

```

1825 \fi
1826 \fi
1827 {#1}{#2}%
1828 }%

L, K, A, x', y, x, B, «C».

1829 \def\XINT_div_III_aa #1#2#3#4#5#6#7%
1830 {%
1831 \expandafter\expandafter\expandafter
1832 \XINT_div_III_b\xint_cleanupzeros_nostop #3.{0000}%
1833 }%

R.Q«C».

1834 \def\XINT_div_III_b #1%
1835 {%
1836 \if0#1%
1837 \expandafter\XINT_div_III_bRzero
1838 \else
1839 \expandafter\XINT_div_III_bRpos
1840 \fi
1841 #1%
1842 }%
1843 \def\XINT_div_III_bRzero 0.#1#2%
1844 {%
1845 \expandafter\space\expandafter
1846 {\romannumeral0\XINT_cuz_loop #1\W\W\W\W\W\W\W\Z}{0}%
1847 }%
1848 \def\XINT_div_III_bRpos #1.#2#3%
1849 {%
1850 \expandafter\XINT_div_III_c \XINT_div_cleanR #1#3.{#2}%
1851 }%
1852 \def\XINT_div_III_c #1#2%
1853 {%
1854 \expandafter\space\expandafter
1855 {\romannumeral0\XINT_cuz_loop #2\W\W\W\W\W\W\W\Z}{#1}%
1856 }%

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

1857 \def\XINT_div_start_b #1#2#3#4#5#6%
1858 {%
1859 \XINT_div_start_c {#2}.#3.{#6}{#1}{#2}{#4}{#5}{#6}%
1860 }%

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

1861 \def\XINT_div_start_c #1#2.#3#4#5#6%
1862 {%
1863 \ifnum #1=\xint_c_iv\XINT_div_start_ca\fi
1864 \expandafter\XINT_div_start_c\expandafter
1865 {\the\numexpr #1-\xint_c_iv}#2#3#4#5#6.%
1866 }%
1867 \def\XINT_div_start_ca\fi\expandafter\XINT_div_start_c\expandafter
1868 #1#2#3#4#5{\fi\XINT_div_start_d {#2#3#4#5}#2#3#4#5}%

```

3 Package *xintcore* implementation

#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.

```
1869 \def\XINT_div_start_d #1#2.#3.#4#5#6%
1870 {%
1871   \XINT_div_I_a {#1}{#4}{#2}{#6}{0000}#5{#3}{#6}{}}%
1872 }%
```

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

```
1873 \def\XINT_div_I_a #1#2%
1874 {%
1875   \expandafter\XINT_div_I_b\the\numexpr #1/#2.{#1}{#2}%
1876 }%
1877 \def\XINT_div_I_b #1%
1878 {%
1879   \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1880 }%
```

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»

```
1881 \def\XINT_div_I_czero 0%
1882   \XINT_div_I_c 0.#1#2#3#4#5{\XINT_div_I_g {#5}{#3}}%
1883 \def\XINT_div_I_c #1.#2#3%
1884 {%
1885   \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3.#1.%
1886 }%
```

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

```
1887 \def\XINT_div_I_da #1.%
1888 {%
1889   \ifnum #1>\xint_c_ix
1890     \expandafter\XINT_div_I_dP
1891   \else
1892     \ifnum #1<\xint_c_
1893       \expandafter\expandafter\expandafter\XINT_div_I_dN
1894     \else
1895       \expandafter\expandafter\expandafter\XINT_div_I_db
1896     \fi
1897   \fi
1898 }%
1899 \def\XINT_div_I_dN #1.%
1900 {%
1901   \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i.%
1902 }%
1903 \def\XINT_div_I_db #1.#2#3% #1=q=un chiffre, #2=alpha, #3=B
1904 {%
1905   \expandafter\XINT_div_I_dc\expandafter
1906   {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1907     {\romannumeral0\xintreverseorder{#2}}}%

```

3 Package *xintcore* implementation

```

1908      {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }}%
1909      #1{#2}{#3}%
1910 }%
1911 \def\XINT_div_I_dc #1#2%
1912 {%
1913     \if-#1% s'arranger pour que si n\'egatif on ait renvoy\'e alpha=-.
1914         \expandafter\xint_firstoftwo
1915     \else\expandafter\xint_secondoftwo\fi
1916     {\expandafter\XINT_div_I_dP\the\numexpr #2-\xint_c_i.}%
1917     {\XINT_div_I_e {#1}#2}%
1918 }%

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

1919 \def\XINT_div_I_e #1#2#3#4#5%
1920 {%
1921     \expandafter\XINT_div_I_f \the\numexpr \xint_c_x^iv+#2+#5{#1}%
1922 }%

    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»

1923 \def\XINT_div_I_dP #1.#2#3#4%
1924 {%
1925     \expandafter \XINT_div_I_f \the\numexpr \xint_c_x^iv+#1+#4\expandafter
1926     {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1927         {\romannumeral0\xintreverseorder{#2}}}%
1928     {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }}%
1929 }%

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

1930 \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}

1931 \def\XINT_div_I_g #1#2#3#4#5#6#7#8#9%
1932 {%
1933     \ifnum#3=#4
1934         \expandafter\XINT_div_III_ab
1935     \else
1936         \expandafter\XINT_div_I_h
1937     \fi
1938     {#5}#2.#7.{{#5}{#6}{#4}{#3}}{#8}{#9#1}%
1939 }%

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

1940 \def\XINT_div_III_ab #1#2.#3.#4#5%
1941 {%
1942     \expandafter\XINT_div_III_b
1943     {\romannumeral0\XINT_cuz_loop #2#3\W\W\W\W\W\W\W\Z.%
1944 }%
```

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»

```
1945 \def\XINT_div_I_h #1.#2#3#4#5#6.#7#8%
```

```
1946 {%
```

```
1947 \XINT_div_II_b #1#2#3#4#5.{#8}{#7}{#6}{#8}%
```

```
1948 }%
```

{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

```
1949 \def\XINT_div_II_b #1#2#3#4#5#6#7#8#9%
```

```
1950 {%
```

```
1951 \xint_gob_til_zeros_iv #2#3#4#5\XINT_div_II_skipc 0000%
```

```
1952 \XINT_div_II_c #1{#2#3#4#5}{#6#7#8#9}%
```

```
1953 }%
```

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»

```
1954 \def\XINT_div_II_skipc 0000\XINT_div_II_c #1#2#3#4#5.#6#7%
```

```
1955 {%
```

```
1956 \XINT_div_II_k #7{#4#5}{#6}{0000}%
```

```
1957 }%
```

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

```
1958 \def\XINT_div_II_c #1#2#3#4%
```

```
1959 {%
```

```
1960 \expandafter\XINT_div_II_d\the\numexpr (#3#4+#2)/#1+\xint_c_ixixixix\relax
```

```
1961 {#1}{#2}#3#4%
```

```
1962 }%
```

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»

```
1963 \def\XINT_div_II_d 1#1#2#3#4#5#6#7.#8%
```

```
1964 {%
```

```
1965 \expandafter\XINT_div_II_e
```

```
1966 \romannumeral0\expandafter\XINT_div_sub_xp xp\expandafter
```

```
1967 {\romannumeral0\xintreverseorder{#7}}%
```

```
1968 {\romannumeral0\XINT_mul_Mr {#1#2#3#4}#8\Z\Z\Z }.%
```

```
1969 {#5}{#6}{#8}{#1#2#3#4}%
```

```
1970 }%
```

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

```
1971 \def\XINT_div_II_e #1#2#3#4%
```

```
1972 {%
```

```
1973 \xint_gob_til_zeros_iv #1#2#3#4\XINT_div_II_skipf 0000%
```

```
1974 \XINT_div_II_f #1#2#3#4%
```

```
1975 }%
```

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«c» -> {x'y}x,K,L
(à diminuer de 4), {alpha sur K}B{q1}{alpha'}BQ«c»

1976 \def\XINT_div_II_skipf 0000\XINT_div_II_f 0000#1.#2#3#4#5#6%

1977 {%

1978 \XINT_div_II_k #6{#1}{#4}{#5}%

1979 }%

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

1980 \def\XINT_div_II_f #1#2#3#4#5#6#7#8#9.%

1981 {%

1982 \XINT_div_II_fa {#1#2#3#4#5#6#7#8}{#1#2#3#4#5#6#7#8#9}%

1983 }%

1984 \def\XINT_div_II_fa #1#2#3#4%

1985 {%

1986 \expandafter\XINT_div_II_g\expandafter

1987 {\the\numexpr (#1+#4)/#3-\xint_c_i}{#2}%

1988 }%

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

1989 \def\XINT_div_II_g #1#2#3#4%

1990 {%

1991 \expandafter \XINT_div_II_h

1992 \the\numexpr #4+#1+\xint_c_x^iv\expandafter\expandafter\expandafter

1993 {\expandafter\xint_gobble_iv

1994 \romannumeral0\expandafter\XINT_div_sub_xp_xp\expandafter

1995 {\romannumeral0\xintreverseorder{#2}}%

1996 {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }{#3}%

1997 }%

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«c» -> {x'y}x,K,L à diminuer de 4, {alpha}B{q}, alpha', BQ«c»

1998 \def\XINT_div_II_h 1#1#2#3#4#5#6#7%

1999 {%

2000 \XINT_div_II_k #7{#5}{#6}{#1#2#3#4}%

2001 }%

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

2002 \def\XINT_div_II_k #1#2#3#4#5%

2003 {%

2004 \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_iv.{#3}#1{#2}#5.%

2005 }%

2006 \def\XINT_div_II_l #1.#2#3#4#5#6#7#8#9%

2007 {%

2008 \XINT_div_II_m {{#1}{#2}{{#3}{#4}}{#5}}{#5}{#6#7#8#9}#6#7#8#9%

2009 }%

3 Package *xintcore* implementation

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

```
2010 \def\XINT_div_II_m #1#2#3#4.#5#6%
2011 {%
2012   \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
2013 }%
```

$L, K, A, y, \{x\}, x, \{y\}, \langle c \rangle \rightarrow A.\{yx\}L\{y\}\langle c \rangle$ Comme ici $K=4$, dans la phase I on n'a pas besoin de α , 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 q_0 ici. Enfin, le y est $(x/2)$ pas $((x+1)/2)$ il n'y a pas de $x'=x+1$

```
2014 \def\XINT_div_little_b #1#2#3#4#5#6#7%
2015 {%
2016   \XINT_div_little_c #3.{#4}{#6}{#1}%
2017 }%
```

$\#1\#2\#3\#4=a, \#5=\alpha'=\text{reste de } A.\#6=\{yx\}, \#7=L, \langle c \rangle \rightarrow a, y, x, L, \alpha'=\text{reste de } A, \langle c \rangle.$

```
2018 \def\XINT_div_little_c #1#2#3#4#5.#6#7%
2019 {%
2020   \XINT_div_littleI_a {#1#2#3#4}\#6\#7\#5}%
2021 }%
```

$a, y, x, L, \alpha', \langle c \rangle$ 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.

```
2022 \def\XINT_div_littleI_a #1#2#3%
2023 {%
2024   \expandafter\XINT_div_littleI_b
2025   \the\numexpr (#1+#2)/#3-\xint_c_i{#1}{#2}{#3}%
2026 }%
```

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

```
2027 \def\XINT_div_littleI_b #1%
2028 {%
2029   \xint_gob_til_zero #1\XINT_div_littleI_skip 0\XINT_div_littleI_c #1%
2030 }%
2031 \def\XINT_div_littleI_skip 0\XINT_div_littleI_c 0#1#2#3#4#5%
2032   {\XINT_div_littleII_a {#4}{#1}\#5.{#2}{#3}\{0000\}}%
2033 \def\XINT_div_littleI_c #1#2#3#4%
2034 {%
2035   \expandafter\expandafter\expandafter\XINT_div_littleI_e
2036   \expandafter\expandafter\expandafter
2037   {\expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+\#2-\#1*\#4\#1\{#3\}{#4}\}%
2038 }%
```


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

```
2039 \def\XINT_div_littleI_e #1#2#3#4#5%
2040   {\XINT_div_littleII_a {#4}{#1}#5.{#3}{000#2}}%
```

L{alpha}alpha'.{yx}Q«c» et c'est là qu'on boucle

```
2041 \def\XINT_div_littleII_a #1%
2042 {%
2043   \ifnum#1=\xint_c_iv
2044     \expandafter\XINT_div_littleIII_ab
2045   \else
2046     \expandafter\XINT_div_littleII_b
2047   \fi {#1}%
2048 }%
```

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

```
2049 \def\XINT_div_littleIII_ab #1#2#3.#4%
2050 {%
2051   \expandafter\XINT_div_III_b\the\numexpr #2#3.%
2052 }%
```

L{alpha}alpha'.{yx}Q«c». On diminue L de quatre, comme cela c'est fait.

```
2053 \def\XINT_div_littleII_b #1%
2054 {%
2055   \expandafter\XINT_div_littleII_c\expandafter {\the\numexpr #1-\xint_c_iv}%
2056 }%
```

{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.

```
2057 \def\XINT_div_littleII_c #1#2#3#4#5#6#7.#8%
2058 {%
2059   \XINT_div_littleII_d {#2#3#4#5#6}#8{#1}{#7}%
2060 }%
2061 \def\XINT_div_littleII_d #1#2#3%
2062 {%
2063   \expandafter\XINT_div_littleII_e\the\numexpr (#1+#2)/#3+\xint_c_ixixixix.%
2064   {#1}{#2}{#3}%
2065 }%
```

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»

```
2066 \def\XINT_div_littleII_e 1#1.#2#3#4%
2067 {%
2068   \expandafter\expandafter\expandafter\XINT_div_littleII_f
2069   \expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4.%
2070   {#1}{#3}{#4}%
2071 }%
```

3 Package *xintcore* implementation

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

```
2072 \def\XINT_div_littleII_f #1.#2#3#4#5#6%
2073 {%
2074   \XINT_div_littleII_a {#4}{#1}#5.{#3}{#6#2}%
2075 }%
```

La soustraction spéciale. Dans 1.09j, elle fait $A - qB$, pour A (en fait α 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 α toujours à l'envers afin d'éviter le reverse order systématique fait sur A (ou plutôt α), 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.

```
2076 \def\XINT_div_sub_xpxp #1#2% #1= $\alpha$  d\'ej\`a renvers\'e, #2 se d\'eveloppe en qB
2077 {%
2078   \expandafter\XINT_div_sub_xpxp_b #2\W\X\Y\Z #1\W\X\Y\Z
2079 }%
2080 \def\XINT_div_sub_xpxp_b
2081 {%
2082   \XINT_div_sub_A 1{%
2083 }%
2084 \def\XINT_div_sub_A #1#2#3#4#5#6%
2085 {%
2086   \xint_gob_til_W #3\xint_div_sub_az\W
2087   \XINT_div_sub_B #1{#3#4#5#6}{#2}%
2088 }%
2089 \def\XINT_div_sub_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
2090 {%
2091   \xint_gob_til_W #5\xint_div_sub_bz\W
2092   \XINT_div_sub_onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
2093 }%
2094 \def\XINT_div_sub_onestep #1#2#3#4#5#6%
2095 {%
2096   \expandafter\XINT_div_sub_backtoA
2097   \the\numexpr 11#6-#5#4#3#2+#1-\xint_c_i.%
2098 }%
2099 \def\XINT_div_sub_backtoA #1#2#3.#4%
2100 {%
2101   \XINT_div_sub_A #2{#3#4}%
2102 }%
```

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 -

```
2103 \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.

```
2104 \def\xint_div_sub_az\W\XINT_div_sub_B #1#2{\XINT_div_sub_C #1}%
2105 \def\XINT_div_sub_C #1#2#3#4#5#6%
2106 {%
2107   \xint_gob_til_W #3\xint_div_sub_cz\W
2108   \XINT_div_sub_C_onestep #1{#6#5#4#3}{#2}%
2109 }%
2110 \def\XINT_div_sub_C_onestep #1#2%
2111 {%
2112   \expandafter\XINT_div_sub_backtoC \the\numexpr 11#2+#1-\xint_c_i.%
2113 }%
2114 \def\XINT_div_sub_backtoC #1#2#3.#4%
2115 {%
2116   \XINT_div_sub_C #2{#3#4}%
2117 }%
```

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 -

```
2118 \def\xint_div_sub_cz\W\XINT_div_sub_C_onestep #1#2%
2119 {%
2120   \if#10% retenue
2121     \expandafter\xint_div_sub_neg
2122   \else\expandafter\xint_div_sub_ok
2123   \fi
2124 }%
2125 \def\xint_div_sub_neg #1{ -}%
2126 \def\xint_div_sub_ok #1{ #1}%
```

3.25 *\xintiDivRound*, *\xintiiDivRound*

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

```
2127 \def\xintiDivRound {\romannumeral0\xintidivround}%
2128 \def\xintidivround #1{\expandafter\XINT_iidivround\romannumeral0\xintnum{#1}\Z}%
2129 \def\xintiiDivRound {\romannumeral0\xintiidivround}%
2130 \def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral-`0#1\Z}%
2131 \def\XINT_iidivround #1#2\Z #3{\expandafter\XINT_iidivround_a\expandafter #1%
2132   \romannumeral-`0#3\Z #2\Z}%
2133 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
2134 {%
2135   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
2136   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
2137   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
2138   \xint_orthat{\XINT_iidivround_bpos #1#2}%
2139 }%
2140 \def\XINT_iidivround_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space 0}%
```

```

2141 \def\XINT_iidivround_aiszero #1\Z #2\Z { 0}%
2142 \def\XINT_iidivround_bpos #1%
2143 {%
2144   \xint_UDsignfork
2145     #1{\xintiiopp\XINT_iidivround_pos {}}%
2146     -{\XINT_iidivround_pos #1}%
2147   \krof
2148 }%
2149 \def\XINT_iidivround_bneg #1%
2150 {%
2151   \xint_UDsignfork
2152     #1{\XINT_iidivround_pos {}}%
2153     -{\xintiiopp\XINT_iidivround_pos #1}%
2154   \krof
2155 }%
2156 \def\XINT_iidivround_pos #1#2\Z #3\Z{\expandafter\XINT_iidivround_pos_a
2157   \romannumeral0\XINT_div_prepare {#2}{#1#30}}%
2158 \def\XINT_iidivround_pos_a #1#2{\xintReverseOrder {#1\XINT_iidivround_pos_b}\Z }%
2159 \def\XINT_iidivround_pos_b #1#2{\xint_gob_til_Z #2\XINT_iidivround_pos_small\Z
2160   \XINT_iidivround_pos_c #1#2}%
2161 \def\XINT_iidivround_pos_c #1#2\Z {\ifnum #1>\xint_c_iv
2162   \expandafter\XINT_iidivround_pos_up
2163   \else \expandafter\xintreverseorder
2164   \fi {#2}}%
2165 \def\XINT_iidivround_pos_up #1{\xintinc {\xintReverseOrder{#1}}}%
2166 \def\XINT_iidivround_pos_small\Z\XINT_iidivround_pos_c #1#2%
2167   {\ifnum #1>\xint_c_iv\expandafter\xint_secondoftwo\else\expandafter
2168     \xint_firstoftwo\fi { 0}{ 1}}%

```

3.26 \xintiDivTrunc, \xintiiDivTrunc

```

2169 \def\xintiDivTrunc {\romannumeral0\xintidivtrunc }%
2170 \def\xintidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
2171 \def\xintiiDivTrunc {\romannumeral0\xintiidivtrunc }%
2172 \def\xintiidivtrunc #1{\expandafter\XINT_iidivtrunc \romannumeral-`0#1\Z }%
2173 \def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
2174   \romannumeral-`0#3\Z #2\Z }%
2175 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
2176 {%
2177   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
2178   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
2179   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
2180   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
2181 }%
2182 \def\XINT_iidivtrunc_bpos #1%
2183 {%
2184   \xint_UDsignfork
2185     #1{\xintiiopp\XINT_iidivtrunc_pos {}}%
2186     -{\XINT_iidivtrunc_pos #1}%
2187   \krof
2188 }%
2189 \def\XINT_iidivtrunc_bneg #1%
2190 {%

```

```

2191 \xint_UDsignfork
2192 #1{\XINT_iidivtrunc_pos {}}%
2193 -{\xintiioopp\XINT_iidivtrunc_pos #1}%
2194 \krof
2195 }%
2196 \def\XINT_iidivtrunc_pos #1#2\Z #3\Z%
2197 {\expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

3.27 `\xintiMod`, `\xintiiMod`

```

2198 \def\xintiMod {\romannumeral0\xintimod }%
2199 \def\xintimod #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
2200 \def\xintiiMod {\romannumeral0\xintiimod }%
2201 \def\xintiimod #1{\expandafter\XINT_iimod \romannumeral-`0#1\Z }%
2202 \def\XINT_iimod #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
2203 \romannumeral-`0#3\Z #2\Z }%
2204 \def\XINT_iimod_a #1#2# #1 de A, #2 de B.
2205 {%
2206 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
2207 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
2208 \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
2209 \xint_orthat{\XINT_iimod_bpos #1#2}%
2210 }%
2211 \def\XINT_iimod_bpos #1%
2212 {%
2213 \xint_UDsignfork
2214 #1{\xintiioopp\XINT_iimod_pos {}}%
2215 -{\XINT_iimod_pos #1}%
2216 \krof
2217 }%
2218 \def\XINT_iimod_bneg #1%
2219 {%
2220 \xint_UDsignfork
2221 #1{\xintiioopp\XINT_iimod_pos {}}%
2222 -{\XINT_iimod_pos #1}%
2223 \krof
2224 }%
2225 \def\XINT_iimod_pos #1#2\Z #3\Z%
2226 {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare {#2}{#1#3}}%
2227 \XINT_restorecatcodes_endinput%

```

4 Package `xint` implementation

.1	Package identification	87	.10	<code>\xintifOne</code> , <code>\xintiiifOne</code>	90
.2	More token management	87	.11	<code>\xintifTrueAelseB</code> , <code>\xintifFalseAelseB</code>	90
.3	<code>\xintSgnFork</code>	87	.12	<code>\xintifCmp</code> , <code>\xintiiifCmp</code>	91
.4	<code>\xintIsOne</code> , <code>\xintiiIsOne</code>	87	.13	<code>\xintifEq</code> , <code>\xintiiifEq</code>	91
.5	<code>\xintRev</code>	88	.14	<code>\xintifGt</code> , <code>\xintiiifGt</code>	91
.6	<code>\xintLen</code>	88	.15	<code>\xintifLt</code> , <code>\xintiiifLt</code>	92
.7	<code>\xintBool</code> , <code>\xintToggle</code>	89	.16	<code>\xintifOdd</code> , <code>\xintiiifOdd</code>	92
.8	<code>\xintifSgn</code> , <code>\xintiiifSgn</code>	89	.17	<code>\xintCmp</code> , <code>\xintiiCmp</code>	93
.9	<code>\xintifZero</code> , <code>\xintifNotZero</code> , <code>\xintiiifZero</code> , <code>\xintiiifNotZero</code>	89	.18	<code>\xintEq</code> , <code>\xintGt</code> , <code>\xintLt</code>	95
			.19	<code>\xintNeq</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code>	95

.20	<code>\xintiiEq, \xintiiGt, \xintiiLt</code>	96	.34	<code>\xintiiPrd</code>	103
.21	<code>\xintiiNeq, \xintiiGtorEq, \xintiiLtorEq</code>	96	.35	<code>\xintFac</code>	104
.22	<code>\xintIsZero, \xintIsNotZero, \xintii- IsZero, \xintiiIsNotZero</code>	96	.36	<code>\xintMON, \xintMMON, \xintiiMON, \xinti- imMON</code>	106
.23	<code>\xintIsTrue, \xintNot, \xintIsFalse</code>	96	.37	<code>\xintOdd, \xintiiOdd, \xintEven, \xinti- iEven</code>	107
.24	<code>\xintAND, \xintOR, \xintXOR</code>	96	.38	<code>\xintDSL</code>	108
.25	<code>\xintANDof</code>	97	.39	<code>\xintDSR</code>	108
.26	<code>\xintORof</code>	97	.40	<code>\xintDSH, \xintDSHr</code>	109
.27	<code>\xintXORof</code>	97	.41	<code>\xintDSx</code>	110
.28	<code>\xintGeq</code>	97	.42	<code>\xintDecSplit, \xintDecSplitL, \xintDec- SplitR</code>	112
.29	<code>\xintiMax, \xintiiMax</code>	99	.43	<code>\xintiiSqrt, \xintiiSqrtR, \xintiiSquare- Root</code>	116
.30	<code>\xintMaxof</code>	100	.44	<code>\xintiiE</code>	119
.31	<code>\xintiMin, \xintiiMin</code>	101			
.32	<code>\xintiMinof</code>	102			
.33	<code>\xintiiSum</code>	102			

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 [2015/09/12 v1.1c 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\xintsngfork }%
54 \def\xintsngfork #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 `\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.

```

77 \def\xintRev {\romannumeral0\xintrev }%
78 \def\xintrev #1%
79 {%
80   \expandafter\XINT_rev_fork
81   \romannumeral-`0#1\xint_relax % empty #1 ok, \xint_relax stops expansion
82   \xint_bye\xint_bye\xint_bye\xint_bye
83   \xint_bye\xint_bye\xint_bye\xint_bye
84   \xint_relax
85 }%
86 \def\XINT_rev_fork #1%
87 {%
88   \xint_UDsignfork
89   #1{\expandafter\xint_minus_thenstop\romannumeral0\XINT_rord_main {}}%
90   -{\XINT_rord_main {}}#1}%
91   \krof
92 }%

```

4.6 `\xintLen`

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

```

93 \def\xintLen {\romannumeral0\xintlen }%
94 \def\xintlen #1%
95 {%
96   \expandafter\XINT_len_fork
97   \romannumeral0\xintnum{#1}\xint_relax\xint_relax\xint_relax\xint_relax
98   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
99 }%
100 \def\XINT_Len #1% variant which does not expand via \xintnum.
101 {%
102   \romannumeral0\XINT_len_fork
103   #1\xint_relax\xint_relax\xint_relax\xint_relax
104   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
105 }%
106 \def\XINT_len_fork #1%
107 {%
108   \expandafter\XINT_length_loop
109   \xint_UDsignfork
110   #1{0.}%
111   -{0.#1}%
112   \krof

```



```
113 }%
```

4.7 `\xintBool`, `\xintToggle`

1.09c

```
114 \def\xintBool #1{\romannumeral-`0%
115         \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
116 \def\xintToggle #1{\romannumeral-`0\iftoggle{#1}{1}{0}}%
```

4.8 `\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`)

```
117 \def\xintifSgn {\romannumeral0\xintifsgn }%
118 \def\xintifsgn #1%
119 {%
120     \ifcase \xintSgn{#1}
121         \expandafter\xint_secondofthree_thenstop
122         \or\expandafter\xint_thirdofthree_thenstop
123         \else\expandafter\xint_firstofthree_thenstop
124     \fi
125 }%
126 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
127 \def\xintiiifsgn #1%
128 {%
129     \ifcase \xintiiSgn{#1}
130         \expandafter\xint_secondofthree_thenstop
131         \or\expandafter\xint_thirdofthree_thenstop
132         \else\expandafter\xint_firstofthree_thenstop
133     \fi
134 }%
```

4.9 `\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.

```
135 \def\xintifZero {\romannumeral0\xintifzero }%
136 \def\xintifzero #1%
137 {%
138     \if0\xintSgn{#1}%
139         \expandafter\xint_firstoftwo_thenstop
140     \else
141         \expandafter\xint_secondoftwo_thenstop
142     \fi
143 }%
144 \def\xintifNotZero {\romannumeral0\xintifnotzero }%
```

```

145 \def\xintifnotzero #1%
146 {%
147   \if0\xintSgn{#1}%
148     \expandafter\xint_secondoftwo_thenstop
149   \else
150     \expandafter\xint_firstoftwo_thenstop
151   \fi
152 }%
153 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
154 \def\xintiiifzero #1%
155 {%
156   \if0\xintiiSgn{#1}%
157     \expandafter\xint_firstoftwo_thenstop
158   \else
159     \expandafter\xint_secondoftwo_thenstop
160   \fi
161 }%
162 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
163 \def\xintiiifnotzero #1%
164 {%
165   \if0\xintiiSgn{#1}%
166     \expandafter\xint_secondoftwo_thenstop
167   \else
168     \expandafter\xint_firstoftwo_thenstop
169   \fi
170 }%

```

4.10 `\xintifOne`, `\xintiiifOne`

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

```

171 \def\xintiiifOne {\romannumeral0\xintiiifone }%
172 \def\xintiiifone #1%
173 {%
174   \if1\xintiiIsOne{#1}%
175     \expandafter\xint_firstoftwo_thenstop
176   \else
177     \expandafter\xint_secondoftwo_thenstop
178   \fi
179 }%
180 \def\xintifOne {\romannumeral0\xintifone }%
181 \def\xintifone #1%
182 {%
183   \if1\xintIsOne{#1}%
184     \expandafter\xint_firstoftwo_thenstop
185   \else
186     \expandafter\xint_secondoftwo_thenstop
187   \fi
188 }%

```

4.11 `\xintifTrueAelseB`, `\xintifFalseAelseB`

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

```
189 \let\xintifTrueAelseB\xintifNotZero
190 \let\xintifFalseAelseB\xintifZero
191 \let\xintifTrue\xintifNotZero
192 \let\xintifTrueFalse\xintifNotZero
```

4.12 `\xintifCmp`, `\xintiiifCmp`

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

```
193 \def\xintifCmp {\romannumeral0\xintifcmp }%
194 \def\xintifcmp #1#2%
195 {%
196     \ifcase\xintCmp {#1}{#2}
197         \expandafter\xint_secondofthree_thenstop
198         \or\expandafter\xint_thirdofthree_thenstop
199         \else\expandafter\xint_firstofthree_thenstop
200     \fi
201 }%
202 \def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
203 \def\xintiiifcmp #1#2%
204 {%
205     \ifcase\xintiiCmp {#1}{#2}
206         \expandafter\xint_secondofthree_thenstop
207         \or\expandafter\xint_thirdofthree_thenstop
208         \else\expandafter\xint_firstofthree_thenstop
209     \fi
210 }%
```

4.13 `\xintifEq`, `\xintiiifEq`

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

```
211 \def\xintifEq {\romannumeral0\xintifeq }%
212 \def\xintifeq #1#2%
213 {%
214     \if0\xintCmp{#1}{#2}%
215         \expandafter\xint_firstoftwo_thenstop
216         \else\expandafter\xint_secondoftwo_thenstop
217     \fi
218 }%
219 \def\xintiiifEq {\romannumeral0\xintiiifeq }%
220 \def\xintiiifeq #1#2%
221 {%
222     \if0\xintiiCmp{#1}{#2}%
223         \expandafter\xint_firstoftwo_thenstop
224         \else\expandafter\xint_secondoftwo_thenstop
225     \fi
226 }%
```

4.14 `\xintifGt`, `\xintiiifGt`

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

```

227 \def\xintifGt {\romannumeral0\xintifgt }%
228 \def\xintifgt #1#2%
229 {%
230     \if1\xintCmp{#1}{#2}%
231         \expandafter\xint_firstoftwo_thenstop
232     \else\expandafter\xint_secondoftwo_thenstop
233     \fi
234 }%
235 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
236 \def\xintiiifgt #1#2%
237 {%
238     \if1\xintiiCmp{#1}{#2}%
239         \expandafter\xint_firstoftwo_thenstop
240     \else\expandafter\xint_secondoftwo_thenstop
241     \fi
242 }%

```

4.15 \xintifLt, \xintiiifLt

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

```

243 \def\xintifLt {\romannumeral0\xintiflt }%
244 \def\xintiflt #1#2%
245 {%
246     \ifnum\xintCmp{#1}{#2}<\xint_c_
247         \expandafter\xint_firstoftwo_thenstop
248     \else \expandafter\xint_secondoftwo_thenstop
249     \fi
250 }%
251 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
252 \def\xintiiiflt #1#2%
253 {%
254     \ifnum\xintiiCmp{#1}{#2}<\xint_c_
255         \expandafter\xint_firstoftwo_thenstop
256     \else \expandafter\xint_secondoftwo_thenstop
257     \fi
258 }%

```

4.16 \xintifOdd, \xintiiifOdd

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

```

259 \def\xintiiifOdd {\romannumeral0\xintiiifodd }%
260 \def\xintiiifodd #1%
261 {%
262     \if\xintiiOdd{#1}1%
263         \expandafter\xint_firstoftwo_thenstop
264     \else
265         \expandafter\xint_secondoftwo_thenstop
266     \fi
267 }%
268 \def\xintifOdd {\romannumeral0\xintifodd }%
269 \def\xintifodd #1%

```

```

270 {%
271   \if\xintOdd{#1}1%
272     \expandafter\xint_firstoftwo_thenstop
273   \else
274     \expandafter\xint_secondoftwo_thenstop
275   \fi
276 }%

```

4.17 *\xintCmp*, *\xintiCmp*

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*)

```

277 \def\xintCmp {\romannumeral0\xintcmp }%
278 \def\xintcmp #1%
279 {%
280   \expandafter\xint_cmp\expandafter{\romannumeral0\xintnum{#1}}%
281 }%
282 \def\xint_cmp #1#2%
283 {%
284   \expandafter\XINT_cmp_fork \romannumeral0\xintnum{#2}\Z #1\Z
285 }%
286 \def\xintiiCmp {\romannumeral0\xintiicmp }%
287 \def\xintiicmp #1%
288 {%
289   \expandafter\xint_iicmp\expandafter{\romannumeral-`0#1}%
290 }%
291 \def\xint_iicmp #1#2%
292 {%
293   \expandafter\XINT_cmp_fork \romannumeral-`0#2\Z #1\Z
294 }%
295 \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*

```

296 \def\XINT_cmp_fork #1#2\Z #3#4\Z
297 {%
298   \xint_UDsignsfork
299     #1#3\XINT_cmp_minusminus
300     #1-\XINT_cmp_minusplus
301     #3-\XINT_cmp_plusminus
302     --{\xint_UDzerosfork
303       #1#3\XINT_cmp_zerozero
304       #10\XINT_cmp_zeroplus
305       #30\XINT_cmp_pluszero
306       00\XINT_cmp_plusplus
307     \krof }%
308   \krof
309   {#2}{#4}#1#3%
310 }%

```

4 Package *xint* implementation

```

311 \def\XINT_cmp_minusplus #1#2#3#4{ 1}%
312 \def\XINT_cmp_plusminus #1#2#3#4{ -1}%
313 \def\XINT_cmp_zerozero #1#2#3#4{ 0}%
314 \def\XINT_cmp_zeroplus #1#2#3#4{ 1}%
315 \def\XINT_cmp_pluszero #1#2#3#4{ -1}%
316 \def\XINT_cmp_plusplus #1#2#3#4%
317 {%
318     \XINT_cmp_pre {#4#2}{#3#1}%
319 }%
320 \def\XINT_cmp_minusminus #1#2#3#4%
321 {%
322     \XINT_cmp_pre {#1}{#2}%
323 }%
324 \def\XINT_cmp_pre #1%
325 {%
326     \expandafter\XINT_cmp_pre_b\expandafter
327     {\romannumeral0\XINT_RQ {}}#1\R\R\R\R\R\R\R\R\Z }%
328 }%
329 \def\XINT_cmp_pre_b #1#2%
330 {%
331     \expandafter\XINT_cmp_A
332     \expandafter1\expandafter{\expandafter}%
333     \romannumeral0\XINT_RQ {}}#2\R\R\R\R\R\R\R\R\Z
334     \W\X\Y\Z #1\W\X\Y\Z
335 }%

```

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

```

336 \def\XINT_cmp_A #1#2#3\W\X\Y\Z #4#5#6#7%
337 {%
338     \xint_gob_til_W #4\xint_cmp_az\W
339     \XINT_cmp_B #1{#4#5#6#7}{#2}{#3}\W\X\Y\Z
340 }%
341 \def\XINT_cmp_B #1#2#3#4#5#6#7%
342 {%
343     \xint_gob_til_W#4\xint_cmp_bz\W
344     \XINT_cmp_onestep #1#2{#7#6#5#4}{#3}%
345 }%
346 \def\XINT_cmp_onestep #1#2#3#4#5#6%
347 {%
348     \expandafter\XINT_cmp_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
349 }%
350 \def\XINT_cmp_backtoA #1#2#3.#4%
351 {%
352     \XINT_cmp_A #2{#3#4}%
353 }%
354 \def\xint_cmp_bz\W\XINT_cmp_onestep #1\Z { 1}%
355 \def\xint_cmp_az\W\XINT_cmp_B #1#2#3#4#5#6#7%
356 {%

```

4 Package `xint` implementation

```
357 \xint_gob_til_W #4\xint_cmp_ez\W
358 \XINT_cmp_Eenter #1{#3}#4#5#6#7%
359 }%
360 \def\XINT_cmp_Eenter #1\Z { -1}%
361 \def\xint_cmp_ez\W\XINT_cmp_Eenter #1%
362 {%
363 \xint_UDzerofork
364 #1\XINT_cmp_K % il y a une retenue
365 0\XINT_cmp_L % pas de retenue
366 \krof
367 }%
368 \def\XINT_cmp_K #1\Z { -1}%
369 \def\XINT_cmp_L #1{\XINT_OneIfPositive_main #1}%
370 \def\XINT_OneIfPositive #1%
371 {%
372 \XINT_OneIfPositive_main #1\W\X\Y\Z%
373 }%
374 \def\XINT_OneIfPositive_main #1#2#3#4%
375 {%
376 \xint_gob_til_Z #4\xint_OneIfPositive_terminated\Z
377 \XINT_OneIfPositive_onestep #1#2#3#4%
378 }%
379 \def\xint_OneIfPositive_terminated\Z\XINT_OneIfPositive_onestep\W\X\Y\Z { 0}%
380 \def\XINT_OneIfPositive_onestep #1#2#3#4%
381 {%
382 \expandafter\XINT_OneIfPositive_check\the\numexpr #1#2#3#4\relax
383 }%
384 \def\XINT_OneIfPositive_check #1%
385 {%
386 \xint_gob_til_zero #1\xint_OneIfPositive_backtomain 0%
387 \XINT_OneIfPositive_finish #1%
388 }%
389 \def\XINT_OneIfPositive_finish #1\W\X\Y\Z{ 1}%
390 \def\xint_OneIfPositive_backtomain 0\XINT_OneIfPositive_finish 0%
391 {\XINT_OneIfPositive_main }%
```

4.18 `\xintEq`, `\xintGt`, `\xintLt`

1.09a.

```
392 \def\xintEq {\romannumeral0\xinteq }\def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
393 \def\xintGt {\romannumeral0\xintgt }\def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
394 \def\xintLt {\romannumeral0\xintlt }\def\xintlt #1#2{\xintiflt{#1}{#2}{1}{0}}%
```

4.19 `\xintNeq`, `\xintGtorEq`, `\xintLtorEq`

1.1. Pour `xintexpr`. No lowercase macros

```
395 \def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
396 \def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
397 \def\xintNeq #1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
```

4.20 `\xintiiEq`, `\xintiiGt`, `\xintiiLt`

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

```
398 \def\xintiiEq #1#2{\romannumeral0\xintiiifeq{#1}{#2}{1}{0}}%
399 \def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%
400 \def\xintiiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%
```

4.21 `\xintiiNeq`, `\xintiiGtorEq`, `\xintiiLtorEq`

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

```
401 \def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
402 \def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
403 \def\xintiiNeq #1#2{\romannumeral0\xintiiifeq {#1}{#2}{0}{1}}%
```

4.22 `\xintIsZero`, `\xintIsNotZero`, `\xintiiIsZero`, `\xintiiIsNotZero`

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

```
404 \def\xintIsZero {\romannumeral0\xintiszero }%
405 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
406 \def\xintIsNotZero {\romannumeral0\xintisnotzero }%
407 \def\xintisnotzero
408 #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
409 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
410 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
411 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
412 \def\xintiiisnotzero
413 #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%

```

4.23 `\xintIsTrue`, `\xintNot`, `\xintIsFalse`

1.09c

```
414 \let\xintIsTrue\xintIsNotZero
415 \let\xintNot\xintIsZero
416 \let\xintIsFalse\xintIsZero
```

4.24 `\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

```
417 \def\xintAND {\romannumeral0\xintand }%
418 \def\xintand #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
419 \else\expandafter\xint_secondoftwo\fi
420 { 0}{\xintisnotzero{#2}}}%
421 \def\xintOR {\romannumeral0\xintor }%
422 \def\xintor #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
423 \else\expandafter\xint_secondoftwo\fi
424 {\xintisnotzero{#2}}{ 1}}%
425 \def\xintXOR {\romannumeral0\xintxor }%
426 \def\xintxor #1#2{\if\xintIsZero{#1}\xintIsZero{#2}%
427 \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```


4.25 `\xintANDof`

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

```

428 \def\xintANDof      {\romannumeral0\xintandof }%
429 \def\xintandof      #1{\expandafter\XINT_andof_a\romannumeral-`0#1\relax }%
430 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral-`0#1\Z }%
431 \def\XINT_andof_b #1%
432      {\xint_gob_til_relax #1\XINT_andof_e\relax\XINT_andof_c #1}%
433 \def\XINT_andof_c #1\Z
434      {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
435 \def\XINT_andof_no #1\relax { 0}%
436 \def\XINT_andof_e #1\Z { 1}%

```

4.26 `\xintORof`

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

```

437 \def\xintORof      {\romannumeral0\xintorof }%
438 \def\xintorof      #1{\expandafter\XINT_orof_a\romannumeral-`0#1\relax }%
439 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral-`0#1\Z }%
440 \def\XINT_orof_b #1%
441      {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
442 \def\XINT_orof_c #1\Z
443      {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
444 \def\XINT_orof_yes #1\relax { 1}%
445 \def\XINT_orof_e #1\Z { 0}%

```

4.27 `\xintXORof`

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

```

446 \def\xintXORof      {\romannumeral0\xintxorof }%
447 \def\xintxorof      #1{\expandafter\XINT_xorof_a\expandafter
448      0\romannumeral-`0#1\relax }%
449 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral-`0#2\Z #1}%
450 \def\XINT_xorof_b #1%
451      {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
452 \def\XINT_xorof_c #1\Z #2%
453      {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
454      \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
455      {\XINT_xorof_a #2}%
456      }%
457 \def\XINT_xorof_e #1\Z #2{ #2}%

```

4.28 `\xintGeq`

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

```

458 \def\xintGeq {\romannumeral0\xintgeq }%
459 \def\xintgeq #1%
460 {%

```

4 Package *xint* implementation

```

461 \expandafter\xint_geq\expandafter {\romannumeral0\xintnum{#1}}%
462 }%
463 \def\xint_geq #1#2%
464 {%
465 \expandafter\XINT_geq_fork \romannumeral0\xintnum{#2}\Z #1\Z
466 }%
467 \def\XINT_Geq #1#2{\romannumeral0\XINT_geq_fork #2\Z #1\Z }%

```

PLUS GRAND OU ÉGAL ATTENTION, TESTE les VALEURS ABSOLUES

```

468 \def\XINT_geq_fork #1#2\Z #3#4\Z
469 {%
470 \xint_UDzerofork
471 #1\XINT_geq_secondiszero % |#1#2|=0
472 #3\XINT_geq_firstiszero % |#1#2|>0
473 0{\xint_UDsignsfork
474 #1#3\XINT_geq_minusminus
475 #1-\XINT_geq_minusplus
476 #3-\XINT_geq_plusminus
477 --\XINT_geq_plusplus
478 \krof }%
479 \krof
480 {#2}{#4}#1#3%
481 }%
482 \def\XINT_geq_secondiszero #1#2#3#4{ 1}%
483 \def\XINT_geq_firstiszero #1#2#3#4{ 0}%
484 \def\XINT_geq_plusplus #1#2#3#4{\XINT_geq_pre {#4#2}{#3#1}}%
485 \def\XINT_geq_minusminus #1#2#3#4{\XINT_geq_pre {#2}{#1}}%
486 \def\XINT_geq_minusplus #1#2#3#4{\XINT_geq_pre {#4#2}{#1}}%
487 \def\XINT_geq_plusminus #1#2#3#4{\XINT_geq_pre {#2}{#3#1}}%
488 \def\XINT_geq_pre #1%
489 {%
490 \expandafter\XINT_geq_pre_b\expandafter
491 {\romannumeral0\XINT_RQ {#1}\R\R\R\R\R\R\R\R\Z }%
492 }%
493 \def\XINT_geq_pre_b #1#2%
494 {%
495 \expandafter\XINT_geq_A
496 \expandafter1\expandafter{\expandafter}%
497 \romannumeral0\XINT_RQ {#2}\R\R\R\R\R\R\R\R\Z
498 \W\X\Y\Z #1 \W\X\Y\Z
499 }%

```

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 MULTIPLES 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

```

500 \def\XINT_geq_A #1#2#3\W\X\Y\Z #4#5#6#7%
501 {%
502 \xint_gob_til_W #4\xint_geq_az\W
503 \XINT_geq_B #1{#4#5#6#7}{#2}{#3}\W\X\Y\Z

```

4 Package `xint` implementation

```
504 }%
505 \def\XINT_geq_B #1#2#3#4#5#6#7%
506 {%
507   \xint_gob_til_W #4\xint_geq_bz\W
508   \XINT_geq_onestep #1#2{#7#6#5#4}{#3}%
509 }%
510 \def\XINT_geq_onestep #1#2#3#4#5#6%
511 {%
512   \expandafter\XINT_geq_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
513 }%
514 \def\XINT_geq_backtoA #1#2#3.#4%
515 {%
516   \XINT_geq_A #2{#3#4}%
517 }%
518 \def\xint_geq_bz\W\XINT_geq_onestep #1\W\X\Y\Z { 1}%
519 \def\xint_geq_az\W\XINT_geq_B #1#2#3#4#5#6#7%
520 {%
521   \xint_gob_til_W #4\xint_geq_ez\W
522   \XINT_geq_Enter #1%
523 }%
524 \def\XINT_geq_Enter #1\W\X\Y\Z { 0}%
525 \def\xint_geq_ez\W\XINT_geq_Enter #1%
526 {%
527   \xint_UDzerofork
528   #1{ 0}          %      il y a une retenue
529   0{ 1}          %      pas de retenue
530   \krof
531 }%
```

4.29 `\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`.

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

```
532 \def\xintiMax {\romannumeral0\xintimax }%
533 \def\xintimax #1%
534 {%
535   \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
536 }%
537 \def\xint_max #1#2%
538 {%
539   \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
540 }%
541 \def\xintiiMax {\romannumeral0\xintiiimax }%
542 \def\xintiiimax #1%
543 {%
544   \expandafter\xint_iiimax\expandafter {\romannumeral-`0#1}%
545 }%
546 \def\xint_iiimax #1#2%
547 {%
```

4 Package `xint` implementation

```

548 \expandafter\XINT_max_pre\expandafter {\romannumeral-`0#2}{#1}%
549 }%
550 \let\xintMax\xintiMax \let\xintmax\xintimax % deprecated, should be only with xintfrac
551 \def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
552 \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*

553 \def\XINT_max_fork #1#2\Z #3#4\Z
554 {%
555 \xint_UDsignsfork
556 #1#3\XINT_max_minusminus % A < 0, B < 0
557 #1-\XINT_max_minusplus % B < 0, A >= 0
558 #3-\XINT_max_plusminus % A < 0, B >= 0
559 --{\xint_UDzerosfork
560 #1#3\XINT_max_zerozero % A = B = 0
561 #10\XINT_max_zeroplus % B = 0, A > 0
562 #30\XINT_max_pluszero % A = 0, B > 0
563 00\XINT_max_plusplus % A, B > 0
564 \krof }%
565 \krof
566 {#2}{#4}#1#3%
567 }%

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

568 \def\XINT_max_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
569 \def\XINT_max_zeroplus #1#2#3#4{\xint_firstoftwo_thenstop }%
570 \def\XINT_max_pluszero #1#2#3#4{\xint_secondoftwo_thenstop }%
571 \def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
572 \def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
573 \def\XINT_max_plusplus #1#2#3#4%
574 {%
575 \ifodd\XINT_Geq {#4#2}{#3#1}
576 \expandafter\xint_firstoftwo_thenstop
577 \else
578 \expandafter\xint_secondoftwo_thenstop
579 \fi
580 }%

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

581 \def\XINT_max_minusminus #1#2#3#4%
582 {%
583 \ifodd\XINT_Geq {#1}{#2}
584 \expandafter\xint_firstoftwo_thenstop
585 \else
586 \expandafter\xint_secondoftwo_thenstop
587 \fi
588 }%

```

4.30 `\xintMaxof`

New with 1.09a.

4 Package `xint` implementation

```

589 \def\xintiMaxof      {\romannumeral0\xintimaxof }%
590 \def\xintimaxof      #1{\expandafter\XINT_imaxof_a\romannumeral-`0#1\relax }%
591 \def\XINT_imaxof_a    #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
592 \def\XINT_imaxof_b    #1\Z #2%
593       {\expandafter\XINT_imaxof_c\romannumeral-`0#2\Z {#1}\Z}%
594 \def\XINT_imaxof_c    #1%
595       {\xint_gob_til_relax #1\XINT_imaxof_e\relax\XINT_imaxof_d #1}%
596 \def\XINT_imaxof_d    #1\Z
597       {\expandafter\XINT_imaxof_b\romannumeral0\xintimax {#1}}%
598 \def\XINT_imaxof_e    #1\Z #2\Z { #2}%
599 \let\xintMaxof\xintiMaxof \let\xintmaxof\xintimaxof

```

4.31 `\xintiMin`, `\xintiiMin`

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

```

600 \def\xintiMin {\romannumeral0\xintimin }%
601 \def\xintimin #1%
602 {%
603   \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
604 }%
605 \def\xint_min #1#2%
606 {%
607   \expandafter\XINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
608 }%
609 \def\xintiiMin {\romannumeral0\xintiiimin }%
610 \def\xintiiimin #1%
611 {%
612   \expandafter\xint_iimin\expandafter {\romannumeral-`0#1}%
613 }%
614 \def\xint_iimin #1#2%
615 {%
616   \expandafter\XINT_min_pre\expandafter {\romannumeral-`0#2}{#1}%
617 }%
618 \let\xintMin\xintiMin \let\xintmin\xintimin % deprecated
619 \def\XINT_min_pre #1#2{\XINT_min_fork #1\Z #2\Z {#2}{#1}}%
620 \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*`

```

621 \def\XINT_min_fork #1#2\Z #3#4\Z
622 {%
623   \xint_UDsignsfork
624     #1#3\XINT_min_minusminus % A < 0, B < 0
625     #1-\XINT_min_minusplus % B < 0, A >= 0
626     #3-\XINT_min_plusminus % A < 0, B >= 0
627     --{\xint_UDzerosfork
628       #1#3\XINT_min_zerozero % A = B = 0
629       #10\XINT_min_zeropplus % B = 0, A > 0
630       #30\XINT_min_pluszero % A = 0, B > 0
631       00\XINT_min_plusplus % A, B > 0
632     \krof }%

```

4 Package `xint` implementation

```

633 \krof
634 {#2}{#4}#1#3%
635 }%

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

636 \def\XINT_min_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
637 \def\XINT_min_zeroplus #1#2#3#4{\xint_secondoftwo_thenstop }%
638 \def\XINT_min_pluszero #1#2#3#4{\xint_firstoftwo_thenstop }%
639 \def\XINT_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
640 \def\XINT_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
641 \def\XINT_min_plusplus #1#2#3#4%
642 {%
643 \ifodd\XINT_Geq {#4#2}{#3#1}
644 \expandafter\xint_secondoftwo_thenstop
645 \else
646 \expandafter\xint_firstoftwo_thenstop
647 \fi
648 }%

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

649 \def\XINT_min_minusminus #1#2#3#4%
650 {%
651 \ifodd\XINT_Geq {#1}{#2}
652 \expandafter\xint_secondoftwo_thenstop
653 \else
654 \expandafter\xint_firstoftwo_thenstop
655 \fi
656 }%

```

4.32 `\xintiMinof`

1.09a

```

657 \def\xintiMinof {\romannumeral0\xintiminof }%
658 \def\xintiminof #1{\expandafter\XINT_iminof_a\romannumeral-\`0#1\relax }%
659 \def\XINT_iminof_a #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
660 \def\XINT_iminof_b #1\Z #2%
661 {\expandafter\XINT_iminof_c\romannumeral-\`0#2\Z {#1}\Z}%
662 \def\XINT_iminof_c #1%
663 {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
664 \def\XINT_iminof_d #1\Z
665 {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
666 \def\XINT_iminof_e #1\Z #2\Z { #2}%
667 \let\xintMinof\xintiMinof \let\xintminof\xintiminof

```

4.33 `\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`

4 Package `xint` implementation

`\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.

```

668 \def\xintiiSum {\romannumeral0\xintiisum }%
669 \def\xintiisum #1{\xintiisumexpr #1\relax }%
670 \def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
671 \def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral-`0}%
672 \let\xintSum\xintiiSum \let\xintsum\xintiisum
673 \let\xintSumExpr\xintiiSumExpr \let\xintsumexpr\xintiisumexpr
674 \def\XINT_sumexpr {\XINT_sum_loop {0000}{0000}}%
675 \def\XINT_sum_loop #1#2#3%
676 {%
677   \expandafter\XINT_sum_checksign\romannumeral-`0#3\Z {#1}{#2}%
678 }%
679 \def\XINT_sum_checksign #1%
680 {%
681   \xint_gob_til_relax #1\XINT_sum_finished\relax
682   \xint_gob_til_zero #1\XINT_sum_skipzeroinput0%
683   \xint_UDsignfork
684   #1\XINT_sum_N
685   -{\XINT_sum_P #1}%
686   \krof
687 }%
688 \def\XINT_sum_finished #1\Z #2#3%
689 {%
690   \XINT_sub_A 1{#3}\W\X\Y\Z #2\W\X\Y\Z
691 }%
692 \def\XINT_sum_skipzeroinput #1\krof #2\Z {\XINT_sum_loop }%
693 \def\XINT_sum_P #1\Z #2%
694 {%
695   \expandafter\XINT_sum_loop\expandafter
696   {\romannumeral0\expandafter
697   \XINT_addr_A\expandafter0\expandafter{\expandafter}%
698   \romannumeral0\XINT_RQ {#1\R\R\R\R\R\R\R\R\Z
699   \W\X\Y\Z #2\W\X\Y\Z }%
700 }%
701 \def\XINT_sum_N #1\Z #2#3%
702 {%
703   \expandafter\XINT_sum_NN\expandafter
704   {\romannumeral0\expandafter
705   \XINT_addr_A\expandafter0\expandafter{\expandafter}%
706   \romannumeral0\XINT_RQ {#1\R\R\R\R\R\R\R\R\Z
707   \W\X\Y\Z #3\W\X\Y\Z }{#2}%
708 }%
709 \def\XINT_sum_NN #1#2{\XINT_sum_loop {#2}{#1}}%

```

4.34 `\xintiiPrd`

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

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

Release 1.02 modified the product routine. The earlier version was faster in situations where

4 Package *xint* implementation

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.

```
710 \def\xintiiPrd {\romannumeral0\xintiiprd }%
711 \def\xintiiprd #1{\xintiiprdexpr #1\relax }%
712 \let\xintPrd\xintiiPrd
713 \let\xintprd\xintiiprd
714 \def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
715 \def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral-`0}%
716 \let\xintPrdExpr\xintiiPrdExpr
717 \let\xintprdexpr\xintiiprdexpr
718 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
719 \def\XINT_prod_loop_a #1\Z #2%
720     {\expandafter\XINT_prod_loop_b \romannumeral-`0#2\Z #1\Z \Z}%
721 \def\XINT_prod_loop_b #1%
722     {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
723 \def\XINT_prod_loop_c
724     {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
725 \def\XINT_prod_finished #1\Z #2\Z \Z { #2}%
```

4.35 `\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...

```
726 \def\xintiFac {\romannumeral0\xintifac }%
727 \def\xintifac #1%
728 {%
729     \expandafter\XINT_fac_fork\expandafter{\the\numexpr #1}%
730 }%
731 \let\xintFac\xintiFac \let\xintfac\xintifac
732 \def\XINT_fac_fork #1%
733 {%
```


4 Package *xint* implementation

```

734 \ifcase\XINT_cntSgn #1\Z
735 \xint_afterfi{\expandafter\space\expandafter 1\xint_gobble_i }%
736 \or
737 \expandafter\XINT_fac_checklength
738 \else
739 \xint_afterfi{\expandafter\xintError:FactorialOfNegativeNumber
740 \expandafter\space\expandafter 1\xint_gobble_i }%
741 \fi
742 {#1}%
743 }%
744 \def\XINT_fac_checklength #1%
745 {%
746 \ifnum #1>100000
747 \xint_afterfi{\expandafter\xintError:FactorialOfTooBigNumber
748 \expandafter\space\expandafter 1\xint_gobble_i }%
749 \else
750 \xint_afterfi{\ifnum #1>\xint_c_ixixixix
751 \expandafter\XINT_fac_big_loop
752 \else
753 \expandafter\XINT_fac_loop
754 \fi }%
755 \fi
756 {#1}%
757 }%
758 \def\XINT_fac_big_loop #1{\XINT_fac_big_loop_main {10000}{#1}{}}%
759 \def\XINT_fac_big_loop_main #1#2#3%
760 {%
761 \ifnum #1<#2
762 \expandafter
763 \XINT_fac_big_loop_main
764 \expandafter
765 {\the\numexpr #1+1\expandafter }%
766 \else
767 \expandafter\XINT_fac_big_docomputation
768 \fi
769 {#2}{#3{#1}}%
770 }%
771 \def\XINT_fac_big_docomputation #1#2%
772 {%
773 \expandafter \XINT_fac_bigcompute_loop \expandafter
774 {\romannumeral0\XINT_fac_loop {9999}}#2\relax
775 }%
776 \def\XINT_fac_bigcompute_loop #1#2%
777 {%
778 \xint_gob_til_relax #2\XINT_fac_bigcompute_end\relax
779 \expandafter\XINT_fac_bigcompute_loop\expandafter
780 {\expandafter\XINT_mul_enter
781 \romannumeral0\XINT_RQ }#2\R\R\R\R\R\R\R\Z
782 \Z\Z\Z\Z #1\W\W\W\W }%
783 }%
784 \def\XINT_fac_bigcompute_end #1#2#3#4#5%
785 {%

```

```

786 \XINT_fac_bigcompute_end_ #5%
787 }%
788 \def\XINT_fac_bigcompute_end_ #1\R #2\Z \W\X\Y\Z #3\W\X\Y\Z { #3}%
789 \def\XINT_fac_loop #1{\XINT_fac_loop_main 1{1000}{#1}}%
790 \def\XINT_fac_loop_main #1#2#3%
791 {%
792 \ifnum #3>#1
793 \else
794 \expandafter\XINT_fac_loop_exit
795 \fi
796 \expandafter\XINT_fac_loop_main\expandafter
797 {\the\numexpr #1+1\expandafter }\expandafter
798 {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z }%
799 {#3}%
800 }%
801 \def\XINT_fac_loop_exit #1#2#3#4#5#6#7%
802 {%
803 \XINT_fac_loop_exit_ #6%
804 }%
805 \def\XINT_fac_loop_exit_ #1#2#3%
806 {%
807 \XINT_mul_M
808 }%

```


 DECIMAL OPERATIONS: FIRST DIGIT, LASTDIGIT, (<- moved to xintcore because xintiLDg need by division macros) ODDNESS, MULTIPLICATION BY TEN, QUOTIENT BY TEN, QUOTIENT OR MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.

4.36 \xintMON, \xintMMON, \xintiiMON, \xintiiMMON

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

```

809 \def\xintiiMON {\romannumeral0\xintiimon }%
810 \def\xintiimon #1%
811 {%
812 \ifodd\xintiilDg {#1}
813 \xint_afterfi{ -1}%
814 \else
815 \xint_afterfi{ 1}%
816 \fi
817 }%
818 \def\xintiiMMON {\romannumeral0\xintiimmon }%
819 \def\xintiimmon #1%
820 {%
821 \ifodd\xintiilDg {#1}
822 \xint_afterfi{ 1}%
823 \else
824 \xint_afterfi{ -1}%
825 \fi
826 }%
827 \def\xintMON {\romannumeral0\xintmon }%

```

```

828 \def\xintmon #1%
829 {%
830   \ifodd\xintLDg {#1}
831     \xint_afterfi{ -1}%
832   \else
833     \xint_afterfi{ 1}%
834   \fi
835 }%
836 \def\xintMMON {\romannumeral0\xintmmon }%
837 \def\xintmmon #1%
838 {%
839   \ifodd\xintLDg {#1}
840     \xint_afterfi{ 1}%
841   \else
842     \xint_afterfi{ -1}%
843   \fi
844 }%

```

4.37 *\xintOdd*, *\xintiOdd*, *\xintEven*, *\xintiiEven*

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 *\xintiiOdd*. 1.1: *\xintEven* and *\xintiiEven* added for *\xintiiexpr*.

```

845 \def\xintiiOdd {\romannumeral0\xintiiodd }%
846 \def\xintiiodd #1%
847 {%
848   \ifodd\xintiiLDg{#1}
849     \xint_afterfi{ 1}%
850   \else
851     \xint_afterfi{ 0}%
852   \fi
853 }%
854 \def\xintiiEven {\romannumeral0\xintiieven }%
855 \def\xintiieven #1%
856 {%
857   \ifodd\xintiiLDg{#1}
858     \xint_afterfi{ 0}%
859   \else
860     \xint_afterfi{ 1}%
861   \fi
862 }%
863 \def\xintOdd {\romannumeral0\xintodd }%
864 \def\xintodd #1%
865 {%
866   \ifodd\xintLDg{#1}
867     \xint_afterfi{ 1}%
868   \else
869     \xint_afterfi{ 0}%
870   \fi
871 }%
872 \def\xintEven {\romannumeral0\xinteven }%

```

```

873 \def\xinteven #1%
874 {%
875     \ifodd\xintLDg{#1}
876     \xint_afterfi{ 0}%
877     \else
878     \xint_afterfi{ 1}%
879     \fi
880 }%

```

4.38 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```

881 \def\xintDSL {\romannumeral0\xintdsl }%
882 \def\xintdsl #1%
883 {%
884     \expandafter\XINT_dsl \romannumeral-`0#1\Z
885 }%
886 \def\XINT_DSL #1{\romannumeral0\XINT_dsl #1\Z }%
887 \def\XINT_dsl #1%
888 {%
889     \xint_gob_til_zero #1\xint_dsl_zero 0\XINT_dsl_ #1%
890 }%
891 \def\xint_dsl_zero 0\XINT_dsl_ 0#1\Z { 0}%
892 \def\XINT_dsl_ #1\Z { #10}%

```

4.39 \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

```

893 \def\xintDSR {\romannumeral0\xintdsr }%
894 \def\xintdsr #1%
895 {%
896     \expandafter\XINT_dsr_a\expandafter {\romannumeral-`0#1}\W\Z
897 }%
898 \def\XINT_DSR #1{\romannumeral0\XINT_dsr_a {#1}\W\Z }%
899 \def\XINT_dsr_a
900 {%
901     \expandafter\XINT_dsr_b\romannumeral0\xintreverseorder
902 }%
903 \def\XINT_dsr_b #1#2#3\Z
904 {%
905     \xint_gob_til_W #2\xint_dsr_onedigit\W
906     \xint_gob_til_minus #2\xint_dsr_onedigit-%
907     \expandafter\XINT_dsr_removev
908     \romannumeral0\xintreverseorder {#2#3}%
909 }%
910 \def\xint_dsr_onedigit #1\xintreverseorder #2{ 0}%
911 \def\XINT_dsr_removev #1\W { }%

```

4.40 `\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.

```

912 \def\xintDSHr {\romannumeral0\xintdshr }%
913 \def\xintdshr #1%
914 {%
915   \expandafter\XINT_dshr_checkxpositive \the\numexpr #1\relax\Z
916 }%
917 \def\XINT_dshr_checkxpositive #1%
918 {%
919   \xint_UDzerominusfork
920   0#1\XINT_dshr_xzeroorneg
921   #1-\XINT_dshr_xzeroorneg
922   0-\XINT_dshr_xpositive
923   \krof #1%
924 }%
925 \def\XINT_dshr_xzeroorneg #1\Z #2{ 0}%
926 \def\XINT_dshr_xpositive #1\Z
927 {%
928   \expandafter\xint_secondoftwo_thenstop\romannumeral0\xintdsx {#1}%
929 }%
930 \def\xintDSH {\romannumeral0\xintdsh }%
931 \def\xintdsh #1#2%
932 {%
933   \expandafter\xint_dsh\expandafter {\romannumeral-`0#2}{#1}%
934 }%
935 \def\xint_dsh #1#2%
936 {%
937   \expandafter\XINT_dsh_checksignx \the\numexpr #2\relax\Z {#1}%
938 }%
939 \def\XINT_dsh_checksignx #1%
940 {%
941   \xint_UDzerominusfork
942   #1-\XINT_dsh_xiszero
943   0#1\XINT_dsx_xisNeg_checkA      % on passe direct dans DSx
944   0-\XINT_dsh_xisPos #1}%
945   \krof
946 }%
947 \def\XINT_dsh_xiszero #1\Z #2{ #2}%
948 \def\XINT_dsh_xisPos #1\Z #2%
949 {%
950   \expandafter\xint_firstoftwo_thenstop
951   \romannumeral0\XINT_dsx_checksignA #2\Z {#1}% via DSx
952 }%

```

4.41 `\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 $10^x Q \text{ pm } R$ 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 faille, 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.

```

953 \def\xintDSx {\romannumeral0\xintdsx}%
954 \def\xintdsx #1#2%
955 {%
956   \expandafter\xint_dsx\expandafter {\romannumeral-`0#2}{#1}%
957 }%
958 \def\xint_dsx #1#2%
959 {%
960   \expandafter\XINT_dsx_checksignx \the\numexpr #2\relax\Z {#1}%
961 }%
962 \def\XINT_DSx #1#2{\romannumeral0\XINT_dsx_checksignx #1\Z {#2}}%
963 \def\XINT_dsx #1#2{\XINT_dsx_checksignx #1\Z {#2}}%
964 \def\XINT_dsx_checksignx #1%
965 {%
966   \xint_UDzerominusfork
967   #1-\XINT_dsx_xisZero
968   0#1\XINT_dsx_xisNeg_checkA
969   0-\XINT_dsx_xisPos #1}%
970   \krof
971 }%
972 \def\XINT_dsx_xisZero #1\Z #2{ {#2}{0}}% attention comme  $x > 0$ 
973 \def\XINT_dsx_xisNeg_checkA #1\Z #2%
974 {%
975   \XINT_dsx_xisNeg_checkA_ #2\Z {#1}%
976 }%
977 \def\XINT_dsx_xisNeg_checkA_ #1#2\Z #3%
978 {%
979   \xint_gob_til_zero #1\XINT_dsx_xisNeg_Azero 0%

```

4 Package `xint` implementation

```

980 \XINT_dsx_xisNeg_checkx {#3}{#3}{ }\Z {#1#2}%
981 }%
982 \def\XINT_dsx_xisNeg_Azero #1\Z #2{ 0}%
983 \def\XINT_dsx_xisNeg_checkx #1%
984 {%
985 \ifnum #1>10000000
986 \xint_afterfi
987 {\xintError:TooBigDecimalShift
988 \expandafter\space\expandafter 0\xint_gobble_iv }%
989 \else
990 \expandafter \XINT_dsx_zeroloop
991 \fi
992 }%
993 \def\XINT_dsx_addzerosnofuss #1{\XINT_dsx_zeroloop {#1}{ }\Z }%
994 \def\XINT_dsx_zeroloop #1#2%
995 {%
996 \ifnum #1<\xint_c_ix \XINT_dsx_exita\fi
997 \expandafter\XINT_dsx_zeroloop\expandafter
998 {\the\numexpr #1-\xint_c_viii}{#2000000000}%
999 }%
1000 \def\XINT_dsx_exita\fi\expandafter\XINT_dsx_zeroloop
1001 {%
1002 \fi\expandafter\XINT_dsx_exitb
1003 }%
1004 \def\XINT_dsx_exitb #1#2%
1005 {%
1006 \expandafter\expandafter\expandafter
1007 \XINT_dsx_addzeros\csname xint_gobble_\romannumeral -#1\endcsname #2%
1008 }%
1009 \def\XINT_dsx_addzeros #1\Z #2{ #2#1}%
1010 \def\XINT_dsx_xisPos #1\Z #2%
1011 {%
1012 \XINT_dsx_checksiga #2\Z {#1}%
1013 }%
1014 \def\XINT_dsx_checksiga #1%
1015 {%
1016 \xint_UDzerominusfork
1017 #1-\XINT_dsx_AisZero
1018 0#1\XINT_dsx_AisNeg
1019 0-{\XINT_dsx_AisPos #1}%
1020 \krof
1021 }%
1022 \def\XINT_dsx_AisZero #1\Z #2{ {0}{0}}%
1023 \def\XINT_dsx_AisNeg #1\Z #2%
1024 {%
1025 \expandafter\XINT_dsx_AisNeg_dosplit_andcheckfirst
1026 \romannumeral0\XINT_split_checksizex {#2}{#1}%
1027 }%
1028 \def\XINT_dsx_AisNeg_dosplit_andcheckfirst #1%
1029 {%
1030 \XINT_dsx_AisNeg_checkiffirstempty #1\Z
1031 }%

```

```

1032 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
1033 {%
1034     \xint_gob_til_Z #1\XINT_dsx_AisNeg_finish_zero\Z
1035     \XINT_dsx_AisNeg_finish_notzero #1%
1036 }%
1037 \def\XINT_dsx_AisNeg_finish_zero\Z
1038     \XINT_dsx_AisNeg_finish_notzero\Z #1%
1039 {%
1040     \expandafter\XINT_dsx_end
1041     \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
1042 }%
1043 \def\XINT_dsx_AisNeg_finish_notzero #1\Z #2%
1044 {%
1045     \expandafter\XINT_dsx_end
1046     \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
1047 }%
1048 \def\XINT_dsx_AisPos #1\Z #2%
1049 {%
1050     \expandafter\XINT_dsx_AisPos_finish
1051     \romannumeral0\XINT_split_checksizex {#2}{#1}%
1052 }%
1053 \def\XINT_dsx_AisPos_finish #1#2%
1054 {%
1055     \expandafter\XINT_dsx_end
1056     \expandafter {\romannumeral0\XINT_num {#2}}%
1057     {\romannumeral0\XINT_num {#1}}%
1058 }%
1059 \edef\XINT_dsx_end #1#2%
1060 {%
1061     \noexpand\expandafter\space\noexpand\expandafter{#2}{#1}%
1062 }%

```

4.42 \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.

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.

```

1063 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
1064 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%

```



```

1065 \def\xintdecsplitl
1066 {%
1067     \expandafter\xint_firstoftwo_thenstop
1068     \romannumeral0\xintdecsplit
1069 }%
1070 \def\xintdecsplitr
1071 {%
1072     \expandafter\xint_secondoftwo_thenstop
1073     \romannumeral0\xintdecsplit
1074 }%
1075 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
1076 \def\xintdecsplit #1#2%
1077 {%
1078     \expandafter \xint_split \expandafter
1079     {\romannumeral0\xintiiaabs {#2}}{#1}% fait expansion de A
1080 }%
1081 \def\xint_split #1#2%
1082 {%
1083     \expandafter\XINT_split_checksizex\expandafter{\the\numexpr #2}{#1}%
1084 }%
1085 \def\XINT_split_checksizex #1% 999999999 is anyhow very big, could be reduced
1086 {%
1087     \ifnum\numexpr\XINT_Abs{#1}>999999999
1088         \xint_afterfi {\xintError:TooBigDecimalSplit\XINT_split_bigx }%
1089     \else
1090         \expandafter\XINT_split_xfork
1091     \fi
1092     #1\Z
1093 }%
1094 \def\XINT_split_bigx #1\Z #2%
1095 {%
1096     \ifcase\XINT_cntSgn #1\Z
1097     \or \xint_afterfi { }{#2}% positive big x
1098     \else
1099         \xint_afterfi { }{#2}% negative big x
1100     \fi
1101 }%
1102 \def\XINT_split_xfork #1%
1103 {%
1104     \xint_UDzerominusfork
1105     #1-\XINT_split_zerosplit
1106     0#1\XINT_split_fromleft
1107     0-\XINT_split_fromright #1}%
1108     \krof
1109 }%
1110 \def\XINT_split_zerosplit #1\Z #2{ }{#2}%
1111 \def\XINT_split_fromleft #1\Z #2%
1112 {%
1113     \XINT_split_fromleft_loop {#1}{#2\W\W\W\W\W\W\W\W\Z
1114 }%
1115 \def\XINT_split_fromleft_loop #1%
1116 {%

```

4 Package `xint` implementation

```

1117 \ifnum #1<\xint_c_viii\xint_split_fromleft_exita\fi
1118 \expandafter\xint_split_fromleft_loop_perhaps\expandafter
1119 {\the\numexpr #1-\xint_c_viii\expandafter}\xint_split_fromleft_eight
1120 }%
1121 \def\xint_split_fromleft_eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
1122 \def\xint_split_fromleft_loop_perhaps #1#2%
1123 {%
1124 \xint_gob_til_W #2\xint_split_fromleft_toofar\W
1125 \xint_split_fromleft_loop {#1}%
1126 }%
1127 \def\xint_split_fromleft_toofar\W\xint_split_fromleft_loop #1#2#3\Z
1128 {%
1129 \xint_split_fromleft_toofar_b #2\Z
1130 }%
1131 \def\xint_split_fromleft_toofar_b #1\W #2\Z { {#1}{}}%
1132 \def\xint_split_fromleft_exita\fi
1133 \expandafter\xint_split_fromleft_loop_perhaps\expandafter #1#2%
1134 {\fi \xint_split_fromleft_exitb #1}%
1135 \def\xint_split_fromleft_exitb\the\numexpr #1-\xint_c_viii\expandafter
1136 {%
1137 \csname XINT_split_fromleft_endsplit_\romannumeral #1\endcsname
1138 }%
1139 \def\xint_split_fromleft_endsplit_ #1#2\W #3\Z { {#1}{#2}}%
1140 \def\xint_split_fromleft_endsplit_i #1#2%
1141 {\xint_split_fromleft_checkiftoofar #2{#1#2}}%
1142 \def\xint_split_fromleft_endsplit_ii #1#2#3%
1143 {\xint_split_fromleft_checkiftoofar #3{#1#2#3}}%
1144 \def\xint_split_fromleft_endsplit_iii #1#2#3#4%
1145 {\xint_split_fromleft_checkiftoofar #4{#1#2#3#4}}%
1146 \def\xint_split_fromleft_endsplit_iv #1#2#3#4#5%
1147 {\xint_split_fromleft_checkiftoofar #5{#1#2#3#4#5}}%
1148 \def\xint_split_fromleft_endsplit_v #1#2#3#4#5#6%
1149 {\xint_split_fromleft_checkiftoofar #6{#1#2#3#4#5#6}}%
1150 \def\xint_split_fromleft_endsplit_vi #1#2#3#4#5#6#7%
1151 {\xint_split_fromleft_checkiftoofar #7{#1#2#3#4#5#6#7}}%
1152 \def\xint_split_fromleft_endsplit_vii #1#2#3#4#5#6#7#8%
1153 {\xint_split_fromleft_checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
1154 \def\xint_split_fromleft_checkiftoofar #1#2#3\W #4\Z
1155 {%
1156 \xint_gob_til_W #1\xint_split_fromleft_wenttoofar\W
1157 \space {#2}{#3}%
1158 }%
1159 \def\xint_split_fromleft_wenttoofar\W\space #1%
1160 {%
1161 \xint_split_fromleft_wenttoofar_b #1\Z
1162 }%
1163 \def\xint_split_fromleft_wenttoofar_b #1\W #2\Z { {#1}}%
1164 \def\xint_split_fromright #1\Z #2%
1165 {%
1166 \expandafter \xint_split_fromright_a \expandafter
1167 {\romannumeral0\xintreverseorder {#2}}{#1}{#2}%
1168 }%

```

4 Package *xint* implementation

```

1169 \def\XINT_split_fromright_a #1#2%
1170 {%
1171   \XINT_split_fromright_loop {#2}{#1\W\W\W\W\W\W\W\W\Z
1172 }%
1173 \def\XINT_split_fromright_loop #1%
1174 {%
1175   \ifnum #1<\xint_c_viii\XINT_split_fromright_exita\fi
1176   \expandafter\XINT_split_fromright_loop_perhaps\expandafter
1177   {\the\numexpr #1-\xint_c_viii\expandafter }\XINT_split_fromright_eight
1178 }%
1179 \def\XINT_split_fromright_eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
1180 \def\XINT_split_fromright_loop_perhaps #1#2%
1181 {%
1182   \xint_gob_til_W #2\XINT_split_fromright_toofar\W
1183   \XINT_split_fromright_loop {#1}%
1184 }%
1185 \def\XINT_split_fromright_toofar\W\XINT_split_fromright_loop #1#2#3\Z { {}}%
1186 \def\XINT_split_fromright_exita\fi
1187   \expandafter\XINT_split_fromright_loop_perhaps\expandafter #1#2%
1188   {\fi \XINT_split_fromright_exitb #1}%
1189 \def\XINT_split_fromright_exitb\the\numexpr #1-\xint_c_viii\expandafter
1190 {%
1191   \csname XINT_split_fromright_endsplit_\romannumeral #1\endcsname
1192 }%
1193 \edef\XINT_split_fromright_endsplit_ #1#2\W #3\Z #4%
1194 {%
1195   \noexpand\expandafter\space\noexpand\expandafter
1196   {\noexpand\romannumeral0\noexpand\xintreverseorder {#2}}{#1}%
1197 }%
1198 \def\XINT_split_fromright_endsplit_i #1#2%
1199   {\XINT_split_fromright_checkiftoofar #2{#2#1}}%
1200 \def\XINT_split_fromright_endsplit_ii #1#2#3%
1201   {\XINT_split_fromright_checkiftoofar #3{#3#2#1}}%
1202 \def\XINT_split_fromright_endsplit_iii #1#2#3#4%
1203   {\XINT_split_fromright_checkiftoofar #4{#4#3#2#1}}%
1204 \def\XINT_split_fromright_endsplit_iv #1#2#3#4#5%
1205   {\XINT_split_fromright_checkiftoofar #5{#5#4#3#2#1}}%
1206 \def\XINT_split_fromright_endsplit_v #1#2#3#4#5#6%
1207   {\XINT_split_fromright_checkiftoofar #6{#6#5#4#3#2#1}}%
1208 \def\XINT_split_fromright_endsplit_vi #1#2#3#4#5#6#7%
1209   {\XINT_split_fromright_checkiftoofar #7{#7#6#5#4#3#2#1}}%
1210 \def\XINT_split_fromright_endsplit_vii #1#2#3#4#5#6#7#8%
1211   {\XINT_split_fromright_checkiftoofar #8{#8#7#6#5#4#3#2#1}}%
1212 \def\XINT_split_fromright_checkiftoofar #1%
1213 {%
1214   \xint_gob_til_W #1\XINT_split_fromright_wenttoofar\W
1215   \XINT_split_fromright_endsplit_
1216 }%
1217 \def\XINT_split_fromright_wenttoofar\W\XINT_split_fromright_endsplit_ #1\Z #2%
1218   { {}{#2}}%

```

4.43 `\xintiiSqrt`, `\xintiiSqrtR`, `\xintiiSquareRoot`

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

Some overhead was added inadvertently in 1.09a to inner routines when `\xintquo` 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.

```

1219 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
1220 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
1221 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
1222 \def\xintisqrt {\romannumeral0\xintisqrt }%
1223 \def\xintisquareRoot {\romannumeral0\xintisquareroot }%
1224 \def\xintisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot }%
1225 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1226 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
1227 \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, <= N-0.75<N, donc rounded->#1 si #2>=#1, (#1-.5)^2>=N+.25>N, donc rounded-
->#1-1.

1228 \def\XINT_sqrtr_post #1#2{\xintiiiflt {#2}{#1}% Lt <-> a<b
1229 { #1}{\XINT_dec_pos #1\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }}%
1230 \def\xintisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\Z }%
1231 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral-`0#1\Z }%
1232 \def\XINT_sqrt_checkin #1%
1233 {%
1234 \xint_UDzerominusfork
1235 #1-\XINT_sqrt_iszero
1236 0#1\XINT_sqrt_isneg
1237 0-{\XINT_sqrt #1}%
1238 \krof
1239 }%
1240 \def\XINT_sqrt_iszero #1\Z { 11}%
1241 \edef\XINT_sqrt_isneg #1\Z {\noexpand\xintError:RootOfNegative\space 11}%
1242 \def\XINT_sqrt #1\Z
1243 {%
1244 \expandafter\XINT_sqrt_start\expandafter {\romannumeral0\xintlenght {#1}}{#1}%
1245 }%
1246 \def\XINT_sqrt_start #1%
1247 {%
1248 \ifnum #1<\xint_c_x
1249 \expandafter\XINT_sqrt_small_a
1250 \else
1251 \expandafter\XINT_sqrt_big_a
1252 \fi
1253 {#1}%
1254 }%
1255 \def\XINT_sqrt_small_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_small_d }%
1256 \def\XINT_sqrt_big_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_big_d }%
1257 \def\XINT_sqrt_a #1%
```

```

1258 {%
1259   \ifodd #1
1260     \expandafter\XINT_sqrt_bB
1261   \else
1262     \expandafter\XINT_sqrt_bA
1263   \fi
1264   {#1}%
1265 }%
1266 \def\XINT_sqrt_bA #1#2#3%
1267 {%
1268   \XINT_sqrt_bA_b #3\Z #2{#1}{#3}%
1269 }%
1270 \def\XINT_sqrt_bA_b #1#2#3\Z
1271 {%
1272   \XINT_sqrt_c {#1#2}%
1273 }%
1274 \def\XINT_sqrt_bB #1#2#3%
1275 {%
1276   \XINT_sqrt_bB_b #3\Z #2{#1}{#3}%
1277 }%
1278 \def\XINT_sqrt_bB_b #1#2\Z
1279 {%
1280   \XINT_sqrt_c #1%
1281 }%
1282 \def\XINT_sqrt_c #1#2%
1283 {%
1284   \expandafter #2\expandafter
1285   {\the\numexpr\ifnum #1>\xint_c_iii
1286     \ifnum #1>\xint_c_viii
1287       \ifnum #1>15 \ifnum #1>24 \ifnum #1>35
1288         \ifnum #1>48 \ifnum #1>63 \ifnum #1>80
1289       10\else 9\fi \else 8\fi \else 7\fi \else 6\fi
1290     \else 5\fi \else 4\fi \else 3\fi \else 2\fi \relax }%
1291 }%
1292 \def\XINT_sqrt_small_d #1#2%
1293 {%
1294   \expandafter\XINT_sqrt_small_e\expandafter
1295   {\the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1296     \or 0\or 00\or 000\or 0000\fi }%
1297 }%
1298 \def\XINT_sqrt_small_e #1#2%
1299 {%
1300   \expandafter\XINT_sqrt_small_f\expandafter {\the\numexpr #1*#1-#2}{#1}%
1301 }%
1302 \def\XINT_sqrt_small_f #1#2%
1303 {%
1304   \expandafter\XINT_sqrt_small_g\expandafter
1305   {\the\numexpr ((#1+#2)/(\xint_c_ii*#2))-\xint_c_i}{#1}{#2}%
1306 }%
1307 \def\XINT_sqrt_small_g #1%
1308 {%
1309   \ifnum #1>\xint_c_

```

4 Package `xint` implementation

```

1310     \expandafter\XINT_sqrt_small_h
1311   \else
1312     \expandafter\XINT_sqrt_small_end
1313   \fi
1314   {#1}%
1315 }%
1316 \def\XINT_sqrt_small_h #1#2#3%
1317 {%
1318   \expandafter\XINT_sqrt_small_f\expandafter
1319   {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
1320   {\the\numexpr #3-#1}%
1321 }%
1322 \def\XINT_sqrt_small_end #1#2#3{ {#3}{#2}}%
1323 \def\XINT_sqrt_big_d #1#2%
1324 {%
1325   \ifodd #2
1326     \expandafter\expandafter\expandafter\XINT_sqrt_big_eB
1327   \else
1328     \expandafter\expandafter\expandafter\XINT_sqrt_big_eA
1329   \fi
1330   \expandafter {\the\numexpr #2/\xint_c_ii }{#1}%
1331 }%
1332 \def\XINT_sqrt_big_eA #1#2#3%
1333 {%
1334   \XINT_sqrt_big_eA_a #3\Z {#2}{#1}{#3}%
1335 }%
1336 \def\XINT_sqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
1337 {%
1338   \XINT_sqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
1339 }%
1340 \def\XINT_sqrt_big_eA_b #1#2%
1341 {%
1342   \expandafter\XINT_sqrt_big_f
1343   \romannumeral0\XINT_sqrt_small_e {#2000}{#1}{#1}%
1344 }%
1345 \def\XINT_sqrt_big_eB #1#2#3%
1346 {%
1347   \XINT_sqrt_big_eB_a #3\Z {#2}{#1}{#3}%
1348 }%
1349 \def\XINT_sqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
1350 {%
1351   \XINT_sqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
1352 }%
1353 \def\XINT_sqrt_big_eB_b #1#2\Z #3%
1354 {%
1355   \expandafter\XINT_sqrt_big_f
1356   \romannumeral0\XINT_sqrt_small_e {#30000}{#1}{#1}%
1357 }%
1358 \def\XINT_sqrt_big_f #1#2#3#4%
1359 {%
1360   \expandafter\XINT_sqrt_big_f_a\expandafter
1361   {\the\numexpr #2+#3\expandafter}\expandafter

```

5 Package *xintbinhex* implementation

```

1362   {\romannumeral0\XINT_dsx_addzerosnofuss
1363           {\numexpr #4-\xint_c_iv\relax}{#1}}{#4}%
1364 }%
1365 \def\XINT_sqrt_big_f_a #1#2#3#4%
1366 {%
1367   \expandafter\XINT_sqrt_big_g\expandafter
1368   {\romannumeral0\xintiisub
1369     {\XINT_dsx_addzerosnofuss
1370       {\numexpr \xint_c_ii*#3-\xint_c_viii\relax}{#1}}{#4}}%
1371   {#2}{#3}%
1372 }%
1373 \def\XINT_sqrt_big_g #1#2%
1374 {%
1375   \expandafter\XINT_sqrt_big_j
1376   \romannumeral0\xintiidivision{#1}%
1377   {\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }{#2}%
1378 }%
1379 \def\XINT_sqrt_big_j #1%
1380 {%
1381   \if0\XINT_Sgn #1\Z
1382     \expandafter \XINT_sqrt_big_end
1383   \else \expandafter \XINT_sqrt_big_k
1384   \fi {#1}%
1385 }%
1386 \def\XINT_sqrt_big_k #1#2#3%
1387 {%
1388   \expandafter\XINT_sqrt_big_l\expandafter
1389   {\romannumeral0\xintiisub {#3}{#1}}%
1390   {\romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}}%
1391 }%
1392 \def\XINT_sqrt_big_l #1#2%
1393 {%
1394   \expandafter\XINT_sqrt_big_g\expandafter
1395   {#2}{#1}%
1396 }%
1397 \def\XINT_sqrt_big_end #1#2#3#4{ {#3}{#2}}%

```

4.44 *\xintiiE*

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

```

1398 \def\xintiiE {\romannumeral0\xintiiE }% used in \xintMod.
1399 \def\xintiiE #1#2%
1400   {\expandafter\XINT_iiE\the\numexpr #2\expandafter.\expandafter{\romannumeral-`0#1}}%
1401 \def\XINT_iiE #1.#2{\ifnum#1>\xint_c_ \xint_dothis{\xint_dsh {#2}{-#1}}\fi
1402           \xint_orthat{ #2}}%
1403 \XINT_restorecatcodes_endinput%

```

5 Package *xintbinhex* implementation

.1	Catcodes, ε -TeX and reload detection	120	.3	Constants, etc...	121
.2	Package identification	121	.4	<i>\xintDecToHex</i> , <i>\xintDecToBin</i>	123

.5	<code>\xintHexToDec</code>	126	.8	<code>\xintHexToBin</code>	130
.6	<code>\xintBinToDec</code>	127	.9	<code>\xintCHexToBin</code>	131
.7	<code>\xintBinToHex</code>	129			

The commenting is currently (2015/09/17) very sparse.

5.1 Catcodes, ε -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintbinhex.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@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 [2015/09/12 v1.1c 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}%
85 \expandafter\def\csname XINT_sbtd_1110\endcsname {14}%
86 \expandafter\def\csname XINT_sbtd_1111\endcsname {15}%
87 \expandafter\let\csname XINT_sbth_0000\endcsname\expandafter\endcsname
88   \csname XINT_sbtd_0000\endcsname
89 \expandafter\let\csname XINT_sbth_0001\endcsname\expandafter\endcsname
90   \csname XINT_sbtd_0001\endcsname

```

5 Package *xintbinhex* implementation

```

91 \expandafter\let\csname XINT_sbth_0010\expandafter\endcsname
92     \csname XINT_sbtd_0010\endcsname
93 \expandafter\let\csname XINT_sbth_0011\expandafter\endcsname
94     \csname XINT_sbtd_0011\endcsname
95 \expandafter\let\csname XINT_sbth_0100\expandafter\endcsname
96     \csname XINT_sbtd_0100\endcsname
97 \expandafter\let\csname XINT_sbth_0101\expandafter\endcsname
98     \csname XINT_sbtd_0101\endcsname
99 \expandafter\let\csname XINT_sbth_0110\expandafter\endcsname
100     \csname XINT_sbtd_0110\endcsname
101 \expandafter\let\csname XINT_sbth_0111\expandafter\endcsname
102     \csname XINT_sbtd_0111\endcsname
103 \expandafter\let\csname XINT_sbth_1000\expandafter\endcsname
104     \csname XINT_sbtd_1000\endcsname
105 \expandafter\let\csname XINT_sbth_1001\expandafter\endcsname
106     \csname XINT_sbtd_1001\endcsname
107 \expandafter\def\csname XINT_sbth_1010\endcsname {A}%
108 \expandafter\def\csname XINT_sbth_1011\endcsname {B}%
109 \expandafter\def\csname XINT_sbth_1100\endcsname {C}%
110 \expandafter\def\csname XINT_sbth_1101\endcsname {D}%
111 \expandafter\def\csname XINT_sbth_1110\endcsname {E}%
112 \expandafter\def\csname XINT_sbth_1111\endcsname {F}%
113 \expandafter\def\csname XINT_shtb_0\endcsname {0000}%
114 \expandafter\def\csname XINT_shtb_1\endcsname {0001}%
115 \expandafter\def\csname XINT_shtb_2\endcsname {0010}%
116 \expandafter\def\csname XINT_shtb_3\endcsname {0011}%
117 \expandafter\def\csname XINT_shtb_4\endcsname {0100}%
118 \expandafter\def\csname XINT_shtb_5\endcsname {0101}%
119 \expandafter\def\csname XINT_shtb_6\endcsname {0110}%
120 \expandafter\def\csname XINT_shtb_7\endcsname {0111}%
121 \expandafter\def\csname XINT_shtb_8\endcsname {1000}%
122 \expandafter\def\csname XINT_shtb_9\endcsname {1001}%
123 \def\XINT_shtb_A {1010}%
124 \def\XINT_shtb_B {1011}%
125 \def\XINT_shtb_C {1100}%
126 \def\XINT_shtb_D {1101}%
127 \def\XINT_shtb_E {1110}%
128 \def\XINT_shtb_F {1111}%
129 \def\XINT_shtb_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 {%
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\xintdecto hex}%
151 \def\xintdecto hex #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
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

```

5 Package *xintbinhex* implementation

```

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 {%
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.%

```

5 Package *xintbinhex* implementation

```

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 }%
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 {\XINT_htd_II_d {}{"#2}{\xint_c_xvi}}%
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

```

5 Package *xintbinhex* implementation

```
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 }%
385 \def\XINT_btd_checkin #1%
386 {%
387 \xint_UDsignfork
388 #1\XINT_btd_neg
389 -{\XINT_btd_I {000000}{#1}}%
390 \krof
```

5 Package *xintbinhex* implementation

```

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
436 {%
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+%

```


5 Package *xintbinhex* implementation

```
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#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#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#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
```

5 Package *xintbinhex* implementation

```
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\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#1GGGGGGGG\T
530 }%
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_cho_b_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_cho_b_checkin #1%
573 {%
574   \xint_UDsignfork
575   #1\XINT_cho_b_N
576   -{\XINT_cho_b_P #1}%
577   \krof
578 }%
579 \def\XINT_cho_b_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_cho_b_P }%
580 \def\XINT_cho_b_P {\expandafter\XINT_cho_b_I\expandafter{\expandafter}%
581   \romannumeral0\XINT_OQ {}}%
582 \def\XINT_cho_b_I #1#2#3#4#5#6#7#8#9%
583 {%
584   \xint_gob_til_W #9\XINT_cho_b_end_a\W
585   \expandafter\expandafter\expandafter
586   \XINT_cho_b_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, ε - \TeX and reload detection	132	.7	<code>\xintBezoutAlgorithm</code>	140
.2	Package identification	133	.8	<code>\xintGCDof</code>	142
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	133	.9	<code>\xintLCMof</code>	142
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	134	.10	<code>\xintTypesetEuclideanAlgorithm</code>	143
.5	<code>\xintBezout</code>	135	.11	<code>\xintTypesetBezoutAlgorithm</code>	143
.6	<code>\xintEuclideanAlgorithm</code>	139			

The commenting is currently (2015/09/17) 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`.

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

6.1 Catcodes, ε - \TeX and reload detection

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

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

```

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

```

```

9  \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2015/09/12 v1.1c 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.

```

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\xintiiGCD {\romannumeral0\xintiigcd }%
57 \def\xintiigcd #1%
58 {%
59   \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiabs {#1}}%
60 }%
61 \def\XINT_iigcd #1#2%
62 {%
63   \expandafter\XINT_gcd_fork\romannumeral0\xintiabs {#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. The advantage is that we can thus use `lcm` in `\xintexpr`. The disadvantage is that this has overhead in `\xintiiexpr`. Thus 1.1a has `\xintiiLCM`.

```

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\xintiilcm {\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 {%
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_mm_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\xinttiopp {#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\xinttiopp{#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 }%

```

n = 0: 1BAalpha(0)beta(0)alpha(-1)beta(-1)

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

#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}%

{\alpha(n)}{\beta(n)}{(-1)^{n+1}}\{r(n)\}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\xinteucclideanalgorithm }%
264 \def\xinteucclideanalgorithm #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%
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)}{al
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)}...

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

$$\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$$

$$\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$$

```

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

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:

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

$\backslash U1 = N = \text{nombre d'étapes}$, $\backslash U3 = \text{PGCD}$, $\backslash U2 = A$, $\backslash U4=B$ $q_1 = \backslash U5$, $q_2 = \backslash U7 \rightarrow q_n = \backslash U<2n+3>$, $r_n = \backslash U<2n+4>$ $bn = rn$. $B = r_0$. $A=r(-1)$

$r(n-2) = q(n)r(n-1)+r(n)$ (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: $U1 = N$, $U2= A$, $U5=D$, $U6=B$, $q_1 = U9$, $q_n = U\{4n+5\}$, n au moins 1

$r_n = U\{4n+6\}$, n au moins -1

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

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

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

```

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

.1	Catcodes, ε -TeX and reload detection	. . . 145	.9	<code>\XINT_factortens</code> , <code>\XINT_cuz_cnt</code>	. . . 151
.2	Package identification	. . . 146	.10	<code>\xintRaw</code>	. . . 152
.3	<code>\XINT_cntSgnFork</code>	. . . 146	.11	<code>\xintPRaw</code>	. . . 153
.4	<code>\xintLen</code>	. . . 146	.12	<code>\xintRawWithZeros</code>	. . . 153
.5	<code>\XINT_lenrord_loop</code>	. . . 147	.13	<code>\xintFloor</code> , <code>\xintiFloor</code>	. . . 154
.6	<code>\XINT_outfrac</code>	. . . 147	.14	<code>\xintCeil</code> , <code>\xintiCeil</code>	. . . 154
.7	<code>\XINT_inFrac</code>	. . . 148	.15	<code>\xintNumerator</code>	. . . 154
.8	<code>\XINT_frac</code>	. . . 149	.16	<code>\xintDenominator</code>	. . . 154

.17	<code>\xintFrac</code>	155	.43	<code>\xintFac</code>	182
.18	<code>\xintSignedFrac</code>	155	.44	<code>\xintPrd</code>	183
.19	<code>\xintFwOver</code>	156	.45	<code>\xintDiv</code>	183
.20	<code>\xintSignedFwOver</code>	156	.46	<code>\xintDivFloor</code>	183
.21	<code>\xintREZ</code>	157	.47	<code>\xintDivTrunc</code>	184
.22	<code>\xintE, \xintFloatE, \XINTinFloatE</code>	157	.48	<code>\xintDivRound</code>	184
.23	<code>\xintIrr</code>	159	.49	<code>\xintMod</code>	184
.24	<code>\xintifInt</code>	160	.50	<code>\XINTinFloatMod</code>	185
.25	<code>\xintJrr</code>	160	.51	<code>\xintIsOne</code>	185
.26	<code>\xintTFrac</code>	161	.52	<code>\xintGeq</code>	185
.27	<code>\XINTinFloatFracdigits</code>	162	.53	<code>\xintMax</code>	186
.28	<code>\xintTrunc, \xintiTrunc</code>	162	.54	<code>\xintMaxof</code>	187
.29	<code>\xintTTrunc</code>	164	.55	<code>\xintMin</code>	187
.30	<code>\xintNum</code>	164	.56	<code>\xintMinof</code>	188
.31	<code>\xintRound, \xintiRound</code>	165	.57	<code>\xintCmp</code>	188
.32	<code>\xintXTrunc</code>	166	.58	<code>\xintAbs</code>	190
.33	<code>\xintDigits</code>	171	.59	<code>\xintOpp</code>	190
.34	<code>\xintFloat</code>	172	.60	<code>\xintSgn</code>	190
.35	<code>\xintPFloat</code>	175	.61	<code>\xintFloatAdd, \XINTinFloatAdd</code>	190
.36	<code>\XINTinFloat</code>	176	.62	<code>\xintFloatSub, \XINTinFloatSub</code>	191
.37	<code>\xintAdd</code>	178	.63	<code>\xintFloatMul, \XINTinFloatMul</code>	191
.38	<code>\xintSub</code>	180	.64	<code>\xintFloatDiv, \XINTinFloatDiv</code>	192
.39	<code>\xintSum</code>	180	.65	<code>\xintFloatPow, \XINTinFloatPow</code>	193
.40	<code>\xintMul</code>	181	.66	<code>\xintFloatPower, \XINTinFloatPower</code>	196
.41	<code>\xintSqr</code>	181	.67	<code>\xintFloatSqrt, \XINTinFloatSqrt</code>	198
.42	<code>\xintPow</code>	181			

The commenting is currently (2015/09/17) very sparse.

7.1 Catcodes, ε -T_EX and reload detection

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

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

```

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

```

```

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

```

7.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2015/09/12 v1.1c 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 {%
58   \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
59 }%
60 \def\XINT_flen #1#2#3%
61 {%

```

```

65 \def\XINT_lenrord_loop #1#2#3#4#5#6#7#8#9%
66 {% faire \romannumeral-`0\XINT_lenrord_loop 0{ }#1\Z\W\W\W\W\W\W\W\Z
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 }%

```

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, xintcfac, to make sure the output format for fractions was always A/B[n]. (except \xintIrr, \xintJrr, \xintRawWithZeros)

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.

147

```

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}%
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 }%
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%

```

7 Package *xintfrac* implementation

```

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\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\xinrawwithzeros }%
367 \def\xinrawwithzeros
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.{\xintiigo {#1}{#2}}%

```

7.14 `\xintCeil`, `\xintiCeil`

1.09a

```

393 \def\xintCeil {\romannumeral0\xintceil }%
394 \def\xintceil #1{\xintiiopp {\xintFloor {\xintOpp{#1}}}}%
395 \def\xintiCeil {\romannumeral0\xinticeil }%
396 \def\xinticeil #1{\xintiiopp {\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 {%
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
595   \expandafter\xint_exchangenew_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_denomisone #1}%
618   \fi
619   #2\Z {#3}%
620 }%
621 \def\XINT_irr_denomisone #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\xintrawwithzeros {#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\xintrawwithzeros {#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}%
683   \else
684     \xint_afterfi{\XINT_jrr_denomisone #1}%
685   \fi
686   #2\Z {#3}%
687 }%
688 \def\XINT_jrr_denomisone #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\xinttiadd{\XINT_Mul{#4}{#1}}{#6}}%
713     {\romannumeral0\xinttiadd{\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\xintrawwithzeros {#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 `\xintfrac` 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 `\xintquo` was redefined to use `\xintnum`. Now uses `\xintiquo`, 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 }%
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 {%

```

7 Package *xintfrac* implementation

```

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 }%
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_zeroloop {#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 `\xintrawithzeros`). 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.

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`).

[illegible]

```

972 \def\XINT_xtrunc_negN_Q #1#2.#3#4#5#6%
973 {%
974     \expandafter\XINT_xtrunc_negN_R
975     \romannumeral0\XINT_div_prepare {#3}{#2}{#3}{#1}{#4}%
976 }%

    #1=Q, #2=R, #3=B, #4=N<0, #5=D

977 \def\XINT_xtrunc_negN_R #1#2#3#4#5%
978 {%
979     \expandafter\XINT_xtrunc_negN_S\expandafter
980     {\the\numexpr -#4}{#5}{#2}{#3}{#1}%
981 }%
982 \def\XINT_xtrunc_negN_S #1#2%
983 {%
984     \expandafter\XINT_xtrunc_negN_T\expandafter
985     {\the\numexpr #2-#1}{#1}{#2}%
986 }%
987 \def\XINT_xtrunc_negN_T #1%
988 {%
989     \ifnum \xint_c_<#1
990         \expandafter\XINT_xtrunc_negNA
991     \else
992         \expandafter\XINT_xtrunc_negNW
993     \fi {#1}%
994 }%

    #1=D-|N|>0, #2=|N|, #3=D, #4=R, #5=B, #6=Q

995 \def\XINT_xtrunc_unlock #10.{ }%
996 \def\XINT_xtrunc_negNA #1#2#3#4#5#6%
997 {%
998     \expandafter\XINT_xtrunc_negNB\expandafter
999     {\romannumeral0\expandafter\expandafter\expandafter
1000      \XINT_xtrunc_unlock\expandafter|string
1001      \csize\XINT_xtrunc_b {#1}#4/#5[0]\expandafter\endcsname
1002      \expandafter}\expandafter
1003      {\the\numexpr\xintLength{#6}-#2}{#6}%
1004 }%
1005 \def\XINT_xtrunc_negNB #1#2#3{\XINT_xtrunc_negNC {#2}{#3}{#1}%
1006 \def\XINT_xtrunc_negNC #1%
1007 {%
1008     \ifnum \xint_c_ < #1
1009         \expandafter\XINT_xtrunc_negNDa
1010     \else
1011         \expandafter\XINT_xtrunc_negNE
1012     \fi {#1}%
1013 }%
1014 \def\XINT_xtrunc_negNDa #1#2%
1015 {%
1016     \expandafter\XINT_xtrunc_negNDb%
1017     \romannumeral0\XINT_split_fromleft_loop {#1}{#2\W\W\W\W\W\W\W\W\Z
1018 }%
1019 \def\XINT_xtrunc_negNDb #1#2{#1.#2}%

```


[illegible]

[illegible]

[illegible]

7.33 \xintDigits

The mathchardef used to be called `\XINT_digits`, but for reasons originating in `\xintNewExpr` (and now obsolete), release 1.09a uses `\XINTdigits` without underscore.

```

1162 \mathchardef\XINTdigits 16
1163 \def\xintDigits #1#2%
1164   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=%}
1165 \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.

```

1166 \def\xintFloat    {\romannumeral0\xintfloat }%
1167 \def\xintfloat #1{\XINT_float_chkopt #1\xint_relax }%
1168 \def\XINT_float_chkopt #1%
1169 {%
1170   \ifx [#1\expandafter\XINT_float_opt
1171     \else\expandafter\XINT_float_noopt
1172   \fi #1%
1173 }%
1174 \def\XINT_float_noopt #1\xint_relax
1175 {%
1176   \expandafter\XINT_float_a\expandafter\XINTdigits
1177   \romannumeral0\XINT_infrac {#1}\XINT_float_Q
1178 }%
1179 \def\XINT_float_opt [\xint_relax #1]#2%
1180 {%
1181   \expandafter\XINT_float_a\expandafter
1182   {\the\numexpr #1\expandafter}%
1183   \romannumeral0\XINT_infrac {#2}\XINT_float_Q
1184 }%
1185 \def\XINT_float_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1186 {%
1187   \XINT_float_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1188 }%
1189 \def\XINT_float_fork #1%
1190 {%
1191   \xint_UDzerominusfork
1192   #1-\XINT_float_zero
1193   0#1\XINT_float_J
1194   0-\XINT_float_K #1}%
1195   \krof
1196 }%
1197 \def\XINT_float_zero #1\Z #2#3#4#5{ 0.e0}%
1198 \def\XINT_float_J {\expandafter\xint_minus_thenstop\romannumeral0\XINT_float_K }%
1199 \def\XINT_float_K #1\Z #2% #1=A, #2=P, #3=n, #4=B
1200 {%
1201   \expandafter\XINT_float_L\expandafter
1202   {\the\numexpr\xintLength{#1}\expandafter}\expandafter
1203   {\the\numexpr #2+\xint_c_ii}{#1}{#2}%
1204 }%
1205 \def\XINT_float_L #1#2%
1206 {%
1207   \ifnum #1>#2
1208     \expandafter\XINT_float_Ma
1209   \else
1210     \expandafter\XINT_float_Mc

```

```

1211 \fi {#1}{#2}%
1212 }%
1213 \def\XINT_float_Ma #1#2#3%
1214 {%
1215 \expandafter\XINT_float_Mb\expandafter
1216 {\the\numexpr #1-#2\expandafter\expandafter\expandafter}%
1217 \expandafter\expandafter\expandafter
1218 {\expandafter\xint_firstoftwo
1219 \romannumeral0\XINT_split_fromleft_loop {#2}{#3}W\W\W\W\W\W\W\W\Z
1220 }{#2}%
1221 }%
1222 \def\XINT_float_Mb #1#2#3#4#5#6% #2=A', #3=P+2, #4=P, #5=n, #6=B
1223 {%
1224 \expandafter\XINT_float_N\expandafter
1225 {\the\numexpr\xintLength{#6}\expandafter}\expandafter
1226 {\the\numexpr #3\expandafter}\expandafter
1227 {\the\numexpr #1+#5}%
1228 {#6}{#3}{#2}{#4}%
1229 }% long de B, P+2, n', B, |A'|=P+2, A', P
1230 \def\XINT_float_Mc #1#2#3#4#5#6%
1231 {%
1232 \expandafter\XINT_float_N\expandafter
1233 {\romannumeral0\xintlength{#6}}{#2}{#5}{#6}{#1}{#3}{#4}%
1234 }% long de B, P+2, n, B, |A|, A, P
1235 \def\XINT_float_N #1#2%
1236 {%
1237 \ifnum #1>#2
1238 \expandafter\XINT_float_O
1239 \else
1240 \expandafter\XINT_float_P
1241 \fi {#1}{#2}%
1242 }%
1243 \def\XINT_float_O #1#2#3#4%
1244 {%
1245 \expandafter\XINT_float_P\expandafter
1246 {\the\numexpr #2\expandafter}\expandafter
1247 {\the\numexpr #2\expandafter}\expandafter
1248 {\the\numexpr #3-#1+#2\expandafter\expandafter\expandafter}%
1249 \expandafter\expandafter\expandafter
1250 {\expandafter\xint_firstoftwo
1251 \romannumeral0\XINT_split_fromleft_loop {#2}{#4}W\W\W\W\W\W\W\W\Z
1252 }%
1253 }% |B|, P+2, n, B, |A|, A, P
1254 \def\XINT_float_P #1#2#3#4#5#6#7#8%
1255 {%
1256 \expandafter #8\expandafter {\the\numexpr #1-#5+#2-\xint_c_i}%
1257 {#6}{#4}{#7}{#3}%
1258 }% |B|-|A|+P+1, A, B, P, n
1259 \def\XINT_float_Q #1%
1260 {%
1261 \ifnum #1<\xint_c_
1262 \expandafter\XINT_float_Ri

```

```

1263   \else
1264     \expandafter\XINT_float_Rii
1265   \fi {#1}%
1266 }%
1267 \def\XINT_float_Ri #1#2#3%
1268 {%
1269   \expandafter\XINT_float_Sa
1270   \romannumeral0\xintiigo {#2}%
1271   {\XINT_dsx_addzerosnofuss {-#1}{#3}}\Z {#1}%
1272 }%
1273 \def\XINT_float_Rii #1#2#3%
1274 {%
1275   \expandafter\XINT_float_Sa
1276   \romannumeral0\xintiigo
1277   {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
1278 }%
1279 \def\XINT_float_Sa #1%
1280 {%
1281   \if #19%
1282     \xint_afterfi {\XINT_float_Sb\XINT_float_Wb }%
1283   \else
1284     \xint_afterfi {\XINT_float_Sb\XINT_float_Wa }%
1285   \fi #1%
1286 }%
1287 \def\XINT_float_Sb #1#2\Z #3#4%
1288 {%
1289   \expandafter\XINT_float_T\expandafter
1290   {\the\numexpr #4+\xint_c_i\expandafter}%
1291   \romannumeral-\`0\XINT_lenrord_loop 0{ }#2\Z\W\W\W\W\W\W\W\Z #1{#3}{#4}%
1292 }%
1293 \def\XINT_float_T #1#2#3%
1294 {%
1295   \ifnum #2>#1
1296     \xint_afterfi{\XINT_float_U\XINT_float_Xb}%
1297   \else
1298     \xint_afterfi{\XINT_float_U\XINT_float_Xa #3}%
1299   \fi
1300 }%
1301 \def\XINT_float_U #1#2%
1302 {%
1303   \ifnum #2<\xint_c_v
1304     \expandafter\XINT_float_Va
1305   \else
1306     \expandafter\XINT_float_Vb
1307   \fi #1%
1308 }%
1309 \def\XINT_float_Va #1#2\Z #3%
1310 {%
1311   \expandafter#1%
1312   \romannumeral0\expandafter\XINT_float_Wa
1313   \romannumeral0\XINT_rord_main { }#2%
1314   \xint_relax

```

```

1315      \xint_bye\xint_bye\xint_bye\xint_bye
1316      \xint_bye\xint_bye\xint_bye\xint_bye
1317      \xint_relax \Z
1318 }%
1319 \def\XINT_float_Vb #1#2\Z #3%
1320 {%
1321     \expandafter #1%
1322     \romannumeral0\expandafter #3%
1323     \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1324 }%
1325 \def\XINT_float_Wa #1{ #1.}%
1326 \def\XINT_float_Wb #1#2%
1327     {\if #11\xint_afterfi{ 10.}\else\xint_afterfi{ #1.#2}\fi }%
1328 \def\XINT_float_Xa #1\Z #2#3#4%
1329 {%
1330     \expandafter\XINT_float_Y\expandafter
1331     {\the\numexpr #3+#4-#2}{#1}%
1332 }%
1333 \def\XINT_float_Xb #1\Z #2#3#4%
1334 {%
1335     \expandafter\XINT_float_Y\expandafter
1336     {\the\numexpr #3+#4+\xint_c_i-#2}{#1}%
1337 }%
1338 \def\XINT_float_Y #1#2{ #2e#1}%

```

7.35 \xintPFloat

1.1

```

1339 \def\xintPFloat {\romannumeral0\xintpfloat }%
1340 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
1341 \def\XINT_pfloat_chkopt #1%
1342 {%
1343     \ifx [#1\expandafter\XINT_pfloat_opt
1344         \else\expandafter\XINT_pfloat_noopt
1345     \fi #1%
1346 }%
1347 \def\XINT_pfloat_noopt #1\xint_relax
1348 {%
1349     \expandafter\XINT_pfloat_a\expandafter\XINTdigits
1350     \romannumeral0\XINTinfloat [\XINTdigits]{#1}%
1351 }%
1352 \def\XINT_pfloat_opt [\xint_relax #1]##2%
1353 {%
1354     \expandafter\XINT_pfloat_a\expandafter {\the\numexpr #1\expandafter}%
1355     \romannumeral0\XINTinfloat [\numexpr #1\relax]##2}%
1356 }%
1357 \def\XINT_pfloat_a #1#2%
1358 {%
1359     \xint_UDzerominusfork
1360         #2-\XINT_pfloat_zero
1361         0#2\XINT_pfloat_neg
1362         0-{\XINT_pfloat_pos #2}%

```

```

1363 \krof {#1}%
1364 }%
1365 \def\XINT_pfloat_zero #1[#2]{ 0}%
1366 \def\XINT_pfloat_neg
1367 {\expandafter\xint_minus_thenstop\romannumeral0\XINT_pfloat_pos {}}%
1368 \def\XINT_pfloat_pos #1#2#3[#4]%
1369 {%
1370 \ifnum#4>0 \xint_dothis\XINT_pfloat_no\fi
1371 \ifnum#4>\numexpr-#2\relax \xint_dothis\XINT_pfloat_b\fi
1372 \ifnum#4>\numexpr-#2-\xint_c_v\relax \xint_dothis\XINT_pfloat_B\fi
1373 \xint_orthat\XINT_pfloat_no {#2}{#4}{#1#3}%
1374 }%
1375 \def\XINT_pfloat_no #1#2%
1376 {%
1377 \expandafter\XINT_pfloat_no_b\expandafter{\the\numexpr #2+#1-\xint_c_i\relax}%
1378 }%
1379 \def\XINT_pfloat_no_b #1#2{\XINT_pfloat_no_c #2e#1}%
1380 \def\XINT_pfloat_no_c #1{ #1.}%
1381 \def\XINT_pfloat_b #1#2#3%
1382 {\expandafter\XINT_pfloat_c
1383 \romannumeral0\expandafter\XINT_split_fromleft_loop
1384 \expandafter {\the\numexpr #1+#2-\xint_c_i}#3\W\W\W\W\W\W\W\W\Z }%
1385 \def\XINT_pfloat_c #1#2{ #1.#2}% #2 peut \^etre vide
1386 \def\XINT_pfloat_B #1#2#3%
1387 {\expandafter\XINT_pfloat_C
1388 \romannumeral0\XINT_dsx_zeroloop {\numexpr -#1-#2}{\Z {}}#3}%
1389 \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.

```

1390 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1391 \def\XINTinfloat [#1]#2%
1392 {%
1393 \expandafter\XINT_infloat_a\expandafter
1394 {\the\numexpr #1\expandafter}%
1395 \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
1396 }%
1397 \def\XINT_infloat_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1398 {%
1399 \XINT_infloat_fork #3\Z {#1}{#2}% #1 = precision, #2=n

```



```

1400 }%
1401 \def\XINT_infloat_fork #1%
1402 {%
1403   \xint_UDzerominusfork
1404   #1-\XINT_infloat_zero
1405   0#1\XINT_infloat_J
1406   0-\XINT_float_K #1}%
1407   \krof
1408 }%
1409 \def\XINT_infloat_zero #1\Z #2#3#4#5{ 0[0]}%

the 0[0] was stupidly changed to 0/1[0] in 1.09i, with the result that the Float addition would
crash when an operand was zero.

1410 \def\XINT_infloat_J {\expandafter-\romannumeral0\XINT_float_K }%
1411 \def\XINT_infloat_Q #1%
1412 {%
1413   \ifnum #1<\xint_c_
1414     \expandafter\XINT_infloat_Ri
1415   \else
1416     \expandafter\XINT_infloat_Rii
1417   \fi {#1}%
1418 }%
1419 \def\XINT_infloat_Ri #1#2#3%
1420 {%
1421   \expandafter\XINT_infloat_S\expandafter
1422   {\romannumeral0\xintiigo {#2}%
1423     {\XINT_dsx_addzerosnofuss {-#1}{#3}}}{#1}%
1424 }%
1425 \def\XINT_infloat_Rii #1#2#3%
1426 {%
1427   \expandafter\XINT_infloat_S\expandafter
1428   {\romannumeral0\xintiigo
1429     {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}}{#1}%
1430 }%
1431 \def\XINT_infloat_S #1#2#3%
1432 {%
1433   \expandafter\XINT_infloat_T\expandafter
1434   {\the\numexpr #3+\xint_c_i\expandafter}%
1435   \romannumeral-\`0\XINT_lenrord_loop 0{}#1\Z\W\W\W\W\W\W\W\Z
1436   {#2}%
1437 }%
1438 \def\XINT_infloat_T #1#2#3%
1439 {%
1440   \ifnum #2>#1
1441     \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wb}%
1442   \else
1443     \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wa #3}%
1444   \fi
1445 }%
1446 \def\XINT_infloat_U #1#2%
1447 {%
1448   \ifnum #2<\xint_c_v

```

```

1449 \expandafter\XINT_infloat_Va
1450 \else
1451 \expandafter\XINT_infloat_Vb
1452 \fi #1%
1453 }%
1454 \def\XINT_infloat_Va #1#2\Z
1455 {%
1456 \expandafter#1%
1457 \romannumeral0\XINT_rord_main {}#2%
1458 \xint_relax
1459 \xint_bye\xint_bye\xint_bye\xint_bye
1460 \xint_bye\xint_bye\xint_bye\xint_bye
1461 \xint_relax \Z
1462 }%
1463 \def\XINT_infloat_Vb #1#2\Z
1464 {%
1465 \expandafter #1%
1466 \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1467 }%
1468 \def\XINT_infloat_Wa #1\Z #2#3%
1469 {%
1470 \expandafter\XINT_infloat_X\expandafter
1471 {\the\numexpr #3+\xint_c_i-#2}{#1}%
1472 }%
1473 \def\XINT_infloat_Wb #1\Z #2#3%
1474 {%
1475 \expandafter\XINT_infloat_X\expandafter
1476 {\the\numexpr #3+\xint_c_ii-#2}{#1}%
1477 }%
1478 \def\XINT_infloat_X #1#2{ #2[#1]}%

```

7.37 *\xintAdd*

modified in v1.1. Et aussi 25 juin pour intercepter summand nul.

```

1479 \def\xintAdd {\romannumeral0\xintadd }%
1480 \def\xintadd #1{\expandafter\xint_fadd\romannumeral0\xintraw {#1}}%
1481 \def\xint_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1482 \def\XINT_fadd_Azero #1]{\xintraw }%
1483 \def\XINT_fadd_a #1/#2[#3]#4%
1484 {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1485 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1486 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1487 \def\XINT_fadd_c #1/#2[#3]#4%
1488 {%
1489 \expandafter\XINT_fadd_Aa\expandafter{\the\numexpr #4-#3}{#3}{#4}{#1}{#2}%
1490 }%
1491 \def\XINT_fadd_Aa #1%
1492 {%
1493 \ifcase\XINT_cntSgn #1\Z
1494 \expandafter\XINT_fadd_B
1495 \or
1496 \expandafter \XINT_fadd_Ba

```

7 Package *xintfrac* implementation

```

1497 \else
1498 \expandafter \XINT_fadd_Bb
1499 \fi {#1}%
1500 }%
1501 \def\XINT_fadd_B #1#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1502 \def\XINT_fadd_Ba #1#2#3#4#5#6#7%
1503 {%
1504 \expandafter\XINT_fadd_C\expandafter
1505 {\romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {#6}}%
1506 {#7}{#5}{#4}[#2]%
1507 }%
1508 \def\XINT_fadd_Bb #1#2#3#4#5#6#7%
1509 {%
1510 \expandafter\XINT_fadd_C\expandafter
1511 {\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {#4}}%
1512 {#5}{#7}{#6}[#3]%
1513 }%
1514 \def\XINT_fadd_C #1#2#3%
1515 {%
1516 \ifcase\romannumeral0\XINT_cmp_pre {#2}{#3} %<- intentional space here.
1517 \expandafter\XINT_fadd_eq
1518 \or\expandafter\XINT_fadd_D
1519 \else\expandafter\XINT_fadd_Da
1520 \fi {#2}{#3}{#1}%
1521 }%
1522 \def\XINT_fadd_eq #1#2#3#4#5%
1523 {%
1524 \expandafter\XINT_fadd_G
1525 \romannumeral0\xintiiadd {#3}{#4}/#1[#5]%
1526 }%
1527 \def\XINT_fadd_D #1#2%
1528 {%
1529 \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
1530 }%
1531 \def\XINT_fadd_E #1#2%
1532 {%
1533 \if0\XINT_Sgn #2\Z
1534 \expandafter\XINT_fadd_F
1535 \else\expandafter\XINT_fadd_K
1536 \fi {#1}%
1537 }%
1538 \def\XINT_fadd_F #1#2#3#4#5#6%
1539 {%
1540 \expandafter\XINT_fadd_G
1541 \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2[#6]%
1542 }%
1543 \def\XINT_fadd_Da #1#2%
1544 {%
1545 \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
1546 }%
1547 \def\XINT_fadd_Ea #1#2%
1548 {%

```

```

1549 \if0\XINT_Sgn #2\Z
1550 \expandafter\XINT_fadd_Fa
1551 \else\expandafter\XINT_fadd_K
1552 \fi {#1}%
1553 }%
1554 \def\XINT_fadd_Fa #1#2#3#4#5#6%
1555 {%
1556 \expandafter\XINT_fadd_G
1557 \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3%[#6]%
1558 }%
1559 \def\XINT_fadd_G #1{\if0#1\XINT_fadd_iszero\fi\space #1}%
1560 \def\XINT_fadd_K #1#2#3#4#5%
1561 {%
1562 \expandafter\XINT_fadd_L
1563 \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1564 {{#2}{#3}}%
1565 }%
1566 \def\XINT_fadd_L #1{\if0#1\XINT_fadd_iszero\fi \XINT_fadd_M #1}%
1567 \def\XINT_fadd_M #1.#2{\expandafter\XINT_fadd_N \expandafter
1568 {\romannumeral0\xintiimul #2}{#1}}%
1569 \def\XINT_fadd_N #1#2{ #2/#1}%
1570 \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.

```

1571 \def\xintSub {\romannumeral0\xintsub }%
1572 \def\xintsub #1{\expandafter\xint_fsub\romannumeral0\xintra {#1}}%
1573 \def\xint_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1574 \def\XINT_fsub_Azero #1]{\xintopp }%
1575 \def\XINT_fsub_a #1/#2[#3]#4%
1576 {\expandafter\XINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1577 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1578 #1-\XINT_fadd_Bzero
1579 0#1\XINT_fadd_c
1580 0-\XINT_fadd_c -#1}%
1581 \krof }%

```

7.39 \xintSum

```

1582 \def\xintSum {\romannumeral0\xintsum }%
1583 \def\xintsum #1{\xintsumexpr #1\relax }%
1584 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1585 \def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral-`0}%
1586 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1587 \def\XINT_fsum_loop_a #1#2%
1588 {%
1589 \expandafter\XINT_fsum_loop_b \romannumeral-`0#2\Z {#1}%
1590 }%
1591 \def\XINT_fsum_loop_b #1%
1592 {%
1593 \xint_gob_til_relax #1\XINT_fsum_finished\relax

```

```

1594 \XINT_fsum_loop_c #1%
1595 }%
1596 \def\XINT_fsum_loop_c #1\Z #2%
1597 {%
1598 \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1599 }%
1600 \def\XINT_fsum_finished #1\Z #2{ #2}%

```

7.40 \xintMul

modif 1.1 25-juin-14 pour vérifier plus tôt si nul

```

1601 \def\xintMul {\romannumeral0\xintmul }%
1602 \def\xintmul #1{\expandafter\xint_fmulo\romannumeral0\xintraw {#1}.}%
1603 \def\xint_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_a #1}%
1604 \def\XINT_fmulo_a #1[#2].#3%
1605 {\expandafter\XINT_fmulo_b\romannumeral0\xintraw {#3}#1[#2.]}%
1606 \def\XINT_fmulo_b #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1607 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]%
1608 {%
1609 \expandafter\XINT_fmulo_d
1610 \expandafter{\the\numexpr #3+#6\expandafter}%
1611 \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1612 {\romannumeral0\xintiimul {#4}{#1}}%
1613 }%
1614 \def\XINT_fmulo_d #1#2#3%
1615 {%
1616 \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1617 }%
1618 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1619 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

7.41 \xintSqr

1.1 modifs comme xintMul

```

1620 \def\xintSqr {\romannumeral0\xintsqr }%
1621 \def\xintsqr #1{\expandafter\xint_fsqr\romannumeral0\xintraw {#1}}%
1622 \def\xint_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1623 \def\XINT_fsqr_a #1/#2[#3]%
1624 {%
1625 \expandafter\XINT_fsqr_b
1626 \expandafter{\the\numexpr #3+#3\expandafter}%
1627 \expandafter{\romannumeral0\xintiisqr {#2}}%
1628 {\romannumeral0\xintiisqr {#1}}%
1629 }%
1630 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1631 \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.

```

1632 \def\xintPow {\romannumeral0\xintpow }%
1633 \def\xintpow #1%
1634 {%
1635   \expandafter\xint_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1636 }%
1637 \def\xint_fpow #1#2%
1638 {%
1639   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1640 }%
1641 \def\XINT_fpow_fork #1#2\Z
1642 {%
1643   \xint_UDzerominusfork
1644   #1-\XINT_fpow_zero
1645   0#1\XINT_fpow_neg
1646   0-{\XINT_fpow_pos #1}%
1647   \krof
1648   {#2}%
1649 }%
1650 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1651 \def\XINT_fpow_pos #1#2#3#4#5%
1652 {%
1653   \expandafter\XINT_fpow_pos_A\expandafter
1654   {\the\numexpr #1#2*#3\expandafter}\expandafter
1655   {\romannumeral0\xintiipow {#5}{#1#2}}%
1656   {\romannumeral0\xintiipow {#4}{#1#2}}%
1657 }%
1658 \def\XINT_fpow_neg #1#2#3#4%
1659 {%
1660   \expandafter\XINT_fpow_pos_A\expandafter
1661   {\the\numexpr -#1*#2\expandafter}\expandafter
1662   {\romannumeral0\xintiipow {#3}{#1}}%
1663   {\romannumeral0\xintiipow {#4}{#1}}%
1664 }%
1665 \def\XINT_fpow_pos_A #1#2#3%
1666 {%
1667   \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1668 }%
1669 \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 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`.

```

1670 \def\xintFac {\romannumeral0\xintfac }%
1671 \def\xintfac #1%
1672 {%
1673   \expandafter\XINT_fac_fork\expandafter{\the\numexpr \xintNum{#1}}%
1674 }%

```

7.44 \xintPrd

```

1675 \def\xintPrd {\romannumeral0\xintprd }%
1676 \def\xintprd #1{\xintprdexpr #1\relax }%
1677 \def\xintPrdExpr {\romannumeral0\xintprdexpr }%
1678 \def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral-`0}%
1679 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1680 \def\XINT_fprod_loop_a #1#2%
1681 {%
1682   \expandafter\XINT_fprod_loop_b \romannumeral-`0#2\Z {#1}%
1683 }%
1684 \def\XINT_fprod_loop_b #1%
1685 {%
1686   \xint_gob_til_relax #1\XINT_fprod_finished\relax
1687   \XINT_fprod_loop_c #1%
1688 }%
1689 \def\XINT_fprod_loop_c #1\Z #2%
1690 {%
1691   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1692 }%
1693 \def\XINT_fprod_finished #1\Z #2{ #2}%

```

7.45 \xintDiv

```

1694 \def\xintDiv {\romannumeral0\xintdiv }%
1695 \def\xintdiv #1%
1696 {%
1697   \expandafter\xint_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1698 }%
1699 \def\xint_fdiv #1#2%
1700   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}{#1}}%
1701 \def\XINT_fdiv_A #1#2#3#4#5#6%
1702 {%
1703   \expandafter\XINT_fdiv_B
1704   \expandafter{\the\numexpr #4-#1\expandafter}%
1705   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1706   {\romannumeral0\xintiimul {#3}{#5}}%
1707 }%
1708 \def\XINT_fdiv_B #1#2#3%
1709 {%
1710   \expandafter\XINT_fdiv_C
1711   \expandafter{#3}{#1}{#2}%
1712 }%
1713 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

7.46 \xintDivFloor

1.1

```

1714 \def\xintDivFloor      {\romannumeral0\xintdivfloor }%
1715 \def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%

```

7.47 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```

1716 \def\xintDivTrunc      {\romannumeral0\xintdivtrunc }%
1717 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

7.48 \xintDivRound

1.1

```

1718 \def\xintDivRound      {\romannumeral0\xintdivround }%
1719 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

7.49 \xintMod

1.1. \xintMod {q1}{q2} computes $q2 * t(q1/q2)$ with $t(q1/q2)$ equal to the truncated division of two arbitrary fractions $q1$ and $q2$. We put some efforts into minimizing the amount of computations.

```

1720 \def\xintMod {\romannumeral0\xintmod }%
1721 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1722 \def\XINT_mod_a #1#2.#3%
1723   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1724 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1725 {%
1726   \if0#2\xint_dothis\XINT_mod_divbyzero\fi
1727   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1728   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1729   \xint_orthat{\XINT_mod_bpos #1#2}%
1730 }%
1731 \def\XINT_mod_bpos #1%
1732 {%
1733   \xint_UDsignfork
1734     #1{\xintiiopp\XINT_mod_pos {}}%
1735     -{\XINT_mod_pos #1}%
1736   \krof
1737 }%
1738 \def\XINT_mod_bneg #1%
1739 {%
1740   \xint_UDsignfork
1741     #1{\xintiiopp\XINT_mod_pos {}}%
1742     -{\XINT_mod_pos #1}%
1743   \krof
1744 }%
1745 \def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
1746 \def\XINT_mod_aiszero #1.{ 0/1[0]}%
1747 \def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
1748 {%
1749   \expandafter\XINT_mod_pos_a
1750   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter

```



```

1751    {\romannumeral0\xintiimul {#6}{#3}}%      n fois u
1752    {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}% m fois u
1753    {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%    t fois n
1754 }%
1755 \def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

7.50 \XINTinFloatMod

Pour emploi dans *xintexpr* 1.1

```

1756 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
1757 \def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
1758     {\romannumeral0\XINTinfloat[#1]{#2}}}%
1759     {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}}%
1760 \def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
1761 \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.

```

1762 \def\xintIsOne {\romannumeral0\xintisone }%
1763 \def\xintisone #1{\expandafter\XINT_fracisone
1764     {\romannumeral0\xintraewithzeros{#1}\Z }%
1765 \def\XINT_fracisone #1/#2\Z
1766     {\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.

```

1767 \def\xintGeq {\romannumeral0\xintgeq }%
1768 \def\xintgeq #1%
1769 {%
1770     \expandafter\xint_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1771 }%
1772 \def\xint_fgeq #1#2%
1773 {%
1774     \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1775 }%
1776 \def\XINT_fgeq_A #1%
1777 {%
1778     \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1779     \XINT_fgeq_B #1%
1780 }%
1781 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1782 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1783 {%
1784     \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1785     \expandafter\XINT_fgeq_C\expandafter
1786     {\the\numexpr #7-#3\expandafter}\expandafter
1787     {\romannumeral0\xintiimul {#4#5}{#2}}}%

```

```

1788     {\romannumeral0\xintiimul {#6}{#1}}%
1789 }%
1790 \def\xINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1791 \def\xINT_fgeq_C #1#2#3%
1792 {%
1793     \expandafter\xINT_fgeq_D\expandafter
1794     {#3}{#1}{#2}%
1795 }%
1796 \def\xINT_fgeq_D #1#2#3%
1797 {%
1798     \expandafter\xINT_cntSgnFork\romannumeral-`0\expandafter\xINT_cntSgn
1799     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1800     { 0}{\xINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1801 }%
1802 \def\xINT_fgeq_E #1%
1803 {%
1804     \xint_UDsignfork
1805     #1\xINT_fgeq_Fd
1806     -{\xINT_fgeq_Fn #1}%
1807     \krof
1808 }%
1809 \def\xINT_fgeq_Fd #1\Z #2#3%
1810 {%
1811     \expandafter\xINT_fgeq_Fe\expandafter
1812     {\romannumeral0\xINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1813 }%
1814 \def\xINT_fgeq_Fe #1#2{\xINT_geq_pre {#2}{#1}}%
1815 \def\xINT_fgeq_Fn #1\Z #2#3%
1816 {%
1817     \expandafter\xINT_geq_pre\expandafter
1818     {\romannumeral0\xINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1819 }%

```

7.53 \xintMax

Rewritten completely in 1.08a.

```

1820 \def\xintMax {\romannumeral0\xintmax }%
1821 \def\xintmax #1%
1822 {%
1823     \expandafter\xint_fmax\expandafter {\romannumeral0\xintra {#1}}%
1824 }%
1825 \def\xint_fmax #1#2%
1826 {%
1827     \expandafter\xINT_fmax_A\romannumeral0\xintra {#2}#1%
1828 }%
1829 \def\xINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1830 {%
1831     \xint_UDsignsfork
1832     #1#5\xINT_fmax_minusminus
1833     -#5\xINT_fmax_firstneg
1834     #1-\xINT_fmax_secondneg
1835     --\xINT_fmax_nonneg_a

```

```

1836 \krof
1837 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1838 }%
1839 \def\XINT_fmax_minusminus --%
1840 {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmin_nonneg_b }%
1841 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1842 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1843 \def\XINT_fmax_nonneg_a #1#2#3#4%
1844 {%
1845 \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1846 }%
1847 \def\XINT_fmax_nonneg_b #1#2%
1848 {%
1849 \if0\romannumeral0\XINT_fgeq_A #1#2%
1850 \xint_afterfi{ #1}%
1851 \else \xint_afterfi{ #2}%
1852 \fi
1853 }%

```

7.54 \xintMaxof

```

1854 \def\xintMaxof {\romannumeral0\xintmaxof }%
1855 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral-`0#1\relax }%
1856 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}\Z }%
1857 \def\XINT_maxof_b #1\Z #2%
1858 {\expandafter\XINT_maxof_c\romannumeral-`0#2\Z {#1}\Z}%
1859 \def\XINT_maxof_c #1%
1860 {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%
1861 \def\XINT_maxof_d #1\Z
1862 {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1863 \def\XINT_maxof_e #1\Z #2\Z { #2}%

```

7.55 \xintMin

Rewritten completely in 1.08a.

```

1864 \def\xintMin {\romannumeral0\xintmin }%
1865 \def\xintmin #1%
1866 {%
1867 \expandafter\xint_fmin\expandafter {\romannumeral0\xintra{#1}}%
1868 }%
1869 \def\xint_fmin #1#2%
1870 {%
1871 \expandafter\XINT_fmin_A\romannumeral0\xintra{#2}#1%
1872 }%
1873 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1874 {%
1875 \xint_UDsignsfork
1876 #1#5\XINT_fmin_minusminus
1877 -#5\XINT_fmin_firstneg
1878 #1-\XINT_fmin_secondneg
1879 --\XINT_fmin_nonneg_a
1880 \krof
1881 #1#5{#2/#3[#4]}{#6/#7[#8]}%

```

```

1882 }%
1883 \def\XINT_fmin_minusminus --%
1884   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmax_nonneg_b }%
1885 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1886 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1887 \def\XINT_fmin_nonneg_a #1#2#3#4%
1888 {%
1889   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1890 }%
1891 \def\XINT_fmin_nonneg_b #1#2%
1892 {%
1893   \if0\romannumeral0\XINT_fgeq_A #1#2%
1894     \xint_afterfi{ #2}%
1895   \else \xint_afterfi{ #1}%
1896   \fi
1897 }%

```

7.56 \xintMinof

```

1898 \def\xintMinof      {\romannumeral0\xintminof }%
1899 \def\xintminof      #1{\expandafter\XINT_minof_a\romannumeral-`0#1\relax }%
1900 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}\Z }%
1901 \def\XINT_minof_b #1\Z #2%
1902   {\expandafter\XINT_minof_c\romannumeral-`0#2\Z {#1}\Z}%
1903 \def\XINT_minof_c #1%
1904   {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
1905 \def\XINT_minof_d #1\Z
1906   {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1907 \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.

```

1908 %\def\xintCmp {\romannumeral0\xintcmp }%
1909 \def\xintcmp #1%
1910 {%
1911   \expandafter\xint_fcmp\expandafter {\romannumeral0\xintra{#1}}%
1912 }%
1913 \def\xint_fcmp #1#2%
1914 {%
1915   \expandafter\XINT_fcmp_A\romannumeral0\xintra{#2}#1%
1916 }%
1917 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1918 {%
1919   \xint_UDsignsfork
1920     #1#5\XINT_fcmp_minusminus
1921     -#5\XINT_fcmp_firstneg
1922     #1-\XINT_fcmp_secondneg
1923     --\XINT_fcmp_nonneg_a
1924   \krof
1925   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1926 }%
1927 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%

```

```

1928 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1929 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1930 \def\XINT_fcmp_nonneg_a #1#2%
1931 {%
1932   \xint_UDzerosfork
1933   #1#2\XINT_fcmp_zerozero
1934   0#2\XINT_fcmp_firstzero
1935   #10\XINT_fcmp_secondzero
1936   00\XINT_fcmp_pos
1937   \krof
1938   #1#2%
1939 }%
1940 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}% 1.08b had some [ and ] here!!!
1941 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}% incredibly I never saw that until
1942 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}% preparing 1.09a.
1943 \def\XINT_fcmp_pos #1#2#3#4%
1944 {%
1945   \XINT_fcmp_B #1#3#2#4%
1946 }%
1947 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1948 {%
1949   \expandafter\XINT_fcmp_C\expandafter
1950   {\the\numexpr #6-#3\expandafter}\expandafter
1951   {\romannumeral0\xintiimul {#4}{#2}}%
1952   {\romannumeral0\xintiimul {#5}{#1}}%
1953 }%
1954 \def\XINT_fcmp_C #1#2#3%
1955 {%
1956   \expandafter\XINT_fcmp_D\expandafter
1957   {#3}{#1}{#2}%
1958 }%
1959 \def\XINT_fcmp_D #1#2#3%
1960 {%
1961   \expandafter\XINT_cntSgnFork\romannumeral-`0\expandafter\XINT_cntSgn
1962   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1963   { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1964 }%
1965 \def\XINT_fcmp_E #1%
1966 {%
1967   \xint_UDsignfork
1968   #1\XINT_fcmp_Fd
1969   -{\XINT_fcmp_Fn #1}%
1970   \krof
1971 }%
1972 \def\XINT_fcmp_Fd #1\Z #2#3%
1973 {%
1974   \expandafter\XINT_fcmp_Fe\expandafter
1975   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1976 }%
1977 \def\XINT_fcmp_Fe #1#2{\XINT_cmp_pre {#2}{#1}}%
1978 \def\XINT_fcmp_Fn #1\Z #2#3%
1979 {%

```

```

1980 \expandafter\XINT_cmp_pre\expandafter
1981 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1982 }%

```

7.58 \xintAbs

Simplified in 1.09i. (original macro had been written before \xintRaw)

```

1983 \def\xintAbs {\romannumeral0\xintabs }%
1984 \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)

```

1985 \def\xintOpp {\romannumeral0\xintopp }%
1986 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

7.60 \xintSgn

Simplified in 1.09i. (original macro had been written before \xintRaw)

```

1987 \def\xintSgn {\romannumeral0\xintsgn }%
1988 \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.

```

1989 \def\xintFloatAdd {\romannumeral0\xintfloatadd }%
1990 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax }%
1991 \def\XINTinFloatAdd {\romannumeral0\XINTinfloatadd }%
1992 \def\XINTinfloatadd #1{\XINT_fladd_chkopt \XINTinfloat #1\xint_relax }%
1993 \def\XINT_fladd_chkopt #1#2%
1994 {%
1995 \ifx [#2\expandafter\XINT_fladd_opt
1996 \else\expandafter\XINT_fladd_noopt
1997 \fi #1#2%
1998 }%
1999 \def\XINT_fladd_noopt #1#2\xint_relax #3%
2000 {%
2001 #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2002 }%
2003 \def\XINT_fladd_opt #1[\xint_relax #2]#3#4%
2004 {%
2005 #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{#4}}%
2006 }%
2007 \def\XINT_FL_Add #1#2%
2008 {%
2009 \expandafter\XINT_FL_Add_a\expandafter{\the\numexpr #1\expandafter}%
2010 \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2011 }%

```

```

2012 \def\XINT_FL_Add_a #1#2#3%
2013 {%
2014   \expandafter\XINT_FL_Add_b\romannumeral0\XINTinfloat [#1]{#3}#2{#1}%
2015 }%
2016 \def\XINT_FL_Add_b #1%
2017 {%
2018   \xint_gob_til_zero #1\XINT_FL_Add_zero 0\XINT_FL_Add_c #1%
2019 }%
2020 \def\XINT_FL_Add_c #1[#2]#3%
2021 {%
2022   \xint_gob_til_zero #3\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]#3%
2023 }%
2024 \def\XINT_FL_Add_d #1[#2]#3[#4]#5%
2025 {%
2026   \ifnum \numexpr #2-#4-#5>\xint_c_i
2027     \expandafter \xint_secondofthree_thenstop
2028   \else
2029     \ifnum \numexpr #4-#2-#5>\xint_c_i
2030       \expandafter\expandafter\expandafter\xint_thirdofthree_thenstop
2031     \fi
2032   \fi
2033   \xintadd {#1[#2]}{#3[#4]}%
2034 }%
2035 \def\XINT_FL_Add_zero 0\XINT_FL_Add_c 0[0]#1[#2]#3{#1[#2]}%
2036 \def\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]0[0]#3{#1[#2]}%

```

7.62 \xintFloatSub, \XINTinFloatSub

1.07

```

2037 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2038 \def\xintfloatsub #1{\XINT_fsub_chkopt \xintfloat #1\xint_relax }%
2039 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2040 \def\XINTinfloatsub #1{\XINT_fsub_chkopt \XINTinfloat #1\xint_relax }%
2041 \def\XINT_fsub_chkopt #1#2%
2042 {%
2043   \ifx [#2\expandafter\XINT_fsub_opt
2044     \else\expandafter\XINT_fsub_noopt
2045   \fi #1#2%
2046 }%
2047 \def\XINT_fsub_noopt #1#2\xint_relax #3%
2048 {%
2049   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{\xintOpp{#3}}}%
2050 }%
2051 \def\XINT_fsub_opt #1[\xint_relax #2]#3#4%
2052 {%
2053   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{\xintOpp{#4}}}%
2054 }%

```

7.63 \xintFloatMul, \XINTinFloatMul

1.07

```

2055 \def\xintFloatMul    {\romannumeral0\xintfloatmul}%
2056 \def\xintfloatmul    #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
2057 \def\XINTinFloatMul  {\romannumeral0\XINTinfloatmul }%
2058 \def\XINTinfloatmul  #1{\XINT_flmul_chkopt \XINTinfloat #1\xint_relax }%
2059 \def\XINT_flmul_chkopt #1#2%
2060 {%
2061   \ifx [#2\expandafter\XINT_flmul_opt
2062     \else\expandafter\XINT_flmul_noopt
2063   \fi #1#2%
2064 }%
2065 \def\XINT_flmul_noopt #1#2\xint_relax #3%
2066 {%
2067   #1[\XINTdigits]{\XINT_FL_Mul {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2068 }%
2069 \def\XINT_flmul_opt #1[\xint_relax #2]#3#4%
2070 {%
2071   #1[#2]{\XINT_FL_Mul {#2+\xint_c_ii}{#3}{#4}}%
2072 }%
2073 \def\XINT_FL_Mul #1#2%
2074 {%
2075   \expandafter\XINT_FL_Mul_a\expandafter{\the\numexpr #1\expandafter}%
2076   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2077 }%
2078 \def\XINT_FL_Mul_a #1#2#3%
2079 {%
2080   \expandafter\XINT_FL_Mul_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2081 }%
2082 \def\XINT_FL_Mul_b #1[#2]#3[#4]{\xintE{\xintiMul {#1}{#3}}{#2+#4}}%

```

7.64 \xintFloatDiv, \XINTinFloatDiv

1.07

```

2083 \def\xintFloatDiv    {\romannumeral0\xintfloatdiv}%
2084 \def\xintfloatdiv    #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
2085 \def\XINTinFloatDiv  {\romannumeral0\XINTinfloatdiv }%
2086 \def\XINTinfloatdiv  #1{\XINT_fldiv_chkopt \XINTinfloat #1\xint_relax }%
2087 \def\XINT_fldiv_chkopt #1#2%
2088 {%
2089   \ifx [#2\expandafter\XINT_fldiv_opt
2090     \else\expandafter\XINT_fldiv_noopt
2091   \fi #1#2%
2092 }%
2093 \def\XINT_fldiv_noopt #1#2\xint_relax #3%
2094 {%
2095   #1[\XINTdigits]{\XINT_FL_Div {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2096 }%
2097 \def\XINT_fldiv_opt #1[\xint_relax #2]#3#4%
2098 {%
2099   #1[#2]{\XINT_FL_Div {#2+\xint_c_ii}{#3}{#4}}%
2100 }%
2101 \def\XINT_FL_Div #1#2%
2102 {%

```



```

2103 \expandafter\XINT_FL_Div_a\expandafter{\the\numexpr #1\expandafter}%
2104 \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2105 }%
2106 \def\XINT_FL_Div_a #1#2#3%
2107 {%
2108 \expandafter\XINT_FL_Div_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2109 }%
2110 \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.

```

2111 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2112 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax }%
2113 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
2114 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloat #1\xint_relax }%
2115 \def\XINT_flpow_chkopt #1#2%
2116 {%
2117 \ifx [#2\expandafter\XINT_flpow_opt
2118 \else\expandafter\XINT_flpow_noopt
2119 \fi
2120 #1#2%
2121 }%
2122 \def\XINT_flpow_noopt #1#2\xint_relax #3%
2123 {%
2124 \expandafter\XINT_flpow_checkB_start\expandafter
2125 {\the\numexpr #3\expandafter}\expandafter
2126 {\the\numexpr \XINTdigits}{#2}{#1[\XINTdigits]}%
2127 }%
2128 \def\XINT_flpow_opt #1[\xint_relax #2]#3#4%
2129 {%
2130 \expandafter\XINT_flpow_checkB_start\expandafter
2131 {\the\numexpr #4\expandafter}\expandafter
2132 {\the\numexpr #2}{#3}{#1[#2]}%
2133 }%
2134 \def\XINT_flpow_checkB_start #1{\XINT_flpow_checkB_a #1\Z }%
2135 \def\XINT_flpow_checkB_a #1%
2136 {%
2137 \xint_UDzerominusfork
2138 #1-\XINT_flpow_BisZero
2139 0#1{\XINT_flpow_checkB_b 1}%
2140 0-{\XINT_flpow_checkB_b 0#1}%
2141 \krof
2142 }%
2143 \def\XINT_flpow_BisZero \Z #1#2#3{#3{1/1[0]}}%
2144 \def\XINT_flpow_checkB_b #1#2\Z #3%
2145 {%
2146 \expandafter\XINT_flpow_checkB_c \expandafter
2147 {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
2148 }%
2149 \def\XINT_flpow_checkB_c #1#2%

```

7 Package *xintfrac* implementation

```

2150 {%
2151   \expandafter\XINT_flpow_checkB_d \expandafter
2152   {\the\numexpr \expandafter\xintLength\expandafter
2153     {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2154 }%
2155 \def\XINT_flpow_checkB_d #1#2#3#4%
2156 {%
2157   \expandafter \XINT_flpow_a
2158   \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2159 }%
2160 \def\XINT_flpow_a #1%
2161 {%
2162   \xint_UDzerominusfork
2163   #1-\XINT_flpow_zero
2164   0#1{\XINT_flpow_b 1}%
2165   0-{\XINT_flpow_b 0#1}%
2166   \krof
2167 }%
2168 \def\XINT_flpow_b #1#2[#3]#4#5%
2169 {%
2170   \XINT_flpow_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2171   {#1*\ifodd #5 1\else 0\fi}%
2172 }%
2173 \def\XINT_flpow_zero [#1]#2#3#4#5%

```

xint is not equipped to signal infinity, the 2^{31} will provoke deliberately a number too big and arithmetic overflow in `\XINT_float_Xb`

```

2174 {%
2175   \if #41\xint_afterfi {\xintError:DivisionByZero #5{1[2147483648]}}%
2176   \else \xint_afterfi {#5{0[0]}}\fi
2177 }%
2178 \def\XINT_flpow_loopI #1%
2179 {%
2180   \ifnum #1=\xint_c_i\XINT_flpow_ItoIII\fi
2181   \ifodd #1
2182     \expandafter\XINT_flpow_loopI_odd
2183   \else
2184     \expandafter\XINT_flpow_loopI_even
2185   \fi
2186   {#1}%
2187 }%
2188 \def\XINT_flpow_ItoIII\fi #1\fi #2#3#4#5%
2189 {%
2190   \fi\expandafter\XINT_flpow_III\the\numexpr #5\relax #3%
2191 }%
2192 \def\XINT_flpow_loopI_even #1#2#3%
2193 {%
2194   \expandafter\XINT_flpow_loopI\expandafter
2195   {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2196   {#3{#2}{#2}}{#3}%
2197 }%
2198 \def\XINT_flpow_loopI_odd #1#2#3%

```

```

2199 {%
2200   \expandafter\XINT_flpow_loopII\expandafter
2201   {\the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2202   {\#3{\#2}{\#2}}{\#3}{\#2}%
2203 }%
2204 \def\XINT_flpow_loopII #1%
2205 {%
2206   \ifnum #1 = \xint_c_i\XINT_flpow_IItoIII\fi
2207   \ifodd #1
2208     \expandafter\XINT_flpow_loopII_odd
2209   \else
2210     \expandafter\XINT_flpow_loopII_even
2211   \fi
2212   {\#1}%
2213 }%
2214 \def\XINT_flpow_loopII_even #1#2#3%
2215 {%
2216   \expandafter\XINT_flpow_loopII\expandafter
2217   {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2218   {\#3{\#2}{\#2}}{\#3}%
2219 }%
2220 \def\XINT_flpow_loopII_odd #1#2#3#4%
2221 {%
2222   \expandafter\XINT_flpow_loopII_odda\expandafter
2223   {\#3{\#2}{\#4}}{\#1}{\#2}{\#3}%
2224 }%
2225 \def\XINT_flpow_loopII_odda #1#2#3#4%
2226 {%
2227   \expandafter\XINT_flpow_loopII\expandafter
2228   {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2229   {\#4{\#3}{\#3}}{\#4}{\#1}%
2230 }%
2231 \def\XINT_flpow_IItoIII\fi #1\fi #2#3#4#5#6%
2232 {%
2233   \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2234   #4{\#3}{\#5}%
2235 }%
2236 \def\XINT_flpow_III #1#2[#3]#4%
2237 {%
2238   \expandafter\XINT_flpow_IIIend\expandafter
2239   {\the\numexpr\if #41-\fi#3\expandafter}%
2240   \xint_UDzerofork
2241     #4{\#2}%
2242     0{\{1/#2}}%
2243   \krof #1%
2244 }%
2245 \def\XINT_flpow_IIIend #1#2#3#4%
2246 {%
2247   \xint_UDzerofork
2248   #3{\#4{\#2[#1]}}%
2249   0{\#4{-#2[#1]}}%
2250   \krof

```

2251 }%

7.66 `\xintFloatPower`, `\XINTinFloatPower`

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain.

```

2252 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2253 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
2254 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower}%
2255 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloat #1\xint_relax }%
2256 \def\XINT_flpower_chkopt #1#2%
2257 {%
2258   \ifx [#2\expandafter\XINT_flpower_opt
2259     \else\expandafter\XINT_flpower_noopt
2260   \fi
2261   #1#2%
2262 }%
2263 \def\XINT_flpower_noopt #1#2\xint_relax #3%
2264 {%
2265   \expandafter\XINT_flpower_checkB_start\expandafter
2266     {\the\numexpr \XINTdigits\expandafter}\expandafter
2267     {\romannumeral0\xintnum{#3}}{#2}{#1[\XINTdigits]}%
2268 }%
2269 \def\XINT_flpower_opt #1[\xint_relax #2]#3#4%
2270 {%
2271   \expandafter\XINT_flpower_checkB_start\expandafter
2272     {\the\numexpr #2\expandafter}\expandafter
2273     {\romannumeral0\xintnum{#4}}{#3}{#1[#2]}%
2274 }%
2275 \def\XINT_flpower_checkB_start #1#2{\XINT_flpower_checkB_a #2\Z {#1}}%
2276 \def\XINT_flpower_checkB_a #1%
2277 {%
2278   \xint_UDzerominusfork
2279   #1-\XINT_flpower_BisZero
2280   0#1{\XINT_flpower_checkB_b 1}%
2281   0-{\XINT_flpower_checkB_b 0#1}%
2282   \krof
2283 }%
2284 \def\XINT_flpower_BisZero \Z #1#2#3{#3{1/1[0]}}%
2285 \def\XINT_flpower_checkB_b #1#2\Z #3%
2286 {%
2287   \expandafter\XINT_flpower_checkB_c \expandafter
2288     {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
2289 }%
2290 \def\XINT_flpower_checkB_c #1#2%
2291 {%
2292   \expandafter\XINT_flpower_checkB_d \expandafter
2293     {\the\numexpr \expandafter\xintlength\expandafter
2294       {\the\numexpr #1*20/\xint_c_iii }+ #1+ #2+\xint_c_i }%
2295 }%
2296 \def\XINT_flpower_checkB_d #1#2#3#4%
2297 {%
2298   \expandafter \XINT_flpower_a

```

```

2299 \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2300 }%
2301 \def\XINT_flpower_a #1%
2302 {%
2303 \xint_UDzerominusfork
2304 #1-\XINT_flpow_zero
2305 0#1{\XINT_flpower_b 1}%
2306 0-{\XINT_flpower_b 0#1}%
2307 \krof
2308 }%
2309 \def\XINT_flpower_b #1#2[#3]#4#5%
2310 {%
2311 \XINT_flpower_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2312 {#1*\xintiiOdd {#5}}%
2313 }%
2314 \def\XINT_flpower_loopI #1%
2315 {%
2316 \if1\XINT_isOne {#1}\XINT_flpower_ItoIII\fi
2317 \if1\xintiiOdd{#1}%
2318 \expandafter\expandafter\expandafter\XINT_flpower_loopI_odd
2319 \else
2320 \expandafter\expandafter\expandafter\XINT_flpower_loopI_even
2321 \fi
2322 \expandafter {\romannumeral0\xinthalft{#1}}%
2323 }%
2324 \def\XINT_flpower_ItoIII\fi #1\fi\expandafter #2#3#4#5%
2325 {%
2326 \fi\expandafter\XINT_flpow_III \the\numexpr #5\relax #3%
2327 }%
2328 \def\XINT_flpower_loopI_even #1#2#3%
2329 {%
2330 \expandafter\XINT_flpower_toI\expandafter {#3{#2}{#2}}{#1}{#3}%
2331 }%
2332 \def\XINT_flpower_loopI_odd #1#2#3%
2333 {%
2334 \expandafter\XINT_flpower_toII\expandafter {#3{#2}{#2}}{#1}{#3}{#2}%
2335 }%
2336 \def\XINT_flpower_toI #1#2{\XINT_flpower_loopI {#2}{#1}}%
2337 \def\XINT_flpower_toII #1#2{\XINT_flpower_loopII {#2}{#1}}%
2338 \def\XINT_flpower_loopII #1%
2339 {%
2340 \if1\XINT_isOne {#1}\XINT_flpower_IItoIII\fi
2341 \if1\xintiiOdd{#1}%
2342 \expandafter\expandafter\expandafter\XINT_flpower_loopII_odd
2343 \else
2344 \expandafter\expandafter\expandafter\XINT_flpower_loopII_even
2345 \fi
2346 \expandafter {\romannumeral0\xinthalft{#1}}%
2347 }%
2348 \def\XINT_flpower_loopII_even #1#2#3%
2349 {%
2350 \expandafter\XINT_flpower_toII\expandafter

```

```

2351      {#3{#2}{#2}}{#1}{#3}%
2352 }%
2353 \def\XINT_flpower_loopII_odd #1#2#3#4%
2354 {%
2355     \expandafter\XINT_flpower_loopII_odda\expandafter
2356     {#3{#2}{#4}}{#2}{#3}{#1}%
2357 }%
2358 \def\XINT_flpower_loopII_odda #1#2#3#4%
2359 {%
2360     \expandafter\XINT_flpower_toII\expandafter
2361     {#3{#2}{#2}}{#4}{#3}{#1}%
2362 }%
2363 \def\XINT_flpower_IItoIII\fi #1\fi\expandafter #2#3#4#5#6%
2364 {%
2365     \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2366     #4{#3}{#5}%
2367 }%

```

7.67 \xintFloatSqrt, \XINTinFloatSqrt

1.08

```

2368 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqrt }%
2369 \def\xintfloatsqrt      #1{\XINT_flsqrt_chkopt \xintfloat #1\xint_relax }%
2370 \def\XINTinFloatSqrt    {\romannumeral0\XINTinfloatsqrt }%
2371 \def\XINTinfloatsqrt    #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
2372 \def\XINT_flsqrt_chkopt #1#2%
2373 {%
2374     \ifx [#2\expandafter\XINT_flsqrt_opt
2375     \else\expandafter\XINT_flsqrt_noopt
2376     \fi #1#2%
2377 }%
2378 \def\XINT_flsqrt_noopt #1#2\xint_relax
2379 {%
2380     #1[\XINTdigits]{\XINT_FL_sqrt \XINTdigits {#2}}%
2381 }%
2382 \def\XINT_flsqrt_opt #1[\xint_relax #2]#3%
2383 {%
2384     #1[#2]{\XINT_FL_sqrt {#2}{#3}}%
2385 }%
2386 \def\XINT_FL_sqrt #1%
2387 {%
2388     \ifnum\numexpr #1<\xint_c_xviii
2389         \xint_afterfi {\XINT_FL_sqrt_a\xint_c_xviii}%
2390     \else
2391         \xint_afterfi {\XINT_FL_sqrt_a {#1+\xint_c_i}}%
2392     \fi
2393 }%
2394 \def\XINT_FL_sqrt_a #1#2%
2395 {%
2396     \expandafter\XINT_FL_sqrt_checkifzeroorneg
2397     \romannumeral0\XINTinfloat [#1]{#2}%
2398 }%

```

```

2399 \def\XINT_FL_sqrt_checkifzeroorneg #1%
2400 {%
2401   \xint_UDzerominusfork
2402   #1-\XINT_FL_sqrt_iszero
2403   0#1\XINT_FL_sqrt_isneg
2404   0-\XINT_FL_sqrt_b #1}%
2405   \krof
2406 }%
2407 \def\XINT_FL_sqrt_iszero #1[#2]{0[0]}%
2408 \def\XINT_FL_sqrt_isneg #1[#2]{\xintError:RootOfNegative 0[0]}%
2409 \def\XINT_FL_sqrt_b #1[#2]%
2410 {%
2411   \ifodd #2
2412     \xint_afterfi{\XINT_FL_sqrt_c 01}%
2413   \else
2414     \xint_afterfi{\XINT_FL_sqrt_c {}0}%
2415   \fi
2416   {#1}{#2}%
2417 }%
2418 \def\XINT_FL_sqrt_c #1#2#3#4%
2419 {%
2420   \expandafter\XINT_flsqrt\expandafter {\the\numexpr #4-#2}{#3#1}%
2421 }%
2422 \def\XINT_flsqrt #1#2%
2423 {%
2424   \expandafter\XINT_sqrt_a
2425   \expandafter{\romannumeral0\xintlength {#2}}\XINT_flsqrt_big_d {#2}{#1}%
2426 }%
2427 \def\XINT_flsqrt_big_d #1#2%
2428 {%
2429   \ifodd #2
2430     \expandafter\expandafter\expandafter\XINT_flsqrt_big_eB
2431   \else
2432     \expandafter\expandafter\expandafter\XINT_flsqrt_big_eA
2433   \fi
2434   \expandafter {\the\numexpr (#2-\xint_c_i)/\xint_c_ii }{#1}%
2435 }%
2436 \def\XINT_flsqrt_big_eA #1#2#3%
2437 {%
2438   \XINT_flsqrt_big_eA_a #3\Z {#2}{#1}{#3}%
2439 }%
2440 \def\XINT_flsqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
2441 {%
2442   \XINT_flsqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
2443 }%
2444 \def\XINT_flsqrt_big_eA_b #1#2%
2445 {%
2446   \expandafter\XINT_flsqrt_big_f
2447   \romannumeral0\XINT_flsqrt_small_e {#2001}{#1}%
2448 }%
2449 \def\XINT_flsqrt_big_eB #1#2#3%
2450 {%

```

7 Package *xintfrac* implementation

```

2451 \XINT_flsqrt_big_eB_a #3\Z {#2}{#1}{#3}%
2452 }%
2453 \def\XINT_flsqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
2454 {%
2455 \XINT_flsqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
2456 }%
2457 \def\XINT_flsqrt_big_eB_b #1#2\Z #3%
2458 {%
2459 \expandafter\XINT_flsqrt_big_f
2460 \romannumeral0\XINT_flsqrt_small_e {#30001}{#1}%
2461 }%
2462 \def\XINT_flsqrt_small_e #1#2%
2463 {%
2464 \expandafter\XINT_flsqrt_small_f\expandafter
2465 {\the\numexpr #1*#1-#2-\xint_c_i}{#1}%
2466 }%
2467 \def\XINT_flsqrt_small_f #1#2%
2468 {%
2469 \expandafter\XINT_flsqrt_small_g\expandafter
2470 {\the\numexpr (#1+#2)/(2*#2)-\xint_c_i }{#1}{#2}%
2471 }%
2472 \def\XINT_flsqrt_small_g #1%
2473 {%
2474 \ifnum #1>\xint_c_
2475 \expandafter\XINT_flsqrt_small_h
2476 \else
2477 \expandafter\XINT_flsqrt_small_end
2478 \fi
2479 {#1}%
2480 }%
2481 \def\XINT_flsqrt_small_h #1#2#3%
2482 {%
2483 \expandafter\XINT_flsqrt_small_f\expandafter
2484 {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
2485 {\the\numexpr #3-#1}%
2486 }%
2487 \def\XINT_flsqrt_small_end #1#2#3%
2488 {%
2489 \expandafter\space\expandafter
2490 {\the\numexpr \xint_c_i+#3*\xint_c_x^iv-
2491 (#2*\xint_c_x^iv+#3)/(\xint_c_ii*#3)}%
2492 }%
2493 \def\XINT_flsqrt_big_f #1%
2494 {%
2495 \expandafter\XINT_flsqrt_big_fa\expandafter
2496 {\romannumeral0\xintiisqr {#1}}{#1}%
2497 }%
2498 \def\XINT_flsqrt_big_fa #1#2#3#4%
2499 {%
2500 \expandafter\XINT_flsqrt_big_fb\expandafter
2501 {\romannumeral0\XINT_dsx_addzerosnofuss
2502 {\numexpr #3-\xint_c_viii\relax}{#2}}%

```



```

2503     {\romannumeral0\xintiisub
2504     {\XINT_dsx_addzerosnofuss
2505     {\numexpr \xint_c_ii*(#3-\xint_c_viii)\relax}{#1}}{#4}}%
2506     {#3}}%
2507 }%
2508 \def\xint_flsqrt_big_fb #1#2%
2509 {%
2510     \expandafter\xint_flsqrt_big_g\expandafter {#2}{#1}%
2511 }%
2512 \def\xint_flsqrt_big_g #1#2%
2513 {%
2514     \expandafter\xint_flsqrt_big_j
2515     \romannumeral0\xintiidivision
2516     {#1}{\romannumeral0\xint_dbl_pos #2\R\R\R\R\R\R\R\Z \W\W\W\W\W\W }{#2}}%
2517 }%
2518 \def\xint_flsqrt_big_j #1%
2519 {%
2520     \if0\xint_Sgn #1\Z
2521         \expandafter \xint_flsqrt_big_end_a
2522     \else \expandafter \xint_flsqrt_big_k
2523     \fi {#1}%
2524 }%
2525 \def\xint_flsqrt_big_k #1#2#3%
2526 {%
2527     \expandafter\xint_flsqrt_big_l\expandafter
2528     {\romannumeral0\xint_sub_pre {#3}}{#1}}%
2529     {\romannumeral0\xintiiaadd {#2}{\romannumeral0\xint_sqr {#1}}}%
2530 }%
2531 \def\xint_flsqrt_big_l #1#2%
2532 {%
2533     \expandafter\xint_flsqrt_big_g\expandafter
2534     {#2}{#1}%
2535 }%
2536 \def\xint_flsqrt_big_end_a #1#2#3#4#5%
2537 {%
2538     \expandafter\xint_flsqrt_big_end_b\expandafter
2539     {\the\numexpr -#4+#5/\xint_c_ii\expandafter}\expandafter
2540     {\romannumeral0\xintiisub
2541     {\XINT_dsx_addzerosnofuss {#4}}{#3}}%
2542     {\xintHalf{\xintiiQuo{\XINT_dsx_addzerosnofuss {#4}}{#2}}{#3}}}%
2543 }%
2544 \def\xint_flsqrt_big_end_b #1#2{#2[ #1]}%
2545 \XINT_restorecatcodes_endinput%

```

8 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection	202	.7	<code>\xintRationalSeries</code>	205
.2	Package identification	202	.8	<code>\xintRationalSeriesX</code>	206
.3	<code>\xintSeries</code>	203	.9	<code>\xintFxFtPowerSeries</code>	207
.4	<code>\xintiSeries</code>	203	.10	<code>\xintFxFtPowerSeriesX</code>	208
.5	<code>\xintPowerSeries</code>	204	.11	<code>\xintFloatPowerSeries</code>	208
.6	<code>\xintPowerSeriesX</code>	205	.12	<code>\xintFloatPowerSeriesX</code>	210

The commenting is currently (2015/09/17) very sparse.

8.1 Catcodes, ε -TeX and reload detection

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2015/09/12 v1.1c 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_iseries_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%
87 \def\XINT_iseries_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_iseries_exit \fi
90   \expandafter\XINT_iseries_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\xintiiadd {#2}{#4{#1}}}%
93   {#3}{#4}%
94 }%
95 \def\XINT_iseries_exit \fi #1#2#3#4#5#6#7#8%
96 {%
97   \fi\xint_gobble_ii #6%
98 }%

```

8.5 \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators. (this was at a time \xintAdd always multiplied denominators, by the way) The Horner scheme for polynomial evaluation is used in 1.04, this cures the denominator problem and drastically improves the efficiency of the macro. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

99 \def\xintPowerSeries {\romannumeral0\xintpowerseries }%
100 \def\xintpowerseries #1#2%
101 {%
102   \expandafter\XINT_powseries\expandafter
103   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
104 }%
105 \def\XINT_powseries #1#2#3#4%
106 {%
107   \ifnum #2<#1
108     \xint_afterfi { 0/1[0]}%
109   \else
110     \xint_afterfi
111     {\XINT_powseries_loop_i {#3{#2}}{#1}{#2}{#3}{#4}}%
112   \fi
113 }%
114 \def\XINT_powseries_loop_i #1#2#3#4#5%
115 {%
116   \ifnum #3>#2 \else\XINT_powseries_exit_i\fi
117   \expandafter\XINT_powseries_loop_ii\expandafter
118   {\the\numexpr #3-1\expandafter}\expandafter
119   {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
120 }%
121 \def\XINT_powseries_loop_ii #1#2#3#4%
122 {%
123   \expandafter\XINT_powseries_loop_i\expandafter
124   {\romannumeral0\xintadd {#4{#1}}{#2}}{#3}{#1}{#4}%
125 }%

```

```

126 \def\XINT_powseries_exit_i\fi #1#2#3#4#5#6#7#8#9%
127 {%
128   \fi \XINT_powseries_exit_ii #6{#7}%
129 }%
130 \def\XINT_powseries_exit_ii #1#2#3#4#5#6%
131 {%
132   \xintmul{\xintPow {#5}{#6}}{#4}%
133 }%

```

8.6 \xintPowerSeriesX

Same as `\xintPowerSeries` except for the initial expansion of the `x` parameter. Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

134 \def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
135 \def\xintpowerseriesx #1#2%
136 {%
137   \expandafter\XINT_powseriesx\expandafter
138   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
139 }%
140 \def\XINT_powseriesx #1#2#3#4%
141 {%
142   \ifnum #2<#1
143     \xint_afterfi { 0/1[0]}%
144   \else
145     \xint_afterfi
146     {\expandafter\XINT_powseriesx_pre\expandafter
147      {\romannumeral-`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,radiofunction,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 \xintFxFtPowerSeries

I am not two happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to \numexpr.

```

210 \def\xintFxFtPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213     \expandafter\XINT_fppowseries\expandafter
214     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218     \ifnum #2<#1
219         \xint_afterfi { 0}%
220     \else
221         \xint_afterfi
222         {\expandafter\XINT_fppowseries_loop_pre\expandafter
223          {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224          {#1}{#4}{#2}{#3}{#5}%
225         }%
226     \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230     \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231     \expandafter\XINT_fppowseries_loop_i\expandafter
232     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233     {\romannumeral0\xintitrunc {#6}{\xintMul {#5}{#2}}{#1}}%
234     {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237     {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241     \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242     \expandafter\XINT_fppowseries_loop_ii\expandafter
243     {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244     {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%

```

```

247 {%
248   \expandafter\XINT_fppowseries_loop_i\expandafter
249   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250   {\romannumeral0\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}%
251   {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
254   {\fi \expandafter\XINT_fppowseries_exit_ii }%
255 \def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257   \xinttrunc {#7}
258   {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

8.10 \xintFxFtPowerSeriesX

a,b,coeff,x,D

Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

260 \def\xintFxFtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
261 \def\xintfxptpowerseriesx #1#2%
262 {%
263   \expandafter\XINT_fppowseriesx\expandafter
264   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
265 }%
266 \def\XINT_fppowseriesx #1#2#3#4#5%
267 {%
268   \ifnum #2<#1
269     \xint_afterfi { 0}%
270   \else
271     \xint_afterfi
272     {\expandafter \XINT_fppowseriesx_pre \expandafter
273     {\romannumeral-`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 `\xintFxFtPowerSeries`; temporarily I just adapted the code to the case of floats.

```

283 \def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
284 \def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint_relax }%
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, ε - \TeX and reload detection	211	.16	<code>\xintiGctoF</code>	223
.2	Package identification	212	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	224
.3	<code>\xintCFrac</code>	212	.18	<code>\xintiCstoCv</code>	225
.4	<code>\xintGCFrac</code>	213	.19	<code>\xintGctoCv</code>	225
.5	<code>\xintGGCFrac</code>	215	.20	<code>\xintiGctoCv</code>	227
.6	<code>\xintGctoGCx</code>	216	.21	<code>\xintFtoCv</code>	228
.7	<code>\xintFtoCs</code>	216	.22	<code>\xintFtoCCv</code>	228
.8	<code>\xintFtoCx</code>	217	.23	<code>\xintCntoF</code>	228
.9	<code>\xintFtoC</code>	217	.24	<code>\xintGCntoF</code>	229
.10	<code>\xintFtoGC</code>	218	.25	<code>\xintCntoCs</code>	230
.11	<code>\xintFGtoC</code>	218	.26	<code>\xintCntoGC</code>	230
.12	<code>\xintFtoCC</code>	219	.27	<code>\xintGCntoGC</code>	231
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	220	.28	<code>\xintCstoGC</code>	232
.14	<code>\xintiCstoF</code>	221	.29	<code>\xintGctoGC</code>	232
.15	<code>\xintGctoF</code>	221			

The commenting is currently (2015/09/17) 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, ε - \TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2015/09/12 v1.1c 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\xintraawithzeros {#1}\Z
73 \relax\relax
74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77 \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
78 \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82 \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86 \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90 \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}%
94 \def\XINT_cfrac_loop_a
95 {%
96 \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100 \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104 \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108 \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111 {\XINT_cfrac_T #5#6{#2#4\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114 \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#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 }%
122 \def\xintgcfrac #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\xintrawwithzeros {#1}\Z
246 }%
247 \def\XINT_ftc_A #1/#2\Z
248 {%
249   \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
250 }%
251 \def\XINT_ftc_B #1#2%
252 {%
253   \XINT_ftc_C #2.{#1}%
254 }%
255 \def\XINT_ftc_C #1%
256 {%
257   \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
258 }%
259 \def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
260 \def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
261 \def\XINT_ftc_loop_a
262 {%
263   \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_div_prepare
264 }%
265 \def\XINT_ftc_loop_d #1#2%
266 {%
267   \XINT_ftc_loop_e #2.{#1}%
268 }%
269 \def\XINT_ftc_loop_e #1%

```



```

270 {%
271   \xint_gob_til_zero #1\xint_ftc_loop_exit0\XINT_ftc_loop_f #1%
272 }%
273 \def\XINT_ftc_loop_f #1.#2#3#4%
274 {%
275   \XINT_ftc_loop_a {#1}{#3}{#1}{#4#2, }% 1.09m has an added space here
276 }%
277 \def\xint_ftc_loop_exit0\XINT_ftc_loop_f #1.#2#3#4{ #4#2}%

```

9.8 \xintFtoCx

```

278 \def\xintFtoCx {\romannumeral0\xintftocx }%
279 \def\xintftocx #1#2%
280 {%
281   \expandafter\XINT_ftcx_A\romannumeral0\xintrawithzeros {#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 \backslash xintFtoGC

```

316 \def\xintFtoGC {\romannumeral0\xintftogc}%
317 \def\xintftogc {\xintftocx {+1/}}%

```

9.11 \backslash xintFGtoC

New with 1.09m of 2014/02/26. Computes the common initial coefficients for the two fractions f and g , and outputs them as a sequence of braced items.

```

318 \def\xintFGtoC {\romannumeral0\xintfgtoc}%
319 \def\xintfgtoc#1%
320 {%
321   \expandafter\XINT_fgtc_a\romannumeral0\xintraewithzeros {#1}\Z
322 }%
323 \def\XINT_fgtc_a #1/#2\Z #3%
324 {%
325   \expandafter\XINT_fgtc_b\romannumeral0\xintraewithzeros {#3}\Z #1/#2\Z { }%
326 }%
327 \def\XINT_fgtc_b #1/#2\Z
328 {%
329   \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
330 }%
331 \def\XINT_fgtc_c #1#2#3#4/#5\Z
332 {%
333   \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
334     {#4}{#5}{#5}{#1}{#2}{#3}%
335 }%
336 \def\XINT_fgtc_d #1#2#3#4#5#6#7%
337 {%
338   \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
339     {\xint_thirdofthree}%
340 }%
341 \def\XINT_fgtc_da #1#2#3#4#5#6#7%
342 {%
343   \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7}{#1}}%
344 }%
345 \def\XINT_fgtc_e #1%
346 {%
347   \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\xintrawwithzeros {#1}}%
367 }%
368 \def\XINT_ftcc_A #1%
369 {%
370   \expandafter\XINT_ftcc_B
371   \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1[0]}}\Z {#1[0]}%
372 }%
373 \def\XINT_ftcc_B #1/#2\Z
374 {%
375   \expandafter\XINT_ftcc_C\expandafter {\romannumeral0\xintiiquo {#1}{#2}}%
376 }%
377 \def\XINT_ftcc_C #1#2%
378 {%
379   \expandafter\XINT_ftcc_D\romannumeral0\xintsub {#2}{#1}\Z {#1}%
380 }%
381 \def\XINT_ftcc_D #1%
382 {%
383   \xint_UDzerominusfork
384   #1-\XINT_ftcc_integer
385   0#1\XINT_ftcc_En
386   0-{\XINT_ftcc_Ep #1}%
387   \krof
388 }%
389 \def\XINT_ftcc_Ep #1\Z #2%
390 {%
391   \expandafter\XINT_ftcc_loop_a\expandafter
392   {\romannumeral0\xintdiv {1[0]}{#1}}{#2+1/}%
393 }%
394 \def\XINT_ftcc_En #1\Z #2%
395 {%
396   \expandafter\XINT_ftcc_loop_a\expandafter
397   {\romannumeral0\xintdiv {1[0]}{#1}}{#2+-1/}%
398 }%
399 \def\XINT_ftcc_integer #1\Z #2{ #2}%
400 \def\XINT_ftcc_loop_a #1%
401 {%
402   \expandafter\XINT_ftcc_loop_b
403   \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1}}\Z {#1}%
404 }%
405 \def\XINT_ftcc_loop_b #1/#2\Z
406 {%
407   \expandafter\XINT_ftcc_loop_c\expandafter
408   {\romannumeral0\xintiiquo {#1}{#2}}%

```

```

409 }%
410 \def\XINT_ftcc_loop_c #1#2%
411 {%
412   \expandafter\XINT_ftcc_loop_d
413   \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
414 }%
415 \def\XINT_ftcc_loop_d #1%
416 {%
417   \xint_UDzerominusfork
418   #1-\XINT_ftcc_end
419   0#1\XINT_ftcc_loop_N
420   0-\XINT_ftcc_loop_P #1}%
421 \krof
422 }%
423 \def\XINT_ftcc_end #1\Z #2#3{ #3#2}%
424 \def\XINT_ftcc_loop_P #1\Z #2#3%
425 {%
426   \expandafter\XINT_ftcc_loop_a\expandafter
427   {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+1/}%
428 }%
429 \def\XINT_ftcc_loop_N #1\Z #2#3%
430 {%
431   \expandafter\XINT_ftcc_loop_a\expandafter
432   {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+-1/}%
433 }%

```

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 {%

```

```

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%

```

```

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\xintraawithzeros {#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\xintraawithzeros {#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 \xintiGctoF

```

563 \def\xintiGctoF {\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{\xintrawithzeros {#4/#5}}% 1.09b removes [0]
```

9.17 `\xintCtoCv`, `\xintCstoCv`

1.09m uses `\xintCSVtoList` on the argument of `\xintCstoCv` to allow spaces also before the commas. The original `\xintCstoCv` code became the one of the new `\xintCtoF` dealing with a braced rather than comma separated list.

```
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\xintrawithzeros {#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\xintrawithzeros {#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\xintiadd {#5}{\XINT_Mul {#1}{#3}}}%
675   {\romannumeral0\xintiadd {#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}%

```

```

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\xintrawithzeros {#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\xintrawithzeros {#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_l\expandafter
806 {\romannumeral0\xintraewithzeros {#1/#2}}%
807 }%
808 \def\XINT_igctcv_loop_l #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, ε -TeX and reload detection	238
10.2	Package identification	239
10.3	Locking and unlocking	239
10.4	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	239
10.5	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	239
10.6	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	239
10.7	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiiepr</code> , <code>\xinttheexpr</code> , etc...	239
10.8	<code>\xintthe</code>	240
10.9	<code>\xintthecoords</code>	240
10.10	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	240
10.11	<code>\xinteval</code> , <code>\xintiieval</code>	240
10.12	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	240
10.13	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	241
10.14	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code>	241
10.15	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiexpr</code>	242
10.16	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	242
10.16.1	<code>\XINT::_end</code>	242
10.16.2	<code>\xintCSV::csv</code>	242
10.16.3	<code>\xintSPRaw</code> , <code>\xintSPRaw::csv</code>	242
10.16.4	<code>\xintIsTrue::csv</code>	243
10.16.5	<code>\xintRound::csv</code>	243
10.16.6	<code>\XINTinFloat::csv</code>	243
10.16.7	<code>\xintPFloat::csv</code>	243
10.17	<code>\XINT_expr_getnext</code> : fetching some number then an operator	244
10.18	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	245
10.18.1	Integral part	245
10.18.2	Fractional part	246
10.18.3	Scientific notation	246
10.18.4	Hexadecimal numbers	247
10.18.5	Function and variable names	248
10.19	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	249
10.20	Opening and closing parentheses, square brackets for lists, the <code>^C</code> for omit and abort within seq or rseq	250
10.21	<code> </code> , <code> </code> , <code>&</code> , <code>&&</code> , <code><</code> , <code>></code> , <code>=</code> , <code>==</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>][</code> , <code>][:</code> , <code>:</code> , <code>^C</code> , and <code>++</code> operators	252
10.21.1	The <code> </code> , <code>&</code> , <code>xor</code> , <code><</code> , <code>></code> , <code>=</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , and <code>]</code> operators	252
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	254
10.21.3	The <code>'and'</code> , <code>'or'</code> , <code>'xor'</code> , and <code>'mod'</code> as infix operator words	256
10.21.4	The <code> </code> , <code>&&</code> , <code>**</code> , <code>**[</code> , <code>]**</code> operators as synonyms	256
10.21.5	List selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	257
10.22	Macros for a..b list generation	259
10.22.1	<code>\xintSeq::csv</code>	259
10.22.2	<code>\xintiiSeq::csv</code>	260
10.23	Macros for a..[d]..b list generation	261
10.23.1	<code>\xintSeqA::csv</code> , <code>\xintiiSeqA::csv</code> , <code>\XINTinFloatSeqA::csv</code>	261
10.23.2	<code>\xintSeqB::csv</code>	261

Contents

10.23.3	<code>\xintiiSeqB::csv</code>	262
10.23.4	<code>\XINTinFloatSeqB::csv</code>	262
10.24	The comma as binary operator	263
10.25	The minus as prefix operator of variable precedence level	263
10.26	? as two-way and ?? as three-way conditionals with braced branches	264
10.27	! as postfix factorial operator	264
10.28	The A/B[N] mechanism	265
10.29	For variables	265
10.29.1	Defining variables	265
10.29.2	Letters as dummy variables; the nil list	266
10.29.3	The omit and abort constructs	266
10.29.4	The @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	266
10.30	For functions	267
10.31	The bool, togl, protect, unknown, and break "functions"	267
10.32	seq and the implementation of dummy variables	268
10.32.1	<code>\XINT_expr_onlitteral_seq</code>	268
10.32.2	<code>\XINT_expr_onlitteral_seq_a</code>	268
10.32.3	<code>\XINT_isbalanced_a</code> for <code>\XINT_expr_onlitteral_seq_a</code>	269
10.32.4	<code>\XINT_allexpr_func_seqx</code> , <code>\XINT_allexpr_func_subx</code>	269
10.32.5	break, abort, omit within seq	270
10.32.6	<code>\XINT_expr_seq:_A</code>	270
10.32.7	add and mul, <code>\XINT_expr_onlitteral_add</code> , <code>\XINT_expr_onlitteral_mul</code>	270
10.32.8	<code>\XINT_expr_func_opx</code> , <code>\XINT_flexpr_func_opx</code> , <code>\XINT_iiexpr_func_opx</code>	271
10.32.9	<code>\XINT_expr_op:_a</code> , ...	271
10.32.10	subs, <code>\XINT_expr_onlitteral_subs</code>	271
10.33	rseq	272
10.33.1	<code>\XINT_expr_rseqx</code>	272
10.33.2	<code>\XINT_expr_rseqy</code>	272
10.33.3	<code>\XINT_expr_rseq:_a</code> etc...	272
10.33.4	<code>\XINT_expr_rseq:_A</code> etc...	273
10.34	rrseq	273
10.34.1	<code>\XINT_expr_rrseqx</code>	273
10.34.2	<code>\XINT_expr_rrseqy</code>	274
10.34.3	<code>\XINT_expr_rrseq:_a</code> etc...	274
10.34.4	<code>\XINT_expr_rrseq:_A</code> etc...	274
10.35	iter	275
10.35.1	<code>\XINT_expr_iterx</code>	275
10.35.2	<code>\XINT_expr_itory</code>	275
10.35.3	<code>\XINT_expr_iter:_a</code> etc...	276
10.35.4	<code>\XINT_expr_iter:_A</code> etc...	276
10.36	Macros handling csv lists for functions with multiple comma separated arguments in expressions	277
10.36.1	<code>\xintANDof:csv</code>	277
10.36.2	<code>\xintORof:csv</code>	277
10.36.3	<code>\xintXORof:csv</code>	277
10.36.4	Generic csv routine	278
10.36.5	<code>\xintMaxof:csv</code> , <code>\xintiiMaxof:csv</code>	278
10.36.6	<code>\xintMinof:csv</code> , <code>\xintiiMinof:csv</code>	278
10.36.7	<code>\xintSum:csv</code> , <code>\xintiiSum:csv</code>	278
10.36.8	<code>\xintPrd:csv</code> , <code>\xintiiPrd:csv</code>	278
10.36.9	<code>\xintGCDof:csv</code> , <code>\xintLCMof:csv</code>	279
10.36.10	<code>\xintiiGCDof:csv</code> , <code>\xintiiLCMof:csv</code>	279
10.36.11	<code>\XINTinFloatdigits</code> , <code>\XINTinFloatSqrtdigits</code>	279

10.36.12	<code>\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv</code>	279
10.36.13	<code>\XINTinFloatSum:csv, \XINTinFloatPrd:csv</code>	279
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	279
10.38	f-expandable versions of the SeqB::csv routines, for <code>\xintNewExpr</code>	285
10.38.1	<code>\xintSeqB:f:csv</code>	285
10.38.2	<code>\xintiiSeqB:f:csv</code>	286
10.38.3	<code>\XINTinFloatSeqB:f:csv</code>	286
10.39	<code>\xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr</code>	287
10.39.1	<code>\xintApply::csv</code>	287
10.39.2	<code>\xintApply:::csv</code>	287
10.39.3	<code>\XINT_expr_RApply::csv, \XINT_expr_LApply::csv, \XINT_expr_RLApply:::csv</code>	288
10.39.4	Mysterious stuff	288

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 **1.1c** of [2015/09/12].

10.1 Catcodes, ε -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \def\z {\endgroup}%
13 \expandafter\let\expandafter\x\csname ver@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 \XINTprovidespackage
56 \ProvidesPackage{xintexpr}%
57 [2015/09/12 v1.1c 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_xintthe!}%

```

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\xintiiepr {\romannumeral0\xintiieval }%
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_iiepr_print\romannumeral0\xintbareiieval }%

```

10.8 \xintthe

```

85 \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 craziness 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!

```

86 \def\xintthecoords #1{\romannumeral-`0\expandafter\expandafter\expandafter
87   \XINT_thecoords_a
88   \expandafter\xint_gobble_iii\romannumeral0#1}%
89 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
90   {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
91     \romannumeral-`0#1#2,!,!,^ \endcsname }%
92 \def\XINT_thecoords_b #1#2,#3#4,%
93   {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
94 \def\XINT_thecoords_c #1^{}%

```

10.10 \xintbareeval, \xintbarefloateval, \xintbareiieval

```

95 \def\xintbareeval
96   {\expandafter\XINT_expr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
97 \def\xintbarefloateval
98   {\expandafter\XINT_flexpr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
99 \def\xintbareiieval
100  {\expandafter\XINT_iiepr_until_end_a\romannumeral-`0\XINT_expr_getnext }%

```

10.11 \xintieval, \xintiieval

```

101 \def\xintieval {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
102 \def\xintiieval {\expandafter\XINT_iiepr_wrap\romannumeral0\xintbareiieval }%

```

10.12 \xintieval, \XINT_iexpr_wrap

Optional argument since 1.1

```

103 \def\xintieval #1%
104   {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
105 \def\XINT_iexpr_noopt
106   {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%

```



```

107 \def\XINT_iexpr_withopt [#1]%
108 {%
109   \expandafter\XINT_iexpr_wrap\expandafter
110   {\the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter}%
111   \romannumeral0\xintbareeval
112 }%
113 \def\XINT_iexpr_wrap #1#2%
114 {%
115   \expandafter\XINT_expr_wrap
116   \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
117 }%

```

10.13 \xintfloateval, \XINT_flexpr_wrap, \XINT_flexpr_print

Optional argument since 1.1

```

118 \def\xintfloateval #1%
119 {%
120   \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
121   \fi #1%
122 }%
123 \def\XINT_flexpr_noopt
124 {%
125   \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
126   \romannumeral0\xintbarefloateval
127 }%
128 \def\XINT_flexpr_withopt_a [#1]%
129 {%
130   \expandafter\XINT_flexpr_withopt_b\expandafter
131   {\the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter}%
132   \romannumeral0\xintbarefloateval
133 }%
134 \def\XINT_flexpr_withopt_b #1#2%
135 {%
136   \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
137   \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
138 }%
139 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
140 \def\XINT_flexpr_print #1%
141 {%
142   \expandafter\xintPFloat::csv
143   \romannumeral-\`0\expandafter\XINT_expr_unlock_sp\string #1!%
144 }%
145 \catcode`: 12
146   \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
147 \catcode`: 11

```

10.14 \xintboolexpr, \xinttheboolexpr

```

148 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
149   \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
150 \def\xinttheboolexpr    {\romannumeral-\`0\expandafter\expandafter\expandafter
151   \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%

```

```
152 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%
```

10.15 \xintifboolexpr, \xintifboolfloatexpr, \xintifboolliexpr

Do not work with comma separated expressions.

```
153 \def\xintifboolexpr      #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
154 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
155 \def\xintifboolliexpr    #1{\romannumeral0\xintifnotzero {\xinttheiexpr #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

```
156 \def\XINT::_end #1,#2{\xint_gobble_i #2}%
```

10.16.2 \xintCSV::csv

pour \xinttheiexpr. 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.

```
157 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral-`0#1,^,%}
158 \def\XINT_csv::_a {\XINT_csv::_b {}}%
159 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral-`0#2,{#1}}%
160 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
161 \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.

```
162 \def\xintSPRaw    {\romannumeral0\xintspraw }%
163 \def\xintspraw    #1{\expandafter\XINT_spraw\romannumeral-`0#1[\W]}%
164 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
165 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
166 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
167 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral-`0#1,^,%}
168 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
169 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral-`0#2,{#1}}%
170 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
171     \if ^#1\xint_dothis\XINT::_end\fi
172     \xint_orthat\XINT_spraw::_d #1}%
173 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
174 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

10.16.4 `\xintIsTrue::csv`

```

175 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\xint_istrue::_a\romannumeral-`0#1,^,%}
176 \def\xint_istrue::_a {\xint_istrue::_b {}}%
177 \def\xint_istrue::_b #1#2,{\expandafter\xint_istrue::_c \romannumeral-`0#2,{#1}}%
178 \def\xint_istrue::_c #1{\if ,#1\xint_dothis\xint_istrue::_e\fi
179         \if ^#1\xint_dothis\xint_istrue::_d #1}%
180         \xint_orthat\xint_istrue::_d #1}%
181 \def\xint_istrue::_d #1,{\expandafter\xint_istrue::_e\romannumeral0\xintisnotzero {#1},}%
182 \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`.

```

183 \def\xint_:::end #1,#2#3{\xint_gobble_i #3}%
184 \def\xintRound::csv #1#2{\romannumeral0\expandafter\xint_round::_b\expandafter
185     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,%}
186 \def\xint_round::_b #1#2#3,{\expandafter\xint_round::_c \romannumeral-`0#3,{#1}{#2}}%
187 \def\xint_round::_c #1{\if ,#1\xint_dothis\xint_round::_e\fi
188         \if ^#1\xint_dothis\xint_:::end\fi
189         \xint_orthat\xint_round::_d #1}%
190 \def\xint_round::_d #1,#2{%
191     \expandafter\xint_round::_e\romannumeral0\ifnum#2>\xint_c_
192     \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
193 \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.

```

194 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
195     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,%}
196 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
197 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
198         \if ^#1\xint_dothis\XINT_:::end\fi
199         \xint_orthat\XINT_infloat::_d #1}%
200 \def\XINT_infloat::_d #1,#2%
201     {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
202 \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`

```

203 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter

```

```

204   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,%}
205 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral-`0#3,{#1}{#2}}%
206 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
207       \if ^#1\xint_dothis\XINT::_:_end\fi
208       \xint_orthat\XINT_pfloat::_d #1}%
209 \def\XINT_pfloat::_d #1,#2%
210   {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint_relax #2]{#1},{#2}}%
211 \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.

```

212 \def\XINT_expr_getnext #1%
213 {%
214   \expandafter\XINT_expr_getnext_a\romannumeral-`0#1%
215 }%
216 \def\XINT_expr_getnext_a #1%
217 {% screens out sub-expressions and \count or \dimen registers/variables
218   \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
219   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
220     \expandafter\XINT_expr_countetc
221   \else
222     \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
223     \fi
224     #1%
225 }%
226 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%
227 \def\XINT_expr_countetc #1%
228 {%
229   \ifx\count#1\else\ifx#1\dimen\else\ifx#1\numexpr\else\ifx#1\dimexpr\else
230     \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else
231       \XINT_expr_unpackvar
232     \fi\fi\fi\fi\fi\fi\fi
233     \expandafter\XINT_expr_getnext\number #1%
234 }%
235 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getnext\number #1%
236   {\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
237 \begingroup
238 \lccode`*=`#
239 \lowercase{\endgroup
240 \def\XINT_expr_getnextfork #1{%
241   \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
242   \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
243   \if#1+\xint_dothis \XINT_expr_getnext \fi
244   \if#1.\xint_dothis {\XINT_expr_scandec_II\XINT_expr_infracpart}\fi
245   \if#1-\xint_dothis -\fi
246   \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
247   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
248 }%
249 \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

```

250 \catcode96 11 % `
251 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
252 {%
253   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
254   \if `#1\xint_dothis {\XINT_expr_onlitteral_}\fi
255   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_scandec_I\fi
256   \xint_orthat \XINT_expr_scanfunc #1%
257 }%
258 \catcode96 12 % `
259 \def\XINT_expr_scandec_I
260 {%
261   \expandafter\XINT_expr_getop\romannumeral-`0\expandafter
262   \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
263   \romannumeral-`0\XINT_expr_scanintpart_b
264 }%
265 \def\XINT_expr_scandec_II
266 {%
267   \expandafter\XINT_expr_getop\romannumeral-`0\expandafter
268   \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
269   \romannumeral-`0\XINT_expr_scanfracpart_b
270 }%

```

10.18.1 Integral part

```

271 \def\XINT_expr_scanintpart_a #1%
272 {% careful that ! has catcode letter here
273   \ifcat \relax #1\xint_dothis{!!#1}\fi % stops the scan
274   \if e#1\xint_dothis{\expandafter\XINT_expr_inexppart
275     \romannumeral-`0\XINT_expr_scanexppart_a e}\fi
276   \if E#1\xint_dothis{\expandafter\XINT_expr_inexppart
277     \romannumeral-`0\XINT_expr_scanexppart_a e}\fi

  \if @#1\xint_dothis{!*#1}\fi tacit multiplication later

  \if _#1\xint_dothis{!*#1}\fi tacit multiplication for variables

278   \ifcat a#1\xint_dothis{!!*#1}\fi % includes subexpressions (#1!= letter)
279   \xint_orthat {\expandafter\XINT_expr_scanintpart_aa\string #1}%
280 }%
281 \def\XINT_expr_scanintpart_aa #1%
282 {%
283   \if .#1\xint_dothis\XINT_expr_scandec_transition\fi
284   \ifnum \xint_c_ix<1#1 \xint_dothis\XINT_expr_scanintpart_b\fi
285   \xint_orthat {!!}#1%
286 }%
287 \def\XINT_expr_scanintpart_b #1#2%
288 {%
289   \expandafter #1\romannumeral-`0\expandafter
290   \XINT_expr_scanintpart_a\romannumeral-`0#2%
291 }%
292 \def\XINT_expr_scandec_transition .#1%
293 {%

```

```

294 \expandafter\XINT_expr_scandec_trans_a\romannumeral-`0#1%
295 }%
296 \def\XINT_expr_scandec_trans_a #1%
297 {%
298 \if .#1\xint_dothis{!!..}\fi
299 \xint_orthat {\expandafter\XINT_expr_infracpart
300 \romannumeral-`0\XINT_expr_scanfracpart_a #1}%
301 }%

```

10.18.2 Fractional part

```

302 \def\XINT_expr_scanfracpart_a #1%
303 {%
304 \ifcat \relax #1\xint_dothis{e!#1}\fi % stops the scan
305 \if e#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
306 \if E#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
307 \ifcat a#1\xint_dothis{e!*#1}\fi % and also the case of subexpressions (!)
308 \xint_orthat {\expandafter\XINT_expr_scanfracpart_aa\string #1}%
309 }%
310 \def\XINT_expr_scanfracpart_aa #1%
311 {%
312 \ifnum \xint_c_ix<1#1
313 \expandafter\XINT_expr_scanfracpart_b
314 \else
315 \xint_afterfi {e!}%
316 \fi
317 #1%
318 }%
319 \def\XINT_expr_scanfracpart_b #1#2%
320 {%
321 \expandafter #1\romannumeral-`0\expandafter
322 \XINT_expr_scanfracpart_a\romannumeral-`0#2%
323 }%

```

10.18.3 Scientific notation

```

324 \def\XINT_expr_scanexppart_a #1#2%
325 {%
326 \expandafter #1\romannumeral-`0\expandafter
327 \XINT_expr_scanexppart_b\romannumeral-`0#2%
328 }%
329 \def\XINT_expr_scanexppart_b #1%
330 {%
331 \ifcat \relax #1\xint_dothis{0!#1}\fi % stops the scan (incorrect syntax)
332 \ifcat a#1\xint_dothis{0!*#1}\fi % idem
333 \if +#1\xint_dothis {\XINT_expr_scanexppart_a +}\fi
334 \if -#1\xint_dothis {\XINT_expr_scanexppart_a -}\fi
335 \xint_orthat {\expandafter\XINT_expr_scanexppart_c\string #1}%
336 }%
337 \def\XINT_expr_scanexppart_c #1%
338 {%
339 \ifnum \xint_c_ix<1#1
340 \expandafter\XINT_expr_scanexppart_d
341 \else
342 \expandafter !%

```

```

343 \fi
344 #1%
345 }%
346 \def\XINT_expr_scanexppart_d #1#2%
347 {%
348 \expandafter #1\romannumeral-\`0\expandafter
349 \XINT_expr_scanexppart_e\romannumeral-\`0#2%
350 }%
351 \def\XINT_expr_scanexppart_e #1%
352 {%
353 \ifcat \relax #1\xint_dothis{!#1}\fi % stops the scan
354 \ifcat a#1\xint_dothis{!#1}\fi % idem
355 \xint_orthat {\expandafter\XINT_expr_scanexppart_f\string #1}%
356 }%
357 \def\XINT_expr_scanexppart_f #1%
358 {%
359 \ifnum \xint_c_ix<1#1
360 \expandafter\XINT_expr_scanexppart_d
361 \else
362 \expandafter !%
363 \fi
364 #1%
365 }%

```

10.18.4 Hexadecimal numbers

```

366 \def\XINT_expr_scanhex_I #1%
367 {%
368 \expandafter\XINT_expr_getop\romannumeral-\`0\expandafter
369 \XINT_expr_lockscan\expandafter\XINT_expr_inhex
370 \romannumeral-\`0\XINT_expr_scanhexI_a
371 }%
372 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
373 {%
374 \if#2I\xintHexToDec{#1}%
375 \else
376 \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
377 [\the\numexpr-4*\xintLength{#3}]%
378 \fi
379 }%
380 \def\XINT_expr_scanhexI_a #1%
381 {%
382 \ifcat #1\relax\xint_dothis{.I;!#1}\fi
383 \ifx !#1\xint_dothis{.I;!#1}\fi % tacit multiplication
384 \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
385 }%
386 \def\XINT_expr_scanhexI_aa #1%
387 {%
388 \if\ifnum`#1>`/
389 \ifnum`#1>`9
390 \ifnum`#1>`@
391 \ifnum`#1>`F
392 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
393 \expandafter\XINT_expr_scanhexI_b

```

```

394   \else
395     \if .#1%
396       \expandafter\xint_firstoftwo
397     \else % gather what we got so far, leave catcode 12 #1 in stream
398       \expandafter\xint_secondoftwo
399     \fi
400     {\expandafter\XINT_expr_scanhex_transition}%
401     {\xint_afterfi {.I;!}}%
402   \fi
403   #1%
404 }%
405 \def\XINT_expr_scanhexI_b #1#2%
406 {%
407   \expandafter #1\romannumeral-`0\expandafter
408   \XINT_expr_scanhexI_a\romannumeral-`0#2%
409 }%
410 \def\XINT_expr_scanhex_transition .#1%
411 {%
412   \expandafter.\expandafter.\romannumeral-`0\expandafter
413   \XINT_expr_scanhexII_a\romannumeral-`0#1%
414 }%
415 \def\XINT_expr_scanhexII_a #1%
416 {%
417   \ifcat #1\relax\xint_dothis{;!#1}\fi
418   \ifx      !#1\xint_dothis{;!#1}\fi % tacit multiplication
419   \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
420 }%
421 \def\XINT_expr_scanhexII_aa #1%
422 {%
423   \if\ifnum`#1>`/
424     \ifnum`#1>`9
425     \ifnum`#1>`@
426     \ifnum`#1>`F
427     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
428     \expandafter\XINT_expr_scanhexII_b
429   \else
430     \xint_afterfi {;!}%
431   \fi
432   #1%
433 }%
434 \def\XINT_expr_scanhexII_b #1#2%
435 {%
436   \expandafter #1\romannumeral-`0\expandafter
437   \XINT_expr_scanhexII_a\romannumeral-`0#2%
438 }%

```

10.18.5 Function and variable names

```

439 \def\XINT_expr_scanfunc
440 {%
441   \expandafter\XINT_expr_func\romannumeral-`0\XINT_expr_scanfunc_a
442 }%
443 \def\XINT_expr_scanfunc_a #1#2%
444 {%

```



```

445 \expandafter #1\romannumeral-`0\expandafter\XINT_expr_scanfunc_b\romannumeral-`0#2%
446 }%
447 \def\XINT_expr_scanfunc_b #1%
448 {%
449 \ifx !#1\xint_dothis{\xint_firstoftwo{(_*!)}\fi
450 \ifcat \relax#1\xint_dothis{(_)\fi
451 \if (#1\xint_dothis{\xint_firstoftwo{(`)}\fi
452 \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
453 \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
454 \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
455 \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
456 \xint_orthat {(_}%
457 #1%
458 }%
459 \def\XINT_expr_func #1(#2%
460 {% #2=` pour une fonction, #2=_ pour une variable
461 \if #2`\ifcsname XINT_expr_var_#1\endcsname
462 \expandafter\expandafter\expandafter\xint_thirdofthree
463 \fi\fi
464 \xint_firstoftwo {\xint_c_xviii #2{#1}}{\xint_c_xviii _{#1}*}%
465 }%

```

10.19 *\XINT_expr_getop*: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

```

466 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
467 {%
468 \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral-`0#2%
469 }%
470 \catcode`* 11
471 \def\XINT_expr_getop_a #1#2%
472 {%
473 \ifx \relax #2\xint_dothis\xint_firstofthree\fi
474 \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
475 \if _#2\xint_dothis \xint_secondofthree\fi
476 \if @#2\xint_dothis \xint_secondofthree\fi
477 \if (#2\xint_dothis \xint_secondofthree\fi
478 \ifx !#2\xint_dothis \xint_secondofthree\fi
479 \xint_orthat \xint_thirdofthree
480 {\XINT_expr_foundend #1}%
481 {\XINT_expr_precedence_* *#1#2}% tacit multiplication
482 {\XINT_expr_getop_b #2#1}%
483 }%
484 \catcode`* 12
485 \def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.

```

? and : a special syntax in *\xintexpr* as they are followed by braced arguments, and thus we must intercept them here. I wanted to change this but now I don't have time to think about it. 1.1 removes : as logic operator. Replaced by ??.

```

486 \def\XINT_expr_getop_b #1%
487 {%

```

```

488 \if '#1\xint_dothis{\XINT_expr_binopwrd }\fi
489 \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
490 \xint_orthat {\XINT_expr_scanop_a #1}%
491 }%
492 \def\XINT_expr_binopwrd #1#2'\{\expandafter\XINT_expr_foundop_a
493 \csname XINT_expr_itself_\xint_zapspaces #2 \xint_gobble_i\endcsname #1}%
494 \def\XINT_expr_scanop_a #1#2#3%
495 {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral-`0#3}%
496 \def\XINT_expr_scanop_b #1#2#3%
497 {%
498 \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
499 \ifcsname XINT_expr_itself_#1#3\endcsname
500 \xint_dothis
501 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
502 \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
503 }%
504 \def\XINT_expr_scanop_c #1#2#3%
505 {%
506 \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral-`0#3%
507 }%
508 \def\XINT_expr_scanop_d #1#2#3%
509 {%
510 \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
511 \ifcsname XINT_expr_itself_#1#3\endcsname
512 \xint_dothis
513 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
514 \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
515 }%
516 \def\XINT_expr_foundop_a #1%
517 {%
518 \ifcsname XINT_expr_precedence_#1\endcsname
519 \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
520 \expandafter #1%
521 \else
522 \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
523 \fi
524 }%
525 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
526 \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

```

527 \catcode`) 11
528 \def\XINT_tmpa #1#2#3#4% (avant #4#5)
529 {%
530 \def#1##1%
531 {%
532 \xint_UDsignfork
533 ##1{\expandafter#1\romannumeral-`0#3}%
534 -{#2##1}%
535 \krof

```

```

536 }%
537 \def#2##1##2%
538 {%
539     \ifcase ##1\xint_afterfi
540     {\ifx\xINT_expr_itself^C ##2\xint_dothis
541     {\expandafter#1\romannumeral-`0\expandafter\xINT_expr_getnext\xint_gobble_i}\fi
542     \xint_orthat \XINT_expr_done }%
543     \or\xint_afterfi{\XINT_expr_extra_)
544         \expandafter #1\romannumeral-`0\xINT_expr_getop }%
545     \else
546     \xint_afterfi{\expandafter#1\romannumeral-`0\csname XINT_#4_op_##2\endcsname }%
547     \fi
548 }%
549 }%
550 \def\xINT_expr_extra_) {\xintError:removed }%
551 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
552     \expandafter\xINT_tmpa
553     \csname XINT_#1_until_end_a\expandafter\endcsname
554     \csname XINT_#1_until_end_b\expandafter\endcsname
555     \csname XINT_#1_op_-vi\endcsname
556     {#1}%
557 }%
558 \def\xINT_tmpa #1#2#3#4#5#6%
559 {%
560     \def #1##1{\expandafter #3\romannumeral-`0\xINT_expr_getnext }%
561     \def #2{\expandafter #3\romannumeral-`0\xINT_expr_getnext }%
562     \def #3##1{\xint_UDsignfork
563         ##1{\expandafter #3\romannumeral-`0#5}%
564         -{#4##1}%
565         \krof }%
566     \def #4##1##2{\ifcase ##1%
567         \xint_afterfi{\ifx\xINT_expr_itself^C ##2\xint_dothis{\xint_c_ ##2}\fi
568         \xint_orthat\xINT_expr_missing_) }%
569         \or \csname XINT_#6_op_##2\expandafter\endcsname
570         \else
571         \xint_afterfi{\expandafter #3\romannumeral-`0\csname XINT_#6_op_##2\endcsname }%
572         \fi
573     }%
574 }%
575 \def\xINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%
576 \catcode` ) 12
577 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
578     \expandafter\xINT_tmpa
579     \csname XINT_#1_op_(\expandafter\endcsname
580     \csname XINT_#1_oparen\expandafter\endcsname
581     \csname XINT_#1_until_)_a\expandafter\endcsname
582     \csname XINT_#1_until_)_b\expandafter\endcsname
583     \csname XINT_#1_op_-vi\endcsname
584     {#1}%
585 }%
586 \expandafter\let\csname XINT_expr_precedence_)\endcsname\xint_c_i
587 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i

```

```

588 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
589 \let\xint_c_xviii\xint_c_xviii
590 \expandafter\let\csname XINT_expr_precedence_^\endcsname\xint_c_
591 \expandafter\let\csname XINT_expr_precedence_++\endcsname\xint_c_i
592 \catcode`. 11 \catcode`= 11 \catcode`+ 11
593 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
594     \expandafter\let\csname XINT_#1_op_)\endcsname\xint_c_xviii
595     \expandafter\let\csname XINT_#1_op_;\endcsname\xint_c_xviii
596     \expandafter\def\csname XINT_#1_op_]\endcsname##1{\xint_c_xviii_#1a}%
597     \expandafter\let\csname XINT_#1_op_a]\endcsname\xint_c_xviii
598     \expandafter\def\csname XINT_#1_op_++\endcsname##1##2\relax
599     {\expandafter\xint_c_xviii_foundend\expandafter
600         {\expandafter\.\expandafter\xint_c_xviii_ceil{\xint_c_xviii_unlock_#1}}}%
601 }%
602 \catcode`. 12 \catcode`= 12 \catcode`+ 12

```

10.21 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, -, *, /, ^, **, //, /:, .., ..[,
], .., ||, ||:, :, ^C, and ++ operators

```

603 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{.:.}{..}{.[]}{.[]}{.}%
604      {+}{-}{*}{/}{[]}{**}{^}{a+}{a-}{a*}{a/}{a**}{a^}%
605      {}[]{}[:]{:}{}^C{++}{++}}
606 \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%

```

10.21.1 The |, &, xor, <, >, =, <=, >=, !=, //, /:, ..., ..[, and].. operators

```

607 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
608 {%
609   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iexpr
610   {% keep value, get next number and operator, then do until
611     \expandafter #2\expandafter ##1%
612     \romannumeral-`0\expandafter\XINT_expr_getnext }%
613   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iexpr
614   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
615     -{#3##1##2}%
616     \krof }%
617   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iexpr
618   {% either execute next operation now, or first do next (possibly unary)
619     \ifnum ##2>#5%
620       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
621         \csname XINT_#8_op_##3\endcsname {##4}}%
622       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
623         \csname .=#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
624       \fi }%
625   \let #7#5%
626 }%
627 \def\XINT_tmpb #1#2#3#4#5#6%
628 {%
629   \expandafter\XINT_tmpc
630   \csname XINT_#1_op_#3\expandafter\endcsname
631   \csname XINT_#1_until_#3_a\expandafter\endcsname
632   \csname XINT_#1_until_#3_b\expandafter\endcsname
633   \csname XINT_#1_op_-#5\expandafter\endcsname
634   \csname xint_c_#4\expandafter\endcsname
635   \csname #2#6\expandafter\endcsname

```

```

636 \csname XINT_expr_precedence_#3\endcsname {#1}%
637 }%
638 \xintFor #1 in {expr, flexpr} \do {%
639 \def\XINT_tmpa ##1{\XINT_tmpb {#1}{xint}##1}%
640 \xintApplyInline {\XINT_tmpa }{%
641   {|{iii}{vi}{OR}}%
642   {&{iv}{vi}{AND}}%
643   {{xor}{iii}{vi}{XOR}}%
644   {<{v}{vi}{Lt}}%
645   {>{v}{vi}{Gt}}%
646   {={v}{vi}{Eq}}%
647   {{<=}{v}{vi}{LtorEq}}%
648   {{>=}{v}{vi}{GtorEq}}%
649   {{!=}{v}{vi}{Neq}}%
650   {{.}{.}{iii}{vi}{Seq::csv}}%
651   {{/}{/}{vii}{vii}{DivTrunc}}%
652   {{/}{:}{vii}{vii}{Mod}}%
653 }%
654 }%
655 \def\XINT_tmpa #1{\XINT_tmpb {expr}{xint}#1}%
656 \xintApplyInline {\XINT_tmpa }{%
657   {+{vi}{vi}{Add}}%
658   {-{vi}{vi}{Sub}}%
659   {*{vii}{vii}{Mul}}%
660   {/{vii}{vii}{Div}}%
661   {^{viii}{viii}{Pow}}%
662   {{.}{.}{iii}{vi}{SeqA::csv}}%
663   {{.}{.}{iii}{vi}{SeqB::csv}}%
664 }%
665 \def\XINT_tmpa #1{\XINT_tmpb {flexpr}{XINTinFloat}#1}%
666 \xintApplyInline {\XINT_tmpa }{%
667   {+{vi}{vi}{Add}}%
668   {-{vi}{vi}{Sub}}%
669   {*{vii}{vii}{Mul}}%
670   {/{vii}{vii}{Div}}%
671   {^{viii}{viii}{Power}}%
672   {{.}{.}{iii}{vi}{SeqA::csv}}%
673   {{.}{.}{iii}{vi}{SeqB::csv}}%
674 }%
675 \def\XINT_tmpa #1{\XINT_tmpb {iiexpr}{xint}#1}%
676 \xintApplyInline {\XINT_tmpa }{%
677   {|{iii}{vi}{OR}}%
678   {&{iv}{vi}{AND}}%
679   {{xor}{iii}{vi}{XOR}}%
680   {<{v}{vi}{iiLt}}%
681   {>{v}{vi}{iiGt}}%
682   {={v}{vi}{iiEq}}%
683   {{<=}{v}{vi}{iiLtorEq}}%
684   {{>=}{v}{vi}{iiGtorEq}}%
685   {{!=}{v}{vi}{iiNeq}}%
686   {+{vi}{vi}{iiAdd}}%
687   {-{vi}{vi}{iiSub}}%

```

```

688  {*\{vii\}{vii}\{iiMul\}}%
689  {/\{vii\}{vii}\{iiDivRound\}}% CHANGED IN 1.1! PREVIOUSLY DID EUCLIDEAN QUOTIENT
690  {^\{viii\}{viii}\{iiPow\}}%
691  {\{..\}\{iii\}{vi}\{iiSeqA::csv\}}%
692  {\{..\}\{iii\}{vi}\{iiSeqB::csv\}}%
693  {\{..\}\{iii\}{vi}\{iiSeq::csv\}}%
694  {\{/\}\{vii\}{vii}\{iiDivTrunc\}}%
695  {\{/\}\{vii\}{vii}\{iiMod\}}%
696 }%

```

10.21.2 The $+$, $-$, $*$, $/$, $^$, $+$, $-$, $*$, $/$, and $^$ list operators

\backslash XINT_expr_binop_inline_b

```

697 \def\XINT_expr_binop_inline_a
698   {\expandafter\xint_gobble_i\romannumeral-`0\XINT_expr_binop_inline_b}%
699 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
700 \def\XINT_expr_binop_inline_c #1{%
701   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
702   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
703   \xint_orthat\XINT_expr_binop_inline_d #1}%
704 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
705 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
706 \def\XINT_expr_binop_inline_end #1,#2{%
707 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
708 {%
709   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iiexpr
710   {% keep value, get next number and operator, then do until
711     \expandafter #2\expandafter ##1%
712     \romannumeral-`0\expandafter\XINT_expr_getnext}%
713   \def #2#1#1#2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
714   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
715     -{#3##1#2}}%
716   \krof}%
717   \def #3#1#1#2#3#4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
718   {% either execute next operation now, or first do next (possibly unary)
719     \ifnum ##2>#5%
720       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
721         \csname XINT_#8_op_##3\endcsname {##4}}%
722     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
723       \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
724         {\expandafter\expandafter\expandafter#6\expandafter
725           \xint_exchangetwo_keepbraces\expandafter
726           {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
727         \romannumeral-`0\XINT_expr_unlock ##1,^,\endcsname}%
728     \fi}%
729   \let #7#5%
730}%
731 \def\XINT_tmpb #1#2#3#4%
732 {%
733   \expandafter\XINT_tmpc
734   \csname XINT_#1_op_#2\expandafter\endcsname
735   \csname XINT_#1_until_#2_a\expandafter\endcsname
736   \csname XINT_#1_until_#2_b\expandafter\endcsname
737   \csname XINT_#1_op_#3\expandafter\endcsname

```

```

738 \csname xint_c_#3\expandafter\endcsname
739 \csname #4\expandafter\endcsname
740 \csname XINT_expr_precedence_#2\endcsname {#1}%
741 }%
742 \xintApplyInline {\expandafter\XINT_tmpb \xint_firstofone}{%
743   {{expr}{+}{vi}{xintAdd}}}%
744   {{expr}{-}{vi}{xintSub}}}%
745   {{expr}{*}{vii}{xintMul}}}%
746   {{expr}{/}{vii}{xintDiv}}}%
747   {{expr}{a^}{viii}{xintPow}}}%
748   {{iiexpr}{a+}{vi}{xintiiAdd}}}%
749   {{iiexpr}{a-}{vi}{xintiiSub}}}%
750   {{iiexpr}{a*}{vii}{xintiiMul}}}%
751   {{iiexpr}{a/}{vii}{xintiiDivRound}}}%
752   {{iiexpr}{a^}{viii}{xintiiPow}}}%
753   {{flexpr}{a+}{vi}{XINTinFloatAdd}}}%
754   {{flexpr}{a-}{vi}{XINTinFloatSub}}}%
755   {{flexpr}{a*}{vii}{XINTinFloatMul}}}%
756   {{flexpr}{a/}{vii}{XINTinFloatDiv}}}%
757   {{flexpr}{a^}{viii}{XINTinFloatPower}}}%
758 }%
759 \def\XINT_tmpc #1#2#3#4#5#6#7%
760 {%
761   \def #1##1{\expandafter#2\expandafter##1\romannumeral-`0%
762     \expandafter #3\romannumeral-`0\XINT_expr_getnext }%
763   \def #2##1##2##3##4%
764   {% either execute next operation now, or first do next (possibly unary)
765     \ifnum ##2>#4%
766       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
767         \csname XINT_#7_op_##3\endcsname {##4}}}%
768       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
769         \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
770           {\expandafter#5\expandafter
771             {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
772           \romannumeral-`0\XINT_expr_unlock ##4,^,\endcsname }%
773       \fi }%
774   \let #6#4%
775 }%
776 \def\XINT_tmpb #1#2#3#4%
777 {%
778   \expandafter\XINT_tmpc
779   \csname XINT_#1_op_#2\expandafter\endcsname
780   \csname XINT_#1_until_#2\expandafter\endcsname
781   \csname XINT_#1_until_)_a\expandafter\endcsname
782   \csname xint_c_#3\expandafter\endcsname
783   \csname #4\expandafter\endcsname
784   \csname XINT_expr_precedence_#2\endcsname {#1}%
785 }%
786 \xintApplyInline {\expandafter\XINT_tmpb\xint_firstofone }{%
787   {{expr}{+}[]{}{vi}{xintAdd}}}%
788   {{expr}{-}[]{}{vi}{xintSub}}}%
789   {{expr}{*}[]{}{vii}{xintMul}}}%

```

```

790  {{expr}}{/{vii}{xintDiv}}%
791  {{expr}}{^[{viii}{xintPow}}%
792  {{iexpr}}{+[{vi}{xintiiAdd}}%
793  {{iexpr}}{-[{vi}{xintiiSub}}%
794  {{iexpr}}{*[{vii}{xintiiMul}}%
795  {{iexpr}}{/[{vii}{xintiiDivRound}}%
796  {{iexpr}}{^[{viii}{xintiiPow}}%
797  {{flexpr}}{+[{vi}{XINTinFloatAdd}}%
798  {{flexpr}}{-[{vi}{XINTinFloatSub}}%
799  {{flexpr}}{*[{vii}{XINTinFloatMul}}%
800  {{flexpr}}{/[{vii}{XINTinFloatDiv}}%
801  {{flexpr}}{^[{viii}{XINTinFloatPower}}%
802 }%

```

10.21.3 The 'and', 'or', 'xor', and 'mod' as infix operator words

```

803 \xintFor #1 in {and,or,xor,mod} \do {%
804   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
805 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
806   \csname XINT_expr_precedence_&\endcsname
807 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
808   \csname XINT_expr_precedence_|\endcsname
809 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
810   \csname XINT_expr_precedence_/\endcsname
811 \xintFor #1 in {expr, flexpr, iexpr} \do {%
812   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
813     \csname XINT_#1_op_&\endcsname
814   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
815     \csname XINT_#1_op_|\endcsname
816   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
817     \csname XINT_#1_op_/\endcsname
818 }%

```

10.21.4 The ||, &&, **, **[,]** operators as synonyms

```

819 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
820   \csname XINT_expr_precedence_=\endcsname
821 \expandafter\let\csname XINT_expr_precedence_&&\expandafter\endcsname
822   \csname XINT_expr_precedence_&\endcsname
823 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
824   \csname XINT_expr_precedence_|\endcsname
825 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
826   \csname XINT_expr_precedence_^\endcsname
827 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
828   \csname XINT_expr_precedence_a^\endcsname
829 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
830   \csname XINT_expr_precedence_^\endcsname
831 \xintFor #1 in {expr, flexpr, iexpr} \do {%
832   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
833     \csname XINT_#1_op_=\endcsname
834   \expandafter\let\csname XINT_#1_op_&&\expandafter\endcsname
835     \csname XINT_#1_op_&\endcsname
836   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
837     \csname XINT_#1_op_|\endcsname

```



```

838 \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
839 \csname XINT_#1_op_^\endcsname
840 \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
841 \csname XINT_#1_op_a^\endcsname
842 \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
843 \csname XINT_#1_op_^\endcsname
844 }%

```

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.

```

845 \def\xINT_tmpa #1#2#3#4#5#6%
846 {%
847 \def #1##1% \XINT_expr_op_[
848 {%
849 \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
850 }%
851 \def #2##1##2% \XINT_expr_until_[_a
852 {\xint_UDsignfork
853 ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
854 -{#3##1##2}%
855 \krof }%
856 \def #3##1##2##3##4% \XINT_expr_until_[_b
857 {%
858 \ifnum ##2>\xint_c_ii
859 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
860 \csname XINT_#6_op_##3\endcsname {##4}}%
861 \else
862 \xint_afterfi
863 {\expandafter ##2\expandafter ##3\csname
864 .=\expandafter\xintListSel:csv \romannumeral-`0\XINT_expr_unlock ##4;%
865 \XINT_expr_unlock ##1;\endcsname % unlock for \xintNewExpr
866 }%
867 \fi
868 }%
869 \let #5\xint_c_ii
870 }%
871 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
872 \expandafter\xINT_tmpa
873 \csname XINT_#1_op_[\expandafter\endcsname
874 \csname XINT_#1_until_[_a\expandafter\endcsname
875 \csname XINT_#1_until_[_b\expandafter\endcsname
876 \csname XINT_#1_op_-vi\expandafter\endcsname
877 \csname XINT_expr_precedence_[\endcsname {#1}%
878 }%
879 \def\xINT_tmpa #1#2#3#4#5#6%
880 {%

```

```

881 \def #1##1% \XINT_expr_op_:
882 {%
883   \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
884 }%
885 \def #2##1##2% \XINT_expr_until:_a
886 {\xint_UDsignfork
887   ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
888   -{##3##1##2}%
889   \krof }%
890 \def #3##1##2##3##4% \XINT_expr_until:_b
891 {%
892   \ifnum ##2>\xint_c_iii
893     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
894                   \csname XINT_#6_op_##3\endcsname {##4}}%
895   \else
896     \xint_afterfi
897     {\expandafter ##2\expandafter ##3\csname
898       .:=\xintiifSgn{\XINT_expr_unlock ##1}NPP.%
899       \xintiifSgn{\XINT_expr_unlock ##4}NPP.%
900       \xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}\endcsname
901     }%
902   \fi
903 }%
904 \let #5\xint_c_iii
905 }%
906 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
907 \expandafter\XINT_tmpa
908   \csname XINT_#1_op_:\expandafter\endcsname
909   \csname XINT_#1_until:_a\expandafter\endcsname
910   \csname XINT_#1_until:_b\expandafter\endcsname
911   \csname XINT_#1_op_-vi\expandafter\endcsname
912   \csname XINT_expr_precedence_:\endcsname {#1}%
913 }%
914 \catcode`[ 11 \catcode`] 11
915 \let\XINT_expr_precedence_:\xint_c_iii
916 \def\XINT_expr_op_:] #1{\expandafter\xint_c_i\expandafter }%
917   \csname .:=\xintiifSgn{\XINT_expr_unlock #1}npp\XINT_expr_unlock #1\endcsname }%
918 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
919 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
920 \let\XINT_expr_precedence_[: \xint_c_iii
921 \edef\XINT_expr_op_[: #1{\xint_c_ii \expandafter\noexpand
922   \csname XINT_expr_itself_][\endcsname #10\string :}%

: must be catcode 12, else will be mistaken for start of variable by expression parser. In
\xintListSel:csv some complications are due to \xintNewExpr matters.

923 \let\XINT_flexpr_op_[: \XINT_expr_op_[:
924 \let\XINT_iiexpr_op_[: \XINT_expr_op_[:
925 \catcode`[ 12 \catcode`] 12
926 \def\xintListSel:csv #1{%
927   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral-`0}\fi
928   \if : \noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
929   \xint_orthat {\XINT_listsel:_nth #1}%

```

```

930 }%
931 \def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
932     \expandafter\XINT_listsel:_keep\fi }%
933 \def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname }%
934 \def\XINT_listsel:_trim #1;#2;%
935     {\xintListWithSep,{\xintTrim {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
936 \def\XINT_listsel:_keep #1;#2;%
937     {\xintListWithSep,{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
938 \def\XINT_listsel:_nth#1;#2;%
939     {\xintNthElt {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}%
940 \def\XINT_listsel:_PP #1;#2;#3;%
941     {\xintListWithSep,%
942     {\xintTrim {\xintNum{#1}}%
943     {\xintKeep {\xintNum{#2}}%
944     {\xintCSVtoListNonStripped{#3}}}%
945     }%
946     }%
947 }%
948 \def\XINT_listsel:_NN #1;#2;#3;%
949     {\xintListWithSep,%
950     {\xintTrim {\xintNum{#2}}%
951     {\xintKeep {\xintNum{#1}}%
952     {\xintCSVtoListNonStripped{#3}}}%
953     }%
954     }%
955 }%
956 \def\XINT_listsel:_NP #1;#2;#3;%
957     {\expandafter\XINT_listsel:_NP_a \the\numexpr #1+%
958     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#2;#3;%
959 \def\XINT_listsel:_NP_a #1#2;{\if -#1\expandafter\XINT_listsel:_OP\fi
960     \XINT_listsel:_PP #1#2;}%
961 \def\XINT_listsel:_OP\XINT_listsel:_PP #1;{\XINT_listsel:_PP 0;}%
962 \def\XINT_listsel:_PN #1;#2;#3;%
963     {\expandafter\XINT_listsel:_PN_a \the\numexpr #2+%
964     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#1;#3;%
965 \def\XINT_listsel:_PN_a #1#2;#3;{\if -#1\expandafter\XINT_listsel:_PO\fi
966     \XINT_listsel:_PP #3;#1#2;}%
967 \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.

```

968 \def\xintSeq::csv {\romannumeral0\xintseq::csv }%
969 \def\xintseq::csv #1#2%
970 {%

```

```

971 \expandafter\XINT_seq::csv\expandafter
972 {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
973 {\the\numexpr \xintiFloor{#2}}%
974 }%
975 \def\XINT_seq::csv #1#2%
976 {%
977 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
978 \expandafter\XINT_seq::csv_z
979 \or
980 \expandafter\XINT_seq::csv_p
981 \else
982 \expandafter\XINT_seq::csv_n
983 \fi
984 {#2}{#1}%
985 }%
986 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
987 \def\XINT_seq::csv_p #1#2%
988 {%
989 \ifnum #1>#2
990 \expandafter\expandafter\expandafter\XINT_seq::csv_p
991 \else
992 \expandafter\XINT_seq::csv_e
993 \fi
994 \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
995 }%
996 \def\XINT_seq::csv_n #1#2%
997 {%
998 \ifnum #1<#2
999 \expandafter\expandafter\expandafter\XINT_seq::csv_n
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_e #1,{ }%

```

10.22.2 \xintiiSeq::csv

```

1006 \def\xintiiSeq::csv {\romannumeral0\xintiiSeq::csv}%
1007 \def\xintiiSeq::csv #1#2%
1008 {%
1009 \expandafter\XINT_iiSeq::csv\expandafter
1010 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1011 }%
1012 \def\XINT_iiSeq::csv #1#2%
1013 {%
1014 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1015 \expandafter\XINT_iiSeq::csv_z
1016 \or
1017 \expandafter\XINT_iiSeq::csv_p
1018 \else
1019 \expandafter\XINT_iiSeq::csv_n
1020 \fi

```

```

1021    {#2}{#1}%
1022 }%
1023 \def\XINT_iiseq::csv_z #1#2{ #1}%
1024 \def\XINT_iiseq::csv_p #1#2%
1025 {%
1026     \ifnum #1>#2
1027         \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1028     \else
1029         \expandafter\XINT_seq::csv_e
1030     \fi
1031     \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1032 }%
1033 \def\XINT_iiseq::csv_n #1#2%
1034 {%
1035     \ifnum #1<#2
1036         \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1037     \else
1038         \expandafter\XINT_seq::csv_e
1039     \fi
1040     \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1041 }%
1042 \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

```

1043 \def\xintSeqA::csv #1%
1044     {\expandafter\XINT_seq::csv\expandafter{\romannumeral0\xintra {#1}}}%
1045 \def\XINT_seq::csv #1#2{\expandafter\XINT_seq::csv_a \romannumeral0\xintra {#2};#1;}%
1046 \def\xintiiSeqA::csv #1#2{\XINT_iiseq::csv #1#2}%
1047 \def\XINT_iiseq::csv #1#2#3#4{\expandafter\XINT_seq::csv_a
1048     \romannumeral-`0\expandafter \XINT_expr_unlock\expandafter#4%
1049     \expandafter;\romannumeral-`0\XINT_expr_unlock #2;}%
1050 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseq::csv\expandafter
1051     {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1052 \def\XINT_flseq::csv #1#2%
1053     {\expandafter\XINT_seq::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1054 \def\XINT_seq::csv_a #1{\xint_UDzerominusfork
1055     #1-{z}%
1056     0#1{n}%
1057     0-{p}%
1058     \krof #1}%

```

10.23.2 \xintSeqB::csv

```

1059 \def\xintSeqB::csv #1#2%
1060     {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1061 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral-`0#2#1!}%
1062 \def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1063     \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%

```

```

1064 \def\XINT_seqb::csv_p #1#2#3%
1065 {%
1066   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1067   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

\romannumeral0 stopped by *\endcsname*, *XINT_expr_seq_empty?* constructs "nil".

```

1068   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1069 }%
1070 \def\XINT_seqb::csv_n #1#2#3%
1071 {%
1072   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1073   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1074   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1075 }%
1076 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

10.23.3 *\xintiiSeqB::csv*

```

1077 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1078 \def\XINT_iiseqb::csv #1#2#3#4%
1079   {\expandafter\XINT_iiseqb::csv_a
1080   \romannumeral-`0\expandafter \XINT_expr_unlock\expandafter#2%
1081   \romannumeral-`0\XINT_expr_unlock #4!}%
1082 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1083   \romannumeral-`0\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1084 \def\XINT_iiseqb::csv_p #1#2#3%
1085 {%
1086   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1087   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1088   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1089 }%
1090 \def\XINT_iiseqb::csv_n #1#2#3%
1091 {%
1092   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1093   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1094   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1095 }%
1096 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

10.23.4 *\XINTinFloatSeqB::csv*

```

1097 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1098   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1099 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral-`0#2#1!}%
1100 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1101   \romannumeral-`0\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1102 \def\XINT_flseqb::csv_p #1#2#3%
1103 {%
1104   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1105   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1106   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1107 }%
1108 \def\XINT_flseqb::csv_n #1#2#3%
1109 {%
1110   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%

```

```

1111 {,#1\expandafter\XINT_flseqb::csv_n\expandafter}%
1112 {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1113 }%
1114 \def\XINT_flseqb::csv_z #1#2#3{,#1}%

```

10.24 The comma as binary operator

New with 1.09a.

```

1115 \def\XINT_tmpa #1#2#3#4#5#6%
1116 {%
1117   \def #1##1% \XINT_expr_op_,
1118   {%
1119     \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
1120   }%
1121   \def #2##1##2% \XINT_expr_until_,_a
1122   {\xint_UDsignfork
1123     ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
1124     -{#3##1##2}%
1125   \krof }%
1126   \def #3##1##2##3##4% \XINT_expr_until_,_b
1127   {%
1128     \ifnum ##2>\xint_c_ii
1129     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
1130                   \csname XINT_#6_op_##3\endcsname {##4}}%
1131     \else
1132     \xint_afterfi
1133     {\expandafter ##2\expandafter ##3%
1134     \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1135     \fi
1136   }%
1137   \let #5\xint_c_ii
1138 }%
1139 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1140 \expandafter\XINT_tmpa
1141   \csname XINT_#1_op_,\expandafter\endcsname
1142   \csname XINT_#1_until_,_a\expandafter\endcsname
1143   \csname XINT_#1_until_,_b\expandafter\endcsname
1144   \csname XINT_#1_op_-vi\expandafter\endcsname
1145   \csname XINT_expr_precedence_,\endcsname {#1}%
1146 }%

```

10.25 The minus as prefix operator of variable precedence level

```

1147 \def\XINT_tmpa #1#2#3%
1148 {%
1149   \expandafter\XINT_tmppb
1150   \csname XINT_#1_op_-#3\expandafter\endcsname
1151   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1152   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1153   \csname xint_c_#3\endcsname {#1}#2%
1154 }%
1155 \def\XINT_tmppb #1#2#3#4#5#6%
1156 {%

```

```

1157 \def #1% \XINT_expr_op_-<level>
1158 {% get next number+operator then switch to _until macro
1159 \expandafter #2\romannumeral-`0\XINT_expr_getnext
1160 }%
1161 \def #2##1% \XINT_expr_until_-<l>_a
1162 {\xint_UDsignfork
1163 ##1{\expandafter #2\romannumeral-`0#1}%
1164 -{#3##1}%
1165 \krof }%
1166 \def #3##1##2##3% \XINT_expr_until_-<l>_b
1167 {% _until tests precedence level with next op, executes now or postpones
1168 \ifnum ##1>#4%
1169 \xint_afterfi {\expandafter #2\romannumeral-`0%
1170 \csname XINT_#5_op_##2\endcsname {##3}}%
1171 \else
1172 \xint_afterfi {\expandafter ##1\expandafter ##2%
1173 \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1174 \fi
1175 }%
1176 }%
1177 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1178 \xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1179 \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}

```

1180 \let\XINT_expr_precedence_? \xint_c_x
1181 \def\XINT_expr_op_? #1#2{\if ?#2\expandafter \XINT_expr_op_??\fi
1182 \XINT_expr_op_?a #1{#2}}%
1183 \def\XINT_expr_op_?a #1#2#3%
1184 {%
1185 \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1186 }%
1187 \let\XINT_flexpr_op_?\XINT_expr_op_?
1188 \let\XINT_iexpr_op_?\XINT_expr_op_?
1189 \def\XINT_expr_op_?? #1#2#3#4#5#6%
1190 {%
1191 \xintiiifSgn {\XINT_expr_unlock #2}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1192 {\XINT_expr_getnext #6}%
1193 }%

```

10.27 ! as postfix factorial operator

As of 2014/11/07, not yet a float version of factorial. I must do it!

```

1194 \let\XINT_expr_precedence_! \xint_c_x
1195 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1196 \csname .=\xintFac{\XINT_expr_unlock #1}\endcsname }%
1197 \let\XINT_flexpr_op_!\XINT_expr_op_!

```



```
1198 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1199 \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 CRAZYNESS OF NUMEXPR: \the\numexpr 3 + 7 9 \relax !! Hence we have to do the job ourselves.

```
1200 \catcode`[ 11
1201 \catcode`* 11
1202 \let\XINT_expr_precedence_ \xint_c_vii
1203 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1204 \csname .=\xintE{\XINT_expr_unlock #1}%
1205 {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1206 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1207 \csname .=\xintiiE{\XINT_expr_unlock #1}%
1208 {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1209 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1210 \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1211 {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1212 \catcode`[ 12
1213 \catcode`* 12
```

10.29 For variables

```
1214 \def\XINT_expr_op__ #1% op__ with two _'s
1215 {%
1216 \ifcsname XINT_expr_var_#1\endcsname
1217 \expandafter\xint_firstoftwo
1218 \else
1219 \expandafter\xint_secondoftwo
1220 \fi
1221 {\expandafter\expandafter\expandafter\expandafter
1222 \expandafter\expandafter\expandafter
1223 \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1224 {\XINT_expr_unknown_variable {#1}%
1225 \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1226 }%
1227 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1228 \let\XINT_flexpr_op__ \XINT_expr_op__
1229 \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.

```
1230 \catcode`: 12
```

```

1231 \def\xintdefvar #1:=#2;\expandafter\odef
1232   \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
1233   {\expandafter\empty\romannumeral0\xintbareeval #2\relax }}%
1234 \def\xintdefiivar #1:=#2;\expandafter\odef
1235   \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
1236   {\expandafter\empty\romannumeral0\xintbareiieval #2\relax }}%
1237 }%
1238 \def\xintdeffloatvar #1:=#2;\expandafter\odef
1239   \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
1240   {\expandafter\empty\romannumeral0\xintbarefloateval #2\relax }}%
1241 }%
1242 \catcode`.: 11

```

10.29.2 Letters as dummy variables; the nil list

```

1243 \def\XINT_tmpa #1%
1244 {%
1245   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1246   {\romannumeral0\XINT_expr_lockscan ##2!##1\relax !#1{##2}}%
1247 }%
1248 \xintApplyUnbraced \XINT_tmpa {abcdefghijklmnopqrstuvwxyz}%
1249 \xintApplyUnbraced \XINT_tmpa {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1250 \expandafter\def\expandafter\XINT_expr_var_nil\expandafter
1251   {\expandafter\empty\csname . = \endcsname}%

```

10.29.3 The omit and abort constructs

```

1252 \catcode`. 11 \catcode`= 11
1253 \def\XINT_expr_var_omit #1\relax !{1^C!{}{}{}.\!=!\relax !}% 24 juin
1254 \def\XINT_expr_var_abort #1\relax !{1^C!{}{}{}.\!=^\relax !}% 25 juin
1255 \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!

```

1256 \catcode`? 3
1257 \def\XINT_expr_var_@ #1~#2{ #2#1~#2}%
1258 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1259 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{ #3#1~#2#3}%
1260 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{ #4#1~#2#3#4}%
1261 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{ #5#1~#2#3#4#5}%
1262 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1263 {%
1264   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1265   {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1266 }%
1267 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1268 {%
1269   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1270   {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1271 }%
1272 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%

```

```

1273 {%
1274   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1275   {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1276 }%
1277 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1278 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1279 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
1280 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1281 {%
1282   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1283   {\XINT_expr_unlock#3}{#5}#4~#5?%
1284 }%
1285 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1286 {%
1287   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1288   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1289 }%
1290 \def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1291 {%
1292   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1293   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1294 }%
1295 \catcode`? 11

```

10.30 For functions

```

1296 \def\XINT_tmpa #1#2#3{%
1297   \def #1##1% \XINT_expr_op_`, #2=\XINT_expr_oparen
1298   {%
1299     \ifcsname XINT_expr_onlitteral_##1\endcsname
1300     \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi
1301     \ifcsname XINT_#3_func_##1\endcsname
1302     \xint_dothis{\expandafter\expandafter
1303       \csname XINT_#3_func_##1\endcsname\romannumeral-`0#2}\fi
1304     \xint_orthat{\XINT_expr_unknown_function {##1}%
1305       \expandafter\XINT_expr_func_unknown\romannumeral-`0#2}%
1306   }%
1307 }%
1308 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1309 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1310   \expandafter\XINT_tmpa
1311   \csname XINT_#1_op_`\expandafter\endcsname
1312   \csname XINT_#1_oparen\endcsname
1313   {#1}%
1314 }%
1315 \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.

```

1316 \def\XINT_expr_onlitteral_bool #1)%

```

```

1317      {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1318 \def\XINT_expr_onlitteral_togl #1)%
1319      {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1320 \def\XINT_expr_onlitteral_protect #1)%
1321      {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%
1322 \def\XINT_expr_func_unknown #1#2#3{\expandafter #1\expandafter #2\csname .=#0\endcsname }%
1323 \def\XINT_expr_func_break #1#2#3%
1324 {\expandafter #1\expandafter #2\csname .=?\romannumeral-`0\XINT_expr_unlock #3\endcsname }%
1325 \let\XINT_flexpr_func_break \XINT_expr_func_break
1326 \let\XINT_iexpr_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 \xintNew-Expr !)

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 iexpr.

10.32.1 \XINT_expr_onlitteral_seq

```

1327 \def\XINT_expr_onlitteral_seq
1328 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
1329 \def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

10.32.2 \XINT_expr_onlitteral_seq_a

```

1330 \def\XINT_expr_onlitteral_seq_a #1#2,%
1331 {% checks balancing of parentheses
1332   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1333   \expandafter\XINT_expr_onlitteral_seq_c
1334   \or\expandafter\XINT_expr_onlitteral_seq_b
1335   \else\expandafter\xintError:we_are_doomed
1336   \fi {#1#2},%
1337 }%
1338 \def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1},}%
1339 \def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
1340 {%
1341   \XINT_expr_onlitteral_seq_d {#2{#1}}}%
1342 }%
1343 \def\XINT_expr_onlitteral_seq_d #1#2#3)%
1344 {%

```

```

1345 \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1346 \or\expandafter\XINT_expr_onlitteral_seq_e
1347 \else\expandafter\xintError:we_are_doomed
1348 \fi
1349 {#1}{#2#3}%
1350 }%
1351 \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.

```

1352 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1353 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1354 \def\XINT_isbalanced_b #1)#2%
1355 {\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

1356 \def\XINT_isbalanced_error #1)\xint_bye {\m@ne}%

  #2 was \xint_bye, was there a ) in original #1?

1357 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1358 {\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.

1359 \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.

1360 \def\XINT_isbalanced_d #1)#2%
1361 {\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.

1362 \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

```

1363 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareeval }%
1364 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbarefloateval}%
1365 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareiieval}%
1366 \def\XINT_allexpr_seqx #1#2#3#4% #2 is the index list, fully evaluated and encapsulated
1367 {% #3 is the Latin letter serving as dummy variable, #4 is the expression to evaluate
1368 \expandafter \XINT_expr_getop
1369 \csname .=\expandafter\XINT_expr_seq:_aa
1370 \romannumeral-`0\XINT_expr_unlock #2!{#1#4\relax !#3},^,\endcsname
1371 }%
1372 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1373 \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.

```

1374 \def\xint_expr_seq:_a #1!#2{\expandafter\xint_expr_seq_empty?
1375         \romannumeral0\xint_expr_seq:_b {#2}#1}%
1376 \def\xint_expr_seq:_b #1#2,{\xint_expr_seq:_c #2,{#1}}%
1377 \def\xint_expr_seq:_c #1{\if ,#1\xint_dothis\xint_expr_seq:_noop\fi
1378         \if ^#1\xint_dothis\xint_expr_seq:_end\fi
1379         \xint_orthat\xint_expr_seq:_d #1}%
1380 \def\xint_expr_seq:_d #1,#2{\expandafter\xint_expr_seq:_e
1381     \romannumeral-`0\expandafter\xint_expr_unlock\romannumeral0#2{#1}{#2}}%
1382 \def\xint_expr_seq:_e #1{\if #1^\xint_dothis\xint_expr_seq:_abort\fi
1383     \if #1?\xint_dothis\xint_expr_seq:_break\fi
1384     \if #1!\xint_dothis\xint_expr_seq:_omit\fi
1385     \xint_orthat{\xint_expr_seq:_goon #1}}%
1386 \def\xint_expr_seq:_goon #1!#2#3#4{,#1\xint_expr_seq:_b {#4}}%
1387 \def\xint_expr_seq:_omit #1!#2#3#4{\xint_expr_seq:_b {#4}}%
1388 \def\xint_expr_seq:_abort #1!#2#3#4#5^,{}%
1389 \def\xint_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1390 \def\xint_expr_seq:_noop ,#1{\xint_expr_seq:_b {#1}}%
1391 \def\xint_expr_seq:_end ^,#1}% if all is omit, _empty? constructs "nil"
1392 \def\xint_expr_seq_empty? #1{%
1393 \def\xint_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1394 \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.

```

1395 \def\xint_expr_seq:_A +#1!#2,^,%
1396     {\expandafter\xint_expr_seq_empty?\romannumeral0\xint_expr_seq:_D {#1}{#2}}%
1397 \def\xint_expr_seq:_D #1#2{\expandafter\xint_expr_seq:_E
1398     \romannumeral-`0\expandafter\xint_expr_unlock\romannumeral0#2{#1}{#2}}%
1399 \def\xint_expr_seq:_E #1{\if #1^\xint_dothis\xint_expr_seq:_Abort\fi
1400     \if #1?\xint_dothis\xint_expr_seq:_Break\fi
1401     \if #1!\xint_dothis\xint_expr_seq:_Omit\fi
1402     \xint_orthat{\xint_expr_seq:_Goon #1}}%
1403 \def\xint_expr_seq:_Goon #1!#2#3#4%
1404     {,#1\expandafter\xint_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
1405 \def\xint_expr_seq:_Omit #1!#2#3#4%
1406     {\expandafter\xint_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
1407 \def\xint_expr_seq:_Abort #1!#2#3#4{%
1408 \def\xint_expr_seq:_Break #1!#2#3#4{,#1}%

```

10.32.7 add and mul, \XINT_expr_onlitteral_add, \XINT_expr_onlitteral_mul

```

1409 \def\xint_expr_onlitteral_add
1410 {\expandafter\xint_expr_onlitteral_add_f\romannumeral-`0\xint_expr_onlitteral_seq_a {}}%

```

```

1411 \def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opx}#2)\relax #1+}%
1412 \def\XINT_expr_onlitteral_mul
1413 {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
1414 \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

```

1415 \expandafter\edef\csname XINT_expr_op:_{\endcsname
1416 {\noexpand\xint_gobble_v {}}{\expandafter\noexpand\csname .={0\endcsname}%
1417 \expandafter\edef\csname XINT_expr_op:_{*\endcsname
1418 {\noexpand\xint_gobble_v {}}{\expandafter\noexpand\csname .={1\endcsname}%
1419 \def\XINT_expr_func_opx #1#2{\XINT_allexpr_opx \xintexpr }%
1420 \def\XINT_flexpr_func_opx #1#2{\XINT_allexpr_opx \xintfloatexpr }%
1421 \def\XINT_iiexpr_func_opx #1#2{\XINT_allexpr_opx \xintiiepr }%
1422 \def\XINT_allexpr_opx #1#2#3#4#5%

```

au d' 'e part on avait op(#4,#3=#2 ('evalu' 'e ici)) #3=la variable, #4=expression, #5=+ ou*.

```

1423 {%
1424 \expandafter\XINT_expr_getop\romannumeral0\expandafter\XINT_expr_op:_a
1425 \csname XINT_expr_op:_{#5\expandafter\endcsname
1426 \romannumeral-`0\XINT_expr_unlock #2!#5#1#3{#4}%
1427 }%

```

10.32.9 \XINT_expr_op:_a, ...

```

1428 \def\XINT_expr_op:_a #1#2!#3#4#5#6{\XINT_expr_op:_b {#1#4#3{#6\relax\relax !#5}}#2,^,%

```

#1=op_+ ou op_*, #2=liste, #3=+ou*, #4=\xintexpr, etc, #5=la var,#6=expression

```

1429 \def\XINT_expr_op:_b #1#2,{\XINT_expr_op:_c #2,#1}%
1430 \def\XINT_expr_op:_c #1{\if ,#1\xint_dothis\XINT_expr_op:_noop\fi
1431 \if ^#1\xint_dothis\XINT_expr_op:_end\fi
1432 \xint_orthat\XINT_expr_op:_d #1}%
1433 \def\XINT_expr_op:_noop #1,#2#3#4#5{\XINT_expr_op:_b {#2}#3#4{#5}}%
1434 \def\XINT_expr_op:_d #1,#2#3#4#5%

```

#1=valeur, #2=partiel, #3=\xintexpr #4=+ ou *, #5 = expression

```

1435 {\expandafter\expandafter\expandafter\XINT_expr_op:_e #3#2#4#3#5{#1}{#3#4{#5}}}%

```

#2=nom de la variable, #3=ancienne valeur variable

```

1436 \def\XINT_expr_op:_e !#1!#2#3#4{\XINT_expr_op:_b {#!#1}#4}%
1437 \def\XINT_expr_op:_end ^,#1#2#3#4{\expandafter\expandafter\expandafter\space
1438 \expandafter\xint_gobble_iv #1}%

```

10.32.10 subs, \XINT_expr_onlitteral_subs

```

1439 \def\XINT_expr_onlitteral_subs
1440 {\expandafter\XINT_expr_onlitteral_subs_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
1441 \def\XINT_expr_onlitteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1442 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1443 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1444 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1445 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1446 {% #3 is the dummy variable, #4 is the expression to evaluate
1447 \expandafter \XINT_expr_getop
1448 \csname .={\expandafter\XINT_expr_subx:_a

```



```

1449      \romannumeral-`0\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1450 }%
1451 \def\XINT_expr_subx:_a #1!#2%
1452   {\expandafter\XINT_expr_subx:_end \romannumeral0#2{#1}}%

```

attention, if one day I add a space in unlock, will need \romannumeral-`0

```

1453 \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.

```

1454 \def\XINT_expr_func_rseq   {\XINT_allexpr_rseq \xintbareeval      }%
1455 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
1456 \def\XINT_iexpr_func_rseq  {\XINT_allexpr_rseq \xintbareiieval   }%
1457 \def\XINT_allexpr_rseq #1#2%
1458 {%
1459   \expandafter\XINT_expr_rseqx\expandafter #1\expandafter
1460   #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
1461 }%

```

10.33.1 \XINT_expr_rseqx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```

1462 \def\XINT_expr_rseqx #1#2#3#4%
1463 {%
1464   \expandafter\XINT_expr_rseqy\romannumeral0#1(#4)\relax
1465   #2#3#1%
1466 }%

```

10.33.2 \XINT_expr_rseqy

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```

1467 \def\XINT_expr_rseqy #1#2#3#4#5%
1468 {%
1469   \expandafter \XINT_expr_getop
1470   \csname .=\XINT_expr_unlock #2%
1471   \expandafter\XINT_expr_rseq:_aa
1472   \romannumeral-`0\XINT_expr_unlock #1!{#5#4\relax !#3}#2,^,\endcsname
1473 }%
1474 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1475   \expandafter\XINT_expr_rseq:_a\fi #1}%

```

10.33.3 \XINT_expr_rseq:_a etc...

```

1476 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b #3{#2}#1}%
1477 \def\XINT_expr_rseq:_b #1#2#3,{\XINT_expr_rseq:_c #3,~#1{#2}}%
1478 \def\XINT_expr_rseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rseq:_noop\fi
1479   \if ^#1\xint_dothis\XINT_expr_rseq:_end\fi
1480   \xint_orthat\XINT_expr_rseq:_d #1}%

```



```

1481 \def\XINT_expr_rseq:_d #1,~#2#3{\expandafter\XINT_expr_rseq:_e
1482   \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2{#3}}%
1483 \def\XINT_expr_rseq:_e #1{%
1484   \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1485   \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1486   \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1487   \xint_orthat{\XINT_expr_rseq:_goon #1}}%
1488 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1489   \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1490 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1491 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{}%
1492 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%
1493 \def\XINT_expr_rseq:_noop ,~#1#2{\XINT_expr_rseq:_b #1{#2}}%
1494 \def\XINT_expr_rseq:_end ^,~#1#2{}% no nil for rseq

```

10.33.4 \XINT_expr_rseq:_A etc...

n++ for rseq

```

1495 \def\XINT_expr_rseq:_A +#1!#2#3,^,{\XINT_expr_rseq:_D {#1}#3{#2}}%
1496 \def\XINT_expr_rseq:_D #1#2#3{\expandafter\XINT_expr_rseq:_E
1497   \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2{#3}}%
1498 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1499   \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1500   \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1501   \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1502 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1503   {,#1\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1504   \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1505 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1506   {\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1507 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{}%
1508 \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.

```

1509 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval }%
1510 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
1511 \def\XINT_iexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval }%
1512 \def\XINT_allexpr_rrseq #1#2%
1513 {%
1514   \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter
1515   #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
1516 }%

```

10.34.1 \XINT_expr_rrseqx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```

1517 \def\XINT_expr_rrseqx #1#2#3#4%
1518 {%

```

```

1519 \expandafter\XINT_expr_rrseq\romannumeral0#1(#4)\expandafter\relax
1520 \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1521 {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
1522 #2#3#1%
1523 }%

```

10.34.2 \XINT_expr_rrseq

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```

1524 \def\XINT_expr_rrseq #1#2#3#4#5#6%
1525 {%
1526 \expandafter \XINT_expr_getop
1527 \csname .=\XINT_expr_unlock #3%
1528 \expandafter\XINT_expr_rrseq:_aa
1529 \romannumeral-`0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
1530 }%
1531 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1532 \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

10.34.3 \XINT_expr_rrseq:_a etc...

```

1533 \catcode`? 3
1534 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1}%
1535 \def\XINT_expr_rrseq:_b #1#2#3,{\XINT_expr_rrseq:_c #3,~#1?{#2}}%
1536 \def\XINT_expr_rrseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rrseq:_noop\fi
1537 \if ^#1\xint_dothis\XINT_expr_rrseq:_end\fi
1538 \xint_orthat\XINT_expr_rrseq:_d #1}%
1539 \def\XINT_expr_rrseq:_d #1,~#2?#3{\expandafter\XINT_expr_rrseq:_e
1540 \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1541 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,~#1\expandafter\XINT_expr_rrseq:_b\expandafter
1542 {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1543 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1544 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{,}%
1545 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,~#1}%
1546 \def\XINT_expr_rrseq:_noop ,~#1?#2{\XINT_expr_rrseq:_b {#1}{#2}}%
1547 \def\XINT_expr_rrseq:_end ^,~#1?#2{}% No nil for rrseq.
1548 \catcode`? 11
1549 \def\XINT_expr_rrseq:_e #1{%
1550 \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1551 \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1552 \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1553 \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1554 }%

```

10.34.4 \XINT_expr_rrseq:_A etc...

n++ for rrseq

```

1555 \catcode`? 3
1556 \def\XINT_expr_rrseq:_A +#1!#2#3,^,{\XINT_expr_rrseq:_D {#1}{#3}{#2}}%
1557 \def\XINT_expr_rrseq:_D #1#2#3{\expandafter\XINT_expr_rrseq:_E

```

```

1558 \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1559 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1560 {,#1\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1561 \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1562 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1563 {\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1564 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5}%
1565 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1566 \catcode`? 11
1567 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1568 \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1569 \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1570 \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

10.35 iter

```

1571 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval }%
1572 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
1573 \def\XINT_iexpr_func_iter {\XINT_allexpr_iter \xintbareiieval }%
1574 \def\XINT_allexpr_iter #1#2%
1575 {%
1576 \expandafter\XINT_expr_iterx\expandafter #1\expandafter
1577 #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
1578 }%

```

10.35.1 \XINT_expr_iterx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```

1579 \def\XINT_expr_iterx #1#2#3#4%
1580 {%
1581 \expandafter\XINT_expr_itery\romannumeral0#1(#4)\expandafter\relax
1582 \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1583 {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
1584 #2#3#1%
1585 }%

```

10.35.2 \XINT_expr_itery

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```

1586 \def\XINT_expr_itery #1#2#3#4#5#6%
1587 {%
1588 \expandafter \XINT_expr_getop
1589 \csname .=%
1590 \expandafter\XINT_expr_iter:_aa
1591 \romannumeral-`0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
1592 }%
1593 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1594 \expandafter\XINT_expr_iter:_a\fi #1}%

```

10.35.3 \XINT_expr_iter:_a etc...

```

1595 \catcode`? 3
1596 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1}%
1597 \def\XINT_expr_iter:_b #1!#2#3,{\XINT_expr_iter:_c #3,~#1?{#2}}%
1598 \def\XINT_expr_iter:_c #1{\if ,#1\xint_dothis\XINT_expr_iter:_noop\fi
1599         \if ^#1\xint_dothis\XINT_expr_iter:_end\fi
1600         \xint_orthat\XINT_expr_iter:_d #1}%
1601 \def\XINT_expr_iter:_d #1,~#2?#3{\expandafter\XINT_expr_iter:_e
1602         \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1603 \def\XINT_expr_iter:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iter:_b\expandafter
1604         {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}}%
1605 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1606 \def\XINT_expr_iter:_abort #1!#2#3~#4?#5#6^,%
1607     {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1608         {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1609 \def\XINT_expr_iter:_break #1!#2#3~#4?#5#6^,%
1610     {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1611         {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1612 \def\XINT_expr_iter:_noop ,~#1?#2{\XINT_expr_iter:_b {#1}{#2}}%
1613 \def\XINT_expr_iter:_end ^,~#1?#2%
1614     {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1615         {,\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1616 \catcode`? 11
1617 \def\XINT_expr_iter:_e #1{%
1618     \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1619     \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1620     \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1621     \xint_orthat{\XINT_expr_iter:_goon #1}%
1622 }%
1623 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

10.35.4 \XINT_expr_iter:_A etc...*n++ for iter*

```

1624 \catcode`? 3
1625 \def\XINT_expr_iter:_A +#1!#2#3,^,{\XINT_expr_iter:_D {#1}{#3}{#2}}%
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%
1629     {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1630     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}}%
1631 \def\XINT_expr_iter:_Omit #1!#2#3~%#4?#5%
1632     {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1633 \def\XINT_expr_iter:_Abort #1!#2#3~#4?#5%
1634     {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1635         {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1636 \def\XINT_expr_iter:_Break #1!#2#3~#4?#5%
1637     {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1638         {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1639 \catcode`? 11
1640 \def\XINT_expr_iter:_E #1{\if #1^ \xint_dothis\XINT_expr_iter:_Abort\fi
1641     \if #1? \xint_dothis\XINT_expr_iter:_Break\fi

```

```
1642 \if #1!\xint_dothis\xINT_expr_iter:_Omit\fi
1643 \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.

```
1644 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral-`0#1,,^}%
1645 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1646 \else\expandafter\XINT_andof:_c\fi #1}%
1647 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1648 \def\XINT_andof:_no #1^{0}%
1649 \def\XINT_andof:_e #1^{1}% works with empty list
```

10.36.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```
1650 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral-`0#1,,^}%
1651 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1652 \else\expandafter\XINT_orof:_c\fi #1}%
1653 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1654 \def\XINT_orof:_yes #1^{1}%
1655 \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`).

```
1656 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral-`0#1,,^}%
1657 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1658 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
1659 \else\expandafter\XINT_xorof:_c\fi #1}%
1660 \def\XINT_xorof:_c #1,#2%
1661 {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1662 \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1663 {\XINT_xorof:_a #2}%
1664 }%
1665 \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.

```

1666 \def\XINT_oncsv:_empty #1,^,#2{#2}%
1667 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
1668 \def\XINT_oncsv:_a #1#2#3%
1669 {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
1670 \def\XINT_oncsv:_b #1#2#3,%
1671 {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral-`0#2{#3}}#1#2}%
1672 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral-`0#4,{#1}#2#3}%
1673 \def\XINT_oncsv:_d #1%
1674 {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
1675 \def\XINT_oncsv:_e #1,#2#3#4%
1676 {\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.

```

1677 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1678 \expandafter\xint_firstofone\romannumeral-`0#1,^{0/1[0]}}%
1679 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
1680 \expandafter\xint_firstofone\romannumeral-`0#1,^{0}%

```

10.36.6 \xintMinof:csv, \xintiiMinof:csv

1.09i. Rewritten for 1.1. For use by \xintiiexpr.

```

1681 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1682 \expandafter\xint_firstofone\romannumeral-`0#1,^{0/1[0]}}%
1683 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimin
1684 \expandafter\xint_firstofone\romannumeral-`0#1,^{0}%

```

10.36.7 \xintSum:csv, \xintiiSum:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```

1685 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
1686 \expandafter\xint_firstofone\romannumeral-`0#1,^{0/1[0]}}%
1687 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
1688 \expandafter\xint_firstofone\romannumeral-`0#1,^{0}%

```

10.36.8 \xintPrd:csv, \xintiiPrd:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```

1689 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
1690 \expandafter\xint_firstofone\romannumeral-`0#1,^{1/1[0]}}%
1691 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimul
1692 \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

```
1693 \def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
1694 \expandafter\xint_firstofone\romannumeral-`0#1,^,1}%
1695 \def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
1696 \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.10 `\xintiiGCDof:csv`, `\xintiiLCMof:csv`

1.1a pour `\xintiiexpr`. Ces histoires de `ii` sont pénibles à la fin.

```
1697 \def\xintiiGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiigcd
1698 \expandafter\xint_firstofone\romannumeral-`0#1,^,1}%
1699 \def\xintiiLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiilcm
1700 \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`

for `\xintNewExpr` matters, mainly.

```
1701 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
1702 \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

```
1703 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1704 \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{0[0]}}}%
1705 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1706 \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{0[0]}}}%
```

10.36.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```
1707 \def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatadd
1708 \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{0[0]}}}%
1709 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatmul
1710 \expandafter\XINTinFloatdigits\romannumeral-`0#1,^{1[0]}}}%
```

10.37 The `num`, `reduce`, `abs`, `sgn`, `frac`, `floor`, `ceil`, `sqr`, `sqrt`, `sqrtr`, `float`, `round`, `trunc`, `mod`, `quo`, `rem`, `gcd`, `lcm`, `max`, `min`, ``+``, ``*``, `?`, `!`, `not`, `all`, `any`, `xor`, `if`, `ifsgn`, `first`, `last`, `even`, `odd`, and `reversed` functions

```
1711 \def\XINT_expr_twoargs #1,#2,{#{#1}{#2}}%
1712 \def\XINT_expr_argadopt #1,#2,#3.#4#5%
1713 {%
1714 \if\relax#3\relax\expandafter\xint_firstoftwo\else
1715 \expandafter\xint_secondoftwo\fi
```



```

1716     {#4}{#5[\xintNum {#2}]}{#1}%
1717 }%
1718 \def\xINT_expr_oneortwo #1#2#3,#4,#5.%
1719 {%
1720     \if\relax#5\relax\expandafter\xint_firstoftwo\else
1721         \expandafter\xint_secondoftwo\fi
1722     {#1{0}}{#2{\xintNum {#4}}}{#3}%
1723 }%
1724 \def\xINT_iiexpr_oneortwo #1#2,#3,#4.%
1725 {%
1726     \if\relax#4\relax\expandafter\xint_firstoftwo\else
1727         \expandafter\xint_secondoftwo\fi
1728     {#1{0}}{#1{#3}}{#2}%
1729 }%
1730 \def\xINT_expr_func_num #1#2#3%
1731     {\expandafter #1\expandafter #2\csname.=\xintNum {\xINT_expr_unlock #3}\endcsname }%
1732 \let\xINT_flexpr_func_num\xINT_expr_func_num
1733 \let\xINT_iiexpr_func_num\xINT_expr_func_num
1734 % [0] added Oct 25. For interaction with SPRaw::csv
1735 \def\xINT_expr_func_reduce #1#2#3%
1736     {\expandafter #1\expandafter #2\csname.=\xintIrr {\xINT_expr_unlock #3}[0]\endcsname }%
1737 \let\xINT_flexpr_func_reduce\xINT_expr_func_reduce
1738 % no \XINT_iiexpr_func_reduce
1739 \def\xINT_expr_func_abs #1#2#3%
1740     {\expandafter #1\expandafter #2\csname.=\xintAbs {\xINT_expr_unlock #3}\endcsname }%
1741 \let\xINT_flexpr_func_abs\xINT_expr_func_abs
1742 \def\xINT_iiexpr_func_abs #1#2#3%
1743     {\expandafter #1\expandafter #2\csname.=\xintiAbs {\xINT_expr_unlock #3}\endcsname }%
1744 \def\xINT_expr_func_sgn #1#2#3%
1745     {\expandafter #1\expandafter #2\csname.=\xintSgn {\xINT_expr_unlock #3}\endcsname }%
1746 \let\xINT_flexpr_func_sgn\xINT_expr_func_sgn
1747 \def\xINT_iiexpr_func_sgn #1#2#3%
1748     {\expandafter #1\expandafter #2\csname.=\xintiiSgn {\xINT_expr_unlock #3}\endcsname }%
1749 \def\xINT_expr_func_frac #1#2#3%
1750     {\expandafter #1\expandafter #2\csname.=\xintTFrac {\xINT_expr_unlock #3}\endcsname }%
1751 \def\xINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
1752     .=\XINTinFloatFracdigits {\xINT_expr_unlock #3}\endcsname }%
1753 % no \XINT_iiexpr_func_frac
1754 \def\xINT_expr_func_floor #1#2#3%
1755     {\expandafter #1\expandafter #2\csname .=\xintFloor {\xINT_expr_unlock #3}\endcsname }%
1756 \let\xINT_flexpr_func_floor\xINT_expr_func_floor
1757 \def\xINT_iiexpr_func_floor #1#2#3%
1758 {% mais absurde si on ne peut pas avoir quotient comme input
1759     \expandafter #1\expandafter #2\csname.=\xintiFloor {\xINT_expr_unlock #3}\endcsname }%
1760 \def\xINT_expr_func_ceil #1#2#3%
1761     {\expandafter #1\expandafter #2\csname .=\xintCeil {\xINT_expr_unlock #3}\endcsname }%
1762 \let\xINT_flexpr_func_ceil\xINT_expr_func_ceil
1763 \def\xINT_iiexpr_func_ceil #1#2#3%
1764 {% mais absurde si on ne peut pas avoir quotient comme input
1765     \expandafter #1\expandafter #2\csname.=\xintiCeil {\xINT_expr_unlock #3}\endcsname }%
1766 \def\xINT_expr_func_sqr #1#2#3%
1767     {\expandafter #1\expandafter #2\csname.=\xintSqr {\xINT_expr_unlock #3}\endcsname }%

```



```

1768 \def\XINT_flexpr_func_sqr #1#2#3%
1769 {%
1770     \expandafter #1\expandafter #2\csname
1771     .=\XINTinFloatMul{\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
1772 }%
1773 \def\XINT_iiexpr_func_sqr #1#2#3%
1774     {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
1775 \def\XINT_expr_func_sqrt #1#2#3%
1776 {%
1777     \expandafter #1\expandafter #2\csname .=%
1778     \expandafter\XINT_expr_argandopt
1779     \romannumeral-`0\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\XINTinFloatSqrt
1780     \endcsname
1781 }%
1782 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
1783 \def\XINT_iiexpr_func_sqrt #1#2#3%
1784     {\expandafter #1\expandafter #2\csname.=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
1785 \def\XINT_iiexpr_func_sqrtr #1#2#3%
1786     {\expandafter #1\expandafter #2\csname.=\xintiiSqrtR {\XINT_expr_unlock #3}\endcsname }%
1787 \def\XINT_expr_func_round #1#2#3%
1788 {%
1789     \expandafter #1\expandafter #2\csname .=%
1790     \expandafter\XINT_expr_oneortwo
1791     \expandafter\xintiRound\expandafter\xintRound
1792     \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1793 }%
1794 \let\XINT_flexpr_func_round\XINT_expr_func_round
1795 \def\XINT_iiexpr_func_round #1#2#3%
1796 {%
1797     \expandafter #1\expandafter #2\csname .=%
1798     \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiRound
1799     \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1800 }%
1801 \def\XINT_expr_func_trunc #1#2#3%
1802 {%
1803     \expandafter #1\expandafter #2\csname .=%
1804     \expandafter\XINT_expr_oneortwo
1805     \expandafter\xintiTrunc\expandafter\xintTrunc
1806     \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1807 }%
1808 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
1809 \def\XINT_iiexpr_func_trunc #1#2#3%
1810 {%
1811     \expandafter #1\expandafter #2\csname .=%
1812     \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
1813     \romannumeral-`0\XINT_expr_unlock #3,,.\endcsname
1814 }%
1815 \def\XINT_expr_func_float #1#2#3%
1816 {%
1817     \expandafter #1\expandafter #2\csname .=%
1818     \expandafter\XINT_expr_argandopt
1819     \romannumeral-`0\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat

```

```

1820 \endcsname
1821 }%
1822 \let\XINT_flexpr_func_float\XINT_expr_func_float
1823 % \XINT_iiexpr_func_float not defined
1824 \def\XINT_expr_func_mod #1#2#3%
1825 {%
1826 \expandafter #1\expandafter #2\csname .=%
1827 \expandafter\expandafter\expandafter\xintMod
1828 \expandafter\XINT_expr_twoargs
1829 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1830 }%
1831 \def\XINT_flexpr_func_mod #1#2#3%
1832 {%
1833 \expandafter #1\expandafter #2\csname .=%
1834 \expandafter\XINTinFloatMod
1835 \romannumeral-`0\expandafter\XINT_expr_twoargs
1836 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1837 }%
1838 \def\XINT_iiexpr_func_mod #1#2#3%
1839 {%
1840 \expandafter #1\expandafter #2\csname .=%
1841 \expandafter\expandafter\expandafter\xintiMod
1842 \expandafter\XINT_expr_twoargs
1843 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1844 }%
1845 \def\XINT_expr_func_quo #1#2#3%
1846 {%
1847 \expandafter #1\expandafter #2\csname .=%
1848 \expandafter\expandafter\expandafter\xintiQuo
1849 \expandafter\XINT_expr_twoargs
1850 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1851 }%
1852 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
1853 \def\XINT_iiexpr_func_quo #1#2#3%
1854 {%
1855 \expandafter #1\expandafter #2\csname .=%
1856 \expandafter\expandafter\expandafter\xintiQuo
1857 \expandafter\XINT_expr_twoargs
1858 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1859 }%
1860 \def\XINT_expr_func_rem #1#2#3%
1861 {%
1862 \expandafter #1\expandafter #2\csname .=%
1863 \expandafter\expandafter\expandafter\xintiRem
1864 \expandafter\XINT_expr_twoargs
1865 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1866 }%
1867 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
1868 \def\XINT_iiexpr_func_rem #1#2#3%
1869 {%
1870 \expandafter #1\expandafter #2\csname .=%
1871 \expandafter\expandafter\expandafter\xintiRem

```

```

1872 \expandafter\XINT_expr_twoargs
1873 \romannumeral-`0\XINT_expr_unlock #3,\endcsname
1874 }%
1875 \def\XINT_expr_func_gcd #1#2#3%
1876 {\expandafter #1\expandafter #2\csname
1877     .=\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
1878 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
1879 \def\XINT_iiexpr_func_gcd #1#2#3%
1880 {\expandafter #1\expandafter #2\csname
1881     .=\xintiigCDof:csv{\XINT_expr_unlock #3}\endcsname }%
1882 \def\XINT_expr_func_lcm #1#2#3%
1883 {\expandafter #1\expandafter #2\csname
1884     .=\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
1885 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
1886 \def\XINT_iiexpr_func_lcm #1#2#3%
1887 {\expandafter #1\expandafter #2\csname
1888     .=\xintiilCMof:csv{\XINT_expr_unlock #3}\endcsname }%
1889 \def\XINT_expr_func_max #1#2#3%
1890 {\expandafter #1\expandafter #2\csname
1891     .=\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1892 \def\XINT_iiexpr_func_max #1#2#3%
1893 {\expandafter #1\expandafter #2\csname
1894     .=\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1895 \def\XINT_flexpr_func_max #1#2#3%
1896 {\expandafter #1\expandafter #2\csname
1897     .=\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1898 \def\XINT_expr_func_min #1#2#3%
1899 {\expandafter #1\expandafter #2\csname
1900     .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1901 \def\XINT_iiexpr_func_min #1#2#3%
1902 {\expandafter #1\expandafter #2\csname
1903     .=\xintiiminof:csv{\XINT_expr_unlock #3}\endcsname }%
1904 \def\XINT_flexpr_func_min #1#2#3%
1905 {\expandafter #1\expandafter #2\csname
1906     .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1907 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
1908 {\expandafter #1\expandafter #2\csname
1909     .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
1910 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
1911 {\expandafter #1\expandafter #2\csname
1912     .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
1913 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
1914 {\expandafter #1\expandafter #2\csname
1915     .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
1916 \expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
1917 {\expandafter #1\expandafter #2\csname
1918     .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1919 \expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
1920 {\expandafter #1\expandafter #2\csname
1921     .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1922 \expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
1923 {\expandafter #1\expandafter #2\csname

```

```

1924                                     .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1925 \def\XINT_expr_func_? #1#2#3%
1926     {\expandafter #1\expandafter #2\csname
1927         .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
1928 \let\XINT_flexpr_func_? \XINT_expr_func_?
1929 \let\XINT_iiexpr_func_? \XINT_expr_func_?
1930 \def\XINT_expr_func_! #1#2#3%
1931     {\expandafter #1\expandafter #2\csname.=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
1932 \let\XINT_flexpr_func_! \XINT_expr_func_!
1933 \let\XINT_iiexpr_func_! \XINT_expr_func_!
1934 \def\XINT_expr_func_not #1#2#3%
1935     {\expandafter #1\expandafter #2\csname.=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
1936 \let\XINT_flexpr_func_not \XINT_expr_func_not
1937 \let\XINT_iiexpr_func_not \XINT_expr_func_not
1938 \def\XINT_expr_func_all #1#2#3%
1939     {\expandafter #1\expandafter #2\csname
1940         .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
1941 \let\XINT_flexpr_func_all\XINT_expr_func_all
1942 \let\XINT_iiexpr_func_all\XINT_expr_func_all
1943 \def\XINT_expr_func_any #1#2#3%
1944     {\expandafter #1\expandafter #2\csname
1945         .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
1946 \let\XINT_flexpr_func_any\XINT_expr_func_any
1947 \let\XINT_iiexpr_func_any\XINT_expr_func_any
1948 \def\XINT_expr_func_xor #1#2#3%
1949     {\expandafter #1\expandafter #2\csname
1950         .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
1951 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
1952 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
1953 \def\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
1954 \def\XINT_expr_func_if #1#2#3%
1955     {\expandafter #1\expandafter #2\csname
1956         .=\expandafter\xintifNotZero:\romannumeral-`\0\XINT_expr_unlock #3,\endcsname }%
1957 \let\XINT_flexpr_func_if\XINT_expr_func_if
1958 \let\XINT_iiexpr_func_if\XINT_expr_func_if
1959 \def\xintifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
1960 \def\XINT_expr_func_ifsgn #1#2#3%
1961 {%
1962     \expandafter #1\expandafter #2\csname
1963         .=\expandafter\xintifSgn:\romannumeral-`\0\XINT_expr_unlock #3,\endcsname
1964 }%
1965 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
1966 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
1967 \def\XINT_expr_func_first #1#2#3%
1968     {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
1969         \romannumeral-`\0\XINT_expr_unlock #3,\endcsname }%
1970 \def\XINT_expr_func_firsta #1,#2^{#1}%
1971 \let\XINT_flexpr_func_first\XINT_expr_func_first
1972 \let\XINT_iiexpr_func_first\XINT_expr_func_first
1973 \def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
1974     {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
1975         \romannumeral-`\0\XINT_expr_unlock #3,\endcsname }%

```

```

1976 \def\XINT_expr_func_lasta #1,#2%
1977   {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
1978 \let\XINT_flexpr_func_last\XINT_expr_func_last
1979 \let\XINT_iiexpr_func_last\XINT_expr_func_last
1980 \def\XINT_expr_func_odd #1#2#3%
1981   {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
1982 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
1983 \def\XINT_iiexpr_func_odd #1#2#3%
1984   {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
1985 \def\XINT_expr_func_even #1#2#3%
1986   {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
1987 \let\XINT_flexpr_func_even\XINT_expr_func_even
1988 \def\XINT_iiexpr_func_even #1#2#3%
1989   {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
1990 \def\XINT_expr_func_nuple #1#2#3%
1991   {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname}%
1992 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
1993 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
1994 \def\XINT_expr_func_reversed #1#2#3%
1995   {\expandafter #1\expandafter #2\csname .=\xintReversed::csv
1996                                     {\XINT_expr_unlock #3}\endcsname}%
1997 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
1998 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
1999 \def\xintReversed::csv #1% should be done directly, of course
2000   {\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.

```

2001 \def\xintSeqB:f:csv #1#2%
2002   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2003 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral-`0#2#1!}%
2004 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2005   \expandafter\xint_gobble_i\romannumeral-`0%
2006   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2007   #1{#3}{#4}{#2}}%
2008 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2009 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2010                                     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi}%
2011 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2012                                     {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2013 \def\XINT_seqb:f:csv_p #1#2%
2014   {%
2015     \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2016     {#1}{#2}%
2017   }%
2018 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2019 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2020 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2021                                     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi}%

```

```

2022 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2023         {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2024 \def\XINT_seqb:f:csv_n #1#2%
2025 {%
2026     \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2027     {#1}{#2}%
2028 }%
2029 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2030 \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.

```

2031 \def\xintiiSeqb:f:csv #1#2%
2032     {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral-`0#2}{#1}}%
2033 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral-`0#2#1!}%
2034 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2035     \expandafter\xint_gobble_i\romannumeral-`0%
2036     \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2037         \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2038     #1{#3}{#4}}{#2}}%
2039 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2040     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2041 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2042     {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2043 \def\XINT_iiseqb:f:csv_p #1#2%
2044 {%
2045     \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2046     \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2047 }%
2048 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2049 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2050 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2051     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2052 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2053     {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2054 \def\XINT_iiseqb:f:csv_n #1#2%
2055 {%
2056     \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2057     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2058 }%

```

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.

```

2059 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2060     {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2061 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral-`0#2#1!}%
2062 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%

```

```

2063 \expandafter\xint_gobble_i\romannumeral-\`0%
2064 \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2065 #1{#3}{#4}{#2}}%
2066 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2067 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi}%
2068 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2069 \romannumeral0\XINTinfloatadd{#4}{#1}{#2}{#3,#1}{#4}}%
2070 \def\XINT_flseqb:f:csv_p #1#2%
2071 {%
2072 \xintifCmp {#1}{#2}%
2073 \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2074}%
2075 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2076 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2077 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2078 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi}%
2079 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2080 \romannumeral0\XINTinfloatadd{#4}{#1}{#2}{#3,#1}{#4}}%
2081 \def\XINT_flseqb:f:csv_n #1#2%
2082 {%
2083 \xintifCmp {#1}{#2}%
2084 \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2085}%

```

10.39 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

10.39.1 \xintApply::csv

Don't ask me what this if for. I wrote it in June, and we are now late October.

```

2086 \def\xintApply::csv #1#2%
2087 {\expandafter\XINT_applyon::_a\expandafter {\romannumeral-\`0#2}{#1}}%
2088 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},}%
2089 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral-\`0#3,{#1}{#2}}%
2090 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2091 \else\expandafter\XINT_applyon::_d\fi #1}%
2092 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral-\`0#2{#1},{#2}}%
2093 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}%
2094 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%

```

10.39.2 \xintApply::csv

```

2095 \def\xintApply::csv #1#2#3%
2096 {\expandafter\XINT_applyon::_a\expandafter{\romannumeral-\`0#2}{#1}{#3}}%
2097 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},}%
2098 \def\XINT_applyon::_b #1#2#3#4,%
2099 {\expandafter\XINT_applyon::_c \romannumeral-\`0#4,{#1}{#2}{#3}}%
2100 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2101 \else\expandafter\XINT_applyon::_d\fi #1}%
2102 \def\XINT_applyon::_d #1,#2#3%
2103 {\expandafter\XINT_applyon::_e\expandafter
2104 {\romannumeral-\`0\xintApply::csv {#2}{#1}{#3}}, {#2}{#3}}%
2105 \def\XINT_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4}, #1}%

```



```
2106 \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.

```
2107 \def\XINT_expr_RApply::csv #1#2#3#4%
2108   {\xintApply::csv{\xint_expandafter#1\xint_exchangetwo_keepbraces{#4}}{#3}}%
2109 \def\XINT_expr_LApply::csv #1#2#3#4{\xintApply::csv{#1{#3}}{#4}}%
2110 \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.

```
2111 \catcode`~ 12 % by the way, catcode is set to 3 in \XINTsetupcatcodes
2112 \catcode`$ 12 % $
2113 \def\XINT_xptwo_getab_b #1#2!#3%
2114   {\expandafter\XINT_xptwo_getab_c\romannumeral-`0#3!#1{#1#2}}%
2115 \def\XINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2116 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2117 \def\XINT_NEfork #1#2{\xint_ddfork
2118   #1#2\XINT_expr_RLApply::csv
2119   #1$\XINT_expr_RApply::csv% $
2120   $#2\XINT_expr_LApply::csv% $
2121   $${\XINT_NEfork_nn #1#2}% $$
2122   \krof }%
2123 \def\XINT_NEfork_nn #1#2#3#4{%
2124   \if #1#\xint_dothis{#3}\fi
2125   \if #1~\xint_dothis{#3}\fi
2126   \if #2#\xint_dothis{#3}\fi
2127   \if #2~\xint_dothis{#3}\fi
2128   \xint_orthat {\csname #4NE\endcsname }%
2129   }%
2130 \def\XINT_NEfork_one #1#2!#3#4#5#6{%
2131   \if ###1\xint_dothis {#3}\fi
2132   \if ~#1\xint_dothis {#3}\fi
2133   \if $#1\xint_dothis {\xintApply::csv{#3#5}}\fi %$
2134   \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2135 }%
2136 \toks0 {}%
2137 \xintFor #1 in {DivTrunc,iiDivTrunc,iiDivRound,Mod,iiMod,iRound,Round,iTrunc,Trunc,%
2138   Lt,Gt,Eq,LtorEq,GtorEq,Neq,AND,OR,XOR,iQuo,iRem,Add,Sub,Mul,Div,Pow,E,%
2139   iiAdd,iiSub,iiMul,iiPow,iiQuo,iiRem,iiE,SeqA::csv,iiSeqA::csv}\do
2140 {\toks0
2141   \expandafter{\the\toks0% no space! (makes shorter macro in the end)
2142   \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2143   \endcsname\expandafter\def\csname xint#1\endcsname #####1####2{%
```



```

2144 \expandafter\XINT_NEfork
2145 \romannumeral-\`0\expandafter\XINT_xptwo_getab_b
2146 \romannumeral-\`0####2!{####1}{~xint#1}{xint#1}}%
2147 }%
2148 }%
2149 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2150 Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%
2151 Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,Fac,iFac,Bool,Toggle}\do
2152 {\toks0
2153 \expandafter{\the\toks0%
2154 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2155 \endcsname\expandafter\def\csname xint#1\endcsname ####1{%
2156 \expandafter\XINT_NEfork_one\romannumeral-\`0####1!{~xint#1}{xint#1}{}}}%
2157 }%
2158 }%
2159 \xintFor #1 in {Add,Sub,Mul,Div,Power,E,Mod,SeqA::csv}\do
2160 {\toks0
2161 \expandafter{\the\toks0%
2162 \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2163 \csname XINTinFloat#1\expandafter\endcsname
2164 \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
2165 \expandafter\XINT_NEfork
2166 \romannumeral-\`0\expandafter\XINT_xptwo_getab_b
2167 \romannumeral-\`0####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}}%
2168 }%
2169 }%
2170 \xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits}\do
2171 {\toks0
2172 \expandafter{\the\toks0%
2173 \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2174 \endcsname\expandafter\def\csname #1\endcsname ####1{\expandafter
2175 \XINT_NEfork_one\romannumeral-\`0####1!{~#1}{#1}{}}}%
2176 }%
2177 }%
2178 \xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
2179 {\toks0
2180 \expandafter{\the\toks0% no space
2181 \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2182 \endcsname\expandafter\def\csname #1\endcsname ####1####2{%
2183 \expandafter\XINT_NEfork
2184 \romannumeral-\`0\expandafter\XINT_xptwo_getab_b
2185 \romannumeral-\`0####2!{####1}{$noexpand$#1}{#1}}}%
2186 }%
2187 }%
2188 \xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
2189 {\toks0
2190 \expandafter{\the\toks0% no space
2191 \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
2192 \endcsname\expandafter\def\csname #1::csv\endcsname ####1####2{%
2193 \expandafter\XINT_NEfork
2194 \romannumeral-\`0\expandafter\XINT_xptwo_getab_b
2195 \romannumeral-\`0####2!{####1}{$noexpand$#1:f:csv}{#1::csv}}}%

```

```

2196 }%
2197 }%
2198 \toks0
2199 \expandafter{\the\toks0
2200 \let\XINTinFloatNE\XINTinFloat
2201 \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
2202     \expandafter\XINT_NEfork_one
2203     \romannumeral-`0##2!\XINTinFloat[##1]\XINTinFloat\}{\{##1\}}%
2204 \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2205 \def\XINTinFloatSqrt [##1]##2{%
2206     \expandafter\XINT_NEfork_one
2207     \romannumeral-`0##2!\XINTinFloatSqrt[##1]\XINTinFloatSqrt\}{\{##1\}}%
2208 }%
2209 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2210               GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2211 {\toks0
2212 \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {\xint#1:csv}}%
2213 }%
2214 \xintFor #1 in {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2215 {\toks0
2216 \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {\xint#1:csv}}%
2217 }%
2218 \expandafter\def\expandafter\XINT_expr_redefinmacros\expandafter
2219 {\the\toks0
2220 \def\XINT_flexpr_noopt {\expandafter\XINT_flexpr_withopt_b\expandafter-%
2221     \romannumeral0\xintbarefloateval }%
2222 \def\XINT_flexpr_withopt_b ##1##2%
2223     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2224 \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!{\if -##2\expandafter\xint_firstoftwo
2225     \else\expandafter\xint_secondoftwo\fi \XINTdigits{\{##2##3\}\{##4\}}%
2226 \def\XINT_expr_print ##1{\expandafter\xintSPRaw::csv\expandafter
2227     {\romannumeral-`0\XINT_expr_unlock ##1}}%
2228 \def\XINT_iiexpr_print ##1{\expandafter\xintCSV::csv\expandafter
2229     {\romannumeral-`0\XINT_expr_unlock ##1}}%
2230 \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2231     {\romannumeral-`0\XINT_expr_unlock ##1}}%
2232 \def\xintCSV::csv {\xintCSV::csv }% spaces to separate from possible catcode 11
2233 \def\xintSPRaw::csv {\xintSPRaw::csv }% stuff after
2234 \def\xintPFloat::csv {\xintPFloat::csv }%
2235 \def\xintIsTrue::csv {\xintIsTrue::csv }%
2236 \def\xintRound::csv {\xintRound::csv }%

\def\XINTinFloat::csv {\XINTinFloat::csv } should not be needed.

2237 \def\xintReversed::csv {\xintReversed::csv }%
2238 \def\xintListSel:csv {\xintListSel:csv }%
2239 }%
2240 \toks0 {}%
2241 \def\xintNewExpr {\xint_NewExpr\xinttheexpr }%
2242 \def\xintNewFloatExpr {\xint_NewExpr\xintthefloatexpr }%
2243 \def\xintNewIExpr {\xint_NewExpr\xinttheiexpr }%
2244 \def\xintNewIIExpr {\xint_NewExpr\xinttheiiexpr }%
2245 \def\xintNewBoolExpr {\xint_NewExpr\xinttheboolexpr }%

```

```

2246 \def\XINT_newexpr_finish #1>{\noexpand\romannumeral-`0}%
2247 \def\xint_NewExpr #1#2[#3]%
2248 {%
2249   \begingroup
2250     \ifcase #3\relax
2251       \toks0 {\xdef #2}%
2252     \or \toks0 {\xdef #2##1}%
2253     \or \toks0 {\xdef #2##1##2}%
2254     \or \toks0 {\xdef #2##1##2##3}%
2255     \or \toks0 {\xdef #2##1##2##3##4}%
2256     \or \toks0 {\xdef #2##1##2##3##4##5}%
2257     \or \toks0 {\xdef #2##1##2##3##4##5##6}%
2258     \or \toks0 {\xdef #2##1##2##3##4##5##6##7}%
2259     \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8}%
2260     \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8##9}%
2261     \fi
2262     \xintexprSafeCatcodes
2263     \XINT_NewExpr #1%
2264  }%
2265 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
2266 \def\XINT_NewExpr %1%2@
2267 {@
2268   \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%2}@
2269   \XINT_expr_redefinmacros
2270   \def~{\$noexpand$}@
2271   \catcode`: 11 \catcode`_ 11
2272   \catcode`# 12 \catcode`~ 13 \escapechar 126
2273   \endlinechar -1 \everyeof {\noexpand }@
2274   \edef\XINT_tmppb
2275   {\scantokens\expandafter
2276   {\romannumeral-`0\expandafter%1\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2277   }@
2278   \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
2279   \the\toks0
2280   {\scantokens\expandafter{\expandafter\XINT_newexpr_finish\meaning\XINT_tmppb}}@
2281   \endgroup
2282 }@
2283 \catcode`% 14
2284 \let\xintexprRestoreCatcodes\empty
2285 \def\xintexprSafeCatcodes
2286 {%
2287   \edef\xintexprRestoreCatcodes {%
2288     \catcode59=\the\catcode59 % ;
2289     \catcode34=\the\catcode34 % "
2290     \catcode63=\the\catcode63 % ?
2291     \catcode124=\the\catcode124 % |
2292     \catcode38=\the\catcode38 % &
2293     \catcode33=\the\catcode33 % !
2294     \catcode93=\the\catcode93 % ]
2295     \catcode91=\the\catcode91 % [
2296     \catcode94=\the\catcode94 % ^
2297     \catcode95=\the\catcode95 % _

```

```

2298      \catcode47=\the\catcode47  % /
2299      \catcode41=\the\catcode41  % )
2300      \catcode40=\the\catcode40  % (
2301      \catcode42=\the\catcode42  % *
2302      \catcode43=\the\catcode43  % +
2303      \catcode62=\the\catcode62  % >
2304      \catcode60=\the\catcode60  % <
2305      \catcode58=\the\catcode58  % :
2306      \catcode46=\the\catcode46  % .
2307      \catcode45=\the\catcode45  % -
2308      \catcode44=\the\catcode44  % ,
2309      \catcode61=\the\catcode61  % =
2310      \catcode32=\the\catcode32\relax % space
2311  }%
2312      \catcode59=12  % ;
2313      \catcode34=12  % "
2314      \catcode63=12  % ?
2315      \catcode124=12  % |
2316      \catcode38=4   % &
2317      \catcode33=12  % !
2318      \catcode93=12  % ]
2319      \catcode91=12  % [
2320      \catcode94=7   % ^
2321      \catcode95=8   % _
2322      \catcode47=12  % /
2323      \catcode41=12  % )
2324      \catcode40=12  % (
2325      \catcode42=12  % *
2326      \catcode43=12  % +
2327      \catcode62=12  % >
2328      \catcode60=12  % <
2329      \catcode58=12  % :
2330      \catcode46=12  % .
2331      \catcode45=12  % -
2332      \catcode44=12  % ,
2333      \catcode61=12  % =
2334      \catcode32=10  % space
2335  }%
2336  \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
2337  \XINT_restorecatcodes_endinput%

xintkernel: 224. Total number of code lines: 12251. Each package starts with
xinttools:1044. circa 50 lines dealing with catcodes, package identification
xintcore:2227. and reloading management, also for Plain TEX. Version 1.1c of
xint:1403. 2015/09/12.
xintbinhex: 609.
xintgcd: 455.
xintfrac:2545.
xintseries: 386.
xintcfraction:1021.
xintexpr:2337.

```