

# The **xint** bundle: **xint**, **xintgcd**, **xintfrac**, **xintseries** and **xintcfrac**.

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.06 (2013/05/07)

Documentation generated from the source file  
with timestamp "07-05-2013 at 19:08:02 CEST"

## Abstract

The **xint** package implements with expandable  $\text{\TeX}$  macros the basic arithmetic operations of addition, subtraction, multiplication and division, as applied to arbitrarily long numbers represented as chains of digits with an optional minus sign. The **xintgcd** package provides implementations of the Euclidean algorithm and of its typesetting.

The **xintfrac** package extends the scope of **xint** to fractional numbers of arbitrary sizes ; **xintseries** provides some basic functionality for computing in an expandable manner partial sums of series and power series with fractional coefficients. And **xintcfrac** deals with the computation of continued fractions.

The packages may be used with Plain and with  $\text{\LaTeX}$ . Most macros, and all of those doing computations, work purely by expansion without assignments, and may thus be used almost everywhere in  $\text{\TeX}$ .

## Contents

<b>1</b>	<b>Raison d'être of these packages</b>	<b>3</b>
1	Some examples . . . . .	3
2	Expandability, (in)-efficiency . . . . .	5
3	Missing things . . . . .	6
4	Origins of the package . . . . .	6
<b>2</b>	<b>Expansions</b>	<b>6</b>
<b>3</b>	<b>Inputs and outputs</b>	<b>9</b>
<b>4</b>	<b>More on fractions</b>	<b>11</b>
<b>5</b>	<b><math>\text{\ifcase}</math>, <math>\text{\ifnum}</math>, ... constructs</b>	<b>13</b>
<b>6</b>	<b>Multiple outputs</b>	<b>13</b>
<b>7</b>	<b>Assignments</b>	<b>13</b>
<b>8</b>	<b>Utilities for expandable manipulations</b>	<b>15</b>
<b>9</b>	<b>Exceptions (error messages)</b>	<b>15</b>
<b>10</b>	<b>Common input errors when using the package macros</b>	<b>16</b>
<b>11</b>	<b>Package namespace</b>	<b>16</b>
<b>12</b>	<b>Loading and usage</b>	<b>17</b>

<b>13 Installation</b>	<b>17</b>
<b>14 Commands of the <code>xint</code> package</b>	<b>18</b>
1 <code>\xintRev</code> . . . . .	18
2 <code>\xintReverseOrder</code> . . . . .	19
3 <code>\xintRevWithBraces</code> . . . . .	19
4 <code>\xintLen</code> . . . . .	19
5 <code>\xintLength</code> . . . . .	19
6 <code>\xintCSVtoList</code> . . . . .	20
7 <code>\xintNthElt</code> . . . . .	20
8 <code>\xintListWithSep</code> . . . . .	20
9 <code>\xintApply</code> . . . . .	20
10 <code>\xintAssign</code> . . . . .	21
11 <code>\xintAssignArray</code> . . . . .	21
12 <code>\xintRelaxArray</code> . . . . .	21
13 <code>\xintDigitsOf</code> . . . . .	22
14 <code>\xintNum</code> . . . . .	22
15 <code>\xintSgn</code> . . . . .	22
16 <code>\xintOpp</code> . . . . .	22
17 <code>\xintAbs</code> . . . . .	22
18 <code>\xintAdd</code> . . . . .	22
19 <code>\xintSub</code> . . . . .	22
20 <code>\xintCmp</code> . . . . .	22
21 <code>\xintGeq</code> . . . . .	22
22 <code>\xintMax</code> . . . . .	23
23 <code>\xintMin</code> . . . . .	23
24 <code>\xintSum</code> . . . . .	23
25 <code>\xintSumExpr</code> . . . . .	23
26 <code>\xintMul</code> . . . . .	23
27 <code>\xintSqr</code> . . . . .	23
28 <code>\xintPrd</code> . . . . .	24
29 <code>\xintProductExpr</code> . . . . .	24
30 <code>\xintFac</code> . . . . .	24
31 <code>\xintPow</code> . . . . .	24
32 <code>\xintDivision</code> . . . . .	24
33 <code>\xintQuo</code> . . . . .	25
34 <code>\xintRem</code> . . . . .	25
35 <code>\xintFDg</code> . . . . .	25
36 <code>\xintLDg</code> . . . . .	25
37 <code>\xintMON, \xintMMON</code> . . . . .	25
38 <code>\xintOdd</code> . . . . .	25
39 <code>\xintDSL</code> . . . . .	25
40 <code>\xintDSR</code> . . . . .	25
41 <code>\xintDSH</code> . . . . .	26
42 <code>\xintDSHr, \xintDSx</code> . . . . .	26
43 <code>\xintDecSplit</code> . . . . .	26
44 <code>\xintDecSplitL</code> . . . . .	27
45 <code>\xintDecSplitR</code> . . . . .	27
<b>15 Commands of the <code>xintgcd</code> package</b>	<b>27</b>
1 <code>\xintGCD</code> . . . . .	27
2 <code>\xintBezout</code> . . . . .	27
3 <code>\xintEuclideanAlgorithm</code> . . . . .	28
4 <code>\xintBezoutAlgorithm</code> . . . . .	28
5 <code>\xintTypesetEuclideanAlgorithm</code> . . . . .	28
6 <code>\xintTypesetBezoutAlgorithm</code> . . . . .	29
<b>16 Commands of the <code>xintfrac</code> package</b>	<b>29</b>
1 <code>\xintLen</code> . . . . .	29
2 <code>\xintRaw</code> . . . . .	29
3 <code>\xintNumerator</code> . . . . .	30
4 <code>\xintDenominator</code> . . . . .	30
5 <code>\xintFrac</code> . . . . .	30
6 <code>\xintSignedFrac</code> . . . . .	30
7 <code>\xintFwOver</code> . . . . .	31
8 <code>\xintSignedFwOver</code> . . . . .	31
9 <code>\xintREZ</code> . . . . .	31
10 <code>\xintIrr</code> . . . . .	31
11 <code>\xintJrr</code> . . . . .	31
12 <code>\xintTrunc</code> . . . . .	31
13 <code>\xintiTrunc</code> . . . . .	32
14 <code>\xintRound</code> . . . . .	32
15 <code>\xintiRound</code> . . . . .	32
16 <code>\xintAdd</code> . . . . .	33
17 <code>\xintSub</code> . . . . .	33
18 <code>\xintMul</code> . . . . .	33
19 <code>\xintSqr</code> . . . . .	33
20 <code>\xintPow</code> . . . . .	33
21 <code>\xintSum, \xintSumExpr</code> . . . . .	33
22 <code>\xintPrd, \xintProductExpr</code> . . . . .	33
23 <code>\xintDiv</code> . . . . .	33
24 <code>\xintCmp</code> . . . . .	34
25 <code>\xintMax</code> . . . . .	34
26 <code>\xintMin</code> . . . . .	34
27 <code>\xintAbs</code> . . . . .	34
28 <code>\xintSgn</code> . . . . .	34

## 1 Raison d'être of these packages

29	<code>\xintOpp</code>	34	<code>MON, \xintMMON</code>	34
30	<code>\xintGeq, \xintDivision, \xintQuo,</code>	31	<code>\xintNum</code>	34
	<code>\xintRem, \xintFDg, \xintLDg, \xint-</code>			

### 17 Commands of the `xintseries` package 35

1	<code>\xintSeries</code>	35	6	<code>\xintPowerSeriesX</code>	44
2	<code>\xintiSeries</code>	36	7	<code>\xintFxFtPowerSeries</code>	45
3	<code>\xintRationalSeries</code>	37	8	<code>\xintFxFtPowerSeriesX</code>	46
4	<code>\xintRationalSeriesX</code>	40	9	Computing $\log 2$ and $\pi$	47
5	<code>\xintPowerSeries</code>	42			

### 18 Commands of the `xintcfrac` package 51

1	Package overview	51	13	<code>\xintCstoGC</code>	61
2	<code>\xintCFrac</code>	58	14	<code>\xintGCtoF</code>	62
3	<code>\xintGCfrac</code>	59	15	<code>\xintGCtoCv</code>	62
4	<code>\xintGCtoGCx</code>	59	16	<code>\xintCntoF</code>	63
5	<code>\xintFtoCs</code>	59	17	<code>\xintGCntoF</code>	63
6	<code>\xintFtoCx</code>	59	18	<code>\xintCntoCs</code>	63
7	<code>\xintFtoGC</code>	59	19	<code>\xintCntoGC</code>	64
8	<code>\xintFtoCC</code>	60	20	<code>\xintGCntoGC</code>	64
9	<code>\xintFtoCv</code>	60	21	<code>\xintiCstoF, \xintiGCtoF, \xint-</code>	
10	<code>\xintFtoCCv</code>	60		<code>iCstoCv, \xintiGCtoCv</code>	65
11	<code>\xintCstoF</code>	60	22	<code>\xintGCtoGC</code>	65
12	<code>\xintCstoCv</code>	61			

### 19 Package `xint` implementation 66

### 20 Package `xintgcd` implementation 144

### 21 Package `xintfrac` implementation 157

### 22 Package `xintseries` implementation 185

### 23 Package `xintcfrac` implementation 195

## 1 Raison d'être of these packages

### 1.1 Some examples

The main goal is to allow computations with integers and fractions of arbitrary sizes.<sup>1</sup>

Here are some examples:

`123456^99`:

`\xintiPow{123456}{99}`: 1147381811662665566332733300084545867470254804234  
261029758895454373590894697032027622647054266320583469027086822116813341  
525003240387627761689532221176342958720337622160886069158507571680197167

<sup>1</sup> Here and elsewhere, “arbitrarily big” means roughly numbers with numerators and denominators having strictly less than  $2^{31}=2147483648$  digits. Memory constraints from the `etex` or `pdftex` executables presumably limit even more the possible computations, not to mention the time taken by them.

## 1 Raison d'être of these packages

107120876970335365073774877787377849878160674999979836658125172327521549  
705416595667384911533326748541075607669718906235189958323778263699981109  
532393993235189992220564587812701495877679143167735437253858445948715594  
1215197416398666125896983737258716757394943552017095026186580166519903  
071841443223116967837696

1234/56789 with 1500 digits after the decimal point:

`\xintTrunc{1500}{1234/56789}\dots`: 0.021729560302171195125816619415731919  
914067865255595273732589057740055292398175703041081899663667259504481501  
699272746482593460001056542640300058109845216503196041486907675782281779  
922168025497895719241402384264558277131134550705242212400288788321682015  
883357692510873584673088098047157019845392593636091496592649985032312595  
749176777192766204722745602141259751008117769286305446477310746799556252  
091073975593865009068657662575498776171441652432689429290883797918610998  
608885523604923488703798270791878708904893553328989769145433094437302998  
820194051664935110672841571431087006286428709785345753579038194016446847  
100670904576590536899751712479529486344186374121748930250576696191163781  
718290514007994505978270439697828804874183380584268080085932134744404726  
267410942259944707601824296958918100336332740495518498300727253517406539  
998943457359699941890154783496803958513092324217718220077831974502104280  
758597615735441722868865449294757787599711211678317984116642307489126415  
326911901952842980154607406363908503407350014967687404250823222807233795  
277254397858740248991882230713694553522689253200443747908926024406134990  
931342337424501223828558347567310570709116202081389001391114476395076511  
296201729208121291095106446671010230854566905562697001179805948335064889  
327158428568912993713571290214654246420961805983553152899329095423409463  
100248287520470513655813625878251069749423303808836218281709485992005494  
021729560302171195125816619415731919914067865255595273732589057740055292  
398175703041081899663667...

0.99^{-100} with 200 digits after the decimal point:

`\xintTrunc{200}{\xintPow{.99}{-100}}\dots`: 2.7319990264290260038466717212  
578374355053516429385720708334305725082464555187053430448143013784806140  
368055624765019253070342696854891531946166122710159206719138403488514857  
4794308647096392073177979303...

Computation of a Bezout identity with  $7^{200}-3^{200}$  and  $2^{200}-1$ :

`\xintAssign\xintBezout`  
`\xintiSub {\xintiPow {7}{200}}{\xintiPow{3}{200}}`  
`\xintiSub {\xintiPow {2}{200}}{1}\to\A\B\U\V\D`  
`\U$\times$(7^{200}-3^{200})+\xintiOpp\V$\times$(2^{200}-1)=\D`  
-220045702773594816771390169652074193009609478853 $\times(7^{200}-3^{200})+14325894$   
936276369318591306832683204654744168633877140891583816724789919211328201  
191274624371580391777549768571912876931442406050669914563361432056776967  
74891 $\times(2^{200}-1)=1803403947125$

The Euclidean algorithm applied to 179,876,541,573 and 66,172,838,904:

`\xintTypesetEuclideanAlgorithm {179876541573}{66172838904}`  
 $179876541573 = 2 \times 66172838904 + 47530863765$   
 $66172838904 = 1 \times 47530863765 + 18641975139$

$$\begin{aligned}
 47530863765 &= 2 \times 18641975139 + 10246913487 \\
 18641975139 &= 1 \times 10246913487 + 8395061652 \\
 10246913487 &= 1 \times 8395061652 + 1851851835 \\
 8395061652 &= 4 \times 1851851835 + 987654312 \\
 1851851835 &= 1 \times 987654312 + 864197523 \\
 987654312 &= 1 \times 864197523 + 123456789 \\
 864197523 &= 7 \times 123456789 + 0
 \end{aligned}$$

The first example uses only the base module **xint**, the next two require loading also the **xintfrac** package, which deals with fractions. The last two require the **xintgcd** package. The bundle also comprises the **xintseries** package, for partial sums of series with fractional coefficients, and **xintcfrac** for continued fractions computations.

To see more of **xint** in action, jump to the [section 17](#) describing the commands of the **xintseries** package, especially as illustrated with the [traditional computations of  \$\pi\$  and  \$\log 2\$](#) , or also see the [computation of the convergents of  \$e\$](#)  made with the **xintcfrac** package.

Note that almost all of the computational results interspersed through the documentation are not hard-coded in the source of the document but just written there using the package macros, and were selected to not impact too much the compilation time.

## 1.2 Expandability, (in)-efficiency

For some initially circumstantial reasons (related to the origins of the package) all macros performing computations are compatible with an expansion-only context. This programming constraint of expandability weighs in a lot on the computation time as the macros may have to shuffle around data containing hundreds of tokens: our current implementation of addition doesn't even achieve linear computation time!

For addition, I try to optimize things for the 50-500 digits range. I have a variant of addition which is twice faster on numbers with 1000 digits, but it is slower than the original for numbers with less than 200 digits, and adding to the code a fork to choose what to do would mean overhead; besides it wouldn't be that easy to use this variant of addition in the other routines such as multiplication and division. And multiplication is anyhow too slow on numbers with 1000 digits, even dividing the time by two would not be enough.

Analogously to the not even linear addition, multiplication is worse than quadratic. Same causes, same effects. It is about cubic in the 100-1000 digits range: on my laptop, with release 1.04 of the bundle, squaring a randomly chosen number with 200 digits takes about 4 hundredths of a second, and squaring a 400 digits number about a quarter of a second. But squaring a 500 digits number is about 1.9 times as costly as one with 400 digits, and squaring a 1000 digits number is 8 times more expensive than for a 500 digits number (about 3.5 seconds). Implementation of a Gauss-Karatsuba scheme for intelligent multiplication has not been attempted so far. This kind of thing is motivating when one has instant memory access!

As clearly demonstrated long ago by the [pi computing file](#) by D. ROEGEL one can program T<sub>E</sub>X to compute with many digits at a much higher speed than what **xint** achieves: but, direct access to memory storage in one form or another seems a necessity for this kind of

## 2 Expansions

speed and one has to renounce at the complete expandability.<sup>2 3</sup>

### 1.3 Missing things

Currently **xint** does not provide ‘floating-point’ operations. The L<sup>A</sup>T<sub>E</sub>X3 project has implemented expandably floating-point computations with 16 significant figures (**l3fp**), including special functions such as exp, log, sine and cosine.

The most blatantly lacking thing in the **xint** project is a decent input parser, allowing to type in computations in a usual infix form such as, for example  $3*14+2.7^2-2*5$ . At this time, one has to type `\xintAdd {\xintMul {3}{14}}{\xintMul{\xintPow{2.7}{-2}}{5}}`. Previous computation results can be stored in macros and given as arguments to the package macros (see further on for important aspects of this).

### 1.4 Origins of the package

Package **bigintcalc** by HEIKO OBERDIEK already provides expandable arithmetic operations on “big integers”, exceeding the T<sub>E</sub>X limits (of  $2^{31}-1$ ), so why another<sup>4</sup> one?

I got started on this in early March 2013, via a thread on the `c.t.tex` usenet group, where ULRICH DIEZ used the previously cited package together with a macro (`\ReverseOrder`) which I had contributed to another thread.<sup>5</sup> What I had learned in this other thread thanks to interaction with ULRICH DIEZ and GL on expandable manipulations of tokens motivated me to try my hands at addition and multiplication.

I wrote macros `\bigMul` and `\bigAdd` which I posted to the newsgroup; they appeared to work comparatively fast. These first versions did not use the  $\varepsilon$ -T<sub>E</sub>X `\numexpr` primitive, they worked one digit at a time, having previously stored carry-arithmetic in 1200 macros.

I noticed that the **bigintcalc** package used `\numexpr` if available, but (as far as I could tell) not to do computations many digits at a time. Using `\numexpr` for one digit at a time for `\bigAdd` and `\bigMul` slowed them a tiny bit but avoided cluttering T<sub>E</sub>X memory with the 1200 macros storing pre-computed digit arithmetic. I wondered if some speed could be gained by using `\numexpr` to do four digits at a time for elementary multiplications (as the maximal admissible number for `\numexpr` has ten digits).

The present package is the result of this initial questioning.

**xint** requires the  $\varepsilon$ -T<sub>E</sub>X `\numexpr` primitive.

## 2 Expansions

Except for some specific macros dealing with assignments or typesetting, the bundle macros all work in expansion-only context. For example, with the following code snippet within `myfile.tex`:

---

<sup>2</sup>I could, naturally, be proven wrong!

<sup>3</sup>The LuaT<sub>E</sub>X project possibly makes endeavours such as **xint** appear even more insane that they are, in truth.

<sup>4</sup>this section was written before the **xintfrac** package; the author is not aware of another package allowing expandable computations with arbitrarily big fractions.

<sup>5</sup>the `\ReverseOrder` could be avoided in that circumstance, but it does play a crucial rôle here.

## 2 Expansions

```
\newwrite\outfile
\immediate\openout\outfile \jobname-out\relax
\immediate\write\outfile {\xintQuo{\xintPow{2}{1000}}{\xintFac{100}}}
% \immediate\closeout\outfile
```

the tex run creates a file `myfile-out.tex` containing the decimal representation of the integer quotient  $2^{1000}/100!$ . Such macros can also be used inside a `\csname... \endcsname`, and of course in an `\edef`.

Furthermore the package macros give their final results in two expansion steps. They expand ‘fully’ (the first token of) their arguments so that they can be arbitrarily chained. Hence

```
\xintLen{\xintQuo{\xintPow{2}{1000}}{\xintFac{100}}}
```

expands in two steps and tells us that  $[2^{1000}/100!]$  has 144 digits. This is not so many, let us print them here: 114813249641507505482278393872551066259805517784186172883663478065826541894704737970419535798876630484358265060061503749531707793118627774829601.

For the sake of typesetting this documentation and not have big numbers extend into the margin and go beyond the page physical limits, I use these commands (not provided by the package):

```
\def\allowsplits #1{\ifx #1\relax \else #1\hskip 0pt plus 1pt \relax
\expandafter\allowsplits\fi}%
\def\printnumber #1{\expandafter\expandafter\expandafter
\allowsplits #1\relax }%
% Expands twice before printing.
```

The `\printnumber` macro is not part of the package and would need additional thinking for more general use. It may be used as `\printnumber {\xintQuo{\xintPow{2}{1000}}{\xintFac{100}}}`, or as `\printnumber\mynumber` if the macro `\mynumber` was previously defined via an `\edef`, as for example:

```
\edef\mynumber {\xintQuo{\xintPow {2}{1000}}{\xintFac{100}}}
```

or as `\expandafter\printnumber\expandafter{\mynumber}`, if the macro `\mynumber` is defined by a `\newcommand` or a `\def` (see below [item 3](#) for the underlying expansion issue; adding four `\expandafter`’s to `\printnumber` would allow to use it directly as `\printnumber\mynumber` with a `\mynumber` itself defined via a `\def` or `\newcommand`).

Just to show off, let’s print 300 digits (after the decimal point) of the decimal expansion of  $0.7^{-25}$ :

```
\printnumber {\xintTrunc {300}{\xintPow{.7}{-25}}}\dots
7456.7399858373588376091197273418534888533391015795335848127921083943053
372463282318528184075067673537414907699005707631450150814361392271887429
728266459679048963813786168152282545091498481687823094059852453689236788
162567790831369386453622401300364894165620674502128974076460364640746484
84309937461948589...
```

This computation uses **xintfrac** which extends to fractions the basic arithmetic operations defined for integers by **xint**.

Important points, to be noted, related to the expansion of arguments:

- CHANGED! (1.06) →
1. the macros expand ‘fully’ their arguments, this means that they expand the first token seen (for each argument), then expand again, etc..., until something un-expandable



## 2 Expansions

such as a digit or a brace is hit against.<sup>6</sup> This example

```
\def\x{12}\def\y{34}\xintAdd {\x}{\x\y}
```

is *not* a legal construct, as the `\y` will remain untouched by expansion and not get converted into the digits which are expected by the sub-routines of `\xintAdd`. It works here by sheer luck as the `\y` gets expanded inside a `\numexpr`. But this would fail in general: if you need a more complete (expandable...) expansion of your initial input, you should use the `\bigintcalcNum` macro from the `bigintcalc` package. Or, outside of an expandable-only context, just massage your inputs through `\edef`'s.

2. Unfortunately, after `\def\x {12}`, one can not use just `-\x` as input to one of the package macros: the rules above explain that the expansion will act only on the minus sign, hence do nothing. The only way is to use the `\xintOpp` macro, which replaces a number with its opposite.
3. With the definition

```
\def\AplusBC #1#2#3{\xintAdd {#1}{\xintMul {#2}{#3}}}
```

one obtains an expandable macro producing the expected result, not in two, but rather in three steps: a first expansion is consumed by the macro expanding to its definition. The new expansion policy starting with the package release 1.06 allows to use this inside other package 'primitives' or also similar macros: `\xintAdd {\AplusBC {1}{2}{3}}{4}` does work and returns `11/1[0]`.<sup>7</sup>

If, for some reason, it is important to create a macro expanding in two steps to its final value, the solution is to use the *lowercase* form of `\xintAdd`:

```
\def\AplusBC #1#2#3{\romannumeral0\xintadd {#1}{\xintMul {#2}{#3}}}
```

and then `\AplusBC` will share the same properties as do the other `xint` 'primitive' macros.

The lowercase form is *only* for the external highest level of chained commands. All `xint` provided public macros have such a lowercase form. To more fully imitate the `xint` standard habits, the example above should thus be treated via the creation of two macros:

```
\def\aplusbc #1#2#3{\xintadd {#1}{\xintMul {#2}{#3}}}
```

```
\def\AplusBC {\romannumeral0\aplusbc}
```

Or, for people using the  $\LaTeX$  vocabulary:

```
\newcommand{\aplusbc}[3]{\xintadd {#1}{\xintMul {#2}{#3}}}
```

```
\newcommand{\AplusBC}{\romannumeral0\aplusbc}
```

This then allows further definitions of macros expanding in two steps only, such as:

```
\def\aplusbcsquared #1#2#3{\aplusbc {#1}{#2}{\xintSqr{#3}}}
```

```
\def\AplusBCSquared {\romannumeral0\aplusbcsquared}
```

```
\newcommand\myalgebra [6]{\xintmul {\AplusBC {#1}{#2}{#3}}{\AplusBC {#4}{#5}{#6}}}
```

```
\newcommand\MyAlgebra {\romannumeral0\myalgebra}
```

The `\romannumeral0` things above look like an invitation to hacker's territory; if it is not important that the macro expands in two steps only, there is no reason to follow these

<sup>6</sup>the knowledgeable people will have recognized `\romannumeral-0`

<sup>7</sup>this strange thing is because this document uses `xintfrac`, and we have printed the raw output of addition which is automatically a fraction.



### 3 Inputs and outputs

guidelines. Just chain arbitrarily the package macros, and the new ones will be completely expandable and usable one within the other.

**New with 1.06:** those macro arguments which are intrinsically constrained to obey the  $\text{\TeX}$  bounds on integers (see the next section) are now systematically fed to a  $\text{\numexpr}$ , hence they will be subjected to a complete expansion, registers are allowed, and things such as  $\text{\mycount}+\text{\myothercount}*17$  become admissible arguments.

## 3 Inputs and outputs

The arguments to most of the **xint** macros are of three types:

1. ‘short’ integers, *i.e.* less than (or equal to) in absolute value 2,147,483,647. I will refer to this as the ‘ $\text{\TeX}$ ’ or ‘ $\text{\numexpr}$ ’ limit. This is the case for arguments which serve to count or index something. It is also the case for exponent in the power function and for the argument to the factorial function. The bounds have been (arbitrarily) lowered to 999,999,999 and 999,999 respectively for the latter cases. When the argument exceeds the  $\text{\TeX}$  bound (either positively or negatively), an error will originate from a  $\text{\numexpr}$  expression and it may sometimes be followed by a more specific error ‘message’ from a package macros.
2. ‘long’ integers, which are the bread and butter of the package commands. They are signed integers with a practically illimited number of digits. Theoretically though, most of the macros require that the number of digits itself be less than the  $\text{\TeX}$ - $\text{\numexpr}$  bound (more precisely  $2^{31-9}$ ). Some macros, such as addition when **xintfrac** has not been loaded, do not measure first the length of their arguments and could theoretically be used with ‘gigantic’ integers with a larger number of digits. However memory constraints from the  $\text{\TeX}$  implementation probably exclude such inputs. Concretely though, multiplying out two 1000 digits numbers is already a longish operation.
3. ‘fractions’: they become available after having loaded the **xintfrac** package. Their format on input will be described next, a fraction has a numerator, a forward slash and then a denominator.

$\text{\TeX}$ ’s count registers cannot serve directly as arguments to the package macros accepting ‘long numbers’ or fractions on input: they must be prefixed by  $\text{\the}$  or  $\text{\number}$ . The same for  $\text{\numexpr}$  expressions. However, count registers and  $\text{\numexpr}$  expressions are allowed in arguments intrinsically constrained to obey the  $\text{\TeX}$  bounds.

NEW WITH 1.06 →

NEW WITH 1.06 → The package macros first operate a ‘full’ expansion of their arguments, as explained above: only the first token is repeatedly expanded until no more is possible.

NEW WITH 1.06 → On the other hand, this expansion is a *complete one* for those arguments which are constrained to obey the  $\text{\TeX}$  bounds on numbers, as they are systematically inserted inside a  $\text{\numexpr}\dots\text{\relax}$  expression.

The allowed input formats for ‘long numbers’ and ‘fractions’ are:

### 3 Inputs and outputs

1. the strict format is when `xintfrac` is not loaded. The number should be a string of digits, optionally preceded by a unique minus sign. The first digit can be zero only if the number is zero. A plus sign is not accepted. There is a macro `\xintNum` which normalizes to this form an input having arbitrarily many minus and plus signs, followed by a string of zeros, then digits:

`\xintNum {+---+-----+---00000000009876543210}=-9876543210`

Note that `-0` is not legal input and will confuse `xint` (but not `\xintNum` which even accepts an empty input).

2. the relaxed format is when `xintfrac` is loaded. Most macros are then modified to accept inputs of the form  $A/B$  (or just  $A$ ), where  $A$  and  $B$  will be automatically given to the normalizing `\xintNum` macro. Additionally, each of  $A$  and  $B$  may have an optional decimal point with digits following it. Here is an example:

`\xintAdd {+--0367.8920280/-++278.289287}{-109.2882/+270.12898}`

Incidentally this evaluates to

$=-129792033529284840/7517400124223726[-1]$

$=-6489601676464242/3758700062111863$  (irreducible)

$=-1.72655481129771093694248704898677881556360055242806...$

where the second line was produced with `\xintIrr` and the next with `\xintTrunc {50}` to get fifty digits of the decimal expansion following the decimal mark.

Of course, even when `xintfrac` is loaded, some macros can not treat fractions on input. With release 1.05 they have, for the most part, been also extended to accept the relaxed format as long as the denominator turns out to be a divisor of the numerator (once the decimal points are suitably transformed into powers of ten). For example it used to be the case with the earlier releases that `\xintQuo {100/2}{12/3}` would not work (the macro `\xintQuo` computes a euclidean quotient). It now does, because its arguments are in truth integers.

A number can start directly with a decimal point:

`\xintPow{-.3/.7}{11}=-177147/1977326743[0]`

It is also licit to use  $A/B$  as input if each of  $A$  and  $B$  expands (in the sense previously described) to a “decimal number” as exemplified above by the numerators and denominators. Or one may have just one macro  $C$  which expands to such a “fraction with optional decimal points”, or mixed things such as  $A\ 245/7.77$ , where the numerator will be the concatenation of the expansion of  $A$  and 245. But, as explained already  $123A$  is a no-go.

Loading `xintfrac` not only relaxes the format of the inputs; it also modifies the format of the outputs: except when filtered through the `\xintIrr` macro, a fraction is always output in the  $A/B[n]$  form (which stands for  $(A/B)10^n$ ; some macros print  $A[n]$  in certain circumstances when the denominator is one). The  $A$  and  $B$  may end in zeros (*i.e.*,  $n$  does not represent all powers of ten), and will generally have a common factor. The denominator  $B$  is always strictly positive.

A macro `\xintFrac` is provided for the typesetting (math-mode only) of such a ‘raw’ output. Of course, the `\xintFrac` itself is not accepted as input to the package macros.

Direct user input of things such as  $16000/289072[17]$  or  $3[-4]$  is authorized. It is even possible to use  $A/B[17]$  if  $A$  expands to 16000 and  $B$  to 289072, or  $A$  if  $A$  expands to  $3[-4]$ . However, NEITHER the numerator NOR the denominator may then have a decimal point. And, for this format, ONLY the numerator may carry a UNIQUE minus sign (and

IMPORTANT! →

## 4 More on fractions

no superfluous leading zeros; and NO plus sign).

The, more demanding, format with a power of ten represented by a number within square brackets is the output format used by (almost all) **xintfrac** macros dealing with fractions. It is allowed for user input but the parsing is minimal and it is very important to follow the above rules. This reduced flexibility, compared to the format without the square brackets, allows chaining package macros without too much speed impact, as they always output computation results in the  $A/B[n]$  form (or  $A[n]$ ).

All computations done by **xintfrac** on fractions are exact. Inputs containing decimal points do not make the package switch to a (currently non-existent) ‘floating-point’ mode. The inputs, however long, are always converted into an exact internal representation.

Generally speaking, there should be no spaces among the digits in the inputs. Although most would be harmless in most macros, there are some cases where spaces could break havoc. So the best is to avoid them entirely.

It would certainly be nice to be able to input directly expressions such as  $2.3 \cdot 5.6^3 - 17728/189.5$ , but this is not possible. One must use, for example:

```
\xintSub {\xintMul {2.3}{\xintPow {5.6}{3}}} {17728/189.5}
```

or, an option in this case is:

```
\xintAdd {\xintPrd {{2.3}{5.6}{5.6}{5.6}}}{-17728/189.5}
```

Syntax such as `\xintMul\A\B` is accepted and equivalent<sup>8</sup> to `\xintMul {\A}{\B}`. Or course `\xintAdd\xintMul\A\B\C` does not work, the product operation must be put within braces: `\xintAdd{\xintMul\A\B}\C`. It would be nice to have a functional form `\add(x,\mul(y,z))` but this is not provided by the package. Arguments must be either within braces or a single control sequence.

Note that `-` and `+` may serve only as unary operators, on *explicit* numbers. They can not serve to prefix macros evaluating to such numbers.

## 4 More on fractions

With package **xintfrac** loaded, the routines `\xintAdd`, `\xintSub`, `\xintMul`, `\xintPow`, `\xintSum`, `\xintPrd` are modified to allow fractions on input,<sup>9 10 11 12</sup> and always produce on output a fractional number  $f=A/B[n]$  where  $A$  and  $B$  are integers, with  $B$  positive, and  $n$  is a signed “small” integer (*i.e.* less in absolute value than  $2^{31}-9$ ). This represents  $(A/B)$  times  $10^n$ . The fraction  $f$  may be, and generally is, reducible, and  $A$  and  $B$  may

<sup>8</sup>see however near the end of [this later section](#) for the important difference when used in contexts where  $\TeX$  expects a number, such as following an `\ifcase` or an `\ifnum`.

<sup>9</sup>of course, the power function does not accept a fractional exponent. Or rather, does not expect, and errors will result if one is provided.

<sup>10</sup>macros `\xintiAdd`, `\xintiSub`, `\xintiMul`, `\xintiPow`, `\xintiSum`, `\xintiPrd` are the original ones dealing only with integers. They are available as synonyms, also when **xintfrac** is not loaded.

<sup>11</sup>also `\xintCmp`, `\xintSgn`, `\xintOpp`, `\xintAbs`, `\xintMax`, `\xintMin` are extended to fractions and have their integer-only initial synonyms.

<sup>12</sup>and `\xintQuo`, `\xintRem`, `\xintDivision`, `\xintGeq`, `\xintFDg`, `\xintLDg`, `\xintOdd`, `\xintMON`, `\xintMMON` all accept a fractional input as long as it reduces to an integer. Note that `\xintGeq` still only works on (non-negative) integers, to compare fractions one must use `\xintCmp`.

## 4 More on fractions

well end up with zeros (*i.e.*  $n$  does not contain all powers of 10). Conversely, this format is accepted on input (and is parsed more quickly than fractions containing decimal points).<sup>13</sup>

The `\xintiAdd`, `\xintiSub`, `\xintiMul`, `\xintiPow`, `\xintiSum`, `\xintiPrd`, etc... are the original un-modified integer-only versions. They have less parsing overhead.

The macro `\xintRaw` prints the fraction in A/B form, with the trailing  $[n]$  converted into explicit zeros either at the numerator or the denominator. The B is printed even if it has value 1.

Conversely (sort of), the macro `\xintREZ` puts all powers of ten into the  $[n]$  (REZ stands for remove zeros). Here also, the B is printed even if it has value 1.

The macro `\xintIrr` reduces the fraction to its irreducible form C/D (thus, without a trailing  $[0]$ ), and it prints only the C if D=1. The macro `\xintNum` from `xint` is extended to act like `\xintIrr` but additionally raises an error when the fraction doesn't simplify to an integer. When one knows that necessarily the result of a computation is an integer, and one wants to get rid of the denominator and trailing  $[n]$ , one can thus use `\xintIrr` or `\xintNum` (if the fraction has internally a denominator equal to 1, this is quickly identified, there is little overhead; else, the denominator will be discovered in the next step to be a divisor of the numerator).

The macro `\xintTrunc{N}{f}` prints<sup>14</sup> the decimal expansion of  $f$  with  $N$  digits after the decimal point.<sup>15</sup> Currently, it does not verify that  $N$  is non-negative and strange things could happen with a negative  $N$ . Of course a negative  $f$  is no problem, needless to say. When the original fraction is negative and its truncation has only zeros, it is printed as  $-0.0\dots0$ , with  $N$  zeros following the decimal point:

```
\xintTrunc {5}{\xintPow {-13}{-9}}=-0.00000
```

```
\xintTrunc {20}{\xintPow {-13}{-9}}=-0.00000000009429959537
```

The output always contains a decimal point (even for  $N=0$ ) followed by  $N$  digits, except when the original fraction was zero. In that case the output is  $0$ , with no decimal point.

```
\xintTrunc {10}{\xintSum {{1/2}{1/3}{1/5}{-31/30}}}=0
```

The output of `\xintTrunc` may of course serve as input to the other macros. And this is almost necessary when summing hundreds of terms of a series with fractional coefficients, as the exact rational number quickly becomes quite big (when doing the sum from  $n=1$  to  $n=1000$  of  $1/n$ , the raw denominator is  $1000!$ , which has 2568 digits); but for less than fifty terms with small denominators it is often possible to work with the exact value without too much toll on the compilation time.

The macro `\xintiTrunc{N}{f}` is like `\xintTrunc{N}{f}` followed by multiplication by  $10^N$ . Thus, it outputs an integer in a format acceptable by the integer-only macros. This is also convenient when computing partial sums of series, with a fixed number of digits after the decimal point: it is a bit faster to sum with `\xintiSeries` the integers produced by `\xintiTrunc{N}` than it is to use the general `\xintSeries` on the decimal numbers

<sup>13</sup>at each stage of the computations, the sum of  $n$  and the length of A, or of the absolute value of  $n$  and the length of B, must be kept less than  $2^{31}-9$ .

<sup>14</sup>'prints' does not at all mean that this macro is designed for typesetting; I am just using the verb here in analogy to the effect of the functioning of a computing software in console mode. The package does not provide any 'printing' facility, besides its rudimentary `\xintFrac` and `\xintFwOver` math-mode only macros. To deal with really long numbers, some macros are necessary as  $\text{\TeX}$  by default will print a long number on a single line extending beyond the page limits. The `\printnumber` command used in this documentation is just one way to address this problem, some other method should be used if it is important that digits occupy the same width always.

<sup>15</sup>the current release does not provide a macro to get the period of the decimal expansion.

## 5 `\ifcase`, `\ifnum`, ... constructs

produced by `\xintTrunc{N}`. These latter macros belong to the **xintseries** package.

Needless to say when using `\xintTrunc` or `\xintiTrunc` on intermediate computations the ending digits of the final result are, pending further analysis, only indications of those of the fraction an exact computation would have produced.

To get the integer part of the decimal expansion of  $f$ , use `\xintiTrunc{0}{f}`:

```
\xintiTrunc {0}{\xintPow {1.01}{100}}=2
\xintiTrunc {0}{\xintPow{0.123}{-10}}=1261679032
```

## 5 `\ifcase`, `\ifnum`, ... constructs

When using things such as `\ifcase \xintSgn{A}` one has to leave a space after the closing brace for  $\TeX$  to stop its scanning for a number: once  $\TeX$  has finished expanding `\xintSgn{A}` and has so far obtained either 1, 0, or -1, a space (or something ‘unexpandable’) must stop it looking for more digits. Using `\ifcase\xintSgnA` without the braces is very dangerous, because the blanks (including the end of line) following `\A` will be skipped and not serve to stop the number which `\ifcase` is looking for. With `\defA{1}`:

```
\ifcase \xintSgnA 0\or OK\else ERROR\fi ---> gives ERROR
\ifcase \xintSgn{A} 0\or OK\else ERROR\fi ---> gives OK
```

## 6 Multiple outputs

Some macros have an output consisting of more than one number, each one is then within braces. Examples of multiple-output macros are `\xintDivision` which gives first the quotient and then the remainder of euclidean division, `\xintBezout` from the **xintgcd** package which outputs five numbers, `\xintFtoCv` from the **xintcfac** package which returns the list of the convergents of a fraction, ... the next two sections explain ways to deal, expandably or not, with such outputs.

See the subsection 14.43 for a rare example of a bundle macro which may return an empty string, or a number prefixed by a chain of zeros. This is the only situation where a macro from the package **xint** may output something which could require parsing through `\xintNum` before further processing by the other (integer-only) package macros from **xint**.

## 7 Assignments

It might not be necessary to maintain at all times complete expandability. For example why not allow oneself the two definitions `\edefA {\xintQuo{100}{3}}` and `\edefB {\xintRem {100}{3}}`. A special syntax is provided to make these things more efficient, as the package provides `\xintDivision` which computes both quotient and remainder at the same time:

```
\xintAssign\xintDivision{100}{3}\toA\B
\xintAssign\xintDivision{\xintiPow {2}{1000}}{\xintFac{100}}\toA\B
gives \meaningA: macro:->1148132496415075054822783938725510662598055177
84186172883663478065826541894704737970419535798876630484358265060061503
749531707793118627774829601 and \meaningB: macro:->54936294521339832251
```

## 7 Assignments

38128786223912807341050049847605059532189961231327664902288388132878702  
444582075129603152041054804964625083138567652624386837205668069376.

Another example (which uses a macro from the **xintgcd** package):

`\xintAssign\xintBezout{357}{323}\to\A\B\U\VD`

is equivalent to setting  $\A$  to 357,  $\B$  to 323,  $\U$  to -9,  $\V$  to -10, and  $\D$  to 17. And indeed  $(-9) \times 357 - (-10) \times 323 = 17$  is a Bezout Identity.

`\xintAssign\xintBezout{3570902836026}{200467139463}\to\A\B\U\VD`

gives then  $\U$ : macro:  $\rightarrow 5812117166$ ,  $\V$ : macro:  $\rightarrow 103530711951$  and  $\D=3$ .

When one does not know in advance the number of tokens, one can use `\xintAssignArray` or its synonym `\xintDigitsOf`:

`\xintDigitsOf\xintiPow{2}{100}\to\Out`

This defines  $\Out$  to be macro with one parameter,  $\Out\{0\}$  gives the size  $N$  of the array and  $\Out\{n\}$ , for  $n$  from 1 to  $N$  then gives the  $n$ th element of the array, here the  $n$ th digit of  $2^{100}$ , from the most significant to the least significant. As usual, the generated macro  $\Out$  is completely expandable (in two steps). As it wouldn't make much sense to allow indices exceeding the  $\TeX$  bounds, the macros created by `\xintAssignArray` put their argument inside a  $\numexpr$ , so it is completely expanded and may be a count register, not necessarily prefixed by  $\the$  or  $\number$ . Consider the following code snippet:

```
\newcount\cnta
\newcount\cntb
\begingroup
\xintDigitsOf\xintiPow{2}{100}\to\Out
\cnta = 1
\cntb = 0
\loop
\advance \cntb \xintiSqr{\Out{\cnta}}
\ifnum \cnta < \Out{0}
\advance\cnta 1
\repeat
```

$|2^{100}|$  ( $=\xintiPow\{2\}\{100\}$ ) has  $\Out\{0\}$  digits and the sum of their squares is  $\the\cntb$ . These digits are, from the least to the most significant:  $\cnta = \Out\{0\}$   
 $\loop \Out\{\cnta\}\ifnum \cnta > 1 \advance\cnta -1 , \repeat$ .  
 $\endgroup$

$2^{100}$  ( $=1267650600228229401496703205376$ ) has 31 digits and the sum of their squares is 679. These digits are, from the least to the most significant: 6, 7, 3, 5, 0, 2, 3, 0, 7, 6, 9, 4, 1, 0, 4, 9, 2, 2, 8, 2, 2, 0, 0, 6, 0, 5, 6, 7, 6, 2, 1.

We used a group in order to release the memory taken by the  $\Out$  array: indeed internally, besides  $\Out$  itself, additional macros are defined which are  $\Out0$ ,  $\Out00$ ,  $\Out1$ ,  $\Out2$ , ...,  $\OutN$ , where  $N$  is the size of the array (which is the value returned by  $\Out\{0\}$ ; the digits are parts of the names not arguments).

The command `\xintRelaxArray\Out` sets all these macros to  $\relax$ , but it was simpler to put everything withing a group.

Needless to say `\xintAssign`, `\xintAssignArray` and `\xintDigitsOf` do not do any check on whether the macros they define are already defined.

In the example above, we deliberately broke all rules of complete expandability, but had we wanted to compute the sum of the digits, not the sum of the squares, we could just have



written:

```
\xintiSum{\xintiPow{2}{100}}=115
```

Indeed, `\xintiSum` is usually used as in

```
\xintiSum{{123}{-345}}{\xintFac{7}}{\xintiOpp{\xintRem{3347}{591}}}=4426
```

but in the example above each digit of  $2^{100}$  is treated as would have been a summand enclosed within braces, due to the rules of  $\text{\TeX}$  for parsing macro arguments.

Note that `{-\xintRem{3347}{591}}` is not a valid input, because the expansion will apply only to the minus sign and leave unaffected the `\xintRem`. So we used `\xintiOpp` which replaces a number with its opposite.

As a last example with `\xintAssignArray` here is one line extracted from the source code of the `xintgcd` macro `\xintTypesetEuclideanAlgorithm`:

```
\xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
```

This is done inside a group. After this command `\U{1}` contains the number  $N$  of steps of the algorithm (not to be confused with `\U{0}=2N+4` which is the number of elements in the `\U` array), and the GCD is to be found in `\U{3}`, a convenient location between `\U{2}` and `\U{4}` which are (absolute values of the twice expansion of) the initial inputs. Then follow  $N$  quotients and remainders from the first to the last step of the algorithm. The `\xintTypesetEuclideanAlgorithm` macro organizes this data for typesetting: this is just an example of one way to do it.

## 8 Utilities for expandable manipulations

**EXTENDED (1.06)** → The package now has more utilities to deal expandably with ‘lists of things’, which were treated un-expandably in the previous section with `\xintAssign` and `\xintAssignArray`: `\xintRev`, `\xintReverseOrder`, `\xintLen` and `\xintLength` since the first release, `\xintApply` and `\xintListWithSep` since 1.04, `\xintRevWithBraces`, `\xintCSVtoList`, `\xintNthElt` now with 1.06.

As an example the following code uses only expandable operations:

```
|2^{100}| (= \xintiPow {2}{100}) has \xintLen{\xintiPow {2}{100}} digits
and the sum of their squares is
\xintiSum{\xintApply {\xintiSqr}{\xintiPow {2}{100}}}.
These digits are, from the least to the most significant:
\xintListWithSep {, }{\xintRev{\xintiPow {2}{100}}}. The thirteenth most
significant digit is \xintNthElt{13}{\xintiPow {2}{100}}. The seventh
least significant one is \xintNthElt{7}{\xintRev{\xintiPow {2}{100}}}.
```

$2^{100}$  (=1267650600228229401496703205376) has 31 digits and the sum of their squares is 679. These digits are, from the least to the most significant: 6, 7, 3, 5, 0, 2, 3, 0, 7, 6, 9, 4, 1, 0, 4, 9, 2, 2, 8, 2, 2, 0, 0, 6, 0, 5, 6, 7, 6, 2, 1. The thirteenth most significant digit is 8. The seventh least significant one is 3.

Of course, with an earlier `\edef\z{\xintiPow {2}{100}}`, using `\z` in place of `\xintiPow {2}{100}` everywhere would spare the CPU some repetitions.

## 9 Exceptions (error messages)

In situations such as division by zero, the package will insert in the  $\text{\TeX}$  processing an undefined control sequence (we copy this method from the `bigintcalc` package). This



will trigger the writing to the log of a message signaling an undefined control sequence. The name of the control sequence is the message. The error is raised *before* the end of the expansion so as to not disturb further processing of the token stream, after completion of the operation. Generally the problematic operation will output a zero. Possible such error message control sequences:

```
\xintError:ArrayIndexIsNegative
\xintError:ArrayIndexBeyondLimit
\xintError:FactorialOfNegativeNumber
\xintError:FactorialOfTooBigNumber
\xintError:DivisionByZero
\xintError:NaN
\xintError:FractionRoundedToZero
\xintError:NotAnInteger
\xintError:ExponentTooBig
\xintError:TooBigDecimalShift
\xintError:TooBigDecimalsplit
\xintError:NoBezoutForZeros
```

## 10 Common input errors when using the package macros

Here is a list of common input errors. Some will cause compilation errors, others are more annoying as they may pass through unsignaled.

- using - to prefix some macro: `-\xintiSqr{35}/271`.
- using one pair of braces too many `\xintIrr{\{\xintiPow {3}{13}\}/243}` (the computation goes through with no error signaled, but the result is completely wrong).
- using [] and decimal points at the same time `1.5/3.5[2]`.
- using [] with a sign in the denominator `3/-5[7]`.
- making a mistake in a macro name `\xintProduct {{2}{3}{4}}`. Well I should \let it to be `\xintPrd...` at least such errors are not dangerous because they do provoke compilation errors.
- loading **xintfrac** and using expressions previously producing integers as numerators or denominators: `\edef\x{\xintMul {3}{5}/\xintMul{7}{9}}`. The problem is that this expands to `15[0]/63[0]` which is invalid on input. Using this `\x` in a fraction macro will most certainly cause a compilation error, with its usual arcane and undecipherable accompanying message.

## 11 Package namespace

Inner macros of **xint**, **xintgcd**, **xintfrac**, **xintseries**, and **xintcfrac** all begin either with `\XINT@` or with `\xint@`. The package public commands all start with `\xint`. The major forms have their initials capitalized, and lowercase forms, prefixed with `\romannumeral0`, allow definitions of further macros expanding in only two steps to their final outputs. Some other control sequences are used only as delimiters, and left undefined, they may have been defined elsewhere, their meaning doesn't matter and is not touched.

## 12 Loading and usage

```
Usage with LaTeX: \usepackage{xint}
                  \usepackage{xintgcd}    % (loads xint)
                  \usepackage{xintfrac}    % (loads xint)
                  \usepackage{xintseries}  % (loads xintfrac)
                  \usepackage{xintcfrac}   % (loads xintfrac)

Usage with TeX:   \input xint.sty\relax
                  \input xintgcd.sty\relax % (loads xint)
                  \input xintfrac.sty\relax % (loads xint)
                  \input xintseries.sty\relax % (loads xintfrac)
                  \input xintcfrac.sty\relax % (loads xintfrac)
```

We have added, directly copied from packages by HEIKO OBERDIEK, a mechanism of re-load and  $\varepsilon$ -TeX detection, especially for Plain TeX. As  $\varepsilon$ -TeX is required, the executable `tex` can not be used, `etex` or `pdftex` (version 1.40 or later) or ..., must be invoked.

Furthermore, the packages **xintgcd** and **xintfrac** will check for the previous loading of **xint**, and will try to load it if this was not already done. Similarly **xintseries** and **xintcfrac** do the necessary loading of **xintfrac**.

Also inspired from the HEIKO OBERDIEK packages we have included a complete catcode protection mechanism. The packages may be loaded in any catcode configuration satisfying these requirements: the percent is of category code comment character, the backslash is of category code escape character, digits have category code other and letters have category code letter. Nothing else is assumed, and the previous configuration is restored after the loading of each one of the packages.

This is for the loading of the packages. For the actual use of the macros, note that when feeding them with negative numbers the minus sign must have category code other, as is standard. Similarly the slash used for inputting fractions must be of category other, as usual. And the square brackets also must be of category code other, if used on input.

The components of the **xint** bundle presuppose that the usual `\space` and `\empty` macros are pre-defined, which is the case in Plain TeX as well as in LaTeX.

Lastly, the macros `\xintRelaxArray` (of **xint**) and `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` (of **xintgcd**) use `\loop`, both Plain and LaTeX incarnations are compatible. `\xintTypesetBezoutAlgorithm` also uses the `\endgraf` macro.

## 13 Installation

Run `tex` or `latex` on `xint.dtx`.

This will extract the style files `xint.sty`, `xintgcd.sty`, `xintfrac.sty`, `xintseries.sty`, `xintcfrac.sty` (and `xint.ins`). Files with the same names and in the same repertory will be overwritten. The `tex` (not `latex`) run will stop with the complaint that it does not understand `\NeedsTeXFormat`, but the style files will already have been extracted by that time.

Alternatively, run `tex` or `latex` on `xint.ins` if available.

To get `xint.pdf` run `pdflatex` thrice on `xint.dtx`

```

xint.sty |
xintgcd.sty |
xintfrac.sty | --> TDS:tex/generic/xint/
xintseries.sty |
xintcfrac.sty |
xint.dtx --> TDS:source/generic/xint/
xint.pdf --> TDS:doc/generic/xint/

```

It may be necessary to then refresh the TeX installation filename database.

## 14 Commands of the **xint** package

`{N}` (or also `{M}`) stands for a normalised number within braces as described in the documentation, or for a control sequence expanding (in the sense previously described) to such a number (without the braces!), or for a control sequence within braces expanding to such a number, or for material within braces which expands to such a number after repeated expansions of the first token. A count register or `\numexpr` expression must thus come first and be prefixed by `\the` or `\number`.

The letter `x` stands for something which will be inserted in-between a `\numexpr` and a `\relax`. It will thus be completely expanded and must give an integer obeying the TeX bounds. Thus, it may be for example a count register, or itself a `\numexpr` expression, or just a number written explicitly with digits or something like `4*\count 255 + 17`, etc...

Some of these macros are extended by **xintfrac** to accept fractions on input, and, generally, to output a fraction. This will be mentioned and the original macro `\xintAbc` remains then available under the name `\xintiAbc`. There are also macros such as `\xintQuo` or `\xintNum` which are made to accept fractions on input, under the condition that this fraction turns out to be an integer, but still do produce pure integers without any forward slash mark nor trailing `[n]`. Again the original is still available with an additional ‘i’ in the name, in case it is important to skip the parsing, but here the output format is the same. See the **xintfrac** documentation for more information.

The integer-only macros are a bit more efficient, even for simple things such as determining the sign of a (long) number, as there is always some overhead due to the parsing the fraction format on input; however except if one does thousands of times the same computation with various inputs, there is no need in general to employ the integer-only variants. The exception is when the context requires that the macro returns a (possibly long) integer, with no forward slash nor trailing `[n]`. This may be because they are used in **xint** macros which remain strictly integer-only on input, such as `\xintDecSplit`, or in places where a (short) number is expected by TeX such as after an `\ifnum` or inside a `\numexpr`.

IMPORTANT!

### 14.1 `\xintRev`

`\xintRev{N}` will revert the order of the digits of the number, keeping the optional sign. Leading zeros resulting from the operation are not removed (see the `\xintNum` macro for this). As all other macros dealing with numbers it first expands its argument (in the manner

described, triggered by a `\romannumeral-‘0’`.

```
\xintRev{-123000}=-000321
\xintNum{\xintRev{-123000}}=-321
```

## 14.2 `\xintReverseOrder`

`\xintReverseOrder{<list>}` does not do any expansion of its argument and just reverses the order of the tokens in the ‘list’.<sup>16</sup> Brace pairs encountered are removed once and the enclosed material does not get reverted. Spaces are gobbled.

```
\xintReverseOrder{\xintDigitsOf\xintiPow {2}{100}\to\Stuff}
gives: \Stuff\to1002\xintiPow\xintDigitsOf
```

## 14.3 `\xintRevWithBraces`

New in release 1.06.

`\xintRevWithBraces{<list>}` first does the expansion of its argument (which thus may be macro), then it reverses the order of the tokens, or braced material, it encounters, adding a pair of braces to each (thus, maintaining brace pairs already existing). Spaces (in-between external brace pairs) are gobbled. This macro is mainly thought out for use on a ‘list’ of such braced material; with such a list as argument the expansion will only hit against the first opening brace, hence do nothing, and the braced stuff may thus be macros one does not want to expand.

```
\edef\x{\xintRevWithBraces{12345}}
\meaning\x:macro:->{5}{4}{3}{2}{1}
\edef\y{\xintRevWithBraces\x}
\meaning\y:macro:->{1}{2}{3}{4}{5}
```

The examples above could be defined with `\edef`’s because the braced material did not contain macros. Alternatively:

```
\expandafter\def\expandafter\w\expandafter
{\romannumeral0\xintrevwithbraces{{\A}{\B}{\C}{\D}{\E}}}}
\meaning\w:macro:->{\E }{\D }{\C }{\B }{\A }
```

The private macro `\XINT@RWB` does the same job without the initial expansion of its argument.

## 14.4 `\xintLen`

`\xintLen{N}` returns the length of the number, not counting the sign.

```
\xintLen{-12345678901234567890123456789}=29
```

Extended by **xintfrac** to fractions: the length of  $A/B[n]$  is the length of  $A$  plus the length of  $B$  plus the absolute value of  $n$  and minus one (an integer input as  $N$  is internally  $N/1[0]$  so the minus one means that the extended `\xintLen` behaves the same as the original for integers). The whole thing should sum up to less than circa  $2^{31}$ .

## 14.5 `\xintLength`

`\xintLength{<list>}` does not do any expansion of its argument and just counts how many

<sup>16</sup>the argument is not a token list variable, just a ‘list’ of tokens.

tokens there are (possibly none). Things enclosed in braces count as one.

```
\xintLength {\xintiPow {2}{100}}=3
≠ \xintLen {\xintiPow {2}{100}}=31
```

## 14.6 **\xintCSVtoList**

New with release 1.06.

`\xintCSVtoList{a,b,c...,z}` returns `{a}{b}{c}...{z}`. The argument may be a macro. It is first expanded: this means that if the argument is `a,b,...`, then `a`, if a macro, will be expanded which may or may not be a good thing. Chains of contiguous spaces are collapsed by the  $\TeX$  scanning into single spaces.

```
\xintCSVtoList {1,2,a , b ,c d,x,y }->{1}{2}{a }{ b }{c d}{x}{y }
\def\y{a,b,c,d,e}\xintCSVtoList\y->{a}{b}{c}{d}{e}
```

The private macro `\XINT@CSVtoL` does the same job without the initial expansion.

## 14.7 **\xintNthElt**

New in release 1.06.

`\xintNthElt{i}{list}` gets (expandably) the *i*th element of the list. The ‘list’ may be a macro expanding to the ‘list’ (which is just a sequence of tokens; spaces are gobbled as the macro will get the required element as an undelimited macro parameter). The first argument *i* may be a  $\TeX$  count register (it will be given to a `\numexpr`). The seeked element is returned with one pair of braces removed (if initially present).

```
\xintNthElt {37}{\xintFac {100}}=9
```

is the thirty-seventh digit of 100!.

```
\xintNthElt {10}{\xintFtoCv {566827/208524}}=1457/536[0]
```

is the tenth convergent of 566827/208524 (uses **xintcfrc** package). Error cases, where *i* < 0 or is larger than the number of elements in the list are kept silent; the macro then returns nothing. Perhaps this will be changed in future versions.

```
\xintNthElt {7}{\xintCSVtoList {1,2,3,4,5,6,7,8,9}}=7
```

The private macro `\XINT@NthElt` does the same job without first expanding its second argument.

## 14.8 **\xintListWithSep**

New with release 1.04.

`\xintListWithSep{sep}{list}` just inserts the given separator *sep* in-between all elements of the given list. One level of braces is removed. An empty input gives an empty output, a singleton gives a singleton, the separator is used starting with at least two elements. The ‘list’ argument may be a macro: it is expanded.

```
\xintListWithSep{:}{\xintFac {20}}=2:4:3:2:9:0:2:0:0:8:1:7:6:6:4:0:0:0:0
```

The private macro `\XINT@LWS` does the same job without the initial expansion.

## 14.9 **\xintApply**

New in release 1.04.

`\xintApply{\macro}{list}` applies the one parameter command `\macro` to each item in the ‘list’ (no separator) given as second argument. Each item is given in turn as parameter to `\macro` which is (fully, as usual) expanded, and the result is braced. On output, a new list with these braced results. The ‘list’ may itself be some macro expanding (in the previously described way) to the list of tokens to which the command `\macro` will be applied. For example, if the ‘list’ expands to some positive number, then each digit will be replaced by the result of applying `\macro` on it.

```
\def\macro #1{\the\numexpr 9-#1\relax}
\xintApply\macro{\xintFac {20}}=7567097991823359999
```

The private macro `XINT@Apply` does the same job without the first initial expansion providing the ‘list’.

### 14.10 `\xintAssign`

`\xintAssign<braced things>\to<as many cs as they are things>` defines (without checking if something gets overwritten) the control sequences on the right of `\to` to be the complete expansions of the successive things on the left of `\to` enclosed within braces.

Important: a ‘full’ expansion (as previously described) is applied first to the material in front of `\xintAssign`.

As a special exception, if after this initial expansion a brace does not immediately follows `\xintAssign`, it is assumed that there is only one control sequence to define and it is then defined to be the complete expansion of the entire material between `\xintAssign` and `\to`.

```
\xintAssign\xintDivision{10000000000000}{133333333}\to\Q\R
\meaning\Q: macro:->7500,\meaning\R: macro:->2500
\xintAssign\xintiPow {7}{13}\to\SevenToThePowerThirteen
\SevenToThePowerThirteen=96889010407
```

Of course this macro and its cousins completely break usage in pure expansion contexts, as assignments are made via the `\edef` primitive.

### 14.11 `\xintAssignArray`

Changed in release 1.06.

`\xintAssignArray<braced things>\to\myArray` first expands fully the first token then defines `\myArray` to be a macro with one parameter, such that `\myArray{x}` expands in two steps (which provoke the full expansion of the ‘short’ number `{x}`, given to a `\numexpr`) to give the Nth braced thing, itself completely expanded. `\myArray{0}` returns the number `M` of elements of the array so that the successive elements are `\myArray{1}`, ..., `\myArray{M}`.

```
\xintAssignArray\xintBezout {1000}{113}\to\Bez
```

will set `\Bez{0}` to 5, `\Bez{1}` to 1000, `\Bez{2}` to 113, `\Bez{3}` to -20, `\Bez{4}` to -177, and `\Bez{5}` to 1:  $(-20) \times 1000 - (-177) \times 113 = 1$ .

### 14.12 `\xintRelaxArray`

`\xintRelaxArray\myArray` sets to `\relax` all macros which were defined by the previous `\xintAssignArray` with `\myArray` as array name.

**14.13 \xintDigitsOf**

This is a synonym for `\xintAssignArray`, to be used to define an array giving all the digits of a given number.

`\xintDigitsOf\xintiPow {7}{500}\to\digits`

$7^{500}$  has `\digits{0}=423` digits, and the 123rd among them (starting from the most significant) is `\digits{123}=3`.

**14.14 \xintNum**

`\xintNum{N}` removes chains of plus or minus signs, followed by zeros.

`\xintNum{+-----000000000367941789479}=-367941789479`

Extended by **xintfrac** to accept also a fraction on input, as long as it reduces to an integer after division of the numerator by the denominator.

`\xintNum{123.48/-0.03}=-4116`

**14.15 \xintSgn**

`\xintSgn{N}` returns 1 if the number is positive, 0 if it is zero and -1 if it is negative. Extended by **xintfrac** to fractions.

**14.16 \xintOpp**

`\xintOpp{N}` returns the opposite  $-N$  of the number  $N$ . Extended by **xintfrac** to fractions.

**14.17 \xintAbs**

`\xintAbs{N}` returns the absolute value of the number. Extended by **xintfrac** to fractions.

**14.18 \xintAdd**

`\xintAdd{N}{M}` returns the sum of the two numbers. Extended by **xintfrac** to fractions.

**14.19 \xintSub**

`\xintSub{N}{M}` returns the difference  $N-M$ . Extended by **xintfrac** to fractions.

**14.20 \xintCmp**

`\xintCmp{N}{M}` returns 1 if  $N > M$ , 0 if  $N = M$ , and -1 if  $N < M$ . Extended by **xintfrac** to fractions.

**14.21 \xintGeq**

`\xintGeq{N}{M}` returns 1 if the absolute value of the first number is at least equal to the absolute value of the second number. If  $|N| < |M|$  it returns 0.



**14.22 \xintMax**

`\xintMax{N}{M}` returns the largest of the two in the sense of the order structure on the relative integers (*i.e.* the right-most number if they are put on a line with positive numbers on the right): `\xintiMax {-5}{-6}=-5`. Extended by **xintfrac** to fractions.

**14.23 \xintMin**

`\xintMin{N}{M}` returns the smallest of the two in the sense of the order structure on the relative integers (*i.e.* the left-most number if they are put on a line with positive numbers on the right): `\xintiMin {-5}{-6}=-6`. Extended by **xintfrac** to fractions.

**14.24 \xintSum**

`\xintSum{⟨braced things⟩}` after expanding its argument expects to find a sequence of tokens (or braced material). Each is expanded (with the usual meaning), and the sum of all these numbers is returned.

```
\xintiSum{{123}{-98763450}}{\xintFac{7}}{\xintiMul{3347}{591}}=-96780210
\xintiSum{1234567890}=45
```

An empty sum is no error and returns zero: `\xintiSum {}=0`. A sum with only one term returns that number: `\xintiSum {{-1234}}=-1234`. Attention that `\xintiSum {-1234}` is not legal input and will make the  $\TeX$  run fail. On the other hand `\xintiSum {1234}=10`. Extended by **xintfrac** to fractions.

**14.25 \xintSumExpr**

`\xintSumExpr⟨braced things⟩\relax` is to what `\xintSum` expands. The argument is then expanded (with the usual meaning) and should give a list of braced quantities or macros, each one will be expanded in turn.

```
\xintiSumExpr {123}{-98763450}}{\xintFac{7}}{\xintiMul{3347}{591}}\relax=-96780210
```

Note: I am not so happy with the name which seems to suggest that the + sign should be used instead of braces. Perhaps this will change in the future.

Extended by **xintfrac** to fractions.

**14.26 \xintMul**

Modified in release 1.03.

`\xintMul{N}{M}` returns the product of the two numbers. Starting with release 1.03 of **xint**, the macro checks the lengths of the two numbers and then activates its algorithm with the best (or at least, hoped-so) choice of which one to put first. This makes the macro a bit slower for numbers up to 50 digits, but may give substantial speed gain when one of the number has 100 digits or more. Extended by **xintfrac** to fractions.

**14.27 \xintSqr**

`\xintSqr{N}` returns the square. Extended by **xintfrac** to fractions.

**14.28 \xintPrd**

`\xintPrd{⟨braced things⟩}` after expanding its argument expects to find a sequence of tokens (or braced material). Each is expanded (with the usual meaning), and the product of all these numbers is returned.

```
\xintiPrd{{-9876}}{\xintFac{7}}{\xintiMul{3347}{591}}=-98458861798080
\xintiPrd{123456789123456789}=131681894400
```

An empty product is no error and returns 1: `\xintiPrd {}=1`. A product reduced to a single term returns this number: `\xintiPrd {-1234}=-1234`. Attention that `\xintiPrd {-1234}` is not legal input and will make the  $\text{\TeX}$  compilation fail. On the other hand `\xintiPrd {1234}=24`.

```
2^{200}3^{100}7^{100}
=\xintiPrd {\xintiPow {2}{200}}{\xintiPow {3}{100}}{\xintiPow {7}{100}}
=2678727931661577575766279517007548402324740266374015348974459614815426
412965499490000444007240765727130000165312076406545621180143571994015903
343539244028212438966822248927862988084382716133376
=\xintiPow {\xintiMul {\xintiPow {42}{9}}{43008}}{10}
```

Extended by **xintfrac** to fractions.

**14.29 \xintProductExpr**

`\xintProductExpr{⟨argument⟩}\relax` is to what `\xintPrd` expands ; its argument is expanded (with the usual meaning) and should give a list of braced numbers or macros. Each will be expanded when it is its turn.

```
\xintiProductExpr 123456789123456789\relax=131681894400
```

Note: I am not so happy with the name which seems to suggest that the `*` sign should be used instead of braces. Perhaps this will change in the future.

Extended by **xintfrac** to fractions.

**14.30 \xintFac**

`\xintFac{x}` returns the factorial. It is an error if the argument is negative or at least  $10^6$ . It is not recommended to launch the computation of things such as  $100000!$ , if you need your computer for other tasks. Note that the argument is of the `x` type, it must obey the  $\text{\TeX}$  bounds, but on the other hand may involve count registers and even arithmetic operations as it will be completely expanded inside a `\numexpr`.

**14.31 \xintPow**

`\xintPow{N}{x}` returns  $N^x$ . When `x` is zero, this is 1. Some cases (`N` zero and `x` negative,  $|N| > 1$  and `x` negative,  $|N| > 1$  and `x` at least  $10^9$ ) make **xint** throw errors.

Extended by **xintfrac** to fractions. Of course, negative exponents do not then cause errors anymore.

**14.32 \xintDivision**

`\xintDivision{N}{M}` returns {quotient `Q`}{remainder `R`}. This is euclidean division:  $N = QM + R$ ,  $0 \leq R < |M|$ . So the remainder is always non-negative and the for-

mula  $N = QM + R$  always holds independently of the signs of  $N$  or  $M$ . Division by zero is of course an error (even if  $N$  vanishes) and returns  $\{0\}\{0\}$ .

This macro is integer only (with **xintfrac** loaded it accepts fractions on input, but they must be integers in disguise) and not to be confused with the **xintfrac** macro `\xintDiv` which divides one fraction by another.

### 14.33 `\xintQuo`

`\xintQuo{N}{M}` returns the quotient from the euclidean division. When both  $N$  and  $M$  are positive one has `\xintQuo{N}{M}=\xintiTrunc {0}{N/M}` (using package **xintfrac**). With **xintfrac** loaded it accepts fractions on input, but they must be integers in disguise.

### 14.34 `\xintRem`

`\xintRem{N}{M}` returns the remainder from the euclidean division. With **xintfrac** loaded it accepts fractions on input, but they must be integers in disguise.

### 14.35 `\xintFDg`

`\xintFDg{N}` returns the first digit (most significant) of the decimal expansion.

### 14.36 `\xintLDg`

`\xintLDg{N}` returns the least significant digit. When the number is positive, this is the same as the remainder in the euclidean division by ten.

### 14.37 `\xintMON`, `\xintMMON`

New in version 1.03.

`\xintMON{N}` returns  $(-1)^N$  and `\xintMMON{N}` returns  $(-1)^{N-1}$ .

`\xintMON {-280914019374101929}=-1`, `\xintMMON {-280914019374101929}=1`

### 14.38 `\xintOdd`

`\xintOdd{N}` is 1 if the number is odd and 0 otherwise.

### 14.39 `\xintDSL`

`\xintDSL{N}` is decimal shift left, *i.e.* multiplication by ten.

### 14.40 `\xintDSR`

`\xintDSR{N}` is decimal shift right, *i.e.* it removes the last digit (keeping the sign). For a positive number, this is the same as the quotient from the euclidean division by ten (of course, done in a more efficient manner than via the general division algorithm). For  $N$  from  $-9$  to  $-1$ , the macro returns 0.

**14.41 \xintDSH**

`\xintDSH{x}{N}` is parametrized decimal shift. When  $x$  is negative, it is like iterating `\xintDSL |x|` times (*i.e.* multiplication by  $10^{-x}$ ). When  $x$  positive, it is like iterating `\DSR x` times (and is more efficient of course), and for a non-negative  $N$  this is thus the same as the quotient from the euclidean division by  $10^x$ .

**14.42 \xintDSHr, \xintDSx**

New in release 1.01.

`\xintDSHr{x}{N}` expects  $x$  to be zero or positive and it returns then a value  $R$  which is correlated to the value  $Q$  returned by `\xintDSH{x}{N}` in the following manner:

- if  $N$  is positive or zero,  $Q$  and  $R$  are the quotient and remainder in the euclidean division by  $10^x$  (obtained in a more efficient manner than using `\xintDivision`),
- if  $N$  is negative let  $Q1$  and  $R1$  be the quotient and remainder in the euclidean division by  $10^x$  of the absolute value of  $N$ . If  $Q1$  does not vanish, then  $Q=-Q1$  and  $R=R1$ . If  $Q1$  vanishes, then  $Q=0$  and  $R=-R1$ .
- for  $x=0$ ,  $Q=N$  and  $R=0$ .

So one has  $N = 10^x Q + R$  if  $Q$  turns out to be zero or positive, and  $N = 10^x Q - R$  if  $Q$  turns out to be negative, which is exactly the case when  $N$  is at most  $-10^x$ .

`\xintDSx{x}{N}` for  $x$  negative is exactly as `\xintDSH{x}{N}`, *i.e.* multiplication by  $10^{-x}$ . For  $x$  zero or positive it returns the two numbers  $\{Q\}{R}$  described above, each one within braces. So  $Q$  is `\xintDSH{x}{N}`, and  $R$  is `\xintDSHr{x}{N}`, but computed simultaneously.

```
\xintAssign\xintDSx {-1}{-123456789}\to\M
\meaning\M: macro:->-1234567890.
\xintAssign\xintDSx {-20}{123456789}\to\M
\meaning\M: macro:->123456789000000000000000000000.
\xintAssign\xintDSx {0}{-123004321}\to\Q\R
\meaning\Q: macro:->-123004321, \meaning\R: macro:->0.
\xintDSH {0}{-123004321}=-123004321, \xintDSHr {0}{-123004321}=0
\xintAssign\xintDSx {6}{-123004321}\to\Q\R
\meaning\Q: macro:->-123, \meaning\R: macro:->4321.
\xintDSH {6}{-123004321}=-123, \xintDSHr {6}{-123004321}=4321
\xintAssign\xintDSx {8}{-123004321}\to\Q\R
\meaning\Q: macro:->-1, \meaning\R: macro:->23004321.
\xintDSH {8}{-123004321}=-1, \xintDSHr {8}{-123004321}=23004321
\xintAssign\xintDSx {9}{-123004321}\to\Q\R
\meaning\Q: macro:->0, \meaning\R: macro:->-123004321.
\xintDSH {9}{-123004321}=0, \xintDSHr {9}{-123004321}=-123004321
```

**14.43 \xintDecSplit**

This has been modified in release 1.01.

## 15 Commands of the **xintgcd** package

`\xintDecSplit{x}{N}` cuts the number into two pieces (each one within a pair of enclosing braces). First the sign if present is *removed*. Then, for  $x$  positive or null, the second piece contains the  $x$  least significant digits (*empty* if  $x=0$ ) and the first piece the remaining digits (*empty* when  $x$  equals or exceeds the length of  $N$ ). Leading zeros in the second piece are not removed. When  $x$  is negative the first piece contains the  $|x|$  most significant digits and the second piece the remaining digits (*empty* if  $|x|$  equals or exceeds the length of  $N$ ). Leading zeros in this second piece are not removed. So the absolute value of the original number is always the concatenation of the first and second piece.

This macro's behavior for  $N$  non-negative is final and will not change. I am still hesitant about what to do with the sign of a negative  $N$ .

```
\xintAssign\xintDecSplit {0}{-123004321}\to\L\R
\meaning\L: macro:->123004321, \meaning\R: macro:->.
\xintAssign\xintDecSplit {5}{-123004321}\to\L\R
\meaning\L: macro:->1230, \meaning\R: macro:->04321.
\xintAssign\xintDecSplit {9}{-123004321}\to\L\R
\meaning\L: macro:->, \meaning\R: macro:->123004321.
\xintAssign\xintDecSplit {10}{-123004321}\to\L\R
\meaning\L: macro:->, \meaning\R: macro:->123004321.
\xintAssign\xintDecSplit {-5}{-12300004321}\to\L\R
\meaning\L: macro:->12300, \meaning\R: macro:->004321.
\xintAssign\xintDecSplit {-11}{-12300004321}\to\L\R
\meaning\L: macro:->12300004321, \meaning\R: macro:->.
\xintAssign\xintDecSplit {-15}{-12300004321}\to\L\R
\meaning\L: macro:->12300004321, \meaning\R: macro:->.
```

### 14.44 `\xintDecSplitL`

`\xintDecSplitL{x}{N}` returns the first piece after the action of `\xintDecSplit`.

### 14.45 `\xintDecSplitR`

`\xintDecSplitR{x}{N}` returns the second piece after the action of `\xintDecSplit`.

## 15 Commands of the **xintgcd** package

This package was included in the original release 1.0 of the **xint** bundle.

### 15.1 `\xintGCD`

`\xintGCD{N}{M}` computes the greatest common divisor. It is positive, except when both  $N$  and  $M$  vanish, in which case the macro returns zero.

```
\xintGCD{10000}{1113}=1
\xintGCD{123456789012345}{9876543210321}=3
```

### 15.2 `\xintBezout`

`\xintBezout{N}{M}` returns five numbers  $A$ ,  $B$ ,  $U$ ,  $V$ ,  $D$  within braces.  $A$  is the first (expanded, as usual) input number,  $B$  the second,  $D$  is the GCD, and  $UA - VB = D$ .

```

\mintAssign {{\mintBezout {10000}{1113}}}\to\X
\meaning\X: macro:->{10000}{1113}{-131}{-1177}{1}.
\mintAssign {\mintBezout {10000}{1113}}\to\A\B\U\V\D
\A: 10000, \B: 1113, \U: -131, \V: -1177, \D: 1.
\mintAssign {\mintBezout {123456789012345}{9876543210321}}\to\A\B\U\V\D
\A: 123456789012345, \B: 9876543210321, \U: 256654313730, \V: 3208178892607,
\D: 3.

```

### 15.3 \mintEuclideanAlgorithm

`\mintEuclideanAlgorithm{N}{M}` applies the Euclidean algorithm and keeps a copy of all quotients and remainders.

```

\mintAssign {{\mintEuclideanAlgorithm {10000}{1113}}}\to\X
\meaning\X: macro:->{5}{10000}{1}{1113}{8}{1096}{1}{17}{64}{8}{2}{1}
{8}{0}. The first token is the number of steps, the second is N, the third is the GCD, the
fourth is M then the first quotient and remainder, the second quotient and remainder, ... until
the final quotient and last (zero) remainder.

```

### 15.4 \mintBezoutAlgorithm

`\mintBezoutAlgorithm{N}{M}` applies the Euclidean algorithm and keeps a copy of all quotients and remainders. Furthermore it computes the entries of the successive products of the 2 by 2 matrices  $\begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix}$  formed from the quotients arising in the algorithm.

```

\mintAssign {{\mintEuclideanAlgorithm {10000}{1113}}}\to\X
\meaning\X: macro:->{5}{10000}{0}{1}{1}{1113}{1}{0}{8}{1096}{8}{1}{1}
{17}{9}{1}{64}{8}{584}{65}{2}{1}{1177}{131}{8}{0}{10000}{1113}.

```

The first token is the number of steps, the second is N, then 0, 1, the GCD, M, 1, 0, the first quotient, the first remainder, the top left entry of the first matrix, the bottom left entry, and then these four things at each step until the end.

### 15.5 \mintTypesetEuclideanAlgorithm

This macro is just an example of how to organize the data returned by `\mintEuclideanAlgorithm`. Copy the source code to a new macro and modify it to what is needed.

```

\mintTypesetEuclideanAlgorithm {123456789012345}{9876543210321}
123456789012345 = 12 × 9876543210321 + 4938270488493
9876543210321 = 2 × 4938270488493 + 2233335
4938270488493 = 2211164 × 2233335 + 536553
2233335 = 4 × 536553 + 87123
536553 = 6 × 87123 + 13815
87123 = 6 × 13815 + 4233
13815 = 3 × 4233 + 1116
4233 = 3 × 1116 + 885
1116 = 1 × 885 + 231
885 = 3 × 231 + 192
231 = 1 × 192 + 39
192 = 4 × 39 + 36

```

$$39 = 1 \times 36 + 3$$

$$36 = 12 \times 3 + 0$$

### 15.6 **\xintTypesetBezoutAlgorithm**

This macro is just an example of how to organize the data returned by **\xintBezoutAlgorithm**. Copy the source code to a new macro and modify it to what is needed.

```
\xintTypesetBezoutAlgorithm {10000}{1113}
10000 = 8 \times 1113 + 1096
  8 = 8 \times 1 + 0
  1 = 8 \times 0 + 1
1113 = 1 \times 1096 + 17
  9 = 1 \times 8 + 1
  1 = 1 \times 1 + 0
1096 = 64 \times 17 + 8
  584 = 64 \times 9 + 8
  65 = 64 \times 1 + 1
  17 = 2 \times 8 + 1
1177 = 2 \times 584 + 9
  131 = 2 \times 65 + 1
  8 = 8 \times 1 + 0
10000 = 8 \times 1177 + 584
  1113 = 8 \times 131 + 65
131 \times 10000 - 1177 \times 1113 = -1
```

## 16 Commands of the **xintfrac** package

This package was first included in release 1.03 of the **xint** bundle. The general rule of the bundle that each macro first expands (what comes first, fully) each one of its arguments applies. As in the previous documentation, **x** stands for something which will be internally embedded in a **\numexpr**, thus completely expanded and then must deliver a number obeying the TeX bounds. It may be a count register or something like `4*\count 255 + 17`, etc...

**f** stands for a fraction (or a possibly ‘long’ integer), or something which expands to a fraction or a possibly long integer. See the earlier section on fraction formats.

### 16.1 **\xintLen**

The original macro is extended to accept a fraction on input.

```
\xintLen {201710/298219}=11, \xintLen {1234/1}=4, \xintLen {1234}=4
```

### 16.2 **\xintRaw**

New with release 1.04.

This macro ‘prints’ the fraction **f** (after its parsing and expansion) in A/B form, with A as returned by **\xintNumerator{f}** and B as returned by **\xintDenominator{f}**.



```
\xintRaw{\the\numexpr 571*987\relax.123/\the\numexpr -201+59\relax}=-
-563577123/142000
```

### 16.3 `\xintNumerator`

This returns the numerator corresponding to the internal representation of a fraction, with positive powers of ten converted into zeros of this numerator:

```
\xintNumerator {178000/25600000[17]}=178000000000000000000000
\xintNumerator {312.289001/20198.27}=312289001
\xintNumerator {178.000/25600000}=178000
```

As shown by the examples, no simplification of the input is done. For a result uniquely associated to the value of the fraction first apply `\xintIrr`.

### 16.4 `\xintDenominator`

This returns the denominator corresponding to the internal representation of the fraction:<sup>17</sup>

```
\xintDenominator {178000/25600000[17]}=25600000
\xintDenominator {312.289001/20198.27}=20198270000
\xintDenominator {178.000/25600000}=25600000000
```

As shown by the examples, no simplification of the input is done. The denominator looks wrong in the last example, but the numerator was tacitly multiplied by 1000 through the removal of the decimal point. For a result uniquely associated to the value of the fraction first apply `\xintIrr`.

### 16.5 `\xintFrac`

This is a  $\text{\LaTeX}$  only command, to be used in math mode only. It will print a fraction, internally represented as something equivalent to  $A/B[n]$  as `\frac {A}{B}10^n`. The power of ten is omitted when  $n=0$ , the denominator is omitted when it has value one, the number being separated from the power of ten by a `\cdot`. `\xintFrac {178.000/25600000}` gives  $\frac{178000}{25600000}10^{-3}$ , `\xintFrac {178.000/1}` gives  $178000 \cdot 10^{-3}$ , `\xintFrac {3.5/5.7}` gives  $\frac{35}{57}$ , and `\xintFrac {\xintIrr {\xintFac{10}}/\xintiSqr{\xintFac {5}}}` gives 252. As shown by the examples, simplification of the input (apart from removing the decimal points and moving the minus sign to the numerator) is not done automatically and must be the result of macros such as `\xintIrr` or `\xintREZ`.

### 16.6 `\xintSignedFrac`

New with release 1.04.

This is as `\xintFrac` except that a negative fraction has the sign put in front, not in the numerator.

```
\[\xintFrac {-355/113}=\xintSignedFrac {-355/113}\]
```

$$\frac{-355}{113} = -\frac{355}{113}$$

<sup>17</sup>recall that the `[]` construct excludes presence of a decimal point.

### 16.7 `\xintFwOver`

This does the same as `\xintFrac` except that the `\over` primitive is used for the fraction (in case the denominator is not one; and a pair of braces contains the A`\over` B part).  
`\xintFwOver {178.000/256000000}` gives  $\frac{178000}{256000000} 10^{-3}$ , `\xintFwOver {178.000/1}` gives  $178000 \cdot 10^{-3}$ , `\xintFwOver {3.5/5.7}` gives  $\frac{35}{57}$ , and `\xintFwOver {\xintIrr {\xintFac{10}}/\xintiSqr{\xintFac {5}}}` gives 252.

### 16.8 `\xintSignedFwOver`

New with release 1.04.

This is as `\xintFwOver` except that a negative fraction has the sign put in front, not in the numerator.

`\[\xintFwOver {-355/113}=\xintSignedFwOver {-355/113}\]`

$$\frac{-355}{113} = -\frac{355}{113}$$

### 16.9 `\xintREZ`

This command normalizes a fraction by removing the powers of ten in its numerator and denominator: `\xintREZ {178000/25600000[17]}=178/256[15]`. As shown by the example, it does not otherwise simplify the fraction.

### 16.10 `\xintIrr`

This puts the fraction into its unique irreducible form:

$$\xintIrr {178.256/256.178}=6856/9853 = \frac{6856}{9853}$$

Note that the current implementation does not cleverly first factor powers of 2 and 5, so input such as `\xintIrr {2/3[100]}` will make **xintfrac** do the Euclidean division of  $2 \cdot 10^{100}$  by 3, which is a bit stupid.

### 16.11 `\xintJrr`

This also puts the fraction into its unique irreducible form:

$$\xintJrr {178.256/256.178}=6856/9853$$

This is faster than `\xintIrr` for fractions having some big common factor in the numerator and the denominator.

$$\xintJrr {\xintiPow{\xintFac {15}}{3}/\xintiProductExpr {\xintFac{10}}{\xintFac{30}}{\xintFac{5}}\relax }=1001/51705840$$

But to notice the difference one would need computations with much bigger numbers than in this example.

### 16.12 `\xintTrunc`

`\xintTrunc{x}{f}` returns the start of the decimal expansion of the fraction `f`, with `x` digits after the decimal point. The argument `x` should be non-negative. When `x=0`, the integer part of `f` results, with an ending decimal point. Only when `f` evaluates to zero does `\xintTrunc` not print a decimal point. When `f` is not zero, the sign is maintained in the

output, also when the digits are all zero.

```
\xintTrunc {16}{-803.2028/20905.298}=-0.0384210165289200
\xintTrunc {20}{-803.2028/20905.298}=-0.03842101652892008523
  \xintTrunc {10}{\xintPow {-11}{-11}}=-0.0000000000
  \xintTrunc {12}{\xintPow {-11}{-11}}=-0.0000000000003
    \xintTrunc {12}{\xintAdd {-1/3}{3/9}}=0
```

The digits printed are exact up to and including the last one. The identity  $\backslash\xintTrunc{x}{f} = -\backslash\xintTrunc{x}{-f}$  holds.<sup>18</sup>

### 16.13 $\backslash\xintiTrunc$

$\backslash\xintiTrunc{x}{f}$  returns the integer equal to  $10^x$  times what  $\backslash\xintTrunc{x}{f}$  would return.

```
\xintiTrunc {16}{-803.2028/20905.298}=-384210165289200
  \xintiTrunc {10}{\xintPow {-11}{-11}}=0
  \xintiTrunc {12}{\xintPow {-11}{-11}}=-3
```

Differences between  $\backslash\xintTrunc{0}{f}$  and  $\backslash\xintiTrunc{0}{f}$ : the former cannot be used inside integer-only macros, and the latter removes the decimal point, and never returns  $-0$  (and of course removes all superfluous leading zeros.)

### 16.14 $\backslash\xintRound$

New with release 1.04.

$\backslash\xintRound{x}{f}$  returns the start of the decimal expansion of the fraction  $f$ , rounded to  $x$  digits precision after the decimal point. The argument  $x$  should be non-negative. Only when  $f$  evaluates exactly to zero does  $\backslash\xintRound$  return  $0$  without decimal point. When  $f$  is not zero, its sign is given in the output, also when the digits printed are all zero.

```
\xintRound {16}{-803.2028/20905.298}=-0.0384210165289201
\xintRound {20}{-803.2028/20905.298}=-0.03842101652892008523
  \xintRound {10}{\xintPow {-11}{-11}}=-0.0000000000
  \xintRound {12}{\xintPow {-11}{-11}}=-0.0000000000004
    \xintRound {12}{\xintAdd {-1/3}{3/9}}=0
```

The identity  $\backslash\xintRound{x}{-f} = -\backslash\xintRound{x}{f}$  holds. And regarding  $(-11)^{-11}$  here is some more or its expansion:

```
-0.000000000000350493899481392497604003313162598556370...
```

### 16.15 $\backslash\xintiRound$

New with release 1.04.

$\backslash\xintiRound{x}{f}$  returns the integer equal to  $10^x$  times what  $\backslash\xintRound{x}{f}$  would return.

```
\xintiRound {16}{-803.2028/20905.298}=-384210165289201
  \xintiRound {10}{\xintPow {-11}{-11}}=0
```

Differences between  $\backslash\xintRound{0}{f}$  and  $\backslash\xintiRound{0}{f}$ : the former cannot be

<sup>18</sup>this is just a notation; currently  $-\backslash\text{macro}$  is not valid input to any package macro, one must use  $\backslashxintOpp{\backslash\text{macro}}$  or  $\backslashxintiOpp{\backslash\text{macro}}$ .

used inside integer-only macros, and the latter removes the decimal point, and never returns  $-0$  (and of course removes all superfluous leading zeros.)

### 16.16 `\xintAdd`

The original macro is extended to accept fractions on input. Its output will now always be in the form  $A/B[n]$  or  $A[n]$ . The original is available as `\xintiAdd`.

### 16.17 `\xintSub`

The original macro is extended to accept fractions on input. Its output will now always be in the form  $A/B[n]$  or  $A[n]$ . The original is available as `\xintiSub`.

### 16.18 `\xintMul`

The original macro is extended to accept fractions on input. Its output will now always be in the form  $A/B[n]$  or  $A[n]$ . The original is available as `\xintiMul`.

### 16.19 `\xintSqr`

The original macro is extended to accept a fraction on input. Its output will now always be in the form  $A/B[n]$  or  $A[n]$ . The original is available as `\xintiSqr`.

### 16.20 `\xintPow`

The original macro is extended to accept a fraction on input (the exponent must be a signed integer of course). Its output will now always be in the form  $A/B[n]$  or  $A[n]$ . The original is available as `\xintiPow`.

### 16.21 `\xintSum`, `\xintSumExpr`

The original commands are extended to accept fractions on input and produce fractions on output. Their outputs will now always be in the form  $A/B[n]$  or  $A[n]$ . The originals are available as `\xintiSum` and `\xintiSumExpr`.

### 16.22 `\xintPrd`, `\xintProductExpr`

The originals are extended to accept fractions on input and produce fractions on output. Their outputs will now always be in the form  $A/B[n]$  or  $A[n]$ . The originals are available as `\xintiPrd` and `\xintiPrdExpr`.

### 16.23 `\xintDiv`

`\xintDiv{f}{g}` computes the fraction  $f/g$ . As with all other computation macros, no simplification is done on the output, which is in the form  $A/B[n]$  or  $A[n]$ .

**16.24 `\xintCmp`**

The macro is extended to fractions. Of course its output is still either  $-1$ ,  $0$ , or  $1$  with no forward slash nor trailing  $[n]$ . The original, which skips the overhead of the fraction format parsing, is available as `\xintiCmp`.

**16.25 `\xintMax`**

The macro is extended to fractions. But now `\xintMax {2}{3}` returns  $3/1[0]$ . The original is available as `\xintiMax`.

**16.26 `\xintMin`**

The macro is extended to fractions. The original is available as `\xintiMin`.

**16.27 `\xintAbs`**

The macro is extended to fractions. The original is available as `\xintiAbs`. Note that `\xintAbs {-2}=2/1[0]` whereas `\xintiAbs {-2}=2`.

**16.28 `\xintSgn`**

The macro is extended to fractions. Of course its output is still either  $-1$ ,  $0$ , or  $1$  with no forward slash nor trailing  $[n]$ . The original, which skips the overhead of the fraction format parsing, is available as `\xintiSgn`.

**16.29 `\xintOpp`**

The macro is extended to fractions. The original is available as `\xintiOpp`. Note that `\xintOpp {3}` now outputs  $-3/1[0]$ .

**16.30 `\xintGeq`, `\xintDivision`, `\xintQuo`, `\xintRem`, `\xintFDg`,  
`\xintLDg`, `\xintMON`, `\xintMMON`**

These macros are extended to accept a fraction on input if this fraction in fact reduces to an integer (if not an `\xintError:NotAnInteger` will be raised). As usual, the ‘i’ variants all exist, they accept on input only integers in the strict format and have less overhead. There is no difference in the output, the difference is only in the accepted format for the inputs.

**16.31 `\xintNum`**

The macro is extended to accept a fraction on input. But this fraction should reduce to an integer. If not an error will be raised. The original is available as `\xintiNum`.

## 17 Commands of the **xintseries** package

Some arguments to the package commands are macros which are expanded only later, when given their parameters. The arguments serving as indices (new with 1.06) are systematically given to a `\numexpr` expressions, hence fully expanded, they may be count registers, etc...

This package was first released with version 1.03 of the **xint** bundle.

### 17.1 `\xintSeries`

`\xintSeries{A}{B}{\coeff}` evaluates the sum of all values of the `\coeff {n}` from  $n=A$  to and including  $n=B$ . The initial and final indices must obey the `\numexpr` constraint of expanding to numbers at most  $2^{31}-1$ . The `\coeff` macro (which, as argument to `\xintSeries` is expanded only at the time of computing the successive `\coeff {n}`) should be defined as a one-parameter fully expandable command, providing its output from an input being an explicit number (string of digits, no need to make proviso for a count register).

```
\def\coeff #1{\xintiMON{#1}/#1.5} % (-1)^n/(n+1/2)
\edef\w {\xintSeries {0}{50}{\coeff}} % we want to re-use it
\edef\z {\xintJrr {\w}[0]} % the [0] for a microsecond gain.
% \xintJrr preferred to \xintIrr: a big common factor is suspected.
% But numbers much bigger would be needed to show the greater efficiency.
\[ \sum_{n=0}^{50} (-1)^n \frac{1}{n + \frac{1}{2}} = \frac{173909338287370940432112792101626602278714}{110027467159390003025279917226039729050575}
```

For info, before action by `\xintJrr` the inner representation of the result has a denominator of `\xintLen {\xintDenominator\w}=117` digits. This troubled me as 101!! has only 81 digits: `\xintLen {\xintQuo {\xintFac {101}}{\xintiMul {\xintiPow {2}{50}}{\xintFac{50}}}}=81`. The explanation lies in the too clever to be efficient #1.5 trick. It leads to a silly extra  $5^{51}$  (which has 36 digits) in the denominator. See the explanations in the next section.

Note: as soon as the coefficients look like factorials, it is more efficient to use the `\xintRationalSeries` macro whose evaluation will avoid a denominator build-up; indeed the raw operations of addition and subtraction of fractions blindly multiply out denominators. So the raw evaluation of  $\sum_{n=0}^N 1/n!$  with `\xintSeries` will have a denominator equal to  $\prod_{n=0}^N n!$ . Needless to say this makes it more difficult to compute the exact value of this sum with  $N=50$ , for example, whereas with `\xintRationalSeries` the denominator does not get bigger than 50!.

For info: by the way  $\prod_{n=0}^{50} n!$  is easily computed by **xint** and is a number with 1394 digits. And  $\prod_{n=0}^{100} n!$  is also computable by **xint** (24 seconds on my laptop for the brute force iterated multiplication of all factorials, a specialized routine would do it faster) and has 6941 digits (this means more than two pages if printed...). Whereas 100! only has 158 digits.

```
\def\coeffleibnitz #1{\the\numexpr\ifodd #1 1\else-1\fi\relax/#1[0]}
\cnta 1
```

```

\loop % in this loop we recompute from scratch each partial sum!
% we can afford that, as \xintSeries is fast enough.
\noindent\hbox to 2em{\hfil\texttt{\the\cnta.} }%
      \xintTrunc {12}
      {\xintSeries {1}{\cnta}{\coeffleibnitz}}\dots
\endgraf
\ifnum\cnta < 30 \advance\cnta 1 \repeat

1. 1.000000000000...    11. 0.736544011544...    21. 0.716390450794...
2. 0.500000000000...    12. 0.653210678210...    22. 0.670935905339...
3. 0.833333333333...    13. 0.730133755133...    23. 0.714414166209...
4. 0.583333333333...    14. 0.658705183705...    24. 0.672747499542...
5. 0.783333333333...    15. 0.725371850371...    25. 0.712747499542...
6. 0.616666666666...    16. 0.662871850371...    26. 0.674285961081...
7. 0.759523809523...    17. 0.721695379783...    27. 0.711322998118...
8. 0.634523809523...    18. 0.666139824228...    28. 0.675608712404...
9. 0.745634920634...    19. 0.718771403175...    29. 0.710091471024...
10. 0.645634920634...   20. 0.668771403175...    30. 0.676758137691...

```

## 17.2 \xintiSeries

`\xintiSeries{A}{B}{\coeff}` evaluates the sum of `\coeff {n}` from  $n=A$  to and including  $n=B$ . The initial and final indices are given to a `\numexpr` expression. The `\coeff` macro (which, as argument to `\xintiSeries` is expanded only at the time of computing `\coeff {n}`) should be defined as a one-parameter fully expandable command, accepting on input an explicit number, and returning a (long) integer in the format understood by the integer-only `\xintiAdd`.

```

\def\coeff #1{\xintiTrunc {40}{\xintMON{#1}/#1.5}}%
% better:
\def\coeff #1{\xintiTrunc {40}
  {\the\numexpr 2*\xintMON{#1}\relax/\the\numexpr 2*#1+1\relax [0]}}%
% better still:
\def\coeff #1{\xintiTrunc {40}
  {\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}}%
% (-1)^n/(n+1/2) times 10^40, truncated to an integer.
\[ \sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac{1}{2}} \approx
  \xintiTrunc {40}{\xintiSeries {0}{50}{\coeff}[-40]}\dots\]

```

The #1.5 trick to define the `\coeff` macro was neat, but  $1/3.5$ , for example, turns internally into  $10/35$  whereas it would be more efficient to have  $2/7$ . The second way of coding the wanted coefficient avoids a superfluous factor of five and leads to a faster evaluation. The third way is faster, after all there is no need to use `\xintMON` (or rather `\xintiMON` which has less parsing overhead) on integers obeying the  $\text{T}_{\text{E}}\text{X}$  bound. The denominator having no sign, we have added the `[0]` as this speeds up (infinitesimally) the parsing.

$$\sum_{n=0}^{n=50} \frac{(-1)^n}{n + \frac{1}{2}} \approx 1.5805993064935250412367895069567264144810$$

We should have cut out at least the last two digits: truncating errors originating with the first coefficients of the sum will never go away, and each truncation introduces an uncertainty in



the last digit, so as we have 40 terms, we should trash the last two digits, or at least round at 38 digits. It is interesting to compare with the computation where rounding rather than truncation is used, and with the decimal expansion of the exactly computed partial sum of the series:

```
\def\coeff #1{\xintiRound {40} % rounding at 40
  {\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}}%
% (-1)^n/(n+1/2) times 10^40, rounded to an integer.
\[ \sum_{n=0}^{50} \frac{(-1)^n}{n+\frac{1}{2}} \approx
  \xintTrunc {40}{\xintiSeries {0}{50}{\coeff}{-40}}\]
\def\exactcoeff #1%
  {\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}}%
\[ \sum_{n=0}^{50} \frac{(-1)^n}{n+\frac{1}{2}}
  = \xintTrunc {50}{\xintSeries {0}{50}{\exactcoeff}}\dots\]
```

$$\sum_{n=0}^{50} \frac{(-1)^n}{n + \frac{1}{2}} \approx 1.5805993064935250412367895069567264144804$$

$$\sum_{n=0}^{50} \frac{(-1)^n}{n + \frac{1}{2}} = 1.58059930649352504123678950695672641448068680288367 \dots$$

This shows indeed that our sum of truncated terms estimated wrongly the 39th and 40th digits of the exact result<sup>19</sup> and that the sum of rounded terms fared a bit better.

### 17.3 `\xintRationalSeries`

New with release 1.04.

`\xintRationalSeries{A}{B}{f}{\ratio}` evaluates the sum of  $F(n)$ <sup>20</sup> from  $n=A$  up to and including  $n=B$ , with the parameter  $f$  being (or expanding to) the value  $F(A)$  and `\ratio` being a one-parameter expandable command, accepting on input an explicit number  $n$  and producing after (full iterated) expansion (of the first token)  $F(n)/F(n-1)$ . The initial and final indices are given to a `\numexpr` expression.

```
\def\ratio #1{2/#1[0]}% 2/n, comes from the series of exp(2)
\cnta 0 % previously declared count
\loop
\edef\z {\xintRationalSeries {0}{\cnta}{1}{\ratio }}%
\noindent$\sum_{n=0}^{\the\cnta} \frac{2^n}{n!}=
  \xintTrunc{12}\z\dots=
  \xintFrac\z=\xintFrac{\xintIrr\z}\vtop to 5pt{}\endgraf
\ifnum\cnta<20 \advance\cnta 1 \repeat
```

$$\begin{aligned} \sum_{n=0}^0 \frac{2^n}{n!} &= 1.000000000000 \dots = 1 = 1 \\ \sum_{n=0}^1 \frac{2^n}{n!} &= 3.000000000000 \dots = 3 = 3 \\ \sum_{n=0}^2 \frac{2^n}{n!} &= 5.000000000000 \dots = \frac{10}{2} = 5 \\ \sum_{n=0}^3 \frac{2^n}{n!} &= 6.333333333333 \dots = \frac{38}{6} = \frac{19}{3} \end{aligned}$$

<sup>19</sup>as the series is alternating, we can roughly expect an error of  $\sqrt{40}$  and the last two digits are off by 4 units, which is not contradictory to our expectations.

<sup>20</sup>the macro is designed to be useful when  $F(n)/F(n-1)$  is a rational function of  $n$  but it may be used of course with any sort of general term.

$$\begin{aligned}
\sum_{n=0}^4 \frac{2^n}{n!} &= 7.000000000000 \dots = \frac{168}{24} = 7 \\
\sum_{n=0}^5 \frac{2^n}{n!} &= 7.266666666666 \dots = \frac{872}{120} = \frac{109}{15} \\
\sum_{n=0}^6 \frac{2^n}{n!} &= 7.355555555555 \dots = \frac{5296}{720} = \frac{331}{45} \\
\sum_{n=0}^7 \frac{2^n}{n!} &= 7.380952380952 \dots = \frac{37200}{5040} = \frac{155}{21} \\
\sum_{n=0}^8 \frac{2^n}{n!} &= 7.387301587301 \dots = \frac{297856}{40320} = \frac{2327}{315} \\
\sum_{n=0}^9 \frac{2^n}{n!} &= 7.388712522045 \dots = \frac{2681216}{362880} = \frac{20947}{2835} \\
\sum_{n=0}^{10} \frac{2^n}{n!} &= 7.388994708994 \dots = \frac{26813184}{3628800} = \frac{34913}{4725} \\
\sum_{n=0}^{11} \frac{2^n}{n!} &= 7.389046015712 \dots = \frac{294947072}{39916800} = \frac{164591}{22275} \\
\sum_{n=0}^{12} \frac{2^n}{n!} &= 7.389054566832 \dots = \frac{3539368960}{479001600} = \frac{691283}{93555} \\
\sum_{n=0}^{13} \frac{2^n}{n!} &= 7.389055882389 \dots = \frac{46011804672}{6227020800} = \frac{14977801}{2027025} \\
\sum_{n=0}^{14} \frac{2^n}{n!} &= 7.389056070325 \dots = \frac{644165281792}{87178291200} = \frac{314533829}{42567525} \\
\sum_{n=0}^{15} \frac{2^n}{n!} &= 7.389056095384 \dots = \frac{9662479259648}{1307674368000} = \frac{4718007451}{638512875} \\
\sum_{n=0}^{16} \frac{2^n}{n!} &= 7.389056098516 \dots = \frac{154599668219904}{20922789888000} = \frac{1572669151}{212837625} \\
\sum_{n=0}^{17} \frac{2^n}{n!} &= 7.389056098884 \dots = \frac{2628194359869440}{355687428096000} = \frac{16041225341}{2170943775} \\
\sum_{n=0}^{18} \frac{2^n}{n!} &= 7.389056098925 \dots = \frac{47307498477912064}{6402373705728000} = \frac{103122162907}{13956067125} \\
\sum_{n=0}^{19} \frac{2^n}{n!} &= 7.389056098930 \dots = \frac{898842471080853504}{121645100408832000} = \frac{4571749222213}{618718975875} \\
\sum_{n=0}^{20} \frac{2^n}{n!} &= 7.389056098930 \dots = \frac{17976849421618118656}{2432902008176640000} = \frac{68576238333199}{9280784638125}
\end{aligned}$$

Such computations would become quickly completely inaccessible via the `\xintSeries` macros, as the factorials in the denominators would get all multiplied together: the raw addition and subtraction on fractions just blindly multiplies denominators! Whereas `\xintRationalSeries` evaluate the partial sums via a less silly iterative scheme.

```

\def\ratio #1{-1/#1[0]}% -1/n, comes from the series of exp(-1)
\cnta 0 % previously declared count
\loop
\edef\z {\xintRationalSeries {0}{\cnta}{1}{\ratio }}%
\noindent$\sum_{n=0}^{\the\cnta} \frac{(-1)^n}{n!}=
\xintTrunc{20}\z\dots=\xintFrac{\z}=\xintFrac{\xintIrr\z}$
\vtop to 5pt{}\endgraf
\ifnum\cnta<20 \advance\cnta 1 \repeat

```

$$\begin{aligned}
\sum_{n=0}^0 \frac{(-1)^n}{n!} &= 1.000000000000000000 \dots = 1 = 1 \\
\sum_{n=0}^1 \frac{(-1)^n}{n!} &= 0 \dots = 0 = 0 \\
\sum_{n=0}^2 \frac{(-1)^n}{n!} &= 0.500000000000000000 \dots = \frac{1}{2} = \frac{1}{2} \\
\sum_{n=0}^3 \frac{(-1)^n}{n!} &= 0.333333333333333333 \dots = \frac{2}{6} = \frac{1}{3} \\
\sum_{n=0}^4 \frac{(-1)^n}{n!} &= 0.375000000000000000 \dots = \frac{9}{24} = \frac{3}{8} \\
\sum_{n=0}^5 \frac{(-1)^n}{n!} &= 0.366666666666666666 \dots = \frac{44}{120} = \frac{11}{30} \\
\sum_{n=0}^6 \frac{(-1)^n}{n!} &= 0.368055555555555555 \dots = \frac{265}{720} = \frac{53}{144} \\
\sum_{n=0}^7 \frac{(-1)^n}{n!} &= 0.36785714285714285714 \dots = \frac{1854}{5040} = \frac{103}{280} \\
\sum_{n=0}^8 \frac{(-1)^n}{n!} &= 0.367881944444444444 \dots = \frac{14833}{40320} = \frac{2119}{5760} \\
\sum_{n=0}^9 \frac{(-1)^n}{n!} &= 0.36787918871252204585 \dots = \frac{133496}{362880} = \frac{16687}{45360} \\
\sum_{n=0}^{10} \frac{(-1)^n}{n!} &= 0.36787946428571428571 \dots = \frac{1334961}{3628800} = \frac{16481}{44800} \\
\sum_{n=0}^{11} \frac{(-1)^n}{n!} &= 0.36787943923360590027 \dots = \frac{14684570}{39916800} = \frac{1468457}{3991680}
\end{aligned}$$

$$\begin{aligned} \sum_{n=0}^{12} \frac{(-1)^n}{n!} &= 0.36787944132128159905 \dots = \frac{176214841}{479001600} = \frac{16019531}{43545600} \\ \sum_{n=0}^{13} \frac{(-1)^n}{n!} &= 0.36787944116069116069 \dots = \frac{2290792932}{6227020800} = \frac{63633137}{172972800} \\ \sum_{n=0}^{14} \frac{(-1)^n}{n!} &= 0.36787944117216190628 \dots = \frac{32071101049}{87178291200} = \frac{2467007773}{6706022400} \\ \sum_{n=0}^{15} \frac{(-1)^n}{n!} &= 0.36787944117139718991 \dots = \frac{481066515734}{1307674368000} = \frac{34361893981}{93405312000} \\ \sum_{n=0}^{16} \frac{(-1)^n}{n!} &= 0.36787944117144498468 \dots = \frac{7697064251745}{20922789888000} = \frac{15549624751}{42268262400} \\ \sum_{n=0}^{17} \frac{(-1)^n}{n!} &= 0.36787944117144217323 \dots = \frac{130850092279664}{355687428096000} = \frac{8178130767479}{22230462456000} \\ \sum_{n=0}^{18} \frac{(-1)^n}{n!} &= 0.36787944117144232942 \dots = \frac{2355301661033953}{6402373705728000} = \frac{138547156531409}{376610217984000} \\ \sum_{n=0}^{19} \frac{(-1)^n}{n!} &= 0.36787944117144232120 \dots = \frac{44750731559645106}{121645100408832000} = \frac{92079694567171}{250298560512000} \\ \sum_{n=0}^{20} \frac{(-1)^n}{n!} &= 0.36787944117144232161 \dots = \frac{895014631192902121}{2432902008176640000} = \frac{4282366656425369}{116406794649600000} \end{aligned}$$

We can incorporate an indeterminate if we define `\ratio` to be a macro with two parameters: `\def\ratioexp #1#2{\xintDiv{#1}{#2}}%`  $x/n$ :  $x=\#1$ ,  $n=\#2$ . Then, if `\x` expands to some fraction  $x$ , the command

```

\cmta 0
\def\ratioexp #1#2{\xintDiv{#1}{#2}}% #1/#2
\loop
\noindent

$$\sum_{n=0}^{\cmta} (.57)^n/n! = \xintTrunc{50}$$


$$\xintRationalSeries{0}{\cmta}{1}{\ratioexp{.57}}{\dots}$$

\vtop to 5pt {\endgraf
\ifnum\cmta<50 \advance\cmta 10 \repeat

```

$$\begin{aligned}\sum_{n=0}^0 (.57)^n/n! &= 1.000 \dots \\ \sum_{n=0}^{10} (.57)^n/n! &= 1.76826705137947002480668058035714285714285714285714 \dots \\ \sum_{n=0}^{20} (.57)^n/n! &= 1.76826705143373515162089324271187082272833005529082 \dots \\ \sum_{n=0}^{30} (.57)^n/n! &= 1.76826705143373515162089339282382144915484884979430 \dots \\ \sum_{n=0}^{40} (.57)^n/n! &= 1.76826705143373515162089339282382144915485219867776 \dots \\ \sum_{n=0}^{50} (.57)^n/n! &= 1.76826705143373515162089339282382144915485219867776 \dots\end{aligned}$$

Observe that in this last example the `x` was directly inserted; if it had been a more complicated explicit fraction it would have been worthwhile to use `\ratioexp\x` with `\x` defined to expand to its value. In the further situation where this fraction `x` is not explicit but itself defined via a complicated, and time-costly, formula, it should be noted that `\xintRationalSeries` will do again the evaluation of `\x` for each term of the partial sum. The easiest is thus when `x` can be defined as an `\edef`. If however, you are in an expandable-only context and cannot store in a macro like `\x` the value to be used, a variant of `\xintRationalSeries` is needed which will first evaluate this `\x` and then use this result without recomputing it. This is `\xintRationalSeriesX`, documented next.

Here is a slightly more complicated evaluation:

```

\cnta 1
\loop \edef\z {\xintRationalSeries
      {\cnta}
      {2*\cnta-1}
      {\xintiPow {\the\cnta}{\cnta}/\xintFac{\cnta}}}

```

```

{\ratioexp{\the\cnta}}}%
\edef\w {\xintRationalSeries {0}{2*\cnta-1}{1}{\ratioexp{\the\cnta}}}%
\noindent
$\sum_{n=\the\cnta}^{\the\numexpr 2*\cnta-1\relax} \frac{\the\cnta^n}{n!}/%
\sum_{n=0}^{\the\numexpr 2*\cnta-1\relax} \frac{\the\cnta^n}{n!} =
\xintTrunc{8}{\xintDiv\z\w}\dots$ \vtop to 5pt{}\endgraf
\ifnum\cnta<20 \advance\cnta 1 \repeat

```

$$\begin{array}{ll}
\sum_{n=1}^1 \frac{1^n}{n!} / \sum_{n=0}^1 \frac{1^n}{n!} = 0.50000000\dots & \sum_{n=11}^{21} \frac{11^n}{n!} / \sum_{n=0}^{21} \frac{11^n}{n!} = 0.53907332\dots \\
\sum_{n=2}^3 \frac{2^n}{n!} / \sum_{n=0}^3 \frac{2^n}{n!} = 0.52631578\dots & \sum_{n=12}^{23} \frac{12^n}{n!} / \sum_{n=0}^{23} \frac{12^n}{n!} = 0.53772178\dots \\
\sum_{n=3}^5 \frac{3^n}{n!} / \sum_{n=0}^5 \frac{3^n}{n!} = 0.53804347\dots & \sum_{n=13}^{25} \frac{13^n}{n!} / \sum_{n=0}^{25} \frac{13^n}{n!} = 0.53644744\dots \\
\sum_{n=4}^7 \frac{4^n}{n!} / \sum_{n=0}^7 \frac{4^n}{n!} = 0.54317053\dots & \sum_{n=14}^{27} \frac{14^n}{n!} / \sum_{n=0}^{27} \frac{14^n}{n!} = 0.53525726\dots \\
\sum_{n=5}^9 \frac{5^n}{n!} / \sum_{n=0}^9 \frac{5^n}{n!} = 0.54502576\dots & \sum_{n=15}^{29} \frac{15^n}{n!} / \sum_{n=0}^{29} \frac{15^n}{n!} = 0.53415135\dots \\
\sum_{n=6}^{11} \frac{6^n}{n!} / \sum_{n=0}^{11} \frac{6^n}{n!} = 0.54518217\dots & \sum_{n=16}^{31} \frac{16^n}{n!} / \sum_{n=0}^{31} \frac{16^n}{n!} = 0.53312615\dots \\
\sum_{n=7}^{13} \frac{7^n}{n!} / \sum_{n=0}^{13} \frac{7^n}{n!} = 0.54445274\dots & \sum_{n=17}^{33} \frac{17^n}{n!} / \sum_{n=0}^{33} \frac{17^n}{n!} = 0.53217628\dots \\
\sum_{n=8}^{15} \frac{8^n}{n!} / \sum_{n=0}^{15} \frac{8^n}{n!} = 0.54327992\dots & \sum_{n=18}^{35} \frac{18^n}{n!} / \sum_{n=0}^{35} \frac{18^n}{n!} = 0.53129566\dots \\
\sum_{n=9}^{17} \frac{9^n}{n!} / \sum_{n=0}^{17} \frac{9^n}{n!} = 0.54191055\dots & \sum_{n=19}^{37} \frac{19^n}{n!} / \sum_{n=0}^{37} \frac{19^n}{n!} = 0.53047810\dots \\
\sum_{n=10}^{19} \frac{10^n}{n!} / \sum_{n=0}^{19} \frac{10^n}{n!} = 0.54048295\dots & \sum_{n=20}^{39} \frac{20^n}{n!} / \sum_{n=0}^{39} \frac{20^n}{n!} = 0.52971771\dots
\end{array}$$

## 17.4 \xintRationalSeriesX

New with release 1.04.

`\xintRationalSeriesX{A}{B}{\first}{\ratio}{x}` evaluates the sum of  $F(n, x)$  from  $n=A$  up to and including  $n=B$ , where  $x$  expands to a fraction  $x$ , `\first` is a one-parameter macro such that `\first{x}` expands in two steps at most to the first term  $F(A, x)$  of the series, and `\ratio` is a two parameter macro such that `\ratio{x}{n}` expands to the ratio  $F(n, x)/F(n-1, x)$ . Hence, this is a parametrized version of `\xintRationalSeries`, where the parameter  $x$  is evaluated only once at the beginning of the computation, and can thus itself be the yet unevaluated result of a previous computation.

Note the subtle differences between

```

\xintRationalSeries {a}{b}{\first}{\ratio{x}}
\xintRationalSeriesX {a}{b}{\first}{\ratio}{x}

```

First the location of braces differ... then, in the first one `\first` is a macro expanding to a fractional number, but in the X one, it is a one-parameter macro which will use  $x$ . The `\ratio` macro is in both cases a two-parameters macro, the difference is that in the X variant the  $x$  will be evaluated at the very beginning whereas the former variant replaces it by its evaluation each time it needs it (which is bad if this evaluation is time-costly, but good if it just a big explicit fraction encapsulated in a macro).

The example will use the macro `\xintPowerSeries` which computes efficiently exact partial sums of power series, and is discussed in the next section.

```

\def\firstterm #1{1[0]}% first term of the exponential series
% although it is the constant 1, here it must be defined as a
% one-parameter macro. Next comes the ratio function for exp:
\def\ratioexp #1#2{\xintDiv {#1}{#2}}% x/n
% These are the  $(-1)^{n-1}/n$  of the  $\log(1+h)$  series:
\def\coefflog #1{\the\numexpr\ifodd #1 1\else-1\fi\relax/#1[0]}%

```

```
% Let L(h) be the first 10 terms of the log(1+h) series and
% let E(t) be the first 10 terms of the exp(t) series.
% The following computes E(L(a/10)) for a=1,...,12.
\cnta 0
\loop
\noindent\xintTrunc {18}{%
    \xintRationalSeriesX {0}{9}{\firstterm}{\ratioexp}
        {\xintPowerSeries{1}{10}{\coefflog}{\the\cnta[-1]}}}\dots
\endgraf
\ifnum\cnta < 12 \advance \cnta 1 \repeat
```

1.0999999999999999083906...	1.499954310225476533...	1.870485649686617459...
1.1999999998111624029...	1.599659266069210466...	1.907197560339468199...
1.299999835744121464...	1.698137473697423757...	1.845117565491393752...
1.399996091955359088...	1.791898112718884531...	1.593831932293536053...

These completely exact operations rapidly create numbers with many digits. Let us print in full the raw fractions created by the operation illustrated above:

E(L(1[-1]))=435534952734304993753128478305695755446525998418916420656  
308534427154141471013807206588202981046013155342233701289165089056830056  
93656447898877952000000000/395940866122425193243875570782668457763038822  
4000000000000000000[-90] (length of numerator: 155)

E(L(12[-2]))=44345377005441746544210925234726482471189359916041172960  
38825841980841532261080707050589009628030597103713328020346412371558877  
141883806589829590141346329464027599993974220093034636265326435417048639  
843167445553122713679545984140443648000000000/39594086612242519324387557  
0782668457763038822400000000000000000000[-180] (length of numerator: 245)

E(L(123[-3]))=4446415926519417771542541488488548661989549715526163900  
742959135317921138508647797623508008144169817627741486630524932175667597  
540979774207315163733367897227307654961390791852295451022482823911996210  
292377938117401221109197354331611327571689558640177108818505853950798598  
4383161796620719539156780347183214743630293655563010048000000000/3959408  
66122425193243875570782668457763038822400000000000000000000[-270] (length of  
numerator: 335)

We see that the denominators here remain the same, as our input only had various powers of ten as denominators, and `xintfrac` efficiently assemble (some only, as we can see) powers of ten. Notice that 1 more digit in an input denominator seems to mean 90 more in the raw output. We can check that with some other test cases:

E(L(1/7))=51813851611732260491607483316483334488384059013300616812512  
534667430913353255394804713669158571590044976892591448945234186435192422  
4000000000/4533712016210897917880966278213776528922326538175815254665483  
609508708960102268994279646534211540778635880926390420871577600000000000  
00000000[0] (length of numerator: 141; length of denominator: 141)

E(L(1/71))=1647994891772195564980259558061070982561581017562093698646  
571522821497800830677980391753251868507166092934678546038421637547169191  
232746243941321882088953100899820016273515249100005882385965653808879162  
861533474038814343168000000000/16251060738309150710228315926583043448560  
635097998286551792304600401711584442548604911127392639471285026166742651

015948354491747514663603304596379819982611548681495538153647264137927630  
8916890414267771321449447424000000000000000000[0] (length of numerator: 232;  
length of denominator: 232)

$E(L(1/712)) = 209623173880163120675481637897216200283968902248203238943$   
136902264182865559717266406341976325767001357109452980607391271438079195  
073959301528254006087908156888129567520269011715459969154688799089625738  
271433856535377918700884980798641197021855117078629780316835353043067415  
7534972120128999850190174947982205517824000000000/2093291722337673799732  
719862311619975662927884547744846526034295741465968177583093786412050480  
958301357075221213896546903011983961080605724903426024563430558292203346  
913309844190901402018394162270065876670575550330002721292096217682473000  
829618103432600036119035084894266166648343032219206471638591733760000000  
000000000000[0] (length of numerator: 322; length of denominator: 322)

For info the last fraction put into irreducible form still has 288 digits in its denominator.<sup>21</sup>  
The first conclusion is that decimal numbers such as  $0.123$  (equivalently  $123[-3]$ ) give  
less computing intensive tasks than fractions such as  $1/712$ : in the case of decimal numbers  
the (raw) denominators originate in the coefficients of the series themselves, powers of  
ten of the input within brackets being treated separately. The second conclusion is that  
even then the numerators will grow with the size of the input in a sort of linear way, the  
coefficient being given by the order of series: here 10 from the log and 9 from the exp,  
so 90. One more digit in the input means 90 more digits in the numerator of the output:  
obviously we can not go on composing such partial sums of series and hope that **xint**  
will joyfully do all at the speed of light! Briefly said, imagine that the rules of the game  
make the programmer like a security guard at an airport scanning machine: a never-ending  
flux of passengers keep on arriving and all you can do is re-shuffle the first nine of them,  
organize marriages among some, execute some, move children farther back among the first  
nine only. If a passenger comes along with many hand luggages, this will slow down the  
process even if you move him to ninth position, because sooner or later you will have to  
digest him, and the children will be big too. There is no way to move some guy out of  
the file and to a discrete interrogatory room for separate treatment or to give him/her some  
badge saying “I left my stuff in storage box 357”.

Hence, truncating the output (or better, rounding) is the only way to go if one needs a  
general calculus of special functions. Floating point representation of numbers is currently  
unimplemented in **xint**. But fixed point computations are available via the commands  
`\xintTrunc` and `\xintRound`.

## 17.5 `\xintPowerSeries`

`\xintPowerSeries{A}{B}{\coeff}{x}` evaluates the sum of  $\text{\coeff}\{n\} \cdot x^n$  from  $n=A$   
up to and including  $n=B$ . The initial and final indices are given to a `\numexpr` expression.  
The `\coeff` macro (which, as argument to `\xintPowerSeries` is expanded only at the  
time `\coeff{n}` is needed) should be defined as a one-parameter expandable (in the now  
usual meaning) command, accepting on input an explicit number.

The  $x$  can be either a fraction directly input or a macro expanding to such a fraction. It

<sup>21</sup>putting this fraction in irreducible form takes more time than is typical of the other computations in this  
document; so exceptionally I have hard-coded the 288 in the document source.



is actually more efficient to encapsulate an explicit fraction  $x$  in such a macro (say  $\backslash x$ ), if it has big numerators and denominators ('big' means hundreds of digits) as it will then take less space in the processing until being (repeatedly) used.

This macro computes the *exact* result (one can use it also for polynomial evaluation). With release 1.04 the Horner scheme for polynomial evaluation is used, this avoids a denominator build-up which was plaguing the 1.03 version.<sup>22</sup>

Note: as soon as the coefficients look like factorials, it is more efficient to use the `\xintRationalSeries` macro whose evaluation, also based on a similar Horner scheme, will avoid a denominator build-up originating in the coefficients themselves.

```
\def\geom #1{1[0]} % the geometric series
\def\x {5/17[0]}
\[ \sum_{n=0}^{\infty} \Bigl(\frac{5}{17}\Bigr)^n
=\xintFrac{\xintIrr{\xintPowerSeries {0}{20}{\geom}{\x}}}
=\xintFrac{\xintiSub{\xintiPow {17}{21}}{\xintiPow{5}{21}}%
/\xintiMul{12}{\xintiPow {17}{20}}}\]
% a parser for arbitrary algebraic expressions with the +,-,/,*,and ^
% operations would be dearly appreciated here ; implementing a completely
% expandable one would be quite a lot of work, even if we plagiarize l3fp!
```

$$\sum_{n=0}^{20} \left(\frac{5}{17}\right)^n = \frac{5757661159377657976885341}{4064231406647572522401601} = \frac{69091933912531895722624092}{48770776879770870268819212}$$

```
\def\coefflog #1{1/#1[0]}% 1/n
\def\x {1/2[0]}%
\[ \log 2 \approx \sum_{n=1}^{\infty} \frac{1}{n \cdot 2^n}
=\xintFrac {\xintIrr {\xintPowerSeries {1}{20}{\coefflog}{\x}}}\]
\[ \log 2 \approx \sum_{n=1}^{\infty} \frac{1}{n \cdot 2^n}
=\xintFrac {\xintIrr {\xintPowerSeries {1}{50}{\coefflog}{\x}}}\]
```

$$\log 2 \approx \sum_{n=1}^{20} \frac{1}{n \cdot 2^n} = \frac{42299423848079}{61025172848640}$$

$$\log 2 \approx \sum_{n=1}^{50} \frac{1}{n \cdot 2^n} = \frac{60463469751752265663579884559739219}{87230347965792839223946208178339840}$$

```
\cnta 1 % previously declared count
\loop % in this loop we recompute from scratch each partial sum!
% we can afford that, as \xintPowerSeries is fast enough.
\noindent\hbox to 2em{\hfil\texttt{\the\cnta.} }%
\xintTrunc {12}
{\xintPowerSeries {1}{\cnta}{\coefflog}{\x}}\dots
\endgraf
\ifnum \cnta < 30 \advance\cnta 1 \repeat
```

<sup>22</sup>with powers  $x^k$ , from  $k=0$  to  $N$ , a denominator  $d$  of  $x$  became  $d^{1+2+\dots+N}$ , which is bad. With the 1.04 method, the part of the denominator originating from  $x$  does not accumulate to more than  $d^N$ .



1. 0.500000000000...	11. 0.693109245355...	21. 0.693147159757...
2. 0.625000000000...	12. 0.693129590407...	22. 0.693147170594...
3. 0.666666666666...	13. 0.693138980431...	23. 0.693147175777...
4. 0.682291666666...	14. 0.693143340085...	24. 0.693147178261...
5. 0.688541666666...	15. 0.693145374590...	25. 0.693147179453...
6. 0.691145833333...	16. 0.693146328265...	26. 0.693147180026...
7. 0.692261904761...	17. 0.693146777052...	27. 0.693147180302...
8. 0.692750186011...	18. 0.693146988980...	28. 0.693147180435...
9. 0.692967199900...	19. 0.693147089367...	29. 0.693147180499...
10. 0.693064856150...	20. 0.693147137051...	30. 0.693147180530...

```
%\def\coeffarctg #1{1/\the\numexpr\xintMON{#1}*(2*#1+1)\relax}%
\def\coeffarctg #1{1/\the\numexpr\ifodd #1 -2*#1-1\else2*#1+1\fi\relax}%
% the above gives (-1)^n/(2n+1). The sign being in the denominator,
% **** no [0] should be added ****,
% else nothing is guaranteed to work (even if it could by sheer luck)
% NOTE in passing this aspect of \numexpr:
% **** \numexpr -(1)\relax does not work!!! ****
\def\x {1/25[0]}% 1/5^2
\[\mathrm{Arctg}(\frac{1}{5})\approx
\frac{1}{5}\sum_{n=0}^{15}\frac{(-1)^n}{(2n+1)25^n}
= \xintFrac{\xintIrr {\xintDiv
{\xintPowerSeries {0}{15}{\coeffarctg}{\x}{5}}}}{\]
```

$$\mathrm{Arctg}\left(\frac{1}{5}\right) \approx \frac{1}{5} \sum_{n=0}^{15} \frac{(-1)^n}{(2n+1)25^n} = \frac{165918726519122955895391793269168}{840539304153062403202056884765625}$$

## 17.6 \xintPowerSeriesX

New with release 1.04.

This is the same as `\xintPowerSeries` apart from the fact that the last parameter (aka `x`), is first expanded before being then used. If the `x` parameter is to be an explicit big fraction `f` with many (dozens) digits, rather than using `f` directly it is slightly better to have some macro `\x` `\def`'ined to expand to the explicit `f` and use `\xintPowerSeries`; but if `f` has not yet been evaluated and will be the output of a complicated expansion of some `\x`, and if, due to an expanding only context, an `\edef\z{\x}` is no option, then `\xintPowerSeriesX` should be used with `\x` as last parameter. This `\x` will be expanded (as usual) and then its (explicit) output will be used. The reason why `\xintPowerSeries` doesn't do the same is that explicit fractions with many (dozens) digits slow down a bit the processing as there is some shuffling of tokens going on. With `\xintPowerSeriesX` the slowing down in token shuffling due to a very big fraction will not be avoided, but the far worse cost of re-doing each time the computations leading to such a fraction will be. The constraints of expandability make it impossible to encapsulate the result of this initial computation in a macro and have the best of both worlds.

```
\def\ratioexp #1#2{\xintDiv {#1}{#2}}% x/n
% These are the (-1)^{n-1}/n of the log(1+h) series:
\def\coefflog #1{\the\numexpr\ifodd #1 1\else-1\fi\relax/#1[0]}%
% Let L(h) be the first 10 terms of the log(1+h) series and
```

```
% let E(t) be the first 10 terms of the exp(t) series.
% The following computes L(E(a/10)-1) for a=1,..., 12.
\cnta 1
\loop
\noindent\xintTrunc {18}{%
  \xintPowerSeriesX {1}{10}{\coefflog}
  {\xintSub
    {\xintRationalSeries {0}{9}{1[0]}\ratioexp{\the\cnta[-1]}}
    {1}}}\dots
\endgraf
\ifnum\cnta < 12 \advance \cnta 1 \repeat
```

0.099999999998556159...	0.499511320760604148...	-1.597091692317639401...
0.199999995263443554...	0.593980619762352217...	-12.648937932093322763...
0.299999338075041781...	0.645144282733914916...	-66.259639046914679687...
0.399974460740121112...	0.398118280111436442...	-304.768437445462801227...

### 17.7 `\xintFxFtPowerSeries`

`\xintFxFtPowerSeries{A}{B}{\coeff}{x}{D}` computes the sum of  $\text{coeff}\{n\} \cdot x^n$  from  $n=A$  to  $n=B$  with each term of the series truncated to  $D$  digits after the decimal point. As usual,  $A$  and  $B$  are completely expanded through their inclusion in a `\numexpr` expression. Regarding  $D$  it will be similarly be expanded each time it is used inside an `\xintTrunc`. The one-parameter macro `\coeff` is similarly only expanded (in the usual meaning) when it is used inside the computations. Idem for  $x$ . If  $x$  itself is some complicated macro it is thus better to use the variant `\xintFxFtPowerSeriesX` which expands it first and then uses the result of that expansion.

The current (1.04) implementation is: the first power  $x^A$  is computed exactly, then *truncated*. Then each successive power is obtained from the previous one by multiplication by the exact value of  $x$ , and truncated. And  $\text{coeff}\{n\} \cdot x^n$  is obtained from that by multiplying by `\coeff{n}` (untruncated) and then truncating. Finally the sum is computed exactly. Apart from that `\xintFxFtPowerSeries` (where `FxFt` means ‘fixed-point’) is like `\xintPowerSeries`.

There should be a variant for things of the type  $\sum c_n \frac{x^n}{n!}$  to avoid having to compute the factorial from scratch at each coefficient, the same way `\xintFxFtPowerSeries` does not compute  $x^n$  from scratch at each  $n$ . Perhaps in the next package release.

$$e^{-\frac{1}{2}} \approx$$

1.00000000000000000000	0.60653056795634920635	0.60653065971263344622
0.50000000000000000000	0.60653066483754960317	0.60653065971263342289
0.62500000000000000000	0.60653065945526069224	0.60653065971263342361
0.60416666666666666667	0.60653065972437513778	0.60653065971263342359
0.60677083333333333333	0.60653065971214266299	0.60653065971263342359
0.60651041666666666667	0.60653065971265234943	0.60653065971263342359
0.60653211805555555555	0.60653065971263274611	

```
\def\coeffexp #1{1/\xintFac {#1}[0]}% 1/n!
\def\x {-1/2[0]}% [0] for faster input parsing
\def\ApproxExp #1#2{\xintFxFtPowerSeries {0}{#1}{\coeffexp}{\x}{#2}}%
\cnta 0 % previously declared \count register
```

```

\noindent\loop
$\ApproxExp {\cnta}{20}$\ % truncates 20 digits after decimal point
\ifnum\cnta<19 \advance\cnta 1 \repeat\par
% One should **not** trust the final digits, as the potential truncation
% errors of up to  $10^{-20}$  per term accumulate and never disappear! (the
% effect is attenuated by the alternating signs in the series). We can
% confirm that the last two digits (of our evaluation of the nineteenth
% partial sum) are wrong via the evaluation with more digits:

\xintFxFtPowerSeries {0}{19}{\coeffexp}{x}{25}= 0.6065306597126334236037992

```

It is no difficulty for *xintfrac* to compute exactly, with the help of *\xintPowerSeries*, the nineteenth partial sum, and to then give (the start of) its exact decimal expansion:

$$\begin{aligned} \xintPowerSeries {0}{19}{\coeffexp}{x} &= \frac{38682746160036397317757}{63777066403145711616000} \\ &= 0.606530659712633423603799152126\dots \end{aligned}$$

Thus, one should always estimate a priori how many ending digits are not reliable: if there are  $N$  terms and  $N$  has  $k$  digits, then digits up to but excluding the last  $k$  may usually be trusted. If we are optimistic and the series is alternating we may even replace  $N$  with  $\sqrt{N}$  to get the number  $k$  of digits possibly of dubious significance.

## 17.8 *\xintFxFtPowerSeriesX*

New with release 1.04.

*\xintFxFtPowerSeriesX*{A}{B}{\coeff}{x}{D} computes, exactly as *\xintFxFtPowerSeries*, the sum of  $\text{\coeff}{n} \cdot x^n$  from  $n=A$  to  $n=B$  with each term of the series being *truncated* to  $D$  digits after the decimal point. The sole difference is that  $x$  is first expanded and it is the result of this which is used in the computations.

Let us illustrate this on the numerical exploration of the identity

$$\log(1+x) = -\log(1/(1+x))$$

Let  $L(h)=\log(1+h)$ , and  $D(h)=L(h)+L(-h/(1+h))$ . Theoretically thus,  $D(h)=0$  but we shall evaluate  $L(h)$  and  $-h/(1+h)$  keeping only 10 terms of their respective series. We will assume  $|h|<0.5$ . With only ten terms kept in the power series we do not have quite 3 digits precision as  $2^{10}=1024$ . So it wouldn't make sense to evaluate things more precisely than, say circa 5 digits after the decimal points.

```

\cnta 0
\def\coefflog #1{\the\numexpr\ifodd#1 1\else-1\fi\relax/#1[0]}% (-1)^{n-1}/n
\def\coeffalt #1{\the\numexpr\ifodd#1 -1\else1\fi\relax [0]}% (-1)^n
\loop
\noindent \hbox to 2.5cm {\hss\texttt{D(\the\cnta/100): }}%
\xintAdd {\xintFxFtPowerSeriesX {1}{10}{\coefflog}{\the\cnta [-2]}}{5}}
{\xintFxFtPowerSeriesX {1}{10}{\coefflog}
{\xintFxFtPowerSeriesX {1}{10}{\coeffalt}{\the\cnta [-2]}}{5}}
{5}}\endgraf
\ifnum\cnta < 49 \advance\cnta 7 \repeat

```

D(0/100): 0[0]	D(28/100): 4/1[-5]
D(7/100): 2/1[-5]	D(35/100): 4/1[-5]
D(14/100): 2/1[-5]	D(42/100): 9/1[-5]
D(21/100): 3/1[-5]	D(49/100): 42/1[-5]

Let's say we evaluate functions on  $[-1/2, +1/2]$  with values more or less also in  $[-1/2, +1/2]$  and we want to keep 4 digits of precision. So, roughly we need at least 14 terms in series like the geometric or log series. Let's make this 15. Then it doesn't make sense to compute intermediate summands with more than 6 digits precision. So we compute with 6 digits precision but return only 4 digits (rounded) after the decimal point. This result with 4 post-decimal points precision is then used as input to the next evaluation.

```
\loop
\noindent \hbox to 2.5cm {\hss\texttt{D(\the\cnta/100): }}%
\xintRound{4}
{\xintAdd {\xintFxFtPowerSeriesX {1}{15}{\coefflog}{\the\cnta [-2]}}{6}}
{\xintFxFtPowerSeriesX {1}{15}{\coefflog}
{\xintRound {4}{\xintFxFtPowerSeriesX {1}{15}{\coeffalt}
{\the\cnta [-2]}}{6}}}}
{6}}}%
}\endgraf
\ifnum\cnta < 49 \advance\cnta 7 \repeat
```

D(0/100): 0	D(28/100): -0.0001
D(7/100): 0.0000	D(35/100): -0.0001
D(14/100): 0.0000	D(42/100): -0.0000
D(21/100): -0.0001	D(49/100): -0.0001

Not bad... I have cheated a bit: the 'four-digits precise' numeric evaluations were left unrounded in the final addition. However the inner rounding to four digits worked fine and made the next step faster than it would have been with longer inputs. The morale is that one should not use the raw results of `\xintFxFtPowerSeriesX` with the D digits with which it was computed, as the last are to be considered garbage. Rather, one should keep from the output only some smaller number of digits. This will make further computations faster and not less precise. I guess there should be some command to do this final truncating, or better, rounding, at a given number  $D' < D$  of digits. Maybe for the next release.

## 17.9 Computing $\log 2$ and $\pi$

In this final section, the use of `\xintFxFtPowerSeries` (and `\xintPowerSeries`) will be illustrated on the (expandable... why make things simple when it is so easy to make them difficult!) computations of the first digits of the decimal expansion of the familiar constants  $\log 2$  and  $\pi$ .

Let us start with  $\log 2$ . We will get it from this formula (which is left as an exercise):

$$\log(2) = -2 \log(1 - 13/256) - 5 \log(1 - 1/9)$$

The number of terms to be kept in the log series, for a desired precision of  $10^{-D}$  was roughly estimated without much theoretical analysis. Computing exactly the partial sums with `\xintPowerSeries` and then printing the truncated values, from  $D=0$  up to  $D=100$  showed that it worked in terms of quality of the approximation. Because of possible strings of zeros or nines in the exact decimal expansion (in the present case of  $\log 2$ , strings of zeros around the fortieth and the sixtieth decimals), this does not mean though that all

digits printed were always exact. In the end one always end up having to compute at some higher level of desired precision to validate the earlier result.

Then we tried with `\xintFxFtPowerSeries`: this is worthwhile only for  $D$ 's at least 50, as the exact evaluations are faster (with these short-length  $x$ 's) for a lower number of digits. And as expected the degradation in the quality of approximation was in this range of the order of two or three digits. This meant roughly that the  $3+1=4$  ending digits were wrong. Again, we ended up having to compute with five more digits and compare with the earlier value to validate it. We use truncation rather than rounding because our goal is not to obtain the correct rounded decimal expansion but the correct exact truncated one.

```
\def\coefflog #1{1/#1[0]}% 1/n
\def\xa {13/256[0]}% we will compute log(1-13/256)
\def\xb {1/9[0]}% we will compute log(1-1/9)
\def\LogTwo #1%
% get log(2)=-2log(1-13/256)- 5log(1-1/9)
{% we want to use \printnumber, hence need something expanding in two steps
% only, so we use here the \romannumeral0 method
\romannumeral0\expandafter\LogTwoDoIt \expandafter
% Nb Terms for 1/9:
{\the\numexpr #1*150/143\expandafter}\expandafter
% Nb Terms for 13/256:
{\the\numexpr #1*100/129\expandafter}\expandafter
% We print #1 digits, but we know the ending ones are garbage
{\the\numexpr #1\relax}% allows #1 to be a count register
}%
\def\LogTwoDoIt #1#2#3%
% #1=nb of terms for 1/9, #2=nb of terms for 13/256,
{% #3=nb of digits for computations, also used for printing
\xinttrunc {#3} % lowercase form to stop the \romannumeral0 expansion!
{\xintAdd
{\xintMul {2}{\xintFxFtPowerSeries {1}{#2}{\coefflog}{\xa}{#3}}}
{\xintMul {5}{\xintFxFtPowerSeries {1}{#1}{\coefflog}{\xb}{#3}}}%
}%
}%
\noindent $\log 2 \approx \LogTwo {60}\dots\endgraf
\noindent\phantom{$\log 2$}\approx{}\printnumber{\LogTwo {65}}\dots\endgraf
\noindent\phantom{$\log 2$}\approx{}\printnumber{\LogTwo {70}}\dots\endgraf
```

```
log 2 ≈ 0.693147180559945309417232121458176568075500134360255254120484...
≈ 0.6931471805599453094172321214581765680755001343602552541206800071
1...
≈ 0.6931471805599453094172321214581765680755001343602552541206800094
933723...
```

Here is the code doing an exact evaluation of the partial sums. We have added a +1 to the number of digits for estimating the number of terms to keep from the log series: we experimented that this gets exactly the first  $D$  digits, for all values from  $D=0$  to  $D=100$ , except in one case ( $D=40$ ) where the last digit is wrong. For values of  $D$  higher than 100 it is more efficient to use the code using `\xintFxFtPowerSeries`.

```
\def\LogTwo #1% get log(2)=-2log(1-13/256)- 5log(1-1/9)
{%
```

```

\romannumeral0\expandafter\LogTwoDoIt \expandafter
{\the\numexpr (#1+1)*150/143\expandafter}\expandafter
{\the\numexpr (#1+1)*100/129\expandafter}\expandafter
{\the\numexpr #1\relax}%
}%
\def\LogTwoDoIt #1#2#3%
{% #3=nb of digits for truncating an EXACT partial sum
\ointttrunc {#3}
{\ointAdd
{\ointMul {2}{\ointPowerSeries {1}{#2}{\coefflog}{\xa}}}}
{\ointMul {5}{\ointPowerSeries {1}{#1}{\coefflog}{\xb}}}%
}%
}%

```

Let us turn now to  $\pi$ , computed with the Machin formula. Again the numbers of terms to keep in the two arctg series were roughly estimated, and some experimentations showed that removing the last three digits was enough (at least for  $D=0-100$  range). And the algorithm does print the correct digits when used with  $D=1000$  (to be convinced of that one needs to run it for  $D=1000$  and again, say for  $D=1010$ .) A theoretical analysis could help confirm that this algorithm always gets better than  $10^{-D}$  precision, but again, strings of zeros or nines encountered in the decimal expansion may falsify the ending digits, nines may be zeros (and the last non-nine one should be increased) and zeros may be nine (and the last non-zero one should be decreased).

```

% pi = 16 Arctg(1/5) - 4 Arctg(1/239) (John Machin's formula)
\def\coeffarctg #1{\the\numexpr ifodd#1 -1\else 1\fi\relax/%
\the\numexpr 2*#1+1\relax [0]}%
% the above computes  $(-1)^n/(2n+1)$ .
% Alternatives:
% \def\coeffarctg #1{1/\the\numexpr \ointiMON{#1}*(2*#1+1)\relax}%
% The [0] can *not* be used above, as the denominator is signed.
% \def\coeffarctg #1{\ointiMON{#1}/\the\numexpr 2*#1+1\relax [0]}%
\def\xa {1/25[0]}% 1/5^2, the [0] for faster parsing
\def\xb {1/57121[0]}% 1/239^2, the [0] for faster parsing
\def\Machin #1{% \Machin {\mycount} is allowed
\romannumeral0\expandafter\MachinA \expandafter
% number of terms for arctg(1/5):
{\the\numexpr (#1+3)*5/7\expandafter}\expandafter
% number of terms for arctg(1/239):
{\the\numexpr (#1+3)*10/45\expandafter}\expandafter
% do the computations with 3 additional digits:
{\the\numexpr #1+3\expandafter}\expandafter
% allow #1 to be a count register:
{\the\numexpr #1\relax }}%
\def\MachinA #1#2#3#4%
% #4: digits to keep after decimal point for final printing
% #3=#4+3: digits for evaluation of the necessary number of terms
% to be kept in the arctangent series, also used to truncate each
% individual summand.
{\ointttrunc {#4} % must be lowercase to stop \romannumeral0!
{\ointSub
{\ointMul {16/5}{\ointFxPtPowerSeries {0}{#1}{\coeffarctg}{\xa}{#3}}}}

```

```
{\xintMul {4/239}{\xintFxFtPowerSeries {0}{#2}{\coeffarctg}{\xb}{#3}}}%
}}%
\[\pi = \Machin {60}\dots \]
```

$$\pi = 3.141592653589793238462643383279502884197169399375105820974944 \dots$$

Here is a variant `\MachinBis`, which evaluates the partial sums *exactly* using `\xintPowerSeries`, before their final truncation. No need for a “+3” then.

```
\def\MachinBis #1{% #1 may be a count register,
% the final result will be truncated to #1 digits post decimal point
  \romannumeral0\expandafter\MachinBisA \expandafter
    % number of terms for arctg(1/5):
    {\the\numexpr #1*5/7\expandafter}\expandafter
    % number of terms for arctg(1/239):
    {\the\numexpr #1*10/45\expandafter}\expandafter
    % allow #1 to be a count register:
    {\the\numexpr #1\relax }}%
\def\MachinBisA #1#2#3%
{\xinttrunc {#3} %
{\xintSub
  {\xintMul {16/5}{\xintPowerSeries {0}{#1}{\coeffarctg}{\xa}}}
  {\xintMul{4/239}{\xintPowerSeries {0}{#2}{\coeffarctg}{\xb}}}%
}}%
```

Let us use this variant for a loop showing the build-up of digits:

```
\cnta 0 % previously declared \count register
\loop
\MachinBis{\cnta} \endgraf % Plain's \loop does not accept \par
\ifnum\cnta < 30 \advance\cnta 1 \repeat
```

	3.141592653589793
3.	3.1415926535897932
3.1	3.14159265358979323
3.14	3.141592653589793238
3.141	3.1415926535897932384
3.1415	3.14159265358979323846
3.14159	3.141592653589793238462
3.141592	3.1415926535897932384626
3.1415926	3.14159265358979323846264
3.14159265	3.141592653589793238462643
3.141592653	3.1415926535897932384626433
3.1415926535	3.14159265358979323846264338
3.14159265358	3.141592653589793238462643383
3.141592653589	3.1415926535897932384626433832
3.1415926535897	3.14159265358979323846264338327
3.14159265358979	3.141592653589793238462643383279



## 18 Commands of the **xintcfrac** package

You want more digits and have some time? Copy the `\Machin` code to a Plain  $\text{\TeX}$  or  $\text{\LaTeX}$  document loading **xintseries**, and compile:

```
\newwrite\outfile
\immediate\openout\outfile \jobname-out\relax
\immediate\write\outfile {\Machin {1000}}
\immediate\closeout\outfile
```

This will create a file with the correct first 1000 digits of  $\pi$  after the decimal point. On my laptop (a 2012 model) this took about 44 seconds last time I tried (and for 200 digits it is less than 1 second). As mentioned in the introduction, the file `pi.tex` by D. ROEGEL shows that orders of magnitude faster computations are possible within  $\text{\TeX}$ , but recall our constraints of complete expandability and be merciful, please.

**Why truncating rather than rounding?** One of our main competitors on the market of scientific computing, a canadian product (not encumbered with expandability constraints, and having barely ever heard of  $\text{\TeX}$  ;-), prints numbers rounded in the last digit. Why didn't we follow suit in the macros `\xintFxFtPowerSeries` and `\xintFxFtPowerSeriesX`? To round at  $D$  digits, and excluding a rewrite or cloning of the division algorithm which anyhow would add to it some overhead in its final steps, **xintcfrac** needs to truncate at  $D+1$ , then round. And rounding loses information! So, with more time spent, we obtain a worst result than the one truncated at  $D+1$  (one could imagine that additions and so on, done with only  $D$  digits, cost less; true, but this is a negligible effect per summand compared to the additional cost for this term of having been truncated at  $D+1$  then rounded). Rounding is the way to go when setting up algorithms to evaluate functions destined to be composed one after the other: exact algebraic operations with many summands and an  $x$  variable which is a fraction are costly and create an even bigger fraction; replacing  $x$  with a reasonable rounding, and rounding the result, is necessary to allow arbitrary chaining.

But, for the computation of a single constant, we are really interested in the exact decimal expansion, so we truncate and compute more terms until the earlier result gets validated. Finally if we do want the rounding we can always do it on a value computed with  $D+1$  truncation.

## 18 Commands of the **xintcfrac** package

This package was first included in release 1.04 of the **xint** bundle.

### 18.1 Package overview

A *simple* continued fraction has coefficients  $[c_0, c_1, \dots, c_N]$  (usually called partial quotients, but I really dislike this entrenched terminology), where  $c_0$  is a positive or negative integer and the others are positive integers. As we will see it is possible with **xintcfrac** to specify the coefficient function  $c:n \rightarrow c_n$ . Note that the index then starts at zero as indi-

cated. With the `amsmath` macro `\cfrac` one can display such a continued fraction as

$$c_0 + \frac{1}{c_1 + \frac{1}{c_2 + \frac{1}{c_3 + \frac{1}{\ddots}}}}$$

Here is a concrete example:

$$\frac{208341}{66317} = 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{1 + \frac{1}{292 + \frac{1}{2}}}}}$$

But the difference with `amsmath`'s `\cfrac` is that this was input as

```
\[ \xintFrac {208341/66317}=\xintCFrac {208341/66317} \]
```

The command `\xintCFrac` produces in two expansion steps the whole thing with the many chained `\cfrac`'s and all necessary braces, ready to be printed, in math mode. This is  $\text{\LaTeX}$  only and with the `amsmath` package (we shall mention another method for Plain  $\text{\TeX}$  users of `amstex`).

A *generalized* continued fraction has the same structure but the numerators are not restricted to be ones, and numbers used in the continued fraction may be arbitrary, also fractions, irrationals, indeterminates. The *centered* continued fraction associated to a rational number is an example:

```
\[ \xintFrac {915286/188421}=\xintGCFrac {\xintFtoCC {915286/188421}} \]
```

$$\frac{915286}{188421} = 5 - \frac{1}{7 + \frac{1}{39 - \frac{1}{53 - \frac{1}{13}}}} = 4 + \frac{1}{1 + \frac{1}{6 + \frac{1}{38 + \frac{1}{1 + \frac{1}{51 + \frac{1}{1 + \frac{1}{12}}}}}}}$$

The command `\xintGCFrac`, contrarily to `\xintCFrac`, does not compute anything, it just typesets. Here, it is the command `\xintFtoCC` which did the computation of the centered continued fraction of  $f$ . Its output has the ‘inline format’ described in the next paragraph. In the display, we also used `\xintCFrac` (code not shown), for comparison of the two types of continued fractions.

A generalized continued fraction may be input ‘inline’ as:

$$a_0 + b_0/a_1 + b_1/a_2 + b_2/\dots/a_{(n-1)} + b_{(n-1)}/a_n$$

Fractions among the coefficients are allowed but they must be enclosed within braces. Signed integers may be left without braces (but the + signs are mandatory). Or, they may be macros expanding (in two steps) to some number or fractional number.

`\xintGCFrac {1+-1/57+\xintPow {-3}{7}/\xintQuo {132}{25}}`

$$\frac{1907}{1902} = 1 - \frac{1}{57 - \frac{2187}{5}}$$

The left hand side was obtained with the following code:

`\xintFrac{\xintGCToF {1+-1/57+\xintPow {-3}{7}/\xintQuo {132}{25}}}`

It uses the macro `\xintGCToF` to convert a generalized fraction from the ‘inline format’ to the fraction it evaluates to.

A simple continued fraction is a special case of a generalized continued fraction and may be input as such to macros expecting the ‘inline format’, for example  $-7 + 1/6 + 1/19 + 1/1 + 1/33$ . There is a simpler comma separated format:

`\xintFrac{\xintCstoF{-7,6,19,1,33}}=\xintCFrac{\xintCstoF{-7,6,19,1,33}}`

$$\frac{-28077}{4108} = -7 + \frac{1}{6 + \frac{1}{19 + \frac{1}{1 + \frac{1}{33}}}}$$

This comma separated format may also be used with fractions among the coefficients: of course in that case, computing with `\xintFtoCs` from the resulting `f` its real coefficients will give a new comma separated list with only integers. This list has no spaces: the spaces in the display below arise from the math mode processing.

`\xintFrac{1084483/398959}=[\xintFtoCs{1084483/398959}]`

$$\frac{1084483}{398959} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 2]$$

If one prefers other separators, one can use `\xintFtoCx` whose first argument will be the separator to be used.

`\xintFrac{2721/1001}=\xintFtoCx {+1/({}{2721/1001})\cdots}`

$$\frac{2721}{1001} = 2 + 1/(1 + 1/(2 + 1/(1 + 1/(1 + 1/(4 + 1/(1 + 1/(1 + 1/(6 + 1/(2) \cdots)))$$

People using Plain  $\text{\TeX}$  and `amstex` can achieve the same effect as `\xintCFrac` with: `$$\xintFwOver{2721/1001}=\xintFtoCx {+\cfrac1{\ }}{2721/1001}\endcfrac$$`

Using `\xintFtoCx` with first argument an empty pair of braces `{}` will return the list of the coefficients of the continued fraction of `f`, without separator, and each one enclosed in a pair of group braces. This can then be manipulated by the non-expandable macro `\xintAssignArray` or the expandable ones `\xintApply` and `\xintListWithSep`.

As a shortcut to using `\xintFtoCx` with separator `1+/,` there is `\xintFtoGC`:

`2721/1001=\xintFtoGC {2721/1001}`  
`2721/1001=2+1/1+1/2+1/1+1/1+1/4+1/1+1/1+1/6+1/2`

Let us compare in that case with the output of `\xintFtoCC`:

$$\begin{aligned} 2721/1001 &= \text{\xintFtoCC } \{2721/1001\} \\ 2721/1001 &= 3 + -1/4 + -1/2 + 1/5 + -1/2 + 1/7 + -1/2 \end{aligned}$$

The ‘`\printnumber`’ macro which we use to print long numbers can also be useful on long continued fractions.

$$\begin{aligned} &\text{\printnumber}\{\text{\xintFtoCC } \{35037018906350720204351049/\% \\ &\quad 244241737886197404558180\}\} \\ 143 + 1/2 + 1/5 + -1/4 + -1/4 + -1/4 + -1/3 + 1/2 + 1/2 + 1/6 + -1/22 + 1/2 + 1/10 + -1/5 + -1/11 + - \\ 1/3 + 1/4 + -1/2 + 1/2 + 1/4 + -1/2 + 1/23 + 1/3 + 1/8 + -1/6 + -1/9. \text{ If we apply } \text{\xintGctoF} \text{ to } \\ \text{\xintGctoF } \{143 + 1/2 + \dots + -1/9\} &= 2897319801297630107/20197107104701740[0] \end{aligned}$$

When a generalized continued fraction is built with integers, and numerators are only 1’s or –1’s, the produced fraction is irreducible. And if we compute it again with the last sub-fraction omitted we get another irreducible fraction related to the bigger one by a Bezout identity. Doing this here we get:

$$\text{\xintGctoF } \{143 + 1/2 + \dots + -1/6\} = 328124887710626729/2287346221788023[0]$$

and indeed:

$$\begin{vmatrix} 2897319801297630107 & 328124887710626729 \\ 20197107104701740 & 2287346221788023 \end{vmatrix} = 1$$

More generally the various fractions obtained from the truncation of a continued fraction to its initial terms are called the convergents. The commands of *xintcfrac* such as `\xintFtoCv`, `\xintFtoCCv`, and others which compute such convergents, return them as a list of braced items, with no separator. This list can then be treated either with `\xintAssignArray`, or `\xintListWithSep`, or any other way (but then, some T<sub>E</sub>X programming knowledge will be necessary). Here is an example:

$$\begin{aligned} &\$ \text{\xintFrac}\{915286/188421\} \text{\to } \text{\xintListWithSep } \{, \} \% \\ &\{ \text{\xintApply}\{\text{\xintFrac}\}\{\text{\xintFtoCv}\{915286/188421\}\} \} \$ \end{aligned}$$

$$\frac{915286}{188421} \rightarrow 4, 5, \frac{34}{7}, \frac{1297}{267}, \frac{1331}{274}, \frac{69178}{14241}, \frac{70509}{14515}, \frac{915286}{188421}$$

$$\begin{aligned} &\$ \text{\xintFrac}\{915286/188421\} \text{\to } \text{\xintListWithSep } \{, \} \% \\ &\{ \text{\xintApply}\{\text{\xintFrac}\}\{\text{\xintFtoCCv}\{915286/188421\}\} \} \$ \end{aligned}$$

$$\frac{915286}{188421} \rightarrow 5, \frac{34}{7}, \frac{1331}{274}, \frac{70509}{14515}, \frac{915286}{188421}$$

We thus see that the ‘centered convergents’ obtained with `\xintFtoCCv` are among the fuller list of convergents as returned by `\xintFtoCv`.

Here is a more complicated use of `\xintApply` and `\xintListWithSep`. We first define a macro which will be applied to each convergent:

`\newcommand{\mymacro}[1]{\$ \text{\xintFrac}\{#1\} = [\text{\xintFtoCs}\{#1\}] \$ \text{\vtop to 6pt}\}`  
Next, we use the following code:

$$\begin{aligned} &\$ \text{\xintFrac}\{49171/18089\} \text{\to } \{ \$ \\ &\quad \text{\xintListWithSep } \{, \} \{ \text{\xintApply}\{\text{\mymacro}\}\{\text{\xintFtoCv}\{49171/18089\}\} \} \end{aligned}$$

It produces:

$\frac{49171}{18089} \rightarrow 2 = [2], 3 = [3], \frac{8}{3} = [2, 1, 2], \frac{11}{4} = [2, 1, 3], \frac{19}{7} = [2, 1, 2, 2], \frac{87}{32} = [2, 1, 2, 1, 1, 4],$   
 $\frac{106}{39} = [2, 1, 2, 1, 1, 5], \frac{193}{71} = [2, 1, 2, 1, 1, 4, 2], \frac{1264}{465} = [2, 1, 2, 1, 1, 4, 1, 1, 6], \frac{1457}{536} =$   
 $[2, 1, 2, 1, 1, 4, 1, 1, 7], \frac{2721}{1001} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 2], \frac{23225}{8544} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8],$   
 $\frac{49171}{18089} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 2].$

The macro `\xintCntoF` allows to specify the coefficients as functions of the index. The values to which expand the coefficient function do not have to be integers.

```
\def\cn #1{\xintiPow {2}{#1}}% 2^n
\[\xintFrac{\xintCntoF {6}{\cn}}=\xintCFrac [1]{\xintCntoF {6}{\cn}}\]
```

$$\frac{3541373}{2449193} = 1 + \frac{1}{2 + \frac{1}{4 + \frac{1}{8 + \frac{1}{16 + \frac{1}{32 + \frac{1}{64}}}}}}$$

Notice the use of the optional argument [1] to `\xintCFrac`. Other possibilities are [r] and (default) [c].

```
\def\cn #1{\xintiPow {2}{-#1}}% 1/2^n
\[\xintFrac{\xintCntoF {6}{\cn}} = \xintGCFrac [r]{\xintCntoGC {6}{\cn}}
= [\xintFtoCs {\xintCntoF {6}{\cn}}]\]
```

$$\frac{3159019}{2465449} = 1 + \frac{1}{\frac{1}{2} + \frac{1}{\frac{1}{4} + \frac{1}{\frac{1}{8} + \frac{1}{\frac{1}{16} + \frac{1}{\frac{1}{32} + \frac{1}{64}}}}}} = [1, 3, 1, 1, 4, 14, 1, 1, 1, 1, 79, 2, 1, 1, 2]$$

We used `\xintCntoGC` as we wanted to display also the continued fraction and not only the fraction returned by `\xintCntoF`.

There are also `\xintGcntoF` and `\xintGcntoGC` which allow the same for generalized fractions. The following initial portion of a generalized continued fraction for  $\pi$ :

$$\frac{92736}{29520} = \frac{4}{1 + \frac{1}{3 + \frac{4}{5 + \frac{9}{7 + \frac{16}{9 + \frac{25}{11}}}}}} = 3.1414634146\dots$$

was obtained with this code:

```
\def\an #1{\the\numexpr 2*#1+1\relax }%
\def\bn #1{\the\numexpr (#1+1)*(#1+1)\relax }%
\[ \xintFrac{\xintDiv {4}{\xintGCntoF {5}{\an}{\bn}}}{\cfrac{4}{\xintGCFrac{\xintGCntoGC {5}{\an}{\bn}}}} =
\xintTrunc {10}{\xintDiv {4}{\xintGCntoF {5}{\an}{\bn}}}\dots\]
```

We see that the quality of approximation is not fantastic compared to the simple continued fraction of  $\pi$  with about as many terms:

```
\[ \xintFrac{\xintCstoF{3,7,15,1,292,1,1}}{=
\xintGCFrac{3+1/7+1/15+1/1+1/292+1/1+1/1}{=
\xintTrunc{10}{\xintCstoF{3,7,15,1,292,1,1}}\dots\]
```

$$\frac{208341}{66317} = 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{1 + \frac{1}{292 + \frac{1}{1 + \frac{1}{1}}}}}} = 3.1415926534\dots$$

To conclude this overview of most of the package functionalities, let us explore the convergents of Euler's number  $e$ .

```
\def\cn #1{\the\numexpr\ifcase \numexpr #1+3-3*((#1+2)/3)\relax
1\or1\or2*(#1/3)\fi\relax }
% produces the pattern 1,1,2,1,1,4,1,1,6,1,1,8,... which are the
% coefficients of the simple continued fraction of e-1.
\cnta 0
\def\mymacro #1{\advance\cnta by 1
\noindent
\hbox to 3em {\hfil\small\texttt{\the\cnta.} }%
$\xintTrunc {30}{\xintAdd {1[0]}{\#1}}\dots=
\xintFrac{\xintAdd {1[0]}{\#1}}{\$}%
\xintListWithSep{\vtop to 6pt}{\vbox to 12pt}{\par}
{\xintApply\mymacro{\xintiCstoCv{\xintCntoCs {35}{\cn}}}}
```

The volume of computation is kept minimal by the following steps:

- a comma separated list of the first 36 coefficients is produced by `\xintCntoCs`,
- this is then given to `\xintiCstoCv` which produces the list of the convergents (there is also `\xintCstoCv`, but our coefficients being integers we used the infinitesimally faster `\xintiCstoCv`),
- then the whole list was converted into a sequence of one-line paragraphs, each convergent becomes the argument to a macro printing it together with its decimal expansion with 30 digits after the decimal point.

- A count register \cnta was used to give a line count serving as a visual aid: we could also have done that in an expandable way, but well, let's relax from time to time...

1.  $2.00000000000000000000000000000000 \dots = 2$
2.  $3.00000000000000000000000000000000 \dots = 3$
3.  $2.66666666666666666666666666666666 \dots = \frac{8}{3}$
4.  $2.75000000000000000000000000000000 \dots = \frac{11}{4}$
5.  $2.714285714285714285714285714285 \dots = \frac{19}{7}$
6.  $2.71875000000000000000000000000000 \dots = \frac{87}{32}$
7.  $2.717948717948717948717948717948 \dots = \frac{106}{39}$
8.  $2.718309859154929577464788732394 \dots = \frac{193}{71}$
9.  $2.718279569892473118279569892473 \dots = \frac{1264}{465}$
10.  $2.718283582089552238805970149253 \dots = \frac{1457}{536}$
11.  $2.718281718281718281718281718281 \dots = \frac{2721}{1001}$
12.  $2.718281835205992509363295880149 \dots = \frac{23225}{8544}$
13.  $2.718281822943949711891042430591 \dots = \frac{25946}{9545}$
14.  $2.718281828735695726684725523798 \dots = \frac{49171}{18089}$
15.  $2.718281828445401318035025074172 \dots = \frac{517656}{190435}$
16.  $2.718281828470583721777828930962 \dots = \frac{566827}{208524}$
17.  $2.718281828458563411277850606202 \dots = \frac{1084483}{398959}$
18.  $2.718281828459065114074529546648 \dots = \frac{13580623}{4996032}$
19.  $2.718281828459028013207065591026 \dots = \frac{14665106}{5394991}$
20.  $2.718281828459045851404621084949 \dots = \frac{28245729}{10391023}$
21.  $2.718281828459045213521983758221 \dots = \frac{410105312}{150869313}$
22.  $2.718281828459045254624795027092 \dots = \frac{438351041}{161260336}$
23.  $2.718281828459045234757560631479 \dots = \frac{848456353}{312129649}$
24.  $2.718281828459045235379013372772 \dots = \frac{14013652689}{5155334720}$
25.  $2.718281828459045235343535532787 \dots = \frac{14862109042}{5467464369}$
26.  $2.718281828459045235360753230188 \dots = \frac{28875761731}{10622799089}$
27.  $2.718281828459045235360274593941 \dots = \frac{534625820200}{196677847971}$
28.  $2.718281828459045235360299120911 \dots = \frac{563501581931}{207300647060}$
29.  $2.718281828459045235360287179900 \dots = \frac{1098127402131}{403978495031}$
30.  $2.718281828459045235360287478611 \dots = \frac{22526049624551}{8286870547680}$
31.  $2.718281828459045235360287464726 \dots = \frac{23624177026682}{8690849042711}$



32.  $2.718281828459045235360287471503 \dots = \frac{46150226651233}{16977719590391}$
33.  $2.718281828459045235360287471349 \dots = \frac{1038929163353808}{382200680031313}$
34.  $2.718281828459045235360287471355 \dots = \frac{1085079390005041}{399178399621704}$
35.  $2.718281828459045235360287471352 \dots = \frac{2124008553358849}{781379079653017}$
36.  $2.718281828459045235360287471352 \dots = \frac{52061284670617417}{19152276311294112}$

The actual computation of the list of all 36 convergents accounts for only 8% of the total time (total time equal to about 5 hundredths of a second in my testing, on my laptop): another 80% is occupied with the computation of the truncated decimal expansions (and the addition of 1 to everything as the formula gives the continued fraction of  $e - 1$ ). One can with no problem compute much bigger convergents. Let's get the 200th convergent. It turns out to have the same first 268 digits after the decimal point as  $e - 1$ . Higher convergents get more and more digits in proportion to their index: the 500th convergent already gets 799 digits correct! To allow speedy compilation of the source of this document when the need arises, I limit here to the 200th convergent (getting the 500th took about 1.2s on my laptop last time I tried, and the 200th convergent is obtained ten times faster).

```
\edef\z {\xintCnToF {199}{\cn}}%
\begin{group}\par\indent 0pt \leftskip 2.5cm
\indent\llap {Numerator = }\printnumber{\xintNumerator\z}\par
\indent\llap {Denominator = }\printnumber{\xintDenominator\z}\par
\indent\llap {Expansion = }\printnumber{\xintTrunc{268}\z}\dots
\par\endgroup
```

Numerator = 56896403887189626759752389231580787529388901766791744605723  
20245471922969611182301752438601749953108177313670124170860  
9749634329382906

Denominator = 33112381766973761930625636081635675336546882372931443815620  
56154632466597285818654613376920631489160195506145705925533  
7661142645217223

Expansion = 1.718281828459045235360287471352662497757247093699959574966  
96762772407663035354759457138217852516642742746639193200305  
99218174135966290435729003342952605956307381323286279434907  
63233829880753195251019011573834187930702154089149934884167  
5092447614606680822648001684774118...

One can also use a centered continued fraction: we get more digits but there are also more computations as the numerators may be either 1 or  $-1$ .

## 18.2 `\xintCFrac`

`\xintCFrac{f}` is a math-mode only, L<sup>A</sup>T<sub>E</sub>X with `amsmath` only, macro which first computes then displays with the help of `\cfrac` the simple continued fraction corresponding to the given fraction (or macro expanding in two steps to one such). It admits an optional argument which may be [l], [r] or (the default) [c] to specify the location of the one's in the numerators of the sub-fractions. Each coefficient is typeset using the `\xintFrac` macro from the *xintfrac* package.

### 18.3 `\xintGCFrac`

`\xintGCFrac{a+b/c+d/e+f/g+h/...}` uses similarly `\cfrac` to typeset a generalized continued fraction in inline format. It admits the same optional argument as `\xintCFrac`.

`\[\xintGCFrac {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintFac {6}}\]`

$$1 + \frac{3375 \cdot 10^{-3}}{\frac{1}{7} - \frac{\frac{3}{5}}{720}}$$

As can be seen this is typesetting macro, although it does proceed to the evaluation of the coefficients themselves. See `\xintGctoF` if you are impatient to see this fraction computed. Numerators and denominators are made arguments to the `\xintFrac` macro.

### 18.4 `\xintGctoGCx`

New with release 1.05.

`\xintGctoGCx{sepa}{sepb}{a+b/c+d/e+f/...+x/y}` returns the list of the coefficients of the generalized continued fraction of  $f$ , each one within a pair of braces, and separated with the help of `sepa` and `sepb`. Thus

`\xintGctoGCx :;{1+2/3+4/5+6/7}` gives 1:2;3:4;5:6;7

Plain  $\text{\TeX}$ +`amstex` users may be interested in:

`$$\xintGctoGCx {+\cfrac{}{}{a+b/...}}\endcfrac$$`

`$$\xintGctoGCx {+\cfrac\xintFwOver{}{}{a+b/...}}\endcfrac$$`

### 18.5 `\xintFtoCs`

`\xintFtoCs{f}` returns the comma separated list of the coefficients of the simple continued fraction of  $f$ .

`\[ \xintSignedFrac{-5262046/89233} = [\xintFtoCs{-5262046/89233}]\]`

$$-\frac{5262046}{89233} = [-59, 33, 27, 100]$$

### 18.6 `\xintFtoCx`

`\xintFtoCx{sep}{f}` returns the list of the coefficients of the simple continued fraction of  $f$ , withing group braces and separated with the help of `sep`.

`$$\xintFtoCx {+\cfrac1{}{}{f}}\endcfrac$$`

will display the continued fraction in `\cfrac` format, with Plain  $\text{\TeX}$  and `amstex`.

### 18.7 `\xintFtoGC`

`\xintFtoGC{f}` does the same as `\xintFtoCx{+1/}{f}`. Its output may thus be used in the package macros expecting such an ‘inline format’. This continued fraction is a *simple* one, not a *generalized* one, but as it is produced in the format used for user input of generalized continued fractions, the macro was called `\xintFtoGC` rather than `\xintFtoC` for example.

`566827/208524=\xintFtoGC {566827/208524}`

`566827/208524=2+1/1+1/2+1/1+1/1+1/4+1/1+1/1+1/6+1/1+1/1+1/8+1/1+1/1+1/11`

**18.8 \xintFtoCC**

`\xintFtoCC{f}` returns the ‘centered’ continued fraction of  $f$ , in ‘inline format’.

```
566827/208524=\xintFtoCC {566827/208524}
566827/208524=3+-1/4+-1/2+1/5+-1/2+1/7+-1/2+1/9+-1/2+1/11
\[\xintFrac{566827/208524} = \xintGCFrac{\xintFtoCC{566827/208524}}\]
```

$$\frac{566827}{208524} = 3 - \frac{1}{4 - \frac{1}{2 + \frac{1}{5 - \frac{1}{2 + \frac{1}{7 - \frac{1}{2 + \frac{1}{9 - \frac{1}{2 + \frac{1}{11}}}}}}}}}$$

**18.9 \xintFtoCv**

`\xintFtoCv{f}` returns the list of the (braced) convergents of  $f$ , with no separator. To be treated with `\xintAssignArray` or `\xintListWithSep`.

```
\[\xintListWithSep{\to}\{\xintApply\xintFrac{\xintFtoCv{5211/3748}}\}\]
```

$$1 \rightarrow \frac{3}{2} \rightarrow \frac{4}{3} \rightarrow \frac{7}{5} \rightarrow \frac{25}{18} \rightarrow \frac{32}{23} \rightarrow \frac{57}{41} \rightarrow \frac{317}{228} \rightarrow \frac{374}{269} \rightarrow \frac{691}{497} \rightarrow \frac{5211}{3748}$$

**18.10 \xintFtoCCv**

`\xintFtoCCv{f}` returns the list of the (braced) centered convergents of  $f$ , with no separator. To be treated with `\xintAssignArray` or `\xintListWithSep`.

```
\[\xintListWithSep{\to}\{\xintApply\xintFrac{\xintFtoCCv{5211/3748}}\}\]
```

$$1 \rightarrow \frac{4}{3} \rightarrow \frac{7}{5} \rightarrow \frac{32}{23} \rightarrow \frac{57}{41} \rightarrow \frac{374}{269} \rightarrow \frac{691}{497} \rightarrow \frac{5211}{3748}$$

**18.11 \xintCstoF**

`\xintCstoF{a,b,c,d,...,z}` computes the fraction corresponding to the coefficients, which may be fractions or even macros expanding to such fractions (in two steps). The final fraction may then be highly reducible.

```
\[\xintGCFrac {-1+1/3+1/-5+1/7+1/-9+1/11+1/-13}
=\xintSignedFrac{\xintCstoF {-1,3,-5,7,-9,11,-13}}\]
```

`=\xintSignedFrac{\xintGCtoF {-1+1/3+1/-5+1/7+1/-9+1/11+1/-13}}\]`

$$-1 + \frac{1}{3 + \frac{1}{-5 + \frac{1}{7 + \frac{1}{-9 + \frac{1}{11 + \frac{1}{-13}}}}}} = -\frac{75887}{118187} = -\frac{75887}{118187}$$

`\xintGCFrac{{1/2}+1/{1/3}+1/{1/4}+1/{1/5}}=`  
`\xintFrac{\xintCstoF {1/2,1/3,1/4,1/5}}`

$$\frac{1}{2} + \frac{1}{\frac{1}{\frac{1}{3} + \frac{1}{\frac{1}{4} + \frac{1}{\frac{1}{5}}}}} = \frac{159}{66}$$

A generalized continued fraction may produce a reducible fraction (`\xintCstoF` tries its best not to accumulate in a silly way superfluous factors but will not do simplifications which would be obvious to a human, like simplification by 3 in the result above).

## 18.12 `\xintCstoCv`

`\xintCstoCv{a,b,c,d,...,z}` returns the list of the corresponding convergents. It is allowed to use fractions as coefficients (the computed convergents have then no reason to be the real convergents of the final fraction). When the coefficients are integers, the convergents are irreducible fractions, but otherwise it is of course not necessarily the case.

`\xintListWithSep{\xintCstoCv{1,2,3,4,5,6}}`  
`1/1[0]:3/2[0]:10/7[0]:43/30[0]:225/157[0]:1393/972[0]`  
`\xintListWithSep{\xintCstoCv{1,1/2,1/3,1/4,1/5,1/6}}`  
`1/1[0]:3/1[0]:9/7[0]:45/19[0]:225/159[0]:1575/729[0]`

I know that these `[0]` are a bit annoying<sup>23</sup> but this is the way **xintfrac** likes to reception fractions: this format is best for further processing by the bundle macros. For ‘inline’ printing, one may apply `\xintRaw` and for display in math mode `\xintFrac`.

`\[\xintListWithSep{\to}{\xintApply\xintFrac{\xintCstoCv`  
`{\xintPow {- .3}{-5},7.3/4.57,\xintCstoF{3/4,9,-1/3}}}\]`

$$\frac{-100000}{243} \rightarrow \frac{-72888949}{177390} \rightarrow \frac{-2700356878}{6567804}$$

## 18.13 `\xintCstoGC`

`\xintCstoGC{a,b,...,z}` transforms a comma separated list (or something expanding to such a list) into an ‘inline format’ continued fraction  $\{a\}+1/\{b\}+1/\dots+1/\{z\}$ . The coefficients are just copied and put within braces, without expansion. The output can then be

<sup>23</sup>and the awful truth is that it is added forcefully by `\xintCstoCv` at the last step...

used in `\xintGCFrac` for example.

```
\[\xintGCFrac {\xintCstoGC {-1,1/2,-1/3,1/4,-1/5}}
=\xintSignedFrac {\xintCstoF {-1,1/2,-1/3,1/4,-1/5}}\]
```

$$-1 + \frac{1}{\frac{\frac{1}{2} + \frac{1}{\frac{-1}{3} + \frac{1}{\frac{1}{4} + \frac{1}{\frac{-1}{5}}}}}} = -\frac{145}{83}$$

### 18.14 `\xintGctoF`

`\xintGctoF{a+b/c+d/e+f/g+.....+v/w+x/y}` computes the fraction defined by the in-line generalized continued fraction. Coefficients may be fractions but must then be put within braces. They can be macros. The plus signs are mandatory.

```
\[\xintGCFrac {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintFac {6}} =
\xintFrac{\xintGctoF {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintFac {6}}} =
\xintFrac{\xintIrr{\xintGctoF
{1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintFac {6}}}}\]
```

$$1 + \frac{3375 \cdot 10^{-3}}{\frac{1}{7} - \frac{\frac{3}{5}}{720}} = \frac{88629000}{3579000} = \frac{29543}{1193}$$

```
\[ \xintGCFrac{{1/2}+{2/3}/{4/5}+{1/2}/{1/5}+{3/2}/{5/3}} =
\xintFrac{\xintGctoF {{1/2}+{2/3}/{4/5}+{1/2}/{1/5}+{3/2}/{5/3}}} \]
```

$$\frac{1}{2} + \frac{\frac{2}{3}}{\frac{\frac{4}{5} + \frac{\frac{1}{2}}{\frac{3}{\frac{1}{5} + \frac{2}{5}}}}}{\frac{3}{2}} = \frac{4270}{4140}$$

The macro tries its best not to accumulate superfluous factor in the denominators, but doesn't reduce the fraction to irreducible form before returning it and does not do simplifications which would be obvious to a human.

### 18.15 `\xintGctoCv`

`\xintGctoCv{a+b/c+d/e+f/g+.....+v/w+x/y}` returns the list of the corresponding convergents. The coefficients may be fractions, but must then be inside braces. Or they may be macros, too.

The convergents will in the general case be reducible. To put them into irreducible form, one needs one more step, for example it can be done with `\xintApply\xintIrr`.

```
\[\xintListWithSep{,}\{\xintApply\xintFrac
{\xintGctoCv{3+{-2}/{7/2}+{3/4}/12+{-56}/3}}\}\]
\[\xintListWithSep{,}\{\xintApply\xintFrac{\xintApply\xintIrr
{\xintGctoCv{3+{-2}/{7/2}+{3/4}/12+{-56}/3}}}\}\]
```

$$3, \frac{17}{7}, \frac{834}{342}, \frac{1306}{542}$$

$$3, \frac{17}{7}, \frac{139}{57}, \frac{653}{271}$$

**18.16 \xintCntoF**

`\xintCntoF{N}{\macro}` computes the fraction  $f$  having coefficients  $c(j)=\macro{j}$  for  $j=0, 1, \dots, N$ . The  $N$  parameter is given to a `\numexpr`. The values of the coefficients, as returned by `\macro` do not have to be positive, nor integers, and it is thus not necessarily the case that the original  $c(j)$  are the true coefficients of the final  $f$ .

```
\def\macro #1{\the\numexpr 1+#1*#1\relax}\xintCntoF {5}{\macro}
72625/49902[0]
```

**18.17 \xintGCntoF**

`\xintGCntoF{N}{\macroA}{\macroB}` returns the fraction  $f$  corresponding to the inline generalized continued fraction  $a_0 + b_0/a_1 + b_1/a_2 + \dots + b_{(N-1)}/a_N$ , with  $a(j)=\macroA{j}$  and  $b(j)=\macroB{j}$ . The  $N$  parameter is given to a `\numexpr`.

$$1 + \frac{1}{2 - \frac{1}{3 + \frac{1}{1 - \frac{1}{2 + \frac{1}{3 - \frac{1}{1}}}}}} = \frac{39}{25}$$

There is also `\xintGCntoGC` to get the ‘inline format’ continued fraction. The previous display was obtained with:

```
\def\coeffA #1{\the\numexpr #1+4-3*((#1+2)/3)\relax }%
\def\coeffB #1{\xintMON{#1}}% (-1)^n
\[\xintGCfrac{\xintGCntoGC {6}{\coeffA}{\coeffB}}
= \xintFrac{\xintCntoF {6}{\coeffA}{\coeffB}}\]
```

**18.18 \xintCntoCs**

`\xintCntoCs{N}{\macro}` produces the comma separated list of the corresponding coefficients, from  $n=0$  to  $n=N$ . The  $N$  is given to a `\numexpr`.

```
\def\macro #1{\the\numexpr 1+#1*#1\relax}\xintCntoCs {5}{\macro}
1,2,5,10,17,26
```

`\[\xintFrac{\xintCntoF {5}{\macro}}=\xintCFrac{\xintCntoF {5}{\macro}}\]`

$$\frac{72625}{49902} = 1 + \frac{1}{2 + \frac{1}{5 + \frac{1}{10 + \frac{1}{17 + \frac{1}{26}}}}}$$

### 18.19 `\xintCntoGC`

`\xintCntoGC{N}{\macro}` evaluates the  $c(j)=\macro{j}$  from  $j=0$  to  $j=N$  and returns a continued fraction written in inline format:  $\{c(0)\}+1/\{c(1)\}+1/\dots+1/\{c(N)\}$ . The parameter  $N$  is given to a `\numexpr`. The coefficients, after expansion, are, as shown, being enclosed in an added pair of braces, they may thus be fractions.

```
\def\macro #1{\the\numexpr\ifodd#1 -1-#1\else1+#1\fi\relax/%
\the\numexpr 1+#1*#1\relax}
\edef\x{\xintCntoGC {5}{\macro}}\texttt{\meaning\x}
\[\xintGCFrac{\xintCntoGC {5}{\macro}}\]
macro:->\{1/1\}+1/{-2/2}+1/{3/5}+1/{-4/10}+1/{5/17}+1/{-6/26}
```

$$1 + \frac{1}{\frac{-2}{2} + \frac{1}{\frac{3}{5} + \frac{1}{\frac{-4}{10} + \frac{1}{\frac{5}{17} + \frac{1}{\frac{-6}{26}}}}}}$$

### 18.20 `\xintGCntoGC`

`\xintGCntoGC{N}{\macroA}{\macroB}` evaluates the coefficients and then returns the corresponding  $\{a_0\}+\{b_0\}/\{a_1\}+\{b_1\}/\{a_2\}+\dots+\{b_{N-1}\}/\{a_N\}$  inline generalized fraction.  $N$  is given to a `\numexpr`. As shown, the coefficients are enclosed into added pairs of braces, and may thus be fractions.

```
\def\an #1{\the\numexpr #1*#1*#1+1\relax}%
\def\bn #1{\the\numexpr \xintiMON{#1}*(#1+1)\relax}%
\texttt{\xintGCntoGC {5}{\an}{\bn}}%
$\{=\xintGCFrac {\xintGCntoGC {5}{\an}{\bn}}
=\displaystyle\xintFrac {\xintGCntoF {5}{\an}{\bn}}\}$\par
```

$$1+1/2+-2/9+3/28+-4/65+5/126 = 1 + \frac{1}{2 - \frac{2}{9 + \frac{3}{28 - \frac{4}{65 + \frac{5}{126}}}}} = \frac{5797655}{3712466}$$



**18.21 `\xintiCstoF`, `\xintiGctoF`, `\xintiCstoCv`, `\xintiGctoCv`**

The same as the corresponding macros without the ‘i’, but for integer-only input. Infinitely faster; to notice the higher efficiency one would need to use them with an input having (at least) hundreds of coefficients.

**18.22 `\xintGctoGC`**

`\xintGctoGC{a+b/c+d/e+f/g+.....+v/w+x/y}` expands (with the usual meaning) each one of the coefficients and returns an inline continued fraction of the same type, each expanded coefficient being enclosed withing braces.

```
\edef\x {\xintGctoGC
  {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintFac {6}+\xintCstoF {2,-7,-5}/16}}
\texttt{\meaning\x}
```

macro: ->{1}+{3375/1[-3]}/{1/7}+{-3/5}/{720}+{67/36[0]}/{16}

To be honest I have, it seems, forgotten why I wrote this macro in the first place.

## 19 Package **xint** implementation

The commenting of the macros is currently (2013/05/07) very sparse. Some comments may be leftovers from previous versions of the macro, with parameters in another order for example.

### Contents

1	Catcodes, $\varepsilon$ -TeX and reload detection	66	21	\xintCmp	98
2	Package identification	68	22	\xintGeq	100
3	Token management macros	69	23	\xintMax	102
4	\xintRev, \xintReverseOrder	70	24	\xintMin	103
5	\xintRevWithBraces	71	25	\xintSum, \xintSumExpr	104
6	\xintLen, \xintLength	71	26	\xintMul	105
7	\xintCSVtoList	73	27	\xintSqr	114
8	\xintListWithSep	74	28	\xintPrd, \xintProductExpr	115
9	\xintNthElt	74	29	\xintFac	116
10	\xintApply	75	30	\xintPow	117
11	\xintAssign, \xintAssignArray, \xint-		31	\xintDivision, \xintQuo, \xintRem	121
	DigitsOf	76	32	\xintFDg	134
12	\XINT@RQ	78	33	\xintLDg	134
13	\XINT@cuz	79	34	\xintMON	134
14	\XINT@isOne	80	35	\xintOdd	135
15	\xintNum	81	36	\xintDSL	135
16	\xintSgn	81	37	\xintDSR	136
17	\xintOpp	82	38	\xintDSH, \xintDSHr	136
18	\xintAbs	82	39	\xintDSx	137
19	\xintAdd	91	40	\xintDecSplit, \xintDecSplitL, \xint-	
20	\xintSub	92		DecSplitR	140

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

The method for package identification and reload detection is copied verbatim from the packages by HEIKO OBERDIEK.

The method for catcodes was also inspired by these packages, we proceed slightly differently. 1.05 adds a `\relax` near the end of `\XINT@restorecatcodes@endinput`. Plain TeX users following the doc instruction to do `\input xint.sty\relax` were anyhow protected from any side effect. I didn't realize earlier that the `\endinput` would not have had the effect of stopping the scanning from the last `\the\catcode61`.

Starting with version 1.06 of the package, also ' must be sanitized, because we replace everywhere in the code the twice-expansion done with `\expandafter` by the systematic use of `\romannumeral-‘0`.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {

```

## 19 Package *xint* implementation

```

5  \catcode125=2    % }
6  \catcode64=11   % @
7  \catcode35=6    % #
8  \catcode44=12   % ,
9  \catcode45=12   % -
10 \catcode46=12   % .
11 \catcode58=12   % :
12 \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
13 \expandafter
14   \ifx\csname PackageInfo\endcsname\relax
15     \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
16   \else
17     \def\y#1#2{\PackageInfo{#1}{#2}}%
18   \fi
19 \expandafter
20 \ifx\csname numexpr\endcsname\relax
21   \y{xint}{\numexpr not available, aborting input}%
22   \aftergroup\endinput
23 \else
24   \ifx\x\relax % plain-TeX, first loading
25   \else
26     \def\empty {}%
27     \ifx\x\empty % LaTeX, first loading,
28       % variable is initialized, but \ProvidesPackage not yet seen
29     \else
30       \y{xint}{I was already loaded, aborting input}%
31       \aftergroup\endinput
32     \fi
33   \fi
34 \fi
35 \def\ChangeCatcodesIfInputNotAborted
36 {%
37   \endgroup
38   \edef\XINT@restorecatcodes@endinput
39   {%
40     \catcode96=\the\catcode96    % ‘
41     \catcode47=\the\catcode47    % /
42     \catcode41=\the\catcode41    % )
43     \catcode40=\the\catcode40    % (
44     \catcode42=\the\catcode42    % *
45     \catcode43=\the\catcode43    % +
46     \catcode62=\the\catcode62    % >
47     \catcode60=\the\catcode60    % <
48     \catcode58=\the\catcode58    % :
49     \catcode46=\the\catcode46    % .
50     \catcode45=\the\catcode45    % -
51     \catcode44=\the\catcode44    % ,
52     \catcode35=\the\catcode35    % #
53     \catcode64=\the\catcode64    % @

```

## 19 Package *xint* implementation

```
54     \catcode125=\the\catcode125 % }
55     \catcode123=\the\catcode123 % {
56     \endlinechar=\the\endlinechar
57     \catcode13=\the\catcode13 % ^^M
58     \catcode32=\the\catcode32 %
59     \catcode61=\the\catcode61\relax % =
60     \noexpand\endinput
61 }%
62 \def\XINT@setcatcodes
63 {%
64     \catcode61=12 % =
65     \catcode32=10 % space
66     \catcode13=5 % ^^M
67     \endlinechar=13 %
68     \catcode123=1 % {
69     \catcode125=2 % }
70     \catcode64=11 % @
71     \catcode35=6 % #
72     \catcode44=12 % ,
73     \catcode45=12 % -
74     \catcode46=12 % .
75     \catcode58=11 % : (made letter for error cs)
76     \catcode60=12 % <
77     \catcode62=12 % >
78     \catcode43=12 % +
79     \catcode42=12 % *
80     \catcode40=12 % (
81     \catcode41=12 % )
82     \catcode47=12 % /
83     \catcode96=12 % ‘
84 }%
85 \XINT@setcatcodes
86 }%
87 \ChangeCatcodesIfInputNotAborted
```

## 19.2 Package identification

Copied verbatim from HEIKO OBERDIEK's packages.

```
88 \begingroup
89 \catcode91=12 % [
90 \catcode93=12 % ]
91 \catcode58=12 % : (does not really matter, was letter)
92 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
93     \def\x#1#2#3[#4]{\endgroup
94         \immediate\write-1{Package: #3 #4}%
95         \xdef#1{#4}%
96     }%
97 \else
```

```

98   \def\x#1#2[#3]{\endgroup
99     #2[#3]}%
100   \ifx#1\@undefined
101     \xdef#1{#3}%
102   \fi
103   \ifx#1\relax
104     \xdef#1{#3}%
105   \fi
106 }%
107 \fi
108 \expandafter\x\csname ver@xint.sty\endcsname
109 \ProvidesPackage{xint}%
110 [2013/05/07 v1.06 Expandable operations on long numbers (jfb)]%

```

### 19.3 Token management macros

```

111 \def\xint@gobble      #1{}%
112 \def\xint@gobble@    {}%
113 \def\xint@gobble@i    #1{}%
114 \def\xint@gobble@ii   #1#2{}%
115 \def\xint@gobble@iii  #1#2#3{}%
116 \def\xint@gobble@iv   #1#2#3#4{}%
117 \def\xint@gobble@v    #1#2#3#4#5{}%
118 \def\xint@gobble@vi   #1#2#3#4#5#6{}%
119 \def\xint@gobble@vii  #1#2#3#4#5#6#7{}%
120 \def\xint@gobble@viii #1#2#3#4#5#6#7#8{}%
121 \def\xint@firstoftwo  #1#2{#1}%
122 \def\xint@secondoftwo #1#2{#2}%
123 \def\xint@firstoftwo@andstop #1#2{ #1}%
124 \def\xint@secondoftwo@andstop #1#2{ #2}%
125 \def\xint@exchangetwo@keepbraces@andstop #1#2{ {#2}{#1}}%
126 \def\xint@minus@andstop { -}%
127 \def\xint@r          #1\R {}%
128 \def\xint@w          #1\W {}%
129 \def\xint@z          #1\Z {}%
130 \def\xint@zero       #10{}%
131 \def\xint@one        #11{}%
132 \def\xint@minus      #1-{}%
133 \def\xint@relax      #1\relax {}%
134 \def\xint@quatrezeros #10000{}%
135 \def\xint@braced@undef {\xint@undef}%
136 \def\xint@UDzerofork  #10\dummy #2#3\xint@UDkrof {#2}%
137 \def\xint@UDsignfork  #1-\dummy #2#3\xint@UDkrof {#2}%
138 \def\xint@UDwfork     #1\W\dummy #2#3\xint@UDkrof {#2}%
139 \def\xint@UDzerosfork  #100\dummy #2#3\xint@UDkrof {#2}%
140 \def\xint@UDdonezerofork #110\dummy #2#3\xint@UDkrof {#2}%
141 \def\xint@UDzerominusfork #10-\dummy #2#3\xint@UDkrof {#2}%
142 \def\xint@UDsignsfork  #1--\dummy #2#3\xint@UDkrof {#2}%
143 \def\xint@afterfi     #1#2\fi {\fi #1}%

```

**19.4 \xintRev, \xintReverseOrder**

\xintRev: fait l'expansion avec \romannumeral-‘0, vérifie le signe  
\xintReverseOrder: ne fait PAS l'expansion, ne regarde PAS le signe.

```

144 \def\xintRev {\romannumeral0\xintrev}%
145 \def\xintrev #1%
146 {%
147   \expandafter\xint@rev\expandafter {\romannumeral-‘0#1}%
148}%
149 \def\xint@rev #1%
150 {%
151   \XINT@rev@fork #1\Z
152}%
153 \def\XINT@rev@fork #1#2%
154 {%
155   \xint@UDsignfork
156   #1\dummy \XINT@rev@negative
157   -\dummy \XINT@rev@nonnegative
158   \xint@UDkrof
159   #1#2%
160}%
161 \def\XINT@rev@negative #1#2\Z
162 {%
163   \expandafter\xint@minus@andstop\romannumeral0\XINT@rev {#2}%
164}%
165 \def\XINT@rev@nonnegative #1\Z
166 {%
167   \XINT@rev {#1}%
168}%
169 \def\XINT@Rev {\romannumeral0\XINT@rev}%
170 \let\xintReverseOrder \XINT@Rev
171 \def\XINT@rev #1%
172 {%
173   \XINT@rord@main {}#1%
174   \xint@UNDEF
175   \xint@undef\xint@undef\xint@undef\xint@undef
176   \xint@undef\xint@undef\xint@undef\xint@undef
177   \xint@UNDEF
178}%
179 \def\XINT@rord@main #1#2#3#4#5#6#7#8#9%
180 {%
181   \XINT@strip@undef #9\XINT@rord@cleanup\xint@undef
182   \XINT@rord@main {#9#8#7#6#5#4#3#2#1}%
183}%
184 \def\XINT@rord@cleanup\xint@undef\XINT@rord@main #1#2\xint@UNDEF
185 {%
186   \expandafter\space\XINT@strip@UNDEF #1%
187}%
188 \def\XINT@strip@undef #1\xint@undef {}%
```

```
189 \def\XINT@strip@UNDEF #1\xint@UNDEF {}%
```

### 19.5 \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.)

```
190 \def\xintRevWithBraces {\romannumeral0\xintrevwithbraces }%
191 \def\xintrevwithbraces #1%
192 {%
193   \expandafter\XINT@revwbr@prep\expandafter {\romannumeral-‘0#1}%
194 }%
195 \def\XINT@RWB {\romannumeral0\XINT@revwbr@prep }%
196 \def\XINT@revwbr@prep #1%
197 {%
198   \XINT@revwbr@loop
199   {}#1\xint@undef\xint@undef\xint@undef\xint@undef
200     \xint@undef\xint@undef\xint@undef\xint@undef\Z
201 }%
202 \def\XINT@revwbr@loop #1#2#3#4#5#6#7#8#9%
203 {%
204   \XINT@strip@undef #9\XINT@revwbr@finish@a\xint@undef
205   \XINT@revwbr@loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}%
206 }%
207 \def\XINT@revwbr@finish@a\xint@undef\XINT@revwbr@loop #1#2\Z
208 {%
209   \XINT@revwbr@finish@b #2\R\R\R\R\R\R\R\Z #1%
210 }%
211 \def\XINT@revwbr@finish@b #1#2#3#4#5#6#7#8\Z
212 {%
213   \xint@r #1\XINT@revwbr@finish@c 8%
214           #2\XINT@revwbr@finish@c 7%
215           #3\XINT@revwbr@finish@c 6%
216           #4\XINT@revwbr@finish@c 5%
217           #5\XINT@revwbr@finish@c 4%
218           #6\XINT@revwbr@finish@c 3%
219           #7\XINT@revwbr@finish@c 2%
220           \R\XINT@revwbr@finish@c 1\Z
221 }%
222 \def\XINT@revwbr@finish@c #1#2\Z
223 {%
224   \expandafter\expandafter\expandafter
225     \space
226   \csname xint@gobble@\romannumeral #1\endcsname
227 }%
```

### 19.6 \xintLen, \xintLength



## 19 Package *xint* implementation

```

\xintLen    -> fait l'expansion, ne compte PAS le signe
\xintLength -> ne fait PAS l'expansion, compte le signe
1.06: improved code is roughly 20% faster than the one from earlier versions.

228 \def\xintilen {\romannumeral0\xintilen }%
229 \def\xintilen #1%
230 {%
231     \expandafter\XINT@len@prep\expandafter {\romannumeral-'0#1}%
232 }%
233 \let\xintLen\xintilen \let\xintlen\xintilen
234 \def\XINT@Len {\romannumeral0\XINT@len@prep }%
235 \def\XINT@len@prep #1%
236 {%
237     \XINT@length@fork
238     #1\xint@undef\xint@undef\xint@undef\xint@undef
239     \xint@undef\xint@undef\xint@undef\xint@undefZ
240 }%
241 \def\XINT@length@fork #1%
242 {%
243     \expandafter\XINT@length@loop
244     \xint@UDsignfork
245     #1\dummy {{0}}%
246     -\dummy {{0}}#1}%
247     \xint@UDkrof
248 }%
249 \def\XINT@Length {\romannumeral0\XINT@length }%
250 \def\XINT@length #1%
251 {%
252     \XINT@length@loop
253     {0}#1\xint@undef\xint@undef\xint@undef\xint@undef
254     \xint@undef\xint@undef\xint@undef\xint@undefZ
255 }%
256 \let\xintLength\XINT@Length
257 \def\XINT@length@loop #1#2#3#4#5#6#7#8#9%
258 {%
259     \XINT@strip@undef #9\XINT@length@finish@a\xint@undef
260     \expandafter\XINT@length@loop\expandafter {\the\numexpr #1+8\relax}%
261 }%
262 \def\XINT@length@finish@a\xint@undef
263     \expandafter\XINT@length@loop\expandafter #1#2\Z
264 {%
265     \XINT@length@finish@b #2\W\W\W\W\W\W\W\Z {#1}%
266 }%
267 \def\XINT@length@finish@b #1#2#3#4#5#6#7#8\Z
268 {%
269     \xint@w #1\XINT@length@finish@c 8%
270     #2\XINT@length@finish@c 7%
271     #3\XINT@length@finish@c 6%
272     #4\XINT@length@finish@c 5%
273     #5\XINT@length@finish@c 4%

```

```

274      #6\XINT@length@finish@c 3%
275      #7\XINT@length@finish@c 2%
276      \W\XINT@length@finish@c 1\Z
277 }%
278 \def\XINT@length@finish@c #1#2\Z #3{\expandafter\space\the\numexpr #3-#1\relax}%

```

### 19.7 \xintCSVtoList

\xintCSVtoList {a,b,...,z} returns {a}{b}...{z}. The comma separated list may be a macro which is first expanded. Each chain of spaces is collapsed into one space only.

First included in release 1.06.

```

279 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
280 \def\xintcsvtolist #1%
281 {%
282   \expandafter\XINT@csvtol@prep\expandafter {\romannumeral-‘0#1}%
283 }%
284 \def\XINT@CSVtoL {\romannumeral0\XINT@csvtol@prep }%
285 \def\XINT@csvtol@prep #1%
286 {%
287   \XINT@csvtol@loop@a
288   {}#1,\xint@undef,\xint@undef,\xint@undef,\xint@undef,%
289   \xint@undef,\xint@undef,\xint@undef,\xint@undef,\Z
290 }%
291 \def\XINT@csvtol@loop@a #1#2,#3,#4,#5,#6,#7,#8,#9,%
292 {%
293   \XINT@strip@undef #9\XINT@csvtol@finish@a\xint@undef
294   \XINT@csvtol@loop@b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
295 }%
296 \def\XINT@csvtol@loop@b #1#2{\XINT@csvtol@loop@a {#1#2}}%
297 \def\XINT@csvtol@finish@a\xint@undef\XINT@csvtol@loop@b #1#2#3\Z
298 {%
299   \XINT@csvtol@finish@b #3\R,\R,\R,\R,\R,\R,\R,\Z #2{#1}%
300 }%
301 \def\XINT@csvtol@finish@b #1,#2,#3,#4,#5,#6,#7,#8\Z
302 {%
303   \xint@r #1\XINT@csvtol@finish@c 8%
304   #2\XINT@csvtol@finish@c 7%
305   #3\XINT@csvtol@finish@c 6%
306   #4\XINT@csvtol@finish@c 5%
307   #5\XINT@csvtol@finish@c 4%
308   #6\XINT@csvtol@finish@c 3%
309   #7\XINT@csvtol@finish@c 2%
310   \R\XINT@csvtol@finish@c 1\Z
311 }%
312 \def\XINT@csvtol@finish@c #1#2\Z
313 {%
314   \csname XINT@csvtol@finish@d\romannumeral #1\endcsname

```

```

315 }%
316 \def\XINT@csvtol@finish@dvi  #1#2#3#4#5#6#7#8#9{ #9}%
317 \def\XINT@csvtol@finish@dvi  #1#2#3#4#5#6#7#8#9{ #9{#1}}%
318 \def\XINT@csvtol@finish@dvi  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
319 \def\XINT@csvtol@finish@dv   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
320 \def\XINT@csvtol@finish@div  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
321 \def\XINT@csvtol@finish@diii #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
322 \def\XINT@csvtol@finish@dii  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}{#6}}%
323 \def\XINT@csvtol@finish@di   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%

```

## 19.8 \xintListWithSep

\xintListWithSep {sep}{a}{b}...{z} returns a sep b sep .... sep z

Included in release 1.04. The 'sep' can be \par's; the macro

xintlistwithsep etc... are all declared long. 'sep' does not have to be a single token. The list may be a macro it is first expanded.

1.06 modifies the 'feature' of returning sep if the list is empty: the output is now empty in that case. (sep was not used for a one element list, but strangely it was for a zero-element list).

```

324 \def\xintListWithSep {\romannumeral0\xintlistwithsep }%
325 \long\def\xintlistwithsep #1#2%
326 {%
327   \expandafter\XINT@lws\expandafter {\romannumeral-0#2}%
328   {#1}%
329 }%
330 \long\def\XINT@LWS #1#2{\romannumeral0\XINT@lws@start {#1}#2\Z }%
331 \long\def\XINT@lws #1#2%
332 {%
333   \XINT@lws@start {#2}#1\Z
334 }%
335 \long\def\XINT@lws@start #1#2%
336 {%
337   \xint@z #2\XINT@lws@dont\Z
338   \XINT@lws@loop@a {#2}{#1}%
339 }%
340 \long\def\XINT@lws@dont\Z\XINT@lws@loop@a #1#2{ }%
341 \long\def\XINT@lws@loop@a #1#2#3%
342 {%
343   \xint@z #3\XINT@lws@end\Z
344   \XINT@lws@loop@b {#1}{#2#3}{#2}%
345 }%
346 \long\def\XINT@lws@loop@b #1#2{\XINT@lws@loop@a {#1#2}}%
347 \long\def\XINT@lws@end\Z\XINT@lws@loop@b #1#2#3{ #1}%

```

## 19.9 \xintNthElt

\xintNthElt {a}{b}...{z} (or 'tokens' abcd...z) returns the Nth one (one pair of braces removed). The list is first expanded.

## 19 Package *xint* implementation

First included in release 1.06.

```
348 \def\xintNthElt {\romannumeral0\xintnthelt }%
349 \def\xintnthelt #1#2%
350 {%
351   \expandafter\XINT@nthelt\expandafter {\romannumeral-‘0#2}%
352   {\numexpr #1\relax}%
353 }%
354 \def\XINT@NthElt #1#2{\romannumeral0\XINT@nthelt {#2}{\numexpr #1\relax}}%
355 \def\XINT@nthelt #1#2%
356 {%
357   \ifnum #2<1
358     \expandafter\XINT@nthelt@silentend
359   \else
360     \expandafter\XINT@nthelt@loop@a
361   \fi {#2}#1\xint@undef\xint@undef\xint@undef\xint@undef
362     \xint@undef\xint@undef\xint@undef\xint@undef\Z
363 }%
364 \def\XINT@nthelt@silentend #1\Z { }%
365 \def\XINT@nthelt@loop@a #1%
366 {%
367   \ifnum #1>8
368     \expandafter\XINT@nthelt@loop@b
369   \else
370     \expandafter\XINT@nthelt@getit
371   \fi
372   {#1}%
373 }%
374 \def\XINT@nthelt@loop@b #1#2#3#4#5#6#7#8#9%
375 {%
376   \XINT@strip@undef #9\XINT@nthelt@silentend\xint@undef
377   \expandafter\XINT@nthelt@loop@a\expandafter{\the\numexpr #1-8\relax}%
378 }%
379 \def\XINT@nthelt@getit #1%
380 {%
381   \expandafter\expandafter\expandafter\XINT@nthelt@finish
382   \csname xint@gobble@\romannumeral\numexpr#1-1\endcsname
383 }%
384 \def\XINT@nthelt@finish #1#2\Z
385 {%
386   \xint@UDwfork
387   #1\dummy { }%
388   \W\dummy { #1}%
389   \xint@UDkrof
390 }%
```

### 19.10 \xintApply

`\xintApply {\macro}{\{a\}{b}...\{z\}}` returns `{\macro{a}}...\{\macro{b}}`

where each instance of `\macro` is expanded. The list is first expanded. Introduced with release 1.04.

```

391 \def\xintApply {\romannumeral0\xintapply}%
392 \def\xintapply #1#2%
393 {%
394   \expandafter\XINT@apply\expandafter {\romannumeral-‘0#2}%
395   {#1}%
396}%
397 \def\XINT@Apply #1#2{\romannumeral0\XINT@apply@loop@a {}{#1}#2\Z}%
398 \def\XINT@apply #1#2%
399 {%
400   \XINT@apply@loop@a {}{#2}#1\Z
401}%
402 \def\XINT@apply@loop@a #1#2#3%
403 {%
404   \xint@z #3\XINT@apply@end\Z
405   \expandafter
406   \XINT@apply@loop@b
407   \expandafter {\romannumeral-‘0#2{#3}}{#1}{#2}%
408}%
409 \def\XINT@apply@loop@b #1#2{\XINT@apply@loop@a {#2{#1}}}%
410 \def\XINT@apply@end\Z\expandafter\XINT@apply@loop@b\expandafter #1#2#3{ #2}%

```

### 19.11 `\xintAssign`, `\xintAssignArray`, `\xintDigitsOf`

`\xintAssign {a}{b}..{z}\to A\B...Z`,  
`\xintAssignArray {a}{b}..{z}\to U`  
 version 1.01 corrects an oversight in 1.0 related to the value of  
`\escapechar` at the time of using `\xintAssignArray` or `\xintRelaxArray`  
 These macros are an exception in the *xint* bundle, they do not care at  
 all about compatibility with expansion-only contexts.  
 In version 1.05a I suddenly see some incongruous `\expandafter`'s in  
`\XINT@assignarray@@@end`, which I remove.  
 Release 1.06 modifies the macros created by `\xintAssignArray` to feed their  
 argument to a `\numexpr`

```

411 \def\xintAssign #1\to
412 {%
413   \expandafter\XINT@assign@a\romannumeral-‘0#1{}\to
414}%
415 \def\XINT@assign@a #1% attention to the # at the beginning of next line
416 #1%
417   \def\xint@temp {#1}%
418   \ifx\empty\xint@temp
419     \expandafter\XINT@assign@b
420   \else
421     \expandafter\XINT@assign@B
422   \fi
423}%

```

```

424 \def\XINT@assign@b #1#2\to #3%
425 {%
426   \edef #3{#1}\def\xint@temp {#2}%
427   \ifx\empty\xint@temp
428     \else
429     \xint@afterfi{\XINT@assign@a #2\to }%
430   \fi
431 }%
432 \def\XINT@assign@B #1\to #2%
433 {%
434   \edef #2{\xint@temp}%
435 }%
436 \def\xintRelaxArray #1%
437 {%
438   \edef\XINT@restoreescapechar {\escapechar\the\escapechar\relax}%
439   \escapechar -1
440   \edef\xint@arrayname {\string #1}%
441   \XINT@restoreescapechar
442   \expandafter\let\expandafter\xint@temp
443     \csname\xint@arrayname 0\endcsname
444   \count 255 0
445   \loop
446     \global\expandafter\let
447       \csname\xint@arrayname\the\count255\endcsname\relax
448     \ifnum \count 255 < \xint@temp
449       \advance\count 255 1
450     \repeat
451   \global\expandafter\let\csname\xint@arrayname 00\endcsname\relax
452   \global\let #1\relax
453 }%
454 \def\xintAssignArray #1\to #2%
455 {%
456   \edef\XINT@restoreescapechar {\escapechar\the\escapechar\relax}%
457   \escapechar -1
458   \edef\xint@arrayname {\string #1}%
459   \XINT@restoreescapechar
460   \count 255 0
461   \expandafter
462   \XINT@assignarray@loop \romannumeral-'0#1\xint@undef
463   \csname\xint@arrayname 00\endcsname
464   \csname\xint@arrayname 0\endcsname
465   {\xint@arrayname}%
466   #2%
467 }%
468 \def\XINT@assignarray@loop #1%
469 {%
470   \def\xint@temp {#1}%
471   \ifx\xint@bracedundef\xint@temp
472     \edef\xint@temp{\the\count 255 }%

```

## 19 Package *xint* implementation

```

473     \expandafter\let\csname\xint@arrayname0\endcsname\xint@temp
474     \expandafter\XINT@assignarray@end
475     \else
476         \advance\count 255 1
477         \expandafter\edef
478             \csname\xint@arrayname\the\count 255\endcsname{\xint@temp}%
479         \expandafter\XINT@assignarray@loop
480     \fi
481 }%
482 \def\XINT@assignarray@end {\expandafter\XINT@assignarray@@end }%
483 \def\XINT@assignarray@@end #1%
484 {%
485     \expandafter\XINT@assignarray@@@end\expandafter #1%
486 }%
487 \def\XINT@assignarray@@@end #1#2#3%
488 {%
489     \expandafter\XINT@assignarray@@@@end
490     \expandafter #1\expandafter #2\expandafter{#3}%
491 }%
492 \def\XINT@assignarray@@@@end #1#2#3#4%
493 {%
494     \def #4##1%
495     {%
496         \romannumeral0%
497         \expandafter #1\expandafter{\the\numexpr ##1}%
498     }%
499     \def #1##1%
500     {%
501         \ifnum ##1< 0
502             \xint@afterfi {\xintError:ArrayIndexIsNegative\space 0}%
503         \else
504             \xint@afterfi {%
505                 \ifnum ##1> #2
506                     \xint@afterfi {\xintError:ArrayIndexBeyondLimit\space 0}%
507                 \else
508                     \xint@afterfi
509                     {\expandafter\expandafter\expandafter
510                     \space\csname #3##1\endcsname}%
511                 \fi}%
512         \fi
513     }%
514 }%
515 \let\xintDigitsOf\xintAssignArray

```

### 19.12 \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\Z`



## 19 Package **xint** implementation

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

```
516 \def\XINT@RQ #1#2#3#4#5#6#7#8#9%
517 {%
518   \xint@r #9\XINT@RQ@end\R\XINT@RQ {#9#8#7#6#5#4#3#2#1}%
519 }%
520 \def\XINT@RQ@end\R\XINT@RQ #1#2\Z
521 {%
522   \XINT@RQ@end@ #1\Z
523 }%
524 \def\XINT@RQ@end@ #1#2#3#4#5#6#7#8%
525 {%
526   \xint@r #8\XINT@RQ@end@viii
527           #7\XINT@RQ@end@vii
528           #6\XINT@RQ@end@vi
529           #5\XINT@RQ@end@v
530           #4\XINT@RQ@end@iv
531           #3\XINT@RQ@end@iii
532           #2\XINT@RQ@end@ii
533           \R\XINT@RQ@end@i
534           \Z #2#3#4#5#6#7#8%
535 }%
536 \def\XINT@RQ@end@viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
537 \def\XINT@RQ@end@vii #1\Z #2#3#4#5#6#7#8#9\Z { #8#9000}%
538 \def\XINT@RQ@end@vi #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#900}%
539 \def\XINT@RQ@end@v #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90}%
540 \def\XINT@RQ@end@iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#9}%
541 \def\XINT@RQ@end@iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
542 \def\XINT@RQ@end@ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
543 \def\XINT@RQ@end@i \Z #1#2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

### 19.13 \XINT@cuz

```
544 \def\xint@cleanupzeros@andstop #1#2#3#4%
545 {%
546   \expandafter\space\the\numexpr #1#2#3#4\relax
547 }%
548 \def\xint@cleanupzeros@nospace #1#2#3#4%
549 {%
550   \the\numexpr #1#2#3#4\relax
551 }%
552 \def\XINT@rev@andcuz #1%
553 {%
554   \expandafter\xint@cleanupzeros@andstop
555   \romannumeral0\XINT@rord@main {}#1%
556   \xint@UNDEF
557   \xint@undef\xint@undef\xint@undef\xint@undef

```

## 19 Package *xint* implementation

```
558      \xint@undef\xint@undef\xint@undef\xint@undef
559      \xint@UNDEF
560 }%
    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

561 \def\xINT@cuz #1%
562 {%
563     \XINT@cuz@loop #1\W\W\W\W\W\W\W\Z%
564 }%
565 \def\xINT@cuz@loop #1#2#3#4#5#6#7#8%
566 {%
567     \xint@w #8\xint@cuz@enda\W
568     \xint@z #8\xint@cuz@endb\Z
569     \XINT@cuz@checka {#1#2#3#4#5#6#7#8}%
570 }%
571 \def\xint@cuz@enda #1\xINT@cuz@checka #2%
572 {%
573     \xint@cuz@endaa #2%
574 }%
575 \def\xint@cuz@endaa #1#2#3#4#5\Z
576 {%
577     \expandafter\space\the\numexpr #1#2#3#4\relax
578 }%
579 \def\xint@cuz@endb\Z\xINT@cuz@checka #1{ 0}%
580 \def\xINT@cuz@checka #1%
581 {%
582     \expandafter \XINT@cuz@checkb \the\numexpr #1\relax
583 }%
584 \def\xINT@cuz@checkb #1%
585 {%
586     \xint@zero #1\xint@cuz@backto loop 0\xINT@cuz@Stop #1%
587 }%
588 \def\xINT@cuz@Stop #1\W #2\Z{ #1}%
589 \def\xint@cuz@backto loop 0\xINT@cuz@Stop 0{\XINT@cuz@loop }%
```

### 19.14 \XINT@isOne

Added in 1.03. Attention: does not do any expansion.

```
590 \def\xINT@isOne #1{\romannumeral0\xINT@isone #1\W\Z}%
591 \def\xINT@isone #1#2%
592 {%
593     \xint@one #1\xINT@isone@b 1\expandafter\space\expandafter 0\xint@z #2%
594 }%
595 \def\xINT@isone@b #1\xint@z #2%
```

```

596 {%
597   \xint@w #2\XINT@isone@yes\W\expandafter\space\expandafter 0\xint@z
598 }%
599 \def\XINT@isone@yes #1\Z{ 1}%

```

### 19.15 \xintNum

For example `\xintNum {-----+-----0000000000000003}`  
 1.05 defines `\xintiNum`, which allows redefinition of `\xintNum` by `xintfrac.sty`

```

600 \def\xintiNum {\romannumeral0\xintinum }%
601 \def\xintinum #1%
602 {%
603   \expandafter\XINT@num\expandafter {\romannumeral-‘0#1}%
604 }%
605 \let\xintNum\xintiNum \let\xintnum\xintinum
606 \def\XINT@Num {\romannumeral0\XINT@num }%
607 \def\XINT@num #1{\XINT@num@loop #1\R\R\R\R\R\R\R\Z }%
608 \def\XINT@num@loop #1#2#3#4#5#6#7#8%
609 {%
610   \xint@r #8\XINT@num@end\R\XINT@num@NumEight #1#2#3#4#5#6#7#8%
611 }%
612 \def\XINT@num@end\R\XINT@num@NumEight #1\R #2\Z
613 {%
614   \expandafter\space\the\numexpr #1+0\relax
615 }%
616 \def\XINT@num@NumEight #1#2#3#4#5#6#7#8%
617 {%
618   \ifnum \numexpr #1#2#3#4#5#6#7#8+0\relax = 0
619     \xint@afterfi {\expandafter\XINT@num@keepsign@a
620                   \the\numexpr #1#2#3#4#5#6#7#81\relax}%
621   \else
622     \xint@afterfi {\expandafter\XINT@num@finish
623                   \the\numexpr #1#2#3#4#5#6#7#8\relax}%
624   \fi
625 }%
626 \def\XINT@num@keepsign@a #1%
627 {%
628   \xint@one#1\XINT@num@gobackto loop 1\XINT@num@keepsign@b
629 }%
630 \def\XINT@num@gobackto loop 1\XINT@num@keepsign@b {\XINT@num@loop }%
631 \def\XINT@num@keepsign@b #1{\XINT@num@loop -}%
632 \def\XINT@num@finish #1\R #2\Z { #1}%

```

### 19.16 \xintSgn

Changed in 1.05. Earlier code was unnecessarily strange.

```

633 \def\xintiSgn {\romannumeral0\xintisgn }%
634 \def\xintisgn #1%
635 {%
636   \expandafter\XINT@sgn \romannumeral-‘0#1\Z%
637 }%
638 \let\xintSgn\xintiSgn \let\xintsgn\xintisgn
639 \def\XINT@Sgn #1{\romannumeral0\XINT@sgn #1\Z }%
640 \def\XINT@sgn #1#2\Z
641 {%
642   \xint@UDzerominusfork
643   #1-\dummy { 0}%
644   0#1\dummy { -1}%
645   0-\dummy { 1}%
646   \xint@UDkrof
647 }%

```

### 19.17 \xintOpp

```

648 \def\xintiOpp {\romannumeral0\xintiopp }%
649 \def\xintiopp #1%
650 {%
651   \expandafter\XINT@opp \romannumeral-‘0#1%
652 }%
653 \let\xintOpp\xintiOpp \let\xintopp\xintiopp
654 \def\XINT@Opp #1{\romannumeral0\XINT@opp #1}%
655 \def\XINT@opp #1%
656 {%
657   \xint@UDzerominusfork
658   #1-\dummy { 0}%      zero
659   0#1\dummy { }%      negative
660   0-\dummy { -#1}%    positive
661   \xint@UDkrof
662 }%

```

### 19.18 \xintAbs

```

663 \def\xintiAbs {\romannumeral0\xintiabs }%
664 \def\xintiabs #1%
665 {%
666   \expandafter\XINT@abs \romannumeral-‘0#1%
667 }%
668 \let\xintAbs\xintiAbs \let\xintabs\xintiabs
669 \def\XINT@Abs #1{\romannumeral0\XINT@abs #1}%
670 \def\XINT@abs #1%
671 {%
672   \xint@UDsignfork
673   #1\dummy { }%
674   -\dummy { #1}%
675   \xint@UDkrof
676 }%

```

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

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

#### ADDITION

I: \XINT@add@A

INPUT:

\romannumeral0\XINT@add@A 0{ }<N1>\W\X\Y\Z <N2>\W\X\Y\Z

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

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

OUTPUT: la somme <N1>+<N2>, order 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.

677 \def\XINT@add@A #1#2#3#4#5#6%

678 {%

679 \xint@w #3\xint@add@az\W\XINT@add@AB #1{#3#4#5#6}{#2}%

680 }%

681 \def\xint@add@az\W\XINT@add@AB #1#2%

682 {%

683 \XINT@add@AC@checkcarry #1%

684 }%

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.

685 \def\XINT@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%

686 {%

687 \xint@w #5\xint@add@bz\W

688 \XINT@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z

689 }%

690 \def\XINT@add@ABE #1#2#3#4#5#6%

691 {%

## 19 Package *xint* implementation

```

692 \expandafter\XINT@add@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax.%
693 }%
694 \def\XINT@add@ABEA #1#2#3.#4%
695 {%
696 \XINT@add@A #2{#3#4}%
697 }%

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

698 \def\xint@add@bz\W\XINT@add@ABE #1#2#3#4#5#6%
699 {%
700 \expandafter\XINT@add@CC\the\numexpr #1+10#5#4#3#2\relax.%
701 }%
702 \def\XINT@add@CC #1#2#3.#4%
703 {%
704 \XINT@add@AC@checkcarry #2{#3#4}% on va examiner et \'eliminer #2
705 }%

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

706 \def\XINT@add@AC@checkcarry #1%
707 {%
708 \xint@zero #1\xint@add@AC@nocarry 0\XINT@add@C
709 }%
710 \def\xint@add@AC@nocarry 0\XINT@add@C #1#2\W\X\Y\Z
711 {%
712 \expandafter
713 \xint@cleanupzeros@andstop
714 \romannumeral0%
715 \XINT@rord@main {}#2%
716 \xint@UNDEF
717 \xint@undef\xint@undef\xint@undef\xint@undef
718 \xint@undef\xint@undef\xint@undef\xint@undef
719 \xint@UNDEF
720 #1%
721 }%
722 \def\XINT@add@C #1#2#3#4#5%
723 {%
724 \xint@w #2\xint@add@cz\W\XINT@add@CD {#5#4#3#2}{#1}%q
725 }%
726 \def\XINT@add@CD #1%
727 {%
728 \expandafter\XINT@add@CC\the\numexpr 1+10#1\relax.%
729 }%
730 \def\xint@add@cz\W\XINT@add@CD #1#2{ 1#2}%

```

## 19 Package *xint* implementation

Addition II: \XINT@addr@A.

INPUT:

\romannumeral0\XINT@addr@A 0{ }<N1>\W\X\Y\Z <N2>\W\X\Y\Z

Comme \XINT@addr@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@addr@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\*

```

731 \def\XINT@addr@A #1#2#3#4#5#6%
732 {%
733   \xint@w #3\xint@addr@az\W\XINT@addr@B #1{#3#4#5#6}{#2}%
734 }%
735 \def\xint@addr@az\W\XINT@addr@B #1#2%
736 {%
737   \XINT@addr@AC@checkcarry #1%
738 }%
739 \def\XINT@addr@B #1#2#3#4\W\X\Y\Z #5#6#7#8%
740 {%
741   \xint@w #5\xint@addr@bz\W\XINT@addr@E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
742 }%
743 \def\XINT@addr@E #1#2#3#4#5#6%
744 {%
745   \expandafter\XINT@addr@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
746 }%
747 \def\XINT@addr@ABEA #1#2#3#4#5#6#7%
748 {%
749   \XINT@addr@A #2{#7#6#5#4#3}%
750 }%
751 \def\xint@addr@bz\W\XINT@addr@E #1#2#3#4#5#6%
752 {%
753   \expandafter\XINT@addr@CC\the\numexpr #1+10#5#4#3#2\relax
754 }%
755 \def\XINT@addr@CC #1#2#3#4#5#6#7%
756 {%
757   \XINT@addr@AC@checkcarry #2{#7#6#5#4#3}%
758 }%
759 \def\XINT@addr@AC@checkcarry #1%
760 {%
761   \xint@zero #1\xint@addr@AC@nocarry 0\XINT@addr@C
762 }%
763 \def\xint@addr@AC@nocarry 0\XINT@addr@C #1#2\W\X\Y\Z { #1#2}%
764 \def\XINT@addr@C #1#2#3#4#5%

```



## 19 Package *xint* implementation

```

765 {%
766   \xint@w #2\xint@addr@cz\W\XINT@addr@D {#5#4#3#2}{#1}%
767 }%
768 \def\XINT@addr@D #1%
769 {%
770   \expandafter\XINT@addr@CC\the\numexpr 1+10#1\relax
771 }%
772 \def\xint@addr@cz\W\XINT@addr@D #1#2{ #21000}%

  ADDITION III, \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.

773 \def\XINT@addm@A #1#2#3#4#5#6%
774 {%
775   \xint@w #3\xint@addm@az\W\XINT@addm@AB #1{#3#4#5#6}{#2}%
776 }%
777 \def\xint@addm@az\W\XINT@addm@AB #1#2%
778 {%
779   \XINT@addm@AC@checkcarry #1%
780 }%
781 \def\XINT@addm@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
782 {%
783   \XINT@addm@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
784 }%
785 \def\XINT@addm@ABE #1#2#3#4#5#6%
786 {%
787   \expandafter\XINT@addm@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax.%
788 }%
789 \def\XINT@addm@ABEA #1#2#3.#4%
790 {%
791   \XINT@addm@A #2{#3#4}%
792 }%
793 \def\XINT@addm@AC@checkcarry #1%
794 {%
795   \xint@zero #1\xint@addm@AC@nocarry 0\XINT@addm@C
796 }%
797 \def\xint@addm@AC@nocarry 0\XINT@addm@C #1#2\W\X\Y\Z
798 {%
799   \expandafter
800   \xint@cleanupzeros@andstop
801   \romannumeral0%
802   \XINT@rord@main {}#2%
803   \xint@UNDEF
804   \xint@undef\xint@undef\xint@undef\xint@undef

```

## 19 Package *xint* implementation

```

805      \xint@undef\xint@undef\xint@undef\xint@undef
806      \xint@UNDEF
807      #1%
808 }%
809 \def\xint@addm@C #1#2#3#4#5%
810 {%
811     \xint@w
812     #5\xint@addm@cw
813     #4\xint@addm@cx
814     #3\xint@addm@cy
815     #2\xint@addm@cz
816     \W\xint@addm@CD {#5#4#3#2}{#1}%
817 }%
818 \def\xint@addm@CD #1%
819 {%
820     \expandafter\xint@addm@CC\the\numexpr 1+10#1\relax.%
821 }%
822 \def\xint@addm@CC #1#2#3.#4%
823 {%
824     \xint@addm@AC@checkcarry #2{#3#4}%
825 }%
826 \def\xint@addm@cw
827     #1\xint@addm@cx
828     #2\xint@addm@cy
829     #3\xint@addm@cz
830     \W\xint@addm@CD
831 {%
832     \expandafter\xint@addm@CDw\the\numexpr 1+#1#2#3\relax.%
833 }%
834 \def\xint@addm@CDw #1.#2#3\X\Y\Z
835 {%
836     \xint@addm@end #1#3%
837 }%
838 \def\xint@addm@cx
839     #1\xint@addm@cy
840     #2\xint@addm@cz
841     \W\xint@addm@CD
842 {%
843     \expandafter\xint@addm@CDx\the\numexpr 1+#1#2\relax.%
844 }%
845 \def\xint@addm@CDx #1.#2#3\Y\Z
846 {%
847     \xint@addm@end #1#3%
848 }%
849 \def\xint@addm@cy
850     #1\xint@addm@cz
851     \W\xint@addm@CD
852 {%
853     \expandafter\xint@addm@CDy\the\numexpr 1+#1\relax.%

```

## 19 Package *xint* implementation

```

854 }%
855 \def\XINT@addm@CDy #1.#2#3\Z
856 {%
857   \XINT@addm@end #1#3%
858 }%
859 \def\xint@addm@cz\W\XINT@addm@CD #1#2#3{\XINT@addm@end #1#3}%
860 \def\XINT@addm@end #1#2#3#4#5%
861   {\expandafter\space\the\numexpr #1#2#3#4#5\relax}%

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

862 \def\XINT@addp@A #1#2#3#4#5#6%
863 {%
864   \xint@w #3\xint@addp@az\W\XINT@addp@AB #1{#3#4#5#6}{#2}%
865 }%
866 \def\xint@addp@az\W\XINT@addp@AB #1#2%
867 {%
868   \XINT@addp@AC@checkcarry #1%
869 }%
870 \def\XINT@addp@AC@checkcarry #1%
871 {%
872   \xint@zero #1\xint@addp@AC@nocarry 0\XINT@addp@C
873 }%
874 \def\xint@addp@AC@nocarry 0\XINT@addp@C
875 {%
876   \XINT@addp@F
877 }%
878 \def\XINT@addp@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
879 {%
880   \XINT@addp@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
881 }%
882 \def\XINT@addp@ABE #1#2#3#4#5#6%
883 {%
884   \expandafter\XINT@addp@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
885 }%
886 \def\XINT@addp@ABEA #1#2#3#4#5#6#7%
887 {%
888   \XINT@addp@A #2{#7#6#5#4#3}%<-- attention on met donc \'a droite
889 }%
890 \def\XINT@addp@C #1#2#3#4#5%
891 {%
892   \xint@w

```

## 19 Package *xint* implementation

```

893   #5\xint@addp@cw
894   #4\xint@addp@cx
895   #3\xint@addp@cy
896   #2\xint@addp@cz
897   \W\XINT@addp@CD    {#5#4#3#2}{#1}%
898 }%
899 \def\XINT@addp@CD #1%
900 {%
901   \expandafter\XINT@addp@CC\the\numexpr 1+10#1\relax
902 }%
903 \def\XINT@addp@CC #1#2#3#4#5#6#7%
904 {%
905   \XINT@addp@AC@checkcarry #2{#7#6#5#4#3}%
906 }%
907 \def\xint@addp@cw
908   #1\xint@addp@cx
909   #2\xint@addp@cy
910   #3\xint@addp@cz
911   \W\XINT@addp@CD
912 {%
913   \expandafter\XINT@addp@CDw\the\numexpr 1+10#1#2#3\relax
914 }%
915 \def\XINT@addp@CDw #1#2#3#4#5#6%
916 {%
917   \xint@quatrezeros #2#3#4#5\XINT@addp@endDw@zeros
918   0000\XINT@addp@endDw #2#3#4#5%
919 }%
920 \def\XINT@addp@endDw@zeros 0000\XINT@addp@endDw 0000#1\X\Y\Z{ #1}%
921 \def\XINT@addp@endDw #1#2#3#4#5\X\Y\Z{ #5#4#3#2#1}%
922 \def\xint@addp@cx
923   #1\xint@addp@cy
924   #2\xint@addp@cz
925   \W\XINT@addp@CD
926 {%
927   \expandafter\XINT@addp@CDx\the\numexpr 1+100#1#2\relax
928 }%
929 \def\XINT@addp@CDx #1#2#3#4#5#6%
930 {%
931   \xint@quatrezeros #2#3#4#5\XINT@addp@endDx@zeros
932   0000\XINT@addp@endDx #2#3#4#5%
933 }%
934 \def\XINT@addp@endDx@zeros 0000\XINT@addp@endDx 0000#1\Y\Z{ #1}%
935 \def\XINT@addp@endDx #1#2#3#4#5\Y\Z{ #5#4#3#2#1}%
936 \def\xint@addp@cy
937   #1\xint@addp@cz
938   \W\XINT@addp@CD
939 {%
940   \expandafter\XINT@addp@CDy\the\numexpr 1+1000#1\relax
941 }%

```

## 19 Package *xint* implementation

```

942 \def\XINT@addp@CDy #1#2#3#4#5#6%
943 {%
944     \xint@quatrezeros #2#3#4#5\XINT@addp@endDy@zeros
945         0000\XINT@addp@endDy #2#3#4#5%
946 }%
947 \def\XINT@addp@endDy@zeros 0000\XINT@addp@endDy 0000#1\Z{ #1}%
948 \def\XINT@addp@endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
949 \def\xint@addp@cz\W\XINT@addp@CD #1#2{ #21000}%
950 \def\XINT@addp@F #1#2#3#4#5%
951 {%
952     \xint@w
953     #5\xint@addp@Gw
954     #4\xint@addp@Gx
955     #3\xint@addp@Gy
956     #2\xint@addp@Gz
957     \W\XINT@addp@G {#2#3#4#5}{#1}%
958 }%
959 \def\XINT@addp@G #1#2%
960 {%
961     \XINT@addp@F {#2#1}%
962 }%
963 \def\xint@addp@Gw
964     #1\xint@addp@Gx
965     #2\xint@addp@Gy
966     #3\xint@addp@Gz
967     \W\XINT@addp@G #4%
968 {%
969     \xint@quatrezeros #3#2#10\XINT@addp@endGw@zeros
970         0000\XINT@addp@endGw #3#2#10%
971 }%
972 \def\XINT@addp@endGw@zeros 0000\XINT@addp@endGw 0000#1\X\Y\Z{ #1}%
973 \def\XINT@addp@endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
974 \def\xint@addp@Gx
975     #1\xint@addp@Gy
976     #2\xint@addp@Gz
977     \W\XINT@addp@G #3%
978 {%
979     \xint@quatrezeros #2#100\XINT@addp@endGx@zeros
980         0000\XINT@addp@endGx #2#100%
981 }%
982 \def\XINT@addp@endGx@zeros 0000\XINT@addp@endGx 0000#1\Y\Z{ #1}%
983 \def\XINT@addp@endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
984 \def\xint@addp@Gy
985     #1\xint@addp@Gz
986     \W\XINT@addp@G #2%
987 {%
988     \xint@quatrezeros #1000\XINT@addp@endGy@zeros
989         0000\XINT@addp@endGy #1000%
990 }%

```

```

991 \def\XINT@addp@endGy@zeros 0000\XINT@addp@endGy 0000#1\Z{ #1}%
992 \def\XINT@addp@endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
993 \def\xint@addp@Gz\W\XINT@addp@G #1#2{ #2}%

```

### 19.19 \xintAdd

```

994 \def\xintiAdd {\romannumeral0\xintiadd }%
995 \def\xintiadd #1%
996 {%
997   \expandafter\xint@add\expandafter{\romannumeral-‘0#1}%
998 }%
999 \let\xintAdd\xintiAdd \let\xintadd\xintiadd
1000 \def\xint@add #1#2%
1001 {%
1002   \expandafter\XINT@add@fork \romannumeral-‘0#2\Z #1\Z
1003 }%
1004 \def\XINT@Add #1#2{\romannumeral0\XINT@add@fork #2\Z #1\Z }%
1005 \def\XINT@add #1#2{\XINT@add@fork #2\Z #1\Z }%
      ADDITION
      Ici #1#2 vient du *deuxième* argument de \xintAdd et #3#4 donc du *premier*
      [algo plus efficace lorsque le premier est plus long que le second]

1006 \def\XINT@add@fork #1#2\Z #3#4\Z
1007 {%
1008   \xint@UDzerofork
1009     #1\dummy \XINT@add@secondiszero
1010     #3\dummy \XINT@add@firstiszero
1011     0\dummy
1012     {\xint@UDsignsfork
1013       #1#3\dummy \XINT@add@minusminus      % #1 = #3 = -
1014       #1-\dummy \XINT@add@minusplus        % #1 = -
1015       #3-\dummy \XINT@add@plusminus        % #3 = -
1016       --\dummy \XINT@add@plusplus
1017     \xint@UDkrof }%
1018   \xint@UDkrof
1019   {#2}{#4}#1#3%
1020 }%
1021 \def\XINT@add@secondiszero #1#2#3#4{ #4#2}%
1022 \def\XINT@add@firstiszero #1#2#3#4{ #3#1}%

      #1 vient du *deuxième* et #2 vient du *premier*

1023 \def\XINT@add@minusminus #1#2#3#4%
1024 {%
1025   \expandafter\xint@minus@andstop%
1026   \romannumeral0\XINT@add@pre {#2}{#1}%
1027 }%
1028 \def\XINT@add@minusplus #1#2#3#4%
1029 {%
1030   \XINT@sub@pre {#4#2}{#1}%

```

```

1031 }%
1032 \def\XINT@add@plusminus #1#2#3#4%
1033 {%
1034     \XINT@sub@pre {#3#1}{#2}%
1035 }%
1036 \def\XINT@add@plusplus #1#2#3#4%
1037 {%
1038     \XINT@add@pre {#4#2}{#3#1}%
1039 }%
1040 \def\XINT@add@pre #1%
1041 {%
1042     \expandafter\XINT@add@@pre\expandafter
1043     {\romannumeral0\XINT@RQ {}}#1\R\R\R\R\R\R\R\R\Z }%
1044 }%
1045 \def\XINT@add@@pre #1#2%
1046 {%
1047     \expandafter\XINT@add@A
1048     \expandafter0\expandafter{\expandafter}%
1049     \romannumeral0\XINT@RQ {}}#2\R\R\R\R\R\R\R\R\Z
1050     \W\X\Y\Z #1\W\X\Y\Z
1051 }%

```

## 19.20 \xintSub

```

1052 \def\xintiSub {\romannumeral0\xintisub }%
1053 \def\xintisub #1%
1054 {%
1055     \expandafter\xint@sub\expandafter{\romannumeral-‘0#1}%
1056 }%
1057 \let\xintSub\xintiSub \let\xintsub\xintisub
1058 \def\xint@sub #1#2%
1059 {%
1060     \expandafter\XINT@sub@fork \romannumeral-‘0#2\Z #1\Z
1061 }%
1062 \def\XINT@Sub #1#2{\romannumeral0\XINT@sub@fork #2\Z #1\Z }%
1063 \def\XINT@sub #1#2{\XINT@sub@fork #2\Z #1\Z }%
1064     SOUSTRACTION
1065     #3#4-#1#2
1066     #3#4 vient du *premier*
1067     #1#2 vient du *second*
1068
1064 \def\XINT@sub@fork #1#2\Z #3#4\Z
1065 {%
1066     \xint@UDsignsfork
1067     #1#3\dummy \XINT@sub@minusminus
1068     #1-\dummy \XINT@sub@minusplus    % attention, #3=0 possible
1069     #3-\dummy \XINT@sub@plusminus    % attention, #1=0 possible
1070     --\dummy {\xint@UDzerofork
1071     #1\dummy \XINT@sub@secondiszero

```

## 19 Package **xint** implementation

```

1072          #3\dummy \XINT@sub@firstiszero
1073          0\dummy \XINT@sub@plusplus
1074          \xint@UDkrof }%
1075      \xint@UDkrof
1076      {#2}{#4}#1#3%
1077 }%
1078 \def\XINT@sub@secondiszero #1#2#3#4{ #4#2}%
1079 \def\XINT@sub@firstiszero #1#2#3#4{ -#3#1}%
1080 \def\XINT@sub@plusplus #1#2#3#4%
1081 {%
1082     \XINT@sub@pre {#4#2}{#3#1}%
1083 }%
1084 \def\XINT@sub@minusminus #1#2#3#4%
1085 {%
1086     \XINT@sub@pre {#1}{#2}%
1087 }%
1088 \def\XINT@sub@minusplus #1#2#3#4%
1089 {%
1090     \xint@zero #4\xint@sub@mp0\XINT@add@pre {#4#2}{#1}%
1091 }%
1092 \def\xint@sub@mp0\XINT@add@pre #1#2{ #2}%
1093 \def\XINT@sub@plusminus #1#2#3#4%
1094 {%
1095     \xint@zero #3\xint@sub@pm0\expandafter\xint@minus@andstop%
1096     \romannumeral0\XINT@add@pre {#2}{#3#1}%
1097 }%
1098 \def\xint@sub@pm #1\XINT@add@pre #2#3{ -#2}%
1099 \def\XINT@sub@pre #1%
1100 {%
1101     \expandafter\XINT@sub@@@pre\expandafter
1102     {\romannumeral0\XINT@RQ {}}#1\R\R\R\R\R\R\R\R\Z }%
1103 }%
1104 \def\XINT@sub@@@pre #1#2%
1105 {%
1106     \expandafter\XINT@sub@A
1107     \expandafter1\expandafter{\expandafter}%
1108     \romannumeral0\XINT@RQ {}}#2\R\R\R\R\R\R\R\R\Z
1109     \W\X\Y\Z #1 \W\X\Y\Z
1110 }%

\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.

1111 \def\XINT@sub@A #1#2#3\W\X\Y\Z #4#5#6#7%
1112 {%

```



## 19 Package *xint* implementation

```

1113 \xint@w
1114 #4\xint@sub@az
1115 \W\XINT@sub@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
1116 }%
1117 \def\XINT@sub@B #1#2#3#4#5#6#7%
1118 {%
1119 \xint@w
1120 #4\xint@sub@bz
1121 \W\XINT@sub@onestep #1#2{#7#6#5#4}{#3}%
1122 }%

```

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  $N2 - N1$ .

```

1123 \def\XINT@sub@onestep #1#2#3#4#5#6%
1124 {%
1125 \expandafter\XINT@sub@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
1126 }%

```

ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE

```

1127 \def\XINT@sub@backtoA #1#2#3.#4%
1128 {%
1129 \XINT@sub@A #2{#3#4}%
1130 }%
1131 \def\xint@sub@bz
1132 \W\XINT@sub@onestep #1#2#3#4#5#6#7%
1133 {%
1134 \xint@UDzerofork
1135 #1\dummy \XINT@sub@C % une retenue
1136 0\dummy \XINT@sub@D % pas de retenue
1137 \xint@UDkrof
1138 {#7}#2#3#4#5%
1139 }%
1140 \def\XINT@sub@D #1#2\W\X\Y\Z
1141 {%
1142 \expandafter
1143 \xint@cleanupzeros@andstop
1144 \romannumeral0%
1145 \XINT@rord@main {}#2%
1146 \xint@UNDEF
1147 \xint@undef\xint@undef\xint@undef\xint@undef
1148 \xint@undef\xint@undef\xint@undef\xint@undef
1149 \xint@UNDEF
1150 #1%
1151 }%
1152 \def\XINT@sub@C #1#2#3#4#5%
1153 {%

```

## 19 Package *xint* implementation

```

1154 \xint@w
1155 #2\xint@sub@cz
1156 \W\XINT@sub@AC@onestep {#5#4#3#2}{#1}%
1157 }%
1158 \def\XINT@sub@AC@onestep #1%
1159 {%
1160 \expandafter\XINT@sub@backtoC\the\numexpr 11#1-1\relax.%
1161 }%
1162 \def\XINT@sub@backtoC #1#2#3.#4%
1163 {%
1164 \XINT@sub@AC@checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
1165 }%
1166 \def\XINT@sub@AC@checkcarry #1%
1167 {%
1168 \xint@one #1\xint@sub@AC@nocarry 1\XINT@sub@C
1169 }%
1170 \def\xint@sub@AC@nocarry 1\XINT@sub@C #1#2\W\X\Y\Z
1171 {%
1172 \expandafter
1173 \XINT@cuz@loop
1174 \romannumeral0%
1175 \XINT@rord@main {}#2%
1176 \xint@UNDEF
1177 \xint@undef\xint@undef\xint@undef\xint@undef
1178 \xint@undef\xint@undef\xint@undef\xint@undef
1179 \xint@UNDEF
1180 #1\W\W\W\W\W\W\W\Z
1181 }%
1182 \def\xint@sub@cz\W\XINT@sub@AC@onestep #1%
1183 {%
1184 \XINT@cuz
1185 }%
1186 \def\xint@sub@az\W\XINT@sub@B #1#2#3#4#5#6#7%
1187 {%
1188 \xint@w
1189 #4\xint@sub@ez
1190 \W\XINT@sub@Eenter #1{#3}#4#5#6#7%
1191 }%

```

le premier nombre continue, le résultat sera < 0.

```

1192 \def\XINT@sub@Eenter #1#2%
1193 {%
1194 \expandafter
1195 \XINT@sub@E\expandafter1\expandafter{\expandafter}%
1196 \romannumeral0%
1197 \XINT@rord@main {}#2%
1198 \xint@UNDEF
1199 \xint@undef\xint@undef\xint@undef\xint@undef
1200 \xint@undef\xint@undef\xint@undef\xint@undef

```

## 19 Package *xint* implementation

```

1201 \xint@UNDEF
1202 \W\X\Y\Z #1%
1203 }%
1204 \def\xINT@sub@E #1#2#3#4#5#6%
1205 {%
1206 \xint@w #3\xint@sub@F\W\xINT@sub@Eonestep #1{#6#5#4#3}{#2}%
1207 }%
1208 \def\xINT@sub@Eonestep #1#2%
1209 {%
1210 \expandafter\xINT@sub@backtoE\the\numexpr 109999-#2+#1\relax.%
1211 }%
1212 \def\xINT@sub@backtoE #1#2#3.#4%
1213 {%
1214 \XINT@sub@E #2{#3#4}%
1215 }%
1216 \def\xint@sub@F\W\xINT@sub@Eonestep #1#2#3#4%
1217 {%
1218 \xint@UDonezerofork
1219 #4#1\dummy {\XINT@sub@Fdec 0}% soustraire 1. Et faire signe -
1220 #1#4\dummy {\XINT@sub@Finc 1}% additionner 1. Et faire signe -
1221 10\dummy \XINT@sub@DD % terminer. Mais avec signe -
1222 \xint@UDkrof
1223 {#3}%
1224 }%
1225 \def\xINT@sub@DD {\expandafter\xint@minus@andstop\romannumeral0\xINT@sub@D }%
1226 \def\xINT@sub@Fdec #1#2#3#4#5#6%
1227 {%
1228 \xint@w #3\xint@sub@Fdec@finish\W
1229 \XINT@sub@Fdec@onestep #1{#6#5#4#3}{#2}%
1230 }%
1231 \def\xINT@sub@Fdec@onestep #1#2%
1232 {%
1233 \expandafter\xINT@sub@backtoFdec\the\numexpr 11#2+#1-1\relax.%
1234 }%
1235 \def\xINT@sub@backtoFdec #1#2#3.#4%
1236 {%
1237 \XINT@sub@Fdec #2{#3#4}%
1238 }%
1239 \def\xint@sub@Fdec@finish\W\xINT@sub@Fdec@onestep #1#2%
1240 {%
1241 \expandafter\xint@minus@andstop\romannumeral0\xINT@cuz
1242 }%
1243 \def\xINT@sub@Finc #1#2#3#4#5#6%
1244 {%
1245 \xint@w #3\xint@sub@Finc@finish\W
1246 \XINT@sub@Finc@onestep #1{#6#5#4#3}{#2}%
1247 }%
1248 \def\xINT@sub@Finc@onestep #1#2%
1249 {%

```

## 19 Package *xint* implementation

```

1250 \expandafter\XINT@sub@backtoFinc\the\numexpr 10#2+#1\relax.%
1251 }%
1252 \def\XINT@sub@backtoFinc #1#2#3.#4%
1253 {%
1254 \XINT@sub@Finc #2{#3#4}%
1255 }%
1256 \def\xint@sub@Finc@finish\W\XINT@sub@Finc@onestep #1#2#3%
1257 {%
1258 \xint@UDzerofork
1259 #1\dummy {\expandafter\xint@minus@andstop\xint@cleanupzeros@nospace}%
1260 0\dummy { -1}%
1261 \xint@UDkrof
1262 #3%
1263 }%
1264 \def\xint@sub@ez\W\XINT@sub@Eenter #1%
1265 {%
1266 \xint@UDzerofork
1267 #1\dummy \XINT@sub@K % il y a une retenue
1268 0\dummy \XINT@sub@L % pas de retenue
1269 \xint@UDkrof
1270 }%
1271 \def\XINT@sub@L #1\W\X\Y\Z {\XINT@cuz@loop #1\W\W\W\W\W\W\W\W\Z }%
1272 \def\XINT@sub@K #1%
1273 {%
1274 \expandafter
1275 \XINT@sub@KK\expandafter1\expandafter{\expandafter}%
1276 \romannumeral0%
1277 \XINT@rord@main {}#1%
1278 \xint@UNDEF
1279 \xint@undef\xint@undef\xint@undef\xint@undef
1280 \xint@undef\xint@undef\xint@undef\xint@undef
1281 \xint@UNDEF
1282 }%
1283 \def\XINT@sub@KK #1#2#3#4#5#6%
1284 {%
1285 \xint@w #3\xint@sub@KK@finish\W
1286 \XINT@sub@KK@onestep #1{#6#5#4#3}{#2}%
1287 }%
1288 \def\XINT@sub@KK@onestep #1#2%
1289 {%
1290 \expandafter\XINT@sub@backtoKK\the\numexpr 109999-#2+#1\relax.%
1291 }%
1292 \def\XINT@sub@backtoKK #1#2#3.#4%
1293 {%
1294 \XINT@sub@KK #2{#3#4}%
1295 }%
1296 \def\xint@sub@KK@finish\W\XINT@sub@KK@onestep #1#2#3%
1297 {%
1298 \expandafter\xint@minus@andstop

```

```

1299 \romannumeral0\XINT@cuz@loop #3\W\W\W\W\W\W\W\Z
1300 }%

```

### 19.21 \xintCmp

```

1301 \def\xintiCmp {\romannumeral0\xinticmp }%
1302 \def\xinticmp #1%
1303 {%
1304   \expandafter\xint@cmp\expandafter{\romannumeral-‘0#1}%
1305 }%
1306 \let\xintCmp\xintiCmp \let\xintcmp\xinticmp
1307 \def\xint@cmp #1#2%
1308 {%
1309   \expandafter\XINT@cmp@fork \romannumeral-‘0#2\Z #1\Z
1310 }%
1311 \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*

1312 \def\XINT@cmp@fork #1#2\Z #3#4\Z
1313 {%
1314   \xint@UDsignsfork
1315     #1#3\dummy \XINT@cmp@minusminus
1316     #1-\dummy \XINT@cmp@minusplus
1317     #3-\dummy \XINT@cmp@plusminus
1318     --\dummy {\xint@UDzerosfork
1319       #1#3\dummy \XINT@cmp@zerozero
1320       #10\dummy \XINT@cmp@zeroplus
1321       #30\dummy \XINT@cmp@pluszero
1322       00\dummy \XINT@cmp@plusplus
1323       \xint@UDkrof }%
1324   \xint@UDkrof
1325   {#2}{#4}#1#3%
1326 }%
1327 \def\XINT@cmp@minusplus #1#2#3#4{ 1}%
1328 \def\XINT@cmp@plusminus #1#2#3#4{ -1}%
1329 \def\XINT@cmp@zerozero #1#2#3#4{ 0}%
1330 \def\XINT@cmp@zeroplus #1#2#3#4{ 1}%
1331 \def\XINT@cmp@pluszero #1#2#3#4{ -1}%
1332 \def\XINT@cmp@plusplus #1#2#3#4%
1333 {%
1334   \XINT@cmp@pre {#4#2}{#3#1}%
1335 }%
1336 \def\XINT@cmp@minusminus #1#2#3#4%
1337 {%
1338   \XINT@cmp@pre {#1}{#2}%
1339 }%

```

## 19 Package *xint* implementation

```

1340 \def\XINT@cmp@pre #1%
1341 {%
1342   \expandafter\XINT@cmp@@pre\expandafter
1343   {\romannumeral0\XINT@RQ {}}#1\R\R\R\R\R\R\R\R\Z}%
1344}%
1345 \def\XINT@cmp@@pre #1#2%
1346 {%
1347   \expandafter\XINT@cmp@A
1348   \expandafter1\expandafter{\expandafter}%
1349   \romannumeral0\XINT@RQ {}}#2\R\R\R\R\R\R\R\R\Z
1350   \W\X\Y\Z #1\W\X\Y\Z
1351}%

COMPARAISON
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 \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

1352 \def\XINT@cmp@A #1#2#3\W\X\Y\Z #4#5#6#7%
1353 {%
1354   \xint@w #4\xint@cmp@az\W\XINT@cmp@B #1{#4#5#6#7}{#2}{#3\W\X\Y\Z
1355}%
1356 \def\XINT@cmp@B #1#2#3#4#5#6#7%
1357 {%
1358   \xint@w#4\xint@cmp@bz\W\XINT@cmp@onestep #1#2{#7#6#5#4}{#3}%
1359}%
1360 \def\XINT@cmp@onestep #1#2#3#4#5#6%
1361 {%
1362   \expandafter\XINT@cmp@backto\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
1363}%
1364 \def\XINT@cmp@backtoA #1#2#3.#4%
1365 {%
1366   \XINT@cmp@A #2{#3#4}%
1367}%
1368 \def\xint@cmp@bz\W\XINT@cmp@onestep #1\Z { 1}%
1369 \def\xint@cmp@az\W\XINT@cmp@B #1#2#3#4#5#6#7%
1370 {%
1371   \xint@w #4\xint@cmp@ez\W\XINT@cmp@Eenter #1{#3}{#4#5#6#7%
1372}%
1373 \def\XINT@cmp@Eenter #1\Z { -1}%
1374 \def\xint@cmp@ez\W\XINT@cmp@Eenter #1%
1375 {%
1376   \xint@UDzerofork
1377   #1\dummy \XINT@cmp@K % il y a une retenue
1378   0\dummy \XINT@cmp@L % pas de retenue
1379   \xint@UDkrof
1380}%
1381 \def\XINT@cmp@K #1\Z { -1}%

```

## 19 Package *xint* implementation

```

1382 \def\XINT@cmp@L #1{\XINT@OneIfPositive@main #1}%
1383 \def\XINT@OneIfPositive #1%
1384 {%
1385   \XINT@OneIfPositive@main #1\W\X\Y\Z%
1386 }%
1387 \def\XINT@OneIfPositive@main #1#2#3#4%
1388 {%
1389   \xint@z #4\xint@OneIfPositive@terminated\Z
1390   \XINT@OneIfPositive@onestep #1#2#3#4%
1391 }%
1392 \def\xint@OneIfPositive@terminated\Z\XINT@OneIfPositive@onestep\W\X\Y\Z { 0}%
1393 \def\XINT@OneIfPositive@onestep #1#2#3#4%
1394 {%
1395   \expandafter\XINT@OneIfPositive@check\the\numexpr #1#2#3#4\relax
1396 }%
1397 \def\XINT@OneIfPositive@check #1%
1398 {%
1399   \xint@zero #1\xint@OneIfPositive@backtomain 0%
1400   \XINT@OneIfPositive@finish #1%
1401 }%
1402 \def\XINT@OneIfPositive@finish #1\W\X\Y\Z{ 1}%
1403 \def\xint@OneIfPositive@backtomain 0\XINT@OneIfPositive@finish 0%
1404       {\XINT@OneIfPositive@main }%

```

### 19.22 \xintGeq

```

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

1405 \def\xintiGeq {\romannumeral0\xintigeq }%
1406 \def\xintigeq #1%
1407 {%
1408   \expandafter\xint@geq\expandafter {\romannumeral-‘0#1}%
1409 }%
1410 \let\xintGeq\xintiGeq \let\xintgeq\xintigeq
1411 \def\xint@geq #1#2%
1412 {%
1413   \expandafter\XINT@geq@fork \romannumeral-‘0#2\Z #1\Z
1414 }%
1415 \def\XINT@Geq #1#2{\romannumeral0\XINT@geq@fork #2\Z #1\Z }%

PLUS GRAND OU ÉGAL
ATTENTION, TESTE les VALEURS ABSOLUES

1416 \def\XINT@geq@fork #1#2\Z #3#4\Z
1417 {%
1418   \xint@UDzerofork
1419   #1\dummy \XINT@geq@secondiszero % |#1#2|=0
1420   #3\dummy \XINT@geq@firstiszero % |#1#2|>0
1421   0\dummy {\xint@UDsignsfork

```

## 19 Package **xint** implementation

```

1422      #1#3\dummy \XINT@geq@minusminus
1423      #1-\dummy \XINT@geq@minusplus
1424      #3-\dummy \XINT@geq@plusminus
1425      --\dummy \XINT@geq@plusplus
1426      \xint@UDkrof }%
1427      \xint@UDkrof
1428      {#2}{#4}#1#3%
1429 }%
1430 \def\XINT@geq@secondiszero #1#2#3#4{ 1}%
1431 \def\XINT@geq@firstiszero #1#2#3#4{ 0}%
1432 \def\XINT@geq@plusplus #1#2#3#4{\XINT@geq@pre {#4#2}{#3#1}}%
1433 \def\XINT@geq@minusminus #1#2#3#4{\XINT@geq@pre {#2}{#1}}%
1434 \def\XINT@geq@minusplus #1#2#3#4{\XINT@geq@pre {#4#2}{#1}}%
1435 \def\XINT@geq@plusminus #1#2#3#4{\XINT@geq@pre {#2}{#3#1}}%
1436 \def\XINT@geq@pre #1%
1437 {%
1438      \expandafter\XINT@geq@@pre\expandafter
1439      {\romannumeral0\XINT@RQ {}}#1\R\R\R\R\R\R\R\R\Z }%
1440 }%
1441 \def\XINT@geq@@pre #1#2%
1442 {%
1443      \expandafter\XINT@geq@A
1444      \expandafter1\expandafter{\expandafter}%
1445      \romannumeral0\XINT@RQ {}}#2\R\R\R\R\R\R\R\R\Z
1446      \W\X\Y\Z #1 \W\X\Y\Z
1447 }%

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

1448 \def\XINT@geq@A #1#2#3\W\X\Y\Z #4#5#6#7%
1449 {%
1450      \xint@w #4\xint@geq@az\W\XINT@geq@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
1451 }%
1452 \def\XINT@geq@B #1#2#3#4#5#6#7%
1453 {%
1454      \xint@w #4\xint@geq@bz\W\XINT@geq@onestep #1#2{#7#6#5#4}{#3}%
1455 }%
1456 \def\XINT@geq@onestep #1#2#3#4#5#6%
1457 {%
1458      \expandafter\XINT@geq@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
1459 }%
1460 \def\XINT@geq@backtoA #1#2#3.#4%
1461 {%
1462      \XINT@geq@A #2{#3#4}%

```



```

1463 }%
1464 \def\xint@geq@bz\W\XINT@geq@onestep #1\W\X\Y\Z { 1}%
1465 \def\xint@geq@az\W\XINT@geq@B #1#2#3#4#5#6#7%
1466 {%
1467   \xint@w #4\xint@geq@ez\W\XINT@geq@Eenter #1%
1468 }%
1469 \def\XINT@geq@Eenter #1\W\X\Y\Z { 0}%
1470 \def\xint@geq@ez\W\XINT@geq@Eenter #1%
1471 {%
1472   \xint@UDzerofork
1473     #1\dummy { 0}           %      il y a une retenue
1474     0\dummy { 1}           %      pas de retenue
1475   \xint@UDkrof
1476 }%

```

### 19.23 \xintMax

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.

```

1477 \def\xintiMax {\romannumeral0\xintimax }%
1478 \def\xintimax #1%
1479 {%
1480   \expandafter\xint@max\expandafter {\romannumeral-‘0#1}%
1481 }%
1482 \let\xintMax\xintiMax \let\xintmax\xintimax
1483 \def\xint@max #1#2%
1484 {%
1485   \expandafter\XINT@max@pre\expandafter {\romannumeral-‘0#2}{#1}%
1486 }%
1487 \def\XINT@max@pre #1#2{\XINT@max@fork #1\Z #2\Z {#2}{#1}}%
1488 \def\XINT@Max #1#2{\romannumeral0\XINT@max@fork #2\Z #1\Z {#1}{#2}}%
1489
1490   #3#4 vient du *premier*
1491   #1#2 vient du *second*
1489 \def\XINT@max@fork #1#2\Z #3#4\Z
1490 {%
1491   \xint@UDsignsfork
1492     #1#3\dummy \XINT@max@minusminus % A < 0, B < 0
1493     #1-\dummy \XINT@max@minusplus   % B < 0, A >= 0
1494     #3-\dummy \XINT@max@plusminus   % A < 0, B >= 0
1495     --\dummy {\xint@UDzerosfork
1496       #1#3\dummy \XINT@max@zerozero % A = B = 0
1497       #10\dummy \XINT@max@zeroplus  % B = 0, A > 0
1498       #30\dummy \XINT@max@pluszero  % A = 0, B > 0
1499       00\dummy \XINT@max@plusplus  % A, B > 0
1500     \xint@UDkrof }%
1501   \xint@UDkrof
1502   {#2}{#4}#1#3%
1503 }%

```

```

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

1504 \def\XINT@max@zerozero #1#2#3#4{\xint@firstoftwo@andstop}%
1505 \def\XINT@max@zeroplus #1#2#3#4{\xint@firstoftwo@andstop}%
1506 \def\XINT@max@pluszero #1#2#3#4{\xint@secondoftwo@andstop}%
1507 \def\XINT@max@minusplus #1#2#3#4{\xint@firstoftwo@andstop}%
1508 \def\XINT@max@plusminus #1#2#3#4{\xint@secondoftwo@andstop}%
1509 \def\XINT@max@plusplus #1#2#3#4%
1510 {%
1511     \ifodd\XINT@Geq {#4#2}{#3#1}
1512     \expandafter\xint@firstoftwo@andstop
1513     \else
1514     \expandafter\xint@secondoftwo@andstop
1515     \fi
1516}%

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

1517 \def\XINT@max@minusminus #1#2#3#4%
1518 {%
1519     \ifodd\XINT@Geq {#1}{#2}
1520     \expandafter\xint@firstoftwo@andstop
1521     \else
1522     \expandafter\xint@secondoftwo@andstop
1523     \fi
1524}%

```

## 19.24 \xintMin

```

1525 \def\xintiMin {\romannumeral0\xintimin}%
1526 \def\xintimin #1%
1527 {%
1528     \expandafter\xint@min\expandafter {\romannumeral-‘0#1}%
1529}%
1530 \let\xintMin\xintiMin \let\xintmin\xintimin
1531 \def\xint@min #1#2%
1532 {%
1533     \expandafter\XINT@min@pre\expandafter {\romannumeral-‘0#2}{#1}%
1534}%
1535 \def\XINT@min@pre #1#2{\XINT@min@fork #1\Z #2\Z {#2}{#1}}%
1536 \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*

1537 \def\XINT@min@fork #1#2\Z #3#4\Z
1538 {%
1539     \xint@UDsignsfork
1540     #1#3\dummy \XINT@min@minusminus % A < 0, B < 0
1541     #1-\dummy \XINT@min@minusplus % B < 0, A >= 0

```

## 19 Package *xint* implementation

```

1542      #3-\dummy \XINT@min@plusminus % A < 0, B >= 0
1543      --\dummy {\xint@UDzerosfork
1544          #1#3\dummy \XINT@min@zerozero % A = B = 0
1545          #10\dummy \XINT@min@zeroplus % B = 0, A > 0
1546          #30\dummy \XINT@min@pluszero % A = 0, B > 0
1547          00\dummy \XINT@min@plusplus % A, B > 0
1548          \xint@UDkrof }%
1549      \xint@UDkrof
1550      {#2}{#4}#1#3%
1551 }%

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

1552 \def\XINT@min@zerozero #1#2#3#4{\xint@firstoftwo@andstop }%
1553 \def\XINT@min@zeroplus #1#2#3#4{\xint@secondoftwo@andstop }%
1554 \def\XINT@min@pluszero #1#2#3#4{\xint@firstoftwo@andstop }%
1555 \def\XINT@min@minusplus #1#2#3#4{\xint@secondoftwo@andstop }%
1556 \def\XINT@min@plusminus #1#2#3#4{\xint@firstoftwo@andstop }%
1557 \def\XINT@min@plusplus #1#2#3#4%
1558 {%
1559     \ifodd\XINT@Geq {#4#2}{#3#1}
1560     \expandafter\xint@secondoftwo@andstop
1561     \else
1562     \expandafter\xint@firstoftwo@andstop
1563     \fi
1564 }%

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

1565 \def\XINT@min@minusminus #1#2#3#4%
1566 {%
1567     \ifodd\XINT@Geq {#1}{#2}
1568     \expandafter\xint@secondoftwo@andstop
1569     \else
1570     \expandafter\xint@firstoftwo@andstop
1571     \fi
1572 }%

```

### 19.25 \xintSum, \xintSumExpr

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

## 19 Package *xint* implementation

```

1573 \def\xintiSum {\romannumeral0\xintisum }%
1574 \def\xintisum #1{\xintisumexpr #1\relax }%
1575 \def\xintiSumExpr {\romannumeral0\xintisumexpr }%
1576 \def\xintisumexpr {\expandafter\XINT@sumexpr\romannumeral-‘0}%
1577 \let\xintSum\xintiSum \let\xintsum\xintisum
1578 \let\xintSumExpr\xintiSumExpr \let\xintsumexpr\xintisumexpr
1579 \def\XINT@sumexpr {\XINT@sum@loop {0000}{0000}}%
1580 \def\XINT@sum@loop #1#2#3%
1581 {%
1582   \expandafter\XINT@sum@checksign\romannumeral-‘0#3\Z {#1}{#2}%
1583 }%
1584 \def\XINT@sum@checksign #1%
1585 {%
1586   \xint@relax #1\XINT@sum@finished\relax
1587   \xint@zero #1\XINT@sum@skipzeroinput0%
1588   \xint@UDsignfork
1589     #1\dummy \XINT@sum@N
1590     -\dummy {\XINT@sum@P #1}%
1591   \xint@UDkrof
1592 }%
1593 \def\XINT@sum@finished #1\Z #2#3%
1594 {%
1595   \XINT@sub@A 1{ }#3\W\X\Y\Z #2\W\X\Y\Z
1596 }%
1597 \def\XINT@sum@skipzeroinput #1\xint@UDkrof #2\Z {\XINT@sum@loop }%
1598 \def\XINT@sum@P #1\Z #2%
1599 {%
1600   \expandafter\XINT@sum@loop\expandafter
1601   {\romannumeral0\expandafter
1602     \XINT@addr@A\expandafter0\expandafter{\expandafter}%
1603     \romannumeral0\XINT@RQ { }#1\R\R\R\R\R\R\R\Z
1604     \W\X\Y\Z #2\W\X\Y\Z }%
1605 }%
1606 \def\XINT@sum@N #1\Z #2#3%
1607 {%
1608   \expandafter\XINT@sum@NN\expandafter
1609   {\romannumeral0\expandafter
1610     \XINT@addr@A\expandafter0\expandafter{\expandafter}%
1611     \romannumeral0\XINT@RQ { }#1\R\R\R\R\R\R\R\Z
1612     \W\X\Y\Z #3\W\X\Y\Z }{#2}%
1613 }%
1614 \def\XINT@sum@NN #1#2{\XINT@sum@loop {#2}{#1}}%

```

### 19.26 \xintMul

```

1615 \def\xintiMul {\romannumeral0\xintimul }%
1616 \def\xintimul #1%
1617 {%
1618   \expandafter\xint@mul\expandafter {\romannumeral-‘0#1}%

```

```

1619}%
1620\let\xintMul\xintiMul \let\xintmul\xintimul
1621\def\xint@mul #1#2%
1622{%
1623  \expandafter\xINT@mul@fork \romannumeral-'0#2\Z #1\Z
1624}%
1625\def\xINT@Mul #1#2{\romannumeral0\xINT@mul@fork #2\Z #1\Z}%
  MULTIPLICATION
  Ici #1#2 = 2e input et #3#4 = 1er input
  Release 1.03 adds some overhead to first compute and compare the
  lengths of the two inputs. The algorithm is asymmetrical and whether
  the first input is the longest or the shortest sometimes has a strong
  impact. 50 digits times 1000 digits used to be 5 times faster
  than 1000 digits times 50 digits. With the new code, the user input
  order does not matter as it is decided by the routine what is best.
  This is important for the extension to fractions, as there is no way
  then to generally control or guess the most frequent sizes of the
  inputs besides actually computing their lengths.

1626\def\xINT@mul@fork #1#2\Z #3#4\Z
1627{%
1628  \xint@UDzerofork
1629  #1\dummy \XINT@mul@zero
1630  #3\dummy \XINT@mul@zero
1631  0\dummy
1632  {\xint@UDsignsfork
1633    #1#3\dummy \XINT@mul@minusminus % #1 = #3 = -
1634    #1-\dummy {\XINT@mul@minusplus #3}% % #1 = -
1635    #3-\dummy {\XINT@mul@plusminus #1}% % #3 = -
1636    --\dummy {\XINT@mul@plusplus #1#3}%
1637  \xint@UDkrof}%
1638  \xint@UDkrof
1639  {#2}{#4}%
1640}%
1641\def\xINT@mul@zero #1#2{ 0}%
1642\def\xINT@mul@minusminus #1#2%
1643{%
1644  \expandafter\xINT@mul@choice@a
1645  \expandafter{\romannumeral0\xINT@length {#2}}%
1646  {\romannumeral0\xINT@length {#1}}{#1}{#2}%
1647}%
1648\def\xINT@mul@minusplus #1#2#3%
1649{%
1650  \expandafter\xint@minus@andstop\romannumeral0\expandafter
1651  \XINT@mul@choice@a
1652  \expandafter{\romannumeral0\xINT@length {#1#3}}%
1653  {\romannumeral0\xINT@length {#2}}{#2}{#1#3}%
1654}%
1655\def\xINT@mul@plusminus #1#2#3%
1656{%

```

## 19 Package *xint* implementation

```

1657 \expandafter\xint@minus@andstop\romannumeral0\expandafter
1658 \XINT@mul@choice@a
1659 \expandafter{\romannumeral0\XINT@length {#3}}%
1660 {\romannumeral0\XINT@length {#1#2}}{#1#2}{#3}%
1661 }%
1662 \def\xint@mul@plusplus #1#2#3#4%
1663 {%
1664 \expandafter\xint@mul@choice@a
1665 \expandafter{\romannumeral0\XINT@length {#2#4}}%
1666 {\romannumeral0\XINT@length {#1#3}}{#1#3}{#2#4}%
1667 }%
1668 \def\xint@mul@choice@a #1#2%
1669 {%
1670 \expandafter\xint@mul@choice@b\expandafter{#2}{#1}%
1671 }%
1672 \def\xint@mul@choice@b #1#2%
1673 {%
1674 \ifnum #1<5
1675 \expandafter\xint@mul@choice@littlebyfirst
1676 \else
1677 \ifnum #2<5
1678 \expandafter\expandafter\expandafter\xint@mul@choice@littlebysecond
1679 \else
1680 \expandafter\expandafter\expandafter\xint@mul@choice@compare
1681 \fi
1682 \fi
1683 {#1}{#2}%
1684 }%
1685 \def\xint@mul@choice@littlebyfirst #1#2#3#4%
1686 {%
1687 \expandafter\xint@mul@M
1688 \expandafter{\the\numexpr #3\expandafter}%
1689 \romannumeral0\XINT@RQ {}#4\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
1690 }%
1691 \def\xint@mul@choice@littlebysecond #1#2#3#4%
1692 {%
1693 \expandafter\xint@mul@M
1694 \expandafter{\the\numexpr #4\expandafter}%
1695 \romannumeral0\XINT@RQ {}#3\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
1696 }%
1697 \def\xint@mul@choice@compare #1#2%
1698 {%
1699 \ifnum #1>#2
1700 \expandafter \xint@mul@choice@i
1701 \else
1702 \expandafter \xint@mul@choice@ii
1703 \fi
1704 {#1}{#2}%
1705 }%

```

## 19 Package *xint* implementation

```

1706 \def\XINT@mul@choice@i #1#2%
1707 {%
1708   \ifnum #1<\numexpr\ifcase \numexpr (#2-3)/4\relax
1709     \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1710     \expandafter\XINT@mul@choice@same
1711   \else
1712     \expandafter\XINT@mul@choice@permute
1713   \fi
1714 }%
1715 \def\XINT@mul@choice@ii #1#2%
1716 {%
1717   \ifnum #2<\numexpr\ifcase \numexpr (#1-3)/4\relax
1718     \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1719     \expandafter\XINT@mul@choice@permute
1720   \else
1721     \expandafter\XINT@mul@choice@same
1722   \fi
1723 }%
1724 \def\XINT@mul@choice@same #1#2%
1725 {%
1726   \expandafter\XINT@mul@enter
1727   \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
1728   \W\X\Y\Z #2\W\X\Y\Z
1729 }%
1730 \def\XINT@mul@choice@permute #1#2%
1731 {%
1732   \expandafter\XINT@mul@enter
1733   \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
1734   \W\X\Y\Z #1\W\X\Y\Z
1735 }%

```

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.

```

1736 \def\XINT@mul@Ar #1#2#3#4#5#6%
1737 {%
1738   \xint@z #6\xint@mul@br\Z\XINT@mul@Br #1{#6#5#4#3}{#2}%
1739 }%
1740 \def\xint@mul@br\Z\XINT@mul@Br #1#2%
1741 {%
1742   \XINT@addr@AC@checkcarry #1%
1743 }%
1744 \def\XINT@mul@Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
1745 {%
1746   \expandafter\XINT@mul@ABEAr
1747   \the\numexpr #1+10#2+#8#7#6#5\relax.{#3}#4\W\X\Y\Z
1748 }%
1749 \def\XINT@mul@ABEAr #1#2#3#4#5#6.#7%

```

## 19 Package *xint* implementation

```

1750 {%
1751   \XINT@mul@Ar #2{#7#6#5#4#3}%
1752 }%

<< Petite >> multiplication.
mul@Mr renvoie le résultat *à l'envers*, sur *4n*
\romannumeral0\XINT@mul@Mr {<n>}<N>\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.

1753 \def\XINT@mul@Mr #1%
1754 {%
1755   \expandafter\XINT@mul@Mr@checkifzeroorone\expandafter{\the\numexpr #1}%
1756 }%
1757 \def\XINT@mul@Mr@checkifzeroorone #1%
1758 {%
1759   \ifcase #1
1760     \expandafter\XINT@mul@Mr@zero
1761   \or
1762     \expandafter\XINT@mul@Mr@one
1763   \else
1764     \expandafter\XINT@mul@Nr
1765   \fi
1766   {0000}{\#1}%
1767 }%
1768 \def\XINT@mul@Mr@zero #1\Z\Z\Z\Z { 0000}%
1769 \def\XINT@mul@Mr@one #1#2#3#4\Z\Z\Z\Z { #4}%
1770 \def\XINT@mul@Nr #1#2#3#4#5#6#7%
1771 {%
1772   \xint@z #4\xint@mul@pr\Z\XINT@mul@Pr {\#1}{\#3}{\#7#6#5#4}{\#2}{\#3}%
1773 }%
1774 \def\XINT@mul@Pr #1#2#3%
1775 {%
1776   \expandafter\XINT@mul@Lr\the\numexpr 10000#1+#2*#3\relax
1777 }%
1778 \def\XINT@mul@Lr 1#1#2#3#4#5#6#7#8#9%
1779 {%
1780   \XINT@mul@Nr {\#1#2#3#4}{\#9#8#7#6#5}%
1781 }%
1782 \def\xint@mul@pr\Z\XINT@mul@Pr #1#2#3#4#5%
1783 {%
1784   \xint@quatrezeros #1\XINT@mul@Mr@end@nocarry 0000%
1785   \XINT@mul@Mr@end@carry #1{\#4}%
1786 }%
1787 \def\XINT@mul@Mr@end@nocarry 0000\XINT@mul@Mr@end@carry 0000#1{ #1}%
1788 \def\XINT@mul@Mr@end@carry #1#2#3#4#5{ #5#4#3#2#1}%

<< Petite >> multiplication.
renvoie le résultat *à l'endroit*, avec *nettoyage des leading zéros*.
\romannumeral0\XINT@mul@M {<n>}<N>\Z\Z\Z

```



## 19 Package *xint* implementation

Fait la multiplication de  $\langle N \rangle$  par  $\langle n \rangle$ , qui est  $< 10000$ .

$\langle N \rangle$  est présenté \*à l'envers\*, sur  $*4n*$ .

```

1789 \def\XINT@mul@M #1%
1790 {%
1791   \expandafter\XINT@mul@M@checkifzeroorone\expandafter{\the\numexpr #1}%
1792 }%
1793 \def\XINT@mul@M@checkifzeroorone #1%
1794 {%
1795   \ifcase #1
1796     \expandafter\XINT@mul@M@zero
1797   \or
1798     \expandafter\XINT@mul@M@one
1799   \else
1800     \expandafter\XINT@mul@N
1801   \fi
1802   {0000}{\#1}%
1803 }%
1804 \def\XINT@mul@M@zero #1\Z\Z\Z\Z { 0}%
1805 \def\XINT@mul@M@one #1#2#3#4\Z\Z\Z\Z
1806 {%
1807   \expandafter\xint@cleanupzeros@andstop\romannumeral0\XINT@rev{\#4}%
1808 }%
1809 \def\XINT@mul@N #1#2#3#4#5#6#7%
1810 {%
1811   \xint@z #4\xint@mul@p\Z\XINT@mul@P {\#1}{\#3}{\#7#6#5#4}{\#2}{\#3}%
1812 }%
1813 \def\XINT@mul@P #1#2#3%
1814 {%
1815   \expandafter\XINT@mul@L\the\numexpr 10000#1+#2*#3\relax
1816 }%
1817 \def\XINT@mul@L 1#1#2#3#4#5#6#7#8#9%
1818 {%
1819   \XINT@mul@N {\#1#2#3#4}{\#5#6#7#8#9}%
1820 }%
1821 \def\xint@mul@p\Z\XINT@mul@P #1#2#3#4#5%
1822 {%
1823   \XINT@mul@M@end #1#4%
1824 }%
1825 \def\XINT@mul@M@end #1#2#3#4#5#6#7#8%
1826 {%
1827   \expandafter\space\the\numexpr #1#2#3#4#5#6#7#8\relax
1828 }%

```

Routine de multiplication principale

délimiteur  $\backslash W \backslash X \backslash Y \backslash Z$

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

$\backslash \text{romannumeral0} \backslash \text{XINT@mul@enter} \langle N1 \rangle \backslash W \backslash X \backslash Y \backslash Z \langle N2 \rangle \backslash W \backslash X \backslash Y \backslash Z$

avec  $\langle N1 \rangle$  \*renversé\*, \*longueur  $4n$ \* (zéros éventuellement ajoutés

## 19 Package *xint* implementation

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

```

1829 \def\XINT@mul@enter #1\W\X\Y\Z #2#3#4#5%
1830 {%
1831     \xint@w
1832     #5\xint@mul@enterw
1833     #4\xint@mul@enterx
1834     #3\xint@mul@entery
1835     #2\xint@mul@enterz
1836     \W\XINT@mul@start {#2#3#4#5}#1\W\X\Y\Z
1837 }%
1838 \def\xint@mul@enterw
1839     #1\xint@mul@enterx
1840     #2\xint@mul@entery
1841     #3\xint@mul@enterz
1842     \W\XINT@mul@start #4#5\W\X\Y\Z \X\Y\Z
1843 {%
1844     \XINT@mul@M {#3#2#1}#5\Z\Z\Z\Z
1845 }%
1846 \def\xint@mul@enterx
1847     #1\xint@mul@entery
1848     #2\xint@mul@enterz
1849     \W\XINT@mul@start #3#4\W\X\Y\Z \Y\Z
1850 {%
1851     \XINT@mul@M {#2#1}#4\Z\Z\Z\Z
1852 }%
1853 \def\xint@mul@entery
1854     #1\xint@mul@enterz
1855     \W\XINT@mul@start #2#3\W\X\Y\Z \Z
1856 {%
1857     \XINT@mul@M {#1}#3\Z\Z\Z\Z
1858 }%
1859 \def\XINT@mul@start #1#2\W\X\Y\Z
1860 {%
1861     \expandafter\XINT@mul@main\expandafter
1862     {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z}#2\W\X\Y\Z
1863 }%
1864 \def\XINT@mul@main #1#2\W\X\Y\Z #3#4#5#6%
1865 {%
1866     \xint@w
1867     #6\xint@mul@mainw
1868     #5\xint@mul@mainx
1869     #4\xint@mul@mainy
1870     #3\xint@mul@mainz
1871     \W\XINT@mul@compute {#1}{#3#4#5#6}#2\W\X\Y\Z
1872 }%
1873 \def\XINT@mul@compute #1#2#3\W\X\Y\Z
1874 {%

```

## 19 Package *xint* implementation

```

1875 \expandafter\XINT@mul@main\expandafter
1876 {\romannumeral0\expandafter
1877 \XINT@mul@Ar\expandafter0\expandafter{\expandafter}%
1878 \romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z
1879 \W\X\Y\Z 0000#1\W\X\Y\Z }#3\W\X\Y\Z
1880 }%

```

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.

```

1881 \def\xint@mul@mainw
1882   #1\xint@mul@mainx
1883   #2\xint@mul@mainy
1884   #3\xint@mul@mainz
1885   \W\XINT@mul@compute #4#5#6\W\X\Y\Z \X\Y\Z
1886 {%
1887   \expandafter\XINT@addm@A \expandafter0\expandafter{\expandafter}%
1888   \romannumeral0%
1889   \XINT@mul@Mr {#3#2#1}#6\Z\Z\Z\Z
1890   \W\X\Y\Z 000#4\W\X\Y\Z
1891 }%
1892 \def\xint@mul@mainx
1893   #1\xint@mul@mainy
1894   #2\xint@mul@mainz
1895   \W\XINT@mul@compute #3#4#5\W\X\Y\Z \Y\Z
1896 {%
1897   \expandafter\XINT@addm@A\expandafter
1898   0\expandafter{\expandafter}%
1899   \romannumeral0\XINT@mul@Mr {#2#1}#5\Z\Z\Z\Z
1900   \W\X\Y\Z 00#3\W\X\Y\Z
1901 }%
1902 \def\xint@mul@mainy
1903   #1\xint@mul@mainz
1904   \W\XINT@mul@compute #2#3#4\W\X\Y\Z \Z
1905 {%
1906   \expandafter\XINT@addm@A\expandafter
1907   0\expandafter{\expandafter}%
1908   \romannumeral0\XINT@mul@Mr {#1}#4\Z\Z\Z\Z
1909   \W\X\Y\Z 0#2\W\X\Y\Z
1910 }%
1911 \def\xint@mul@mainz\W\XINT@mul@compute #1#2#3\W\X\Y\Z
1912 {%
1913   \expandafter\xint@cleanupzeros@andstop\romannumeral0\XINT@rev{#1}%
1914 }%

```

Variante de la Multiplication

```
\romannumeral0\XINT@mulr@enter <N1>\W\X\Y\Z <N2>\W\X\Y\Z
```

## 19 Package *xint* implementation

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 et aussi par la division.

```

1915 \def\XINT@mulr@enter #1\W\X\Y\Z #2#3#4#5%
1916 {%
1917   \xint@w
1918   #5\xint@mulr@enterw
1919   #4\xint@mulr@enterx
1920   #3\xint@mulr@entery
1921   #2\xint@mulr@enterz
1922   \W\XINT@mulr@start {#2#3#4#5}#1\W\X\Y\Z
1923 }%
1924 \def\xint@mulr@enterw
1925   #1\xint@mulr@enterx
1926   #2\xint@mulr@entery
1927   #3\xint@mulr@enterz
1928   \W\XINT@mulr@start #4#5\W\X\Y\Z \X\Y\Z
1929 {%
1930   \XINT@mul@Mr {#3#2#1}#5\Z\Z\Z\Z
1931 }%
1932 \def\xint@mulr@enterx
1933   #1\xint@mulr@entery
1934   #2\xint@mulr@enterz
1935   \W\XINT@mulr@start #3#4\W\X\Y\Z \Y\Z
1936 {%
1937   \XINT@mul@Mr {#2#1}#4\Z\Z\Z\Z
1938 }%
1939 \def\xint@mulr@entery
1940   #1\xint@mulr@enterz
1941   \W\XINT@mulr@start #2#3\W\X\Y\Z \Z
1942 {%
1943   \XINT@mul@Mr {#1}#3\Z\Z\Z\Z
1944 }%
1945 \def\XINT@mulr@start #1#2\W\X\Y\Z
1946 {%
1947   \expandafter\XINT@mulr@main\expandafter
1948   {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z }#2\W\X\Y\Z
1949 }%
1950 \def\XINT@mulr@main #1#2\W\X\Y\Z #3#4#5#6%
1951 {%
1952   \xint@w
1953   #6\xint@mulr@mainw
1954   #5\xint@mulr@mainx
1955   #4\xint@mulr@mainy
1956   #3\xint@mulr@mainz
1957   \W\XINT@mulr@compute {#1}{#3#4#5#6}#2\W\X\Y\Z
1958 }%
1959 \def\XINT@mulr@compute #1#2#3\W\X\Y\Z

```

## 19 Package *xint* implementation

```

1960 {%
1961   \expandafter\XINT@mulr@main\expandafter
1962   {\romannumeral0\expandafter
1963     \XINT@mul@Ar \expandafter0\expandafter{\expandafter}%
1964     \romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z \W\X\Y\Z 0000#1\W\X\Y\Z
1965     }#3\W\X\Y\Z
1966 }%
1967 \def\xint@mulr@mainw
1968   #1\xint@mulr@mainx
1969   #2\xint@mulr@mainy
1970   #3\xint@mulr@mainz
1971   \W\XINT@mulr@compute #4#5#6\W\X\Y\Z \X\Y\Z
1972 {%
1973   \expandafter\XINT@addp@A
1974   \expandafter0\expandafter{\expandafter}%
1975   \romannumeral0\XINT@mul@Mr {#3#2#1}#6\Z\Z\Z\Z
1976   \W\X\Y\Z 000#4\W\X\Y\Z
1977 }%
1978 \def\xint@mulr@mainx
1979   #1\xint@mulr@mainy
1980   #2\xint@mulr@mainz
1981   \W\XINT@mulr@compute #3#4#5\W\X\Y\Z \Y\Z
1982 {%
1983   \expandafter\XINT@addp@A
1984   \expandafter0\expandafter{\expandafter}%
1985   \romannumeral0\XINT@mul@Mr {#2#1}#5\Z\Z\Z\Z
1986   \W\X\Y\Z 00#3\W\X\Y\Z
1987 }%
1988 \def\xint@mulr@mainy
1989   #1\xint@mulr@mainz
1990   \W\XINT@mulr@compute #2#3#4\W\X\Y\Z \Z
1991 {%
1992   \expandafter\XINT@addp@A
1993   \expandafter0\expandafter{\expandafter}%
1994   \romannumeral0\XINT@mul@Mr {#1}#4\Z\Z\Z\Z
1995   \W\X\Y\Z 0#2\W\X\Y\Z
1996 }%
1997 \def\xint@mulr@mainz\W\XINT@mulr@compute #1#2#3\W\X\Y\Z { #1}%

```

### 19.27 \xintSqr

```

1998 \def\xintiSqr {\romannumeral0\xintisqr }%
1999 \def\xintisqr #1%
2000 {%
2001   \expandafter\XINT@sqr\expandafter {\romannumeral0\xintiabs{#1}}%
2002 }%
2003 \let\xintSqr\xintiSqr \let\xintsqr\xintisqr
2004 \def\XINT@sqr #1%
2005 {%

```

```

2006 \expandafter\XINT@mul@enter
2007 \romannumeral0%
2008 \XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
2009 \W\X\Y\Z #1\W\X\Y\Z
2010 }%

```

## 19.28 \xintPrd, \xintProductExpr

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

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

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

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

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

```

2011 \def\xintiPrd {\romannumeral0\xintiprd }%
2012 \def\xintiprd #1{\xintiprodexpr #1\relax }%
2013 \let\xintPrd\xintiPrd
2014 \let\xintprd\xintiprd
2015 \def\xintiProductExpr {\romannumeral0\xintiprodexpr }%
2016 \def\xintiprodexpr {\expandafter\XINT@productexpr\romannumeral-‘0}%
2017 \let\xintProductExpr\xintiProductExpr
2018 \let\xintprodexpr\xintiprodexpr
2019 \def\XINT@productexpr {\XINT@prod@loop@a 1\Z }%
2020 \def\XINT@prod@loop@a #1\Z #2%
2021 {%
2022 \expandafter\XINT@prod@loop@b \romannumeral-‘0#2\Z #1\Z \Z
2023 }%
2024 \def\XINT@prod@loop@b #1%
2025 {%
2026 \xint@relax #1\XINT@prod@finished\relax
2027 \XINT@prod@loop@c #1%
2028 }%
2029 \def\XINT@prod@loop@c
2030 {%
2031 \expandafter\XINT@prod@loop@a\romannumeral0\XINT@mul@fork
2032 }%
2033 \def\XINT@prod@finished #1\Z #2\Z \Z { #2}%

```

**19.29 \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\XINT@Length {#1}>9` but for the time being I leave things as they stand. With release 1.05, rather than using `\XINT@Length` 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).

```

2034 \def\xintFac {\romannumeral0\xintfac }%
2035 \def\xintfac #1%
2036 {%
2037   \expandafter\XINT@fac@fork\expandafter{\the\numexpr #1}%
2038 }%
2039 \def\XINT@fac@fork #1%
2040 {%
2041   \ifcase\XINT@Sgn {#1}
2042     \xint@afterfi{\expandafter\space\expandafter 1\xint@gobble }%
2043   \or
2044     \expandafter\XINT@fac@checklength
2045   \else
2046     \xint@afterfi{\expandafter\xintError:FactorialOfNegativeNumber
2047                   \expandafter\space\expandafter 1\xint@gobble }%
2048   \fi
2049   {#1}%
2050 }%
2051 \def\XINT@fac@checklength #1%
2052 {%
2053   \ifnum #1>999999
2054     \xint@afterfi{\expandafter\xintError:FactorialOfTooBigNumber
2055                   \expandafter\space\expandafter 1\xint@gobble }%
2056   \else
2057     \xint@afterfi{\ifnum #1>9999
2058                   \expandafter\XINT@fac@big@loop
2059                   \else
2060                     \expandafter\XINT@fac@loop
2061                   \fi }%
2062   \fi
2063   {#1}%
2064 }%
2065 \def\XINT@fac@big@loop #1{\XINT@fac@big@loop@main {10000}{#1}{}}%
2066 \def\XINT@fac@big@loop@main #1#2#3%
2067 {%
2068   \ifnum #1<#2
2069     \expandafter
2070       \XINT@fac@big@loop@main
2071     \expandafter
2072       {\the\numexpr #1+1\expandafter }%
2073   \else

```

```

2074      \expandafter\XINT@fac@big@docomputation
2075      \fi
2076      {#2}{#3{#1}}%
2077 }%
2078 \def\XINT@fac@big@docomputation #1#2%
2079 {%
2080     \expandafter \XINT@fac@bigcompute@loop \expandafter
2081     {\romannumeral0\XINT@fac@loop {9999}}#2\relax
2082 }%
2083 \def\XINT@fac@bigcompute@loop #1#2%
2084 {%
2085     \xint@relax #2\XINT@fac@bigcompute@end\relax
2086     \expandafter\XINT@fac@bigcompute@loop\expandafter
2087     {\expandafter\XINT@mul@enter
2088     \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
2089     \W\X\Y\Z #1\W\X\Y\Z }%
2090 }%
2091 \def\XINT@fac@bigcompute@end #1#2#3#4#5%
2092 {%
2093     \XINT@fac@bigcompute@end@ #5%
2094 }%
2095 \def\XINT@fac@bigcompute@end@ #1\R #2\Z \W\X\Y\Z #3\W\X\Y\Z { #3}%
2096 \def\XINT@fac@loop #1{\XINT@fac@loop@main 1{1000}{#1}}%
2097 \def\XINT@fac@loop@main #1#2#3%
2098 {%
2099     \ifnum #3>#1
2100     \else
2101         \expandafter\XINT@fac@loop@exit
2102     \fi
2103     \expandafter\XINT@fac@loop@main\expandafter
2104     {\the\numexpr #1+1\expandafter }\expandafter
2105     {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z }%
2106     {#3}%
2107 }%
2108 \def\XINT@fac@loop@exit #1#2#3#4#5#6#7%
2109 {%
2110     \XINT@fac@loop@exit@ #6%
2111 }%
2112 \def\XINT@fac@loop@exit@ #1#2#3%
2113 {%
2114     \XINT@mul@M
2115 }%

```

### 19.30 \xintPow

1.02 modified the \XINT@posprod routine, and this meant that the original version was moved here and renamed to \XINT@pow@posprod, as it was well adapted for computing powers. Then I moved in 1.03 the special variants of multiplication (hence of addition) which were needed to earlier in this file.



## 19 Package *xint* implementation

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

```

2116 \def\xintiPow {\romannumeral0\xintipow}%
2117 \def\xintipow #1%
2118 {%
2119   \expandafter\xint@pow\romannumeral-'\0#1\Z%
2120}%
2121 \let\xintPow\xintiPow \let\xintpow\xintipow
2122 \def\xint@pow #1#2\Z
2123 {%
2124   \xint@UDsignfork
2125     #1\dummy \XINT@pow@Aneg
2126     -\dummy \XINT@pow@Anonneg
2127   \xint@UDkrof
2128     #1{#2}%
2129}%
2130 \def\XINT@pow@Aneg #1#2#3%
2131 {%
2132   \expandafter\XINT@pow@Aneg@\expandafter{\the\numexpr #3}{#2}%
2133}%
2134 \def\XINT@pow@Aneg@ #1%
2135 {%
2136   \ifodd #1
2137     \expandafter\XINT@pow@Aneg@Bodd
2138   \fi
2139   \XINT@pow@Anonneg@ {#1}%
2140}%
2141 \def\XINT@pow@Aneg@Bodd #1%
2142 {%
2143   \expandafter\XINT@opp\romannumeral0\XINT@pow@Anonneg@
2144}%

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

2145 \def\XINT@pow@Anonneg #1#2#3%
2146 {%
2147   \expandafter\XINT@pow@Anonneg@\expandafter {\the\numexpr #3}{#1#2}%
2148}%

  #1 = B, #2 = |A|

2149 \def\XINT@pow@Anonneg@ #1#2%
2150 {%
2151   \ifcase\XINT@Cmp {#2}{1}
2152     \expandafter\XINT@pow@AisOne
2153   \or
2154     \expandafter\XINT@pow@AatleastTwo
2155   \else
2156     \expandafter\XINT@pow@AisZero

```

## 19 Package *xint* implementation

```

2157 \fi
2158 {#1}{#2}%
2159 }%
2160 \def\XINT@pow@AisOne #1#2{ 1}%

#1 = B

2161 \def\XINT@pow@AisZero #1#2%
2162 {%
2163 \ifcase\XINT@Sgn {#1}
2164 \xint@afterfi { 1}%
2165 \or
2166 \xint@afterfi { 0}%
2167 \else
2168 \xint@afterfi {\xintError:DivisionByZero\space 0}%
2169 \fi
2170 }%
2171 \def\XINT@pow@AatleastTwo #1%
2172 {%
2173 \ifcase\XINT@Sgn {#1}
2174 \expandafter\XINT@pow@BisZero
2175 \or
2176 \expandafter\XINT@pow@checkBlength
2177 \else
2178 \expandafter\XINT@pow@BisNegative
2179 \fi
2180 {#1}%
2181 }%
2182 \def\XINT@pow@BisNegative #1#2{\xintError:FractionRoundedToZero\space 0}%
2183 \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.

2184 \def\XINT@pow@checkBlength #1#2%
2185 {%
2186 \ifnum #1>999999999
2187 \expandafter\XINT@pow@BtooBig
2188 \else
2189 \expandafter\XINT@pow@loop
2190 \fi
2191 {#1}{#2}\XINT@pow@posprod
2192 \xint@UNDEF
2193 \xint@undef\xint@undef\xint@undef\xint@undef
2194 \xint@undef\xint@undef\xint@undef\xint@undef
2195 \xint@UNDEF
2196 }%
2197 \def\XINT@pow@BtooBig #1\xint@UNDEF #2\xint@UNDEF
2198 {\xintError:ExponentTooBig\space 0}%

```

## 19 Package *xint* implementation

```

2199 \def\XINT@pow@loop #1#2%
2200 {%
2201   \ifnum #1 = 1
2202     \expandafter\XINT@pow@loop@end
2203   \else
2204     \xint@afterfi{\expandafter\XINT@pow@loop@a
2205       \expandafter{\the\numexpr 2*(#1/2)-#1\expandafter }% b mod 2
2206       \expandafter{\the\numexpr #1-#1/2\expandafter }%      [b/2]
2207       \expandafter{\romannumeral0\xintisqr{#2}}}%
2208   \fi
2209   {{#2}}}%
2210 }%
2211 \def\XINT@pow@loop@end {\romannumeral0\XINT@rord@main {} \relax }%
2212 \def\XINT@pow@loop@a #1%
2213 {%
2214   \ifnum #1 = 1
2215     \expandafter\XINT@pow@loop
2216   \else
2217     \expandafter\XINT@pow@loop@throwaway
2218   \fi
2219 }%
2220 \def\XINT@pow@loop@throwaway #1#2#3%
2221 {%
2222   \XINT@pow@loop {#1}{#2}%
2223 }%

```

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.

```

2224 \def\XINT@pow@posprod #1%
2225 {%
2226   \XINT@pow@pprod@checkifempty #1\Z
2227 }%
2228 \def\XINT@pow@pprod@checkifempty #1%
2229 {%
2230   \xint@relax #1\XINT@pow@pprod@emptyproduct\relax
2231   \XINT@pow@pprod@RQfirst #1%
2232 }%
2233 \def\XINT@pow@pprod@emptyproduct #1\Z { 1}%
2234 \def\XINT@pow@pprod@RQfirst #1\Z
2235 {%
2236   \expandafter\XINT@pow@pprod@getnext\expandafter
2237   {\romannumeral0\XINT@RQ { }#1\R\R\R\R\R\R\R\R\Z}%
2238 }%
2239 \def\XINT@pow@pprod@getnext #1#2%
2240 {%

```

```

2241 \XINT@pow@pprod@checkiffinished #2\Z {#1}%
2242 }%
2243 \def\XINT@pow@pprod@checkiffinished #1%
2244 {%
2245 \xint@relax #1\XINT@pow@pprod@end\relax
2246 \XINT@pow@pprod@compute #1%
2247 }%
2248 \def\XINT@pow@pprod@compute #1\Z #2%
2249 {%
2250 \expandafter\XINT@pow@pprod@getnext\expandafter
2251 {\romannumeral0\XINT@mulr@enter #2\W\X\Y\Z #1\W\X\Y\Z}%
2252 }%
2253 \def\XINT@pow@pprod@end\relax\XINT@pow@pprod@compute #1\Z #2%
2254 {%
2255 \expandafter\xint@cleanupzeros@andstop
2256 \romannumeral0\XINT@rev {#2}%
2257 }%

```

### 19.31 \xintDivision, \xintQuo, \xintRem

```

2258 \def\xintiQuo {\romannumeral0\xintiquo }%
2259 \def\xintiRem {\romannumeral0\xintirem }%
2260 \def\xintiquo {\expandafter\xint@firstoftwo@andstop
2261 \romannumeral0\xintidivision }%
2262 \def\xintirem {\expandafter\xint@secondoftwo@andstop
2263 \romannumeral0\xintidivision }%
2264 \let\xintQuo\xintiQuo \let\xintquo\xintiquo
2265 \let\xintRem\xintiRem \let\xintrem\xintirem
2266 #1 = A, #2 = B. On calcule le quotient de A par B
2267 1.03 adds the detection of 1 for B.

2266 \def\xintiDivision {\romannumeral0\xintidivision }%
2267 \def\xintidivision #1%
2268 {%
2269 \expandafter\xint@division\expandafter {\romannumeral-‘0#1}%
2270 }%
2271 \let\xintDivision\xintiDivision \let\xintdivision\xintidivision
2272 \def\xint@division #1#2%
2273 {%
2274 \expandafter\XINT@div@fork \romannumeral-‘0#2\Z #1\Z
2275 }%
2276 \def\XINT@Division #1#2{\romannumeral0\XINT@div@fork #2\Z #1\Z }%

#1#2 = 2e input = diviseur = B
#3#4 = 1er input = divisé = A

2277 \def\XINT@div@fork #1#2\Z #3#4\Z
2278 {%
2279 \xint@UDzerofork
2280 #1\dummy \XINT@div@BisZero

```

## 19 Package *xint* implementation

```

2281      #3\dummy \XINT@div@AisZero
2282      0\dummy
2283      {\xint@UDsignfork
2284          #1\dummy \XINT@div@BisNegative % B < 0
2285          #3\dummy \XINT@div@AisNegative % A < 0, B > 0
2286          -\dummy \XINT@div@plusplus % B > 0, A > 0
2287          \xint@UDkrof }%
2288      \xint@UDkrof
2289      {#2}{#4}#1#3% #1#2=B, #3#4=A
2290 }%
2291 \def\XINT@div@BisZero #1#2#3#4{\xintError:DivisionByZero\space {0}{0}}%
2292 \def\XINT@div@AisZero #1#2#3#4{ {0}{0}}%

```

jusqu'à présent c'est facile.

minusplus signifie  $B < 0, A > 0$

plusminus signifie  $B > 0, A < 0$

Ici #3#1 correspond au diviseur B et #4#2 au divisé A

Cases with  $B < 0$  or especially  $A < 0$  are treated sub-optimally in terms of post-processing, things get reversed which could have been produced directly in the wanted order, but  $A, B > 0$  is given priority for optimization.

```

2293 \def\XINT@div@plusplus #1#2#3#4%
2294 {%
2295     \XINT@div@prepare {#3#1}{#4#2}%
2296 }%

```

$B = \#3\#1 < 0$ , A non nul positif ou négatif

```

2297 \def\XINT@div@BisNegative #1#2#3#4%
2298 {%
2299     \expandafter\XINT@div@BisNegative@post
2300     \romannumeral0\XINT@div@fork #1\Z #4#2\Z
2301 }%
2302 \def\XINT@div@BisNegative@post #1%
2303 {%
2304     \expandafter\space\expandafter {\romannumeral0\XINT@opp #1}%
2305 }%

```

$B = \#3\#1 > 0$ ,  $A = -\#2 < 0$

```

2306 \def\XINT@div@AisNegative #1#2#3#4%
2307 {%
2308     \expandafter\XINT@div@AisNegative@post
2309     \romannumeral0\XINT@div@prepare {#3#1}{#2}{#3#1}%
2310 }%
2311 \def\XINT@div@AisNegative@post #1#2%
2312 {%
2313     \ifcase\XINT@Sgn {#2}
2314         \expandafter \XINT@div@AisNegative@zerorem
2315     \or

```

## 19 Package *xint* implementation

```

2316      \expandafter \XINT@div@AisNegative@posrem
2317      \fi
2318      {#1}{#2}%
2319 }%

    en #3 on a une copie de B (à l'endroit)

2320 \def\XINT@div@AisNegative@zerorem #1#2#3%
2321 {%
2322     \expandafter\space\expandafter {\romannumeral0\XINT@opp #1}{0}%
2323 }%

    #1 = quotient, #2 = reste, #3 = diviseur initial (à l'endroit)
    remplace Reste par B - Reste, après avoir remplacé Q par -(Q+1)
    de sorte que la formule  $a = qb + r$ ,  $0 \leq r < |b|$  est valable

2324 \def\XINT@div@AisNegative@posrem #1%
2325 {%
2326     \expandafter \XINT@div@AisNegative@posrem@b \expandafter
2327     {\romannumeral0\xintiopp{\xintiAdd {#1}{1}}}%
2328 }%
2329 \def\XINT@div@AisNegative@posrem@b #1#2#3%
2330 {%
2331     \expandafter \xint@exchangetwo@keepbraces@andstop \expandafter
2332     {\romannumeral0\XINT@sub {#3}{#2}}{#1}%
2333 }%

    par 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
    1.03 adds the interception of  $B=1$ 

2334 \def\XINT@div@prepare #1%
2335 {%
2336     \expandafter \XINT@div@prepareB@aa \expandafter
2337     {\romannumeral0\XINT@length {#1}}{#1}% B  $> 0$  ici
2338 }%
2339 \def\XINT@div@prepareB@aa #1%
2340 {%
2341     \ifnum #1=1
2342         \expandafter\XINT@div@prepareB@ab
2343     \else
2344         \expandafter\XINT@div@prepareB@a
2345     \fi
2346     {#1}%
2347 }%
2348 \def\XINT@div@prepareB@ab #1#2%
2349 {%
2350     \ifnum #2=1
2351         \expandafter\XINT@div@prepareB@BisOne

```

## 19 Package *xint* implementation

```

2352 \else
2353 \expandafter\XINT@div@prepareB@e
2354 \fi {000}{3}{4}{#2}%
2355 }%
2356 \def\XINT@div@prepareB@BisOne #1#2#3#4#5{ {#5}{0}}%
2357 \def\XINT@div@prepareB@a #1%
2358 {%
2359 \expandafter\XINT@div@prepareB@c\expandafter{\the\numexpr 4*((#1+1)/4)}{#1}%
2360 }%

```

#1 = K

```

2361 \def\XINT@div@prepareB@c #1#2%
2362 {%
2363 \ifcase \numexpr #1-#2\relax
2364 \expandafter\XINT@div@prepareB@d
2365 \or
2366 \expandafter\XINT@div@prepareB@di
2367 \or
2368 \expandafter\XINT@div@prepareB@dii
2369 \or
2370 \expandafter\XINT@div@prepareB@diii
2371 \fi {#1}%
2372 }%
2373 \def\XINT@div@prepareB@d {\XINT@div@prepareB@e {}{0}}%
2374 \def\XINT@div@prepareB@di {\XINT@div@prepareB@e {0}{1}}%
2375 \def\XINT@div@prepareB@dii {\XINT@div@prepareB@e {00}{2}}%
2376 \def\XINT@div@prepareB@diii {\XINT@div@prepareB@e {000}{3}}%

```

#1 = zéros à rajouter à B, #2=c, #3=K, #4 = B

```

2377 \def\XINT@div@prepareB@e #1#2#3#4%
2378 {%
2379 \XINT@div@prepareB@f #4#1\Z {#3}{#2}{#1}%
2380 }%

```

x = #1#2#3#4 = 4 premiers chiffres de B. #1 est non nul.  
Ensuite on renverse B pour calculs plus rapides par la suite.

```

2381 \def\XINT@div@prepareB@f #1#2#3#4#5\Z
2382 {%
2383 \expandafter \XINT@div@prepareB@g \expandafter
2384 {\romannumeral0\XINT@rev {#1#2#3#4#5}}{#1#2#3#4}%
2385 }%

```

#3= K, #4 = c, #5= {} ou {0} ou {00} ou {000}, #6 = A initial  
#1 = B préparé et renversé, #2 = x = quatre premiers chiffres  
On multiplie aussi A par 10^c.  
B, x, K, c, {} ou {0} ou {00} ou {000}, A initial

## 19 Package *xint* implementation

```

2386 \def\XINT@div@prepareB@g #1#2#3#4#5#6%
2387 {%
2388   \XINT@div@prepareA@a {#6#5}{#2}{#3}{#1}{#4}%
2389 }%

A, x, K, B, c,

2390 \def\XINT@div@prepareA@a #1%
2391 {%
2392   \expandafter \XINT@div@prepareA@b \expandafter
2393     {\romannumeral0\XINT@length {#1}}{#1}% A >0 ici
2394 }%

L0, A, x, K, B, ...

2395 \def\XINT@div@prepareA@b #1%
2396 {%
2397   \expandafter\XINT@div@prepareA@c\expandafter{\the\numexpr 4*((#1+1)/4)}{#1}%
2398 }%

L, L0, A, x, K, B,...

2399 \def\XINT@div@prepareA@c #1#2%
2400 {%
2401   \ifcase \numexpr #1-#2\relax
2402     \expandafter\XINT@div@prepareA@d
2403   \or
2404     \expandafter\XINT@div@prepareA@di
2405   \or
2406     \expandafter\XINT@div@prepareA@dii
2407   \or
2408     \expandafter\XINT@div@prepareA@diii
2409   \fi {#1}%
2410 }%
2411 \def\XINT@div@prepareA@d      {\XINT@div@prepareA@e {}}%
2412 \def\XINT@div@prepareA@di    {\XINT@div@prepareA@e {0}}%
2413 \def\XINT@div@prepareA@dii   {\XINT@div@prepareA@e {00}}%
2414 \def\XINT@div@prepareA@diii  {\XINT@div@prepareA@e {000}}%

#1#3 = A préparé, #2 = longueur de ce A préparé,

2415 \def\XINT@div@prepareA@e #1#2#3%
2416 {%
2417   \XINT@div@startswitch {#1#3}{#2}%
2418 }%

A, L, x, K, B, c

```



## 19 Package *xint* implementation

```

2419 \def\XINT@div@startswitch #1#2#3#4%
2420 {%
2421   \ifnum #2 > #4
2422     \expandafter\XINT@div@body@a
2423   \else
2424     \ifnum #2 = #4
2425       \expandafter\expandafter\expandafter\XINT@div@final@a
2426     \else
2427       \expandafter\expandafter\expandafter\XINT@div@finished@a
2428     \fi\fi {#1}{#4}{#3}{0000}{#2}%
2429 }%

  A, K, x, Q, L, B, c
  ---- "Finished"

2430 \def\XINT@div@finished@a #1#2#3%
2431 {%
2432   \expandafter\XINT@div@finished@b\expandafter {\romannumeral0\XINT@cuz {#1}}%
2433 }%

  A, Q, L, B, c
  no leading zeros in A at this stage

2434 \def\XINT@div@finished@b #1#2#3#4#5%
2435 {%
2436   \ifcase \XINT@Sgn {#1}
2437     \xint@afterfi {\XINT@div@finished@c {0}}%
2438   \or
2439     \xint@afterfi {\expandafter\XINT@div@finished@c\expandafter
2440                   {\romannumeral0\XINT@dsh@checksignx #5\Z {#1}}%
2441                   }%
2442   \fi
2443   {#2}%
2444 }%
2445 \def\XINT@div@finished@c #1#2%
2446 {%
2447   \expandafter\space\expandafter {\romannumeral0\XINT@rev@andcuz {#2}}{#1}%
2448 }%

  ---- "Final"
  A, K, x, Q, L, B, c

2449 \def\XINT@div@final@a #1%
2450 {%
2451   \XINT@div@final@b #1\Z
2452 }%
2453 \def\XINT@div@final@b #1#2#3#4#5\Z
2454 {%
2455   \xint@quatrezeros #1#2#3#4\xint@div@final@c0000%
2456   \XINT@div@final@c {#1#2#3#4}{#1#2#3#4#5}%
2457 }%
2458 \def\xint@div@final@c0000\XINT@div@final@c #1{\XINT@div@finished@a }%

```

## 19 Package *xint* implementation

a, A, K, x, Q, L, B ,c  
 1.01: code ré-écrit pour optimisations diverses.  
 1.04: again, code rewritten for tiny speed increase (hopefully).

```

2459 \def\XINT@div@final@c #1#2#3#4%
2460 {%
2461   \expandafter \XINT@div@final@da \expandafter
2462   {\the\numexpr #1-(#1/#4)*#4\expandafter }\expandafter
2463   {\the\numexpr #1/#4\expandafter }\expandafter
2464   {\romannumeral0\xint@cleanupzeros@andstop #2}%
2465 }%

r, q, A sans leading zéros, Q, L, B à l'envers sur 4n, c

2466 \def\XINT@div@final@da #1%
2467 {%
2468   \ifnum #1>9
2469     \expandafter\XINT@div@final@dP
2470   \else
2471     \xint@afterfi
2472     {\ifnum #1<0
2473       \expandafter\XINT@div@final@dN
2474       \else
2475       \expandafter\XINT@div@final@db
2476       \fi }%
2477   \fi
2478 }%
2479 \def\XINT@div@final@dN #1%
2480 {%
2481   \expandafter\XINT@div@final@dP\the\numexpr #1-1\relax
2482 }%
2483 \def\XINT@div@final@dP #1#2#3#4#5% q,A,Q,L,B (puis c)
2484 {%
2485   \expandafter \XINT@div@final@f \expandafter
2486   {\romannumeral0\xintisub {#2}%
2487    {\romannumeral0\XINT@mul@M {#1}#5\Z\Z\Z\Z }}%
2488   {\romannumeral0\XINT@add@A 0{#10000\W\X\Y\Z #3\W\X\Y\Z }}%
2489 }%
2490 \def\XINT@div@final@db #1#2#3#4#5% q,A,Q,L,B (puis c)
2491 {%
2492   \expandafter\XINT@div@final@dc\expandafter
2493   {\romannumeral0\xintisub {#2}%
2494    {\romannumeral0\XINT@mul@M {#1}#5\Z\Z\Z\Z }}%
2495   {#1}{#2}{#3}{#4}{#5}%
2496 }%
2497 \def\XINT@div@final@dc #1#2%
2498 {%
2499   \ifnum\XINT@Sgn{#1}<0
2500   \xint@afterfi {\expandafter\XINT@div@final@dP\the\numexpr #2-1\relax}%

```

## 19 Package *xint* implementation

```

2501   \else \xint@afterfi {\XINT@div@final@e {#1}#2}%
2502   \fi
2503 }%
2504 \def\XINT@div@final@e #1#2#3#4#5#6% A final, q, trash, Q, L, B
2505 {%
2506   \XINT@div@final@f {#1}%
2507   {\romannumeral0\XINT@add@A 0}{#2000\W\X\Y\Z #4\W\X\Y\Z }%
2508 }%
2509 \def\XINT@div@final@f #1#2#3% R,Q \'a d\'evelopper,c
2510 {%
2511   \ifcase \XINT@Sgn {#1}
2512     \xint@afterfi {\XINT@div@final@end {0}}%
2513   \or
2514     \xint@afterfi {\expandafter\XINT@div@final@end\expandafter
2515                   {\romannumeral0\XINT@dsh@checksignx #3\Z {#1}}%
2516                   }%
2517   \fi
2518   {#2}%
2519 }%
2520 \def\XINT@div@final@end #1#2%
2521 {%
2522   \expandafter\space\expandafter {#2}{#1}%
2523 }%

  Boucle Principale (on reviendra en div@body@b pas div@body@a)
  A, K, x, Q, L, B, c

2524 \def\XINT@div@body@a #1%
2525 {%
2526   \XINT@div@body@b #1\Z {#1}%
2527 }%
2528 \def\XINT@div@body@b #1#2#3#4#5#6#7#8#9\Z
2529 {%
2530   \XINT@div@body@c {#1#2#3#4#5#6#7#8}%
2531 }%

  a, A, K, x, Q, L, B, c

2532 \def\XINT@div@body@c #1#2#3%
2533 {%
2534   \XINT@div@body@d {#3}{#2\Z {#1}{#3}%
2535 }%
2536 \def\XINT@div@body@d #1#2#3#4#5#6%
2537 {%
2538   \ifnum #1 > 0
2539     \expandafter\XINT@div@body@d
2540     \expandafter{\the\numexpr #1-4\expandafter }%
2541   \else
2542     \expandafter\XINT@div@body@e
2543   \fi

```

## 19 Package *xint* implementation

```

2544 {#6#5#4#3#2}%
2545 }%
2546 \def\XINT@div@body@e #1#2\Z #3%
2547 {%
2548 \XINT@div@body@f {#3}{#1}{#2}%
2549 }%

a, alpha (à l'envers), alpha' (à l'endroit), K, x, Q, L, B (à l'envers), c

2550 \def\XINT@div@body@f #1#2#3#4#5#6#7#8%
2551 {%
2552 \expandafter\XINT@div@body@gg
2553 \the\numexpr (#1+(#5+1)/2)/(#5+1)+99999\relax
2554 {#8}{#2}{#8}{#4}{#5}{#3}{#6}{#7}{#8}%
2555 }%

q1 sur six chiffres (il en a 5 au max), B, alpha, B, K, x, alpha', Q, L, B, c

2556 \def\XINT@div@body@gg #1#2#3#4#5#6%
2557 {%
2558 \xint@UDzerofork
2559 #2\dummy \XINT@div@body@gk
2560 0\dummy {\XINT@div@body@ggk #2}%
2561 \xint@UDkrof
2562 {#3#4#5#6}%
2563 }%
2564 \def\XINT@div@body@gk #1#2#3%
2565 {%
2566 \expandafter\XINT@div@body@h
2567 \romannumeral0\XINT@div@sub@xpxp
2568 {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z }{#3}\Z {#1}%
2569 }%
2570 \def\XINT@div@body@ggk #1#2#3%
2571 {%
2572 \expandafter \XINT@div@body@ggk \expandafter
2573 {\romannumeral0\XINT@mul@Mr {#1}0000#3\Z\Z\Z\Z }%
2574 {\romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z }%
2575 {#1#2}%
2576 }%
2577 \def\XINT@div@body@ggk #1#2#3#4%
2578 {%
2579 \expandafter\XINT@div@body@h
2580 \romannumeral0\XINT@div@sub@xpxp
2581 {\romannumeral0\expandafter\XINT@mul@Ar
2582 \expandafter0\expandafter{\expandafter}#2\W\X\Y\Z #1\W\X\Y\Z }%
2583 {#4}\Z {#3}%
2584 }%

alpha1 = alpha-q1 B, \Z, q1, B, K, x, alpha', Q, L, B, c

```

## 19 Package *xint* implementation

```

2585 \def\XINT@div@body@h #1#2#3#4#5#6#7#8#9\Z
2586 {%
2587   \ifnum #1#2#3#4>0
2588     \xint@afterfi{\XINT@div@body@i {#1#2#3#4#5#6#7#8}}%
2589   \else
2590     \expandafter\XINT@div@body@k
2591   \fi
2592   {#1#2#3#4#5#6#7#8#9}%
2593 }%
2594 \def\XINT@div@body@k #1#2#3%
2595 {%
2596   \XINT@div@body@l {#1}{#2}%
2597 }%

    a1, alpha1 (à l'endroit), q1, B, K, x, alpha', Q, L, B, c

2598 \def\XINT@div@body@i #1#2#3#4#5#6%
2599 {%
2600   \expandafter\XINT@div@body@j
2601   \expandafter{\the\numexpr (#1+(#6+1)/2)/(#6+1)-1}%
2602   {#2}{#3}{#4}{#5}{#6}%
2603 }%
2604 \def\XINT@div@body@j #1#2#3#4%
2605 {%
2606   \expandafter \XINT@div@body@l \expandafter
2607   {\romannumeral0\XINT@div@sub@xpxp
2608     {\romannumeral0\XINT@mul@Mr {#1}#4\Z\Z\Z\Z }{\XINT@Rev{#2}}}%
2609   {#3+#1}%
2610 }%

    alpha2 (à l'endroit, ou alpha1), q1+q2 (ou q1), K, x, alpha', Q, L, B, c

2611 \def\XINT@div@body@l #1#2#3#4#5#6#7%
2612 {%
2613   \expandafter\XINT@div@body@m
2614   \the\numexpr 1000000000+#2\relax {#6}{#3}{#7}{#1#5}{#4}%
2615 }%

    chiffres de q, Q, K, L, A'=nouveau A, x, B, c

2616 \def\XINT@div@body@m #1#2#3#4#5#6#7#8#9%
2617 {%
2618   \ifnum #2#3#4#5>0
2619     \xint@afterfi {\XINT@div@body@n {#9#8#7#6#5#4#3#2}}%
2620   \else
2621     \xint@afterfi {\XINT@div@body@n {#9#8#7#6}}%
2622   \fi
2623 }%

    q renversé, Q, K, L, A', x, B, c

```

## 19 Package *xint* implementation

```

2624 \def\XINT@div@body@n #1#2%
2625 {%
2626     \expandafter\XINT@div@body@o\expandafter
2627     {\romannumeral0\XINT@addr@A 0}{#1\W\X\Y\Z #2\W\X\Y\Z }%
2628 }%

    q+Q, K, L, A', x, B, c

2629 \def\XINT@div@body@o #1#2#3#4%
2630 {%
2631     \XINT@div@body@p {#3}{#2}{}#4\Z {#1}%
2632 }%

    L, K, {}, A'\Z, q+Q, x, B, c

2633 \def\XINT@div@body@p #1#2#3#4#5#6#7%
2634 {%
2635     \ifnum #1 > #2
2636         \xint@afterfi
2637         {\ifnum #4#5#6#7 > 0
2638             \expandafter\XINT@div@body@q
2639             \else
2640             \expandafter\XINT@div@body@repeatp
2641             \fi }%
2642     \else
2643         \expandafter\XINT@div@gotofinal@a
2644     \fi
2645     {#1}{#2}{#3}#4#5#6#7%
2646 }%

    L, K, zeros, A' avec moins de zéros\Z, q+Q, x, B, c

2647 \def\XINT@div@body@repeatp #1#2#3#4#5#6#7%
2648 {%
2649     \expandafter\XINT@div@body@p\expandafter{\the\numexpr #1-4}{#2}{0000#3}%
2650 }%

    L -> L-4, zeros->zeros+0000, répéter jusqu'à ce que soit L=K
    soit on ne trouve plus 0000
    nouveau L, K, zeros, nouveau A=#4, \Z, Q+q (à l'envers), x, B, c

2651 \def\XINT@div@body@q #1#2#3#4\Z #5#6%
2652 {%
2653     \XINT@div@body@b #4\Z {#4}{#2}{#6}{#3#5}{#1}%
2654 }%

    A, K, x, Q, L, B, c --> iterate
    ----
    Boucle Principale achevée
    ATTENTION IL FAUT AJOUTER 4 ZEROS DE MOINS QUE CEUX
    QUI ONT ÉTÉ PRÉPARÉS DANS #3!!
    L, K (L=K), zeros, A\Z, Q, x, B, c

```

## 19 Package *xint* implementation

```

2655 \def\XINT@div@gotofinal@a #1#2#3#4\Z %
2656 {%
2657   \XINT@div@gotofinal@b #3\Z {#4}{#1}%
2658 }%
2659 \def\XINT@div@gotofinal@b 0000#1\Z #2#3#4#5%
2660 {%
2661   \XINT@div@final@a {#2}{#3}{#5}{#1#4}{#3}%
2662 }%

```

La soustraction spéciale.

Elle fait l'expansion (une fois pour le premier, deux fois pour le second) de ses arguments. Ceux-ci doivent être à l'envers sur 4n. De plus on sait a priori que le second est > le premier. Et le résultat de la différence est renvoyé **\*\*avec la même longueur que le second\*\*** (donc avec des leading zéros éventuels), et \*à l'endroit\*.

```

2663 \def\XINT@div@sub@xpxp #1%
2664 {%
2665   \expandafter \XINT@div@sub@xpxp@ \expandafter{#1}%
2666 }%
2667 \def\XINT@div@sub@xpxp@ #1#2%
2668 {%
2669   \expandafter\expandafter\expandafter\XINT@div@sub@xpxp@@
2670   #2\W\X\Y\Z #1\W\X\Y\Z
2671 }%
2672 \def\XINT@div@sub@xpxp@@
2673 {%
2674   \XINT@div@sub@A 1}%
2675 }%
2676 \def\XINT@div@sub@A #1#2#3#4#5#6%
2677 {%
2678   \xint@w #3\xint@div@sub@az\W
2679   \XINT@div@sub@B #1{#3#4#5#6}{#2}%
2680 }%
2681 \def\XINT@div@sub@B #1#2#3#4\W\X\Y\Z #5#6#7#8%
2682 {%
2683   \xint@w #5\xint@div@sub@bz\W
2684   \XINT@div@sub@onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
2685 }%
2686 \def\XINT@div@sub@onestep #1#2#3#4#5#6%
2687 {%
2688   \expandafter\XINT@div@sub@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
2689 }%
2690 \def\XINT@div@sub@backtoA #1#2#3.#4%
2691 {%
2692   \XINT@div@sub@A #2{#3#4}%
2693 }%
2694 \def\xint@div@sub@bz\W\XINT@div@sub@onestep #1#2#3#4#5#6#7%
2695 {%
2696   \xint@UDzerofork

```

## 19 Package *xint* implementation

```

2697      #1\dummy \XINT@div@sub@C %
2698      0\dummy \XINT@div@sub@D % pas de retenue
2699      \xint@UDkrof
2700      {#7}#2#3#4#5%
2701 }%
2702 \def\XINT@div@sub@D #1#2\W\X\Y\Z
2703 {%
2704     \expandafter\space
2705     \romannumeral0%
2706     \XINT@rord@main {}#2%
2707     \xint@UNDEF
2708     \xint@undef\xint@undef\xint@undef\xint@undef
2709     \xint@undef\xint@undef\xint@undef\xint@undef
2710     \xint@UNDEF
2711     #1%
2712 }%
2713 \def\XINT@div@sub@C #1#2#3#4#5%
2714 {%
2715     \xint@w #2\xint@div@sub@cz\W
2716     \XINT@div@sub@AC@onestep {#5#4#3#2}{#1}%
2717 }%
2718 \def\XINT@div@sub@AC@onestep #1%
2719 {%
2720     \expandafter\XINT@div@sub@backtoC\the\numexpr 11#1-1\relax.%
2721 }%
2722 \def\XINT@div@sub@backtoC #1#2#3.#4%
2723 {%
2724     \XINT@div@sub@AC@checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
2725 }%
2726 \def\XINT@div@sub@AC@checkcarry #1%
2727 {%
2728     \xint@one #1\xint@div@sub@AC@nocarry 1\XINT@div@sub@C
2729 }%
2730 \def\xint@div@sub@AC@nocarry 1\XINT@div@sub@C #1#2\W\X\Y\Z
2731 {%
2732     \expandafter\space
2733     \romannumeral0%
2734     \XINT@rord@main {}#2%
2735     \xint@UNDEF
2736     \xint@undef\xint@undef\xint@undef\xint@undef
2737     \xint@undef\xint@undef\xint@undef\xint@undef
2738     \xint@UNDEF
2739     #1%
2740 }%
2741 \def\xint@div@sub@cz\W\XINT@div@sub@AC@onestep #1#2{ #2}%
2742 \def\xint@div@sub@az\W\XINT@div@sub@B #1#2#3#4\Z { #3}%

```

-----  
-----



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

### 19.32 **\xintFDg**

FIRST DIGIT. Code simplified in 1.05. And prepared for redefinition by  
xintfrac to parse through \xintNum

```

2743 \def\xintFDg {\romannumeral0\xintifdg }%
2744 \def\xintifdg #1%
2745 {%
2746   \expandafter\XINT@fdg \romannumeral-‘0#1\W\Z
2747 }%
2748 \let\xintFDg\xintifdg \let\xintfdg\xintifdg
2749 \def\XINT@FDg #1{\romannumeral0\XINT@fdg #1\W\Z }%
2750 \def\XINT@fdg #1#2#3\Z
2751 {%
2752   \xint@UDzerominusfork
2753   #1-\dummy { 0}% zero
2754   0#1\dummy { #2}% negative
2755   0-\dummy { #1}% positive
2756   \xint@UDkrof
2757 }%
```

### 19.33 **\xintLDg**

LAST DIGIT. Simplified in 1.05. And prepared for extension by xintfrac  
to parse through \xintNum

```

2758 \def\xintLDg {\romannumeral0\xintildg }%
2759 \def\xintildg #1%
2760 {%
2761   \expandafter\XINT@ldg\expandafter {\romannumeral-‘0#1}%
2762 }%
2763 \let\xintLDg\xintildg \let\xintldg\xintildg
2764 \def\XINT@LDg #1{\romannumeral0\XINT@ldg {#1}}%
2765 \def\XINT@ldg #1%
2766 {%
2767   \expandafter\XINT@ldg@\romannumeral0\XINT@rev {#1}\Z
2768 }%
2769 \def\XINT@ldg@ #1#2\Z{ #1}%
```

### 19.34 **\xintMON**

MINUS ONE TO THE POWER N

```

2770 \def\xintiMON {\romannumeral0\xintimon }%
2771 \def\xintimon #1%
2772 {%
2773   \ifodd\xintiLDg {#1}
2774     \xint@afterfi{ -1}%
2775   \else
2776     \xint@afterfi{ 1}%
2777   \fi
2778 }%
2779 \def\xintiMMON {\romannumeral0\xintimmon }%
2780 \def\xintimmon #1%
2781 {%
2782   \ifodd\xintiLDg {#1}
2783     \xint@afterfi{ 1}%
2784   \else
2785     \xint@afterfi{ -1}%
2786   \fi
2787 }%
2788 \let\xintMON\xintiMON \let\xintmon\xintimon
2789 \let\xintMMON\xintiMMON \let\xintmmon\xintimmon

```

### 19.35 \xintOdd

ODDNESS. 1.05 defines \xintiOdd, so \xintOdd can be modified by xintfrac to parse through \xintNum.

```

2790 \def\xintiOdd {\romannumeral0\xintiodd }%
2791 \def\xintiodd #1%
2792 {%
2793   \ifodd\xintiLDg{#1}
2794     \xint@afterfi{ 1}%
2795   \else
2796     \xint@afterfi{ 0}%
2797   \fi
2798 }%
2799 \def\XINT@Odd #1%
2800 {\romannumeral0%
2801   \ifodd\XINT@LDg{#1}
2802     \xint@afterfi{ 1}%
2803   \else
2804     \xint@afterfi{ 0}%
2805   \fi
2806 }%
2807 \let\xintOdd\xintiOdd \let\xintodd\xintiodd

```

### 19.36 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

## 19 Package *xint* implementation

```
2808 \def\xintDSL {\romannumeral0\xintdsl }%
2809 \def\xintdsl #1%
2810 {%
2811   \expandafter\XINT@dsl \romannumeral-‘0#1\Z
2812 }%
2813 \def\XINT@DSL #1{\romannumeral0\XINT@dsl #1\Z }%
2814 \def\XINT@dsl #1%
2815 {%
2816   \xint@zero #1\xint@dsl@zero 0\XINT@dsl@ #1%
2817 }%
2818 \def\xint@dsl@zero 0\XINT@dsl@ 0#1\Z { 0}%
2819 \def\XINT@dsl@ #1\Z { #10}%
```

### 19.37 \xintDSR

DECIMAL SHIFT RIGHT (=DIVISION PAR 10)

```
2820 \def\xintDSR {\romannumeral0\xintdsr }%
2821 \def\xintdsr #1%
2822 {%
2823   \expandafter\XINT@dsr@a\expandafter {\romannumeral-‘0#1}\W\Z
2824 }%
2825 \def\XINT@DSR #1{\romannumeral0\XINT@dsr@a {#1}\W\Z }%
2826 \def\XINT@dsr@a
2827 {%
2828   \expandafter\XINT@dsr@b\romannumeral0\XINT@rev
2829 }%
2830 \def\XINT@dsr@b #1#2#3\Z
2831 {%
2832   \xint@w #2\xint@dsr@onedigit\W
2833   \xint@minus #2\xint@dsr@onedigit-%
2834   \expandafter\XINT@dsr@removew
2835   \romannumeral0\XINT@rev {#2#3}%
2836 }%
2837 \def\xint@dsr@onedigit #1\XINT@rev #2{ 0}%
2838 \def\XINT@dsr@removew #1\W { }%
```

### 19.38 \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 revise the legacy code on another occasion.

```

2839 \def\xintDSHr {\romannumeral0\xintdshr}%
2840 \def\xintdshr #1%
2841 {%
2842   \expandafter\XINT@dshr@checkxpositive \the\numexpr #1\relax\Z
2843}%
2844 \def\XINT@dshr@checkxpositive #1%
2845 {%
2846   \xint@UDzerominusfork
2847   0#1\dummy \XINT@dshr@xzeroorneg
2848   #1-\dummy \XINT@dshr@xzeroorneg
2849   0-\dummy \XINT@dshr@xpositive
2850   \xint@UDkrof #1%
2851}%
2852 \def\XINT@dshr@xzeroorneg #1\Z #2{ 0}%
2853 \def\XINT@dshr@xpositive #1\Z
2854 {%
2855   \expandafter\xint@secondoftwo@andstop\romannumeral0\xintdsx {#1}%
2856}%
2857 \def\xintDSH {\romannumeral0\xintdsh}%
2858 \def\xintdsh #1#2%
2859 {%
2860   \expandafter\xint@dsh\expandafter {\romannumeral-‘0#2’}{#1}%
2861}%
2862 \def\xint@dsh #1#2%
2863 {%
2864   \expandafter\XINT@dsh@checksignx \the\numexpr #2\relax\Z {#1}%
2865}%
2866 \def\XINT@dsh@checksignx #1%
2867 {%
2868   \xint@UDzerominusfork
2869   #1-\dummy \XINT@dsh@xiszero
2870   0#1\dummy \XINT@dsx@xisNeg@checkA      % on passe direct dans DSx
2871   0-\dummy {\XINT@dsh@xisPos #1}%
2872   \xint@UDkrof
2873}%
2874 \def\XINT@dsh@xiszero #1\Z #2{ #2}%
2875 \def\XINT@dsh@xisPos #1\Z #2%
2876 {%
2877   \expandafter\xint@firstoftwo@andstop
2878   \romannumeral0\XINT@dsx@checksignA #2\Z {#1}% via DSx
2879}%

```

### 19.39 \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 <--

## 19 Package *xint* implementation

```

si x < 0, fait A -> A.10^(|x|)
si x >= 0, et A >=0, fait A -> {quo(A,10^(x))}{rem(A,10^(x))}
si x >= 0, et A < 0, d'abord on calcule {quo(-A,10^(x))}{rem(-A,10^(x))}
    puis, si le premier n'est pas nul on lui donne le signe -
        si le premier est nul on donne le signe - au second.
On peut donc toujours reconstituer l'original A par 10^x Q \pm R
où il faut prendre le signe plus si Q est positif ou nul et le signe moins si
Q est strictement négatif.
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.

2880 \def\xintDSx {\romannumeral0\xintdsx}%
2881 \def\xintdsx #1#2%
2882 {%
2883     \expandafter\xint@dsx\expandafter {\romannumeral-‘0#2}{#1}%
2884}%
2885 \def\xint@dsx #1#2%
2886 {%
2887     \expandafter\XINT@dsx@checksignx \the\numexpr #2\relax\Z {#1}%
2888}%
2889 \def\XINT@DSx #1#2{\romannumeral0\XINT@dsx@checksignx #1\Z {#2}}%
2890 \def\XINT@dsx #1#2{\XINT@dsx@checksignx #1\Z {#2}}%
2891 \def\XINT@dsx@checksignx #1%
2892 {%
2893     \xint@UDzerominusfork
2894     #1-\dummy \XINT@dsx@xisZero
2895     0#1\dummy \XINT@dsx@xisNeg@checkA
2896     0-\dummy {\XINT@dsx@xisPos #1}%
2897     \xint@UDkrof
2898}%
2899 \def\XINT@dsx@xisZero #1\Z #2{ {#2}{0}}% attention comme x > 0
2900 \def\XINT@dsx@xisNeg@checkA #1\Z #2%
2901 {%
2902     \XINT@dsx@xisNeg@checkA@ #2\Z {#1}%
2903}%
2904 \def\XINT@dsx@xisNeg@checkA@ #1#2\Z #3%
2905 {%
2906     \xint@zero #1\XINT@dsx@xisNeg@Azero 0%
2907     \XINT@dsx@xisNeg@checkx {#3}{#3}\Z {#1#2}%
2908}%
2909 \def\XINT@dsx@xisNeg@Azero #1\Z #2{ 0}%
2910 \def\XINT@dsx@xisNeg@checkx #1%
2911 {%
2912     \ifnum #1> 999999999
2913         \xint@afterfi
2914         {\xintError:TooBigDecimalShift
2915         \expandafter\space\expandafter 0\xint@gobble@iii}%
2916     \else
2917         \expandafter \XINT@dsx@zeroloop

```

## 19 Package *xint* implementation

```

2918   \fi
2919 }%
2920 \def\XINT@dsx@zeroloop #1%
2921 {%
2922   \ifnum #1<9 \XINT@dsx@exita\fi
2923   \expandafter\XINT@dsx@zeroloop\expandafter
2924     {\the\numexpr #1-8}00000000%
2925 }%
2926 \def\XINT@dsx@exita\fi\expandafter\XINT@dsx@zeroloop
2927 {%
2928   \fi\expandafter\XINT@dsx@exitb
2929 }%
2930 \def\XINT@dsx@exitb #1%
2931 {%
2932   \expandafter\expandafter\expandafter
2933     \XINT@dsx@addzeros\cname xint@gobble@\romannumeral -#1\endcsname
2934 }%
2935 \def\XINT@dsx@addzeros #1\Z #2{ #2#1}%
2936 \def\XINT@dsx@xisPos #1\Z #2%
2937 {%
2938   \XINT@dsx@checksignA #2\Z {#1}%
2939 }%
2940 \def\XINT@dsx@checksignA #1%
2941 {%
2942   \xint@UDzerominusfork
2943     #1-\dummy \XINT@dsx@AisZero
2944     0#1\dummy \XINT@dsx@AisNeg
2945     0-\dummy {\XINT@dsx@AisPos #1}%
2946   \xint@UDkrof
2947 }%
2948 \def\XINT@dsx@AisZero #1\Z #2{ {0}{0}}%
2949 \def\XINT@dsx@AisNeg #1\Z #2%
2950 {%
2951   \expandafter\XINT@dsx@AisNeg@dosplit@andcheckfirst
2952     \romannumeral0\XINT@split@checksize {#2}{#1}%
2953 }%
2954 \def\XINT@dsx@AisNeg@dosplit@andcheckfirst #1%
2955 {%
2956   \XINT@dsx@AisNeg@checkiffirstempty #1\Z
2957 }%
2958 \def\XINT@dsx@AisNeg@checkiffirstempty #1%
2959 {%
2960   \xint@z #1\XINT@dsx@AisNeg@finish@zero\Z
2961   \XINT@dsx@AisNeg@finish@notzero #1%
2962 }%
2963 \def\XINT@dsx@AisNeg@finish@zero\Z
2964   \XINT@dsx@AisNeg@finish@notzero\Z #1%
2965 {%
2966   \expandafter\XINT@dsx@end

```

## 19 Package *xint* implementation

```

2967 \expandafter {\romannumeral0\XINT@num {-#1}}{0}%
2968 }%
2969 \def\XINT@dsx@AisNeg@finish@notzero #1\Z #2%
2970 {%
2971 \expandafter\XINT@dsx@end
2972 \expandafter {\romannumeral0\XINT@num {#2}}{-#1}%
2973 }%
2974 \def\XINT@dsx@AisPos #1\Z #2%
2975 {%
2976 \expandafter\XINT@dsx@AisPos@finish
2977 \romannumeral0\XINT@split@checksize {#2}{#1}%
2978 }%
2979 \def\XINT@dsx@AisPos@finish #1#2%
2980 {%
2981 \expandafter\XINT@dsx@end
2982 \expandafter {\romannumeral0\XINT@num {#2}}%
2983 {\romannumeral0\XINT@num {#1}}%
2984 }%
2985 \def\XINT@dsx@end #1#2%
2986 {%
2987 \expandafter\space\expandafter{#2}{#1}%
2988 }%

```

### 19.40 \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@checksize 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 may then be perhaps made to the code, it is kept as is for the time being.

```

2989 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
2990 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
2991 \def\xintdecsplitl
2992 {%
2993 \expandafter\xint@firstoftwo@andstop

```

## 19 Package *xint* implementation

```

2994 \romannumeral0\xintdecsplit
2995 }%
2996 \def\xintdecsplitr
2997 {%
2998 \expandafter\xint@secondoftwo@andstop
2999 \romannumeral0\xintdecsplit
3000 }%
3001 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
3002 \def\xintdecsplit #1#2%
3003 {%
3004 \expandafter \xint@split \expandafter
3005 {\romannumeral0\xintiabs {#2}}{#1}% fait expansion de A
3006 }%
3007 \def\xint@split #1#2%
3008 {%
3009 \expandafter\XINT@split@checksize\expandafter{\the\numexpr #2}{#1}%
3010 }%
3011 \def\XINT@split@checksize #1% 999999999 is anyhow very big, could be reduced
3012 {%
3013 \ifnum\numexpr\XINT@Abs{#1}\relax > 999999999
3014 \xint@afterfi {\xintError:TooBigDecimalSplit\XINT@split@bigx }%
3015 \else
3016 \expandafter\XINT@split@xfork
3017 \fi
3018 #1\Z
3019 }%
3020 \def\XINT@split@bigx #1\Z #2%
3021 {%
3022 \ifcase\XINT@Sgn {#1}
3023 \or \xint@afterfi { {}{#2}}% positive big x
3024 \else
3025 \xint@afterfi { {}{#2}}% negative big x
3026 \fi
3027 }%
3028 \def\XINT@split@xfork #1%
3029 {%
3030 \xint@UDzerominusfork
3031 #1-\dummy \XINT@split@zerosplit
3032 0#1\dummy \XINT@split@fromleft
3033 0-\dummy {\XINT@split@fromright #1}%
3034 \xint@UDkrof
3035 }%
3036 \def\XINT@split@zerosplit #1\Z #2{ {}{#2}}%
3037 \def\XINT@split@fromleft #1\Z #2%
3038 {%
3039 \XINT@split@fromleft@loop {#1}{#2\W\W\W\W\W\W\W\W\Z
3040 }%
3041 \def\XINT@split@fromleft@loop #1%
3042 {%

```



## 19 Package *xint* implementation

```

3043 \ifnum #1<8 \XINT@split@fromleft@exita\fi
3044 \expandafter\XINT@split@fromleft@loop@perhaps\expandafter
3045 {\the\numexpr #1-8\expandafter}\XINT@split@fromleft@eight
3046 }%
3047 \def\XINT@split@fromleft@eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
3048 \def\XINT@split@fromleft@loop@perhaps #1#2%
3049 {%
3050 \xint@w #2\XINT@split@fromleft@toofar\W\XINT@split@fromleft@loop {#1}%
3051 }%
3052 \def\XINT@split@fromleft@toofar\W\XINT@split@fromleft@loop #1#2#3\Z
3053 {%
3054 \XINT@split@fromleft@toofar@b #2\Z
3055 }%
3056 \def\XINT@split@fromleft@toofar@b #1\W #2\Z { {#1}{}}%
3057 \def\XINT@split@fromleft@exita\fi
3058 \expandafter\XINT@split@fromleft@loop@perhaps\expandafter #1#2%
3059 {\fi \XINT@split@fromleft@exitb #1}%
3060 \def\XINT@split@fromleft@exitb\the\numexpr #1-8\expandafter
3061 {%
3062 \csname XINT@split@fromleft@endsplit@\romannumeral #1\endcsname
3063 }%
3064 \def\XINT@split@fromleft@endsplit@ #1#2\W #3\Z { {#1}{#2}}%
3065 \def\XINT@split@fromleft@endsplit@i #1#2%
3066 {\XINT@split@fromleft@checkiftoofar #2{#1#2}}%
3067 \def\XINT@split@fromleft@endsplit@ii #1#2#3%
3068 {\XINT@split@fromleft@checkiftoofar #3{#1#2#3}}%
3069 \def\XINT@split@fromleft@endsplit@iii #1#2#3#4%
3070 {\XINT@split@fromleft@checkiftoofar #4{#1#2#3#4}}%
3071 \def\XINT@split@fromleft@endsplit@iv #1#2#3#4#5%
3072 {\XINT@split@fromleft@checkiftoofar #5{#1#2#3#4#5}}%
3073 \def\XINT@split@fromleft@endsplit@v #1#2#3#4#5#6%
3074 {\XINT@split@fromleft@checkiftoofar #6{#1#2#3#4#5#6}}%
3075 \def\XINT@split@fromleft@endsplit@vi #1#2#3#4#5#6#7%
3076 {\XINT@split@fromleft@checkiftoofar #7{#1#2#3#4#5#6#7}}%
3077 \def\XINT@split@fromleft@endsplit@vii #1#2#3#4#5#6#7#8%
3078 {\XINT@split@fromleft@checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
3079 \def\XINT@split@fromleft@checkiftoofar #1#2#3\W #4\Z
3080 {%
3081 \xint@w #1\XINT@split@fromleft@wenttoofar\W\space {#2}{#3}%
3082 }%
3083 \def\XINT@split@fromleft@wenttoofar\W\space #1%
3084 {%
3085 \XINT@split@fromleft@wenttoofar@b #1\Z
3086 }%
3087 \def\XINT@split@fromleft@wenttoofar@b #1\W #2\Z { {#1}}%
3088 \def\XINT@split@fromright #1\Z #2%
3089 {%
3090 \expandafter \XINT@split@fromright@a \expandafter
3091 {\romannumeral0\XINT@rev {#2}}{#1}{#2}%

```

## 19 Package *xint* implementation

```

3092 }%
3093 \def\XINT@split@fromright@a #1#2%
3094 {%
3095   \XINT@split@fromright@loop {#2}{#1\W\W\W\W\W\W\W\W\Z
3096 }%
3097 \def\XINT@split@fromright@loop #1%
3098 {%
3099   \ifnum #1<8 \XINT@split@fromright@exit@fi
3100   \expandafter\XINT@split@fromright@loop@perhaps\expandafter
3101   {\the\numexpr #1-8\expandafter }\XINT@split@fromright@eight
3102 }%
3103 \def\XINT@split@fromright@eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
3104 \def\XINT@split@fromright@loop@perhaps #1#2%
3105 {%
3106   \xint@w #2\XINT@split@fromright@toofar\W
3107   \XINT@split@fromright@loop {#1}%
3108 }%
3109 \def\XINT@split@fromright@toofar\W\XINT@split@fromright@loop #1#2#3\Z { {}}%
3110 \def\XINT@split@fromright@exit@fi
3111   \expandafter\XINT@split@fromright@loop@perhaps\expandafter #1#2%
3112   {\fi \XINT@split@fromright@exitb #1}%
3113 \def\XINT@split@fromright@exitb\the\numexpr #1-8\expandafter
3114 {%
3115   \csname XINT@split@fromright@endsplit@\romannumeral #1\endcsname
3116 }%
3117 \def\XINT@split@fromright@endsplit@ #1#2\W #3\Z #4%
3118 {%
3119   \expandafter\space\expandafter {\romannumeral0\XINT@rev{#2}}{#1}%
3120 }%
3121 \def\XINT@split@fromright@endsplit@i #1#2%
3122   {\XINT@split@fromright@checkiftoofar #2{#2#1}}%
3123 \def\XINT@split@fromright@endsplit@ii #1#2#3%
3124   {\XINT@split@fromright@checkiftoofar #3{#3#2#1}}%
3125 \def\XINT@split@fromright@endsplit@iii #1#2#3#4%
3126   {\XINT@split@fromright@checkiftoofar #4{#4#3#2#1}}%
3127 \def\XINT@split@fromright@endsplit@iv #1#2#3#4#5%
3128   {\XINT@split@fromright@checkiftoofar #5{#5#4#3#2#1}}%
3129 \def\XINT@split@fromright@endsplit@v #1#2#3#4#5#6%
3130   {\XINT@split@fromright@checkiftoofar #6{#6#5#4#3#2#1}}%
3131 \def\XINT@split@fromright@endsplit@vi #1#2#3#4#5#6#7%
3132   {\XINT@split@fromright@checkiftoofar #7{#7#6#5#4#3#2#1}}%
3133 \def\XINT@split@fromright@endsplit@vii #1#2#3#4#5#6#7#8%
3134   {\XINT@split@fromright@checkiftoofar #8{#8#7#6#5#4#3#2#1}}%
3135 \def\XINT@split@fromright@checkiftoofar #1%
3136 {%
3137   \xint@w #1\XINT@split@fromright@wenttoofar\W
3138   \XINT@split@fromright@endsplit@
3139 }%
3140 \def\XINT@split@fromright@wenttoofar\W\XINT@split@fromright@endsplit@ #1\Z #2%

```

```

3141 { }{#2}}%
3142 \XINT@restorecatcodes@endinput%

```

## 20 Package **xintgcd** implementation

The commenting is currently (2013/05/07) very sparse.

### Contents

1	Catcodes, $\varepsilon$ -T <sub>E</sub> X and reload detection . . . . .	144	6	\xintBezout . . . . .	148
2	Confirmation of <b>xint</b> loading . . . . .	145	7	\xintEuclideanAlgorithm . . . . .	152
3	Catcodes . . . . .	146	8	\xintBezoutAlgorithm . . . . .	154
4	Package identification . . . . .	147	9	\xintTypesetEuclideanAlgorithm . . . . .	156
5	\xintGCD . . . . .	147	10	\xintTypesetBezoutAlgorithm . . . . .	156

### 20.1 Catcodes, $\varepsilon$ -T<sub>E</sub>X and reload detection

The code for reload detection is copied from HEIKO OBERDIEK's packages, and adapted here to check for previous loading of the master **xint** package.

The method for catcodes is slightly different, but still directly inspired by these packages.

```

3143 \begingroup\catcode61\catcode48\catcode32=10\relax%
3144 \catcode13=5 % ^^M
3145 \endlinechar=13 %
3146 \catcode123=1 % {
3147 \catcode125=2 % }
3148 \catcode64=11 % @
3149 \catcode35=6 % #
3150 \catcode44=12 % ,
3151 \catcode45=12 % -
3152 \catcode46=12 % .
3153 \catcode58=12 % :
3154 \def\space { }%
3155 \let\z\endgroup
3156 \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
3157 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
3158 \expandafter
3159 \ifx\csname PackageInfo\endcsname\relax
3160 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
3161 \else
3162 \def\y#1#2{\PackageInfo{#1}{#2}}%
3163 \fi
3164 \expandafter
3165 \ifx\csname numexpr\endcsname\relax
3166 \y{xintgcd}{\numexpr not available, aborting input}%
3167 \aftergroup\endinput
3168 \else

```

```

3169 \ifx\x\relax % plain-TeX, first loading of xintgcd.sty
3170 \ifx\w\relax % but xint.sty not yet loaded.
3171 \y{xintgcd}{Package xint is required}%
3172 \y{xintgcd}{Will try \string\input\space xint.sty}%
3173 \def\z{\endgroup\input xint.sty\relax}%
3174 \fi
3175 \else
3176 \def\empty {}%
3177 \ifx\x\empty % LaTeX, first loading,
3178 % variable is initialized, but \ProvidesPackage not yet seen
3179 \ifx\w\relax % xint.sty not yet loaded.
3180 \y{xintgcd}{Package xint is required}%
3181 \y{xintgcd}{Will try \string\RequirePackage{xint}}%
3182 \def\z{\endgroup\RequirePackage{xint}}%
3183 \fi
3184 \else
3185 \y{xintgcd}{I was already loaded, aborting input}%
3186 \aftergroup\endinput
3187 \fi
3188 \fi
3189 \fi
3190 \z%

```

## 20.2 Confirmation of **xint** loading

```

3191 \begingroup\catcode61\catcode48\catcode32=10\relax%
3192 \catcode13=5 % ^^M
3193 \endlinechar=13 %
3194 \catcode123=1 % {
3195 \catcode125=2 % }
3196 \catcode64=11 % @
3197 \catcode35=6 % #
3198 \catcode44=12 % ,
3199 \catcode45=12 % -
3200 \catcode46=12 % .
3201 \catcode58=12 % :
3202 \expandafter
3203 \ifx\csname PackageInfo\endcsname\relax
3204 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
3205 \else
3206 \def\y#1#2{\PackageInfo{#1}{#2}}%
3207 \fi
3208 \def\empty {}%
3209 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
3210 \ifx\w\relax % Plain TeX, user gave a file name at the prompt
3211 \y{xintgcd}{Loading of package xint failed, aborting input}%
3212 \aftergroup\endinput
3213 \fi
3214 \ifx\w\empty % LaTeX, user gave a file name at the prompt

```

```

3215      \y{xintgcd}{Loading of package xint failed, aborting input}%
3216      \aftergroup\endinput
3217  \fi
3218 \endgroup%

```

## 20.3 Catcodes

Perhaps catcodes have changed after the loading of **xint** and prior to the current loading of **xintgcd**, so we can not employ the `\XINT@restorecatcodes@endinput` in this style file. But there is no problem using `\XINT@setcatcodes`.

```

3219 \begingroup\catcode61\catcode48\catcode32=10\relax%
3220  \catcode13=5      % ^^M
3221  \endlinechar=13 %
3222  \catcode123=1     % {
3223  \catcode125=2     % }
3224  \catcode64=11    % @
3225  \def\x
3226  {%
3227      \endgroup
3228      \edef\XINT@gcd@restorecatcodes@endinput
3229      {%
3230          \catcode36=\the\catcode36      % $
3231          \catcode96=\the\catcode96      % '
3232          \catcode47=\the\catcode47      % /
3233          \catcode41=\the\catcode41      % )
3234          \catcode40=\the\catcode40      % (
3235          \catcode42=\the\catcode42      % *
3236          \catcode43=\the\catcode43      % +
3237          \catcode62=\the\catcode62      % >
3238          \catcode60=\the\catcode60      % <
3239          \catcode58=\the\catcode58      % :
3240          \catcode46=\the\catcode46      % .
3241          \catcode45=\the\catcode45      % -
3242          \catcode44=\the\catcode44      % ,
3243          \catcode35=\the\catcode35      % #
3244          \catcode64=\the\catcode64      % @
3245          \catcode125=\the\catcode125    % }
3246          \catcode123=\the\catcode123    % {
3247          \endlinechar=\the\endlinechar
3248          \catcode13=\the\catcode13      % ^^M
3249          \catcode32=\the\catcode32      %
3250          \catcode61=\the\catcode61\relax % =
3251          \noexpand\endinput
3252      }%
3253      \XINT@setcatcodes
3254      \catcode36=3      % $
3255  }%
3256 \x

```

**20.4 Package identification**

```

3257 \begingroup
3258   \catcode91=12 % [
3259   \catcode93=12 % ]
3260   \catcode58=12 % :
3261   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
3262     \def\x#1#2#3[#4]{\endgroup
3263       \immediate\write-1{Package: #3 #4}%
3264       \xdef#1{#4}%
3265     }%
3266   \else
3267     \def\x#1#2[#3]{\endgroup
3268       #2[#{#3}]%
3269       \ifx#1\@undefined
3270         \xdef#1{#3}%
3271       \fi
3272       \ifx#1\relax
3273         \xdef#1{#3}%
3274       \fi
3275     }%
3276   \fi
3277 \expandafter\x\csname ver@xintgcd.sty\endcsname
3278 \ProvidesPackage{xintgcd}%
3279 [2013/05/07 v1.06 Euclidean algorithm with xint package (jfb)]%

```

**20.5 \xintGCD**

```

3280 \def\xintGCD {\romannumeral0\xintgcd }%
3281 \def\xintgcd #1%
3282 {%
3283   \expandafter\XINT@gcd\expandafter{\romannumeral0\xintiabs {#1}}%
3284 }%
3285 \def\XINT@gcd #1#2%
3286 {%
3287   \expandafter\XINT@gcd@fork\romannumeral0\xintiabs {#2}\Z #1\Z
3288 }%
3289   Ici #3#4=A, #1#2=B
3289 \def\XINT@gcd@fork #1#2\Z #3#4\Z
3290 {%
3291   \xint@UDzerofork
3292   #1\dummy \XINT@gcd@BisZero
3293   #3\dummy \XINT@gcd@AisZero
3294   0\dummy \XINT@gcd@loop
3295   \xint@UDkrof
3296   {#1#2}{#3#4}%
3297 }%
3298 \def\XINT@gcd@AisZero #1#2{ #1}%
3299 \def\XINT@gcd@BisZero #1#2{ #2}%
3300 \def\XINT@gcd@CheckRem #1#2\Z

```

```

3301 {%
3302   \xint@zero #1\xint@gcd@end0\XINT@gcd@loop {#1#2}%
3303 }%
3304 \def\xint@gcd@end0\XINT@gcd@loop #1#2{ #2}%

    #1=B, #2=A

3305 \def\XINT@gcd@loop #1#2%
3306 {%
3307   \expandafter\expandafter\expandafter
3308     \XINT@gcd@CheckRem
3309   \expandafter\xint@secondoftwo
3310   \romannumeral0\XINT@div@prepare {#1}{#2}\Z
3311   {#1}%
3312 }%

```

## 20.6 \xintBezout

```

3313 \def\xintBezout {\romannumeral0\xintbezout }%
3314 \def\xintbezout #1%
3315 {%
3316   \expandafter\xint@bezout\expandafter {\romannumeral-‘0#1}%
3317 }%
3318 \def\xint@bezout #1#2%
3319 {%
3320   \expandafter\XINT@bezout@fork \romannumeral-‘0#2\Z #1\Z
3321 }%
    #3#4 = A, #1#2=B

3322 \def\XINT@bezout@fork #1#2\Z #3#4\Z
3323 {%
3324   \xint@UDzerosfork
3325   #1#3\dummy \XINT@bezout@botharezero
3326   #10\dummy \XINT@bezout@secondiszero
3327   #30\dummy \XINT@bezout@firstiszero
3328   00\dummy
3329   {\xint@UDsignsfork
3330     #1#3\dummy \XINT@bezout@minusminus % A < 0, B < 0
3331     #1-\dummy \XINT@bezout@minusplus  % A > 0, B < 0
3332     #3-\dummy \XINT@bezout@plusminus  % A < 0, B > 0
3333     --\dummy \XINT@bezout@plusplus   % A > 0, B > 0
3334   \xint@UDkrof }%
3335   \xint@UDkrof
3336   {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
3337 }%
3338 \def\XINT@bezout@botharezero #1#2#3#4#5#6%
3339 {%
3340   \xintError:NoBezoutForZeros
3341   \space {0}{0}{0}{0}{0}%
3342 }%

```

```

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

3343 \def\XINT@bezout@firstiszero #1#2#3#4#5#6%
3344 {%
3345   \xint@UDsignfork
3346   #3\dummy { {0}{#3#1}{0}{1}{#1}}%
3347   -\dummy { {0}{#3#1}{0}{-1}{#1}}%
3348   \xint@UDkrof
3349 }%

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

3350 \def\XINT@bezout@secondiszero #1#2#3#4#5#6%
3351 {%
3352   \xint@UDsignfork
3353   #4\dummy { {#4#2}{0}{-1}{0}{#2}}%
3354   -\dummy { {#4#2}{0}{1}{0}{#2}}%
3355   \xint@UDkrof
3356 }%

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

3357 \def\XINT@bezout@minusminus #1#2#3#4%
3358 {%
3359   \expandafter\XINT@bezout@mm@post
3360   \romannumeral0\XINT@bezout@loop@a 1{#1}{#2}1001%
3361 }%
3362 \def\XINT@bezout@mm@post #1#2%
3363 {%
3364   \expandafter\XINT@bezout@mm@postb\expandafter
3365   {\romannumeral0\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
3366 }%
3367 \def\XINT@bezout@mm@postb #1#2%
3368 {%
3369   \expandafter\XINT@bezout@mm@postc\expandafter {#2}{#1}%
3370 }%
3371 \def\XINT@bezout@mm@postc #1#2#3#4#5%
3372 {%
3373   \space {#4}{#5}{#1}{#2}{#3}%
3374 }%

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

3375 \def\XINT@bezout@minusplus #1#2#3#4%
3376 {%
3377   \expandafter\XINT@bezout@mp@post
3378   \romannumeral0\XINT@bezout@loop@a 1{#1}{#4#2}1001%
3379 }%
```



```

3380 \def\XINT@bezout@mp@post #1#2%
3381 {%
3382   \expandafter\XINT@bezout@mp@postb\expandafter
3383   {\romannumeral0\xintiopp {#2}}{#1}%
3384 }%
3385 \def\XINT@bezout@mp@postb #1#2#3#4#5%
3386 {%
3387   \space {#4}{#5}{#2}{#1}{#3}%
3388 }%

  plusminus  A < 0, B > 0

3389 \def\XINT@bezout@plusminus #1#2#3#4%
3390 {%
3391   \expandafter\XINT@bezout@pm@post
3392   \romannumeral0\XINT@bezout@loop@a 1{#3#1}{#2}1001%
3393 }%
3394 \def\XINT@bezout@pm@post #1%
3395 {%
3396   \expandafter \XINT@bezout@pm@postb \expandafter
3397   {\romannumeral0\xintiopp{#1}}%
3398 }%
3399 \def\XINT@bezout@pm@postb #1#2#3#4#5%
3400 {%
3401   \space {#4}{#5}{#1}{#2}{#3}%
3402 }%

  plusplus

3403 \def\XINT@bezout@plusplus #1#2#3#4%
3404 {%
3405   \expandafter\XINT@bezout@pp@post
3406   \romannumeral0\XINT@bezout@loop@a 1{#3#1}{#4#2}1001%
3407 }%

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

3408 \def\XINT@bezout@pp@post #1#2#3#4#5%
3409 {%
3410   \space {#4}{#5}{#1}{#2}{#3}%
3411 }%

  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

3412 \def\XINT@bezout@loop@a #1#2#3%
3413 {%
3414   \expandafter\XINT@bezout@loop@b
3415   \expandafter{\the\numexpr -#1\expandafter}%
3416   \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
3417 }%

```

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

```

3418 \def\XINT@bezout@loop@b #1#2#3#4#5#6#7#8%
3419 {%
3420     \expandafter \XINT@bezout@loop@c \expandafter
3421         {\romannumeral0\xintiadd{\XINT@Mul{#5}{#2}}{#7}}%
3422         {\romannumeral0\xintiadd{\XINT@Mul{#6}{#2}}{#8}}%
3423     {#1}{#3}{#4}{#5}{#6}%
3424 }%

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

3425 \def\XINT@bezout@loop@c #1#2%
3426 {%
3427     \expandafter \XINT@bezout@loop@d \expandafter
3428         {#2}{#1}%
3429 }%

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

3430 \def\XINT@bezout@loop@d #1#2#3#4#5%
3431 {%
3432     \XINT@bezout@loop@e #4\Z {#3}{#5}{#2}{#1}%
3433 }%

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

3434 \def\XINT@bezout@loop@e #1#2\Z
3435 {%
3436     \xint@zero #1\xint@bezout@loop@exit0\XINT@bezout@loop@f
3437     {#1#2}%
3438 }%

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

3439 \def\XINT@bezout@loop@f #1#2%
3440 {%
3441     \XINT@bezout@loop@a {#2}{#1}%
3442 }%

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

```

```

3443 \def\xint@bezout@loop@exit0\XINT@bezout@loop@f #1#2%
3444 {%
3445   \ifcase #2
3446   \or \expandafter\XINT@bezout@exiteven
3447   \else\expandafter\XINT@bezout@exitodd
3448   \fi
3449 }%
3450 \def\XINT@bezout@exiteven #1#2#3#4#5%
3451 {%
3452   \space {#5}{#4}{#1}%
3453 }%
3454 \def\XINT@bezout@exitodd #1#2#3#4#5%
3455 {%
3456   \space {-#5}{-#4}{#1}%
3457 }%

```

## 20.7 \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

```

3458 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm }%
3459 \def\xinteucclideanalgorithm #1%
3460 {%
3461   \expandafter \XINT@euc \expandafter{\romannumeral0\xintiabs {#1}}%
3462 }%
3463 \def\XINT@euc #1#2%
3464 {%
3465   \expandafter\XINT@euc@fork \romannumeral0\xintiabs {#2}\Z #1\Z
3466 }%

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

3467 \def\XINT@euc@fork #1#2\Z #3#4\Z
3468 {%
3469   \xint@UDzerofork
3470   #1\dummy \XINT@euc@BisZero
3471   #3\dummy \XINT@euc@AisZero
3472   0\dummy \XINT@euc@a
3473   \xint@UDkrof
3474   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
3475 }%

```

Le {} pour protéger  $\{A\}\{B\}$  si on s'arrête après une étape (B divise A)

On va renvoyer:

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

```

3476 \def\XINT@euc@AisZero #1#2#3#4#5#6{ {1}{0}{#2}{#2}{0}{0}}%
3477 \def\XINT@euc@BisZero #1#2#3#4#5#6{ {1}{0}{#3}{#3}{0}{0}}%

```

```

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

3478 \def\XINT@euc@a #1#2#3%
3479 {%
3480   \expandafter\XINT@euc@b
3481   \expandafter {\the\numexpr #1+1\expandafter}%
3482   \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
3483}%

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

3484 \def\XINT@euc@b #1#2#3#4%
3485 {%
3486   \XINT@euc@c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
3487}%

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.

3488 \def\XINT@euc@c #1#2\Z
3489 {%
3490   \xint@zero #1\xint@euc@end0\XINT@euc@a
3491}%

{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}

3492 \def\xint@euc@end0\XINT@euc@a #1#2#3#4\Z%
3493 {%
3494   \expandafter\xint@euc@end@
3495   \romannumeral0%
3496   \XINT@rord@main {}#4{{#1}{#3}}%
3497   \xint@UNDEF
3498   \xint@undef\xint@undef\xint@undef\xint@undef
3499   \xint@undef\xint@undef\xint@undef\xint@undef
3500   \xint@UNDEF
3501}%
3502 \def\xint@euc@end@ #1#2#3%
3503 {%
3504   \space {#1}{#3}{#2}%
3505}%

```

**20.8 \xintBezoutAlgorithm**

```

Pour Bezout: objectif, renvoyer
alpha0=1, beta0=0
alpha(-1)=0, beta(-1)=1
{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}

3506 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm}%
3507 \def\xintbezoutalgorithm #1%
3508 {%
3509   \expandafter \XINT@bevalg \expandafter{\romannumeral0\xintiabs {#1}}%
3510}%
3511 \def\XINT@bevalg #1#2%
3512 {%
3513   \expandafter\XINT@bevalg@fork \romannumeral0\xintiabs {#2}\Z #1\Z
3514}%

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

3515 \def\XINT@bevalg@fork #1#2\Z #3#4\Z
3516 {%
3517   \xint@UDzerofork
3518   #1\dummy \XINT@bevalg@BisZero
3519   #3\dummy \XINT@bevalg@AisZero
3520   0\dummy \XINT@bevalg@a
3521   \xint@UDkrof
3522   0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{\Z}
3523}%
3524 \def\XINT@bevalg@AisZero #1#2#3\Z{ {1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
3525 \def\XINT@bevalg@BisZero #1#2#3#4\Z{ {1}{0}{0}{1}{#3}{#3}{1}{0}{0}{0}{0}{0}{1}}%

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

3526 \def\XINT@bevalg@a #1#2#3%
3527 {%
3528   \expandafter\XINT@bevalg@b
3529   \expandafter {\the\numexpr #1+1\expandafter}%
3530   \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
3531}%

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

3532 \def\XINT@bevalg@b #1#2#3#4#5#6#7#8%
3533 {%
3534   \expandafter\XINT@bevalg@c\expandafter
3535   {\romannumeral0\xintiadd {\xintiMul {#6}{#2}}{#8}}%
3536   {\romannumeral0\xintiadd {\xintiMul {#5}{#2}}{#7}}%
3537   {#1}{#2}{#3}{#4}{#5}{#6}%
3538}%

```

```

{beta(n+1)}{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n)}%
3539 \def\XINT@bezialg@c #1#2#3#4#5#6%
3540 {%
3541   \expandafter\XINT@bezialg@d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
3542 }%

{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{beta(n+1)}
3543 \def\XINT@bezialg@d #1#2#3#4#5#6#7#8%
3544 {%
3545   \XINT@bezialg@e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{#3}{#4}{#1}{#6}}%
3546 }%

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.

3547 \def\XINT@bezialg@e #1#2\Z
3548 {%
3549   \xint@zero #1\xint@bezialg@end0\XINT@bezialg@a
3550 }%

Ici r(n+1) = 0. On arrête on se prépare à inverser.
{n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}%
{alpha(n)}{beta(n)}%
{q,r,alpha,beta(n+1)}...{A}{B}}\Z

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

3551 \def\xint@bezialg@end0\XINT@bezialg@a #1#2#3#4#5#6#7#8\Z
3552 {%
3553   \expandafter\xint@bezialg@end@
3554   \romannumeral0%
3555   \XINT@rord@main {}#8{{#1}{#3}}%
3556   \xint@UNDEF
3557   \xint@undef\xint@undef\xint@undef\xint@undef
3558   \xint@undef\xint@undef\xint@undef\xint@undef
3559   \xint@UNDEF
3560 }%

{N}{D}{A}{B}{q1}{r1}{alpha1=q1}{beta1=1}{q2}{r2}{alpha2}{beta2}
...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

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

3561 \def\xint@bezialg@end@ #1#2#3#4%
3562 {%
3563   \space {#1}{#3}{0}{1}{#2}{#4}{1}{0}%
3564 }%

```

**20.9 \xintTypesetEuclideanAlgorithm**

```

TYPESETTING
Organisation:
{N}{A}{D}{B}{q1}{r1}{q2}{r2}{q3}{r3}...{qN}{rN=0}
\U1 = N = nombre d'étapes, \U3 = PGCD, \U2 = A, \U4=B
q1 = \U5, q2 = \U7 --> qn = \U<2n+3>, rn = \U<2n+4>
bn = rn. B = r0. A=r(-1)
r(n-2) = q(n)r(n-1)+r(n) (n e étape) (n au moins 1)
\U{2n} = \U{2n+3} \times \U{2n+2} + \U{2n+4}, n e étape.
avec n entre 1 et N.

3565 \def\xintTypesetEuclideanAlgorithm #1#2%
3566 {% l'algo remplace #1 et #2 par |#1| et |#2|
3567   \par
3568   \begingroup
3569     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
3570     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
3571     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
3572     \noindent
3573     \count 255 1
3574     \loop
3575       \hbox to \wd 0 {\hfil$\U{\numexpr 2*\count 255\relax}$}%
3576       ${} = \U{\numexpr 2*\count 255 + 3\relax}
3577       \times \U{\numexpr 2*\count 255 + 2\relax}
3578       + \U{\numexpr 2*\count 255 + 4\relax}$%
3579       \ifnum \count 255 < \N
3580         \hfill\break
3581         \advance \count 255 1
3582       \repeat
3583     \par
3584   \endgroup
3585 }%

```

**20.10 \xintTypesetBezoutAlgorithm**

```

Pour Bezout on a:
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
      {q2}{r2}{alpha2}{beta2}...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}%
Donc 4N+8 termes
U1 = N, U2= A, U5=D, U6=B,
q1 = U9, qn = U{4n+5}, n au moins 1
rn = U{4n+6}      , n au moins -1
alpha(n) = U{4n+7}, n au moins -1
beta(n)  = U{4n+8}, n au moins -1

3586 \def\xintTypesetBezoutAlgorithm #1#2%
3587 {%
3588   \par
3589   \begingroup

```

```

3590 \parindent0pt
3591 \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
3592 \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
3593 \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
3594 \count 255 1
3595 \loop
3596 \noindent
3597 \hbox to \wd 0 {\hfil$\BEZ{4*}\count 255 - 2}$}%
3598 ${} = \BEZ{4*}\count 255 + 5}
3599 \times \BEZ{4*}\count 255 + 2}
3600 + \BEZ{4*}\count 255 + 6}$\hfill\break
3601 \hbox to \wd 0 {\hfil$\BEZ{4*}\count 255 +7}$}%
3602 ${} = \BEZ{4*}\count 255 + 5}
3603 \times \BEZ{4*}\count 255 + 3}
3604 + \BEZ{4*}\count 255 - 1}$\hfill\break
3605 \hbox to \wd 0 {\hfil$\BEZ{4*}\count 255 +8}$}%
3606 ${} = \BEZ{4*}\count 255 + 5}
3607 \times \BEZ{4*}\count 255 + 4}
3608 + \BEZ{4*}\count 255 }$
3609 \endgraf
3610 \ifnum \count 255 < \N
3611 \advance \count 255 1
3612 \repeat
3613 \par
3614 \edef\U{\BEZ{4*}\N + 4}}%
3615 \edef\V{\BEZ{4*}\N + 3}}%
3616 \edef\D{\BEZ5}%
3617 \ifodd\N
3618 $\U\times\A - \V\times \B = -\D$%
3619 \else
3620 $\U\times\A - \V\times\B = \D$%
3621 \fi
3622 \par
3623 \endgroup
3624 }%
3625 \XINT@gcd@restorecatcodes@endinput%

```

## 21 Package **xintfrac** implementation

The commenting is currently (2013/05/07) very sparse.

### Contents

1	Catcodes, $\varepsilon$ -TeX and reload detection . . . . .	158	5	\xintLen . . . . .	161
2	Confirmation of <b>xint</b> loading . . . . .	159	6	\XINT@outfrac . . . . .	161
3	Catcodes . . . . .	160	7	\XINT@inFrac . . . . .	162
4	Package identification . . . . .	161	8	\XINT@frac . . . . .	163



9	\XINT@factortens, \XINT@cuz@cnt	164	25	\xintSum, \xintSumExpr	178
10	\xintRaw	166	26	\xintMul	179
11	\xintNumerator	167	27	\xintSqr	179
12	\xintDenominator	167	28	\xintPow	179
13	\xintFrac	167	29	\xintPrd, \xintProductExpr	180
14	\xintSignedFrac	168	30	\xintDiv	181
15	\xintFwOver	168	31	\xintCmp	181
16	\xintSignedFwOver	169	32	\xintMax	182
17	\xintREZ	170	33	\xintMin	182
18	\xintIrr	170	34	\xintAbs	183
19	\xintNum	172	35	\xintOpp	183
20	\xintJrr	172	36	\xintSgn	183
21	\xintTrunc, \xintiTrunc	174	37	\xintGeq	184
22	\xintRound, \xintiRound	176	38	\xintDivision, \xintQuo, \xintRem	184
23	\xintAdd	177	39	\xintFDg, \xintLDg, \xintMON, \xint-	
24	\xintSub	178	MMON, \xintOdd	184	

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

The code for reload detection is copied from HEIKO OBERDIEK's packages, and adapted here to check for previous loading of the master **xint** package.

The method for catcodes is slightly different, but still directly inspired by these packages.

```

3626 \begingroup\catcode61\catcode48\catcode32=10\relax%
3627 \catcode13=5 % ^^M
3628 \endlinechar=13 %
3629 \catcode123=1 % {
3630 \catcode125=2 % }
3631 \catcode64=11 % @
3632 \catcode35=6 % #
3633 \catcode44=12 % ,
3634 \catcode45=12 % -
3635 \catcode46=12 % .
3636 \catcode58=12 % :
3637 \def\space { }%
3638 \let\z\endgroup
3639 \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
3640 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
3641 \expandafter
3642 \ifx\csname PackageInfo\endcsname\relax
3643 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
3644 \else
3645 \def\y#1#2{\PackageInfo{#1}{#2}}%
3646 \fi
3647 \expandafter
3648 \ifx\csname numexpr\endcsname\relax
3649 \y{xintfrac}{numexpr not available, aborting input}%

```

```

3650 \aftergroup\endinput
3651 \else
3652 \ifx\x\relax % plain-TeX, first loading of xintfrac.sty
3653 \ifx\w\relax % but xint.sty not yet loaded.
3654 \y{xintfrac}{Package xint is required}%
3655 \y{xintfrac}{Will try \string\input\space xint.sty}%
3656 \def\z{\endgroup\input xint.sty\relax}%
3657 \fi
3658 \else
3659 \def\empty {}%
3660 \ifx\x\empty % LaTeX, first loading,
3661 % variable is initialized, but \ProvidesPackage not yet seen
3662 \ifx\w\relax % xint.sty not yet loaded.
3663 \y{xintfrac}{Package xint is required}%
3664 \y{xintfrac}{Will try \string\RequirePackage{xint}}%
3665 \def\z{\endgroup\RequirePackage{xint}}%
3666 \fi
3667 \else
3668 \y{xintfrac}{I was already loaded, aborting input}%
3669 \aftergroup\endinput
3670 \fi
3671 \fi
3672 \fi
3673 \z%

```

## 21.2 Confirmation of **xint** loading

```

3674 \begingroup\catcode61\catcode48\catcode32=10\relax%
3675 \catcode13=5 % ^^M
3676 \endlinechar=13 %
3677 \catcode123=1 % {
3678 \catcode125=2 % }
3679 \catcode64=11 % @
3680 \catcode35=6 % #
3681 \catcode44=12 % ,
3682 \catcode45=12 % -
3683 \catcode46=12 % .
3684 \catcode58=12 % :
3685 \expandafter
3686 \ifx\csname PackageInfo\endcsname\relax
3687 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
3688 \else
3689 \def\y#1#2{\PackageInfo{#1}{#2}}%
3690 \fi
3691 \def\empty {}%
3692 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
3693 \ifx\w\relax % Plain TeX, user gave a file name at the prompt
3694 \y{xintfrac}{Loading of package xint failed, aborting input}%
3695 \aftergroup\endinput

```

```

3696 \fi
3697 \ifx\w\empty % LaTeX, user gave a file name at the prompt
3698     \y{xintfrac}{Loading of package xint failed, aborting input}%
3699     \aftergroup\endinput
3700 \fi
3701 \endgroup%

```

### 21.3 Catcodes

Perhaps catcodes have changed after the loading of **xint** and prior to the current loading of **xintfrac**, so we can not employ the `\XINT@restorecatcodes@endinput` in this style file. But there is no problem using `\XINT@setcatcodes`.

```

3702 \begingroup\catcode61\catcode48\catcode32=10\relax%
3703 \catcode13=5 % ^^M
3704 \endlinechar=13 %
3705 \catcode123=1 % {
3706 \catcode125=2 % }
3707 \catcode64=11 % @
3708 \def\x
3709 {%
3710     \endgroup
3711     \edef\XINT@frac@restorecatcodes@endinput
3712     {%
3713         \catcode94=\the\catcode94 % ^
3714         \catcode93=\the\catcode93 % ]
3715         \catcode91=\the\catcode91 % [
3716         \catcode96=\the\catcode96 % '
3717         \catcode47=\the\catcode47 % /
3718         \catcode41=\the\catcode41 % )
3719         \catcode40=\the\catcode40 % (
3720         \catcode42=\the\catcode42 % *
3721         \catcode43=\the\catcode43 % +
3722         \catcode62=\the\catcode62 % >
3723         \catcode60=\the\catcode60 % <
3724         \catcode58=\the\catcode58 % :
3725         \catcode46=\the\catcode46 % .
3726         \catcode45=\the\catcode45 % -
3727         \catcode44=\the\catcode44 % ,
3728         \catcode35=\the\catcode35 % #
3729         \catcode64=\the\catcode64 % @
3730         \catcode125=\the\catcode125 % }
3731         \catcode123=\the\catcode123 % {
3732         \endlinechar=\the\endlinechar
3733         \catcode13=\the\catcode13 % ^^M
3734         \catcode32=\the\catcode32 %
3735         \catcode61=\the\catcode61\relax % =
3736         \noexpand\endinput
3737     }%
3738     \XINT@setcatcodes

```

```

3739      \catcode91=12 % [
3740      \catcode93=12 % ]
3741      \catcode94=7 % ^
3742  }%
3743 \x

```

## 21.4 Package identification

```

3744 \begingroup
3745   \catcode58=12 % :
3746   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
3747     \def\x#1#2#3[#4]{\endgroup
3748       \immediate\write-1{Package: #3 #4}%
3749       \xdef#1{#4}%
3750     }%
3751   \else
3752     \def\x#1#2[#3]{\endgroup
3753       #2[#{#3}]%
3754       \ifx#1\@undefined
3755         \xdef#1{#3}%
3756       \fi
3757       \ifx#1\relax
3758         \xdef#1{#3}%
3759       \fi
3760     }%
3761   \fi
3762 \expandafter\x\csname ver@xintfrac.sty\endcsname
3763 \ProvidesPackage{xintfrac}%
3764 [2013/05/07 v1.06 Expandable operations on fractions (jfb)]%

```

## 21.5 \xintLen

```

3765 \def\xintLen {\romannumeral0\xintlen }%
3766 \def\xintlen #1%
3767 {%
3768   \expandafter\XINT@flen\romannumeral0\XINT@infrac {#1}%
3769 }%
3770 \def\XINT@flen #1#2#3%
3771 {%
3772   \expandafter\space
3773   \the\numexpr -1+\XINT@Abs {#1}+\XINT@Len {#2}+\XINT@Len {#3}\relax
3774 }%

```

## 21.6 \XINT@outfrac

```

3775 \def\XINT@outfrac #1#2#3%
3776 {%
3777   \ifcase\XINT@Sgn{#3}
3778     \expandafter \XINT@outfrac@divisionbyzero
3779   \or

```

```

3780      \expandafter \XINT@outfrac@P
3781    \else
3782      \expandafter \XINT@outfrac@N
3783    \fi
3784    {#2}{#3}[#1]%
3785 }%
3786 \def\XINT@outfrac@divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
3787 \def\XINT@outfrac@P #1#2%
3788 {%
3789   \ifcase\XINT@Sgn{#1}
3790     \expandafter\XINT@outfrac@Zero
3791   \fi
3792   \space #1/#2%
3793 }%
3794 \def\XINT@outfrac@Zero #1[#2]{ 0[0]}%
3795 \def\XINT@outfrac@N #1#2%
3796 {%
3797   \expandafter\XINT@outfrac@N@a\expandafter
3798   {\romannumeral0\XINT@opp #2}{\romannumeral0\XINT@opp #1}%
3799 }%
3800 \def\XINT@outfrac@N@a #1#2%
3801 {%
3802   \expandafter\XINT@outfrac@P\expandafter {#2}{#1}%
3803 }%

```

## 21.7 \XINT@inFrac

```

3804 \def\XINT@inFrac {\romannumeral0\XINT@infrac }%
3805 \def\XINT@infrac #1%
3806 {%
3807   \expandafter\XINT@infrac@ \romannumeral-‘0#1[\W]\Z\T
3808 }%
3809 \def\XINT@infrac@ #1[#2#3]#4\Z
3810 {%
3811   \xint@UDwfork
3812     #2\dummy \XINT@infrac@A
3813     \W\dummy \XINT@infrac@B
3814   \xint@UDkrof
3815   #1[#2#3]#4%
3816 }%
3817 \def\XINT@infrac@A #1[\W]\T
3818 {%
3819   \XINT@frac #1/\W\Z
3820 }%
3821 \def\XINT@infrac@B #1%
3822 {%
3823   \xint@zero #1\XINT@infrac@Zero0\XINT@infrac@BB #1%
3824 }%
3825 \def\XINT@infrac@BB #1[\W]\T {\XINT@infrac@BC #1/\W\Z }%
3826 \def\XINT@infrac@BC #1/#2#3\Z

```

```

3827 {%
3828   \xint@UDwfork
3829   #2\dummy \XINT@infrac@BCa
3830   \W\dummy {\expandafter\XINT@infrac@BCb \romannumeral-‘0#2}%
3831   \xint@UDkrof
3832   #3\Z #1\Z
3833 }%
3834 \def\XINT@infrac@BCa \Z #1[#2]#3\Z { {#2}{#1}{1}}%
3835 \def\XINT@infrac@BCb #1[#2]/\W\Z #3\Z { {#2}{#3}{#1}}%
3836 \def\XINT@infrac@Zero #1\T { {0}{0}{1}}%

```

## 21.8 \XINT@frac

```

3837 \def\XINT@frac #1/#2#3\Z
3838 {%
3839   \xint@UDwfork
3840   #2\dummy \XINT@frac@A
3841   \W\dummy {\expandafter\XINT@frac@B \romannumeral-‘0#2}%
3842   \xint@UDkrof
3843   #3.\W\Z #1.\W\Z
3844 }%
3845 \def\XINT@frac@B #1.#2#3\Z
3846 {%
3847   \xint@UDwfork
3848   #2\dummy \XINT@frac@Ba
3849   \W\dummy {\XINT@frac@Bb #2}%
3850   \xint@UDkrof
3851   #3\Z #1\Z
3852 }%
3853 \def\XINT@frac@Bb #1/\W.\W\Z #2\Z
3854 {%
3855   \expandafter \XINT@frac@C \expandafter
3856   {\romannumeral0\XINT@length {#1}}{#2#1}%
3857 }%
3858 \def\XINT@frac@Ba \Z #1/\W\Z {\XINT@frac@C {0}{#1}}%
3859 \def\XINT@frac@A .\W\Z {\XINT@frac@C {0}{1}}%
3860 \def\XINT@frac@C #1#2#3.#4#5\Z
3861 {%
3862   \xint@UDwfork
3863   #4\dummy \XINT@frac@Ca
3864   \W\dummy {\XINT@frac@Cb #4}%
3865   \xint@UDkrof
3866   #5\Z #3\Z {#1}{#2}%
3867 }%
3868 \def\XINT@frac@Ca \Z #1\Z {\XINT@frac@D {0}{#1}}%
3869 \def\XINT@frac@Cb #1.\W\Z #2\Z
3870 {%
3871   \expandafter\XINT@frac@D\expandafter
3872   {\romannumeral0\XINT@length {#1}}{#2#1}%
3873 }%

```

## 21 Package *xintfrac* implementation

```

3874 \def\XINT@frac@D #1#2#3#4%
3875 {%
3876   \expandafter \XINT@frac@E \expandafter
3877   {\the\numexpr -#1+#3\expandafter}\expandafter
3878   {\romannumeral0\XINT@num@loop #2\R\R\R\R\R\R\R\R\Z }%
3879   {\romannumeral0\XINT@num@loop #4\R\R\R\R\R\R\R\R\Z }%
3880 }%
3881 \def\XINT@frac@E #1#2#3%
3882 {%
3883   \expandafter \XINT@frac@F #3\Z {#2}{#1}%
3884 }%
3885 \def\XINT@frac@F #1%
3886 {%
3887   \xint@UDzerominusfork
3888   #1-\dummy \XINT@frac@Gdivisionbyzero
3889   0#1\dummy \XINT@frac@Gneg
3890   0-\dummy {\XINT@frac@Gpos #1}%
3891   \xint@UDkrof
3892 }%
3893 \def\XINT@frac@Gdivisionbyzero #1\Z #2#3%
3894 {%
3895   \xintError:DivisionByZero\space {0}{#2}{0}%
3896 }%
3897 \def\XINT@frac@Gneg #1\Z #2#3%
3898 {%
3899   \expandafter\XINT@frac@H \expandafter{\romannumeral0\XINT@opp #2}{#3}{#1}%
3900 }%
3901 \def\XINT@frac@H #1#2{ {#2}{#1}}%
3902 \def\XINT@frac@Gpos #1\Z #2#3{ {#3}{#2}{#1}}%

```

### 21.9 \XINT@factortens, \XINT@cuz@cnt

```

3903 \def\XINT@factortens #1%
3904 {%
3905   \expandafter\XINT@cuz@cnt@loop\expandafter
3906   {\expandafter}\romannumeral0\XINT@rord@main {}#1%
3907   \xint@UNDEF
3908   \xint@undef\xint@undef\xint@undef\xint@undef
3909   \xint@undef\xint@undef\xint@undef\xint@undef
3910   \xint@UNDEF
3911   \R\R\R\R\R\R\R\R\Z
3912 }%
3913 \def\XINT@cuz@cnt #1%
3914 {%
3915   \XINT@cuz@cnt@loop {}#1\R\R\R\R\R\R\R\R\Z
3916 }%
3917 \def\XINT@cuz@cnt@loop #1#2#3#4#5#6#7#8#9%
3918 {%
3919   \xint@r #9\XINT@cuz@cnt@toofara \R
3920   \expandafter\XINT@cuz@cnt@checka\expandafter

```

## 21 Package *xintfrac* implementation

```

3921   {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
3922 }%
3923 \def\XINT@cuз@cnt@toofara\R
3924   \expandafter\XINT@cuз@cnt@checka\expandafter #1#2%
3925 {%
3926   \XINT@cuз@cnt@toofarb {#1}#2%
3927 }%
3928 \def\XINT@cuз@cnt@toofarb #1#2\Z {\XINT@cuз@cnt@toofarc #2\Z {#1}}%
3929 \def\XINT@cuз@cnt@toofarc #1#2#3#4#5#6#7#8%
3930 {%
3931   \xint@r #2\XINT@cuз@cnt@toofard 7%
3932   #3\XINT@cuз@cnt@toofard 6%
3933   #4\XINT@cuз@cnt@toofard 5%
3934   #5\XINT@cuз@cnt@toofard 4%
3935   #6\XINT@cuз@cnt@toofard 3%
3936   #7\XINT@cuз@cnt@toofard 2%
3937   #8\XINT@cuз@cnt@toofard 1%
3938   \Z #1#2#3#4#5#6#7#8%
3939 }%
3940 \def\XINT@cuз@cnt@toofard #1#2\Z #3\R #4\Z #5%
3941 {%
3942   \expandafter\XINT@cuз@cnt@toofare
3943   \the\numexpr #3\relax \R\R\R\R\R\R\R\R\Z
3944   {\the\numexpr #5-#1\relax}\R\Z
3945 }%
3946 \def\XINT@cuз@cnt@toofare #1#2#3#4#5#6#7#8%
3947 {%
3948   \xint@r #2\XINT@cuз@cnt@stopc 1%
3949   #3\XINT@cuз@cnt@stopc 2%
3950   #4\XINT@cuз@cnt@stopc 3%
3951   #5\XINT@cuз@cnt@stopc 4%
3952   #6\XINT@cuз@cnt@stopc 5%
3953   #7\XINT@cuз@cnt@stopc 6%
3954   #8\XINT@cuз@cnt@stopc 7%
3955   \Z #1#2#3#4#5#6#7#8%
3956 }%
3957 \def\XINT@cuз@cnt@checka #1#2%
3958 {%
3959   \expandafter\XINT@cuз@cnt@checkb\the\numexpr #2\relax \Z {#1}%
3960 }%
3961 \def\XINT@cuз@cnt@checkb #1%
3962 {%
3963   \xint@zero #1\expandafter\XINT@cuз@cnt@loop\xint@z
3964   0\XINT@cuз@cnt@stopa #1%
3965 }%
3966 \def\XINT@cuз@cnt@stopa #1\Z
3967 {%
3968   \XINT@cuз@cnt@stopb #1\R\R\R\R\R\R\R\R\Z %
3969 }%

```



```

3970 \def\XINT@cuз@cnt@stopb #1#2#3#4#5#6#7#8#9%
3971 {%
3972   \xint@r #2\XINT@cuз@cnt@stopc 1%
3973   #3\XINT@cuз@cnt@stopc 2%
3974   #4\XINT@cuз@cnt@stopc 3%
3975   #5\XINT@cuз@cnt@stopc 4%
3976   #6\XINT@cuз@cnt@stopc 5%
3977   #7\XINT@cuз@cnt@stopc 6%
3978   #8\XINT@cuз@cnt@stopc 7%
3979   #9\XINT@cuз@cnt@stopc 8%
3980   \Z #1#2#3#4#5#6#7#8#9%
3981 }%
3982 \def\XINT@cuз@cnt@stopc #1#2\Z #3\R #4\Z #5%
3983 {%
3984   \expandafter\XINT@cuз@cnt@stopd\expandafter
3985   {\the\numexpr #5-#1}\#3%
3986 }%
3987 \def\XINT@cuз@cnt@stopd #1#2\R #3\Z
3988 {%
3989   \expandafter\space\expandafter
3990   {\romannumeral0\XINT@rord@main }\#2%
3991   \xint@UNDEF
3992   \xint@undef\xint@undef\xint@undef\xint@undef
3993   \xint@undef\xint@undef\xint@undef\xint@undef
3994   \xint@UNDEF }\#1}%
3995 }%

```

## 21.10 \xintRaw

```

3996 \def\xintRaw {\romannumeral0\xintraw }%
3997 \def\xintraw
3998 {%
3999   \expandafter\XINT@raw\romannumeral0\XINT@infrac
4000 }%
4001 \def\XINT@raw #1%
4002 {%
4003   \ifcase\XINT@Sgn {#1}
4004   \expandafter\XINT@raw@Ba
4005   \or
4006   \expandafter\XINT@raw@A
4007   \else
4008   \expandafter\XINT@raw@Ba
4009   \fi
4010   {#1}%
4011 }%
4012 \def\XINT@raw@A #1#2#3{\xint@dsh {#2}{-#1}/#3}%
4013 \def\XINT@raw@Ba #1#2#3{\expandafter\XINT@raw@Bb
4014   \expandafter{\romannumeral0\xint@dsh {#3}{#1}}{#2}}%
4015 \def\XINT@raw@Bb #1#2{ #2/#1}%

```

**21.11 \xintNumerator**

```

4016 \def\xintNumerator {\romannumeral0\xintnumerator }%
4017 \def\xintnumerator
4018 {%
4019   \expandafter\XINT@numer\romannumeral0\XINT@infrac
4020 }%
4021 \def\XINT@numer #1%
4022 {%
4023   \ifcase\XINT@Sgn {#1}
4024     \expandafter\XINT@numer@B
4025   \or
4026     \expandafter\XINT@numer@A
4027   \else
4028     \expandafter\XINT@numer@B
4029   \fi
4030   {#1}%
4031 }%
4032 \def\XINT@numer@A #1#2#3{\xint@dsh {#2}{-#1}}%
4033 \def\XINT@numer@B #1#2#3{ #2}%

```

**21.12 \xintDenominator**

```

4034 \def\xintDenominator {\romannumeral0\xintdenominator }%
4035 \def\xintdenominator
4036 {%
4037   \expandafter\XINT@denom\romannumeral0\XINT@infrac
4038 }%
4039 \def\XINT@denom #1%
4040 {%
4041   \ifcase\XINT@Sgn {#1}
4042     \expandafter\XINT@denom@B
4043   \or
4044     \expandafter\XINT@denom@A
4045   \else
4046     \expandafter\XINT@denom@B
4047   \fi
4048   {#1}%
4049 }%
4050 \def\XINT@denom@A #1#2#3{ #3}%
4051 \def\XINT@denom@B #1#2#3{\xint@dsh {#3}{#1}}%

```

**21.13 \xintFrac**

```

4052 \def\xintFrac {\romannumeral0\xintfrac }%
4053 \def\xintfrac #1%
4054 {%
4055   \expandafter\XINT@@frac@A\romannumeral0\XINT@infrac {#1}%
4056 }%
4057 \def\XINT@@frac@A #1{\XINT@@frac@B #1\Z }%
4058 \def\XINT@@frac@B #1#2\Z

```

## 21 Package *xintfrac* implementation

```

4059 {%
4060   \xint@zero #1\XINT@@frac@C 0\XINT@@frac@D {10^{#1#2}}}%
4061 }%
4062 \def\XINT@@frac@C #1#2#3#4#5%
4063 {%
4064   \ifcase\XINT@isOne {#5}
4065   \or \xint@afterfi {\expandafter\xint@firstoftwo@andstop\xint@gobble@ii }%
4066   \fi
4067   \space
4068   \frac {#4}{#5}%
4069 }%
4070 \def\XINT@@frac@D #1#2#3%
4071 {%
4072   \ifcase\XINT@isOne {#3}
4073   \or \XINT@@frac@E
4074   \fi
4075   \space
4076   \frac {#2}{#3}#1%
4077 }%
4078 \def\XINT@@frac@E \fi #1#2#3#4{\fi \space #3\cdot }%

```

### 21.14 \xintSignedFrac

```

4079 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
4080 \def\xintsignedfrac #1%
4081 {%
4082   \expandafter\XINT@sgnfrac@a\romannumeral0\XINT@infrac {#1}%
4083 }%
4084 \def\XINT@sgnfrac@a #1#2%
4085 {%
4086   \XINT@sgnfrac@b #2\Z {#1}%
4087 }%
4088 \def\XINT@sgnfrac@b #1%
4089 {%
4090   \xint@UDsignfork
4091   #1\dummy \XINT@sgnfrac@N
4092   -\dummy {\XINT@sgnfrac@P #1}%
4093   \xint@UDkrof
4094 }%
4095 \def\XINT@sgnfrac@P #1\Z #2%
4096 {%
4097   \XINT@@frac@A {#2}{#1}%
4098 }%
4099 \def\XINT@sgnfrac@N
4100 {%
4101   \expandafter\xint@minus@andstop\romannumeral0\XINT@sgnfrac@P
4102 }%

```

### 21.15 \xintFwOver

```

4103 \def\xintFwOver {\romannumeral0\xintfwover }%

```

## 21 Package *xintfrac* implementation

```

4104 \def\xintfwover #1%
4105 {%
4106   \expandafter\XINT@fwover@A\romannumeral0\XINT@infrac {#1}%
4107 }%
4108 \def\XINT@fwover@A #1{\XINT@fwover@B #1\Z }%
4109 \def\XINT@fwover@B #1#2\Z
4110 {%
4111   \xint@zero #1\XINT@fwover@C 0\XINT@fwover@D {10^{#1#2}}%
4112 }%
4113 \def\XINT@fwover@C #1#2#3#4#5%
4114 {%
4115   \ifcase\XINT@isOne {#5}
4116     \xint@afterfi { {#4\over #5}}%
4117   \or
4118     \xint@afterfi { #4}%
4119   \fi
4120 }%
4121 \def\XINT@fwover@D #1#2#3%
4122 {%
4123   \ifcase\XINT@isOne {#3}
4124     \xint@afterfi { {#2\over #3}}%
4125   \or
4126     \xint@afterfi { #2\cdot }%
4127   \fi
4128   #1%
4129 }%

```

### 21.16 \xintSignedFwOver

```

4130 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
4131 \def\xintsignedfwover #1%
4132 {%
4133   \expandafter\XINT@sgnfwover@a\romannumeral0\XINT@infrac {#1}%
4134 }%
4135 \def\XINT@sgnfwover@a #1#2%
4136 {%
4137   \XINT@sgnfwover@b #2\Z {#1}%
4138 }%
4139 \def\XINT@sgnfwover@b #1%
4140 {%
4141   \xint@UDsignfork
4142     #1\dummy \XINT@sgnfwover@N
4143     -\dummy {\XINT@sgnfwover@P #1}%
4144   \xint@UDkrof
4145 }%
4146 \def\XINT@sgnfwover@P #1\Z #2%
4147 {%
4148   \XINT@fwover@A {#2}{#1}%
4149 }%
4150 \def\XINT@sgnfwover@N

```

```

4151 {%
4152   \expandafter\xint@minus@andstop\romannumeral0\XINT@sgnfwover@P
4153 }%

```

### 21.17 \xintREZ

```

4154 \def\xintREZ {\romannumeral0\xintrez }%
4155 \def\xintrez
4156 {%
4157   \expandafter\XINT@rez@A\romannumeral0\XINT@infrac
4158 }%
4159 \def\XINT@rez@A #1#2%
4160 {%
4161   \XINT@rez@AB #2\Z {#1}%
4162 }%
4163 \def\XINT@rez@AB #1%
4164 {%
4165   \xint@UDzerominusfork
4166   #1-\dummy \XINT@rez@zero
4167   0#1\dummy \XINT@rez@neg
4168   0-\dummy {\XINT@rez@B #1}%
4169   \xint@UDkrof
4170 }%
4171 \def\XINT@rez@zero #1\Z #2#3{ 0/1[0]}%
4172 \def\XINT@rez@neg {\expandafter\xint@minus@andstop\romannumeral0\XINT@rez@B }%
4173 \def\XINT@rez@B #1\Z
4174 {%
4175   \expandafter\XINT@rez@C\romannumeral0\XINT@factortens {#1}%
4176 }%
4177 \def\XINT@rez@C #1#2#3#4%
4178 {%
4179   \expandafter\XINT@rez@D\romannumeral0\XINT@factortens {#4}{#3}{#2}{#1}%
4180 }%
4181 \def\XINT@rez@D #1#2#3#4#5%
4182 {%
4183   \expandafter\XINT@rez@E\expandafter
4184   {\the\numexpr #3+#4-#2}{#1}{#5}%
4185 }%
4186 \def\XINT@rez@E #1#2#3{ #3/#2[#1]}%

```

### 21.18 \xintIrr

1.04 fixes a buggy \xintIrr {0}.

1.05 modifies the initial parsing and post-processing to use \xintraw and to more quickly deal with an input denominator equal to 1.

```

4187 \def\xintIrr {\romannumeral0\xintirr }%
4188 \def\xintirr #1%
4189 {%
4190   \expandafter\XINT@irr@start\romannumeral0\xintraw {#1}\Z
4191 }%

```

```

4192 \def\XINT@irr@start #1#2/#3\Z
4193 {%
4194   \ifcase\XINT@isOne {#3}
4195     \xint@afterfi
4196       {\xint@UDsignfork
4197         #1\dummy \XINT@irr@negative
4198         -\dummy {\XINT@irr@nonneg #1}%
4199         \xint@UDkrof}%
4200   \or
4201     \xint@afterfi{\XINT@irr@denomisone #1}%
4202   \fi
4203   #2\Z {#3}%
4204 }%
4205 \def\XINT@irr@denomisone #1\Z #2{ #1}%
4206 \def\XINT@irr@negative #1\Z #2{\XINT@irr@D #1\Z #2\Z \xint@minus@andstop}%
4207 \def\XINT@irr@nonneg #1\Z #2{\XINT@irr@D #1\Z #2\Z \space}%
4208 \def\XINT@irr@D #1#2\Z #3#4\Z
4209 {%
4210   \xint@UDzerosfork
4211     #3#1\dummy \XINT@irr@indeterminate
4212     #30\dummy \XINT@irr@divisionbyzero
4213     #10\dummy \XINT@irr@zero
4214     00\dummy \XINT@irr@loop@a
4215   \xint@UDkrof
4216   {#3#4}{#1#2}{#3#4}{#1#2}%
4217 }%
4218 \def\XINT@irr@indeterminate #1#2#3#4#5{\xintError:NaN\space 0/0}%
4219 \def\XINT@irr@divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
4220 \def\XINT@irr@zero #1#2#3#4#5{ 0}%
4221 \def\XINT@irr@loop@a #1#2%
4222 {%
4223   \expandafter\XINT@irr@loop@d
4224   \romannumeral0\XINT@div@prepare {#1}{#2}{#1}%
4225 }%
4226 \def\XINT@irr@loop@d #1#2%
4227 {%
4228   \XINT@irr@loop@e #2\Z
4229 }%
4230 \def\XINT@irr@loop@e #1#2\Z
4231 {%
4232   \xint@zero #1\xint@irr@loop@exit0\XINT@irr@loop@a {#1#2}%
4233 }%
4234 \def\xint@irr@loop@exit0\XINT@irr@loop@a #1#2#3#4%
4235 {%
4236   \expandafter\XINT@irr@loop@exitb\expandafter
4237   {\romannumeral0\xintquo {#3}{#2}}%
4238   {\romannumeral0\xintquo {#4}{#2}}%
4239 }%
4240 \def\XINT@irr@loop@exitb #1#2%

```

```

4241 {%
4242   \expandafter\XINT@irr@finish\expandafter {#2}{#1}%
4243 }%
4244 \def\XINT@irr@finish #1#2#3%
4245 {%
4246   \ifcase\XINT@isOne {#2}
4247     \xint@afterfi {#3#1/#2}%
4248   \or
4249     \xint@afterfi {#3#1}%
4250   \fi
4251 }%

```

### 21.19 \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 \xintraw). This way, macros such as \xintQuo can be modified with minimal overhead to accept fractional input as long as it evaluates to an integer.

```

4252 \def\xintNum {\romannumeral0\xintnum }%
4253 \def\xintnum #1{\expandafter\XINT@intcheck\romannumeral0\xintirr {#1}/\W\Z }%
4254 \def\XINT@intcheck #1/#2#3\Z
4255 {%
4256   \xint@w #2\xint@gobble@ii\W\xintError:NotAnInteger
4257   \space #1%
4258 }%

```

### 21.20 \xintJrr

Modified similarly as \xintIrr in release 1.05

```

4259 \def\xintJrr {\romannumeral0\xintjrr }%
4260 \def\xintjrr #1%
4261 {%
4262   \expandafter\XINT@jrr@start\romannumeral0\xintraw {#1}\Z
4263 }%
4264 \def\XINT@jrr@start #1#2/#3\Z
4265 {%
4266   \ifcase\XINT@isOne {#3}
4267     \xint@afterfi
4268     {\xint@UDsignfork
4269       #1\dummy \XINT@jrr@negative
4270       -\dummy {\XINT@jrr@nonneg #1}%
4271     \xint@UDkrof}%
4272   \or
4273     \xint@afterfi{\XINT@jrr@denomisone #1}%

```

## 21 Package *xintfrac* implementation

```

4274 \fi
4275 #2\Z {#3}%
4276 }%
4277 \def\XINT@jrr@denomisine #1\Z #2{ #1}%
4278 \def\XINT@jrr@negative #1\Z #2{\XINT@jrr@D #1\Z #2\Z \xint@minus@andstop }%
4279 \def\XINT@jrr@nonneg #1\Z #2{\XINT@jrr@D #1\Z #2\Z \space}%
4280 \def\XINT@jrr@D #1#2\Z #3#4\Z
4281 {%
4282 \xint@UDzerosfork
4283 #3#1\dummy \XINT@jrr@indeterminate
4284 #30\dummy \XINT@jrr@divisionbyzero
4285 #10\dummy \XINT@jrr@zero
4286 00\dummy \XINT@jrr@loop@a
4287 \xint@UDkrof
4288 {#3#4}{#1#2}1001%
4289 }%
4290 \def\XINT@jrr@indeterminate #1#2#3#4#5#6#7{\xintError:NaN\space 0/0}%
4291 \def\XINT@jrr@divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
4292 \def\XINT@jrr@zero #1#2#3#4#5#6#7{ 0}%
4293 \def\XINT@jrr@loop@a #1#2%
4294 {%
4295 \expandafter\XINT@jrr@loop@b
4296 \romannumeral0\XINT@div@prepare {#1}{#2}{#1}%
4297 }%
4298 \def\XINT@jrr@loop@b #1#2#3#4#5#6#7%
4299 {%
4300 \expandafter \XINT@jrr@loop@c \expandafter
4301 {\romannumeral0\xintiadd{\XINT@Mul{#4}{#1}}{#6}}%
4302 {\romannumeral0\xintiadd{\XINT@Mul{#5}{#1}}{#7}}%
4303 {#2}{#3}{#4}{#5}%
4304 }%
4305 \def\XINT@jrr@loop@c #1#2%
4306 {%
4307 \expandafter \XINT@jrr@loop@d \expandafter{#2}{#1}%
4308 }%
4309 \def\XINT@jrr@loop@d #1#2#3#4%
4310 {%
4311 \XINT@jrr@loop@e #3\Z {#4}{#2}{#1}%
4312 }%
4313 \def\XINT@jrr@loop@e #1#2\Z
4314 {%
4315 \xint@zero #1\xint@jrr@loop@exit0\XINT@jrr@loop@a {#1#2}%
4316 }%
4317 \def\xint@jrr@loop@exit0\XINT@jrr@loop@a #1#2#3#4#5#6%
4318 {%
4319 \XINT@irr@finish {#3}{#4}%
4320 }%

```



**21.21 \xintTrunc, \xintiTrunc**

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

```

4321 \def\xintTrunc {\romannumeral0\xinttrunc }%
4322 \def\xintiTrunc {\romannumeral0\xintitrunc }%
4323 \def\xinttrunc #1%
4324 {%
4325   \expandafter\xINT@trunc\expandafter {\the\numexpr #1}%
4326 }%
4327 \def\xINT@trunc #1#2%
4328 {%
4329   \expandafter\xINT@trunc@G
4330   \romannumeral0\expandafter\xINT@trunc@A
4331   \romannumeral0\xINT@infrac {#2}{#1}{#1}%
4332 }%
4333 \def\xintitrunc #1%
4334 {%
4335   \expandafter\xINT@itrunc\expandafter {\the\numexpr #1}%
4336 }%
4337 \def\xINT@itrunc #1#2%
4338 {%
4339   \expandafter\xINT@itrunc@G
4340   \romannumeral0\expandafter\xINT@trunc@A
4341   \romannumeral0\xINT@infrac {#2}{#1}{#1}%
4342 }%
4343 \def\xINT@trunc@A #1#2#3#4%
4344 {%
4345   \expandafter\xINT@trunc@checkifzero
4346   \expandafter{\the\numexpr #1+#4}#2\Z {#3}%
4347 }%
4348 \def\xINT@trunc@checkifzero #1#2#3\Z
4349 {%
4350   \xint@zero #2\xINT@trunc@iszero0\xINT@trunc@B {#1}{#2#3}%
4351 }%
4352 \def\xINT@trunc@iszero #1#2#3#4#5{ 0\Z 0}%
4353 \def\xINT@trunc@B #1%
4354 {%
4355   \ifcase\xINT@Sgn {#1}
4356     \expandafter\xINT@trunc@D
4357   \or
4358     \expandafter\xINT@trunc@D
4359   \else
4360     \expandafter\xINT@trunc@C
4361   \fi
4362   {#1}%
4363 }%
4364 \def\xINT@trunc@C #1#2#3%
4365 {%
4366   \expandafter \xINT@trunc@E

```

## 21 Package *xintfrac* implementation

```

4367 \romannumeral0\xint@dsh {#3}{#1}\Z #2\Z
4368 }%
4369 \def\xINT@trunc@D #1#2%
4370 {%
4371 \expandafter \XINT@trunc@DE \expandafter
4372 {\romannumeral0\xint@dsh {#2}{-#1}}%
4373 }%
4374 \def\xINT@trunc@DE #1#2{\XINT@trunc@E #2\Z #1\Z }%
4375 \def\xINT@trunc@E #1#2\Z #3#4\Z
4376 {%
4377 \xint@UDsignsfork
4378 #1#3\dummy \XINT@trunc@minusminus
4379 #1-\dummy {\XINT@trunc@minusplus #3}%
4380 #3-\dummy {\XINT@trunc@plusminus #1}%
4381 --\dummy {\XINT@trunc@plusplus #3#1}%
4382 \xint@UDkrof
4383 {#4}{#2}%
4384 }%
4385 \def\xINT@trunc@minusminus #1#2{\xintiquo {#1}{#2}\Z \space}%
4386 \def\xINT@trunc@minusplus #1#2#3{\xintiquo {#1#2}{#3}\Z \xint@minus@andstop}%
4387 \def\xINT@trunc@plusminus #1#2#3{\xintiquo {#2}{#1#3}\Z \xint@minus@andstop}%
4388 \def\xINT@trunc@plusplus #1#2#3#4{\xintiquo {#1#3}{#2#4}\Z \space}%
4389 \def\xINT@itrunc@G #1#2\Z #3#4%
4390 {%
4391 \xint@zero #1\XINT@trunc@zero 0\xint@firstoftwo {#3#1#2}0%
4392 }%
4393 \def\xINT@trunc@G #1\Z #2#3%
4394 {%
4395 \xint@zero #2\XINT@trunc@zero 0%
4396 \expandafter\xINT@trunc@H\expandafter
4397 {\the\numexpr\romannumeral0\xINT@length {#1}-#3}{#3}{#1}#2%
4398 }%
4399 \def\xINT@trunc@zero 0#10{ 0}%
4400 \def\xINT@trunc@H #1#2%
4401 {%
4402 \ifnum #1 > 0
4403 \xint@afterfi {\XINT@trunc@Ha {#2}}%
4404 \else
4405 \xint@afterfi {\XINT@trunc@Hb {-#1}}% -0,--1,--2, ....
4406 \fi
4407 }%
4408 \def\xINT@trunc@Ha
4409 {%
4410 \expandafter\xINT@trunc@Haa\romannumeral0\xintdecsplit
4411 }%
4412 \def\xINT@trunc@Haa #1#2#3%
4413 {%
4414 #3#1.#2%
4415 }%

```

```

4416 \def\XINT@trunc@Hb #1#2#3%
4417 {%
4418   \expandafter #3\expandafter0\expandafter.%
4419   \romannumeral0\XINT@dsx@zeroloop {#1}\Z {}#2% #1=-0 possible!
4420 }%

```

## 21.22 \xintRound, \xintiRound

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

```

4421 \def\xintRound {\romannumeral0\xintround }%
4422 \def\xintiRound {\romannumeral0\xintiround }%
4423 \def\xintround #1%
4424 {%
4425   \expandafter\XINT@round\expandafter {\the\numexpr #1}%
4426 }%
4427 \def\XINT@round
4428 {%
4429   \expandafter\XINT@trunc@G\romannumeral0\XINT@round@A
4430 }%
4431 \def\xintiround #1%
4432 {%
4433   \expandafter\XINT@iround\expandafter {\the\numexpr #1}%
4434 }%
4435 \def\XINT@iround
4436 {%
4437   \expandafter\XINT@itrunc@G\romannumeral0\XINT@round@A
4438 }%
4439 \def\XINT@round@A #1#2%
4440 {%
4441   \expandafter\XINT@round@B
4442   \romannumeral0\expandafter\XINT@trunc@A
4443   \romannumeral0\XINT@infrac {#2}{\the\numexpr #1+1\relax}{#1}%
4444 }%
4445 \def\XINT@round@B #1\Z
4446 {%
4447   \expandafter\XINT@round@C
4448   \romannumeral0\XINT@rord@main {}#1%
4449   \xint@UNDEF
4450   \xint@undef\xint@undef\xint@undef\xint@undef
4451   \xint@undef\xint@undef\xint@undef\xint@undef
4452   \xint@UNDEF
4453   \Z
4454 }%
4455 \def\XINT@round@C #1%
4456 {%
4457   \ifnum #1<5
4458     \expandafter\XINT@round@Daa
4459   \else

```

```

4460      \expandafter\XINT@round@Dba
4461      \fi
4462 }%
4463 \def\XINT@round@Daa #1%
4464 {%
4465     \xint@z #1\XINT@round@Daz\Z \XINT@round@Da #1%
4466 }%
4467 \def\XINT@round@Daz\Z \XINT@round@Da \Z { 0\Z }%
4468 \def\XINT@round@Da #1\Z
4469 {%
4470     \XINT@rord@main {}#1%
4471     \xint@UNDEF
4472     \xint@undef\xint@undef\xint@undef\xint@undef
4473     \xint@undef\xint@undef\xint@undef\xint@undef
4474     \xint@UNDEF \Z
4475 }%
4476 \def\XINT@round@Dba #1%
4477 {%
4478     \xint@z #1\XINT@round@Dbz\Z \XINT@round@Db #1%
4479 }%
4480 \def\XINT@round@Dbz\Z \XINT@round@Db \Z { 1\Z }%
4481 \def\XINT@round@Db #1\Z
4482 {%
4483     \XINT@addm@A 0{}1000\W\X\Y\Z #1000\W\X\Y\Z \Z
4484 }%

```

### 21.23 \xintAdd

```

4485 \def\xintAdd {\romannumeral0\xintadd }%
4486 \def\xintadd #1%
4487 {%
4488     \expandafter\xint@fadd\expandafter {\romannumeral0\XINT@infrac {#1}}%
4489 }%
4490 \def\xint@fadd #1#2{\expandafter\XINT@fadd@A\romannumeral0\XINT@infrac{#2}#1}%
4491 \def\XINT@fadd@A #1#2#3#4%
4492 {%
4493     \ifnum #4 > #1
4494         \xint@afterfi {\XINT@fadd@B {#1}}%
4495     \else
4496         \xint@afterfi {\XINT@fadd@B {#4}}%
4497     \fi
4498     {#1}{#4}{#2}{#3}%
4499 }%
4500 \def\XINT@fadd@B #1#2#3#4#5#6#7%
4501 {%
4502     \expandafter\XINT@fadd@C\expandafter
4503     {\romannumeral0\xintimul {#7}{#5}}%
4504     {\romannumeral0\xintiadd
4505     {\romannumeral0\xintimul {\xintDSH {\the\numexpr -#3+#1\relax}{#6}}{#5}}%

```

```

4506   {\romannumeral0\xintimul {#7}{\xintDSH {\the\numexpr -#2+#1\relax}{#4}}}%
4507   }%
4508   {#1}%
4509 }%
4510 \def\xINT@fadd@C #1#2#3%
4511 {%
4512   \expandafter\xINT@fadd@D\expandafter {#2}{#3}{#1}%
4513 }%
4514 \def\xINT@fadd@D #1#2{\XINT@outfrac {#2}{#1}}%

```

## 21.24 \xintSub

```

4515 \def\xintSub {\romannumeral0\xintsub }%
4516 \def\xintsub #1%
4517 {%
4518   \expandafter\xint@fsub\expandafter {\romannumeral0\xINT@infrac {#1}}%
4519 }%
4520 \def\xint@fsub #1#2%
4521   {\expandafter\xINT@fsub@A\romannumeral0\xINT@infrac {#2}{#1}}%
4522 \def\xINT@fsub@A #1#2#3#4%
4523 {%
4524   \ifnum #4 > #1
4525     \xint@afterfi {\XINT@fsub@B {#1}}%
4526   \else
4527     \xint@afterfi {\XINT@fsub@B {#4}}%
4528   \fi
4529   {#1}{#4}{#2}{#3}%
4530 }%
4531 \def\xINT@fsub@B #1#2#3#4#5#6#7%
4532 {%
4533   \expandafter\xINT@fsub@C\expandafter
4534   {\romannumeral0\xintimul {#7}{#5}}%
4535   {\romannumeral0\xintisub
4536   {\romannumeral0\xintimul {\xintDSH {\the\numexpr -#3+#1\relax}{#6}}{#5}}%
4537   {\romannumeral0\xintimul {#7}{\xintDSH {\the\numexpr -#2+#1\relax}{#4}}}%
4538   }%
4539   {#1}%
4540 }%
4541 \def\xINT@fsub@C #1#2#3%
4542 {%
4543   \expandafter\xINT@fsub@D\expandafter {#2}{#3}{#1}%
4544 }%
4545 \def\xINT@fsub@D #1#2{\XINT@outfrac {#2}{#1}}%

```

## 21.25 \xintSum, \xintSumExpr

```

4546 \def\xintSum {\romannumeral0\xintsum }%
4547 \def\xintsum #1{\xintsumexpr #1\relax }%
4548 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
4549 \def\xintsumexpr {\expandafter\xINT@fsumexpr\romannumeral-‘0}%
4550 \def\xINT@fsumexpr {\XINT@fsum@loop@a {0[0]}}%

```

```

4551 \def\XINT@fsum@loop@a #1#2%
4552 {%
4553   \expandafter\XINT@fsum@loop@b \romannumeral-'0#2\Z {#1}%
4554 }%
4555 \def\XINT@fsum@loop@b #1%
4556 {%
4557   \xint@relax #1\XINT@fsum@finished\relax
4558   \XINT@fsum@loop@c #1%
4559 }%
4560 \def\XINT@fsum@loop@c #1\Z #2%
4561 {%
4562   \expandafter\XINT@fsum@loop@a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
4563 }%
4564 \def\XINT@fsum@finished #1\Z #2{ #2}%

```

**21.26 \xintMul**

```

4565 \def\xintMul {\romannumeral0\xintmul }%
4566 \def\xintmul #1%
4567 {%
4568   \expandafter\xint@fmul\expandafter {\romannumeral0\XINT@infrac {#1}}%
4569 }%
4570 \def\xint@fmul #1#2%
4571   {\expandafter\XINT@fmul@A\romannumeral0\XINT@infrac {#2}{#1}}%
4572 \def\XINT@fmul@A #1#2#3#4#5#6%
4573 {%
4574   \expandafter\XINT@fmul@B
4575   \expandafter{\the\numexpr #1+#4\expandafter}%
4576   \expandafter{\romannumeral0\xintimul {#6}{#3}}%
4577   {\romannumeral0\xintimul {#5}{#2}}%
4578 }%
4579 \def\XINT@fmul@B #1#2#3%
4580 {%
4581   \expandafter \XINT@fmul@C \expandafter{#3}{#1}{#2}%
4582 }%
4583 \def\XINT@fmul@C #1#2{\XINT@outfrac {#2}{#1}}%

```

**21.27 \xintSqr**

```

4584 \def\xintSqr {\romannumeral0\xintsqr }%
4585 \def\xintsqr #1%
4586 {%
4587   \expandafter\xint@fsqr\expandafter{\romannumeral0\XINT@infrac {#1}}%
4588 }%
4589 \def\xint@fsqr #1{\XINT@fmul@A #1#1}%

```

**21.28 \xintPow**

Modified in 1.06 to give the exponent to a \numexpr

```

4590 \def\xintPow {\romannumeral0\xintpow }%
4591 \def\xintpow #1%

```

```

4592 {%
4593   \expandafter\xint@fpow\expandafter {\romannumeral0\xint@infrac {#1}}%
4594 }%
4595 \def\xint@fpow #1#2%
4596 {%
4597   \expandafter\xint@fpow@fork\the\numexpr #2\relax\Z #1%
4598 }%
4599 \def\xint@fpow@fork #1#2\Z
4600 {%
4601   \xint@UDzerominusfork
4602   #1-\dummy \XINT@fpow@zero
4603   0#1\dummy \XINT@fpow@neg
4604   0-\dummy {\XINT@fpow@pos #1}%
4605   \xint@UDkrof
4606   {#2}%
4607 }%
4608 \def\xint@fpow@zero #1#2#3#4%
4609 {%
4610   \space 1[0]%
4611 }%
4612 \def\xint@fpow@pos #1#2#3#4#5%
4613 {%
4614   \expandafter\xint@fpow@pos@A\expandafter
4615   {\the\numexpr #1#2*#3\expandafter}\expandafter
4616   {\romannumeral0\xintipow {#5}{#1#2}}%
4617   {\romannumeral0\xintipow {#4}{#1#2}}%
4618 }%
4619 \def\xint@fpow@neg #1#2#3#4%
4620 {%
4621   \expandafter\xint@fpow@pos@A\expandafter
4622   {\the\numexpr -#1*#2\expandafter}\expandafter
4623   {\romannumeral0\xintipow {#3}{#1}}%
4624   {\romannumeral0\xintipow {#4}{#1}}%
4625 }%
4626 \def\xint@fpow@pos@A #1#2#3%
4627 {%
4628   \expandafter\xint@fpow@pos@B\expandafter {#3}{#1}{#2}%
4629 }%
4630 \def\xint@fpow@pos@B #1#2{\XINT@outfrac {#2}{#1}}%

```

### 21.29 \xintPrd, \xintProductExpr

```

4631 \def\xintPrd {\romannumeral0\xintprd }%
4632 \def\xintprd #1{\xintproductexpr #1\relax }%
4633 \def\xintProductExpr {\romannumeral0\xintproductexpr }%
4634 \def\xintproductexpr {\expandafter\xint@fproductexpr \romannumeral-‘0}%
4635 \def\xint@fproductexpr {\XINT@fprod@loop@a {1[0]}}%
4636 \def\xint@fprod@loop@a #1#2%
4637 {%

```

```

4638 \expandafter\XINT@fprod@loop@b \romannumeral-'\0#2\Z {#1}%
4639 }%
4640 \def\XINT@fprod@loop@b #1%
4641 {%
4642 \xint@relax #1\XINT@fprod@finished\relax
4643 \XINT@fprod@loop@c #1%
4644 }%
4645 \def\XINT@fprod@loop@c #1\Z #2%
4646 {%
4647 \expandafter\XINT@fprod@loop@a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
4648 }%
4649 \def\XINT@fprod@finished #1\Z #2{ #2}%

```

### 21.30 \xintDiv

```

4650 \def\xintDiv {\romannumeral0\xintdiv }%
4651 \def\xintdiv #1%
4652 {%
4653 \expandafter\xint@fdiv\expandafter {\romannumeral0\XINT@infrac {#1}}%
4654 }%
4655 \def\xint@fdiv #1#2%
4656 {\expandafter\XINT@fdiv@A\romannumeral0\XINT@infrac {#2}{#1}}%
4657 \def\XINT@fdiv@A #1#2#3#4#5#6%
4658 {%
4659 \expandafter\XINT@fdiv@B
4660 \expandafter{\the\numexpr #4-#1\expandafter}%
4661 \expandafter{\romannumeral0\xintimul {#2}{#6}}%
4662 {\romannumeral0\xintimul {#3}{#5}}%
4663 }%
4664 \def\XINT@fdiv@B #1#2#3%
4665 {%
4666 \expandafter\XINT@fdiv@C
4667 \expandafter{#3}{#1}{#2}%
4668 }%
4669 \def\XINT@fdiv@C #1#2{\XINT@outfrac {#2}{#1}}%

```

### 21.31 \xintCmp

```

4670 \def\xintCmp {\romannumeral0\xintcmp }%
4671 \def\xintcmp #1%
4672 {%
4673 \expandafter\xint@fcmp\expandafter {\romannumeral0\XINT@infrac {#1}}%
4674 }%
4675 \def\xint@fcmp #1#2{\expandafter\XINT@fcmp@A\romannumeral0\XINT@infrac {#2}{#1}}%
4676 \def\XINT@fcmp@A #1#2#3#4%
4677 {%
4678 \ifnum #4 > #1
4679 \xint@afterfi {\XINT@fcmp@B {#1}}%
4680 \else
4681 \xint@afterfi {\XINT@fcmp@B {#4}}%
4682 \fi

```



```

4683     {\#1}{\#4}{\#2}{\#3}%
4684 }%
4685 \def\XINT@fcmp@B #1#2#3#4#5#6#7%
4686 {%
4687     \xinticmp
4688     {\romannumeral0\xintimul {\xintDSH {\the\numexpr -#3+#1\relax}{\#6}}{\#5}}%
4689     {\romannumeral0\xintimul {\#7}{\xintDSH {\the\numexpr -#2+#1\relax}{\#4}}}%
4690 }%

```

### 21.32 \xintMax

```

4691 \def\xintMax {\romannumeral0\xintmax }%
4692 \def\xintmax #1%
4693 {%
4694     \expandafter\xint@fmax\expandafter {\romannumeral0\XINT@infrac {\#1}}%
4695 }%
4696 \def\xint@fmax #1#2{\expandafter\XINT@outfrac
4697     \romannumeral0\expandafter\XINT@fmax@A
4698     \romannumeral0\XINT@infrac {\#2}{\#1}}%
4699 \def\XINT@fmax@A #1#2#3#4#5#6%
4700 {%
4701     \ifnum #4 > #1
4702         \xint@afterfi {\XINT@fmax@B {\#1}}%
4703     \else
4704         \xint@afterfi {\XINT@fmax@B {\#4}}%
4705     \fi
4706     {\#1}{\#4}{\#2}{\#3}{\#5}{\#6}{\#4}{\#5}{\#6}}{\#1}{\#2}{\#3}}%
4707 }%
4708 \def\XINT@fmax@B #1#2#3#4#5#6#7%
4709 {%
4710     \expandafter\XINT@fmax@C\expandafter
4711     {\romannumeral0\xintimul {\xintDSH {\the\numexpr -#3+#1\relax}{\#6}}{\#5}}%
4712     {\romannumeral0\xintimul {\#7}{\xintDSH {\the\numexpr -#2+#1\relax}{\#4}}}%
4713 }%
4714 \def\XINT@fmax@C #1#2%
4715 {%
4716     \expandafter\XINT@max@fork #2\Z #1\Z
4717 }%

```

### 21.33 \xintMin

```

4718 \def\xintMin {\romannumeral0\xintmin }%
4719 \def\xintmin #1%
4720 {%
4721     \expandafter\xint@fmin\expandafter {\romannumeral0\XINT@infrac {\#1}}%
4722 }%
4723 \def\xint@fmin #1#2%
4724 {%
4725     \expandafter\XINT@outfrac
4726     \romannumeral0\expandafter\XINT@fmin@A
4727     \romannumeral0\XINT@infrac {\#2}{\#1}}%

```

```

4728 }%
4729 \def\XINT@fmin@A #1#2#3#4#5#6%
4730 {%
4731   \ifnum #4 > #1
4732     \xint@afterfi {\XINT@fmin@B {#1}}%
4733   \else
4734     \xint@afterfi {\XINT@fmin@B {#4}}%
4735   \fi
4736   {#1}{#4}{#2}{#3}{#5}{#6}{#4}{#5}{#6}{#1}{#2}{#3}}%
4737 }%
4738 \def\XINT@fmin@B #1#2#3#4#5#6#7%
4739 {%
4740   \expandafter\XINT@fmin@C\expandafter
4741   {\romannumeral0\xintimul {\xintDSH {\the\numexpr -#3+#1\relax}{#6}}{#5}}%
4742   {\romannumeral0\xintimul {#7}{\xintDSH {\the\numexpr -#2+#1\relax}{#4}}}%
4743 }%
4744 \def\XINT@fmin@C #1#2%
4745 {%
4746   \expandafter\XINT@min@fork #2\Z #1\Z
4747 }%

```

**21.34 \xintAbs**

```

4748 \def\xintAbs {\romannumeral0\xintabs }%
4749 \def\xintabs #1%
4750 {%
4751   \expandafter\xint@fabs\romannumeral0\XINT@infrac {#1}%
4752 }%
4753 \def\xint@fabs #1#2%
4754 {%
4755   \expandafter\XINT@outfrac\expandafter
4756   {\the\numexpr #1\expandafter}\expandafter
4757   {\romannumeral0\XINT@abs #2}%
4758 }%

```

**21.35 \xintOpp**

```

4759 \def\xintOpp {\romannumeral0\xintopp }%
4760 \def\xintopp #1%
4761 {%
4762   \expandafter\xint@fopp\romannumeral0\XINT@infrac {#1}%
4763 }%
4764 \def\xint@fopp #1#2%
4765 {%
4766   \expandafter\XINT@outfrac\expandafter
4767   {\the\numexpr #1\expandafter}\expandafter
4768   {\romannumeral0\XINT@opp #2}%
4769 }%

```

**21.36 \xintSgn**

```

4770 \def\xintSgn {\romannumeral0\xintsgn }%
4771 \def\xintsgn #1%
4772 {%
4773   \expandafter\xint@fsgn\romannumeral0\XINT@infrac {#1}%
4774 }%
4775 \def\xint@fsgn #1#2#3{\xintisgn {#2}}%

```

**21.37 \xintGeq**

```

4776 \def\xintGeq {\romannumeral0\xintgeq }%
4777 \def\xintgeq #1%
4778 {%
4779   \expandafter\xint@xgeq\expandafter{\romannumeral0\xintnum {#1}}%
4780 }%
4781 \def\xint@xgeq #1#2%
4782 {%
4783   \expandafter\XINT@geq@fork\romannumeral0\xintnum {#2}\Z #1\Z
4784 }%

```

**21.38 \xintDivision, \xintQuo, \xintRem**

```

4785 \def\xintDivision {\romannumeral0\xintdivision }%
4786 \def\xintdivision #1%
4787 {%
4788   \expandafter\xint@xdivision\expandafter{\romannumeral0\xintnum {#1}}%
4789 }%
4790 \def\xint@xdivision #1#2%
4791 {%
4792   \expandafter\XINT@div@fork\romannumeral0\xintnum {#2}\Z #1\Z
4793 }%
4794 \def\xintQuo {\romannumeral0\xintquo }%
4795 \def\xintRem {\romannumeral0\xintrem }%
4796 \def\xintquo {\expandafter\xint@firstoftwo@andstop
4797   \romannumeral0\xintdivision }%
4798 \def\xintrem {\expandafter\xint@secondoftwo@andstop
4799   \romannumeral0\xintdivision }%

```

**21.39 \xintFDg, \xintLDg, \xintMON, \xintMMON, \xintOdd**

```

4800 \def\xintFDg {\romannumeral0\xintfdg }%
4801 \def\xintfdg #1%
4802 {%
4803   \expandafter\XINT@fdg\romannumeral0\xintnum {#1}\W\Z
4804 }%
4805 \def\xintLDg {\romannumeral0\xintlbg }%
4806 \def\xintlbg #1%
4807 {%
4808   \expandafter\XINT@ldg\expandafter{\romannumeral0\xintnum {#1}}%
4809 }%
4810 \def\xintMON {\romannumeral0\xintmon }%
4811 \def\xintmon #1%

```

```

4812 {%
4813   \ifodd\xintLDg {#1}
4814     \xint@afterfi{ -1}%
4815   \else
4816     \xint@afterfi{ 1}%
4817   \fi
4818 }%
4819 \def\xintMMON {\romannumeral0\xintmmon }%
4820 \def\xintmmon #1%
4821 {%
4822   \ifodd\xintLDg {#1}
4823     \xint@afterfi{ 1}%
4824   \else
4825     \xint@afterfi{ -1}%
4826   \fi
4827 }%
4828 \def\xintOdd {\romannumeral0\xintodd }%
4829 \def\xintodd #1%
4830 {%
4831   \ifodd\xintLDg{#1}
4832     \xint@afterfi{ 1}%
4833   \else
4834     \xint@afterfi{ 0}%
4835   \fi
4836 }%
4837 \XINT@frac@restorecatcodes@endinput%

```

## 22 Package **xintseries** implementation

The commenting is currently (2013/05/07) very sparse.

### Contents

1	Catcodes, $\varepsilon$ -T <sub>E</sub> X and reload detection . . . . .	185	7	\xintPowerSeries . . . . .	190
2	Confirmation of <b>xintfrac</b> loading . . . . .	186	8	\xintPowerSeriesX . . . . .	191
3	Catcodes . . . . .	187	9	\xintRationalSeries . . . . .	192
4	Package identification . . . . .	188	10	\xintRationalSeriesX . . . . .	193
5	\xintSeries . . . . .	189	11	\xintFxFtPowerSeries . . . . .	193
6	\xintiSeries . . . . .	189	12	\xintFxFtPowerSeriesX . . . . .	194

### 22.1 Catcodes, $\varepsilon$ -T<sub>E</sub>X and reload detection

The code for reload detection is copied from HEIKO OBERDIEK's packages, and adapted here to check for previous loading of the **xintfrac** package.

The method for catcodes is slightly different, but still directly inspired by these packages.

```

4838 \begingroup\catcode61\catcode48\catcode32=10\relax%
4839 \catcode13=5 % ^^M

```

```

4840 \endlinechar=13 %
4841 \catcode123=1 % {
4842 \catcode125=2 % }
4843 \catcode64=11 % @
4844 \catcode35=6 % #
4845 \catcode44=12 % ,
4846 \catcode45=12 % -
4847 \catcode46=12 % .
4848 \catcode58=12 % :
4849 \def\space { }%
4850 \let\z\endgroup
4851 \expandafter\let\expandafter\x\csname ver@xintseries.sty\endcsname
4852 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
4853 \expandafter
4854 \ifx\csname PackageInfo\endcsname\relax
4855 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
4856 \else
4857 \def\y#1#2{\PackageInfo{#1}{#2}}%
4858 \fi
4859 \expandafter
4860 \ifx\csname numexpr\endcsname\relax
4861 \y{xintseries}{\numexpr not available, aborting input}%
4862 \aftergroup\endinput
4863 \else
4864 \ifx\x\relax % plain-TeX, first loading of xintseries.sty
4865 \ifx\w\relax % but xintfrac.sty not yet loaded.
4866 \y{xintseries}{Package xintfrac is required}%
4867 \y{xintseries}{Will try \string\input\space xintfrac.sty}%
4868 \def\z{\endgroup\input xintfrac.sty\relax}%
4869 \fi
4870 \else
4871 \def\empty {}%
4872 \ifx\x\empty % LaTeX, first loading,
4873 % variable is initialized, but \ProvidesPackage not yet seen
4874 \ifx\w\relax % xintfrac.sty not yet loaded.
4875 \y{xintseries}{Package xintfrac is required}%
4876 \y{xintseries}{Will try \string\RequirePackage{xintfrac}}%
4877 \def\z{\endgroup\RequirePackage{xintfrac}}%
4878 \fi
4879 \else
4880 \y{xintseries}{I was already loaded, aborting input}%
4881 \aftergroup\endinput
4882 \fi
4883 \fi
4884 \fi
4885 \z%

```

## 22.2 Confirmation of **xintfrac** loading

```

4886 \begingroup\catcode61\catcode48\catcode32=10\relax%
4887 \catcode13=5 % ^^M
4888 \endlinechar=13 %
4889 \catcode123=1 % {
4890 \catcode125=2 % }
4891 \catcode64=11 % @
4892 \catcode35=6 % #
4893 \catcode44=12 % ,
4894 \catcode45=12 % -
4895 \catcode46=12 % .
4896 \catcode58=12 % :
4897 \expandafter
4898 \ifx\csname PackageInfo\endcsname\relax
4899 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
4900 \else
4901 \def\y#1#2{\PackageInfo{#1}{#2}}%
4902 \fi
4903 \def\empty {}%
4904 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
4905 \ifx\w\relax % Plain TeX, user gave a file name at the prompt
4906 \y{xintseries}{Loading of package xintfrac failed, aborting input}%
4907 \aftergroup\endinput
4908 \fi
4909 \ifx\w\empty % LaTeX, user gave a file name at the prompt
4910 \y{xintseries}{Loading of package xintfrac failed, aborting input}%
4911 \aftergroup\endinput
4912 \fi
4913 \endgroup%

```

## 22.3 Catcodes

Perhaps catcodes have changed after the loading of **xint** and **xintfrac** and prior to the current loading of **xintseries**, so we can not employ the `\XINT@restorecatcodes@endinput` in this style file. But there is no problem using `\XINT@setcatcodes`.

```

4914 \begingroup\catcode61\catcode48\catcode32=10\relax%
4915 \catcode13=5 % ^^M
4916 \endlinechar=13 %
4917 \catcode123=1 % {
4918 \catcode125=2 % }
4919 \catcode64=11 % @
4920 \def\x
4921 {%
4922 \endgroup
4923 \edef\XINT@series@restorecatcodes@endinput
4924 {%
4925 \catcode93=\the\catcode93 % ]
4926 \catcode91=\the\catcode91 % [
4927 \catcode96=\the\catcode96 % '
4928 \catcode47=\the\catcode47 % /

```

```

4929      \catcode41=\the\catcode41    % )
4930      \catcode40=\the\catcode40    % (
4931      \catcode42=\the\catcode42    % *
4932      \catcode43=\the\catcode43    % +
4933      \catcode62=\the\catcode62    % >
4934      \catcode60=\the\catcode60    % <
4935      \catcode58=\the\catcode58    % :
4936      \catcode46=\the\catcode46    % .
4937      \catcode45=\the\catcode45    % -
4938      \catcode44=\the\catcode44    % ,
4939      \catcode35=\the\catcode35    % #
4940      \catcode64=\the\catcode64    % @
4941      \catcode125=\the\catcode125  % }
4942      \catcode123=\the\catcode123  % {
4943      \endlinechar=\the\endlinechar
4944      \catcode13=\the\catcode13    % ^^M
4945      \catcode32=\the\catcode32    %
4946      \catcode61=\the\catcode61\relax % =
4947      \noexpand\endinput
4948  }%
4949  \XINT@setcatcodes
4950  \catcode91=12 % [
4951  \catcode93=12 % ]
4952 }%
4953 \x

```

## 22.4 Package identification

```

4954 \begingroup
4955   \catcode58=12 % :
4956   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
4957     \def\x#1#2#3[#4]{\endgroup
4958       \immediate\write-1{Package: #3 #4}%
4959       \xdef#1{#4}%
4960     }%
4961   \else
4962     \def\x#1#2[#3]{\endgroup
4963       #2[#3]}%
4964     \ifx#1\@undefined
4965       \xdef#1{#3}%
4966     \fi
4967     \ifx#1\relax
4968       \xdef#1{#3}%
4969     \fi
4970   }%
4971 \fi
4972 \expandafter\x\csname ver@xintseries.sty\endcsname
4973 \ProvidesPackage{xintseries}%
4974 [2013/05/07 v1.06 Expandable partial sums with xint package (jfb)]%

```

## 22.5 \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.

```

4975 \def\xintSeries {\romannumeral0\xintseries }%
4976 \def\xintseries #1#2%
4977 {%
4978   \expandafter\XINT@series@i\expandafter {\the\numexpr #2}{#1}%
4979 }%
4980 \def\XINT@series@i #1#2%
4981 {%
4982   \expandafter\XINT@series@ii\expandafter {\the\numexpr #2}{#1}%
4983 }%
4984 \def\XINT@series@ii #1#2#3%
4985 {%
4986   \ifnum #2<#1
4987     \xint@afterfi { 0[0]}%
4988   \else
4989     \xint@afterfi {\XINT@series@loop {#1}{0}{#2}{#3}}%
4990   \fi
4991 }%
4992 \def\XINT@series@loop #1#2#3#4%
4993 {%
4994   \ifnum #3>#1 \else \XINT@series@exit \fi
4995   \expandafter\XINT@series@loop\expandafter
4996   {\the\numexpr #1+1\expandafter }\expandafter
4997   {\romannumeral0\xintadd {#2}{#4{#1}}}%
4998   {#3}{#4}%
4999 }%
5000 \def\XINT@series@exit \fi #1#2#3#4#5#6#7#8%
5001 {%
5002   \fi\xint@gobble@ii #6%
5003 }%

```

## 22.6 \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.

```

5004 \def\xintiSeries {\romannumeral0\xintiseries }%
5005 \def\xintiseries #1#2%
5006 {%
5007   \expandafter\XINT@iseries@i\expandafter {\the\numexpr #2}{#1}%
5008 }%
5009 \def\XINT@iseries@i #1#2%
5010 {%
5011   \expandafter\XINT@iseries@ii\expandafter {\the\numexpr #2}{#1}%
5012 }%
5013 \def\XINT@iseries@ii #1#2#3%

```



```

5014 {%
5015   \ifnum #2<#1
5016     \xint@afterfi { 0}%
5017   \else
5018     \xint@afterfi {\XINT@iseries@loop {#1}{0}{#2}{#3}}%
5019   \fi
5020 }%
5021 \def\XINT@iseries@loop #1#2#3#4%
5022 {%
5023   \ifnum #3>#1 \else \XINT@iseries@exit \fi
5024   \expandafter\XINT@iseries@loop\expandafter
5025   {\the\numexpr #1+1\expandafter }\expandafter
5026   {\romannumeral0\xintiadd {#2}{#4{#1}}}%
5027   {#3}{#4}%
5028 }%
5029 \def\XINT@iseries@exit \fi #1#2#3#4#5#6#7#8%
5030 {%
5031   \fi\xint@gobble@ii #6%
5032 }%

```

## 22.7 \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators.

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.

```

5033 \def\xintPowerSeries {\romannumeral0\xintpowerseries }%
5034 \def\xintpowerseries #1#2%
5035 {%
5036   \expandafter\XINT@powseries@i\expandafter {\the\numexpr #2}{#1}%
5037 }%
5038 \def\XINT@powseries@i #1#2%
5039 {%
5040   \expandafter\XINT@powseries@ii\expandafter {\the\numexpr #2}{#1}%
5041 }%
5042 \def\XINT@powseries@ii #1#2#3#4%
5043 {%
5044   \ifnum #2<#1
5045     \xint@afterfi { 0[0]}%
5046   \else
5047     \xint@afterfi
5048     {\XINT@powseries@loop@i {#3}{#2}}{#1}{#2}{#3}{#4}}%
5049   \fi
5050 }%
5051 \def\XINT@powseries@loop@i #1#2#3#4#5%
5052 {%

```

```

5053 \ifnum #3>#2 \else\XINT@powseries@exit@i\fi
5054 \expandafter\XINT@powseries@loop@ii\expandafter
5055 {\the\numexpr #3-1\expandafter}\expandafter
5056 {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
5057 }%
5058 \def\XINT@powseries@loop@ii #1#2#3#4%
5059 {%
5060 \expandafter\XINT@powseries@loop@i\expandafter
5061 {\romannumeral0\xintadd {#4}{#1}}{#2}}{#3}{#1}{#4}%
5062 }%
5063 \def\XINT@powseries@exit@i\fi #1#2#3#4#5#6#7#8#9%
5064 {%
5065 \fi \XINT@powseries@exit@ii #6{#7}%
5066 }%
5067 \def\XINT@powseries@exit@ii #1#2#3#4#5#6%
5068 {%
5069 \xintmul{\xintPow {#5}{#6}}{#4}%
5070 }%

```

## 22.8 \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.

```

5071 \def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
5072 \def\xintpowerseriesx #1#2%
5073 {%
5074 \expandafter\XINT@powseriesx@i\expandafter {\the\numexpr #2}{#1}%
5075 }%
5076 \def\XINT@powseriesx@i #1#2%
5077 {%
5078 \expandafter\XINT@powseriesx@ii\expandafter {\the\numexpr #2}{#1}%
5079 }%
5080 \def\XINT@powseriesx@ii #1#2#3#4%
5081 {%
5082 \ifnum #2<#1
5083 \xint@afterfi { 0[0]}%
5084 \else
5085 \xint@afterfi
5086 {\expandafter\XINT@powseriesx@pre\expandafter
5087 {\romannumeral-'0#4}{#1}{#2}{#3}%
5088 }%
5089 \fi
5090 }%
5091 \def\XINT@powseriesx@pre #1#2#3#4%
5092 {%
5093 \XINT@powseries@loop@i {#4}{#3}}{#2}{#3}{#4}{#1}%
5094 }%

```

## 22.9 \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.

```

5095 \def\xintRationalSeries {\romannumeral0\xintratseries }%
5096 \def\xintratseries #1#2%
5097 {%
5098   \expandafter\xINT@ratseries@i\expandafter {\the\numexpr #2}\{#1}%
5099 }%
5100 \def\xINT@ratseries@i #1#2%
5101 {%
5102   \expandafter\xINT@ratseries@ii\expandafter {\the\numexpr #2}\{#1}%
5103 }%
5104 \def\xINT@ratseries@ii #1#2#3#4%
5105 {%
5106   \ifnum #2<#1
5107     \xint@afterfi { 0[0]}%
5108   \else
5109     \xint@afterfi
5110     {\XINT@ratseries@loop {\#2}\{1}\{#1}\{#4}\{#3}}%
5111   \fi
5112 }%
5113 \def\xINT@ratseries@loop #1#2#3#4%
5114 {%
5115   \ifnum #1>#3 \else\xINT@ratseries@exit@i\fi
5116   \expandafter\xINT@ratseries@loop\expandafter
5117   {\the\numexpr #1-1\expandafter}\expandafter
5118   {\romannumeral0\xintadd {1}\{ \xintMul {\#2}\{#4\{#1}\}}\{#3}\{#4}%
5119 }%
5120 \def\xINT@ratseries@exit@i\fi #1#2#3#4#5#6#7#8%
5121 {%
5122   \fi \XINT@ratseries@exit@ii #6%
5123 }%
5124 \def\xINT@ratseries@exit@ii #1#2#3#4#5%
5125 {%
5126   \XINT@ratseries@exit@iii #5%
5127 }%
5128 \def\xINT@ratseries@exit@iii #1#2#3#4%
5129 {%
5130   \xintmul{\#2}\{#4}%
5131 }%
```

**22.10 \xintRationalSeriesX**

`a,b,initial,ratiofunction,x`

This computes  $F(a,x)+\dots+F(b,x)$  on the basis of the value of  $F(a,x)$  and the ratios  $F(n,x)/F(n-1,x)$ . The argument  $x$  is first expanded and it is the value resulting from this which is used then throughout. The initial term  $F(a,x)$  must be defined as one-parameter macro which will be given  $x$ .

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

```

5132 \def\xintRationalSeriesX {\romannumeral0\xintratseriesx}%
5133 \def\xintratseriesx #1#2%
5134 {%
5135   \expandafter\XINT@ratseriesx@i\expandafter {\the\numexpr #2}{#1}%
5136 }%
5137 \def\XINT@ratseriesx@i #1#2%
5138 {%
5139   \expandafter\XINT@ratseriesx@ii\expandafter {\the\numexpr #2}{#1}%
5140 }%
5141 \def\XINT@ratseriesx@ii #1#2#3#4#5%
5142 {%
5143   \ifnum #2<#1
5144     \xint@afterfi { 0[0]}%
5145   \else
5146     \xint@afterfi
5147     {\expandafter\XINT@ratseriesx@pre\expandafter
5148      {\romannumeral-'0#5}{#2}{#1}{#4}{#3}%
5149     }%
5150   \fi
5151 }%
5152 \def\XINT@ratseriesx@pre #1#2#3#4#5%
5153 {%
5154   \XINT@ratseries@loop {#2}{1}{#3}{#4}{#1}}{#5}{#1}}%
5155 }%
```

**22.11 \xintFxFtPowerSeries**

I am not too happy with this piece of code. Will make it more economical another day.

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

```

5156 \def\xintFxFtPowerSeries {\romannumeral0\xintfxptpowerseries}%
5157 \def\xintfxptpowerseries #1#2%
5158 {%
5159   \expandafter\XINT@fppowseries@i\expandafter {\the\numexpr #2}{#1}%
5160 }%
5161 \def\XINT@fppowseries@i #1#2%
5162 {%
5163   \expandafter\XINT@fppowseries@ii\expandafter {\the\numexpr #2}{#1}%
5164 }%
```

```

5164 }%
5165 \def\XINT@fppowseries@ii #1#2#3#4#5%
5166 {%
5167   \ifnum #2<#1
5168     \xint@afterfi { 0}%
5169   \else
5170     \xint@afterfi
5171     {\expandafter\XINT@fppowseries@loop@pre\expandafter
5172      {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
5173      {#1}{#4}{#2}{#3}{#5}%
5174     }%
5175   \fi
5176 }%
5177 \def\XINT@fppowseries@loop@pre #1#2#3#4#5#6%
5178 {%
5179   \ifnum #4>#2 \else\XINT@fppowseries@dont@i \fi
5180   \expandafter\XINT@fppowseries@loop@i\expandafter
5181   {\the\numexpr #2+1\expandafter}\expandafter
5182   {\romannumeral0\xintitrunc {#6}{\xintMul {#5}{#2}}{#1}}%
5183   {#1}{#3}{#4}{#5}{#6}%
5184 }%
5185 \def\XINT@fppowseries@dont@i \fi\expandafter\XINT@fppowseries@loop@i
5186   {\fi \expandafter\XINT@fppowseries@dont@ii }%
5187 \def\XINT@fppowseries@dont@ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
5188 \def\XINT@fppowseries@loop@i #1#2#3#4#5#6#7%
5189 {%
5190   \ifnum #5>#1 \else \XINT@fppowseries@exit@i \fi
5191   \expandafter\XINT@fppowseries@loop@ii\expandafter
5192   {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
5193   {#1}{#4}{#2}{#5}{#6}{#7}%
5194 }%
5195 \def\XINT@fppowseries@loop@ii #1#2#3#4#5#6#7%
5196 {%
5197   \expandafter\XINT@fppowseries@loop@i\expandafter
5198   {\the\numexpr #2+1\expandafter}\expandafter
5199   {\romannumeral0\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6}{#2}}{#1}}}%
5200   {#1}{#3}{#5}{#6}{#7}%
5201 }%
5202 \def\XINT@fppowseries@exit@i\fi\expandafter\XINT@fppowseries@loop@ii
5203   {\fi \expandafter\XINT@fppowseries@exit@ii }%
5204 \def\XINT@fppowseries@exit@ii #1#2#3#4#5#6#7%
5205 {%
5206   \xinttrunc {#7}
5207   {\xintiAdd {#4}{\xintiTrunc {#7}{\xintMul {#6}{#2}}{#1}}[-#7]}%
5208 }%

```

## 22.12 \xintFxFtPowerSeriesX

a,b,coeff,x,D

## 23 Package **xintfrac** implementation

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.

```

5209 \def\xintFxFtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx}%
5210 \def\xintfxptpowerseriesx #1#2%
5211 {%
5212   \expandafter\XINT@fppowseriesx@i\expandafter {\the\numexpr #2}{#1}%
5213 }%
5214 \def\XINT@fppowseriesx@i #1#2%
5215 {%
5216   \expandafter\XINT@fppowseriesx@ii\expandafter {\the\numexpr #2}{#1}%
5217 }%
5218 \def\XINT@fppowseriesx@ii #1#2#3#4#5%
5219 {%
5220   \ifnum #2<#1
5221     \xint@afterfi { 0}%
5222   \else
5223     \xint@afterfi
5224     {\expandafter \XINT@fppowseriesx@pre \expandafter
5225      {\romannumeral- '0#4}{#1}{#2}{#3}{#5}%
5226     }%
5227   \fi
5228 }%
5229 \def\XINT@fppowseriesx@pre #1#2#3#4#5%
5230 {%
5231   \expandafter\XINT@fppowseries@loop@pre\expandafter
5232   {\romannumeral0\xinttrunc {#5}{\xintPow {#1}{#2}}}%
5233   {#2}{#1}{#3}{#4}{#5}%
5234 }%
5235 \XINT@series@restorecatcodes@endinput%
```

## 23 Package **xintfrac** implementation

The commenting is currently (2013/05/07) very sparse.

### Contents

1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	196	10	<code>\xintFtoGC</code> . . . . .	203
2	Confirmation of <b>xintfrac</b> loading . . .	197	11	<code>\xintFtoCC</code> . . . . .	204
3	Catcodes . . . . .	197	12	<code>\xintFtoCv</code> . . . . .	205
4	Package identification . . . . .	198	13	<code>\xintFtoCCv</code> . . . . .	205
5	<code>\xintCFrac</code> . . . . .	199	14	<code>\xintCstoF</code> . . . . .	205
6	<code>\xintGCFrac</code> . . . . .	200	15	<code>\xintiCstoF</code> . . . . .	206
7	<code>\xintGctoGCx</code> . . . . .	202	16	<code>\xintGctoF</code> . . . . .	207
8	<code>\xintFtoCs</code> . . . . .	202	17	<code>\xintiGctoF</code> . . . . .	208
9	<code>\xintFtoCx</code> . . . . .	203	18	<code>\xintCstoCv</code> . . . . .	209

19	<code>\xintiCstoCv</code>	210	24	<code>\xintCntoCs</code>	215
20	<code>\xintGctoCv</code>	211	25	<code>\xintCntoGC</code>	215
21	<code>\xintiGctoCv</code>	212	26	<code>\xintGcntoGC</code>	216
22	<code>\xintCntoF</code>	213	27	<code>\xintCstoGC</code>	217
23	<code>\xintGcntoF</code>	214	28	<code>\xintGctoGC</code>	217

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

The code for reload detection is copied from HEIKO OBERDIEK's packages, and adapted here to check for previous loading of the *xintfrac* package.

The method for catcodes is slightly different, but still directly inspired by these packages.

```

5236 \begingroup\catcode61\catcode48\catcode32=10\relax%
5237   \catcode13=5      % ^^M
5238   \endlinechar=13 %
5239   \catcode123=1     % {
5240   \catcode125=2     % }
5241   \catcode64=11    % @
5242   \catcode35=6     % #
5243   \catcode44=12    % ,
5244   \catcode45=12    % -
5245   \catcode46=12    % .
5246   \catcode58=12    % :
5247   \def\space { }%
5248   \let\z\endgroup
5249   \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
5250   \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
5251   \expandafter
5252     \ifx\csname PackageInfo\endcsname\relax
5253       \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
5254     \else
5255       \def\y#1#2{\PackageInfo{#1}{#2}}%
5256     \fi
5257   \expandafter
5258   \ifx\csname numexpr\endcsname\relax
5259     \y{xintfrac}{\numexpr not available, aborting input}%
5260     \aftergroup\endinput
5261   \else
5262     \ifx\x\relax % plain-TeX, first loading of xintfrac.sty
5263       \ifx\w\relax % but xintfrac.sty not yet loaded.
5264         \y{xintfrac}{Package xintfrac is required}%
5265         \y{xintfrac}{Will try \string\input\space xintfrac.sty}%
5266         \def\z{\endgroup\input xintfrac.sty\relax}%
5267       \fi
5268     \else
5269       \def\empty {}%
5270       \ifx\x\empty % LaTeX, first loading,
5271         % variable is initialized, but \ProvidesPackage not yet seen

```

```

5272         \ifx\w\relax % xintfrac.sty not yet loaded.
5273         \y{xintfrac}{Package xintfrac is required}%
5274         \y{xintfrac}{Will try \string\RequirePackage{xintfrac}}%
5275         \def\z{\endgroup\RequirePackage{xintfrac}}%
5276         \fi
5277     \else
5278         \y{xintfrac}{I was already loaded, aborting input}%
5279         \aftergroup\endinput
5280     \fi
5281 \fi
5282 \fi
5283 \z%

```

### 23.2 Confirmation of **xintfrac** loading

```

5284 \begingroup\catcode61\catcode48\catcode32=10\relax%
5285 \catcode13=5 % ^^M
5286 \endlinechar=13 %
5287 \catcode123=1 % {
5288 \catcode125=2 % }
5289 \catcode64=11 % @
5290 \catcode35=6 % #
5291 \catcode44=12 % ,
5292 \catcode45=12 % -
5293 \catcode46=12 % .
5294 \catcode58=12 % :
5295 \expandafter
5296 \ifx\csname PackageInfo\endcsname\relax
5297     \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
5298 \else
5299     \def\y#1#2{\PackageInfo{#1}{#2}}%
5300 \fi
5301 \def\empty {}%
5302 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
5303 \ifx\w\relax % Plain TeX, user gave a file name at the prompt
5304     \y{xintfrac}{Loading of package xintfrac failed, aborting input}%
5305     \aftergroup\endinput
5306 \fi
5307 \ifx\w\empty % LaTeX, user gave a file name at the prompt
5308     \y{xintfrac}{Loading of package xintfrac failed, aborting input}%
5309     \aftergroup\endinput
5310 \fi
5311 \endgroup%

```

### 23.3 Catcodes

Perhaps catcodes have changed after the loading of **xint** and **xintfrac** and prior to the current loading of **xintfrac**, so we can not employ the `\XINT@restorecatcodes@endinput` in this style file. But there is no problem using `\XINT@setcatcodes`.



```

5312 \begingroup\catcode61\catcode48\catcode32=10\relax%
5313   \catcode13=5      % ^^M
5314   \endlinechar=13 %
5315   \catcode123=1     % {
5316   \catcode125=2     % }
5317   \catcode64=11    % @
5318   \def\x
5319   {%
5320     \endgroup
5321     \edef\XINT@cfrac@restorecatcodes@endinput
5322     {%
5323       \catcode93=\the\catcode93 % ]
5324       \catcode91=\the\catcode91 % [
5325       \catcode96=\the\catcode96 % '
5326       \catcode47=\the\catcode47 % /
5327       \catcode41=\the\catcode41 % )
5328       \catcode40=\the\catcode40 % (
5329       \catcode42=\the\catcode42 % *
5330       \catcode43=\the\catcode43 % +
5331       \catcode62=\the\catcode62 % >
5332       \catcode60=\the\catcode60 % <
5333       \catcode58=\the\catcode58 % :
5334       \catcode46=\the\catcode46 % .
5335       \catcode45=\the\catcode45 % -
5336       \catcode44=\the\catcode44 % ,
5337       \catcode35=\the\catcode35 % #
5338       \catcode64=\the\catcode64 % @
5339       \catcode125=\the\catcode125 % }
5340       \catcode123=\the\catcode123 % {
5341       \endlinechar=\the\endlinechar
5342       \catcode13=\the\catcode13 % ^^M
5343       \catcode32=\the\catcode32 %
5344       \catcode61=\the\catcode61\relax % =
5345       \noexpand\endinput
5346     }%
5347     \XINT@setcatcodes
5348     \catcode91=12 % [
5349     \catcode93=12 % ]
5350   }%
5351 \x

```

## 23.4 Package identification

```

5352 \begingroup
5353   \catcode58=12 % :
5354   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
5355     \def\x#1#2#3[#4]{\endgroup
5356       \immediate\write-1{Package: #3 #4}%
5357       \xdef#1{#4}%

```

```

5358 }%
5359 \else
5360 \def\x#1#2[#3]{\endgroup
5361 #2[#3]}%
5362 \ifx#1\@undefined
5363 \xdef#1{#3}%
5364 \fi
5365 \ifx#1\relax
5366 \xdef#1{#3}%
5367 \fi
5368 }%
5369 \fi
5370 \expandafter\x\csname ver@xintcfrac.sty\endcsname
5371 \ProvidesPackage{xintcfrac}%
5372 [2013/05/07 v1.06 Expandable continued fractions with xint package (jfb)]%

```

### 23.5 \xintCFrac

```

5373 \def\xintCFrac {\romannumeral0\xintcfrac }%
5374 \def\xintcfrac #1%
5375 {%
5376 \XINT@cfrac@opt@a #1\Z
5377 }%
5378 \def\XINT@cfrac@opt@a #1%
5379 {%
5380 \ifx#1[\XINT@cfrac@opt@b\fi \XINT@cfrac@noopt #1%
5381 }%
5382 \def\XINT@cfrac@noopt #1\Z
5383 {%
5384 \expandafter\XINT@cfrac@A\romannumeral0\xintra {#1}\Z
5385 \relax\relax
5386 }%
5387 \def\XINT@cfrac@opt@b\fi\XINT@cfrac@noopt [\Z #1]%
5388 {%
5389 \fi\csname XINT@cfrac@opt#1\endcsname
5390 }%
5391 \def\XINT@cfrac@optl #1%
5392 {%
5393 \expandafter\XINT@cfrac@A\romannumeral0\xintra {#1}\Z
5394 \relax\hfill
5395 }%
5396 \def\XINT@cfrac@optc #1%
5397 {%
5398 \expandafter\XINT@cfrac@A\romannumeral0\xintra {#1}\Z
5399 \relax\relax
5400 }%
5401 \def\XINT@cfrac@optr #1%
5402 {%
5403 \expandafter\XINT@cfrac@A\romannumeral0\xintra {#1}\Z
5404 \hfill\relax

```

```

5405 }%
5406 \def\XINT@cfrac@A #1/#2\Z
5407 {%
5408   \expandafter\XINT@cfrac@B\romannumeral0\xintidivision {#1}{#2}{#2}%
5409 }%
5410 \def\XINT@cfrac@B #1#2%
5411 {%
5412   \XINT@cfrac@C #2\Z {#1}%
5413 }%
5414 \def\XINT@cfrac@C #1%
5415 {%
5416   \xint@zero #1\XINT@cfrac@integer 0\XINT@cfrac@D #1%
5417 }%
5418 \def\XINT@cfrac@integer 0\XINT@cfrac@D 0#1\Z #2#3#4#5{ #2}%
5419 \def\XINT@cfrac@D #1\Z #2#3{\XINT@cfrac@loop@a {#1}{#3}{#1}{{#2}}}%
5420 \def\XINT@cfrac@loop@a
5421 {%
5422   \expandafter\XINT@cfrac@loop@d\romannumeral0\XINT@div@prepare
5423 }%
5424 \def\XINT@cfrac@loop@d #1#2%
5425 {%
5426   \XINT@cfrac@loop@e #2.{#1}%
5427 }%
5428 \def\XINT@cfrac@loop@e #1%
5429 {%
5430   \xint@zero #1\xint@cfrac@loop@exit0\XINT@cfrac@loop@f #1%
5431 }%
5432 \def\XINT@cfrac@loop@f #1.#2#3#4%
5433 {%
5434   \XINT@cfrac@loop@a {#1}{#3}{#1}{{#2}#4}%
5435 }%
5436 \def\xint@cfrac@loop@exit0\XINT@cfrac@loop@f #1.#2#3#4#5#6%
5437   {\XINT@cfrac@T #5#6{#2}#4\Z }%
5438 \def\XINT@cfrac@T #1#2#3#4%
5439 {%
5440   \xint@z #4\XINT@cfrac@end\Z\XINT@cfrac@T #1#2{#4+\cfrac{#11#2}{#3}}%
5441 }%
5442 \def\XINT@cfrac@end\Z\XINT@cfrac@T #1#2#3%
5443 {%
5444   \XINT@cfrac@@end #3%
5445 }%
5446 \def\XINT@cfrac@@end \Z+\cfrac#1#2{ #2}%

```

### 23.6 \xintGCFrac

```

5447 \def\xintGCFrac {\romannumeral0\xintgcfrac }%
5448 \def\xintgcfrac #1%
5449 {%
5450   \XINT@gcfrac@opt@a #1\Z
5451 }%

```

```

5452 \def\XINT@gcfrac@opt@a #1%
5453 {%
5454   \ifx#1[\XINT@gcfrac@opt@b\fi \XINT@gcfrac@noopt #1%
5455 }%
5456 \def\XINT@gcfrac@noopt #1\Z
5457 {%
5458   \XINT@gcfrac #1+\W/\relax\relax
5459 }%
5460 \def\XINT@gcfrac@opt@b\fi\XINT@gcfrac@noopt [\Z #1]%
5461 {%
5462   \fi\csname XINT@gcfrac@opt#1\endcsname
5463 }%
5464 \def\XINT@gcfrac@optl #1%
5465 {%
5466   \XINT@gcfrac #1+\W/\relax\hfill
5467 }%
5468 \def\XINT@gcfrac@optc #1%
5469 {%
5470   \XINT@gcfrac #1+\W/\relax\relax
5471 }%
5472 \def\XINT@gcfrac@optr #1%
5473 {%
5474   \XINT@gcfrac #1+\W/\hfill\relax
5475 }%
5476 \def\XINT@gcfrac
5477 {%
5478   \expandafter\XINT@gcfrac@enter\romannumeral-'0%
5479 }%
5480 \def\XINT@gcfrac@enter {\XINT@gcfrac@loop {}}%
5481 \def\XINT@gcfrac@loop #1#2+#3/%
5482 {%
5483   \xint@w #3\XINT@gcfrac@endloop\W\XINT@gcfrac@loop {{#3}{#2}#1}%
5484 }%
5485 \def\XINT@gcfrac@endloop\W\XINT@gcfrac@loop #1#2#3%
5486 {%
5487   \XINT@gcfrac@T #2#3#1\Z\Z
5488 }%
5489 \def\XINT@gcfrac@T #1#2#3#4{\XINT@gcfrac@U #1#2{\xintFrac{#4}}}%
5490 \def\XINT@gcfrac@U #1#2#3#4#5%
5491 {%
5492   \xint@z #5\XINT@gcfrac@end\Z\XINT@gcfrac@U
5493     #1#2{\xintFrac{#5}}%
5494     \ifcase\xintSgn{#4}
5495       +\or+\else-\fi
5496     \cfrac{#1\xintFrac{\xintAbs{#4}}#2}{#3}}%
5497 }%
5498 \def\XINT@gcfrac@end\Z\XINT@gcfrac@U #1#2#3%
5499 {%
5500   \XINT@gcfrac@@end #3%

```

```
5501 }%
5502 \def\XINT@gcfrac@@end #1\cfrac#2#3{ #3}%
```

### 23.7 \xintGctoGCx

```
5503 \def\xintGctoGCx {\romannumeral0\xintgctogcx }%
5504 \def\xintgctogcx #1#2#3%
5505 {%
5506   \expandafter\XINT@gctgcx@start\expandafter {\romannumeral-‘0#3}{#1}{#2}%
5507 }%
5508 \def\XINT@gctgcx@start #1#2#3{\XINT@gctgcx@loop@a {}{#2}{#3}#1+\W/}%
5509 \def\XINT@gctgcx@loop@a #1#2#3#4+#5/%
5510 {%
5511   \xint@w #5\XINT@gctgcx@end\W
5512   \XINT@gctgcx@loop@b {#1{#4}}{#2{#5}#3}{#2}{#3}%
5513 }%
5514 \def\XINT@gctgcx@loop@b #1#2%
5515 {%
5516   \XINT@gctgcx@loop@a {#1#2}%
5517 }%
5518 \def\XINT@gctgcx@end\W\XINT@gctgcx@loop@b #1#2#3#4{ #1}%
```

### 23.8 \xintFtoCs

```
5519 \def\xintFtoCs {\romannumeral0\xintftocs }%
5520 \def\xintftocs #1%
5521 {%
5522   \expandafter\XINT@ftc@A\romannumeral0\xintraw {#1}\Z
5523 }%
5524 \def\XINT@ftc@A #1/#2\Z
5525 {%
5526   \expandafter\XINT@ftc@B\romannumeral0\xintidivision {#1}{#2}{#2}%
5527 }%
5528 \def\XINT@ftc@B #1#2%
5529 {%
5530   \XINT@ftc@C #2.{#1}%
5531 }%
5532 \def\XINT@ftc@C #1%
5533 {%
5534   \xint@zero #1\XINT@ftc@integer 0\XINT@ftc@D #1%
5535 }%
5536 \def\XINT@ftc@integer 0\XINT@ftc@D 0#1.#2#3{ #2}%
5537 \def\XINT@ftc@D #1.#2#3{\XINT@ftc@loop@a {#1}{#3}{#1}{#2,}}%
5538 \def\XINT@ftc@loop@a
5539 {%
5540   \expandafter\XINT@ftc@loop@d\romannumeral0\XINT@div@prepare
5541 }%
5542 \def\XINT@ftc@loop@d #1#2%
5543 {%
5544   \XINT@ftc@loop@e #2.{#1}%
5545 }%
```

```

5546 \def\XINT@ftc@loop@e #1%
5547 {%
5548   \xint@zero #1\xint@ftc@loop@exit0\XINT@ftc@loop@f #1%
5549 }%
5550 \def\XINT@ftc@loop@f #1.#2#3#4%
5551 {%
5552   \XINT@ftc@loop@a {#1}{#3}{#1}{#4#2,}%
5553 }%
5554 \def\xint@ftc@loop@exit0\XINT@ftc@loop@f #1.#2#3#4{ #4#2}%

```

### 23.9 \xintFtoCx

```

5555 \def\xintFtoCx {\romannumeral0\xintftocx }%
5556 \def\xintftocx #1#2%
5557 {%
5558   \expandafter\XINT@ftcx@A\romannumeral0\xintraw {#2}\Z {#1}%
5559 }%
5560 \def\XINT@ftcx@A #1/#2\Z
5561 {%
5562   \expandafter\XINT@ftcx@B\romannumeral0\xintidivision {#1}{#2}{#2}%
5563 }%
5564 \def\XINT@ftcx@B #1#2%
5565 {%
5566   \XINT@ftcx@C #2.{#1}%
5567 }%
5568 \def\XINT@ftcx@C #1%
5569 {%
5570   \xint@zero #1\XINT@ftcx@integer 0\XINT@ftcx@D #1%
5571 }%
5572 \def\XINT@ftcx@integer 0\XINT@ftcx@D 0#1.#2#3#4{ #2}%
5573 \def\XINT@ftcx@D #1.#2#3#4{\XINT@ftcx@loop@a {#1}{#3}{#1}{#2#4}{#4}}%
5574 \def\XINT@ftcx@loop@a
5575 {%
5576   \expandafter\XINT@ftcx@loop@d\romannumeral0\XINT@div@prepare
5577 }%
5578 \def\XINT@ftcx@loop@d #1#2%
5579 {%
5580   \XINT@ftcx@loop@e #2.{#1}%
5581 }%
5582 \def\XINT@ftcx@loop@e #1%
5583 {%
5584   \xint@zero #1\xint@ftcx@loop@exit0\XINT@ftcx@loop@f #1%
5585 }%
5586 \def\XINT@ftcx@loop@f #1.#2#3#4#5%
5587 {%
5588   \XINT@ftcx@loop@a {#1}{#3}{#1}{#4#2#5}{#5}%
5589 }%
5590 \def\xint@ftcx@loop@exit0\XINT@ftcx@loop@f #1.#2#3#4#5{ #4#2}%

```

### 23.10 \xintFtoGC

```

5591 \def\xintFtoGC {\romannumeral0\xintftogc }%
5592 \def\xintftogc {\xintftocx {+1/}}%

```

### 23.11 \xintFtoCC

```

5593 \def\xintFtoCC {\romannumeral0\xintftocc }%
5594 \def\xintftocc #1%
5595 {%
5596   \expandafter\xINT@ftcc@A\expandafter {\romannumeral0\xintraw {#1}}%
5597 }%
5598 \def\xINT@ftcc@A #1%
5599 {%
5600   \expandafter\xINT@ftcc@B
5601   \romannumeral0\xintraw {\xintAdd {1/2[0]}\{#1[0]}\}\Z {#1[0]}}%
5602 }%
5603 \def\xINT@ftcc@B #1/#2\Z
5604 {%
5605   \expandafter\xINT@ftcc@C\expandafter {\romannumeral0\xintiquo {#1}\{#2}}%
5606 }%
5607 \def\xINT@ftcc@C #1#2%
5608 {%
5609   \expandafter\xINT@ftcc@D\romannumeral0\xintsub {#2}\{#1}\Z {#1}%
5610 }%
5611 \def\xINT@ftcc@D #1%
5612 {%
5613   \xint@UDzerominusfork
5614   #1-\dummy \xINT@ftcc@integer
5615   0#1\dummy \xINT@ftcc@En
5616   0-\dummy {\xINT@ftcc@Ep #1}%
5617   \xint@UDkrof
5618 }%
5619 \def\xINT@ftcc@Ep #1\Z #2%
5620 {%
5621   \expandafter\xINT@ftcc@loop@a\expandafter
5622   {\romannumeral0\xintdiv {1[0]}\{#1}\}\{#2+1/}}%
5623 }%
5624 \def\xINT@ftcc@En #1\Z #2%
5625 {%
5626   \expandafter\xINT@ftcc@loop@a\expandafter
5627   {\romannumeral0\xintdiv {1[0]}\{#1}\}\{#2+-1/}}%
5628 }%
5629 \def\xINT@ftcc@integer #1\Z #2{ #2}%
5630 \def\xINT@ftcc@loop@a #1%
5631 {%
5632   \expandafter\xINT@ftcc@loop@b
5633   \romannumeral0\xintraw {\xintAdd {1/2[0]}\{#1}\}\Z {#1}}%
5634 }%
5635 \def\xINT@ftcc@loop@b #1/#2\Z
5636 {%
5637   \expandafter\xINT@ftcc@loop@c\expandafter

```

```

5638   {\romannumeral0\xintiquo {#1}{#2}}%
5639 }%
5640 \def\XINT@ftcc@loop@c #1#2%
5641 {%
5642   \expandafter\XINT@ftcc@loop@d
5643   \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
5644 }%
5645 \def\XINT@ftcc@loop@d #1%
5646 {%
5647   \xint@UDzerominusfork
5648   #1-\dummy \XINT@ftcc@end
5649   0#1\dummy \XINT@ftcc@loop@N
5650   0-\dummy {\XINT@ftcc@loop@P #1}%
5651   \xint@UDkrof
5652 }%
5653 \def\XINT@ftcc@end #1\Z #2#3{ #3#2}%
5654 \def\XINT@ftcc@loop@P #1\Z #2#3%
5655 {%
5656   \expandafter\XINT@ftcc@loop@a\expandafter
5657   {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+1/}%
5658 }%
5659 \def\XINT@ftcc@loop@N #1\Z #2#3%
5660 {%
5661   \expandafter\XINT@ftcc@loop@a\expandafter
5662   {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+-1/}%
5663 }%

```

### 23.12 \xintFtoCv

```

5664 \def\xintFtoCv {\romannumeral0\xintftocv }%
5665 \def\xintftocv #1%
5666 {%
5667   \xinticstocv {\xintFtoCs {#1}}%
5668 }%

```

### 23.13 \xintFtoCCv

```

5669 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
5670 \def\xintftoccv #1%
5671 {%
5672   \xintigctocv {\xintFtoCC {#1}}%
5673 }%

```

### 23.14 \xintCstoF

```

5674 \def\xintCstoF {\romannumeral0\xintcstof }%
5675 \def\xintcstof #1%
5676 {%
5677   \expandafter\XINT@cstf@prep \romannumeral-‘0#1,\W,%
5678 }%
5679 \def\XINT@cstf@prep

```



```

5680 {%
5681   \XINT@cstf@loop@a 1001%
5682 }%
5683 \def\XINT@cstf@loop@a #1#2#3#4#5,%
5684 {%
5685   \xint@w #5\XINT@cstf@end\W\expandafter\XINT@cstf@loop@b
5686   \romannumeral0\xintra {#5}#{1}{#2}{#3}{#4}%
5687 }%
5688 \def\XINT@cstf@loop@b #1/#2.#3#4#5#6%
5689 {%
5690   \expandafter\XINT@cstf@loop@c\expandafter
5691   {\romannumeral0\XINT@mul@fork #2\Z #4\Z }%
5692   {\romannumeral0\XINT@mul@fork #2\Z #3\Z }%
5693   {\romannumeral0\xintiadd {\XINT@Mul {#2}{#6}}{\XINT@Mul {#1}{#4}}}%
5694   {\romannumeral0\xintiadd {\XINT@Mul {#2}{#5}}{\XINT@Mul {#1}{#3}}}%
5695 }%
5696 \def\XINT@cstf@loop@c #1#2%
5697 {%
5698   \expandafter\XINT@cstf@loop@d\expandafter {\expandafter{#2}{#1}}%
5699 }%
5700 \def\XINT@cstf@loop@d #1#2%
5701 {%
5702   \expandafter\XINT@cstf@loop@e\expandafter {\expandafter{#2}{#1}}%
5703 }%
5704 \def\XINT@cstf@loop@e #1#2%
5705 {%
5706   \expandafter\XINT@cstf@loop@a\expandafter{#2}{#1}%
5707 }%
5708 \def\XINT@cstf@end #1.#2#3#4#5{\xintra {#2/#3}[0]}%

```

### 23.15 \xintiCstoF

```

5709 \def\xintiCstoF {\romannumeral0\xinticstof }%
5710 \def\xinticstof #1%
5711 {%
5712   \expandafter\XINT@icstf@prep \romannumeral-‘0#1,\W,%
5713 }%
5714 \def\XINT@icstf@prep
5715 {%
5716   \XINT@icstf@loop@a 1001%
5717 }%
5718 \def\XINT@icstf@loop@a #1#2#3#4#5,%
5719 {%
5720   \xint@w #5\XINT@icstf@end\W
5721   \expandafter
5722   \XINT@icstf@loop@b \romannumeral-‘0#5.#{1}{#2}{#3}{#4}%
5723 }%
5724 \def\XINT@icstf@loop@b #1.#2#3#4#5%
5725 {%
5726   \expandafter\XINT@icstf@loop@c\expandafter

```

```

5727 {\romannumeral0\xintiadd {#5}{\XINT@Mul {#1}{#3}}}%
5728 {\romannumeral0\xintiadd {#4}{\XINT@Mul {#1}{#2}}}%
5729 {#2}{#3}}%
5730 }%
5731 \def\XINT@icstf@loop@c #1#2%
5732 {%
5733 \expandafter\XINT@icstf@loop@a\expandafter {#2}{#1}%
5734 }%
5735 \def\XINT@icstf@end#1.#2#3#4#5{\xintraw {#2/#3}[0]}%

```

### 23.16 \xintGctoF

```

5736 \def\xintGctoF {\romannumeral0\xintgctof }%
5737 \def\xintgctof #1%
5738 {%
5739 \expandafter\XINT@gctf@prep \romannumeral-'0#1+\W/%
5740 }%
5741 \def\XINT@gctf@prep
5742 {%
5743 \XINT@gctf@loop@a 1001%
5744 }%
5745 \def\XINT@gctf@loop@a #1#2#3#4#5+%
5746 {%
5747 \expandafter\XINT@gctf@loop@b
5748 \romannumeral0\xintraw {#5}.{#1}{#2}{#3}{#4}%
5749 }%
5750 \def\XINT@gctf@loop@b #1/#2.#3#4#5#6%
5751 {%
5752 \expandafter\XINT@gctf@loop@c\expandafter
5753 {\romannumeral0\XINT@mul@fork #2\Z #4\Z }%
5754 {\romannumeral0\XINT@mul@fork #2\Z #3\Z }%
5755 {\romannumeral0\xintiadd {\XINT@Mul {#2}{#6}}{\XINT@Mul {#1}{#4}}}%
5756 {\romannumeral0\xintiadd {\XINT@Mul {#2}{#5}}{\XINT@Mul {#1}{#3}}}%
5757 }%
5758 \def\XINT@gctf@loop@c #1#2%
5759 {%
5760 \expandafter\XINT@gctf@loop@d\expandafter {\expandafter{#2}{#1}}%
5761 }%
5762 \def\XINT@gctf@loop@d #1#2%
5763 {%
5764 \expandafter\XINT@gctf@loop@e\expandafter {\expandafter{#2}{#1}}%
5765 }%
5766 \def\XINT@gctf@loop@e #1#2%
5767 {%
5768 \expandafter\XINT@gctf@loop@f\expandafter {\expandafter{#2}{#1}}%
5769 }%
5770 \def\XINT@gctf@loop@f #1#2/%
5771 {%
5772 \xint@w #2\XINT@gctf@end\W\expandafter\XINT@gctf@loop@g
5773 \romannumeral0\xintraw {#2}.#1%

```

```

5774 }%
5775 \def\XINT@gctf@loop@g #1/#2.#3#4#5#6%
5776 {%
5777   \expandafter\XINT@gctf@loop@h\expandafter
5778   {\romannumeral0\XINT@mul@fork #1\Z #6\Z }%
5779   {\romannumeral0\XINT@mul@fork #1\Z #5\Z }%
5780   {\romannumeral0\XINT@mul@fork #2\Z #4\Z }%
5781   {\romannumeral0\XINT@mul@fork #2\Z #3\Z }%
5782 }%
5783 \def\XINT@gctf@loop@h #1#2%
5784 {%
5785   \expandafter\XINT@gctf@loop@i\expandafter {\expandafter{#2}{#1}}%
5786 }%
5787 \def\XINT@gctf@loop@i #1#2%
5788 {%
5789   \expandafter\XINT@gctf@loop@j\expandafter {\expandafter{#2}#1}%
5790 }%
5791 \def\XINT@gctf@loop@j #1#2%
5792 {%
5793   \expandafter\XINT@gctf@loop@a\expandafter {#2}#1%
5794 }%
5795 \def\XINT@gctf@end #1.#2#3#4#5{\xintraw {#2/#3}[0]}%

```

### 23.17 \xintiGctoF

```

5796 \def\xintiGctoF {\romannumeral0\xintigctof }%
5797 \def\xintigctof #1%
5798 {%
5799   \expandafter\XINT@igctf@prep \romannumeral-‘0#1+\W/%
5800 }%
5801 \def\XINT@igctf@prep
5802 {%
5803   \XINT@igctf@loop@a 1001%
5804 }%
5805 \def\XINT@igctf@loop@a #1#2#3#4#5+%
5806 {%
5807   \expandafter\XINT@igctf@loop@b
5808   \romannumeral-‘0#5.{#1}{#2}{#3}{#4}%
5809 }%
5810 \def\XINT@igctf@loop@b #1.#2#3#4#5%
5811 {%
5812   \expandafter\XINT@igctf@loop@c\expandafter
5813   {\romannumeral0\xintiadd {#5}{\XINT@Mul {#1}{#3}}}%
5814   {\romannumeral0\xintiadd {#4}{\XINT@Mul {#1}{#2}}}%
5815   {#2}{#3}%
5816 }%
5817 \def\XINT@igctf@loop@c #1#2%
5818 {%
5819   \expandafter\XINT@igctf@loop@f\expandafter {\expandafter{#2}{#1}}%
5820 }%

```

```

5821 \def\XINT@igctf@loop@f #1#2#3#4/%
5822 {%
5823   \xint@w #4\XINT@igctf@end\W
5824   \expandafter\XINT@igctf@loop@g
5825   \romannumeral-‘0#4.{#2}{#3}#1%
5826 }%
5827 \def\XINT@igctf@loop@g #1.#2#3%
5828 {%
5829   \expandafter\XINT@igctf@loop@h\expandafter
5830   {\romannumeral0\XINT@mul@fork #1\Z #3\Z }%
5831   {\romannumeral0\XINT@mul@fork #1\Z #2\Z }%
5832 }%
5833 \def\XINT@igctf@loop@h #1#2%
5834 {%
5835   \expandafter\XINT@igctf@loop@i\expandafter {#2}{#1}%
5836 }%
5837 \def\XINT@igctf@loop@i #1#2#3#4%
5838 {%
5839   \XINT@igctf@loop@a {#3}{#4}{#1}{#2}%
5840 }%
5841 \def\XINT@igctf@end #1.#2#3#4#5{\xintraw {#4/#5}[0]}%

```

### 23.18 \xintCstoCv

```

5842 \def\xintCstoCv {\romannumeral0\xintcstocv }%
5843 \def\xintcstocv #1%
5844 {%
5845   \expandafter\XINT@cstcv@prep \romannumeral-‘0#1,\W,%
5846 }%
5847 \def\XINT@cstcv@prep
5848 {%
5849   \XINT@cstcv@loop@a {}1001%
5850 }%
5851 \def\XINT@cstcv@loop@a #1#2#3#4#5#6,%
5852 {%
5853   \xint@w #6\XINT@cstcv@end\W
5854   \expandafter\XINT@cstcv@loop@b
5855   \romannumeral0\xintraw {#6}.{#2}{#3}{#4}{#5}{#1}%
5856 }%
5857 \def\XINT@cstcv@loop@b #1/#2.#3#4#5#6%
5858 {%
5859   \expandafter\XINT@cstcv@loop@c\expandafter
5860   {\romannumeral0\XINT@mul@fork #2\Z #4\Z }%
5861   {\romannumeral0\XINT@mul@fork #2\Z #3\Z }%
5862   {\romannumeral0\xintiadd {\XINT@Mul {#2}{#6}}{\XINT@Mul {#1}{#4}}}%
5863   {\romannumeral0\xintiadd {\XINT@Mul {#2}{#5}}{\XINT@Mul {#1}{#3}}}%
5864 }%
5865 \def\XINT@cstcv@loop@c #1#2%
5866 {%
5867   \expandafter\XINT@cstcv@loop@d\expandafter {\expandafter{#2}{#1}}%

```

```

5868 }%
5869 \def\XINT@cstcv@loop@d #1#2%
5870 {%
5871   \expandafter\XINT@cstcv@loop@e\expandafter {\expandafter{#2}#1}%
5872 }%
5873 \def\XINT@cstcv@loop@e #1#2%
5874 {%
5875   \expandafter\XINT@cstcv@loop@f\expandafter{#2}#1%
5876 }%
5877 \def\XINT@cstcv@loop@f #1#2#3#4#5%
5878 {%
5879   \expandafter\XINT@cstcv@loop@g\expandafter
5880   {\romannumeral0\xintra {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
5881 }%
5882 \def\XINT@cstcv@loop@g #1#2{\XINT@cstcv@loop@a {#2{#1[0]}}}%
5883 \def\XINT@cstcv@end #1.#2#3#4#5#6{ #6}%

```

### 23.19 \xintiCstoCv

```

5884 \def\xintiCstoCv {\romannumeral0\xinticstocv }%
5885 \def\xinticstocv #1%
5886 {%
5887   \expandafter\XINT@icstcv@prep \romannumeral-'0#1,\W,%
5888 }%
5889 \def\XINT@icstcv@prep
5890 {%
5891   \XINT@icstcv@loop@a {}1001%
5892 }%
5893 \def\XINT@icstcv@loop@a #1#2#3#4#5#6,%
5894 {%
5895   \xint@w #6\XINT@icstcv@end\W
5896   \expandafter
5897   \XINT@icstcv@loop@b \romannumeral-'0#6.{#2}{#3}{#4}{#5}{#1}%
5898 }%
5899 \def\XINT@icstcv@loop@b #1.#2#3#4#5%
5900 {%
5901   \expandafter\XINT@icstcv@loop@c\expandafter
5902   {\romannumeral0\xintiadd {#5}{\XINT@Mul {#1}{#3}}}%
5903   {\romannumeral0\xintiadd {#4}{\XINT@Mul {#1}{#2}}}%
5904   {{#2}{#3}}}%
5905 }%
5906 \def\XINT@icstcv@loop@c #1#2%
5907 {%
5908   \expandafter\XINT@icstcv@loop@d\expandafter {#2}{#1}%
5909 }%
5910 \def\XINT@icstcv@loop@d #1#2%
5911 {%
5912   \expandafter\XINT@icstcv@loop@e\expandafter
5913   {\romannumeral0\xintra {#1/#2}}{#1}{#2}}%
5914 }%

```

```

5915 \def\XINT@icstcv@loop@a #1#2#3#4{\XINT@icstcv@loop@a {#4{#1[0]}}#2#3}%
5916 \def\XINT@icstcv@end #1.#2#3#4#5#6{ #6}%

```

### 23.20 \xintGctoCv

```

5917 \def\xintGctoCv {\romannumeral0\xintgctocv}%
5918 \def\xintgctocv #1%
5919 {%
5920   \expandafter\XINT@gctcv@prep \romannumeral- '0#1+\W/%
5921 }%
5922 \def\XINT@gctcv@prep
5923 {%
5924   \XINT@gctcv@loop@a {}1001%
5925 }%
5926 \def\XINT@gctcv@loop@a #1#2#3#4#5#6+%
5927 {%
5928   \expandafter\XINT@gctcv@loop@b
5929   \romannumeral0\xintraW {#6}#{2}#{3}#{4}#{5}#{1}%
5930 }%
5931 \def\XINT@gctcv@loop@b #1/#2.#3#4#5#6%
5932 {%
5933   \expandafter\XINT@gctcv@loop@c\expandafter
5934   {\romannumeral0\XINT@mul@fork #2\Z #4\Z}%
5935   {\romannumeral0\XINT@mul@fork #2\Z #3\Z}%
5936   {\romannumeral0\xintiadd {\XINT@Mul {#2}{#6}}{\XINT@Mul {#1}{#4}}}%
5937   {\romannumeral0\xintiadd {\XINT@Mul {#2}{#5}}{\XINT@Mul {#1}{#3}}}%
5938 }%
5939 \def\XINT@gctcv@loop@c #1#2%
5940 {%
5941   \expandafter\XINT@gctcv@loop@d\expandafter {\expandafter{#2}{#1}}%
5942 }%
5943 \def\XINT@gctcv@loop@d #1#2%
5944 {%
5945   \expandafter\XINT@gctcv@loop@e\expandafter {\expandafter{#2}{#1}}%
5946 }%
5947 \def\XINT@gctcv@loop@e #1#2%
5948 {%
5949   \expandafter\XINT@gctcv@loop@f\expandafter {#2}#1%
5950 }%
5951 \def\XINT@gctcv@loop@f #1#2%
5952 {%
5953   \expandafter\XINT@gctcv@loop@g\expandafter
5954   {\romannumeral0\xintraW {#1/#2}}{#1}{#2}%
5955 }%
5956 \def\XINT@gctcv@loop@g #1#2#3#4%
5957 {%
5958   \XINT@gctcv@loop@h {#4{#1[0]}}{#2#3}%
5959 }%
5960 \def\XINT@gctcv@loop@h #1#2#3/%
5961 {%

```

```

5962 \xint@w #3\XINT@gctcv@end\W\expandafter\XINT@gctcv@loop@i
5963 \romannumeral0\xintraw {#3}.#2{#1}%
5964 }%
5965 \def\XINT@gctcv@loop@i #1/#2.#3#4#5#6%
5966 {%
5967 \expandafter\XINT@gctcv@loop@j\expandafter
5968 {\romannumeral0\XINT@mul@fork #1\Z #6\Z }%
5969 {\romannumeral0\XINT@mul@fork #1\Z #5\Z }%
5970 {\romannumeral0\XINT@mul@fork #2\Z #4\Z }%
5971 {\romannumeral0\XINT@mul@fork #2\Z #3\Z }%
5972 }%
5973 \def\XINT@gctcv@loop@j #1#2%
5974 {%
5975 \expandafter\XINT@gctcv@loop@k\expandafter {\expandafter{#2}{#1}}%
5976 }%
5977 \def\XINT@gctcv@loop@k #1#2%
5978 {%
5979 \expandafter\XINT@gctcv@loop@l\expandafter {\expandafter{#2}#1}%
5980 }%
5981 \def\XINT@gctcv@loop@l #1#2%
5982 {%
5983 \expandafter\XINT@gctcv@loop@m\expandafter {\expandafter{#2}#1}%
5984 }%
5985 \def\XINT@gctcv@loop@m #1#2{\XINT@gctcv@loop@a {#2}#1}%
5986 \def\XINT@gctcv@end #1.#2#3#4#5#6{ #6}%

```

### 23.21 \xintiGtoCv

```

5987 \def\xintiGtoCv {\romannumeral0\xintigctocv }%
5988 \def\xintigctocv #1%
5989 {%
5990 \expandafter\XINT@igctcv@prep \romannumeral-‘0#1+\W/%
5991 }%
5992 \def\XINT@igctcv@prep
5993 {%
5994 \XINT@igctcv@loop@a {}1001%
5995 }%
5996 \def\XINT@igctcv@loop@a #1#2#3#4#5#6+%
5997 {%
5998 \expandafter\XINT@igctcv@loop@b
5999 \romannumeral-‘0#6.{#2}{#3}{#4}{#5}{#1}%
6000 }%
6001 \def\XINT@igctcv@loop@b #1.#2#3#4#5%
6002 {%
6003 \expandafter\XINT@igctcv@loop@c\expandafter
6004 {\romannumeral0\xintiadd {#5}{\XINT@Mul {#1}{#3}}}%
6005 {\romannumeral0\xintiadd {#4}{\XINT@Mul {#1}{#2}}}%
6006 {{#2}{#3}}}%
6007 }%
6008 \def\XINT@igctcv@loop@c #1#2%

```

```

6009 {%
6010   \expandafter\XINT@igctcv@loop@f\expandafter {\expandafter{#2}{#1}}%
6011 }%
6012 \def\XINT@igctcv@loop@f #1#2#3#4/%
6013 {%
6014   \xint@w #4\XINT@igctcv@end@a\W
6015   \expandafter\XINT@igctcv@loop@g
6016   \romannumeral- '0#4.#1#2{#3}%
6017 }%
6018 \def\XINT@igctcv@loop@g #1.#2#3#4#5%
6019 {%
6020   \expandafter\XINT@igctcv@loop@h\expandafter
6021   {\romannumeral0\XINT@mul@fork #1\Z #5\Z }%
6022   {\romannumeral0\XINT@mul@fork #1\Z #4\Z }%
6023   {{#2}{#3}}%
6024 }%
6025 \def\XINT@igctcv@loop@h #1#2%
6026 {%
6027   \expandafter\XINT@igctcv@loop@i\expandafter {\expandafter{#2}{#1}}%
6028 }%
6029 \def\XINT@igctcv@loop@i #1#2{\XINT@igctcv@loop@k #2{#2#1}}%
6030 \def\XINT@igctcv@loop@k #1#2%
6031 {%
6032   \expandafter\XINT@igctcv@loop@l\expandafter
6033   {\romannumeral0\xintraW {#1/#2}}%
6034 }%
6035 \def\XINT@igctcv@loop@l #1#2#3{\XINT@igctcv@loop@a {#3{#1[0]}}#2}%
6036 \def\XINT@igctcv@end@a #1.#2#3#4#5%
6037 {%
6038   \expandafter\XINT@igctcv@end@b\expandafter
6039   {\romannumeral0\xintraW {#2/#3}}%
6040 }%
6041 \def\XINT@igctcv@end@b #1#2{ #2{#1[0]}}%

```

## 23.22 \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.

```

6042 \def\xintCntoF {\romannumeral0\xintcntof }%
6043 \def\xintcntof #1%
6044 {%
6045   \expandafter\XINT@cntf\expandafter {\the\numexpr #1}%
6046 }%
6047 \def\XINT@cntf #1#2%
6048 {%
6049   \ifnum #1>0
6050     \xint@afterfi {\expandafter\XINT@cntf@loop\expandafter
6051                   {\the\numexpr #1-1\expandafter}\expandafter
6052                   {\romannumeral- '0#2{#1}}{#2}}%

```



```

6053 \else
6054 \xint@afterfi
6055 {\ifnum #1=0
6056 \xint@afterfi {\expandafter\space \romannumeral-‘0#2{0}}}%
6057 \else \xint@afterfi { 0[0]}}%
6058 \fi}%
6059 \fi
6060 }%
6061 \def\XINT@cntf@loop #1#2#3%
6062 {%
6063 \ifnum #1>0 \else \XINT@cntf@exit \fi
6064 \expandafter\XINT@cntf@loop\expandafter
6065 {\the\numexpr #1-1\expandafter }\expandafter
6066 {\romannumeral0\xintadd {\xintDiv {1[0]}}{#2}}{#3{#1}}}%
6067 {#3}}%
6068 }%
6069 \def\XINT@cntf@exit \fi
6070 \expandafter\XINT@cntf@loop\expandafter
6071 #1\expandafter #2#3%
6072 {%
6073 \fi\xint@gobble@ii #2%
6074 }%

```

### 23.23 \xintGCntoF

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.

```

6075 \def\xintGCntoF {\romannumeral0\xintgcntof }%
6076 \def\xintgcntof #1%
6077 {%
6078 \expandafter\XINT@gcntf\expandafter {\the\numexpr #1}%
6079 }%
6080 \def\XINT@gcntf #1#2#3%
6081 {%
6082 \ifnum #1>0
6083 \xint@afterfi {\expandafter\XINT@gcntf@loop\expandafter
6084 {\the\numexpr #1-1\expandafter}\expandafter
6085 {\romannumeral-‘0#2{#1}}{#2}{#3}}%
6086 \else
6087 \xint@afterfi
6088 {\ifnum #1=0
6089 \xint@afterfi {\expandafter\space\romannumeral-‘0#2{0}}}%
6090 \else \xint@afterfi { 0[0]}}%
6091 \fi}%
6092 \fi
6093 }%
6094 \def\XINT@gcntf@loop #1#2#3#4%
6095 {%

```

```

6096 \ifnum #1>0 \else \XINT@gcntf@exit \fi
6097 \expandafter\XINT@gcntf@loop\expandafter
6098 {\the\numexpr #1-1\expandafter }\expandafter
6099 {\romannumeral0\xintadd {\xintDiv {#4{#1}}{#2}}{#3{#1}}}%
6100 {#3}{#4}%
6101 }%
6102 \def\XINT@gcntf@exit \fi
6103 \expandafter\XINT@gcntf@loop\expandafter
6104 #1\expandafter #2#3#4%
6105 {%
6106 \fi\xint@gobble@ii #2%
6107 }%

```

### 23.24 \xintCntoCs

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.

```

6108 \def\xintCntoCs {\romannumeral0\xintcntocs }%
6109 \def\xintcntocs #1%
6110 {%
6111 \expandafter\XINT@cntcs\expandafter {\the\numexpr #1}%
6112 }%
6113 \def\XINT@cntcs #1#2%
6114 {%
6115 \ifnum #1<0
6116 \xint@afterfi { 0[0]}%
6117 \else
6118 \xint@afterfi {\expandafter\XINT@cntcs@loop\expandafter
6119 {\the\numexpr #1-1\expandafter}\expandafter
6120 {\expandafter{\romannumeral-‘0#2{#1}}{#2}}}%
6121 \fi
6122 }%
6123 \def\XINT@cntcs@loop #1#2#3%
6124 {%
6125 \ifnum #1>-1 \else \XINT@cntcs@exit \fi
6126 \expandafter\XINT@cntcs@loop\expandafter
6127 {\the\numexpr #1-1\expandafter }\expandafter
6128 {\expandafter{\romannumeral-‘0#3{#1}},#2}{#3}%
6129 }%
6130 \def\XINT@cntcs@exit \fi
6131 \expandafter\XINT@cntcs@loop\expandafter
6132 #1\expandafter #2#3%
6133 {%
6134 \fi\XINT@cntcs@@exit #2%
6135 }%
6136 \def\XINT@cntcs@@exit #1,{ }%

```

### 23.25 \xintCntoGC

## 23 Package *xintcfrac* implementation

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.

```

6137 \def\xintCntoGC {\romannumeral0\xintcntogc }%
6138 \def\xintcntogc #1%
6139 {%
6140   \expandafter\XINT@cntgc\expandafter {\the\numexpr #1}%
6141 }%
6142 \def\XINT@cntgc #1#2%
6143 {%
6144   \ifnum #1<0
6145     \xint@afterfi { 0[0]}%
6146   \else
6147     \xint@afterfi {\expandafter\XINT@cntgc@loop\expandafter
6148                   {\the\numexpr #1-1\expandafter}\expandafter
6149                   {\expandafter{\romannumeral-‘0#2{#1}}{#2}}}%
6150   \fi
6151 }%
6152 \def\XINT@cntgc@loop #1#2#3%
6153 {%
6154   \ifnum #1>-1 \else \XINT@cntgc@exit \fi
6155   \expandafter\XINT@cntgc@loop\expandafter
6156   {\the\numexpr #1-1\expandafter }\expandafter
6157   {\expandafter{\romannumeral-‘0#3{#1}}+1/#2}{#3}%
6158 }%
6159 \def\XINT@cntgc@exit \fi
6160   \expandafter\XINT@cntgc@loop\expandafter
6161   #1\expandafter #2#3%
6162 {%
6163   \fi\XINT@cntgc@@exit #2%
6164 }%
6165 \def\XINT@cntgc@@exit #1+1/{ }%

```

### 23.26 `\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.

```

6166 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
6167 \def\xintgcntogc #1%
6168 {%
6169   \expandafter\XINT@gcntgc\expandafter {\the\numexpr #1}%
6170 }%
6171 \def\XINT@gcntgc #1#2#3%
6172 {%
6173   \ifnum #1<0
6174     \xint@afterfi { {0[0]}}%
6175   \else
6176     \xint@afterfi {\expandafter\XINT@gcntgc@loop\expandafter
6177                   {\the\numexpr #1-1\expandafter}\expandafter

```

## 23 Package *xintcfrac* implementation

```

6178          {\expandafter{\romannumeral-'\0#2{#1}}}{#2}{#3}}%
6179    \fi
6180 }%
6181 \def\XINT@gcntgc@loop #1#2#3#4%
6182 {%
6183   \ifnum #1>-1 \else \XINT@gcntgc@exit \fi
6184   \expandafter\XINT@gcntgc@loop@b\expandafter
6185   {\expandafter{\romannumeral-'\0#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
6186 }%
6187 \def\XINT@gcntgc@loop@b #1#2#3%
6188 {%
6189   \expandafter\XINT@gcntgc@loop\expandafter
6190   {\the\numexpr #3-1\expandafter}\expandafter
6191   {\expandafter{\romannumeral-'\0#2}+#1}%
6192 }%
6193 \def\XINT@gcntgc@exit \fi
6194   \expandafter\XINT@gcntgc@loop@b\expandafter #1#2#3#4#5%
6195 }%
6196   \fi\XINT@gcntgc@@exit #1%
6197 }%
6198 \def\XINT@gcntgc@@exit #1/{ %}

```

### 23.27 \xintCstoGC

```

6199 \def\xintCstoGC {\romannumeral0\xintcstogc }%
6200 \def\xintcstogc #1%
6201 {%
6202   \expandafter\XINT@cstc@prep \romannumeral-'\0#1,\W,%
6203 }%
6204 \def\XINT@cstc@prep #1,{\XINT@cstc@loop@a {{#1}}}%
6205 \def\XINT@cstc@loop@a #1#2,%
6206 {%
6207   \xint@w #2\XINT@cstc@end\W\XINT@cstc@loop@b {#1}{#2}%
6208 }%
6209 \def\XINT@cstc@loop@b #1#2{\XINT@cstc@loop@a {#1+1/{#2}}}%
6210 \def\XINT@cstc@end\W\XINT@cstc@loop@b #1#2{ #1}%

```

### 23.28 \xintGctoGC

```

6211 \def\xintGctoGC {\romannumeral0\xintgctogc }%
6212 \def\xintgctogc #1%
6213 {%
6214   \expandafter\XINT@gctgc@start \romannumeral-'\0#1+\W/%
6215 }%
6216 \def\XINT@gctgc@start {\XINT@gctgc@loop@a {}}%
6217 \def\XINT@gctgc@loop@a #1#2+#3/%
6218 {%
6219   \xint@w #3\XINT@gctgc@end\W\expandafter\XINT@gctgc@loop@b\expandafter
6220   {\romannumeral-'\0#2}{#3}{#1}%
6221 }%

```

## 23 Package *xintcfrac* implementation

```

6222 \def\XINT@gctgc@loop@b #1#2%
6223 {%
6224   \expandafter\XINT@gctgc@loop@c\expandafter
6225   {\romannumeral-‘0#2}{#1}%
6226 }%
6227 \def\XINT@gctgc@loop@c #1#2#3%
6228 {%
6229   \XINT@gctgc@loop@a {#3{#2}+{#1}}/%
6230 }%
6231 \def\XINT@gctgc@end\W\expandafter\XINT@gctgc@loop@b
6232 {%
6233   \expandafter\XINT@gctgc@@end
6234 }%
6235 \def\XINT@gctgc@@end #1#2#3{ #3{#1}}%
6236 \XINT@cfrac@restorecatcodes@endinput%

```