

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.3c (2018/06/17); documentation date: 2018/06/17.

From source file xint.dtx. Time-stamp: <17-06-2018 at 20:00:43 CEST>.

Contents

1	Introduction to the implementation (recent changes)	2
2	Package <code>xintkernel</code> implementation	5
3	Package <code>xinttools</code> implementation	19
4	Package <code>xintcore</code> implementation	55
5	Package <code>xint</code> implementation	113
6	Package <code>xintbinhex</code> implementation	153
7	Package <code>xintgcd</code> implementation	165
8	Package <code>xintfrac</code> implementation	178
9	Package <code>xintseries</code> implementation	259
10	Package <code>xintcfrc</code> implementation	268
11	Package <code>xintexpr</code> implementation	291
12	Per package indices of control sequences	376
13	Cumulative index	423
14	Cumulative line count	458

1 Introduction to the implementation (recent changes)

This is 1.3c of 2018/06/17.

- Release 1.3c of 2018/06/17:
 - `\xintglobaldefstrue` to give global scope to definitions done by `\xintdefvar`, `\xintdeffunc`, `\xintNewExpr`, et. al.
 - `qraw()` to allow bypassing the normal mode of functioning and reduce memory footprint from parsing (very many) comma separated numerical items.
 - the colon in the `:=` part of the syntax for `\xintdefvar` and variants is now optional; and if present it may be an active character or have any (reasonable) catcode.
 - `\xintdefvar`, `\xintdeffunc` and their variants try to set the catcode of the semi-colon which delimits their arguments; of course this will not work if that catcode is already frozen.
 - `\xintexprSafeCatcodes` sets a toggle to allow some nested usage.
 - `\xintUniformDeviate` is better documented and [sourcexint.pdf](#) is better hyperlinked and includes indices for the macros defined by each package.
 - since 1.3 release, `xintfrac` (hence `xintexpr`) loaded `xintgcd` in contradiction to what documentation says. Removed for now.
- Release 1.3b of 2018/05/18:
 - `\xintUniformDeviate`, `random()`, `grand()`, `randrange()`,
 - and their support macros `\XINTinRandomFloatSdigits`, `\XINTinRandomFloatSixteen`, `\xintRandomDigits`, `\xintiiRandRange`, `\xintiiRandRangeAtoB`, also `\xintXRandomDigits`,
 - functions defined via `\xintdeffunc` (et al.) to be without argument can be used as `foo():foo(nil)` syntax not anymore required.
- Release 1.3a of 2018/03/07:
 - removes from `xintcore`, `xint` and `xintfrac` the whole deprecation mechanism, as there are no more currently any deprecated macro,
 - adds `ifone()` and `ifint()` conditionals to the expression parsers,
 - has a completely redone `\XINT_factortens`, in the style of 1.2 release (but about 100 digits at least are needed for noticeable speed gain),
 - and last but not least fixes via addition of `\xintExpandArgs` the meaning of user defined functions, which in case of recursivity (as made possible at 1.3) were badly inefficient for lack of expansion of their arguments.
- Release 1.3 of 2018/03/01:
 - removed all macros previously deprecated at 1.2o,
 - modified addition, subtraction and modulo operations to use a least common multiple for the denominator of the result,
 - added `\xintPIrr`, `\xintDecToString` and `preduce()`,
 - and last but not least refactored extensively the `\xintNewExpr`/`\xintdeffunc` mechanism. It got both leaner and stronger and makes possible recursive function definitions.
- Release 1.2q of 2018/02/06 was a bugfix release with these changes:
 - fix to an 1.2l `xintcore` subtraction bug causing a breakage under some rare circumstances `:2-(`. It was caused by a refactoring left-over (extra `!` in `XINT_sub_1_Ida` replacement text), and should have been detected by the test suite, but manual testing from early days was not yet entirely included in our automated procedure.
 - new feature: tacit multiplication in front of digits, for example `10!20!30!` or `(10+10)10`.
- Release 1.2p of 2017/12/05 had some breaking changes:
 - new output for the `\xintBezout` macro (`xintgcd`),

- the [xintexpr](#) operators `/:` (aka 'mod') and `//`, and the supporting macros from [xintcore](#) and [xintfrac](#), are now associated with the *floored* division. Formerly it was the *truncated* division. This is breaking change for operands of opposite signs.

Improvements and new features:

- [xinttools](#) macro `\xintListWithSep` is faster (first update since 1.04-2013/04/25...).
- `divmod()` function added to the [xintexpr](#) parsers,
- `\xintdefvar`, `\xintdeffloatvar`, `\xintdefiivar` extended to allow multiple simultaneous assignments.
- Release 1.2o of 2017/08/29 deprecated those macros from [xintcore](#) and [xint](#) which filtered their arguments via `\xintNum`. Currently these macros execute as formerly but raise an error message. This deprecation is overruled for most if [xintfrac](#) is loaded as it provides their proper definitions. Some however (like `\xintiAdd`) remain deprecated even if loading [xintfrac](#). All these deprecated macros are destined to be removed at some future release.
- A few macros got renamed (e.g. `\xintNot` became `\xintNOT`.) Former names emit a deprecation error and will get removed at some future release.
- Release 1.2n of 2017/08/06 removed the [xintbinhex](#) dependencies upon [xintcore](#); the package now depends upon, and loads, only [xintkernel](#). The allowed maximal size for the inputs of the base conversion macros got increased. The speed got slightly improved.
- Release 1.2m of 2017/07/31 rewrote entirely the [xintbinhex](#) module in the style of the techniques from 1.2 [xintcore](#). The new macros expand faster but their inputs are now limited to a few thousand characters (the earlier routines, which dated back to 1.08 could handle (slowly) tens of thousands of digits.)
- Release 1.2l of 2017/07/26 refactored the subtraction and also `\xintiiCmp` got a rewrite. It should certainly use `\pdfstrcmp` for dramatic speed-up but I do not want to have to worry about multi-engine usage.

Some utility routines in [xintcore](#) manipulating blocks of eight digits and still in $O(N^2)$ style got rewritten analogously to the 1.2i version of macros such as `\xintInc`. Also `\xintiNum` was revisited.

The arithmetic macros of [xintcore](#) and all macros of [xintfrac](#) using `\XINT_infrac` were made compatible with arguments using non-delimited `\the\numexpr` or `\the\mathcode` etc... But `\xintiiAbs` and `\xintiiOpp` were not modified (to avoid some overhead) as well as routines such as `\xintInc` which are primarily for internal usage.

- Release 1.2i of 2016/12/13 rewrote some legacy macros like `\xintDSR` or `\xintDecSplit` in the style of the techniques of 1.2. But this means also that they got limited to about 22480 digits for the former and 19970 digits for the latter (this is with the input stack size at 5000 and the maximal expansion depth at 10000.) This is not really an issue from the point of view of calling macros (such as `\xintTrunc`, `\xintRound`), because they usually had since 1.2 their own limitation at about 19950 digits from other code parts (such as division.) The macro `\xintXTrunc` (which is not f-expandable however) can produce tens of thousands of digits and it escapes these limitations. Old macros such as `\xintLength` are not limited either (incidentally it got a lifting in 1.2i.) The macros from [xinttools](#) (`\xintKeep`, `\xintTrim`, `\xintNthElt`) also are not limited (but slower.)
- Release 1.2 of 2015/10/10 entirely rewrote the core arithmetic routines located in [xintcore](#). The parser of [xintexpr](#) got faster and the limitation at 5000 digits per number was removed.
- Extensive changes in release 1.1 of 2014/10/28 were located in [xintexpr](#). Also with that release, packages [xintkernel](#) and [xintcore](#) were extracted from [xinttools](#) and [xint](#), and `\xintAdd` was modified to not multiply denominators blindly.

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

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

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

2 Package [xintkernel](#) implementation

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

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

1.1. separated package.

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

1.3b. `\xintUniformDeviate`.

2.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

TOC, [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```

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

2.1.1 \XINT_setcatcodes, \XINT_storecatcodes, \XINT_restorecatcodes_endinput

```

32 \def\PrepareCatcodes
33 {%
34 \endgroup
35 \def\XINT_restorecatcodes
36 {% takes care of all, to allow more economical code in modules
37 \catcode0=\the\catcode0 %
38 \catcode59=\the\catcode59 % ; xintexpr
39 \catcode126=\the\catcode126 % ~ xintexpr
40 \catcode39=\the\catcode39 % ' xintexpr
41 \catcode34=\the\catcode34 % " xintbinhex, and xintexpr
42 \catcode63=\the\catcode63 % ? xintexpr
43 \catcode124=\the\catcode124 % | xintexpr
44 \catcode38=\the\catcode38 % & xintexpr
45 \catcode64=\the\catcode64 % @ xintexpr
46 \catcode33=\the\catcode33 % ! xintexpr
47 \catcode93=\the\catcode93 % ] -, xintfrac, xintseries, xintcfrac
48 \catcode91=\the\catcode91 % [ -, xintfrac, xintseries, xintcfrac
49 \catcode36=\the\catcode36 % $ xintgcd only
50 \catcode94=\the\catcode94 % ^
51 \catcode96=\the\catcode96 % `
52 \catcode47=\the\catcode47 % /
53 \catcode41=\the\catcode41 % )
54 \catcode40=\the\catcode40 % (
55 \catcode42=\the\catcode42 % *
56 \catcode43=\the\catcode43 % +
57 \catcode62=\the\catcode62 % >
58 \catcode60=\the\catcode60 % <
59 \catcode58=\the\catcode58 % :
60 \catcode46=\the\catcode46 % .
61 \catcode45=\the\catcode45 % -
62 \catcode44=\the\catcode44 % ,
63 \catcode35=\the\catcode35 % #
64 \catcode95=\the\catcode95 % _
65 \catcode125=\the\catcode125 % }
66 \catcode123=\the\catcode123 % {
67 \endlinechar=\the\endlinechar
68 \catcode13=\the\catcode13 % ^^M
69 \catcode32=\the\catcode32 %
70 \catcode61=\the\catcode61\relax % =
71 }%
72 \edef\XINT_restorecatcodes_endinput
73 {%
74 \XINT_restorecatcodes\noexpand\endinput %
```

```

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

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

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

2.2 Package identification

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

1.09c uses e-TeX `\ifdefined`.

```
123 \ifdefined\ProvidesPackage
124   \let\XINT_providespackage\relax
125 \else
126   \def\XINT_providespackage #1#2[#3]%
127     {\immediate\write-1{Package: #2 #3}%
128      \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
129 \fi
130 \XINT_providespackage
131 \ProvidesPackage {xintkernel}%
132 [2018/06/17 1.3c Paraphernalia for the xint packages (JFB)]%
```

2.3 Constants

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

2.4 (WIP) `\xint_texuniformdeviate` and needed counts

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


2.5 Token management utilities

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

```

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

2.6 “gob til” macros and UD style fork

```

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

```

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

```

2.7 \xint_afterfi

```

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

```

2.8 \xint_bye, \xint_Bye

1.09. \xint_bye

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

```

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

```

2.9 \xintdothis, \xintorthat

1.1.

1.2. names without underscores.

To be used this way:

```

\xif..\xint_dothis{..}\fi
\xif..\xint_dothis{..}\fi
\xif..\xint_dothis{..}\fi
...more such...
\xint_orthat{...}

```

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

```

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

```

2.10 \xint_zapspaces

1.1.

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

Usage: `\xint_zapspaces foo<space>\xint_gobble_i`

Will remove some brace pairs (but not spaces inside them). By the way the `\zap@spaces` of LaTeX2e handles unexpectedly things such as

`\zap@spaces 1 {22} 3 4 \@empty`

(spaces are not all removed). This does not happen with `\xint_zapspaces`.

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

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

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

expands to 12, not to 12\relax.

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

1.2i. made `\long`.

```
241 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
242 \long\def\xint_zapspaces_o #1{\expandafter\xint_zapspaces#1 \xint_gobble_i}%
```

2.11 \odef, \oodef, \fdef

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

```
243 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
244 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
245     \expandafter\expandafter\expandafter#1%
246     \expandafter\expandafter\expandafter }%
247 \def\xintfdef #1#2%
248     {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&#2}}%
249 \ifdefined\odef\else\let\odef\xintodef\fi
250 \ifdefined\oodef\else\let\oodef\xintoodef\fi
251 \ifdefined\fdef\else\let\fdef\xintfdef\fi
```

2.12 \xintReverseOrder

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

Attention: removes brace pairs (and swallows spaces).

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

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

```
252 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
253 \long\def\xintreverseorder #1%
254 {%
255     \XINT_rord_main {}#1%
256     \xint:
257     \xint_bye\xint_bye\xint_bye\xint_bye
258     \xint_bye\xint_bye\xint_bye\xint_bye
259     \xint:
```

```

260 }%
261 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
262 {%
263     \xint_bye #9\XINT_rord_cleanup\xint_bye
264     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
265 }%
266 \def\XINT_rord_cleanup #1{%
267 \long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
268 {%
269     \expandafter#1\xint_gob_til_xint: ##1%
270 }}\XINT_rord_cleanup { }%

```

2.13 \xintLength

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

1.2g. added `\xintCSVLength` to [xinttools](#).

1.2i. rewrote this venerable macro. New code about 40% faster across all lengths.

```

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

```

2.14 \xintLastItem

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

```

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

```

```

300 {%
301     \xint_gob_til_xint: #9%
302     {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
303     \XINT_last_loop {#9}.%
304 }%
305 \long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
306 \long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
307 \long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
308 \long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
309 \long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
310 \long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
311 \long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
312 \long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%

```

2.15 \xintLengthUpTo

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

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

1.2j. changed ending and interface to core loop.

```

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

```

345 }%

2.16 \xintreplicate

1.2i.

This is cloned from LaTeX3's `\prg_replicate:nn`, see Joseph's post at

<http://tex.stackexchange.com/questions/16189/repeat-command-n-times>

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

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

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

```
\romannumeral\XINT rep N\endcsname {foo}
```

to skip the `\numexpr`.

```

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

```

```

383     {\xint_c_ #1#1#1}%
384 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
385     {\xint_c_ #1#1#1#1}%
386 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
387     {\xint_c_ #1#1#1#1#1}%
388 \long\expandafter\def\csname XINT_rep_f6\endcsname #1%
389     {\xint_c_ #1#1#1#1#1#1}%
390 \long\expandafter\def\csname XINT_rep_f7\endcsname #1%
391     {\xint_c_ #1#1#1#1#1#1#1}%
392 \long\expandafter\def\csname XINT_rep_f8\endcsname #1%
393     {\xint_c_ #1#1#1#1#1#1#1#1}%
394 \long\expandafter\def\csname XINT_rep_f9\endcsname #1%
395     {\xint_c_ #1#1#1#1#1#1#1#1#1}%

```

2.17 \xintgobble

1.2i.

I hesitated about allowing as many as $9^6-1=531440$ tokens to gobble, but $9^5-1=59058$ is too low for playing with long decimal expansions.

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

```

396 \def\xintgobble #1%
397     {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
398 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
399 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
400 \def\XINT_gobble_b #1.#2%
401     {\expandafter\XINT_gobble_c
402         \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
403         \the\numexpr #2+\xint_c_i.#1.}%
404 \def\XINT_gobble_c #1.#2.#3.%
405     {\csname XINT_g2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
406 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
407 \expandafter\let\csname XINT_g10\endcsname\endcsname
408 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
409 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
410 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
411 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
412 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
413 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
414 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
415 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
416 \expandafter\let\csname XINT_g20\endcsname\endcsname
417 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
418     {\endcsname}%
419 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
420     {\expandafter\noexpand\csname XINT_g21\endcsname}%
421 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
422     {\expandafter\noexpand\csname XINT_g22\endcsname}%
423 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
424     {\expandafter\noexpand\csname XINT_g23\endcsname}%
425 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
426     {\expandafter\noexpand\csname XINT_g24\endcsname}%
427 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%

```



```

428 {\expandafter\noexpand\csname XINT_g25\endcsname}%
429 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
430 {\expandafter\noexpand\csname XINT_g26\endcsname}%
431 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
432 {\expandafter\noexpand\csname XINT_g27\endcsname}%
433 \expandafter\let\csname XINT_g30\endcsname\endcsname
434 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
435 {\expandafter\noexpand\csname XINT_g28\endcsname}%
436 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
437 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
438 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
439 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
440 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
441 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
442 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
443 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
444 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
445 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
446 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
447 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
448 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
449 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
450 \expandafter\let\csname XINT_g40\endcsname\endcsname
451 \expandafter\edef\csname XINT_g41\endcsname
452 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
453 \expandafter\edef\csname XINT_g42\endcsname
454 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
455 \expandafter\edef\csname XINT_g43\endcsname
456 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
457 \expandafter\edef\csname XINT_g44\endcsname
458 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
459 \expandafter\edef\csname XINT_g45\endcsname
460 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
461 \expandafter\edef\csname XINT_g46\endcsname
462 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
463 \expandafter\edef\csname XINT_g47\endcsname
464 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
465 \expandafter\edef\csname XINT_g48\endcsname
466 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
467 \expandafter\let\csname XINT_g50\endcsname\endcsname
468 \expandafter\edef\csname XINT_g51\endcsname
469 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
470 \expandafter\edef\csname XINT_g52\endcsname
471 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
472 \expandafter\edef\csname XINT_g53\endcsname
473 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
474 \expandafter\edef\csname XINT_g54\endcsname
475 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
476 \expandafter\edef\csname XINT_g55\endcsname
477 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
478 \expandafter\edef\csname XINT_g56\endcsname
479 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%

```



```

480 \expandafter\edef\csname XINT_g57\endcsname
481 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
482 \expandafter\edef\csname XINT_g58\endcsname
483 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
484 \expandafter\let\csname XINT_g60\endcsname\endcsname
485 \expandafter\edef\csname XINT_g61\endcsname
486 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
487 \expandafter\edef\csname XINT_g62\endcsname
488 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
489 \expandafter\edef\csname XINT_g63\endcsname
490 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
491 \expandafter\edef\csname XINT_g64\endcsname
492 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
493 \expandafter\edef\csname XINT_g65\endcsname
494 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
495 \expandafter\edef\csname XINT_g66\endcsname
496 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
497 \expandafter\edef\csname XINT_g67\endcsname
498 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
499 \expandafter\edef\csname XINT_g68\endcsname
500 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%

```

2.18 (WIP) \xintUniformDeviate

1.3b. See user manual for related information.

```

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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```
528 }%
529 \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
530 }%
531 {%
532 \def\xintUniformDeviate#1%
533 {%
534     \the\numexpr
535     \XINT_expandableerror{No uniformdeviate at engine level, returning 0.}%
536     0\relax
537 }%
538 }%
```

2.19 \xintMessage, \ifxintverbose

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

1.2e. uses [\write128](#) rather than [\write16](#) for compatibility with future extended range of output streams, in LuaTeX in particular.

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

2.20 \ifxintglobaldefs, \XINT_global

1.3c.

```
544 \newif\ifxintglobaldefs
545 \def\XINT_global{\ifxintglobaldefs\global\fi}%
```

2.21 (WIP) Expandable error message

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

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

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

3 Package [xinttools](#) implementation

.1	Catcodes, ε -TeX and reload detection . .	19			
.2	Package identification	20	.21	<code>\XINT_xflet</code>	36
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgdef</code> .	20	.22	<code>\xintApplyInline</code>	37
.4	<code>\xintRevWithBraces</code>	20	.23	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xintBreakForAndDo</code>	37
.5	<code>\xintZapFirstSpaces</code>	21	.24	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code> <code>mensions</code> , <code>\xinrationals</code>	40
.6	<code>\xintZapLastSpaces</code>	22	.25	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code> <code>four</code>	42
.7	<code>\xintZapSpaces</code>	23	.26	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code> <code>DigitsOf</code>	43
.8	<code>\xintZapSpacesB</code>	23	.27	<code>\xintExpandArgs</code>	46
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code> <code>Stripped</code>	23	.28	CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse . .	46
.10	<code>\xintListWithSep</code>	25	.28.1	<code>\xintLength:f:csv</code>	47
.11	<code>\xintNthElt</code>	26	.28.2	<code>\xintLengthUpTo:f:csv</code>	47
.12	<code>\xintKeep</code>	27	.28.3	<code>\xintKeep:f:csv</code>	48
.13	<code>\xintKeepUnbraced</code>	28	.28.4	<code>\xintTrim:f:csv</code>	50
.14	<code>\xintTrim</code>	29	.28.5	<code>\xintNthEltPy:f:csv</code>	52
.15	<code>\xintTrimUnbraced</code>	31	.28.6	<code>\xintReverse:f:csv</code>	53
.16	<code>\xintApply</code>	32	.28.7	<code>\xintFirstItem:f:csv</code>	53
.17	<code>\xintApplyUnbraced</code>	32	.28.8	<code>\xintLastItem:f:csv</code>	53
.18	<code>\xintSeq</code>	33	.28.9	Public names for the undocumented csv macros: <code>\xintCSVLength</code> , <code>\xintCSVKeep</code> , <code>\xintCSVTrim</code> , <code>\xintCSVNthEltPy</code> , <code>\xintCSVReverse</code> , <code>\xintCSV-</code> <code>FirstItem</code> , <code>\xintCSVLastItem</code>	54
.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> . . .	35			
.20	<code>\xintiloop</code> , <code>\xintiloopindex</code> , <code>\xint-</code> <code>bracediloopindex</code> , <code>\xintouteriloopindex</code> , <code>\xintbracedouteriloopindex</code> , <code>\xintbreak-</code> <code>iloop</code> , <code>\xintbreakiloopanddo</code> , <code>\xintiloop-</code>				

Release 1.09g of 2013/11/22 splits off `xinttools.sty` from `xint.sty`. Starting with 1.1, [xinttools](#) ceases being loaded automatically by `xint`.

3.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

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

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2018/06/17 1.3c Expandable and non-expandable utilities (JFB)]%

```

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

```

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

```

3.3 `\xintgodef`, `\xintgoodef`, `\xintgfdef`

1.09i. For use in `\xintAssign`.

```

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

```

3.4 `\xintRevWithBraces`

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

```

52 \def\xintRevWithBraces          {\romannumeral0\xintrevwithbraces }%

```

```

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

```

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

```

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

```

3.5 \xintZapFirstSpaces

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

```

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

```

If the original #1 started with a space, the grabbed #1 is empty. Thus `_again?` will see `#1=\xint_bye`, and hand over control to `_again` which will loop back into `\XINT_zapbsp_a`, with one

TOC, xintkernel, [xinttools](#), xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

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

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

3.6 \xintZapLastSpaces

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

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

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

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

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

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

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

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

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

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

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

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

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

3.7 \xintZapSpaces

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

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

3.8 \xintZapSpacesB

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

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

```

3.9 \xintCSVtoList, \xintCSVtoListNonStripped

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

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

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

```

```

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

```

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

```

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

```



```

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

```

3.10 \xintListWithSep

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

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

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

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

```

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

```

```

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

```

3.11 \xintNthElt

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

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

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

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

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

```

240 \def\xintNthElt          {\romannumeral0\xintnthelt }%
241 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand}%
242 \long\def\xintnthelt #1#2{\expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
243   \expandafter{\romannumeral`&&@#2}}%
244 \def\xintntheltnoexpand #1{\expandafter\XINT_nthelt_a\the\numexpr #1.}%
245 \def\XINT_nthelt_a #1%
246 {%
247   \xint_UDzerominusfork
248   #1-\XINT_nthelt_zero
249   0#1\XINT_nthelt_neg
250   0-{\XINT_nthelt_pos #1}%
251   \krof
252 }%
253 \def\XINT_nthelt_zero #1.{\xintlength}%
254 \long\def\XINT_nthelt_neg #1.#2%
255 {%
256   \expandafter\XINT_nthelt_neg_a\the\numexpr\xint_c_i+\XINT_length_loop
257   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
258   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
259   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c\xint_bye
260   -#1.#2\xint_bye
261 }%
262 \def\XINT_nthelt_neg_a #1%
263 {%
264   \xint_UDzerominusfork
265   #1-\xint_bye_thenstop
266   0#1\xint_bye_thenstop
267   0-{}%
268   \krof
269   \expandafter\XINT_nthelt_neg_b
270   \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
271 }%
272 \long\def\XINT_nthelt_neg_b #1#2\xint_bye{ #1}%

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```

273 \long\def\XINT_nthelt_pos #1.#2%
274 {%
275     \expandafter\XINT_nthelt_pos_done
276     \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_x.%
277     #2\xint:\xint:\xint:\xint:\xint:%
278     \xint:\xint:\xint:\xint:\xint:%
279     \xint_bye
280 }%
281 \def\XINT_nthelt_pos_done #1{%
282 \long\def\XINT_nthelt_pos_done ##1##2\xint_bye{%
283     \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:#1##1}%
284 }\XINT_nthelt_pos_done{ }%

```

3.12 \xintKeep

First included in release 1.09m.

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

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

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

`\xintKeepNoExpand` does not expand the L argument.

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

```

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

```

```

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

```

3.13 \xintKeepUnbraced

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

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

Modified for 1.2i like `\xintKeep`.

```

351 \def\xintKeepUnbraced      {\romannumeral0\xintkeepunbraced }%
352 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
353 \long\def\xintkeepunbraced #1#2%

```

```

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

```

3.14 \xintTrim

First included in release 1.09m.

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

`\xintTrimNoExpand` does not expand the L argument.

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

```

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

```

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

```

435 \long\def\XINT_trim_pos #1.#2%
436 {%
437   \expandafter\XINT_trim_pos_done\expandafter\space
438   \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
439   #2\xint:\xint:\xint:\xint:\xint:

```

```

440      \xint:\xint:\xint:\xint:\xint:%
441      \xint_bye
442 }%
443 \def\xINT_trim_loop #1#2.%
444 {%
445      \xint_gob_til_minus#1\xINT_trim_finish-%
446      \expandafter\xINT_trim_loop\the\numexpr#1#2\xINT_trim_loop_trimnine
447 }%
448 \long\def\xINT_trim_loop_trimnine #1#2#3#4#5#6#7#8#9%
449 {%
450      \xint_gob_til_xint: #9\xINT_trim_toofew\xint:-\xint_c_ix.%
451 }%
452 \def\xINT_trim_toofew\xint:{*\xint_c_}%
453 \def\xINT_trim_finish#1{%
454 \def\xINT_trim_finish-%
455      \expandafter\xINT_trim_loop\the\numexpr-##1\xINT_trim_loop_trimnine
456 {%
457      \expandafter\expandafter\expandafter#1%
458      \csname xint_gobble_\romannumeral\numexpr\xint_c_ix-##1\endcsname
459 }}\xINT_trim_finish{ }%
460 \long\def\xINT_trim_pos_done #1\xint:#2\xint_bye {#1}%

```

3.15 \xintTrimUnbraced

1.2a. Modified in 1.2i like \xintTrim

```

461 \def\xintTrimUnbraced      {\romannumeral0\xinttrimunbraced }%
462 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
463 \long\def\xinttrimunbraced #1#2%
464      {\expandafter\xINT_trimunbr_a\the\numexpr #1\expandafter.%
465      \expandafter{\romannumeral`&&@#2}}%
466 \def\xinttrimunbracednoexpand #1%
467      {\expandafter\xINT_trimunbr_a\the\numexpr #1.}%
468 \def\xINT_trimunbr_a #1%
469 {%
470      \xint_UDzerominusfork
471      #1-\xINT_trim_trimnone
472      0#1\xINT_trimunbr_neg
473      0-{\xINT_trim_pos #1}%
474      \krof
475 }%
476 \long\def\xINT_trimunbr_neg #1.#2%
477 {%
478      \expandafter\xINT_trimunbr_neg_a\the\numexpr
479      #1-\numexpr\xINT_length_loop
480      #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
481      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
482      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
483      .{ }#2\xint_bye
484 }%
485 \def\xINT_trimunbr_neg_a #1%
486 {%
487      \xint_UDsignfork

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

```

488      #1{\expandafter\XINT_keeppunbr_loop\the\numexpr-\xint_c_viii+}%
489      -\XINT_trim_trimall
490    \krof
491 }%

```

3.16 \xintApply

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

```

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

```

3.17 \xintApplyUnbraced

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

```

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

```



```

526 }%
527 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
528 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
529 \expandafter #1#2#3{ #2}%

```

3.18 \xintSeq

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

```

530 \def\xintSeq {\romannumeral0\xintseq }%
531 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
532 \def\XINT_seq_chkopt #1%
533 {%
534   \ifx [#1\expandafter\XINT_seq_opt
535     \else\expandafter\XINT_seq_noopt
536   \fi #1%
537 }%
538 \def\XINT_seq_noopt #1\xint_bye #2%
539 {%
540   \expandafter\XINT_seq\expandafter
541     {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
542 }%
543 \def\XINT_seq #1#2%
544 {%
545   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
546     \expandafter\xint_firstoftwo_thenstop
547   \or
548     \expandafter\XINT_seq_p
549   \else
550     \expandafter\XINT_seq_n
551   \fi
552   {#2}{#1}%
553 }%
554 \def\XINT_seq_p #1#2%
555 {%
556   \ifnum #1>#2
557     \expandafter\expandafter\expandafter\XINT_seq_p
558   \else
559     \expandafter\XINT_seq_e
560   \fi
561   \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
562 }%
563 \def\XINT_seq_n #1#2%
564 {%
565   \ifnum #1<#2
566     \expandafter\expandafter\expandafter\XINT_seq_n
567   \else
568     \expandafter\XINT_seq_e
569   \fi
570   \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
571 }%
572 \def\XINT_seq_e #1#2#3{ }%

```

```

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

```

```

625      \expandafter\XINT_sequo_nb
626      \fi
627      {#1}{#2}{#3}{#1}}%
628 }%
629 \def\XINT_sequo_nb #1#2#3%
630 {%
631      \expandafter\XINT_sequo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
632 }%
633 \def\XINT_sequo_nc #1#2%
634 {%
635      \ifnum #1<#2
636          \expandafter\XINT_sequo_o
637      \else
638          \expandafter\XINT_sequo_nd
639      \fi
640      {#1}{#2}%
641 }%
642 \def\XINT_sequo_nd #1#2#3#4{\XINT_sequo_nb {#1}{#2}{#3}{#4{#1}}}%

```

3.19 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

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

```

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

```

3.20 \xintilloop, \xintilloopindex, \xintbracedilloopindex, \xintouterilloopindex, \xintbracedouterilloopindex, \xintbreakilloop, \xintbreakilloopanddo, \xintilloopskiptonext, \xintilloopskipandredo

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

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

```

650 \def\xintilloop [#1+#2]{%
651      \expandafter\xintilloop_a\the\numexpr #1\expandafter.\the\numexpr #2.%
652 \long\def\xintilloop_a #1.#2.#3#4\repeat{%
653      #3#4\xintilloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
654 \def\xintilloop_again\fi\xint_gobble_iii #1#2{%
655      \fi\expandafter\xintilloop_again_b\the\numexpr#1+#2.#2.%
656 \long\def\xintilloop_again_b #1.#2.#3{%
657      #3\xintilloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
658 \long\def\xintbreakilloop #1\xintilloop_again\fi\xint_gobble_iii #2#3#4{%
659 \long\def\xintbreakilloopanddo
660      #1.#2\xintilloop_again\fi\xint_gobble_iii #3#4#5{#1}%
661 \long\def\xintilloopindex #1\xintilloop_again\fi\xint_gobble_iii #2%
662      {#2#1\xintilloop_again\fi\xint_gobble_iii {#2}}%
663 \long\def\xintbracedilloopindex #1\xintilloop_again\fi\xint_gobble_iii #2%

```

```

664      {{#2}#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
665 \long\def\xintouterilooopindex #1\xintilooop_again
666      #2\xintilooop_again\fi\xint_gobble_iii #3%
667      {#3#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
668 \long\def\xintbracedouterilooopindex #1\xintilooop_again
669      #2\xintilooop_again\fi\xint_gobble_iii #3%
670      {{#3}#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
671 \long\def\xintilooopskiptonext #1\xintilooop_again\fi\xint_gobble_iii #2#3{%
672      \expandafter\xintilooop_again_b \the\numexpr#2+#3.#3.}%
673 \long\def\xintilooopskipandredo #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
674      #4\xintilooop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%

```

3.21 \XINT_xflet

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

```

675 \def\XINT_xflet #1%
676 {%
677     \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
678 }%
679 \def\XINT_xflet_zapsp
680 {%
681     \expandafter\futurelet\expandafter\XINT_token
682     \expandafter\XINT_xflet_sp?\romannumeral`&&@%
683 }%
684 \def\XINT_xflet_sp?
685 {%
686     \ifx\XINT_token\XINT_sptoken
687         \expandafter\XINT_xflet_zapsp
688     \else\expandafter\XINT_xflet_zapspB
689     \fi
690 }%
691 \def\XINT_xflet_zapspB
692 {%
693     \expandafter\futurelet\expandafter\XINT_tokenB
694     \expandafter\XINT_xflet_spB?\romannumeral`&&@%
695 }%
696 \def\XINT_xflet_spB?
697 {%
698     \ifx\XINT_tokenB\XINT_sptoken
699         \expandafter\XINT_xflet_zapspB
700     \else\expandafter\XINT_xflet_eq?
701     \fi
702 }%
703 \def\XINT_xflet_eq?
704 {%
705     \ifx\XINT_token\XINT_tokenB
706         \expandafter\XINT_xflet_macro
707     \else\expandafter\XINT_xflet_zapsp
708     \fi
709 }%

```

3.22 \xintApplyInline

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

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

```

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

```

3.23 \xintFor, \xintFor*, \xintBreakFor, \xintBreakForAndDo

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

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

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

`\xint_secondoftwo` are made long.

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

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

```

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

```

```

788 }%
789 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
790 \long\def\XINT_forx *#1#2in#3#4#5%
791 {%
792   \expandafter\XINT_toks\expandafter
793     {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
794   \XINT_xflet\XINT_forx_forever? #3Z%
795 }%
796 \def\XINT_forx_forever?
797 {%
798   \ifx\XINT_token U\XINT_to_forxever\fi
799   \ifx\XINT_token V\XINT_to_forxever\fi
800   \ifx\XINT_token D\XINT_to_forxever\fi
801   \XINT_forx_empty?
802 }%
803 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
804 \catcode`U 11
805 \catcode`D 11
806 \catcode`V 11
807 \def\XINT_forx_empty?
808 {%
809   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
810   \the\XINT_toks
811 }%
812 \long\def\XINT_for_d #1#2#3%
813 {%
814   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
815   \XINT_toks {{#3}}%
816   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
817     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
818   \XINT_toks {\XINT_x\let\xintifForFirst\xint_seconddoftwo
819     \let\xintifForLast\xint_seconddoftwo\XINT_for_d #1{#2}}%
820   \futurelet\XINT_token\XINT_for_last?
821 }%
822 \long\def\XINT_forx_d #1#2#3%
823 {%
824   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
825   \XINT_toks {{#3}}%
826   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
827     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
828   \XINT_toks {\XINT_x\let\xintifForFirst\xint_seconddoftwo
829     \let\xintifForLast\xint_seconddoftwo\XINT_forx_d #1{#2}}%
830   \XINT_xflet\XINT_for_last?
831 }%
832 \def\XINT_for_last?
833 {%
834   \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
835   \the\XINT_toks
836 }%
837 \def\XINT_for_last?yes
838 {%
839   \let\xintifForLast\xint_firstoftwo

```

```
840 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
841 }%
```

3.24 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

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

```
842 \catcode`U 3
843 \catcode`D 3
844 \catcode`V 3
845 \let\xintegers      U%
846 \let\xintintegers  U%
847 \let\xintdimensions D%
848 \let\xintrationals  V%
849 \def\XINT_forever #1%
850 {%
851   \expandafter\XINT_forever_a
852   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
853   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
854   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
855 }%
856 \catcode`U 11
857 \catcode`D 11
858 \catcode`V 11
859 \def\XINT_?expr_Ua #1#2%
860   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
861     \expandafter\relax\expandafter}%
862   \expandafter{\the\numexpr #2}}%
863 \def\XINT_?expr_Da #1#2%
864   {\expandafter{\expandafter\dimexpr\the\dimexpr #1\expandafter\relax
865     \expandafter s\expandafter p\expandafter\relax\expandafter}%
866   \expandafter{\the\dimexpr #2}}%
867 \catcode`Z 11
868 \def\XINT_?expr_Va #1#2%
869 {%
870   \expandafter\XINT_?expr_Vb\expandafter
871   {\romannumeral`&&\xintZapSpacesB{#2}}%
872   {\romannumeral`&&\xintZapSpacesB{#1}}%
873 }%
874 \catcode`Z 3
875 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1}%
876 \def\XINT_?expr_Vc #1/#2.#3/#4.%
877 {%
878   \xintifEq {#2}{#4}%
879     {\XINT_?expr_Vf {#3}{#1}{#2}}%
880     {\expandafter\XINT_?expr_Vd\expandafter
881       {\romannumeral0\xintiimul {#2}{#4}}%
882       {\romannumeral0\xintiimul {#1}{#4}}%
883     }
```



```

883      {\romannumeral0\xintiimul {#2}{#3}}%
884      }%
885 }%
886 \def\xINT_?expr_Vd #1#2#3{\expandafter\xINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
887 \def\xINT_?expr_Ve #1#2{\expandafter\xINT_?expr_Vf\expandafter {#2}{#1}}%
888 \def\xINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}}%
889 \def\xINT_?expr_Ui {\numexpr 1\relax}{1}}%
890 \def\xINT_?expr_Di {\dimexpr 0pt\relax}{65536}}%
891 \def\xINT_?expr_Vi {{1/1}{0111}}}%
892 \def\xINT_?expr_U #1#2%
893   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
894 \def\xINT_?expr_D #1#2%
895   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
896 \def\xINT_?expr_V #1#2{\xINT_?expr_Vx #2}%
897 \def\xINT_?expr_Vx #1#2%
898 {%
899   \expandafter\xINT_?expr_Vy\expandafter
900     {\romannumeral0\xinttiadd {#1}{#2}}{#2}%
901 }%
902 \def\xINT_?expr_Vy #1#2#3#4%
903 {%
904   \expandafter{\romannumeral0\xinttiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
905 }%
906 \def\xINT_forever_a #1#2#3#4%
907 {%
908   \ifx #4[\expandafter\xINT_forever_opt_a
909     \else\expandafter\xINT_forever_b
910   \fi #1#2#3#4%
911 }%
912 \def\xINT_forever_b #1#2#3Z{\expandafter\xINT_forever_c\the\xINT_toks #2#3}%
913 \long\def\xINT_forever_c #1#2#3#4#5%
914   {\expandafter\xINT_forever_d\expandafter #2#4#5{#3}Z}%
915 \def\xINT_forever_opt_a #1#2#3[#4+#5]#6Z%
916 {%
917   \expandafter\expandafter\expandafter
918     \xINT_forever_opt_c\expandafter\the\expandafter\xINT_toks
919     \romannumeral`&&@#1{#4}{#5}#3%
920 }%
921 \long\def\xINT_forever_opt_c #1#2#3#4#5#6{\xINT_forever_d #2{#4}{#5}#6{#3}Z}%
922 \long\def\xINT_forever_d #1#2#3#4#5%
923 {%
924   \long\def\xINT_y ##1##2##3##4##5##6##7##8##9{#5}%
925   \xINT_toks {{#2}}%
926   \long\edef\xINT_x {\noexpand\xINT_y \csname XINT_for_left#1\endcsname
927     \the\xINT_toks \csname XINT_for_right#1\endcsname }%
928   \xINT_x
929   \let\xintifForFirst\xint_secondoftwo
930   \let\xintifForLast\xint_secondoftwo
931   \expandafter\xINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
932 }%

```

3.25 \xintForpair, \xintForthree, \xintForfour

1.09c.

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

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

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

```

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

```

```

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

```

3.26 \xintAssign, \xintAssignArray, \xintDigitsOf

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

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

```

993 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
994 \def\XINT_assign_fork
995 {%
996 \let\XINT_assign_def\def
997 \ifx\XINT_token[\expandafter\XINT_assign_opt
998 \else\expandafter\XINT_assign_a
999 \fi
1000 }%
1001 \def\XINT_assign_opt [#1]%
1002 {%
1003 \ifcsname #1def\endcsname
1004 \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
1005 \else
1006 \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
1007 \fi
1008 \XINT_assign_a
1009 }%
1010 \long\def\XINT_assign_a #1\to
1011 {%
1012 \def\XINT_flet_macro{\XINT_assign_b}%
1013 \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
1014 }%
1015 \long\def\XINT_assign_b
1016 {%
1017 \ifx\XINT_token\bgroup
1018 \expandafter\XINT_assign_c
1019 \else\expandafter\XINT_assign_f
1020 \fi

```

```

1021 }%
1022 \long\def\XINT_assign_f #1\xint:\to #2%
1023 {%
1024     \XINT_assign_def #2{#1}%
1025 }%
1026 \long\def\XINT_assign_c #1%
1027 {%
1028     \def\xint_temp {#1}%
1029     \ifx\xint_temp\xint_bracedstopper
1030         \expandafter\XINT_assign_e
1031     \else
1032         \expandafter\XINT_assign_d
1033     \fi
1034 }%
1035 \long\def\XINT_assign_d #1\to #2%
1036 {%
1037     \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1038     \XINT_assign_c #1\to
1039 }%
1040 \def\XINT_assign_e #1\to {}%
1041 \def\xintRelaxArray #1%
1042 {%
1043     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1044     \escapechar -1
1045     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1046     \XINT_restoreescapechar
1047     \xintilop [\csname\xint_arrayname 0\endcsname+-1]
1048     \global
1049     \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
1050     \ifnum \xintilopindex > \xint_c_
1051     \repeat
1052     \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1053     \global\let #1\relax
1054 }%
1055 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1056     \XINT_flet_zapsp }%
1057 \def\XINT_assignarray_fork
1058 {%
1059     \let\XINT_assignarray_def\def
1060     \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1061         \else\expandafter\XINT_assignarray
1062     \fi
1063 }%
1064 \def\XINT_assignarray_opt [#1]%
1065 {%
1066     \ifcsname #1def\endcsname
1067         \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1068     \else
1069         \expandafter\let\expandafter\XINT_assignarray_def
1070             \csname xint#1def\endcsname
1071     \fi
1072     \XINT_assignarray

```

```

1073 }%
1074 \long\def\XINT_assignarray #1\to #2%
1075 {%
1076     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1077     \escapechar -1
1078     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1079     \XINT_restoreescapechar
1080     \def\xint_itemcount {0}%
1081     \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
1082     \csname\xint_arrayname 00\expandafter\endcsname
1083     \csname\xint_arrayname 0\expandafter\endcsname
1084     \expandafter {\xint_arrayname}#2%
1085 }%
1086 \long\def\XINT_assignarray_loop #1%
1087 {%
1088     \def\xint_temp {#1}%
1089     \ifx\xint_temp\xint_bracedstopper
1090         \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1091             \expandafter{\the\numexpr\xint_itemcount}%
1092         \expandafter\expandafter\expandafter\XINT_assignarray_end
1093     \else
1094         \expandafter\def\expandafter\xint_itemcount\expandafter
1095             {\the\numexpr\xint_itemcount+\xint_c_i}%
1096         \expandafter\XINT_assignarray_def
1097             \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1098             \expandafter{\xint_temp }%
1099         \expandafter\XINT_assignarray_loop
1100     \fi
1101 }%
1102 \def\XINT_assignarray_end #1#2#3#4%
1103 {%
1104     \def #4##1%
1105     {%
1106         \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1107     }%
1108     \def #1##1%
1109     {%
1110         \ifnum ##1<\xint_c_
1111             \xint_afterfi{\XINT_expandableerror{Array index negative: 0 > ##1} }%
1112         \else
1113             \xint_afterfi {%
1114                 \ifnum ##1>#2
1115                     \xint_afterfi
1116                     {\XINT_expandableerror{Array index beyond range: ##1 > #2} }%
1117                 \else\xint_afterfi
1118             {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1119             \fi}%
1120         \fi
1121     }%
1122 }%
1123 \let\xintDigitsOf\xintAssignArray

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

3.27 `\xintExpandArgs`

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

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

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

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

These macros will remain undocumented in the user manual:

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

-- crucial design elements may change in future:

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

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

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

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

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

Kept items (with no leading spaces; but first item special as it will have lost a leading space due to f-expansion) will lose a brace pair under `\xintKeep:f:csv` if the first argument was positive and strictly less than the length of the list. This differs of course from `\xintKeep` (which always braces items it outputs when used with positive first argument) and also from `\xintKeepUnbraced` in the case when the whole list is kept. Actually the case of singleton list is special, and brace removal will happen then.

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

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

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

3.28.1 `\xintLength:f:csv`

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

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

Must first check if empty list.

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

3.28.2 `\xintLengthUpTo:f:csv`

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

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

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

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

```

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

```

3.28.3 \xintKeep:f:csv

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

```

1191 \def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv}%
1192 \long\def\xintkeep:f:csv #1#2%
1193 {%
1194     \expandafter\xint_gobble_thenstop
1195     \romannumeral0\expandafter\XINT_keep:f:csv_a
1196     \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1197 }%
1198 \def\XINT_keep:f:csv_a #1%
1199 {%
1200     \xint_UDzerominusfork
1201     #1-\XINT_keep:f:csv_keeptime

```



```

1202      0#1\XINT_keep:f:csv_neg
1203      0-{XINT_keep:f:csv_pos #1}%
1204      \krof
1205 }%
1206 \long\def\XINT_keep:f:csv_keeptime .#1{,%
1207 \long\def\XINT_keep:f:csv_neg #1.#2%
1208 {%
1209     \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
1210     \romannumeral0%
1211     \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
1212     #1-\numexpr\XINT_length:f:csv_a
1213     #2\xint:,\xint:,\xint:,\xint:,%
1214     \xint:,\xint:,\xint:,\xint:,\xint:,%
1215     \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1216     \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1217     .#2\xint_bye
1218 }%
1219 \def\XINT_keep:f:csv_neg_a #1%
1220 {%
1221     \xint_UDsignfork
1222     #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
1223     -\XINT_keep:f:csv_keeptime
1224     \krof
1225 }%
1226 \def\XINT_keep:f:csv_keeptime #1.{ }%
1227 \long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
1228 \def\XINT_keep:f:csv_trimloop #1#2.%
1229 {%
1230     \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
1231     \expandafter\XINT_keep:f:csv_trimloop
1232     \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimtime
1233 }%
1234 \long\def\XINT_keep:f:csv_trimloop_trimtime #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
1235 \def\XINT_keep:f:csv_trimloop_finish-%
1236     \expandafter\XINT_keep:f:csv_trimloop
1237     \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimtime
1238     {\csname XINT_trim:f:csv_finish#1\endcsname}%
1239 \long\def\XINT_keep:f:csv_pos #1.#2%
1240 {%
1241     \expandafter\XINT_keep:f:csv_pos_fork
1242     \romannumeral0\XINT_lengthupto:f:csv_a
1243     #1.#2\xint:,\xint:,\xint:,\xint:,%
1244     \xint:,\xint:,\xint:,\xint:,%
1245     \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1246     \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1247     .#1.{ }#2\xint_bye%
1248 }%
1249 \def\XINT_keep:f:csv_pos_fork #1#2.%
1250 {%
1251     \xint_UDsignfork
1252     #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1253     -\XINT_keep:f:csv_keeptime

```

```

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

```

3.28.4 \xintTrim:f:csv

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

```

1284 \def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv}%
1285 \long\def\xinttrim:f:csv #1#2%
1286 {%
1287 \expandafter\xint_gobble_thenstop
1288 \romannumeral0\expandafter\XINT_trim:f:csv_a
1289 \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1290 }%
1291 \def\XINT_trim:f:csv_a #1%
1292 {%
1293 \xint_UDzerominusfork
1294 #1-\XINT_trim:f:csv_trimnone
1295 0#1\XINT_trim:f:csv_neg
1296 0-{\XINT_trim:f:csv_pos #1}%
1297 \krof
1298 }%
1299 \long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
1300 \long\def\XINT_trim:f:csv_neg #1.#2%
1301 {%

```

```

1302 \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
1303 #1-\numexpr\XINT_length:f:csv_a
1304 #2\xint:,\xint:,\xint:,\xint:,%
1305 \xint:,\xint:,\xint:,\xint:,\xint:,%
1306 \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1307 \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1308 .{ }#2\xint_bye
1309 }%
1310 \def\XINT_trim:f:csv_neg_a #1%
1311 {%
1312 \xint_UDsignfork
1313 #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1314 -\XINT_trim:f:csv_trimall
1315 \krof
1316 }%
1317 \def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
1318 \long\def\XINT_trim:f:csv_pos #1.#2%
1319 {%
1320 \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
1321 \romannumeral0%
1322 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1323 #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1324 \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
1325 }%
1326 \def\XINT_trim:f:csv_loop #1#2.%
1327 {%
1328 \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
1329 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
1330 }%
1331 \long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1332 {%
1333 \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
1334 }%
1335 \def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
1336 \def\XINT_trim:f:csv_finish-%
1337 \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
1338 {%
1339 \csname XINT_trim:f:csv_finish#1\endcsname
1340 }%
1341 \long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
1342 #1,#2,#3,#4,#5,#6,#7,#8,{ }%
1343 \long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
1344 #1,#2,#3,#4,#5,#6,#7,{ }%
1345 \long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
1346 #1,#2,#3,#4,#5,#6,{ }%
1347 \long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
1348 #1,#2,#3,#4,#5,{ }%
1349 \long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
1350 #1,#2,#3,#4,{ }%
1351 \long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
1352 #1,#2,#3,{ }%
1353 \long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname

```

```

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

```

3.28.5 \xintNthEltPy:f:csv

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

```

1359 \def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv }%
1360 \long\def\xintntheltpy:f:csv #1#2%
1361 {%
1362   \expandafter\XINT_nthelt:f:csv_a
1363   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1364 }%
1365 \def\XINT_nthelt:f:csv_a #1%
1366 {%
1367   \xint_UDsignfork
1368     #1\XINT_nthelt:f:csv_neg
1369     -\XINT_nthelt:f:csv_pos
1370   \krof #1%
1371 }%
1372 \long\def\XINT_nthelt:f:csv_neg -#1.#2%
1373 {%
1374   \expandafter\XINT_nthelt:f:csv_neg_fork
1375   \the\numexpr\XINT_length:f:csv_a
1376   #2\xint:,\xint:,\xint:,\xint:,%
1377   \xint:,\xint:,\xint:,\xint:,\xint:,%
1378   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1379   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1380   -#1.#2,\xint_bye
1381 }%
1382 \def\XINT_nthelt:f:csv_neg_fork #1%
1383 {%
1384   \if#1-\expandafter\xint_bye_thenstop\fi
1385   \expandafter\XINT_nthelt:f:csv_neg_done
1386   \romannumeral0%
1387   \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
1388 }%
1389 \long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
1390 \long\def\XINT_nthelt:f:csv_pos #1.#2%
1391 {%
1392   \expandafter\XINT_nthelt:f:csv_pos_done
1393   \romannumeral0%
1394   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1395   #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1396   \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
1397 }%
1398 \def\XINT_nthelt:f:csv_pos_done #1{%
1399 \long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
1400   \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%

```

```
1401 }\XINT_nthelt:f:csv_pos_done{ }%
```

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

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

3.28.6 `\xintReverse:f:csv`

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

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

3.28.7 `\xintFirstItem:f:csv`

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

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

3.28.8 `\xintLastItem:f:csv`

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

```

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

```

3.28.9 Public names for the undocumented csv macros: `\xintCSVLength`, `\xintCSVKeep`, `\xintCSVTrim`, `\xintCSVNthEltPy`, `\xintCSVReverse`, `\xintCSVFirstItem`, `\xintCSVLastItem`

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

```

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

```

4 Package [xintcore](#) implementation

.1	Catcodes, ε -TeX and reload detection . . .	55	.25	\XINT_zeroes_forviii	68
.2	Package identification	56	.26	\XINT_sepbyviii_Z	68
.3	(WIP!) Error conditions and exceptions . . .	56	.27	\XINT_sepbyviii_andcount	69
.4	Counts for holding needed constants . . .	58	.28	\XINT_rsepyviii	69
Routines handling integers as lists of token digits			.29	\XINT_sepandrev	70
.5	\XINT_cuz_small	59	.30	\XINT_sepandrev_andcount	70
.6	\xintNum, \xintiNum	59	.31	\XINT_rev_nounsep	71
.7	\xintiiSgn	60	.32	\XINT_unrevbyviii	71
.8	\xintiiOpp	61	Core arithmetic		71
.9	\xintiiAbs	61	.33	\xintiiAdd	72
.10	\xintFDg	61	.34	\xintiiCmp	75
.11	\xintLDg	62	.35	\xintiiSub	77
.12	\xintDouble	62	.36	\xintiiMul	83
.13	\xintHalf	63	.37	\xintiiDivision	87
.14	\xintInc	63	Derived arithmetic		102
.15	\xintDec	64	.38	\xintiiQuo, \xintiiRem	102
.16	\xintDSL	64	.39	\xintiiDivRound	102
.17	\xintDSR	64	.40	\xintiiDivTrunc	103
.18	\xintDSRr	65	.41	\xintiiModTrunc	103
Blocks of eight digits42	\xintiiDivMod	104
.19	\XINT_cuz	65	.43	\xintiiDivFloor	105
.20	\XINT_cuz_byviii	66	.44	\xintiiMod	105
.21	\XINT_unsep_loop	66	.45	\xintiiSqr	105
.22	\XINT_unsep_cuzsmall	67	.46	\xintiiPow	106
.23	\XINT_div_unsepQ	67	.47	\xintiiFac	109
.24	\XINT_div_unsepR	68	.48	\XINT_useiimessage	112

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

The core arithmetic routines have been entirely rewritten for release 1.2. The 1.2i and 1.21 brought again some improvements.

The commenting continues (2018/06/17) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

1.3 removes all macros which were deprecated at 1.2o.

4.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

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

```

4.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2018/06/17 1.3c Expandable arithmetic on big integers (JFB)]%

```

4.3 (WIP!) Error conditions and exceptions

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

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

and the Python3 implementation in its Decimal module.

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

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

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

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

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

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

Various conditions can raise the same signal.

Only signals, not conditions, raise Flags.

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

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

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

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

Only four conditions are currently possibly encountered:

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

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

```

47 \csname XINT_Clamped_istrapped\endcsname
48 \csname XINT_ConversionSyntax_istrapped\endcsname
49 \csname XINT_DivisionByZero_istrapped\endcsname
50 \csname XINT_DivisionImpossible_istrapped\endcsname
51 \csname XINT_DivisionUndefined_istrapped\endcsname
52 \csname XINT_InvalidOperation_istrapped\endcsname
53 \csname XINT_Overflow_istrapped\endcsname
54 \csname XINT_Underflow_istrapped\endcsname

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

```

```

73     \xint_dothis{\csname XINT_#2.handler\endcsname {#4}}%
74     \fi
75     \xint_orthat{%
76         % the flag raised is named after the signal #1, but we show condition #2
77         \XINT_expandableerror{#2 (hit <RET> thrice)}%
78         \XINT_expandableerror{#3}%
79         \XINT_expandableerror{next: #5}%
80         % not for X3.274
81         %\XINT_expandableerror{<RET>, or I\xintUse{...}<RET>, or I\xintCTRLC<RET>}%
82         \xint_firstofone_thenstop{#5}%
83     }%
84 }%
85 %% \let\xintUse\xint_firstofthree_thenstop % defined in xint.sty
86 \def\XINT_ifFlagRaised #1{%
87     \ifcsname XINT_#1Flag_ON\endcsname
88         \expandafter\xint_firstoftwo
89     \else
90         \expandafter\xint_secondoftwo
91     \fi}%
92 \def\XINT_resetFlag #1%
93     {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
94 \def\XINT_resetFlags {% WIP
95     \XINT_resetFlag{InvalidOperation}% also from DivisionUndefined
96     \XINT_resetFlag{DivisionByZero}%
97     \XINT_resetFlag{Underflow}% (\xintiiPow with negative exponent)
98     \XINT_resetFlag{Overflow}% not encountered so far in xint code 1.2l
99     % .. others ..
100 }%
101 \def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
102     NOT IMPLEMENTED! WORK IN PROGRESS! (ALL SIGNALS TRAPPED, NO HANDLERS USED)
102 \catcode`. 11
103 \let\XINT_Clamped.handler\xint_firstofone % WIP
104 \def\XINT_InvalidOperation.handler#1{NaN}% WIP
105 \def\XINT_ConversionSyntax.handler#1{NaN}% WIP
106 \def\XINT_DivisionByZero.handler#1{SignedInfinity(#1)}% WIP
107 \def\XINT_DivisionImpossible.handler#1{NaN}% WIP
108 \def\XINT_DivisionUndefined.handler#1{NaN}% WIP
109 \let\XINT_Inexact.handler\xint_firstofone % WIP
110 \def\XINT_InvalidContext.handler#1{NaN}% WIP
111 \let\XINT_Rounded.handler\xint_firstofone % WIP
112 \let\XINT_Subnormal.handler\xint_firstofone% WIP
113 \def\XINT_Overflow.handler#1{NaN}% WIP
114 \def\XINT_Underflow.handler#1{NaN}% WIP
115 \catcode`. 12

```

4.4 Counts for holding needed constants

```

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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

```

121 \ifdefined\xint_c_x^ix\else
122 \csname newcount\endcsname\xint_c_x^ix \xint_c_x^ix 1000000000
123 \fi
124 \newcount\xint_c_x^viii_mone \xint_c_x^viii_mone 99999999
125 \newcount\xint_c_x^ii_e_viii \xint_c_x^ii_e_viii 1200000000
126 \newcount\xint_c_x^xi_e_viii_mone \xint_c_x^xi_e_viii_mone 1099999999

```

Routines handling integers as lists of token digits

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

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

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

4.5 \XINT_cuz_small

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

```

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

```

4.6 \xintNum, \xintiNum

For example `\xintNum {-----+-----0000000000000003}`

Very old routine got completely rewritten at 1.21.

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

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

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

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

`\xintNum` gets redefined by `xintfrac`.

```

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

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfraction, xintexpr, indices

```

146 {%
147   \xint_gob_til_xint: #9\XINT_num_end\xint:
148   #1#2#3#4#5#6#7#8#9%
149   \ifnum \numexpr #1#2#3#4#5#6#7#8#9+\xint_c_ = \xint_c_

```

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

```

150   \expandafter\XINT_num_loop
151   \else

```

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

```

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

```

4.7 \xintiisgn

1.21 made \xintiisgn robust against non terminated input.

1.20 deprecates here \xintSgn (it requires xintfrac.sty).

```

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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

4.8 \xintiiOpp

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

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

4.9 \xintiiAbs

Attention \xintiiAbs non robust against non terminated input.

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

4.10 \xintFDg

FIRST DIGIT.

1.21: \xintiiFDg made robust against non terminated input.

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

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


[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

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

4.13 \xintHalf

Attention \xintHalf non robust against non terminated input.

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

4.14 \xintInc

1.2i much delayed complete rewrite in 1.2 style.

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

Attention \xintInc non robust against non terminated input.

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

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

4.15 \xintDec

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

Attention \xintDec non robust against non terminated input.

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

4.16 \xintDSL

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

Attention \xintDSL non robust against non terminated input.

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

4.17 \xintDSR

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

Attention \xintDSR non robust against non terminated input.

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


[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```

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

```

4.18 \xintDSRr

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

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

Attention \xintDSRr non robust against non terminated input.

```

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

```

Blocks of eight digits

The lingua of release 1.2.

4.19 \XINT_cuz

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

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

TOC, xintkernel, xinttools, [xintcore](#), xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

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

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

4.20 \XINT_cuz_byviii

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

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

4.21 \XINT_unsep_loop

This is used as

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

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

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

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

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

```

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

```

4.22 \XINT_unsep_cuzsmall

This is used as

```

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

```

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

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

```

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

```

4.23 \XINT_div_unsepQ

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

Rewritten for 1.2l in 1.2i style.

```

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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

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

4.24 \XINT_div_unsepR

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

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

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

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

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

4.25 \XINT_zeroes_forviii

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

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

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

4.26 \XINT_sepbyviii_Z

This is used as

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

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

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

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

4.27 \XINT_sepbyviii_andcount

This is used as

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

It will produce

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

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

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

```

4.28 \XINT_rsepbyviii

This is used as

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

and will produce

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

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

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

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

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

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

4.29 \XINT_sepandrev

This is used typically as

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

and will produce

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

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

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

4.30 \XINT_sepandrev_andcount

This is used typically as

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

and will produce

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

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

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

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

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

```

4.31 \XINT_rev_nounsep

This is used as

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

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

```

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

```

4.32 \XINT_unrevbyviii

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

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

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

```

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

```

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

```

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

```

Core arithmetic

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

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

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

4.33 `\xintiiAdd`

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

```

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

```



```

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

```

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

```

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

```

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

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

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

```

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

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

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

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

```

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

```

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

```

4.34 \xintiiCmp

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

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

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

```

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

```

```

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

```

```

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

```

4.35 \xintiiSub

Entirely rewritten for 1.2.

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

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

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

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

```

786 \def\xintiiSub {\romannumeral0\xintiisub }%
787 \def\xintiisub #1{\expandafter\xINT_iisub\romannumeral`&&@#1\xint:}%
788 \def\xINT_iisub #1#2\xint:#3%
789 {%
790 \expandafter\xINT_sub_nfork\expandafter

```

```

791 #1\romannumeral`&&@#3\xint:#2\xint:
792 }%
793 \def\XINT_isub #1#2\xint:#3%
794 {%
795 \expandafter\XINT_sub_nfork\expandafter
796 #1\romannumeral0\xintnum{#3}\xint:#2\xint:
797 }%
798 \def\XINT_sub_nfork #1#2%
799 {%
800 \xint_UDzerofork
801 #1\XINT_sub_firstiszero
802 #2\XINT_sub_secondiszero
803 0{}}%
804 \krof
805 \xint_UDsignsfork
806 #1#2\XINT_sub_minusminus
807 #1-\XINT_sub_minusplus
808 #2-\XINT_sub_plusminus
809 --\XINT_sub_plusplus
810 \krof #1#2%
811 }%
812 \def\XINT_sub_firstiszero #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
813 \def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
814 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
815 \def\XINT_sub_plusplus #1#2%
816 {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
817 \def\XINT_sub_minusplus #1#2%
818 {\expandafter-\romannumeral0\XINT_add_pp_a {}}#2}%
819 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a {}}}%
820 \def\XINT_sub_mm_a #1#2#3\xint:
821 {%
822 \expandafter\XINT_sub_mm_b
823 \romannumeral0\expandafter\XINT_sepandrev_andcount
824 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
825 #2#3\XINT_rsepbyviii_end_A 2345678%
826 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
827 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
828 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
829 \X #1%
830 }%
831 \def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
832 {%
833 \expandafter\XINT_sub_checklengths
834 \the\numexpr #1\expandafter\xint:%
835 \romannumeral0\expandafter\XINT_sepandrev_andcount
836 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
837 #3\XINT_rsepbyviii_end_A 2345678%
838 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
839 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
840 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
841 1;!1;!1;!1;!1\W
842 #21;!1;!1;!1;!1\W

```

```

843      1;!1\R!1\R!1\R!1\R!%
844      1\R!1\R!1\R!1\R!\W
845 }%
846 \def\XINT_sub_checklengths #1\xint:#2\xint:%
847 {%
848     \ifnum #2>#1
849         \expandafter\XINT_sub_exchange
850     \else
851         \expandafter\XINT_sub_aa
852     \fi
853 }%
854 \def\XINT_sub_exchange #1\W #2\W
855 {%
856     \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
857 }%
858 \def\XINT_sub_aa
859 {%
860     \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
861 }%

```

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

```

862 \def\XINT_sub_out {\XINT_unrevbyviii{}}%

```

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

Call: \the\numexpr

```

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

```

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

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

\numexpr governed expansion stops with various possibilities:

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

Various type of post actions are then needed:

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

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

The 1.2l method for this correction is (presumably, testing takes lots of time, which I do not have) more efficient than in 1.2 release.

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

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

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



```

908 {%
909   \XINT_sub_k #1#2!#5!#7!#9!%
910 }%
911 \def\XINT_sub_di;%
912   \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3\xint:
913   #4!#5!#6!#7!#8\W
914 {%
915   \XINT_sub_k #1#2!#5!#7!%
916 }%
917 \def\XINT_sub_fi;%
918   \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
919   #4!#5!#6\W
920 {%
921   \XINT_sub_k #1#2!#5!%
922 }%
923 \def\XINT_sub_hi;%
924   \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3\xint:
925   #4\W
926 {%
927   \XINT_sub_k #1#2!%
928 }%

```

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

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

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

```

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

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

```

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

```

946 \def\XINT_sub_l_Ica#1\W

```

```

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

```

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

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

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

```

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

```

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

```

975 \def\XINT_sub_fix_none;{\XINT_cuz_small}%
976 \def\XINT_sub_fix_cuz ;{\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop}%

```

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

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

```

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

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

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

```

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

```

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

```

4.36 \xintiiMul

Completely rewritten for 1.2.

1.21: \xintiiMul made robust against non terminated input.

```

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

```

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

```

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

```

```

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

```

Cooking recipe, 2015/10/05.

```

1043 \def\XINT_mul_checklengths #1\xint:#2\xint:%
1044 {%
1045   \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
1046   \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
1047   \ifnum #2<#1
1048     \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
1049       \XINT_mul_exchange
1050     \fi
1051   \else
1052     \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
1053       \XINT_mul_exchange
1054     \fi
1055   \fi
1056   \XINT_mul_start
1057 }%
1058 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1;!\W
1059 {%
1060   \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1061   \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1062   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
1063 }%
1064 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1;!\%
1065 {%
1066   \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1067   \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1068   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
1069 }%
1070 \def\XINT_mul_oneisone #1!\{\XINT_mul_out }%
1071 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
1072   \the\numexpr\XINT_smallmul 1#1!%
1073   {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0\xint:#1!}%
1074 \def\XINT_mul_exchange #1\XINT_mul_start #2\W #3!;!\%
1075   {\fi\fi\XINT_mul_start #3!;!\W #2}%

```

```

1076 \def\XINT_mul_start
1077   {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1;!\W}%
1078 \def\XINT_mul_out
1079   {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%

```

Call:

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

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

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

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

```

1080 \def\XINT_mul_loop #1\W #2\W 1#3!%
1081 {%
1082   \xint_gob_til_sc #3\XINT_mul_e ;%
1083   \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
1084   #1\W #2\W
1085 }%

```

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

```

1086 \def\XINT_mul_a #1\W #2\W
1087 {%
1088   \expandafter\XINT_mul_b\the\numexpr
1089   \XINT_add_a \xint_c_ii #21;!1;!1;!\W #11;!1;!1;!\W\W
1090 }%
1091 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
1092 \def\XINT_mul_e;#1\W 1#2\W #3\W {1\relax #2}%

```

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

The `minimulwc` has `1<8digits carry>.<4 high digits>.<4 low digits!<8digits>.`

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

Multiplication by zero will produce blocks of zeros.

```

1093 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1094 {%
1095   \expandafter\XINT_minimulwc_b
1096   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
1097               #3*#4#5#6#7+#2*#8\xint:
1098               #2*#4#5#6#7\xint:%
1099 }%
1100 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1101 {%
1102   \expandafter\XINT_minimulwc_c
1103   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%

```

```

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

1120 \def\XINT_verysmallmul #1\xint:#2!1#3!%
1121 {%
1122     \xint_gob_til_sc #3\XINT_verysmallmul_e;%
1123     \expandafter\XINT_verysmallmul_a
1124     \the\numexpr #2*#3+#1\xint:#2!%
1125 }%
1126 \def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
1127     #1+#2#3\xint:#4!%
1128 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
1129 \def\XINT_verysmallmul_f #1!1{1\relax}%
1130 \def\XINT_verysmallmul_a #1#2\xint:%
1131 {%
1132     \unless\ifnum #1#2<\xint_c_x^ix
1133     \expandafter\XINT_verysmallmul_bi\else
1134     \expandafter\XINT_verysmallmul_bj\fi
1135     \the\numexpr \xint_c_x^ix+#1#2\xint:%
1136 }%
1137 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
1138 \def\XINT_verysmallmul_cj 1#1#2\xint:%
1139     {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:}%
1140 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1141     {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2\xint:}%

```

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

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

```

1142 \def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
1143 {%
1144     \expandafter\XINT_minimul_b
1145     \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%
1146 }%
1147 \def\XINT_minimul_b 1#1#2#3#4#5\xint:#6\xint:%
1148 {%
1149     \expandafter\XINT_minimul_c
1150     \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%

```

```

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

```

4.37 \xintiDivision

Completely rewritten for 1.2.

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

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

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

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

```

1156 \def\xintiDivision    {\romannumeral0\xintiDivision }%
1157 \def\xintiDivision    #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:}%
1158 \def\XINT_iidivision #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1159                                     \romannumeral`&&@#3\xint:#2\xint:}%

```

On regarde les signes de A et de B.

```

1160 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1161 {%
1162     \if0#2\xint_dothis{\XINT_iidivision_divbyzero #1#2}\fi
1163     \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1164     \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1165                         \romannumeral0\XINT_iidivision_bpos #1}\fi
1166     \xint_orthat{\XINT_iidivision_bpos #1#2}%
1167 }%
1168 \def\XINT_iidivision_divbyzero#1#2#3\xint:#4\xint:
1169     {\if0#1\xint_dothis{\XINT_signalcondition{DivisionUndefined}}\fi
1170     \xint_orthat{\XINT_signalcondition{DivisionByZero}}%
1171     {Division of #1#4 by #2#3}{\{0\}{0}}}%
1172 \def\XINT_iidivision_aiszero #1\xint:#2\xint:{\{0\}{0}}%
1173 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1174     {\expandafter{\romannumeral0\XINT_opp #1}}%
1175 \def\XINT_iidivision_bpos #1%
1176 {%
1177     \xint_UDsignfork
1178     #1\XINT_iidivision_aneg
1179     -{\XINT_iidivision_apos #1}%
1180     \krof
1181 }%

```

TOC, xintkernel, xinttools, [xintcore](#), xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

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

```

1182 \def\XINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
1183 \def\XINT_iidivision_aneg #1\xint:#2\xint:
1184   {\expandafter
1185     \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1186 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\xint:
1187   \expandafter\XINT_iidivision_aneg_rzero
1188   \else
1189     \expandafter\XINT_iidivision_aneg_rpos
1190   \fi {#1}{#2}}%
1191 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1192 \def\XINT_iidivision_aneg_rpos #1%
1193 {%
1194   \expandafter\XINT_iidivision_aneg_end\expandafter
1195     {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1196 }%
1197 \def\XINT_iidivision_aneg_end #1#2#3%
1198 {%
1199   \expandafter\xint_exchangetwo_keepbraces
1200   \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\xint:#2\xint:}{#1}% r-> b-r
1201 }%

```

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

```

1202 \def\XINT_div_prepare #1%
1203 {%
1204   \XINT_div_prepare_a #1\R\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1205 }%
1206 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1207 {%
1208   \xint_gob_til_R #9\XINT_div_prepare_small\R
1209   \XINT_div_prepare_b #9%
1210 }%

```

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

```

1211 \def\XINT_div_prepare_small\R #1!#2%
1212 {%
1213   \ifcase #2
1214     \or\expandafter\XINT_div_BisOne
1215     \or\expandafter\XINT_div_BisTwo
1216     \else\expandafter\XINT_div_small_a
1217     \fi {#2}%
1218 }%
1219 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1220 \def\XINT_div_BisTwo #1#2%
1221 {%

```


TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

1222 \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1223 \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1224 }%
1225 \def\XINT_div_BisTwo_a #1#2%
1226 {%
1227 \expandafter{\romannumeral0\XINT_half
1228 #2\xint_bye\xint_Bye345678\xint_bye
1229 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1230 }%

```

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

```

1231 \def\XINT_div_small_a #1#2%
1232 {%
1233 \expandafter\XINT_div_small_b
1234 \the\numexpr #1/\xint_c_ii\expandafter
1235 \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
1236 \romannumeral0%
1237 \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
1238 #2\XINT_sepbyviii_Z_end 2345678\relax
1239 }%

```

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

```

1240 \def\XINT_div_small_b #1!#2{#2#1!}%

```

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

```

1241 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1242 {%
1243 \xint_gob_til_R #9\XINT_div_smallsmall\R
1244 \expandafter\XINT_div_dosmallldiv
1245 \the\numexpr\expandafter\XINT_sepbyviii_Z
1246 \romannumeral0\XINT_zeroes_forviii
1247 #1#2#3#4#5#6#7#8#9%
1248 }%

```

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

```

1249 \def\XINT_div_smallsmall\R
1250 \expandafter\XINT_div_dosmallldiv
1251 \the\numexpr\expandafter\XINT_sepbyviii_Z
1252 \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1253 {{\XINT_div_dosmallsmall}{#1}}%
1254 \def\XINT_div_dosmallsmall #1\xint:1#2!#3%
1255 {%
1256 \expandafter\XINT_div_smallsmallend
1257 \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
1258 }%
1259 \def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
1260 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

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

```
1261 \def\XINT_div_dosmalldiv
1262     {\backslash\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a}}%
```

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

```
1263 \def\XINT_div_prepare_b
1264     {\backslash\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1265 \def\XINT_div_prepare_c #1!%
1266 {%
1267     \XINT_div_prepare_d #1.00000000!{#1}%
1268 }%
1269 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1270 {%
1271     \backslash\expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1272 }%
1273 \def\XINT_div_prepare_e #1!#2!#3#4%
1274 {%
1275     \XINT_div_prepare_f #4#3\X {#1}{#3}%
1276 }%
```

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

```
1277 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1278 {%
1279     \backslash\expandafter\XINT_div_prepare_g
1280     \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\backslash\expandafter
1281     \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\backslash\expandafter
1282     \xint:\the\numexpr #1#2#3#4#5#6#7#8\backslash\expandafter
1283     \xint:\romannumeral0\XINT_sepandrev_andcount
1284     #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1285     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1286     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1287     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1288     \X
1289 }%
1290 \def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
1291 {%
1292     \backslash\expandafter\XINT_div_prepare_h
1293     \the\numexpr\backslash\expandafter\XINT_sepbyviii_andcount
1294     \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1295     #8#7\XINT_sepbyviii_end 2345678\relax
1296     \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1297     \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_W
1298     {#1}{#2}{#3}{#4}{#5}{#6}%
```

```

1299 }%
1300 \def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6#7#8%
1301 {%
1302     \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1303 }%

```

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

```

1304 \def\XINT_div_start_a #1#2%
1305 {%
1306     \ifnum #1 < #2
1307         \expandafter\XINT_div_zeroQ
1308     \else
1309         \expandafter\XINT_div_start_b
1310     \fi
1311     {#1}{#2}%
1312 }%
1313 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1314 {%
1315     \expandafter\XINT_div_zeroQ_end
1316     \romannumeral0\XINT_unsep_cuzsmall
1317     #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
1318 }%
1319 \def\XINT_div_zeroQ_end #1\xint:#2%
1320     {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2\xint:}%

```

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

```

1321 \def\XINT_div_start_b #1#2#3#4#5#6%
1322 {%
1323     \expandafter\XINT_div_finish\the\numexpr
1324     \XINT_div_start_c {#2}\xint:#3\xint:{#6}{#1}{#2}{#4}{#5}{#6}}%
1325 }%
1326 \def\XINT_div_finish
1327 {%
1328     \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1329 }%
1330 \def\XINT_div_finish_a #1\Z #2\xint:{\XINT_div_finish_b #2\xint:{#1}}%

```

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

```

1331 \def\XINT_div_finish_b #1%
1332 {%
1333     \if0#1%
1334         \expandafter\XINT_div_finish_bRzero
1335     \else
1336         \expandafter\XINT_div_finish_bRpos
1337     \fi
1338     #1%
1339 }%
1340 \def\XINT_div_finish_bRzero 0\xint:#1#2{{#1}{0}}%
1341 \def\XINT_div_finish_bRpos #1\xint:#2#3%

```

```

1342 {%
1343   \expandafter\xint_exchangetwo_keepbraces\xINT_div_cleanR #1#3\xint:{#2}%
1344 }%
1345 \def\xINT_div_cleanR #1000000000\xint:{{#1}}%

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

1346 \def\xINT_div_start_c #1%
1347 {%
1348   \ifnum #1>\xint_c_vi
1349     \expandafter\xINT_div_start_ca
1350   \else
1351     \expandafter\xINT_div_start_cb
1352   \fi {#1}%
1353 }%
1354 \def\xINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
1355 {%
1356   \expandafter\xINT_div_start_c\expandafter
1357   {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!\xint:%
1358 }%
1359 \def\xINT_div_start_cb #1%
1360   {\csname XINT_div_start_c_\romannumeral\xintcvi\endcsname}%
1361 \def\xINT_div_start_c_i #1\xint:#2!%
1362   {\XINT_div_start_c_ #1#2!\xint:}%
1363 \def\xINT_div_start_c_ii #1\xint:#2!#3!%
1364   {\XINT_div_start_c_ #1#2!#3!\xint:}%
1365 \def\xINT_div_start_c_iii #1\xint:#2!#3!#4!%
1366   {\XINT_div_start_c_ #1#2!#3!#4!\xint:}%
1367 \def\xINT_div_start_c_iv #1\xint:#2!#3!#4!#5!%
1368   {\XINT_div_start_c_ #1#2!#3!#4!#5!\xint:}%
1369 \def\xINT_div_start_c_v #1\xint:#2!#3!#4!#5!#6!%
1370   {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!\xint:}%
1371 \def\xINT_div_start_c_vi #1\xint:#2!#3!#4!#5!#6!#7!%
1372   {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!#7!\xint:}%

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

1373 \def\xINT_div_start_c_ 1#1!#2\xint:#3\xint:#4#5#6%
1374 {%
1375   \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1376 }%

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

1377 \def\xINT_div_I_a #1#2%
1378 {%
1379   \expandafter\xINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
1380 }%
1381 \def\xINT_div_I_b #1%
1382 {%
1383   \xint_gob_til_zero #1\xINT_div_I_czero 0\xINT_div_I_c #1%
1384 }%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

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

```
1385 \def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1386 \def\XINT_div_I_c #1\xint:#2#3%
1387 {%
1388     \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
1389 }%
```

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

```
1390 \def\XINT_div_I_da #1\xint:%
1391 {%
1392     \ifnum #1>\xint_c_ix
1393         \expandafter\XINT_div_I_dP
1394     \else
1395         \ifnum #1<\xint_c_
1396             \expandafter\expandafter\expandafter\XINT_div_I_dN
1397         \else
1398             \expandafter\expandafter\expandafter\XINT_div_I_db
1399         \fi
1400     \fi
1401 }%
```

attention très mauvaises notations avec _b et _db.

```
1402 \def\XINT_div_I_dN #1\xint:%
1403 {%
1404     \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
1405 }%
1406 \def\XINT_div_I_db #1\xint:#2#3#4#5%
1407 {%
1408     \expandafter\XINT_div_I_dc\expandafter #1%
1409     \romannumeral0\expandafter\XINT_div_sub\expandafter
1410         {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1411         {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1412     \Z {#4}{#5}%
1413 }%
```

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

```
1414 \def\XINT_div_I_dc #1#2%
1415 {%
1416     \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1417     #1#2%
1418 }%
1419 \def\XINT_div_I_dd #1-\Z
1420 {%
1421     \if #11\expandafter\XINT_div_I_dz\fi
1422     \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
1423 }%
1424 \def\XINT_div_I_dz #1XX#2#3#4%
1425 {%
```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

1426      1#4\XINT_div_I_g {#2}%
1427 }%
1428 \def\XINT_div_I_de #1#2\Z #3#4#5{1#5+#1\XINT_div_I_g {#2}}%

      q.alpha, B, q0, L, K, {x'y},x, alpha'B«c» (q=0 has been intercepted) -> 1nouveauq.nouvel alpha,
      L, K, {x'y}, x, alpha',B«c»

1429 \def\XINT_div_I_dP #1\xint:#2#3#4#5#6%
1430 {%
1431      1#6+#1\expandafter\XINT_div_I_g\expandafter
1432      {\romannumeral0\expandafter\XINT_div_sub\expandafter
1433      {\romannumeral0\XINT_rev_nounsep {}}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1434      {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1435      }%
1436 }%

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

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

1437 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1438 {%
1439      \expandafter !\the\numexpr
1440      \ifnum#2=#3
1441          \expandafter\XINT_div_exittofinish
1442      \else
1443          \expandafter\XINT_div_I_h
1444      \fi
1445      {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
1446 }%

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

1447 \def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
1448 {%
1449      1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
1450      \romannumeral0\XINT_div_unsepR #2#3%
1451      \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R\xint:
1452 }%

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

1453 \def\XINT_div_I_h #1\xint:#2!#3\xint:#4#5%
1454 {%
1455      \XINT_div_II_b #1#2!\xint:{{#5}{#4}{#3}{#5}}%
1456 }%

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

1457 \def\XINT_div_II_b #1#2!#3!%
1458 {%

```

```

1459 \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1460 \XINT_div_II_c #1{1#2}{#3}%
1461 }%

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

1462 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
1463 {%
1464 \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1465 }%

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

1466 \def\XINT_div_II_c #1#2#3#4%
1467 {%
1468 \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1469 #1\xint:#2!#3!#4!{#1}{#2}#3!#4!%
1470 }%
1471 \def\XINT_div_xmini #1%
1472 {%
1473 \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1474 }%
1475 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1476 {%
1477 \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1478 }%
1479 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1480 {%
1481 \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1482 }%

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

1483 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000\xint:50000000!#1!#2!{#1!}%

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

1484 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%
1485 {%
1486 \expandafter\XINT_div_II_e
1487 \romannumeral0\expandafter\XINT_div_sub\expandafter
1488 {\romannumeral0\XINT_rev_nounsep }{#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1489 {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#9;!}%
1490 \xint:{#6}{#7}{#9}{#1#2#3#4#5}%
1491 }%

alpha.x',y,B,q1, {{x'y},x,K,L}, alpha', B, «c». Attention la soustraction spéciale doit main-
tenir les blocs 1<8>!

1492 \def\XINT_div_II_e 1#1!%
1493 {%
1494 \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
1495 \XINT_div_II_f 1#1!%
1496 }%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

100000000! alpha sur K chiffres. #2=x', #3=y, #4=B, #5=q1, #6={{x'y},x,K,L}, #7=alpha', B«c» -> {x'y}x,K,L (à diminuer de 1), {alpha sur K}B{q1}{alpha'}B«c»

```
1497 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%
1498 {%
1499     \XINT_div_II_k #6{#1}{#4}{#5}%
1500 }%
```

1<a1>!1<a2>!, alpha (sur K+1 blocs de 8). x', y, B, q1, {{x'y},x,K,L}, alpha', B,«c».

Here also we are dividing with x' which could be 10^8 in the exceptional case $x=99999999$. Must intercept it before sending to \XINT_div_mini.

```
1501 \def\XINT_div_II_f #1!#2!#3\xint:%
1502 {%
1503     \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
1504 }%
1505 \def\XINT_div_II_fa #1#2#3#4%
1506 {%
1507     \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1{#2}%
1508 }%
```

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

```
1509 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1510 {%
1511     \expandafter \XINT_div_II_h
1512     \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1513     \xint:\expandafter\expandafter\expandafter
1514     {\expandafter\xint_gob_til_exclam
1515     \romannumeral0\expandafter\XINT_div_sub\expandafter
1516     {\romannumeral0\XINT_rev_nounsep }#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1517     {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#71;!}%
1518     {#7}%
1519 }%
```

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

```
1520 \def\XINT_div_II_h 1#1\xint:#2#3#4%
1521 {%
1522     \XINT_div_II_k #4{#2}{#3}{#1}%
1523 }%
```

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

```
1524 \def\XINT_div_II_k #1#2#3#4#5%
1525 {%
1526     \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
1527 }%
1528 \def\XINT_div_II_l #1\xint:#2#3#4#51#6!%
1529 {%
1530     \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}}{{#5}{#6}1#6!}%
1531 }%
```


TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

{LK{x'y}x},x,a,alpha.B{q}alpha'B -> a, x, alpha, B, q, L, K, {x'y}, x, alpha', B<<C>

```
1532 \def\XINT_div_II_m #1#2#3#4\xint:#5#6%
1533 {%
1534     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1535 }%
```

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

```
1536 \def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1537 {%
1538     \expandafter\XINT_div_minimulwc_b
1539     \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
1540 }%
1541 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1542 {%
1543     \expandafter\XINT_div_minimulwc_c
1544     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1545 }%
1546 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1547 {%
1548     1#6#7\expandafter!%
1549     \the\numexpr\expandafter\XINT_div_smallmul_a
1550     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1551 }%
1552 \def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
1553 {%
1554     \xint_gob_til_sc #4\XINT_div_smallmul_e;%
1555     \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1556 }%
1557 \def\XINT_div_smallmul_e;\XINT_div_minimulwc_a 1#1\xint:#2;#3!{1\relax #1!}%
```

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

```
1558 \def\XINT_div_verysmallmul #1%
1559     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1560 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
1561     {1\relax #1100000000!}%
1562 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1563 {%
1564     \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1565     \expandafter\XINT_div_verysmallmul_b
1566     \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1567 }%
1568 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1569     {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1570 \def\XINT_div_verysmallmul_e;#1;+#2#3!{1\relax 0000000#2!}%
```

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

```

1571 \def\XINT_div_sub #1#2%
1572 {%
1573     \expandafter\XINT_div_sub_clean
1574     \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1575     1#2;!!;!!;!!\W #1;!!;!!;!!\W
1576 }%
1577 \def\XINT_div_sub_clean #1-#2#3\W
1578 {%
1579     \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1580     {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1581 }%
1582 \def\XINT_div_sub_neg #1\W { -}%
1583 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1584 {%
1585     \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1586 }%
1587 \def\XINT_div_sub_b #1#2#3!#4!%
1588 {%
1589     \xint_gob_til_sc #4\XINT_div_sub_bi ;%
1590     \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1591 }%
1592 \def\XINT_div_sub_c 1#1#2\xint:%
1593 {%
1594     1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1595 }%
1596 \def\XINT_div_sub_d #1#2#3!#4!%
1597 {%
1598     \xint_gob_til_sc #4\XINT_div_sub_di ;%
1599     \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1600 }%
1601 \def\XINT_div_sub_e 1#1#2\xint:%
1602 {%
1603     1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1604 }%
1605 \def\XINT_div_sub_f #1#2#3!#4!%
1606 {%
1607     \xint_gob_til_sc #4\XINT_div_sub_fi ;%
1608     \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1609 }%
1610 \def\XINT_div_sub_g 1#1#2\xint:%
1611 {%
1612     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1613 }%
1614 \def\XINT_div_sub_h #1#2#3!#4!%
1615 {%
1616     \xint_gob_til_sc #4\XINT_div_sub_hi ;%
1617     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1618 }%
1619 \def\XINT_div_sub_i 1#1#2\xint:%
1620 {%
1621     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1622 }%

```

```

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

```

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

```

1654 \def\XINT_sdiv_out #1;!#2!%
1655   {\expandafter
1656   {\romannumeral0\XINT_unsep_cuzsmall
1657   #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
1658   {#2}}%

```

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

```

1659 \def\XINT_smalldivx_a #1\xint:1#2!1#3!%
1660 {%
1661   \expandafter\XINT_smalldivx_b
1662   \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
1663 }%
1664 \def\XINT_smalldivx_b #1#2!%

```

```

1665 {%
1666     \if0#1\else
1667         \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1668     \XINT_smallldiv_c #1#2!%
1669}%
1670 \def\XINT_smallldiv_c #1!#2\xint:#3!#4!%
1671 {%
1672     \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
1673}%

```

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

```

1674 \def\XINT_smallldiv_d #1!#2!1#3#4!%
1675 {%
1676     \xint_gob_til_sc #3\XINT_smallldiv_end;%
1677     \XINT_smallldiv_e #1!#2!1#3#4!%
1678}%
1679 \def\XINT_smallldiv_end;\XINT_smallldiv_e #1!#2!1;!{1!;!#1!}%

```

Il est crucial que le reste #1 est < #3. J'ai documenté cette routine dans le fichier où j'ai préparé 1.2, il faudra transférer ici. Il n'est pas nécessaire pour cette routine que le diviseur B ait au moins 8 chiffres. Mais il doit être < 10⁸.

```

1680 \def\XINT_smallldiv_e #1!#2\xint:#3!%
1681 {%
1682     \expandafter\XINT_smallldiv_f\the\numexpr
1683     \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2\xint:#3!#1!%
1684}%
1685 \def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
1686 {%
1687     \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1688     \expandafter\XINT_smallldiv_g
1689     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
1690}%
1691 \def\XINT_smallldiv_fz 0%
1692     \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
1693     9999\xint:9999!#1!99999999!#2!0!1#3!%
1694 {%
1695     \XINT_smallldiv_i \xint:#3!\xint_c!#2!%
1696}%
1697 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1698 {%
1699     \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1700}%
1701 \def\XINT_smallldiv_h 1#1#2\xint:#3!#4!%
1702 {%
1703     \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1704}%
1705 \def\XINT_smallldiv_i #1\xint:#2!#3!#4\xint:#5!%
1706 {%
1707     \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4\xint:#5!%
1708}%

```

```

1709 \def\XINT_smallldiv_j #1!#2!%
1710 {%
1711     \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1712     #1!%
1713 }%

```

On boucle vers \XINT_smallldiv_d.

```

1714 \def\XINT_smallldiv_k #1!#2!#3\xint:#4!%
1715 {%
1716     \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
1717 }%

```

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

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

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

```

1718 \def\XINT_div_mini #1\xint:#2!#3!%
1719 {%
1720     \expandafter\XINT_div_mini_a\the\numexpr
1721     \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
1722 }%

```

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

```

1723 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7\xint:#8!%
1724 {%
1725     \xint_gob_til_zero #1\XINT_div_mini_w 0%
1726     \expandafter\XINT_div_mini_b
1727     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#7!#2#3#4#5#6!#7\xint:#8!%
1728 }%
1729 \def\XINT_div_mini_w 0%
1730     \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1731     9999\xint:9999!#1!99999999!#2\xint:#3!00000000!#4!%
1732 {%
1733     \xint_c_x^viii_mone+(#4+#3)/#2!%
1734 }%
1735 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1736 {%
1737     \expandafter\XINT_div_mini_c
1738     \the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1739 }%
1740 \def\XINT_div_mini_c 1#1#2\xint:#3!#4!%
1741 {%
1742     \expandafter\XINT_div_mini_d
1743     \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%

```

```

1744 }%
1745 \def\XINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
1746 {%
1747   \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1748 }%

```

Derived arithmetic

4.38 \xintiiQuo, \xintiiRem

```

1749 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1750 \def\xintiiRem {\romannumeral0\xintiirem }%
1751 \def\xintiiquo
1752   {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1753 \def\xintiirem
1754   {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%

```

4.39 \xintiiDivRound

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

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

```

1755 \def\xintiiDivRound {\romannumeral0\xintiividivround }%
1756 \def\xintiividivround #1{\expandafter\XINT_iidivround\romannumeral`&&@#1\xint:}%
1757 \def\XINT_iidivround #1#2\xint:#3%
1758   {\expandafter\XINT_iidivround_a\expandafter #1%
1759     \romannumeral0\xintnum{#3}\xint:#2\xint:}%
1760 \def\XINT_iidivround #1#2\xint:#3%
1761   {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\xint:#2\xint:}%
1762 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
1763 {%
1764   \if0#2\xint_dothis{\XINT_iidivround_divbyzero#1#2}\fi
1765   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1766   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1767   \xint_orthat{\XINT_iidivround_bpos #1#2}%
1768 }%
1769 \def\XINT_iidivround_divbyzero #1#2#3\xint:#4\xint:
1770   {\XINT_signalcondition{DivisionByZero}{Division of #1#4 by #2#3}{0}}%
1771 \def\XINT_iidivround_aiszero #1\xint:#2\xint:{ 0}%
1772 \def\XINT_iidivround_bpos #1%
1773 {%
1774   \xint_UDsignfork
1775     #1{\xintiiopp\XINT_iidivround_pos {}}%
1776     -{\XINT_iidivround_pos #1}%
1777   \krof
1778 }%
1779 \def\XINT_iidivround_bneg #1%
1780 {%
1781   \xint_UDsignfork
1782     #1{\XINT_iidivround_pos {}}%
1783     -{\xintiiopp\XINT_iidivround_pos #1}%
1784   \krof
1785 }%

```

```

1786 \def\XINT_iidivround_pos #1#2\xint:#3\xint:
1787 {%
1788   \expandafter\expandafter\expandafter\XINT_dsrr
1789   \expandafter\xint_firstoftwo
1790   \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1791   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1792 }%

```

4.40 \xintiDivTrunc

1.21: \xintiDivTrunc made robust against non terminated input.

```

1793 \def\xintiDivTrunc {\romannumeral0\xintiDivTrunc }%
1794 \def\xintiDivTrunc #1{\expandafter\XINT_iidivtrunc\romannumeral`&&@#1\xint:}%
1795 \def\XINT_iidivtrunc #1#2\xint:#3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1796   \romannumeral`&&@#3\xint:#2\xint:}%
1797 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1798 {%
1799   \if0#2\xint_dothis{\XINT_iidivtrunc_divbyzero#1#2}\fi
1800   \if0#1\xint_dothis\XINT_iidivtrunc_aiszero\fi
1801   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1802   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1803 }%

```

Attention to not move DivRound code beyond that point.

```

1804 \let\XINT_iidivtrunc_divbyzero\XINT_iidivround_divbyzero
1805 \let\XINT_iidivtrunc_aiszero \XINT_iidivround_aiszero
1806 \def\XINT_iidivtrunc_bpos #1%
1807 {%
1808   \xint_UDsignfork
1809   #1{\xintiopp\XINT_iidivtrunc_pos {}}%
1810   -{\XINT_iidivtrunc_pos #1}%
1811   \krof
1812 }%
1813 \def\XINT_iidivtrunc_bneg #1%
1814 {%
1815   \xint_UDsignfork
1816   #1{\XINT_iidivtrunc_pos {}}%
1817   -{\xintiopp\XINT_iidivtrunc_pos #1}%
1818   \krof
1819 }%
1820 \def\XINT_iidivtrunc_pos #1#2\xint:#3\xint:
1821   {\expandafter\xint_firstoftwo_thenstop
1822   \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

4.41 \xintiModTrunc

Renamed from \xintiMod to \xintiModTrunc at 1.2p.

```

1823 \def\xintiModTrunc {\romannumeral0\xintiModTrunc }%
1824 \def\xintiModTrunc #1{\expandafter\XINT_iimodtrunc\romannumeral`&&@#1\xint:}%
1825 \def\XINT_iimodtrunc #1#2\xint:#3{\expandafter\XINT_iimodtrunc_a\expandafter #1%

```

```

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

```

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

```

1834 \let\XINT_iimodtrunc_divbyzero\XINT_iidivround_divbyzero
1835 \let\XINT_iimodtrunc_aiszero \XINT_iidivround_aiszero
1836 \def\XINT_iimodtrunc_bpos #1%
1837 {%
1838 \xint_UDsignfork
1839 #1{\xintiioopp\XINT_iimodtrunc_pos {}}%
1840 -{\XINT_iimodtrunc_pos #1}%
1841 \krof
1842 }%
1843 \def\XINT_iimodtrunc_bneg #1%
1844 {%
1845 \xint_UDsignfork
1846 #1{\xintiioopp\XINT_iimodtrunc_pos {}}%
1847 -{\XINT_iimodtrunc_pos #1}%
1848 \krof
1849 }%
1850 \def\XINT_iimodtrunc_pos #1#2\xint:#3\xint:
1851 {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
1852 {#2}{#1#3}}%

```

4.42 \xintiDivMod

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

```

1853 \def\xintiDivMod {\romannumeral0\xintiDivMod }%
1854 \def\xintiDivMod #1{\expandafter\XINT_iidivmod\romannumeral`&&@#1\xint:}%
1855 \def\XINT_iidivmod #1#2\xint:#3{\expandafter\XINT_iidivmod_a\expandafter #1%
1856 \romannumeral`&&@#3\xint:#2\xint:}%
1857 \def\XINT_iidivmod_a #1#2% #1 de A, #2 de B.
1858 {%
1859 \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
1860 \if0#1\xint_dothis\XINT_iidivmod_aiszero\fi
1861 \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
1862 \xint_orthat{\XINT_iidivmod_bpos #1#2}%
1863 }%
1864 \def\XINT_iidivmod_divbyzero #1#2\xint:#3\xint:
1865 {%
1866 \XINT_signalcondition{DivisionByZero}{Division by #2 of #1#3}{}%
1867 {{0}{0}}% à revoir...
1868 }%
1869 \def\XINT_iidivmod_aiszero #1#2\xint:#3\xint:{{0}{0}}%

```



```

1870 \def\XINT_iidivmod_bneg #1%
1871 {%
1872   \expandafter\XINT_iidivmod_bneg_finish
1873   \romannumeral0\xint_UDsignfork
1874       #1{\XINT_iidivmod_bpos {}}%
1875       -{\XINT_iidivmod_bpos {-#1}}%
1876   \krof
1877 }%
1878 \def\XINT_iidivmod_bneg_finish#1#2%
1879 {%
1880   \expandafter\xint_exchangetwo_keepbraces\expandafter
1881   {\romannumeral0\xintiiopt#2}{#1}%
1882 }%
1883 \def\XINT_iidivmod_bpos #1#2\xint:#3\xint:{\xintiividivision{#1#3}{#2}}%

```

4.43 \xintiiDivFloor

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

```

1884 \def\xintiiDivFloor {\romannumeral0\xintiividivfloor}%
1885 \def\xintiividivfloor {\expandafter\xint_firstoftwo_thenstop
1886   \romannumeral0\xintiividivmod}%

```

4.44 \xintiiMod

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

```

1887 \def\xintiiMod {\romannumeral0\xintiimod}%
1888 \def\xintiimod {\expandafter\xint_secondoftwo_thenstop
1889   \romannumeral0\xintiividivmod}%

```

4.45 \xintiiSqr

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

```

1890 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1891 \def\xintiisqr #1%
1892 {%
1893   \expandafter\XINT_sqr\romannumeral0\xintiiaabs{#1}\xint:
1894 }%
1895 \def\XINT_sqr #1\xint:
1896 {%
1897   \expandafter\XINT_sqr_a
1898   \romannumeral0\expandafter\XINT_sepandrev_andcount
1899   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
1900   #1\XINT_rsepybviii_end_A 2345678%
1901   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1902   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1903   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1904   \xint:
1905 }%

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

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

```

1906 \def\XINT_sqr_a #1\xint:
1907 {%
1908     \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1909         \else\expandafter\XINT_sqr_start\fi
1910 }%
1911 \def\XINT_sqr_small 1#1#2#3#4#5!\xint:
1912 {%
1913     \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1914     \expandafter\XINT_sqr_small_out
1915     \the\numexpr\XINT_minimul_a #1#2#3#4\xint:#5!#1#2#3#4#5!%
1916 }%
1917 \def\XINT_sqr_verysmall#1{%
1918 \def\XINT_sqr_verysmall
1919     \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a ##1!##2!%
1920     {\expandafter#1\the\numexpr ##2*##2\relax}%
1921 }\XINT_sqr_verysmall{ }%
1922 \def\XINT_sqr_small_out 1#1!1#2!%
1923 {%
1924     \XINT_cuz #2#1\R
1925 }%

```

An ending `1;!` is produced on output for `\XINT_mul_loop` and gets incorporated to the delimiter needed by the `\XINT_unrevbyviii` done by `\XINT_mul_out`.

```

1926 \def\XINT_sqr_start #1\xint:
1927 {%
1928     \expandafter\XINT_mul_out
1929     \the\numexpr\XINT_mul_loop
1930         1000000000!1;!\W #11;!\W #11;!%
1931     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1932 }%

```

4.46 `\xintiiPow`

The exponent is not limited but with current default settings of tex memory, with xint 1.2, the maximal exponent for 2^N is $N = 2^{17} = 131072$.

1.2f Modifies the initial steps: 1) in order to be able to let more easily `\xintiPow` use `\xintNum` on the exponent once `xintfrac.sty` is loaded; 2) also because I noticed it was not very well coded. And it did only a `\numexpr` on the exponent, contradicting the documentation related to the "i" convention in names.

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

```

1933 \def\xintiiPow {\romannumeral0\xintiipow }%
1934 \def\xintiipow #1#2%
1935 {%
1936     \expandafter\xint_pow\the\numexpr #2\expandafter
1937     .\romannumeral`&&@#1\xint:
1938 }%
1939 \def\xint_pow #1.#2%#3\xint:
1940 {%

```

```

1941 \xint_UDzerominusfork
1942 #2-\XINT_pow_AisZero
1943 0#2\XINT_pow_Aneg
1944 0-{\XINT_pow_Apos #2}%
1945 \krof {#1}%
1946 }%
1947 \def\XINT_pow_AisZero #1#2\xint:
1948 {%
1949 \ifcase\XINT_cntSgn #1\xint:
1950 \xint_afterfi { 1}%
1951 \or
1952 \xint_afterfi { 0}%
1953 \else
1954 \xint_afterfi
1955 {\XINT_signalcondition{DivisionByZero}{Zero to power #1}{0}}%
1956 \fi
1957 }%
1958 \def\XINT_pow_Aneg #1%
1959 {%
1960 \ifodd #1
1961 \expandafter\XINT_opp\romannumeral0%
1962 \fi
1963 \XINT_pow_Apos {}{#1}%
1964 }%
1965 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1966 \def\XINT_pow_Apos_a #1#2#3%
1967 {%
1968 \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
1969 \XINT_pow_AatleastTwo {#1}#2#3%
1970 }%
1971 \def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
1972 {%
1973 \ifcase #2
1974 \xintError:thiscannothappen
1975 \or \expandafter\XINT_pow_AisOne
1976 \else\expandafter\XINT_pow_AatleastTwo
1977 \fi {#1}#2\xint:
1978 }%
1979 \def\XINT_pow_AisOne #1\xint:{ 1}%
1980 \def\XINT_pow_AatleastTwo #1%
1981 {%
1982 \ifcase\XINT_cntSgn #1\xint:
1983 \expandafter\XINT_pow_BisZero
1984 \or
1985 \expandafter\XINT_pow_I_in
1986 \else
1987 \expandafter\XINT_pow_BisNegative
1988 \fi
1989 {#1}%
1990 }%
1991 \def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}{Inverse power
1992 can not be represented by an integer}}{0}}%

```

```
1993 \def\XINT_pow_BisZero #1\xint:{ 1}%
```

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

```
1994 \def\XINT_pow_I_in #1#2\xint:
```

```
1995 {%
```

```
1996   \expandafter\XINT_pow_I_loop
```

```
1997   \the\numexpr #1\expandafter\xint:%
```

```
1998   \romannumeral0\expandafter\XINT_sepandrev
```

```
1999   \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\{10}0000001\W
```

```
2000   #2\XINT_rsepbyviii_end_A 2345678%
```

```
2001   \XINT_rsepbyviii_end_B 2345678\relax XX%
```

```
2002   \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
```

```
2003   1;! \W
```

```
2004   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
```

```
2005 }%
```

```
2006 \def\XINT_pow_I_loop #1\xint:%
```

```
2007 {%
```

```
2008   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
```

```
2009   \ifodd #1
```

```
2010     \expandafter\XINT_pow_II_in
```

```
2011   \else
```

```
2012     \expandafter\XINT_pow_I_squareit
```

```
2013   \fi #1\xint:%
```

```
2014 }%
```

```
2015 \def\XINT_pow_I_exit \ifodd #1\fi #2\xint:#3\W {\XINT_mul_out #3}%
```

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

```
2016 \def\XINT_pow_I_squareit #1\xint:#2\W%
```

```
2017 {%
```

```
2018   \expandafter\XINT_pow_I_loop
```

```
2019   \the\numexpr #1/\xint_c_ii\expandafter\xint:%
```

```
2020   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
```

```
2021 }%
```

```
2022 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
```

```
2023 {%
```

```
2024   \xint_gob_til_sc #2\XINT_pow_mul_small;%
```

```
2025   \XINT_mul_loop 100000000!1;! \W #1!1#2%
```

```
2026 }%
```

```
2027 \def\XINT_pow_mul_small;\XINT_mul_loop
```

```
2028   100000000!1;! \W 1#1!1;! \W
```

```
2029 {%
```

```
2030   \XINT_smallmul 1#1!%
```

```
2031 }%
```

```
2032 \def\XINT_pow_II_in #1\xint:#2\W
```

```
2033 {%
```

```
2034   \expandafter\XINT_pow_II_loop
```

```
2035   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
```

```
2036   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
```

```
2037 }%
```

```

2038 \def\XINT_pow_II_loop #1\xint:%
2039 {%
2040     \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
2041     \ifodd #1
2042         \expandafter\XINT_pow_II_odda
2043     \else
2044         \expandafter\XINT_pow_II_even
2045     \fi #1\xint:%
2046 }%
2047 \def\XINT_pow_II_exit\ifodd #1\fi #2\xint:#3\W #4\W
2048 {%
2049     \expandafter\XINT_mul_out
2050     \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
2051 }%
2052 \def\XINT_pow_II_even #1\xint:#2\W
2053 {%
2054     \expandafter\XINT_pow_II_loop
2055     \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2056     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2057 }%
2058 \def\XINT_pow_II_odda #1\xint:#2\W #3\W
2059 {%
2060     \expandafter\XINT_pow_II_oddb
2061     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2062     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
2063 }%
2064 \def\XINT_pow_II_oddb #1\xint:#2\W #3\W
2065 {%
2066     \expandafter\XINT_pow_II_loop
2067     \the\numexpr #1\expandafter\xint:%
2068     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
2069 }%

```

4.47 \xintiFac

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

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

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

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

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

```

\catcode`_ 11
\catcode`^ 11
\long\def\xint_firstofthree #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree #1#2#3{#3}%
% quickly written factorial using binary split recursive method
\def\tFac {\romannumeral-`0\tfac }%

```



```

2090 }%
2091 \def\XINT_fac_toobig #1.#2\W{\XINT_signalcondition{InvalidOperation}{Factorial
2092   of too big argument: #1 > 10000}}{0}}%
2093 \def\XINT_fac_bigloop_a #1.%
2094 {%
2095   \expandafter\XINT_fac_bigloop_b \the\numexpr
2096   #1+\xint_c_i-\xint_c_ii*(#1-464)/\xint_c_ii).#1.%
2097 }%
2098 \def\XINT_fac_bigloop_b #1.#2.%
2099 {%
2100   \expandafter\XINT_fac_medloop_a
2101   \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
2102 }%
2103 \def\XINT_fac_bigloop_loop #1.#2.%
2104 {%
2105   \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
2106   \expandafter\XINT_fac_bigloop_loop
2107   \the\numexpr #1+\xint_c_ii\expandafter.%
2108   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
2109 }%
2110 \def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
2111 \def\XINT_fac_bigloop_mul #1!%
2112 {%
2113   \expandafter\XINT_smallmul
2114   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2115 }%
2116 \def\XINT_fac_medloop_a #1.%
2117 {%
2118   \expandafter\XINT_fac_medloop_b
2119   \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2120 }%
2121 \def\XINT_fac_medloop_b #1.#2.%
2122 {%
2123   \expandafter\XINT_fac_smallloop_a
2124   \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
2125 }%
2126 \def\XINT_fac_medloop_loop #1.#2.%
2127 {%
2128   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2129   \expandafter\XINT_fac_medloop_loop
2130   \the\numexpr #1+\xint_c_iii\expandafter.%
2131   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
2132 }%
2133 \def\XINT_fac_medloop_mul #1!%
2134 {%
2135   \expandafter\XINT_smallmul
2136   \the\numexpr
2137   \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2138 }%
2139 \def\XINT_fac_smallloop_a #1.%
2140 {%
2141   \csname

```

```

2142      \XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2143      \endcsname #1.%
2144 }%
2145 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
2146 {%
2147      \XINT_fac_smallloop_loop 2.#1.1000000001!1;!%
2148 }%
2149 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2150 {%
2151      \XINT_fac_smallloop_loop 3.#1.1000000002!1;!%
2152 }%
2153 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2154 {%
2155      \XINT_fac_smallloop_loop 4.#1.1000000006!1;!%
2156 }%
2157 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2158 {%
2159      \XINT_fac_smallloop_loop 5.#1.10000000024!1;!%
2160 }%
2161 \def\XINT_fac_smallloop_loop #1.#2.%
2162 {%
2163      \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2164      \expandafter\XINT_fac_smallloop_loop
2165      \the\numexpr #1+\xint_c_iv\expandafter.%
2166      \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2167 }%
2168 \def\XINT_fac_smallloop_mul #1!%
2169 {%
2170      \expandafter\XINT_smallmul
2171      \the\numexpr
2172          \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2173 }%
2174 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%

```

4.48 \XINT_useiimessage

1.2o

```

2175 \def\XINT_useiimessage #1% used in LaTeX only
2176 {%
2177      \XINT_ifFlagRaised {#1}%
2178      {\@backslashchar#1
2179       (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
2180      }%
2181 }%
2182 \XINT_restorecatcodes_endinput%

```


5 Package [xint](#) implementation

.1	Package identification	114	.31	\xintiiifGt	125
.2	More token management	114	.32	\xintiiifLt	126
.3	(WIP) A constant needed by \xintRan- domDigits et al.	114	.33	\xintiiifZero	126
.4	\xintLen, \xintiLen	115	.34	\xintiiifNotZero	126
.5	\xintReverseDigits	115	.35	\xintiiifOne	126
.6	\xintiiE	116	.36	\xintiiifOdd	127
.7	\xintDecSplit	116	.37	\xintifTrueAelseB, \xintifFalseAelseB	127
.8	\xintDecSplitL	118	.38	\xintIsTrue, \xintIsFalse	127
.9	\xintDecSplitR	118	.39	\xintNOT	127
.10	\xintDSHr	119	.40	\xintAND, \xintOR, \xintXOR	127
.11	\xintDSH	119	.41	\xintANDof	128
.12	\xintDSx	120	.42	\xintORof	128
.13	\xintiiEq	121	.43	\xintXORof	128
.14	\xintiiNotEq	121	.44	\xintiiMax	129
.15	\xintiiGeq	122	.45	\xintiiMin	130
.16	\xintiiGt	122	.46	\xintiiMaxof	131
.17	\xintiiLt	122	.47	\xintiiMinof	131
.18	\xintiiGtorEq	122	.48	\xintiiSum	131
.19	\xintiiLtorEq	123	.49	\xintiiPrd	132
.20	\xintiiIsZero	123	.50	\xintiiSquareRoot	132
.21	\xintiiIsNotZero	123	.51	\xintiiSqrt, \xintiiSqrtR	139
.22	\xintiiIsOne	123	.52	\xintiiBinomial	139
.23	\xintiiOdd	123	.53	\xintiiPFactorial	145
.24	\xintiiEven	124	.54	\xintBool, \xintToggle	148
.25	\xintiiMON	124	.55	(WIP) \xintRandomDigits	148
.26	\xintiiMMON	124	.56	(WIP) \XINT_eightrandomdigits	149
.27	\xintSgnFork	124	.57	(WIP) \xintXRandomDigits	149
.28	\xintiiifSgn	125	.58	(WIP) \xintiiRandRangeAtoB	150
.29	\xintiiifCmp	125	.59	(WIP) \xintiiRandRange	150
.30	\xintiiifEq	125	.60	Adjustments for engines without uniformde- viate primitive	152

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

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

1.3b adds randomness related macros.

```

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

```

```

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

```

5.1 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xint}%
46 [2018/06/17 1.3c Expandable operations on big integers (JFB)]%

```

5.2 More token management

```

47 \long\def\xint_firstofthree #1#2#3{#1}%
48 \long\def\xint_secondofthree #1#2#3{#2}%
49 \long\def\xint_thirdofthree #1#2#3{#3}%
50 \long\def\xint_firstofthree_thenstop #1#2#3{ #1}% 1.09i
51 \long\def\xint_secondofthree_thenstop #1#2#3{ #2}%
52 \long\def\xint_thirdofthree_thenstop #1#2#3{ #3}%

```

5.3 (WIP) A constant needed by \xintRandomDigits et al.

```

53 \ifdefined\xint_texuniformdeviate
54   \unless\ifdefined\xint_c_nine_x^viii
55     \csname newcount\endcsname\xint_c_nine_x^viii
56     \xint_c_nine_x^viii 900000000
57   \fi
58 \fi

```

5.4 \xintLen, \xintiLen

\xintLen gets extended to fractions by xintfrac.sty: A/B is given length len(A)+len(B)-1 (somewhat arbitrary). It applies \xintNum to its argument. A minus sign is accepted and ignored.

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

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

```

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

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

5.5 \xintReverseDigits

1.2.

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

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

This macro is currently not used elsewhere in xint code.

```

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

```

93 \expandafter\XINT_revdigits_b\expandafter{\expandafter}%
94 \the\numexpr\XINT_microrevsep
95 }%
96 \def\XINT_microrevsep #1#2#3#4#5#6#7#8#9%
97 {%
98 1#9#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
99 }%
100 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{\relax#2!}%
101 \def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
102 {%
103 \xint_gob_til_R #9\XINT_revdigits_end\R
104 \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
105 }%
106 \def\XINT_revdigits_end#1{%
107 \def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
108 {\expandafter#1\xint_gob_til_Z ##1}%
109 }\XINT_revdigits_end{ }%
110 \let\xintRev\xintReverseDigits

```

5.6 \xintiiE

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

```

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

```

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

```

121 \def\XINT_ii_a #1.%
122 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
123 \def\XINT_ii_neg #1.#2;{ #2}%

```

5.7 \xintDecSplit

DECIMAL SPLIT

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

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

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

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

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

```

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

133 \def\XINT_split_xfork #1%
134 {%
135   \xint_UDzerominusfork
136   #1-\XINT_split_zerosplit
137   0#1\XINT_split_fromleft
138   0-\XINT_split_fromright #1}%
139   \krof
140 }%
141 \def\XINT_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
142 \def\XINT_split_fromleft
143   {\expandafter\XINT_split_fromleft_a\the\numexpr\xint_c_viii-}%
144 \def\XINT_split_fromleft_a #1%
145 {%
146   \xint_UDsignfork
147   #1\XINT_split_fromleft_b
148   -{\XINT_split_fromleft_end_a #1}%
149   \krof
150 }%
151 \def\XINT_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
152 {%
153   \expandafter\XINT_split_fromleft_clean
154   \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
155   \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
156 }%
157 \def\XINT_split_fromleft_end_a #1.%
158 {%
159   \expandafter\XINT_split_fromleft_clean
160   \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
161 }%
162 \def\XINT_split_fromleft_clean 1{ }%
163 \expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
164   {#1\XINT_split_fromleft_end_b}%
165 \expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
166   {#1#2\XINT_split_fromleft_end_b}%
167 \expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
168   {#1#2#3\XINT_split_fromleft_end_b}%
169 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%

```

```

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

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

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

```

5.8 \xintDecSplitL

```

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

```

5.9 \xintDecSplitR

```

206 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
207 \def\xintdecsplitr #1#2%
208 {%
209     \expandafter\XINT_splitr_finish
210     \romannumeral0\expandafter\XINT_split_xfork
211     \the\numexpr #1\expandafter.\romannumeral`&&@#2%
212     \xint_bye2345678\xint_bye..%
213 }%
214 \def\XINT_splitr_finish #1.#2.{ #2}%

```

5.10 \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}
 si $x \leq 0$, fait $A \rightarrow A \cdot 10^{|x|}$. si $x > 0$, et $A \geq 0$, fait $A \rightarrow \text{quo}(A, 10^x)$
 si $x > 0$, et $A < 0$, fait $A \rightarrow -\text{quo}(-A, 10^x)$
 (donc pour $x > 0$ c'est comme DSR itéré x fois)
 \xintDSHr donne le 'reste' (si $x \leq 0$ donne zéro).
 Badly named macros.
 Rewritten for 1.2i, this was old code and \xintDSx has changed interface.

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

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

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

231     \expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
232 }%
```

5.11 \xintDSH

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

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

242     0#1\XINT_dsx_xisNeg_checkA
243     0-{ \XINT_dsh_xisPos #1}%
244     \krof
245 }%
246 \def\XINT_dsh_xiszero #1.#2;{ #2}%
247 \def\XINT_dsh_xisPos
```

248 {%

\expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_dsx_xisPos

249 }%

5.12 \xintDSx

--> Attention le cas $x=0$ est traité dans la même catégorie que $x > 0$ <--

si $x < 0$, fait $A \rightarrow A \cdot 10^{|x|}$

si $x \geq 0$, et $A \geq 0$, fait $A \rightarrow \{ \text{quo}(A, 10^x) \} \{ \text{rem}(A, 10^x) \}$

si $x \geq 0$, et $A < 0$, d'abord on calcule $\{ \text{quo}(-A, 10^x) \} \{ \text{rem}(-A, 10^x) \}$

puis, si le premier n'est pas nul on lui donne le signe -

si le premier est nul on donne le signe - au second.

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

Rewritten for 1.2i, this was old code.

250 \def\xintDSx {\romannumeral0\xintdsx }%

251 \def\xintdsx #1#2%

252 {%

253 \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%

254 }%

255 \def\XINT_dsx_fork #1%

256 {%

257 \xint_UDzerominusfork

258 #1-\XINT_dsx_xisZero

259 0#1\XINT_dsx_xisNeg_checkA

260 0-\XINT_dsx_xisPos #1}%

261 \krof

262 }%

263 \def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%

264 \def\XINT_dsx_xisNeg_checkA #1.#2%

265 {%

266 \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%

267 \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%

268 }%

269 \def\XINT_dsx_xisNeg_Azero #1;{ 0}%

270 \def\XINT_dsx_addzeros #1%

271 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%

272 \def\XINT_dsx_addzerosnofuss #1%

273 {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%

274 \def\XINT_dsx_append #1.#2;{ #2#1}%

275 \def\XINT_dsx_xisPos #1.#2%

276 {%

277 \xint_UDzerominusfork

278 #2-\XINT_dsx_AisZero


```

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

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

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

309 \def\XINT_dsx_AisPos_finish #1.#2.%
310 {%
311      \expandafter\XINT_dsx_end
312      \expandafter {\romannumeral0\XINT_num {#2}}%
313      {\romannumeral0\XINT_num {#1}}%
314 }%
315 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%

```

5.13 \xintiiEq

no \xintiieq.

```

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

```

5.14 \xintiiNotEq

Pour xintexpr. Pas de version en lowercase.

```

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

```

5.15 \xintiiGeq

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

1.2l made \xintiiGeq robust against non terminated items.

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

This macro should have been called \xintGEq for example.

```

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

324 \def\XINT_geq #1#2\xint:#3%
325 {%
326   \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
327 }%
328 \def\XINT_geq_fork #1#2%
329 {%
330   \xint_UDzerofork
331   #1\XINT_geq_firstiszero
332   #2\XINT_geq_secondiszero
333   0{ }%
334   \krof
335   \xint_UDsignsfork
336   #1#2\XINT_geq_minusminus
337   #1-\XINT_geq_minusplus
338   #2-\XINT_geq_plusminus
339   --\XINT_geq_plusplus
340   \krof #1#2%
341 }%
342 \def\XINT_geq_firstiszero #1\krof 0#2#3\xint:#4\xint:
343   {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
344 \def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%
345 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{ }}%
346 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus { }#1}%
347 \def\XINT_geq_minusminus --{\XINT_geq_plusplus { }{ }}%
348 \def\XINT_geq_plusplus
349   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
350 \def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
351   \else\expandafter\XINT_geq_yes\fi}%
352 \def\XINT_geq_no 1{ 0}%
353 \def\XINT_geq_yes { 1}%

```

5.16 \xintiiGt

```

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

```

5.17 \xintiiLt

```

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

```

5.18 \xintiiGtorEq

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfraction, xintexpr, indices

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

5.19 \xintiILtorEq

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

5.20 \xintiiIsZero

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

```
358 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
```

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

5.21 \xintiiIsNotZero

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

```
360 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
```

```
361 \def\xintiiisnotzero
```

```
362      #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

5.22 \xintiiIsOne

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

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

```
363 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
```

```
364 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
```

```
365 \def\XINT_isone #1#2#3Y%
```

```
366 {%
```

```
367   \unless\if#2X\xint_dothis{ 0}\fi
```

```
368   \unless\if#11\xint_dothis{ 0}\fi
```

```
369   \xint_orthat{ 1}%
```

```
370 }%
```

```
371 \def\XINT_isOne #1{\XINT_is_One#1XY}%
```

```
372 \def\XINT_is_One #1#2#3Y%
```

```
373 {%
```

```
374   \unless\if#2X\xint_dothis0\fi
```

```
375   \unless\if#11\xint_dothis0\fi
```

```
376   \xint_orthat1%
```

```
377 }%
```

5.23 \xintiiOdd

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

```
378 \def\xintiiOdd {\romannumeral0\xintiiiodd }%
```

```
379 \def\xintiiiodd #1%
```

```
380 {%
```

```
381   \ifodd\xintLDg{#1} %<- intentional space
```

```
382   \xint_afterfi{ 1}%
```

```
383   \else
```

```
384   \xint_afterfi{ 0}%
```

```
385   \fi
```

```
386 }%
```

5.24 \xintiEven

```

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

5.25 \xintiMON

MINUS ONE TO THE POWER N

```

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

5.26 \xintiMMON

MINUS ONE TO THE POWER N-1

```

405 \def\xintiMMON {\romannumeral0\xintiimmon }%
406 \def\xintiimmon #1%
407 {%
408   \ifodd\xintLDg {#1} %<- intentional space
409   \xint_afterfi{ 1}%
410   \else
411   \xint_afterfi{ -1}%
412   \fi
413 }%
```

5.27 \xintSgnFork

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

```

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

5.28 \xintiiifSgn

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

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

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

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

5.29 \xintiiifCmp

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

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

5.30 \xintiiifEq

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

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

5.31 \xintiiifGt

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

```
448 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
449 \def\xintiiifgt #1#2%
450 {%
451     \if1\xintiiCmp{#1}{#2}%
```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```
452             \expandafter\xint_firstoftwo_thenstop
453         \else\expandafter\xint_secondoftwo_thenstop
454     \fi
455 }%
```

5.32 \xintiiifLt

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

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

5.33 \xintiiifZero

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

1.2o deprecates \xintifZero.

```
464 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
465 \def\xintiiifzero #1%
466 {%
467     \if0\xintiiSgn{#1}%
468         \expandafter\xint_firstoftwo_thenstop
469     \else
470         \expandafter\xint_secondoftwo_thenstop
471     \fi
472 }%
```

5.34 \xintiiifNotZero

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

5.35 \xintiiifOne

added in 1.09i. 1.1a adds \xintiiifOne.

```
482 \def\xintiiifOne {\romannumeral0\xintiiifone }%
483 \def\xintiiifone #1%
484 {%
```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```
485 \if1\xintiiIsOne{#1}%
486 \expandafter\xint_firstoftwo_thenstop
487 \else
488 \expandafter\xint_secondoftwo_thenstop
489 \fi
490 }%
```

5.36 \xintiiifOdd

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

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

5.37 \xintifTrueAelseB, \xintifFalseAelseB

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

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

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

5.38 \xintIsTrue, \xintIsFalse

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

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

5.39 \xintNOT

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

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

5.40 \xintAND, \xintOR, \xintXOR

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

```
505 \def\xintAND {\romannumeral0\xintand}%
```

```

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

```

5.41 \xintANDof

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

1.2l made \xintANDof robust against non terminated items.

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

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

```

516 \def\xintANDof {\romannumeral0\xintandof }%
517 \def\xintandof #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\xint:}%
518 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1!}%
519 \def\XINT_andof_b #1%
520     {\xint_gob_til_xint: #1\XINT_andof_e\xint:\XINT_andof_c #1}%
521 \def\XINT_andof_c #1!%
522     {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
523 \def\XINT_andof_no #1\xint:{ 0}%
524 \def\XINT_andof_e #1!{ 1}%

```

5.42 \xintORof

New with 1.09a. Works also with an empty list. Empty items however are not accepted.

1.2l made \xintORof robust against non terminated items.

```

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

```

5.43 \xintXORof

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

1.2l made \xintXORof robust against non terminated items.

```

534 \def\xintXORof {\romannumeral0\xintxorof }%
535 \def\xintxorof #1{\expandafter\XINT_xorof_a\expandafter
536     0\romannumeral`&&@#1\xint:}%

```


TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfac, xintexpr, indices

```

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

```

5.44 \xintiMax

At 1.2m, a long-standing bug was fixed: \xintiMax had the overhead of applying \xintNum to its arguments due to use of a sub-macro of \xintGeq code to which this overhead was added at some point. And on this occasion I reduced even more number of times input is grabbed.

```

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

```

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

```

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

```

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

```

575 \def\XINT_max_zerozero #1\fi{\xint_firstoftwo_thenstop }%

```

```

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

```

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

```

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

```

5.45 \xintiiMin

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

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

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

```

588 \def\xintiiMin {\romannumeral0\xintiimin}%
589 \def\xintiimin #1%
590 {%
591     \expandafter\xint_iimin \romannumeral`&&@#1\xint:
592 }%
593 \def\xint_iimin #1\xint:#2%
594 {%
595     \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
596 }%
597 \def\XINT_min_fork #1#2\xint:#3#4\xint:
598 {%
599     \xint_UDsignsfork
600         #1#3\XINT_min_minusminus % A < 0, B < 0
601         #1-\XINT_min_plusminus % B < 0, A >= 0
602         #3-\XINT_min_minusplus % A < 0, B >= 0
603         --{\xint_UDzerosfork
604             #1#3\XINT_min_zerzero % A = B = 0
605             #10\XINT_min_pluszero % B = 0, A > 0
606             #30\XINT_min_zeroplus % A = 0, B > 0
607             00\XINT_min_plusplus % A, B > 0
608             \krof}%
609     \krof
610     #3#1#2\xint:#4\xint:
611     \expandafter\xint_secondoftwo_thenstop
612     \else
613     \expandafter\xint_firstoftwo_thenstop
614     \fi

```

```

615      {#3#4}{#1#2}%
616 }%
617 \def\XINT_min_zerozero #1\fi{\xint_firstoftwo_thenstop }%
618 \def\XINT_min_zeroplus #1\fi{\xint_firstoftwo_thenstop }%
619 \def\XINT_min_pluszero #1\fi{\xint_secondoftwo_thenstop }%
620 \def\XINT_min_minusplus #1\fi{\xint_firstoftwo_thenstop }%
621 \def\XINT_min_plusminus #1\fi{\xint_secondoftwo_thenstop }%
622 \def\XINT_min_plusplus
623 {%
624     \if1\romannumeral0\XINT_geq_plusplus
625 }%
626 \def\XINT_min_minusminus --%
627 {%
628     \unless\if1\romannumeral0\XINT_geq_plusplus{}\}%
629 }%

```

5.46 \xintiiMaxof

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

NOT compatible with empty list.

1.2l made \xintiiMaxof robust against non terminated items.

```

630 \def\xintiiMaxof      {\romannumeral0\xintiimaxof }%
631 \def\xintiimaxof      #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\xint:}%
632 \def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1!}%
633 \def\XINT_iimaxof_b #1!#2%
634     {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2!{#1}!}%
635 \def\XINT_iimaxof_c #1%
636     {\xint_gob_til_xint: #1\XINT_iimaxof_e\xint:\XINT_iimaxof_d #1}%
637 \def\XINT_iimaxof_d #1!%
638     {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
639 \def\XINT_iimaxof_e #1!#2!{ #2}%

```

5.47 \xintiiMinof

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

```

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

```

5.48 \xintiiSum

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

```

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

```

5.49 \xintiiPrd

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

```

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

```

5.50 \xintiiSquareRoot

First done with 1.08.

1.1 added `\xintiiSquareRoot`.

1.1a added `\xintiiSqrtR`.

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

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

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

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

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

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

```

670 \def\xintiiSquareRoot {\romannumeral0\xintiiSquareroot }%
671 \def\xintiiSquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint:}%

```

```

672 \def\XINT_sqrt_checkin #1%
673 {%
674     \xint_UDzerominusfork
675     #1-\XINT_sqrt_iszero
676     0#1\XINT_sqrt_isneg
677     0-\XINT_sqrt
678     \krof #1%
679 }%
680 \def\XINT_sqrt_iszero #1\xint:{{1}{1}}%
681 \def\XINT_sqrt_isneg #1\xint:{\XINT_signalcondition{InvalidOperation}{square
682     root of negative: #1}{0}{0}}}%
683 \def\XINT_sqrt #1\xint:
684 {%
685     \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
686 }%
687 \def\XINT_sqrt_start #1.%
688 {%
689     \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
690     \xint_orthat\XINT_sqrt_big_a #1.%
691 }%
692 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
693 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
694 \def\XINT_sqrt_a #1.%
695 {%
696     \ifodd #1
697         \expandafter\XINT_sqrt_b0
698     \else
699         \expandafter\XINT_sqrt_bE
700     \fi
701     #1.%
702 }%

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

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

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

```

```

720         \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
721 }%

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

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

732 \def\XINT_sqrt_small_ea #1%
733 {%
734     \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
735     \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
736     \xint_orthat\XINT_sqrt_small_f #1%
737 }%
738 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
739     \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

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

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

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

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

```

```

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

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

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

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

781 \def\XINT_sqrt_big_eE_a #1.#2;#3%
782 {%
783     \expandafter\XINT_sqrt_bigormed_f
784     \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
785 }%

786 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
787 {%
788     \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
789 }%
790 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
791 {%
792     \expandafter\XINT_sqrt_bigormed_f
793     \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
794 }%

795 \def\XINT_sqrt_bigormed_f #1#2#3;%
796 {%
797     \ifnum#3<\xint_c_ix
798         \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
799     \fi
800     \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
801 }%
802 \def\XINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
803 \def\XINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
804 \def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
805 \def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%

```

```

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

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

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

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

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

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

840 \def\XINT_sqrt_big_gc #1.#2.#3.%
841 {%
842     \expandafter\XINT_sqrt_big_gd
843     \romannumeral0\xintiadd
844     {\xintiiSub {#3000000000}{\xintDouble{\xintiiMul{#2}{#1}}}{000000000}%
845     {\xintiiSqr {#1}}}%
846     \romannumeral0\xintiisub{#2000000000}{#1}.%
847 }%

```



```

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

914 \def\XINT_sqrt_big_kc #1%
915 {%
916     \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
917     \xint_orthat\XINT_sqrt_big_kloop #1%
918 }%
919 \def\XINT_sqrt_big_kz 0.#1.%
920 {%
921     \expandafter\XINT_sqrt_big_kend
922     \romannumeral0%
923     \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
924 }%
925 \def\XINT_sqrt_big_kend #1.#2.%
926 {%
927     \expandafter{\romannumeral0\xintinc{#2}}{#1}%
928 }%

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

```

```

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

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

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

```

5.51 \xintiiSqrt, \xintiiSqrtR

```

953 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
954 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
955 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
956 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
957 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%

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

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

```

5.52 \xintiiBinomial

2015/11/28-29 for 1.2f.

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

I really lack some kind of infinity or NaN value.

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

```

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

```

TOC, [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

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

```

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

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

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

```

983 \def\XINT_binom_a #1.#2.%
984 {%
985   \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.1000000001!1!;!0!%
986 }%
```

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

```

987 \def\XINT_binom_b #1.%
988 {%
989   \ifnum #1>9999 \xint_dothis\XINT_binom_vbigloop \fi
990   \ifnum #1>463 \xint_dothis\XINT_binom_bigloop \fi
991   \ifnum #1>98 \xint_dothis\XINT_binom_medloop \fi
992   \ifnum #1>29 \xint_dothis\XINT_binom_smallloop \fi
993     \xint_orthat\XINT_binom_vsmallloop #1.%
994 }%
```

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

```

995 \def\XINT_binom_smallloop #1.#2.#3.%
996 {%
997   \ifcase\numexpr #3-#2\relax
998     \expandafter\XINT_binom_end_
```

```

999    \or \expandafter\XINT_binom_end_i
1000   \or \expandafter\XINT_binom_end_ii
1001   \or \expandafter\XINT_binom_end_iii
1002   \else\expandafter\XINT_binom_smallloop_a
1003   \fi #1.#2.#3.%
1004 }%

```

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

```

1005 \def\XINT_binom_smallloop_a #1.#2.#3.%
1006 {%
1007   \expandafter\XINT_binom_smallloop_b
1008   \the\numexpr #1+\xint_c_iv\expandafter.%
1009   \the\numexpr #2+\xint_c_iv\expandafter.%
1010   \the\numexpr #3\expandafter.%
1011   \the\numexpr\expandafter\XINT_binom_div
1012   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1013   !\romannumeral0\expandafter\XINT_binom_mul
1014   \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1015 }%
1016 \def\XINT_binom_smallloop_b #1.%
1017 {%
1018   \ifnum #1>98 \expandafter\XINT_binom_medloop \else
1019               \expandafter\XINT_binom_smallloop \fi #1.%
1020 }%

```

Ici on prend trois par trois.

```

1021 \def\XINT_binom_medloop #1.#2.#3.%
1022 {%
1023   \ifcase\numexpr #3-#2\relax
1024     \expandafter\XINT_binom_end_
1025   \or \expandafter\XINT_binom_end_i
1026   \or \expandafter\XINT_binom_end_ii
1027   \else\expandafter\XINT_binom_medloop_a
1028   \fi #1.#2.#3.%
1029 }%
1030 \def\XINT_binom_medloop_a #1.#2.#3.%
1031 {%
1032   \expandafter\XINT_binom_medloop_b
1033   \the\numexpr #1+\xint_c_iii\expandafter.%
1034   \the\numexpr #2+\xint_c_iii\expandafter.%
1035   \the\numexpr #3\expandafter.%
1036   \the\numexpr\expandafter\XINT_binom_div
1037   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1038   !\romannumeral0\expandafter\XINT_binom_mul
1039   \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1040 }%
1041 \def\XINT_binom_medloop_b #1.%
1042 {%
1043   \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1044               \expandafter\XINT_binom_medloop \fi #1.%
1045 }%

```

Ici on prend deux par deux.

```

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

```

Et finalement un par un.

```

1070 \def\XINT_binom_vbigloop #1.#2.#3.%
1071 {%
1072   \ifnum #3=#2
1073     \expandafter\XINT_binom_end_
1074   \else\expandafter\XINT_binom_vbigloop_a
1075   \fi #1.#2.#3.%
1076 }%
1077 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1078 {%
1079   \expandafter\XINT_binom_vbigloop
1080   \the\numexpr #1+\xint_c_i\expandafter.%
1081   \the\numexpr #2+\xint_c_i\expandafter.%
1082   \the\numexpr #3\expandafter.%
1083   \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1084   !\romannumeral0\XINT_binom_mul #1!%
1085 }%

```

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

```

1086 \def\XINT_binom_vsmallloop #1.#2.#3.%
1087 {%

```

```

1088 \ifcase\numexpr #3-#2\relax
1089 \expandafter\XINT_binom_vsmallend_
1090 \or \expandafter\XINT_binom_vsmallend_i
1091 \or \expandafter\XINT_binom_vsmallend_ii
1092 \or \expandafter\XINT_binom_vsmallend_iii
1093 \else\expandafter\XINT_binom_vsmallloop_a
1094 \fi #1.#2.#3.%
1095}%
1096\def\XINT_binom_vsmallloop_a #1.%
1097{%
1098 \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1099 \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1100}%
1101\def\XINT_binom_vsmallloop_b #1.#2.#3.%
1102{%
1103 \expandafter\XINT_binom_vsmallloop
1104 \the\numexpr #1+\xint_c_iv\expandafter.%
1105 \the\numexpr #2+\xint_c_iv\expandafter.%
1106 \the\numexpr #3\expandafter.%
1107 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1108 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1109 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1110}%
1111\def\XINT_binom_mul #1!#21!;!0!%
1112{%
1113 \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1114 \the\numexpr\expandafter\XINT_smallmul
1115 \the\numexpr\xint_c_x^viii+#1\expandafter
1116 !\romannumeral0\XINT_rev_nounsep {}1;!#2%
1117 \R!\R!\R!\R!\R!\R!\R!\R!\W
1118 \R!\R!\R!\R!\R!\R!\R!\R!\W
1119 1;!%
1120}%
1121\def\XINT_binom_div #1!1;!%
1122{%
1123 \expandafter\XINT_smalldivx_a
1124 \the\numexpr #1/\xint_c_ii\expandafter\xint:
1125 \the\numexpr \xint_c_x^viii+#1!%
1126}%

```

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

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

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

```

1128 \def\XINT_binom_end_iii #1.#2.#3.%
1129 {%
1130     \expandafter\XINT_binom_finish
1131     \the\numexpr\expandafter\XINT_binom_div
1132         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter

```

```

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

```

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

```

1160 \def\XINT_binom_vsmallend_iii #1.%
1161 {%
1162     \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1163         \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1164 }%
1165 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1166 {%
1167     \expandafter\XINT_binom_vsmallfinish
1168     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1169     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1170     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1171 }%
1172 \def\XINT_binom_vsmallend_ii #1.%
1173 {%
1174     \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1175         \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1176 }%
1177 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1178 {%
1179     \expandafter\XINT_binom_vsmallfinish
1180     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1181     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter

```



```

1182    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1183 }%
1184 \def\xINT_binom_vsmallend_i #1.%
1185 {%
1186     \ifnum #1>28 \expandafter\xINT_binom_end_i \else
1187                 \expandafter\xINT_binom_vsmallend_ib \fi #1.%
1188 }%
1189 \def\xINT_binom_vsmallend_ib #1.#2.#3.%
1190 {%
1191     \expandafter\xINT_binom_vsmallfinish
1192     \the\numexpr \expandafter\xINT_binom_vsmallmuldiv
1193     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1194     !\the\numexpr #1*(#1+\xint_c_i)!%
1195 }%
1196 \def\xINT_binom_vsmallend_ #1.%
1197 {%
1198     \ifnum #1>29 \expandafter\xINT_binom_end_ \else
1199                 \expandafter\xINT_binom_vsmallend_b \fi #1.%
1200 }%
1201 \def\xINT_binom_vsmallend_b #1.#2.#3.%
1202 {%
1203     \expandafter\xINT_binom_vsmallfinish
1204     \the\numexpr\xINT_binom_vsmallmuldiv #2!#1!%
1205 }%
1206 \def\xINT_binom_vsmallfinish#1{%
1207 \def\xINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
1208 }\xINT_binom_vsmallfinish{ }%

```

5.53 \xintiPFactorial

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

1.2h (2016/11/20) removes the non-negativity condition. It was a bit unfortunate that the code raised `\xintError:OutOfRangePFac` if $0 \leq a \leq b < 10^8$ was violated. The rule now applied is to interpret pfac(a,b) as the product for $a < j \leq b$ (not as a ratio of Gamma function), hence if $a \geq b$, return 1 because of an empty product. If $a < b$: if $a < 0$, return 0 for $b \geq 0$ and $(-1)^{(b-a)}$ times $|b| \dots (|a|-1)$ for $b < 0$. But only for the range $0 \leq a \leq b < 10^8$ is the macro result to be considered as stable.

```

1209 \def\xintiPFactorial {\romannumeral0\xintiipfactorial }%
1210 \def\xintiipfactorial #1#2%
1211 {%
1212     \expandafter\xINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1213 }%
1214 \def\xintPFactorial{\romannumeral0\xintpfactorial}%
1215 \let\xintpfactorial\xintiipfactorial

```

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

```

1216 \def\xINT_pfac_fork #1#2.#3#4.%
1217 {%
1218     \unless\ifnum #1#2<#3#4 \xint_dothis\xINT_pfac_one\fi
1219     \if-#3\xint_dothis\xINT_pfac_neg\fi

```

```

1220 \if-#1\xint_dothis\XINT_pfac_zero\fi
1221 \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_pfac_outofrange\fi
1222 \xint_orthat \XINT_pfac_a #1#2.#3#4.%
1223 }%
1224 \def\XINT_pfac_outofrange #1.#2.%
1225 {\XINT_signalcondition{InvalidOperation}{PFactorial with
1226 too big second arg: 99999999 < #2}}{0}}%
1227 \def\XINT_pfac_one #1.#2.{ 1}%
1228 \def\XINT_pfac_zero #1.#2.{ 0}%
1229 \def\XINT_pfac_neg -#1.-#2.%
1230 {%
1231 \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_pfac_outofrange\fi
1232 \xint_orthat
1233 {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\fi
1234 \expandafter\XINT_pfac_a }%
1235 \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1236 }%
1237 \def\XINT_pfac_a #1.#2.%
1238 {%
1239 \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.100000001!1!;%
1240 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
1241 }%
1242 \def\XINT_pfac_b #1.%
1243 {%
1244 \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1245 \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1246 \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1247 \xint_orthat\XINT_pfac_smallloop #1.%
1248 }%
1249 \def\XINT_pfac_smallloop #1.#2.%
1250 {%
1251 \ifcase\numexpr #2-#1\relax
1252 \expandafter\XINT_pfac_end_
1253 \or \expandafter\XINT_pfac_end_i
1254 \or \expandafter\XINT_pfac_end_ii
1255 \or \expandafter\XINT_pfac_end_iii
1256 \else\expandafter\XINT_pfac_smallloop_a
1257 \fi #1.#2.%
1258 }%
1259 \def\XINT_pfac_smallloop_a #1.#2.%
1260 {%
1261 \expandafter\XINT_pfac_smallloop_b
1262 \the\numexpr #1+\xint_c_iv\expandafter.%
1263 \the\numexpr #2\expandafter.%
1264 \the\numexpr\expandafter\XINT_smallmul
1265 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1266 }%
1267 \def\XINT_pfac_smallloop_b #1.%
1268 {%
1269 \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1270 \expandafter\XINT_pfac_smallloop \fi #1.%
1271 }%

```

```

1272 \def\XINT_pfac_medloop #1.#2.%
1273 {%
1274     \ifcase\numexpr #2-#1\relax
1275         \expandafter\XINT_pfac_end_
1276     \or \expandafter\XINT_pfac_end_i
1277     \or \expandafter\XINT_pfac_end_ii
1278     \else\expandafter\XINT_pfac_medloop_a
1279     \fi #1.#2.%
1280}%
1281 \def\XINT_pfac_medloop_a #1.#2.%
1282 {%
1283     \expandafter\XINT_pfac_medloop_b
1284     \the\numexpr #1+\xint_c_iii\expandafter.%
1285     \the\numexpr #2\expandafter.%
1286     \the\numexpr\expandafter\XINT_smallmul
1287     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1288}%
1289 \def\XINT_pfac_medloop_b #1.%
1290 {%
1291     \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1292         \expandafter\XINT_pfac_medloop \fi #1.%
1293}%
1294 \def\XINT_pfac_bigloop #1.#2.%
1295 {%
1296     \ifcase\numexpr #2-#1\relax
1297         \expandafter\XINT_pfac_end_
1298     \or \expandafter\XINT_pfac_end_i
1299     \else\expandafter\XINT_pfac_bigloop_a
1300     \fi #1.#2.%
1301}%
1302 \def\XINT_pfac_bigloop_a #1.#2.%
1303 {%
1304     \expandafter\XINT_pfac_bigloop_b
1305     \the\numexpr #1+\xint_c_ii\expandafter.%
1306     \the\numexpr #2\expandafter.%
1307     \the\numexpr\expandafter
1308     \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1309}%
1310 \def\XINT_pfac_bigloop_b #1.%
1311 {%
1312     \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1313         \expandafter\XINT_pfac_bigloop \fi #1.%
1314}%
1315 \def\XINT_pfac_vbigloop #1.#2.%
1316 {%
1317     \ifnum #2=#1
1318         \expandafter\XINT_pfac_end_
1319     \else\expandafter\XINT_pfac_vbigloop_a
1320     \fi #1.#2.%
1321}%
1322 \def\XINT_pfac_vbigloop_a #1.#2.%
1323 {%

```

```

1324 \expandafter\XINT_pfac_vbigloop
1325 \the\numexpr #1+\xint_c_i\expandafter.%
1326 \the\numexpr #2\expandafter.%
1327 \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1328 }%
1329 \def\XINT_pfac_end_iii #1.#2.%
1330 {%
1331 \expandafter\XINT_mul_out
1332 \the\numexpr\expandafter\XINT_smallmul
1333 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1334 }%
1335 \def\XINT_pfac_end_ii #1.#2.%
1336 {%
1337 \expandafter\XINT_mul_out
1338 \the\numexpr\expandafter\XINT_smallmul
1339 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1340 }%
1341 \def\XINT_pfac_end_i #1.#2.%
1342 {%
1343 \expandafter\XINT_mul_out
1344 \the\numexpr\expandafter\XINT_smallmul
1345 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1346 }%
1347 \def\XINT_pfac_end_ #1.#2.%
1348 {%
1349 \expandafter\XINT_mul_out
1350 \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1351 }%

```

5.54 \xintBool, \xintToggle

1.09c

```

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

```

5.55 (WIP) \xintRandomDigits

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

```

1355 \def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
1356 \def\xintrandomdigits#1%
1357 {%
1358 \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
1359 }%
1360 \def\XINT_randomdigits#1\xint:
1361 {%
1362 \expandafter\XINT_randomdigits_a
1363 \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1364 }%
1365 \def\XINT_randomdigits_a#1\xint:#2\xint:

```

```

1366 {%
1367   \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
1368   \romannumeral\XINT_replicate #1\endcsname \csname
1369   XINT_rdg\endcsname
1370 }%
1371 \def\XINT_rdg
1372 {%
1373   \expandafter\XINT_rdg_aux\the\numexpr%
1374   \xint_c_nine_x^viii%
1375   -\xint_texuniformdeviate\xint_c_ii^vii%
1376   -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1377   -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1378   -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1379   +\xint_texuniformdeviate\xint_c_x^viii%
1380   \relax%
1381 }%
1382 \def\XINT_rdg_aux#1{\XINT_rdg\endcsname}%
1383 \let\XINT_XINT_rdg\endcsname

```

5.56 (WIP) \XINT_eightrandomdigits

1.3b.

```

1384 \def\XINT_eightrandomdigits
1385 {%
1386   \expandafter\xint_gobble_i\the\numexpr%
1387   \xint_c_nine_x^viii%
1388   -\xint_texuniformdeviate\xint_c_ii^vii%
1389   -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1390   -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1391   -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1392   +\xint_texuniformdeviate\xint_c_x^viii%
1393   \relax%
1394 }%

```

5.57 (WIP) \xintXRandomDigits

1.3b.

```

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

```

1409 }%

5.58 (WIP) \xintiiRandRangeAtoB

1.3b. Support for randrange() function.

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

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

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

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

5.59 (WIP) \xintiiRandRange

1.3b. Support for randrange().

```
1421 \def\xintiiRandRange{\romannumeral`&&\xintiirandrange}%
1422 \def\xintiirandrange#1%
1423 {%
1424     \expandafter\XINT_randrange\romannumeral`&&#1\xint:
1425 }%
1426 \def\XINT_randrange #1%
1427 {%
1428     \xint_UDzerominusfork
1429     #1-\XINT_randrange_err:empty
1430     0#1\XINT_randrange_err:empty
1431     0-\XINT_randrange_a
1432     \krof #1%
1433 }%
1434 \def\XINT_randrange_err:empty#1\xint:
1435 {%
1436     \XINT_expandableerror{Empty range for randrange.} 0%
1437 }%
1438 \def\XINT_randrange_a #1\xint:
1439 {%
1440     \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
1441 }%
1442 \def\XINT_randrange_b #1.%
1443 {%
1444     \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
1445     \xint_orthat{\XINT_randrange_c #1.}%
1446 }%
```

```

1447 \def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
1448 {%
1449   \expandafter\XINT_randrange_d
1450   \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
1451     {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
1452     #2#3#4#5#6#7#8#9\xint:#1\xint:
1453 }%

```

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

```

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

```

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

```

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

```

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

```

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

```

5.60 Adjustments for engines without uniformdeviate primitive

1.3b.

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


6 Package xintbinhex implementation

.1	Catcodes, ε -TeX and reload detection . . .	153	.6	<code>\xintDecToBin</code>	158
.2	Package identification	154	.7	<code>\xintHexToDec</code>	159
.3	Constants, etc...	154	.8	<code>\xintBinToDec</code>	161
.4	Helper macros	155	.9	<code>\xintBinToHex</code>	162
.4.1	<code>\XINT_zeroes_foriv</code>	155	.10	<code>\xintHexToBin</code>	163
.5	<code>\xintDecToHex</code>	155	.11	<code>\xintCHexToBin</code>	163

The commenting is currently (2018/06/17) very sparse.

The macros from 1.08 (2013/06/07) remained unchanged until their complete rewrite at 1.2m (2019/07/31).

At 1.2n dependencies on *xintcore* were removed, so now the package loads only *xintkernel* (this could have been done earlier).

Also at 1.2n, macros evolved again, the main improvements being in the increased allowable sizes of the input for `\xintDecToHex`, `\xintDecToBin`, `\xintBinToHex`. Use of `\csname` governed expansion at some places rather than `\numexpr` with some clean-up after it.

6.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

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

```

6.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintbinhex}%
46 [2018/06/17 1.3c Expandable binary and hexadecimal conversions (JFB)]%

```

6.3 Constants, etc...

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

```

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

```

```

77 \expandafter\def\csname XINT_csbth_1110\endcsname {\endcsname E}%
78 \expandafter\def\csname XINT_csbth_1111\endcsname {\endcsname F}%
79 \let\XINT_csbth_none \endcsname
80 \expandafter\def\csname XINT_cshtb_0\endcsname {\endcsname 0000}%
81 \expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname 0001}%
82 \expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname 0010}%
83 \expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname 0011}%
84 \expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname 0100}%
85 \expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname 0101}%
86 \expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname 0110}%
87 \expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname 0111}%
88 \expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname 1000}%
89 \expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname 1001}%
90 \def\XINT_cshtb_A {\endcsname 1010}%
91 \def\XINT_cshtb_B {\endcsname 1011}%
92 \def\XINT_cshtb_C {\endcsname 1100}%
93 \def\XINT_cshtb_D {\endcsname 1101}%
94 \def\XINT_cshtb_E {\endcsname 1110}%
95 \def\XINT_cshtb_F {\endcsname 1111}%
96 \let\XINT_cshtb_none \endcsname

```

6.4 Helper macros

6.4.1 \XINT_zeroes_foriv

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

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

```

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

```

6.5 \xintDecToHex

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

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

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

```

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

```

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

6.6 \xintDecToBin

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

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

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

Most of the code canvas is shared with \xintDecToHex.

```

211 \def\xintDecToBin {\romannumeral0\xintdectobin }%
212 \def\xintdectobin #1%
213 {%
214     \expandafter\XINT_dtb_checkin\romannumeral`&&@#1\xint:
215 }%
216 \def\XINT_dtb_checkin #1%
217 {%
218     \xint_UDsignfork
219     #1\XINT_dtb_neg
220     -{\XINT_dtb_main #1}%
221     \krof
222 }%
223 \def\XINT_dtb_neg {\expandafter-\romannumeral0\XINT_dtb_main}%
224 \def\XINT_dtb_main #1\xint:
225 {%
226     \expandafter\XINT_dtb_finish
227     \romannumeral`&&\expandafter\XINT_dtb_start
228     \romannumeral0\XINT_zeroes_foriv
229     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
230     #1\xint_bye\XINT_dtb_tobin
231 }%
232 \def\XINT_dtb_tobin
233 {%
234     \expandafter\expandafter\expandafter\XINT_dtb_tobin_a\cename\XINT_tosixteenbits
235 }%
236 \def\XINT_dtb_tobin_a\endcsname{!\XINT_dtb_tobin!}%
237 \def\XINT_tosixteenbits #1!%
238 {%
239     \expandafter\XINT_tosixteenbits_a
240     \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
241     #1\xint:
242 }%
243 \def\XINT_tosixteenbits_a #1\xint:#2\xint:
244 {%
245     \expandafter\XINT_tosixteenbits_c
246     \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
247     #1\xint:
248     \the\numexpr #2-\xint_c_ii^viii*#1!%

```

```

249 }%
250 \def\XINT_tosixteenbits_c #1\xint:#2\xint:
251 {%
252     XINT_csdtb_#1%
253     \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\relax
254     \csname \expandafter\XINT_tosixteenbits_d
255 }%
256 \def\XINT_tosixteenbits_d #1!%
257 {%
258     \expandafter\XINT_tosixteenbits_e
259     \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
260     #1\xint:
261 }%
262 \def\XINT_tosixteenbits_e #1\xint:#2\xint:
263 {%
264     XINT_csdtb_#1%
265     \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
266 }%
267 \def\XINT_dtb_finish !\XINT_dtb_tobin!#1#2#3#4#5#6#7#8%
268 {%
269     \expandafter\XINT_dtb_finish_a\the\numexpr #1#2#3#4#5#6#7#8\relax
270 }%
271 \def\XINT_dtb_finish_a #1{%
272 \def\XINT_dtb_finish_a ##1##2##3##4##5##6##7##8##9%
273 {%
274     \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8##9\relax
275 }}\XINT_dtb_finish_a { }%

```

6.7 \xintHexToDec

Completely (and belatedly) rewritten at 1.2m in the 1.2 style.

1.2m version robust against non terminated inputs, but there is no primitive from TeX which may generate hexadecimal digits and provoke expansion ahead, afaik, except of course if decimal digits are treated as hexadecimal. This robustness is not on purpose but from need to expand argument and then grab it again. So we do it safely.

Increased maximal size at 1.2n.

1.2m version robust against non terminated inputs.

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

```

276 \def\xintHexToDec {\romannumeral0\xinthextodec }%
277 \def\xinthextodec #1%
278 {%
279     \expandafter\XINT_htd_checkin\romannumeral`&&@#1\xint:
280 }%
281 \def\XINT_htd_checkin #1%
282 {%
283     \xint_UDsignfork
284     #1\XINT_htd_neg
285     -{\XINT_htd_main #1}%
286     \krof
287 }%
288 \def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_main}%
289 \def\XINT_htd_main #1\xint:

```

```

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

```

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

```

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

```

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

```

332 \def\XINT_htd_update 1#1#2#3#4#5#6!%
333 {%
334   *\xint_c_ii^xvi+100000#1#2#3#4#5!%#6!%

```



```

335 }%
336 \def\XINT_htd_A 1#1%
337 {%
338     \if#10\expandafter\XINT_htd_Aa\else
339         \expandafter\XINT_htd_Ab
340     \fi 1#1%
341 }%
342 \def\XINT_htd_Aa 10#1#2#3#4{\XINT_htd_again #1#2#3#4!}%
343 \def\XINT_htd_Ab 1#1#2#3#4#5{\XINT_htd_again #1!#2#3#4#5!}%
344 \def\XINT_htd_finish\xint_bye
345     \expandafter\XINT_htd_A\the\numexpr \XINT_htd_a #1\XINT_htd_nextfour
346 {%
347     \expandafter\XINT_htd_finish_cuz\the\numexpr0\XINT_htd_unsep_loop #1%
348 }%
349 \def\XINT_htd_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
350 {%
351     \expandafter\XINT_unsep_clean
352     \the\numexpr 1#1#2\expandafter\XINT_unsep_clean
353     \the\numexpr 1#3#4\expandafter\XINT_unsep_clean
354     \the\numexpr 1#5#6\expandafter\XINT_unsep_clean
355     \the\numexpr 1#7#8\expandafter\XINT_unsep_clean
356     \the\numexpr 1#9\XINT_htd_unsep_loop_a
357 }%
358 \def\XINT_htd_unsep_loop_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
359 {%
360     #1\expandafter\XINT_unsep_clean
361     \the\numexpr 1#2#3\expandafter\XINT_unsep_clean
362     \the\numexpr 1#4#5\expandafter\XINT_unsep_clean
363     \the\numexpr 1#6#7\expandafter\XINT_unsep_clean
364     \the\numexpr 1#8#9\XINT_htd_unsep_loop
365 }%
366 \def\XINT_unsep_clean 1{\relax}% also in xintcore
367 \def\XINT_htd_finish_cuz #1{%
368 \def\XINT_htd_finish_cuz ##1##2##3##4##5%
369     {\expandafter#1\the\numexpr ##1##2##3##4##5\relax}%
370 }\XINT_htd_finish_cuz{ }%

```

6.8 \xintBinToDec

Redone entirely for 1.2m. Starts by converting to hexadecimal first.

Increased maximal size at 1.2n.

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

Robust against non-terminated input.

```

371 \def\xintBinToDec {\romannumeral0\xintbintodec }%
372 \def\xintbintodec #1%
373 {%
374     \expandafter\XINT_btd_checkin\romannumeral`&&@#1\xint:
375 }%
376 \def\XINT_btd_checkin #1%
377 {%
378     \xint_UDsignfork
379     #1\XINT_btd_N

```

```

380      -{\XINT_btd_main #1}%
381      \krof
382 }%
383 \def\XINT_btd_N {\expandafter-\romannumeral0\XINT_btd_main }%
384 \def\XINT_btd_main #1\xint:
385 {%
386     \csname XINT_btd_htd\csname\expandafter\XINT_bth_loop
387     \romannumeral0\XINT_zeroes_foriv
388     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
389     #1\xint_bye2345678\xint_bye none\endcsname\xint:
390 }%
391 \def\XINT_btd_htd #1\xint:
392 {%
393     \expandafter\XINT_htd_startb
394     \the\numexpr\expandafter\XINT_htd_starta
395     \romannumeral0\XINT_zeroes_foriv
396     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
397     #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
398 }%

```

6.9 \xintBinToHex

Complete rewrite for 1.2m. But input for 1.2m version limited to about 13320 binary digits (expansion depth=10000).

Again redone for 1.2n for \csname governed expansion: increased maximal size.

Size of output is ceil(size(input)/4), leading zeroes in output (inherited from the input) are not trimmed.

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

Robust against non-terminated input.

```

399 \def\xintBinToHex {\romannumeral0\xintbinto hex }%
400 \def\xintbinto hex #1%
401 {%
402     \expandafter\XINT_bth_checkin\romannumeral`&&@#1\xint:
403 }%
404 \def\XINT_bth_checkin #1%
405 {%
406     \xint_UDsignfork
407     #1\XINT_bth_N
408     -{\XINT_bth_main #1}%
409     \krof
410 }%
411 \def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_main }%
412 \def\XINT_bth_main #1\xint:
413 {%
414     \csname space\csname\expandafter\XINT_bth_loop
415     \romannumeral0\XINT_zeroes_foriv
416     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
417     #1\xint_bye2345678\xint_bye none\endcsname
418 }%
419 \def\XINT_bth_loop #1#2#3#4#5#6#7#8%
420 {%
421     XINT_csbth_#1#2#3#4%

```

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

6.10 \xintHexToBin

Completely rewritten for 1.2m.

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

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

Rewritten again at 1.2n for \csname governed expansion.

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

6.11 \xintCHexToBin

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

Rewritten again at 1.2n for \csname governed expansion.

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

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

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

7 Package [xintgcd](#) implementation

.1	Catcodes, ε -TeX and reload detection . . .	165	.7	<code>\xintBezoutAlgorithm</code>	173
.2	Package identification	166	.8	<code>\xintGCDof</code>	175
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	166	.9	<code>\xintLCMof</code>	175
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	167	.10	<code>\xintTypesetEuclideanAlgorithm</code>	175
.5	<code>\xintBezout</code>	167	.11	<code>\xintTypesetBezoutAlgorithm</code>	176
.6	<code>\xintEuclideanAlgorithm</code>	171			

The commenting is currently (2018/06/17) very sparse. Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the `xinttools` `\xintloop` rather than the Plain TeX or μ TeX's `\loop`.

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

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

7.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

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

```

7.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintgcd}%
46 [2018/06/17 1.3c Euclide algorithm with xint package (JFB)]%

```

7.3 \xintGCD, \xintiigcd

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

1.1a defines \xintiigcd to avoid overhead in \xintiexpr.

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

```

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

```

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

```

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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfac](#), [xintexpr](#), [indices](#)

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

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

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

7.4 \xintLCM, \xintiilCM

See comments of \xintGCD.

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

```

7.5 \xintBezout

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

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

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

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

Note in particular change of sign of V.

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

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

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

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

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



```

143 \krof
144 }%

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

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

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

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

plusminus A < 0, B > 0

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

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

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

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

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfac, xintexpr, indices

```

vv, v = q * vv + v, vv
uu, u = q * uu + u, uu
e = -e

```

puis calcul quotient reste et itération

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

```

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

```

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

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

```

209 \def\xINT_bezout_loop_C #1#2#3#4#5#6#7%
210 {%
211   \expandafter\xINT_bezout_loop_D\expandafter
212     {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
213     {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%
214     {#2}{#3}{#4}{#5}%

```

```

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

```

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

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

```

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

```

7.6 \xintEuclideanAlgorithm

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

$u_{<2n} = u_{<2n+3}u_{<2n+2} + u_{<2n+4}$ à la n ième étape.

Formerly, used \xintiabs, but got deprecated at 1.20.

```

249 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm }%
250 \def\xinteucclideanalgorithm #1%
251 {%
252   \expandafter\XINT_euc\expandafter{\romannumeral0\xintiabs{\xintNum{#1}}}%
253 }%
254 \def\XINT_euc #1#2%
255 {%

```

```
256 \expandafter\XINT_euc_fork\romannumeral0\xinttiabs{\xintNum{#2}}\Z #1\Z
257 }%
```

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

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

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

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

$\{n\}\{r_n\}\{a_n\}\{\{q_n\}\{r_n\}}\dots\{A\}\{B\}}\}\Z$
 $a(n) = r(n-1)$. Pour $n=0$ on a juste $\{0\}\{B\}\{A\}\{\{A\}\{B\}}\}\Z$
 $\backslash\text{XINT_div_prepare } \{u\}\{v\}$ divise v par u

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

$\{n+1\}\{q(n+1)\}\{r(n+1)\}\{r_n\}\{\{q_n\}\{r_n\}}\dots$

```
274 \def\XINT_euc_b #1.#2#3#4%
275 {%
276 \XINT_euc_c #3\Z {#1}{#3}{#4}{#2}{#3}}%
277 }%
```

$r(n+1)\Z \{n+1\}\{r(n+1)\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}}\{\{q_n\}\{r_n\}}\dots$
 Test si $r(n+1)$ est nul.

```
278 \def\XINT_euc_c #1#2\Z
279 {%
280 \xint_gob_til_zero #1\XINT_euc_end0\XINT_euc_a
281 }%
```

$\{n+1\}\{r(n+1)\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}}\dots\}\Z$ Ici $r(n+1) = 0$. On arrête on se prépare à inverser
 $\{n+1\}\{0\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}}\dots\{\{q_1\}\{r_1\}}\{\{A\}\{B\}}\}\Z$
 On veut renvoyer: $\{N=n+1\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$

```
282 \def\XINT_euc_end0\XINT_euc_a #1#2#3#4\Z%
283 {%
284 \expandafter\XINT_euc_end_a
285 \romannumeral0%
286 \XINT_rord_main {}#4{{#1}{#3}}%
287 \xint:
```

```

288      \xint_bye\xint_bye\xint_bye\xint_bye
289      \xint_bye\xint_bye\xint_bye\xint_bye
290      \xint:
291 }%
292 \def\xINT_euc_end_a #1#2#3{{#1}{{#3}}{{#2}}}%

```

7.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

$\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$
 $\alpha_0=1, \beta_0=0, \alpha(-1)=0, \beta(-1)=1$

```

293 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm}%
294 \def\xintbezoutalgorithm #1%
295 {%
296   \expandafter \XINT_bezalg
297   \expandafter{\romannumeral0\xinttiabs{\xintNum{#1}}}%
298}%
299 \def\XINT_bezalg #1#2%
300 {%
301   \expandafter\XINT_bezalg_fork\romannumeral0\xinttiabs{\xintNum{#2}}\Z #1\Z
302}%

```

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

```

303 \def\XINT_bezalg_fork #1#2\Z #3#4\Z
304 {%
305   \xint_UDzerofork
306   #1\XINT_bezalg_BisZero
307   #3\XINT_bezalg_AisZero
308   0\XINT_bezalg_a
309   \krof
310   0{{#1#2}}{{#3#4}}1001{{#3#4}}{{#1#2}}{}{}\Z
311}%
312 \def\XINT_bezalg_AisZero #1#2#3\Z{{1}}{{0}}{{0}}{{1}}{{#2}}{{#2}}{{1}}{{0}}{{0}}{{0}}{{1}}}%
313 \def\XINT_bezalg_BisZero #1#2#3#4\Z{{1}}{{0}}{{0}}{{1}}{{#3}}{{#3}}{{1}}{{0}}{{0}}{{0}}{{0}}{{1}}}%

```

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

```

314 \def\XINT_bezalg_a #1#2#3%
315 {%
316   \expandafter\XINT_bezalg_b\the\numexpr #1+\xint_c_i\expandafter.%
317   \romannumeral0\XINT_div_prepare {#2}{{#3}}{{#2}}%
318}%

```

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

```

319 \def\XINT_bezalg_b #1.#2#3#4#5#6#7#8%
320 {%
321   \expandafter\XINT_bezalg_c\expandafter
322   {\romannumeral0\xinttiadd {\xintiiMul {#6}{{#2}}{{#8}}}%
323   {\romannumeral0\xinttiadd {\xintiiMul {#5}{{#2}}{{#7}}}%

```

```

324      {#1}{#2}{#3}{#4}{#5}{#6}%
325 }%

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

326 \def\XINT_bezalg_c #1#2#3#4#5#6%
327 {%
328     \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
329 }%

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

330 \def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
331 {%
332     \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{#3}{#4}{#1}{#6}%
333 }%

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

334 \def\XINT_bezalg_e #1#2\Z
335 {%
336     \xint_gob_til_zero #1\XINT_bezalg_end0\XINT_bezalg_a
337 }%

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

338 \def\XINT_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
339 {%
340     \expandafter\XINT_bezalg_end_a
341     \romannumeral0%
342     \XINT_rord_main {}#8{{#1}{#3}}%
343     \xint:
344         \xint_bye\xint_bye\xint_bye\xint_bye
345         \xint_bye\xint_bye\xint_bye\xint_bye
346     \xint:
347 }%

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

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

```

7.8 \xintGCDof

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

```
349 \def\xintGCDof      {\romannumeral0\xintgcdof }%
350 \def\xintgcdof      #1{\expandafter\xintgcdof_a\romannumeral`&&@#1\xint:}%
351 \def\xintgcdof_a    #1{\expandafter\xintgcdof_b\romannumeral`&&@#1!}%
352 \def\xintgcdof_b    #1!#2{\expandafter\xintgcdof_c\romannumeral`&&@#2!{#1}!}%
353 \def\xintgcdof_c    #1{\xint_gob_til_xint: #1\xintgcdof_e\xint:\xintgcdof_d #1}%
354 \def\xintgcdof_d    #1!{\expandafter\xintgcdof_b\romannumeral0\xintgcd {#1}}%
355 \def\xintgcdof_e    #1!#2!{ #2}%

```

7.9 \xintLCMof

New with 1.09a

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

```
356 \def\xintLCMof      {\romannumeral0\xintlcmof }%
357 \def\xintlcmof      #1{\expandafter\xintlcmof_a\romannumeral`&&@#1\xint:}%
358 \def\xintlcmof_a    #1{\expandafter\xintlcmof_b\romannumeral`&&@#1!}%
359 \def\xintlcmof_b    #1!#2{\expandafter\xintlcmof_c\romannumeral`&&@#2!{#1}!}%
360 \def\xintlcmof_c    #1{\xint_gob_til_xint: #1\xintlcmof_e\xint:\xintlcmof_d #1}%
361 \def\xintlcmof_d    #1!{\expandafter\xintlcmof_b\romannumeral0\xintlcm {#1}}%
362 \def\xintlcmof_e    #1!#2!{ #2}%

```

7.10 \xintTypesetEuclideanAlgorithm

TYPESETTING

Organisation:

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

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

$r(n-2) = q(n)r(n-1)+r(n)$ (n e étape)

$\backslash U\{2n\} = \backslash U\{2n+3\} \times \backslash U\{2n+2\} + \backslash U\{2n+4\}$, n e étape. (avec n entre 1 et N)

1.09h uses \xintloop, and \par rather than \endgraf; and \par rather than \hfill\break

```
363 \def\xintTypesetEuclideanAlgorithm {%
364   \unless\ifdefined\xintAssignArray
365     \errmessage
366     {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
367     \expandafter\xint_gobble_iii
368   \fi
369   \XINT_TypesetEuclideanAlgorithm
370 }%
371 \def\XINT_TypesetEuclideanAlgorithm #1#2%
372 {% l'algo remplace #1 et #2 par |#1| et |#2|
373   \par
374   \begingroup
375     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
376     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
377     \setbox 0 \vbox{\halign {\$##$\cr \A\cr \B \cr}}%
378     \count 255 1

```

```

379 \xintloop
380 \indent\hbox to \wd 0 {\hfil$U{\numexpr 2*\count255\relax}$}%
381 ${} = \U{\numexpr 2*\count255 + 3\relax}
382 \times \U{\numexpr 2*\count255 + 2\relax}
383 + \U{\numexpr 2*\count255 + 4\relax}$%
384 \ifnum \count255 < \N
385 \par
386 \advance \count255 1
387 \repeat
388 \endgroup
389 }%

```

7.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a: $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$
 Donc $4N+8$ termes: $U_1 = N$, $U_2 = A$, $U_5 = D$, $U_6 = B$, $q_1 = U_9$, $q_n = U_{4n+5}$, n au moins 1
 $r_n = U_{4n+6}$, n au moins -1
 $\alpha(n) = U_{4n+7}$, n au moins -1
 $\beta(n) = U_{4n+8}$, n au moins -1

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

```

390 \def\xintTypesetBezoutAlgorithm {%
391 \unless\ifdefined\xintAssignArray
392 \errmessage
393 {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
394 \expandafter\xint_gobble_iii
395 \fi
396 \XINT_TypesetBezoutAlgorithm
397 }%
398 \def\XINT_TypesetBezoutAlgorithm #1#2%
399 {%
400 \par
401 \begingroup
402 \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
403 \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
404 \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
405 \count255 1
406 \xintloop
407 \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
408 ${} = \BEZ{4*\count255 + 5}
409 \times \BEZ{4*\count255 + 2}
410 + \BEZ{4*\count255 + 6}$\hfill\break
411 \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 7}$}%
412 ${} = \BEZ{4*\count255 + 5}
413 \times \BEZ{4*\count255 + 3}
414 + \BEZ{4*\count255 - 1}$\hfill\break
415 \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 8}$}%
416 ${} = \BEZ{4*\count255 + 5}
417 \times \BEZ{4*\count255 + 4}
418 + \BEZ{4*\count255 }$
419 \par
420 \ifnum \count255 < \N

```


[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```
421 \advance \count255 1
422 \repeat
423 \edef\U{\BEZ{4*\N + 4}}%
424 \edef\V{\BEZ{4*\N + 3}}%
425 \edef\D{\BEZ5}%
426 \ifodd\N
427   $\U\times A - \V\times B = -\D$%
428 \else
429   $\U\times A - \V\times B = \D$%
430 \fi
431 \par
432 \endgroup
433 }%
434 \XINT_restorecatcodes_endinput%
```

8 Package [xintfrac](#) implementation

.1	Catcodes, ε -T _E X and reload detection . . .	178	.41	\xintBinomial	211
.2	Package identification	179	.42	\xintPFactorial	211
.3	\XINT_cntSgnFork	179	.43	\xintPrd	212
.4	\xintLen	180	.44	\xintDiv	212
.5	\XINT_outfrac	180	.45	\xintDivFloor	213
.6	\XINT_inFrac	181	.46	\xintDivTrunc	213
.7	\XINT_frac_gen	183	.47	\xintDivRound	213
.8	\XINT_factortens	185	.48	\xintModTrunc	213
.9	\xintEq, \xintNotEq, \xintGt, \xintLt, \xintGtorEq, \xintLtorEq, \xintIsZero, \xintIsNotZero, \xintOdd, \xintEven, \xintifSgn, \xintifCmp, \xintifEq, \xin- tifGt, \xintifLt, \xintifZero, \xin- tifNotZero, \xintifOne, \xintifOdd . . .	186	.49	\xintDivMod	214
.10	\xintRaw	188	.50	\xintMod	215
.11	\xintPRaw	188	.51	\xintIsOne	216
.12	\xintRawWithZeros	189	.52	\xintGeq	216
.13	\xintDecToString	189	.53	\xintMax	217
.14	\xintFloor, \xintiFloor	189	.54	\xintMaxof	218
.15	\xintCeil, \xintiCeil	190	.55	\xintMin	218
.16	\xintNumerator	190	.56	\xintMinof	219
.17	\xintDenominator	190	.57	\xintCmp	219
.18	\xintFrac	191	.58	\xintAbs	221
.19	\xintSignedFrac	191	.59	\xintOpp	221
.20	\xintFwOver	192	.60	\xintSgn	221
.21	\xintSignedFwOver	192	.61	Floating point macros	221
.22	\xintREZ	193	.62	\xintFloat	222
.23	\xintE	193	.63	\XINTinFloat, \XINTinFloatS	223
.24	\xintIrr, \xintPIrr	194	.64	\xintPFloat	229
.25	\xintifInt	195	.65	\XINTinFloatFracdigits	231
.26	\xintJrr	196	.66	\xintFloatAdd, \XINTinFloatAdd . . .	231
.27	\xintTFrac	197	.67	\xintFloatSub, \XINTinFloatSub . . .	233
.28	\xintTrunc, \xintiTrunc	197	.68	\xintFloatMul, \XINTinFloatMul . . .	233
.29	\xintTTrunc	200	.69	\xintFloatDiv, \XINTinFloatDiv . . .	234
.30	\xintNum	200	.70	\xintFloatPow, \XINTinFloatPow . . .	235
.31	\xintRound, \xintiRound	200	.71	\xintFloatPower, \XINTinFloatPower .	238
.32	\xintXTrunc	201	.72	\xintFloatFac, \XINTFloatFac	242
.33	\xintDigits	207	.73	\xintFloatPFactorial, \XINTinFloatP- Factorial	247
.34	\xintAdd	207	.74	\xintFloatBinomial, \XINTinFloatBino- mial	251
.35	\xintSub	209	.75	\xintFloatSqrt, \XINTinFloatSqrt . . .	253
.36	\xintSum	209	.76	\xintFloatE, \XINTinFloatE	255
.37	\xintMul	209	.77	\XINTinFloatMod	256
.38	\xintSqr	210	.78	\XINTinFloatDivFloor	256
.39	\xintPow	210	.79	\XINTinFloatDivMod	256
.40	\xintFac	211	.80	\xintifFloatInt	257
			.81	(WIP) \XINTinRandomFloatS, \XINTinRan- domFloatSdigits	257
			.82	(WIP) \XINTinRandomFloatSixteen . . .	258

The commenting is currently (2018/06/17) very sparse.

8.1 Catcodes, ε -T_EX and reload detection

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

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

```

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

```

8.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2018/06/17 1.3c Expandable operations on fractions (JFB)]%

```

8.3 \XINT_cntSgnFork

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

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

```

47 \def\XINT_cntSgnFork #1%
48 {%
49   \ifcase #1\expandafter\XINT_secondofthree
50     \or\expandafter\XINT_thirdofthree
51     \else\expandafter\XINT_firstofthree
52   \fi
53 }%

```

8.4 \xintLen

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

```

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

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

```

8.5 \XINT_outfrac

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

```

68 \def\XINT_outfrac #1#2#3%
69 {%
70   \ifcase\XINT_cntSgn #3\xint:
71     \expandafter \XINT_outfrac_divisionbyzero
72   \or
73     \expandafter \XINT_outfrac_P
74   \else
75     \expandafter \XINT_outfrac_N
76   \fi
77   {#2}{#3}[#1]%
78 }%
79 \def\XINT_outfrac_divisionbyzero #1#2%

```

```

80 {%
81   \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}{0/1[0]}%
82 }%
83 \def\XINT_outfrac_P#1{%
84 \def\XINT_outfrac_P ##1##2%
85   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi##1/##2}%
86 }\XINT_outfrac_P{ }%
87 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
88 \def\XINT_outfrac_N #1#2%
89 {%
90   \expandafter\XINT_outfrac_N_a\expandafter
91   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
92 }%
93 \def\XINT_outfrac_N_a #1#2%
94 {%
95   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
96 }%

```

8.6 \XINT_inFrac

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

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

2015/10/09: this venerable macro from the very early days (1.03, 2013/04/14) has gotten a lifting for release 1.2. There were two kinds of issues:

- 1) use of `\W`, `\Z`, `\T` delimiters was very poor choice as this could clash with user input,
- 2) the new `\XINT_frac_gen` handles macros (possibly empty) in the input as general as `\A.\Be\C/\D.\Ee\F`. The earlier version would not have expanded the `\B` or `\E`: digits after decimal mark were constrained to arise from expansion of the first token. Thus the 1.03 original code would have expanded only `\A`, `\D`, `\C`, and `\F` for this input.

This reminded me think I should revisit the remaining earlier portions of code, as I was still learning TeX coding when I wrote them.

Also I thought about parsing even faster the $A/B[N]$ input, not expanding B, but this turned out to clash with some established uses in the documentation such as `1/\xintiiSqr{...}[0]`. For the implementation, careful here about potential brace removals with parameter patterns such as like `#1/#2#3[#4]` for example.

While I was at it 1.2 added `\numexpr` parsing of the N, which earlier was restricted to be only explicit digits. I allowed `[]` with empty N, but the way I did it in 1.2 with `\the\numexpr 0#1` was buggy, as it did not allow `#1` to be a `\count` for example or itself a `\numexpr` (although such inputs were not previously allowed, I later turned out to use them in the code itself, e.g. the float factorial of version 1.2f). The better way would be `\the\numexpr#1+\xint_c_` but 1.2f finally does only `\the\numexpr #1` and `#1` is not allowed to be empty.

The 1.2 `\XINT_frac_gen` had two locations with such a problematic `\numexpr 0#1` which I replaced for 1.2f with `\numexpr#1+\xint_c_`.

Regarding calling the macro with an argument `A[<expression>]`, a / in the expression must be suitably hidden for example in `\firstofone` type constructs.

Note: when the numerator is found to be zero `\XINT_inFrac` *always* returns $\{0\}{0}\{1\}$. This behaviour must not change because 1.2g `\xintFloat` and `XINTinFloat` (for example) rely upon it: if the denominator on output is not 1, then `\xintFloat` assumes that the numerator is not zero.

As described in the manual, if the input contains a (final) [N] part, it is assumed that it is in the shape A[N] or A/B[N] with A (and B) not containing neither decimal mark nor scientific part, moreover B must be positive and A have at most one minus sign (and no plus sign). Else there will be errors, for example -0/2[0] would not be recognized as being zero at this stage and this could cause issues afterwards. When there is no ending [N] part, both numerator and denominator will be parsed for the more general format allowing decimal digits and scientific part and possibly multiple leading signs.

1.2l fixes frailty of \XINT_infrac (hence basically of all xintfrac macros) respective to non terminated \numexpr input: \xintRaw{\the\numexpr1} for example. The issue was that \numexpr sees the / and expands what's next. But even \numexpr 1// for example creates an error, and to my mind this is a defect of \numexpr. It should be able to trace back and see that / was used as delimiter not as operator. Anyway, I thus fixed this problem belatedly here regarding \XINT_infrac.

```

97 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
98 \def\XINT_infrac #1%
99 {%
100   \expandafter\XINT_infrac_fork\romannumeral`&&@#1\xint:/\XINT_W[\XINT_W\XINT_T
101 }%
102 \def\XINT_infrac_fork #1[#2%
103 {%
104   \xint_UDXINTWfork
105   #2\XINT_frac_gen          % input has no brackets [N]
106   \XINT_W\XINT_infrac_res_a % there is some [N], must be strict A[N] or A/B[N] input
107   \krof
108   #1[#2%
109 }%
110 \def\XINT_infrac_res_a #1%
111 {%
112   \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
113 }%

```

Note that input exponent is here ignored and forced to be zero.

```

114 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
115 \def\XINT_infrac_res_b #1/#2%
116 {%
117   \xint_UDXINTWfork
118   #2\XINT_infrac_res_ca      % it was A[N] input
119   \XINT_W\XINT_infrac_res_cb % it was A/B[N] input
120   \krof
121   #1/#2%
122 }%

```

An empty [] is not allowed. (this was authorized in 1.2, removed in 1.2f). As nobody reads xint documentation, no one will have noticed the fleeting possibility.

```

123 \def\XINT_infrac_res_ca #1[#2]\xint:/\XINT_W[\XINT_W\XINT_T
124   {\expandafter{\the\numexpr #2}{#1}{1}}%
125 \def\XINT_infrac_res_cb #1/#2[%
126   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[%
127 \def\XINT_infrac_res_cc #1~#2[#3]\xint:/\XINT_W[\XINT_W\XINT_T
128   {\expandafter{\the\numexpr #3}{#2}{#1}}%

```

8.7 \XINT_frac_gen

Extended in 1.07 to recognize and accept scientific notation both at the numerator and (possible) denominator. Only a lowercase e will do here, but uppercase E is possible within an \xintexpr.\relax

Completely rewritten for 1.2 2015/10/10. The parsing handles inputs such as \A.\Be\C/\D.\Ee\F where each of \A, \B, \D, and \E may need f-expansion and \C and \F will end up in \numexpr.

1.2f corrects an issue to allow \C and \F to be \count variable (or expressions with \numexpr): 1.2 did a bad \numexpr0#1 which allowed only explicit digits for expanded #1.

```
129 \def\XINT_frac_gen #1/#2%
130 {%
131   \xint_UDXINTWfork
132   #2\XINT_frac_gen_A      % there was no /
133   \XINT_W\XINT_frac_gen_B % there was a /
134   \krof
135   #1/#2%
136 }%
```

Note that #1 is only expanded so far up to decimal mark or "e".

```
137 \def\XINT_frac_gen_A #1\xint:\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
138 \def\XINT_frac_gen_B #1/#2\xint:\XINT_W[%\XINT_W
139 {%
140   \expandafter\XINT_frac_gen_Ba
141   \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
142 }%
143 \def\XINT_frac_gen_Ba #1.#2%
144 {%
145   \xint_UDXINTWfork
146   #2\XINT_frac_gen_Bb
147   \XINT_W\XINT_frac_gen_Bc
148   \krof
149   #1.#2%
150 }%
151 \def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
152   {\expandafter\XINT_frac_gen_C\the\numexpr #2+\xint_c~#1!}%
153 \def\XINT_frac_gen_Bc #1.#2e%
154 {%
155   \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
156 }%
157 \def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
158 {%
159   \expandafter\XINT_frac_gen_C\the\numexpr #3-%
160   \numexpr\XINT_length_loop
161   #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
162   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
163   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
164   ~#2#1!%
165 }%
166 \def\XINT_frac_gen_C #1!#2.#3%
167 {%
168   \xint_UDXINTWfork
```



```

219 }%
220 \def\XINT_frac_gen_Ga #1#2~#3~%
221 {%
222     \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
223     {#3}{#1#2}%
224 }%
225 \def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%

```

8.8 \XINT_factortens

This is the core macro for \xintREZ. To be used as \romannumeral0\XINT_factortens{...}. Output is A.N. (formerly {A}{N}) where A is the integer stripped from trailing zeroes and N is the number of removed zeroes. Only for positive strict integers!

Completely rewritten at 1.3a to replace a double \xintReverseOrder by a direct \numexpr governed expansion to the end and back, à la 1.2. I should comment more... and perhaps improve again in future.

Testing shows significant gain at 100 digits or more.

```

226 \def\XINT_factortens #1{\expandafter\XINT_factortens_z
227     \romannumeral0\XINT_factortens_a#1%
228     \XINT_factortens_b123456789.}%
229 \def\XINT_factortens_z.\XINT_factortens_y{ }%
230 \def\XINT_factortens_a #1#2#3#4#5#6#7#8#9%
231     {\expandafter\XINT_factortens_x
232     \the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_factortens_a}%
233 \def\XINT_factortens_b#1\XINT_factortens_a#2#3.%
234     {.\XINT_factortens_cc 000000000-#2.}%
235 \def\XINT_factortens_x1#1.#2{#2#1}%
236 \def\XINT_factortens_y{.\XINT_factortens_y}%
237 \def\XINT_factortens_cc #1#2#3#4#5#6#7#8#9%
238     {\if90\xint_dothis
239     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
240     \xint_c_i 2345678.}\fi
241     \xint_orthat{\XINT_factortens_yy{#1#2#3#4#5#6#7#8#9}}}%
242 \def\XINT_factortens_yy #1#2.{.\XINT_factortens_y#1.0.}%
243 \def\XINT_factortens_c #1#2#3#4#5#6#7#8#9%
244     {\if90\xint_dothis
245     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
246     \xint_c_i 2345678.}\fi
247     \xint_orthat{.\XINT_factortens_y #1#2#3#4#5#6#7#8#9.}}}%
248 \def\XINT_factortens_d #1#2#3#4#5#6#7#8#9%
249     {\if10\expandafter\XINT_factortens_e\fi
250     \XINT_factortens_f #9#9#8#7#6#5#4#3#2#1.}%
251 \def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
252     {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
253 \def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
254 \def\XINT_factortens_e #1..#2.%
255     {\expandafter.\expandafter\XINT_factortens_c
256     \the\numexpr\xint_c_ix+#2.}%

```

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

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

```

257 \def\xintEq    {\romannumeral0\xinteq}%
258 \def\xinteq    #1#2{\xintifeq{#1}{#2}{1}{0}}%
259 \def\xintNotEq#1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
260 \def\xintGt    {\romannumeral0\xintgt}%
261 \def\xintgt    #1#2{\xintifgt{#1}{#2}{1}{0}}%
262 \def\xintLt    {\romannumeral0\xintlt}%
263 \def\xintlt    #1#2{\xintiflt{#1}{#2}{1}{0}}%
264 \def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
265 \def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
266 \def\xintIsZero {\romannumeral0\xintiszero}%
267 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
268 \def\xintIsNotZero{\romannumeral0\xintisnotzero}%
269 \def\xintisnotzero
270     #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
271 \def\xintOdd    {\romannumeral0\xintodd}%
272 \def\xintodd    #1%
273 {%
274     \ifodd\xintLDg{\xintNum{#1}}%<- intentional space
275     \xint_afterfi{ 1}%
276     \else
277     \xint_afterfi{ 0}%
278     \fi
279}%
280 \def\xintEven    {\romannumeral0\xinteven}%
281 \def\xinteven    #1%
282 {%
283     \ifodd\xintLDg{\xintNum{#1}}%<- intentional space
284     \xint_afterfi{ 0}%
285     \else
286     \xint_afterfi{ 1}%
287     \fi
288}%
289 \def\xintifSgn{\romannumeral0\xintifsgn}%
290 \def\xintifsgn #1%
291 {%
292     \ifcase \xintSgn{#1}
293         \expandafter\xint_secondofthree_thenstop
294         \or\expandafter\xint_thirdofthree_thenstop
295         \else\expandafter\xint_firstofthree_thenstop
296     \fi
297}%
298 \def\xintifCmp{\romannumeral0\xintifcmp}%
299 \def\xintifcmp #1#2%
300 {%
301     \ifcase\xintCmp {#1}{#2}
302         \expandafter\xint_secondofthree_thenstop

```

```

303         \or\expandafter\xint_thirdofthree_thenstop
304         \else\expandafter\xint_firstofthree_thenstop
305     \fi
306 }%
307 \def\xintifEq {\romannumeral0\xintifeq }%
308 \def\xintifeq #1#2%
309 {%
310     \if0\xintCmp{#1}{#2}%
311         \expandafter\xint_firstoftwo_thenstop
312     \else\expandafter\xint_secondoftwo_thenstop
313     \fi
314 }%
315 \def\xintifGt {\romannumeral0\xintifgt }%
316 \def\xintifgt #1#2%
317 {%
318     \if1\xintCmp{#1}{#2}%
319         \expandafter\xint_firstoftwo_thenstop
320     \else\expandafter\xint_secondoftwo_thenstop
321     \fi
322 }%
323 \def\xintifLt {\romannumeral0\xintiflt }%
324 \def\xintiflt #1#2%
325 {%
326     \ifnum\xintCmp{#1}{#2}<\xint_c_
327         \expandafter\xint_firstoftwo_thenstop
328     \else \expandafter\xint_secondoftwo_thenstop
329     \fi
330 }%
331 \def\xintifZero {\romannumeral0\xintifzero }%
332 \def\xintifzero #1%
333 {%
334     \if0\xintSgn{#1}%
335         \expandafter\xint_firstoftwo_thenstop
336     \else
337         \expandafter\xint_secondoftwo_thenstop
338     \fi
339 }%
340 \def\xintifNotZero{\romannumeral0\xintifnotzero }%
341 \def\xintifnotzero #1%
342 {%
343     \if0\xintSgn{#1}%
344         \expandafter\xint_secondoftwo_thenstop
345     \else
346         \expandafter\xint_firstoftwo_thenstop
347     \fi
348 }%
349 \def\xintifOne {\romannumeral0\xintifone }%
350 \def\xintifone #1%
351 {%
352     \if1\xintIsOne{#1}%
353         \expandafter\xint_firstoftwo_thenstop
354     \else

```

```

355     \expandafter\xint_secondoftwo_thenstop
356   \fi
357 }%
358 \def\xintifOdd {\romannumeral0\xintifodd }%
359 \def\xintifodd #1%
360 {%
361   \if\xintOdd{#1}1%
362     \expandafter\xint_firstoftwo_thenstop
363   \else
364     \expandafter\xint_secondoftwo_thenstop
365   \fi
366 }%

```

8.10 \xintRaw

1.07: this macro simply prints in a user readable form the fraction after its initial scanning. Useful when put inside braces in an \xintexpr, when the input is not yet in the A/B[n] form.

```

367 \def\xintRaw {\romannumeral0\xinraw }%
368 \def\xinraw
369 {%
370   \expandafter\XINT_raw\romannumeral0\XINT_infrac
371 }%
372 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

8.11 \xintPraw

1.09b

```

373 \def\xintPraw {\romannumeral0\xintpraw }%
374 \def\xintpraw
375 {%
376   \expandafter\XINT_praw\romannumeral0\XINT_infrac
377 }%
378 \def\XINT_praw #1%
379 {%
380   \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
381 }%
382 \def\XINT_praw_A #1#2#3%
383 {%
384   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
385     \else\expandafter\xint_secondoftwo
386   \fi { #2[#1]}{ #2/#3[#1]}%
387 }%
388 \def\XINT_praw_a\XINT_praw_A #1#2#3%
389 {%
390   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
391     \else\expandafter\xint_secondoftwo
392   \fi { #2}{ #2/#3}%
393 }%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

8.12 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```
394 \def\xintRawWithZeros {\romannumeral0\xinrawwithzeros }%
395 \def\xinrawwithzeros
396 {%
397     \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
398 }%

399 \def\XINT_rawz_fork #1%
400 {%
401     \ifnum#1<\xint_c_
402         \expandafter\XINT_rawz_Ba
403     \else
404         \expandafter\XINT_rawz_A
405     \fi
406     #1.%
407 }%
408 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
409 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
410     \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
411 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

8.13 \xintDecToString

1.3. This is a backport from polexpr 0.4. It is definitely not in final form, consider it to be an unstable macro.

```
412 \def\xintDecToString{\romannumeral0\xintdectostr}%
413 \def\xintdectostr#1{\expandafter\XINT_dectostr\romannumeral0\xinraw{#1}}%
414 \def\XINT_dectostr #1/#2[#3]{\xintiifZero {#1}%
415     \XINT_dectostr_z
416     {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
417         \else\expandafter\XINT_dectostr_b
418     \fi}%
419     #1/#2[#3]%
420 }%
421 \def\XINT_dectostr_z#1[#2]{ 0}%
422 \def\XINT_dectostr_a#1/#2[#3]{%
423     \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
424     \xint_orthat{\xintiie{#1}{#3}}%
425 }%
426 \def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
427     \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
428     \xint_orthat{\xintiie{#1}{#3}/#2}%
429 }%

```

8.14 \xintFloor, \xintiFloor

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

```

430 \def\xintFloor {\romannumeral0\xintfloor }%
431 \def\xintfloor #1% devrais-je faire \xintREZ?
432   {\expandafter\XINT_ifloor \romannumeral0\xintraawithzeros {#1}./1[0]}%
433 \def\xintiFloor {\romannumeral0\xintifloor }%
434 \def\xintifloor #1%
435   {\expandafter\XINT_ifloor \romannumeral0\xintraawithzeros {#1}.}%
436 \def\XINT_ifloor #1/#2.{\xintiique {#1}{#2}}%

```

8.15 \xintCeil, \xintiCeil

1.09a

```

437 \def\xintCeil {\romannumeral0\xintceil }%
438 \def\xintceil #1{\xintiopp {\xintFloor {\xintOpp{#1}}}}%
439 \def\xintiCeil {\romannumeral0\xinticeil }%
440 \def\xinticeil #1{\xintiopp {\xintiFloor {\xintOpp{#1}}}}%

```

8.16 \xintNumerator

```

441 \def\xintNumerator {\romannumeral0\xintnumerator }%
442 \def\xintnumerator
443 {%
444   \expandafter\XINT_numer\romannumeral0\XINT_infrac
445 }%
446 \def\XINT_numer #1%
447 {%
448   \ifcase\XINT_cntSgn #1\xint:
449     \expandafter\XINT_numer_B
450   \or
451     \expandafter\XINT_numer_A
452   \else
453     \expandafter\XINT_numer_B
454   \fi
455   {#1}%
456 }%
457 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
458 \def\XINT_numer_B #1#2#3{ #2}%

```

8.17 \xintDenominator

```

459 \def\xintDenominator {\romannumeral0\xintdenominator }%
460 \def\xintdenominator
461 {%
462   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
463 }%
464 \def\XINT_denom_fork #1%
465 {%
466   \ifnum#1<\xint_c_
467     \expandafter\XINT_denom_B
468   \else
469     \expandafter\XINT_denom_A
470   \fi
471   #1.%

```

```

472 }%
473 \def\XINT_denom_A #1.#2#3{ #3}%
474 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

8.18 \xintFrac

Useless typesetting macro.

```

475 \def\xintFrac {\romannumeral0\xintfrac }%
476 \def\xintfrac #1%
477 {%
478   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
479 }%
480 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
481 \catcode`^=7
482 \def\XINT_fracfrac_B #1#2\Z
483 {%
484   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
485 }%
486 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
487 {%
488   \if1\XINT_isOne {#3}%
489   \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
490   \fi
491   \space
492   \frac {#2}{#3}%
493 }%
494 \def\XINT_fracfrac_D #1#2#3%
495 {%
496   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
497   \space
498   \frac {#2}{#3}#1%
499 }%
500 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

8.19 \xintSignedFrac

```

501 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
502 \def\xintsignedfrac #1%
503 {%
504   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
505 }%
506 \def\XINT_sgnfrac_a #1#2%
507 {%
508   \XINT_sgnfrac_b #2\Z {#1}%
509 }%
510 \def\XINT_sgnfrac_b #1%
511 {%
512   \xint_UDsignfork
513   #1\XINT_sgnfrac_N
514   -{\XINT_sgnfrac_P #1}%
515   \krof
516 }%
517 \def\XINT_sgnfrac_P #1\Z #2%

```

```

518 {%
519     \XINT_fracfrac_A {#2}{#1}%
520 }%
521 \def\XINT_sgnfrac_N
522 {%
523     \expandafter\romannumeral0\XINT_sgnfrac_P
524 }%

```

8.20 \xintFwOver

```

525 \def\xintFwOver {\romannumeral0\xintfwover }%
526 \def\xintfwover #1%
527 {%
528     \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
529 }%
530 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
531 \def\XINT_fwover_B #1#2\Z
532 {%
533     \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
534 }%
535 \catcode\^=11
536 \def\XINT_fwover_C #1#2#3#4#5%
537 {%
538     \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
539         \else\xint_afterfi { #4}%
540     \fi
541 }%
542 \def\XINT_fwover_D #1#2#3%
543 {%
544     \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
545         \else\xint_afterfi { #2\cdot }%
546     \fi
547     #1%
548 }%

```

8.21 \xintSignedFwOver

```

549 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
550 \def\xintsignedfwover #1%
551 {%
552     \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
553 }%
554 \def\XINT_sgnfwover_a #1#2%
555 {%
556     \XINT_sgnfwover_b #2\Z {#1}%
557 }%
558 \def\XINT_sgnfwover_b #1%
559 {%
560     \xint_UDsignfork
561     #1\XINT_sgnfwover_N
562     -{\XINT_sgnfwover_P #1}%
563     \krof
564 }%
565 \def\XINT_sgnfwover_P #1\Z #2%
566 {%

```


TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

567 \XINT_fwover_A {#2}{#1}%
568 }%
569 \def\XINT_sgnfwover_N
570 {%
571 \expandafter-\romannumeral0\XINT_sgnfwover_P
572 }%

```

8.22 \xintREZ

Removes trailing zeros from A and B and adjust the N in A/B[N].

The macro really doing the job \XINT_factortens was redone at 1.3a. But speed gain really noticeable only beyond about 100 digits.

```

573 \def\xintREZ {\romannumeral0\xintrez }%
574 \def\xintrez
575 {%
576 \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
577 }%
578 \def\XINT_rez_A #1#2%
579 {%
580 \XINT_rez_AB #2\Z {#1}%
581 }%
582 \def\XINT_rez_AB #1%
583 {%
584 \xint_UDzerominusfork
585 #1-\XINT_rez_zero
586 0#1\XINT_rez_neg
587 0-{\XINT_rez_B #1}%
588 \krof
589 }%
590 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
591 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
592 \def\XINT_rez_B #1\Z
593 {%
594 \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
595 }%
596 \def\XINT_rez_C #1.#2.#3#4%
597 {%
598 \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
599 }%
600 \def\XINT_rez_D #1.#2.#3.%
601 {%
602 \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
603 }%
604 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%

```

8.23 \xintE

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

1.1 modifies and moves \xintiiE to xint.sty.

```

605 \def\xintE {\romannumeral0\xinte }%
606 \def\xinte #1%

```

```

607 {%
608   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
609 }%
610 \def\XINT_e #1#2#3#4%
611 {%
612   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
613 }%
614 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

8.24 \xintIrr, \xintPIrr

\xintPIrr (partial Irr, which ignores the decimal part) added at 1.3.

```

615 \def\xintIrr {\romannumeral0\xintirr }%
616 \def\xintPIrr{\romannumeral0\xintpirr }%
617 \def\xintirr #1%
618 {%
619   \expandafter\XINT_irr_start\romannumeral0\xintraawithzeros {#1}\Z
620 }%
621 \def\xintpirr #1%
622 {%
623   \expandafter\XINT_pirr_start\romannumeral0\xintraaw{#1}%
624 }%
625 \def\XINT_irr_start #1#2/#3\Z
626 {%
627   \if0\XINT_isOne {#3}%
628     \xint_afterfi
629     {\xint_UDsignfork
630       #1\XINT_irr_negative
631       -{\XINT_irr_nonneg #1}%
632     \krof}%
633   \else
634     \xint_afterfi{\XINT_irr_denomisine #1}%
635   \fi
636   #2\Z {#3}%
637 }%
638 \def\XINT_pirr_start #1#2/#3[%
639 {%
640   \if0\XINT_isOne {#3}%
641     \xint_afterfi
642     {\xint_UDsignfork
643       #1\XINT_irr_negative
644       -{\XINT_irr_nonneg #1}%
645     \krof}%
646   \else
647     \xint_afterfi{\XINT_irr_denomisine #1}%
648   \fi
649   #2\Z {#3}[%
650 }%
651 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
652 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
653 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
654 \def\XINT_irr_D #1#2\Z #3#4\Z

```

```

655 {%
656   \xint_UDzerosfork
657     #3#1\XINT_irr_indeterminate
658     #30\XINT_irr_divisionbyzero
659     #10\XINT_irr_zero
660     00\XINT_irr_loop_a
661   \krof
662   {#3#4}{#1#2}{#3#4}{#1#2}%
663 }%
664 \def\XINT_irr_indeterminate #1#2#3#4#5%
665 {%
666   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
667 }%
668 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
669 {%
670   \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{0/1}%
671 }%
672 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
673 \def\XINT_irr_loop_a #1#2%
674 {%
675   \expandafter\XINT_irr_loop_d
676   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
677 }%
678 \def\XINT_irr_loop_d #1#2%
679 {%
680   \XINT_irr_loop_e #2\Z
681 }%
682 \def\XINT_irr_loop_e #1#2\Z
683 {%
684   \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
685 }%
686 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
687 {%
688   \expandafter\XINT_irr_loop_exitb\expandafter
689   {\romannumeral0\xintiique {#3}{#2}}%
690   {\romannumeral0\xintiique {#4}{#2}}%
691 }%
692 \def\XINT_irr_loop_exitb #1#2%
693 {%
694   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
695 }%
696 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

8.25 \xintifInt

```

697 \def\xintifInt {\romannumeral0\xintifint }%
698 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
699 \def\XINT_ifint #1/#2.%
700 {%
701   \if 0\xintiiRem {#1}{#2}%
702   \expandafter\xint_firstoftwo_thenstop
703   \else
704   \expandafter\xint_secondoftwo_thenstop

```

705 \fi

706 }%

8.26 \xintJrr

707 \def\xintJrr {\romannumeral0\xintjrr }%

708 \def\xintjrr #1%

709 {%

710 \expandafter\XINT_jrr_start\romannumeral0\xintraawithzeros {#1}\Z

711 }%

712 \def\XINT_jrr_start #1#2/#3\Z

713 {%

714 \if0\XINT_isOne {#3}\xint_afterfi

715 {\xint_UDsignfork

716 #1\XINT_jrr_negative

717 -{\XINT_jrr_nonneg #1}%

718 \krof}%

719 \else

720 \xint_afterfi{\XINT_jrr_denomiseone #1}%

721 \fi

722 #2\Z {#3}%

723 }%

724 \def\XINT_jrr_denomiseone #1\Z #2{ #1/1}% changed in 1.08

725 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%

726 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%

727 \def\XINT_jrr_D #1#2\Z #3#4\Z

728 {%

729 \xint_UDzerosfork

730 #3#1\XINT_jrr_indeterminate

731 #30\XINT_jrr_divisionbyzero

732 #10\XINT_jrr_zero

733 00\XINT_jrr_loop_a

734 \krof

735 {#3#4}{#1#2}1001%

736 }%

737 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%

738 {%

739 \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%

740 }%

741 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%

742 {%

743 \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%

744 }%

745 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08

746 \def\XINT_jrr_loop_a #1#2%

747 {%

748 \expandafter\XINT_jrr_loop_b

749 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%

750 }%

751 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%

752 {%

753 \expandafter \XINT_jrr_loop_c \expandafter

754 {\romannumeral0\xintiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%

755 {\romannumeral0\xintiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%

```

756     {#2}{#3}{#4}{#5}%
757 }%
758 \def\XINT_jrr_loop_c #1#2%
759 {%
760     \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
761 }%
762 \def\XINT_jrr_loop_d #1#2#3#4%
763 {%
764     \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
765 }%
766 \def\XINT_jrr_loop_e #1#2\Z
767 {%
768     \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
769 }%
770 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
771 {%
772     \XINT_irr_finish {#3}{#4}%
773 }%

```

8.27 \xintTFrac

1.09i, for frac in \xintexpr. And \xintFrac is already assigned. T for truncation. However, potentially not very efficient with numbers in scientific notations, with big exponents. Will have to think it again some day. I hesitated how to call the macro. Same convention as in maple, but some people reserve fractional part to $x - \text{floor}(x)$. Also, not clear if I had to make it negative (or zero) if $x < 0$, or rather always positive. There should be in fact such a thing for each rounding function, trunc, round, floor, ceil.

```

774 \def\xintTFrac {\romannumeral0\xinttfrac }%
775 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
776 \def\XINT_tfrac_fork #1%
777 {%
778     \xint_UDzerominusfork
779     #1-\XINT_tfrac_zero
780     0#1{\xinttiopp\XINT_tfrac_P }%
781     0-{\XINT_tfrac_P #1}%
782     \krof
783 }%
784 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
785 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
786     \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

8.28 \xintTrunc, \xintiTrunc

1.2i release notes: ever since its inception this macro was stupid for a decimal input: it did not handle it separately from the general fraction case $A/B[N]$ with $B > 1$, hence ended up doing divisions by powers of ten. But this meant that nesting \xintTrunc with itself was very inefficient.

1.2i version is better. However it still handles $B > 1$, $N < 0$ via adding zeros to B and dividing with this extended B. A possibly more efficient approach is implemented in \xintXTrunc, but its logic is more complicated, the code is quite longer and making it f-expandable would not shorten it... I decided for the time being to not complicate things here.

```

787 \def\xintTrunc {\romannumeral0\xinttrunc }%

```

```

788 \def\xintiTrunc {\romannumeral0\xintitrunc}%
789 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
790 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
791 \def\XINT_trunc #1.#2#3%
792 {%
793     \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
794 }%
795 \def\XINT_trunc_a #1#2#3#4.#5%
796 {%
797     \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
798     \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
799     \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
800 }%
801 \def\XINT_trunc_zero #1.#2.{ 0}%

802 \def\XINT_trunc_b      {\expandafter\XINT_trunc_B\the\numexpr}%
803 \def\XINT_trunc_sp_b   {\expandafter\XINT_trunc_sp_B\the\numexpr}%

804 \def\XINT_trunc_B #1%
805 {%
806     \xint_UDsignfork
807     #1\XINT_trunc_C
808     -\XINT_trunc_D
809     \krof #1%
810 }%

811 \def\XINT_trunc_sp_B #1%
812 {%
813     \xint_UDsignfork
814     #1\XINT_trunc_sp_C
815     -\XINT_trunc_sp_D
816     \krof #1%
817 }%

818 \def\XINT_trunc_C -#1.#2#3%
819 {%
820     \expandafter\XINT_trunc_CE
821     \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
822 }%
823 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

824 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.%}
825 \def\XINT_trunc_sp_Ca #1%
826 {%
827     \xint_UDsignfork
828     #1{\XINT_trunc_sp_Cb -}%
829     -{\XINT_trunc_sp_Cb \space#1}%
830     \krof
831 }%
832 \def\XINT_trunc_sp_Cb #1#2.#3.%
833 {%
834     \expandafter\XINT_trunc_sp_Cc

```

```

835 \romannumeral0\expandafter\XINT_split_fromright_a
836 \the\numexpr#3-\numexpr\XINT_length_loop
837 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
838 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
839 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
840 .#2\xint_bye2345678\xint_bye..#1%
841 }%

842 \def\XINT_trunc_sp_Cc #1%
843 {%
844 \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
845 \xint_orthat {\XINT_trunc_sp_Cd #1}%
846 }%
847 \def\XINT_trunc_sp_Cd #1.#2.#3%
848 {%
849 \XINT_trunc_sp_F #3#1.%
850 }%
851 \def\XINT_trunc_D #1.#2%
852 {%
853 \expandafter\XINT_trunc_E
854 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
855 }%
856 \def\XINT_trunc_sp_D #1.#2#3%
857 {%
858 \expandafter\XINT_trunc_sp_E
859 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
860 }%
861 \def\XINT_trunc_E #1%
862 {%
863 \xint_UDsignfork
864 #1{\XINT_trunc_F -}%
865 -{\XINT_trunc_F \space#1}%
866 \krof
867 }%
868 \def\XINT_trunc_sp_E #1%
869 {%
870 \xint_UDsignfork
871 #1{\XINT_trunc_sp_F -}%
872 -{\XINT_trunc_sp_F\space#1}%
873 \krof
874 }%
875 \def\XINT_trunc_F #1#2.#3#4%
876 {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
877 \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
878 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
879 \def\XINT_itrunc_G #1#2.#3#4.{\if#10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
880 \def\XINT_trunc_G #1.#2#3.%
881 {%
882 \expandafter\XINT_trunc_H
883 \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
884 }%
885 \def\XINT_trunc_H #1.#2.%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfac, xintexpr, indices

```

886 {%
887   \ifnum #1 > \xint_c_
888     \xint_afterfi {\XINT_trunc_Ha {#2}}%
889   \else
890     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
891   \fi
892}%
893 \def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
894 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
895 \def\XINT_trunc_Hb #1#2#3%
896 {%
897   \expandafter #3\expandafter0\expandafter.%

898   \romannumeral\xintreplicate{#1}0#2%
899}%

```

8.29 \xintTTrunc

1.1. Modified in 1.2i, it does simply \xintiTrunc0 with no shortcut (the latter having been modified)

```

900 \def\xintTTrunc {\romannumeral0\xintttrunc}%
901 \def\xintttrunc {\xintitrunc\xint_c_}%

```

8.30 \xintNum

```

902 \let\xintnum \xintttrunc

```

8.31 \xintRound, \xintiRound

Modified in 1.2i.

It benefits first of all from the faster \xintTrunc, particularly when the input is already a decimal number (denominator B=1).

And the rounding is now done in 1.2 style (with much delay, sorry), like of the rewritten \xintInc and \xintDec.

```

903 \def\xintRound {\romannumeral0\xintround}%
904 \def\xintiRound {\romannumeral0\xintiround}%
905 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
906 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
907 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1}%
908 \def\XINT_round_aa #1.#2.#3#4%
909 {%
910   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
911}%
912 \def\XINT_round_a #1#2#3#4.%
913 {%
914   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
915   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
916   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
917}%
918 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
919 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%

```



```

920 \def\XINT_round_B #1.%
921   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%

```

8.32 \xintXTrunc

1.09j [2014/01/06] This is completely expandable but not f-expandable. Rewritten for 1.2i (2016/12/04):

- no more use of \xintloop from xinttools.sty (replaced by \xintreplicate... from xintkernel.sty),
 - no more use in 0>N>-D case of a dummy control sequence name via \csname...\endcsname
 - handles better the case of an input already a decimal number
- Need to transfer code comments into public dtx.

```

922 \def\xintXTrunc #1#2%
923 {%
924   \expandafter\XINT_xtrunc_a
925   \the\numexpr #1\expandafter.\romannumeral0\xintra
926 }%
927 \def\XINT_xtrunc_a #1.% ?? faire autre chose
928 {%
929   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
930 }%

```

```

931 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

```

```

932 \def\XINT_xtrunc_c #1%
933 {%
934   \xint_UDzerominusfork
935   #1-\XINT_xtrunc_zero
936   0#1{-\XINT_xtrunc_d {}}%
937   0-{\XINT_xtrunc_d #1}%
938   \krof
939 }%[
940 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

```

```

941 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
942 {%
943   \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001W
944   !{#4};{#5}{#2}{#1#3}%
945 }%
946 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
947 {%
948   \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
949   \XINT_xtrunc_prepare_b #9%
950 }%
951 \def\XINT_xtrunc_prepare_small\R #1!#2;%
952 {%
953   \ifcase #2
954   \or\expandafter\XINT_xtrunc_BisOne
955   \or\expandafter\XINT_xtrunc_BisTwo
956   \or

```

```

957 \or\expandafter\XINT_xtrunc_BisFour
958 \or\expandafter\XINT_xtrunc_BisFive
959 \or
960 \or
961 \or\expandafter\XINT_xtrunc_BisEight
962 \fi\XINT_xtrunc_BisSmall {#2}%
963 }%

964 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
965 {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
966 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
967 {%
968 \expandafter\XINT_xtrunc_sp_e\expandafter
969 {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
970 {\romannumeral0\xintiimul 5{#4}{#3}}%
971 }%
972 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
973 {%
974 \expandafter\XINT_xtrunc_sp_e\expandafter
975 {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
976 {\romannumeral0\xintiimul {25}{#4}{#3}}%
977 }%
978 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
979 {%
980 \expandafter\XINT_xtrunc_sp_e\expandafter
981 {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
982 {\romannumeral0\xintdouble {#4}{#3}}%
983 }%
984 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
985 {%
986 \expandafter\XINT_xtrunc_sp_e\expandafter
987 {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
988 {\romannumeral0\xintiimul {125}{#4}{#3}}%
989 }%
990 \def\XINT_xtrunc_BisSmall #1%
991 {%
992 \expandafter\XINT_xtrunc_e\expandafter
993 {\expandafter\XINT_xtrunc_small_a
994 \the\numexpr #1/\xint_c_ii\expandafter
995 .\the\numexpr \xint_c_x^viii+#1!}%
996 }%

997 \def\XINT_xtrunc_small_a #1.#2!#3%
998 {%
999 \expandafter\XINT_div_small_b\the\numexpr #1\expandafter
1000 \xint:\the\numexpr #2\expandafter!%
1001 \romannumeral0\XINT_div_small_ba #3\R\R\R\R\R\R\R\{10}0000001\W
1002 #3\XINT_sepbyviii_Z_end 2345678\relax
1003 }%

1004 \def\XINT_xtrunc_prepare_b

```

```

1005    {\expandafter\XINT_xtrunc_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1006 \def\XINT_xtrunc_prepare_c #1!%
1007 {%
1008     \XINT_xtrunc_prepare_d #1.00000000!{#1}%
1009 }%
1010 \def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1011 {%
1012     \expandafter\XINT_xtrunc_prepare_e
1013     \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1014 }%
1015 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
1016 {%
1017     \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1018 }%
1019 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1020 {%
1021     \expandafter\XINT_xtrunc_prepare_g\expandafter
1022     \XINT_div_prepare_g
1023     \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1024     \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1025     \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1026     \xint:\romannumeral0\XINT_sepandrev_andcount
1027     #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1028         \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1029         \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1030         \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1031     \X
1032 }%

1033 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

1034 \def\XINT_xtrunc_e #1#2%
1035 {%
1036     \ifnum #2<\xint_c_
1037         \expandafter\XINT_xtrunc_I
1038     \else
1039         \expandafter\XINT_xtrunc_II
1040     \fi #2\xint:{#1}%
1041 }%

1042 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1043 {%
1044     \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1045 }%

1046 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1047 {%
1048     \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1049 }%

```

```

1050 \def\XINT_xtrunc_I_b #1%
1051 {%
1052     \xint_UDsignfork
1053     #1\XINT_xtrunc_IA_c
1054     -\XINT_xtrunc_IB_c
1055     \krof #1%
1056 }%

1057 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1058 {%
1059     \expandafter\XINT_xtrunc_IA_d
1060     \the\numexpr#2-\xintLength{#6}\xint:{#6}%
1061     \expandafter\XINT_xtrunc_IA_xd
1062     \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1063 }%

1064 \def\XINT_xtrunc_IA_d #1%
1065 {%
1066     \xint_UDsignfork
1067     #1\XINT_xtrunc_IAA_e
1068     -\XINT_xtrunc_IAB_e
1069     \krof #1%
1070 }%

1071 \def\XINT_xtrunc_IAA_e -#1\xint:#2%
1072 {%
1073     \romannumeral0\XINT_split_fromleft
1074     #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1075 }%

1076 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1077 {%
1078     0.\romannumeral\XINT_rep#1\endcsname0#2%
1079 }%

1080 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1081 {%
1082     \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1083 }%

1084 \def\XINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
1085 {%
1086     \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
1087 }%

1088 \def\XINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
1089 {%
1090     \expandafter\XINT_xtrunc_IB_d
1091     \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1092 }%

```

205

```

1140 \expandafter\XINT_xtrunc_finish_a\romannumeral0#2{#1}%
1141 }%
1142 \def\XINT_xtrunc_finish_a #1#2#3%
1143 {%
1144 \romannumeral\xintreplicate{#3-\xintLength{#1}}0#1%
1145 }%

1146 \def\XINT_xtrunc_sp_e #1%
1147 {%
1148 \ifnum #1<\xint_c_
1149 \expandafter\XINT_xtrunc_sp_I
1150 \else
1151 \expandafter\XINT_xtrunc_sp_II
1152 \fi #1\xint:%
1153 }%

1154 \def\XINT_xtrunc_sp_I -#1\xint:#2#3%
1155 {%
1156 \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
1157 }%

1158 \def\XINT_xtrunc_sp_I_a #1%
1159 {%
1160 \xint_UDsignfork
1161 #1\XINT_xtrunc_sp_IA_b
1162 -\XINT_xtrunc_sp_IB_b
1163 \krof #1%
1164 }%

1165 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1166 {%
1167 \expandafter\XINT_xtrunc_sp_IA_c
1168 \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1169 }%

1170 \def\XINT_xtrunc_sp_IA_c #1%
1171 {%
1172 \xint_UDsignfork
1173 #1\XINT_xtrunc_sp_IAA
1174 -\XINT_xtrunc_sp_IAB
1175 \krof #1%
1176 }%

1177 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1178 {%
1179 \romannumeral0\XINT_split_fromleft
1180 #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1181 }%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

```

1182 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1183 {%
1184     0.\romannumeral\XINT_rep#1\endcsname0#2%
1185 }%

1186 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1187 {%
1188     \expandafter\XINT_xtrunc_sp_IB_c
1189     \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1190 }%

1191 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1192 {%
1193     \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1194 }%

1195 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1196 {%
1197     #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1198 }%

```

8.33 \xintDigits

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

```

1199 \mathchardef\XINTdigits 16
1200 \def\xintDigits #1#2%
1201     {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1202 \def\xinttheDigits {\number\XINTdigits }%

```

8.34 \xintAdd

Big change at 1.3: $a/b+c/d$ uses $\text{lcm}(b,d)$ as denominator.

```

1203 \def\xintAdd {\romannumeral0\xintadd }%
1204 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
1205 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1206 \def\XINT_fadd_Azero #1{\xintraw }%
1207 \def\XINT_fadd_a #1/#2[#3]#4%
1208     {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1209 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1210 \def\XINT_fadd_Bzero #1[#2#3#4{ #3/#4[#2]}%
1211 \def\XINT_fadd_c #1/#2[#3]#4%
1212 {%
1213     \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1214 }%
1215 \def\XINT_fadd_Aa #1%
1216 {%
1217     \xint_UDzerominusfork
1218     #1-\XINT_fadd_B

```

```

1219      0#1\XINT_fadd_Bb
1220      0-\XINT_fadd_Ba
1221      \krof #1%
1222 }%
1223 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}{#3}}%
1224 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1225 {%
1226     \expandafter\XINT_fadd_C\expandafter
1227     {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1228     {#7}{#5}{#4}{#2}%
1229 }%
1230 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1231 {%
1232     \expandafter\XINT_fadd_C\expandafter
1233     {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1234     {#5}{#7}{#6}{#3}%
1235 }%
1236 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine1?
1237 \def\XINT_fadd_C #1#2#3%
1238 {%
1239     \expandafter\XINT_fadd_D_b
1240     \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1241 }%

```

Basically a clone of the \XINT_irr_loop_a loop. I should modify the output of \XINT_div_prepare perhaps to be optimized for checking if remainder vanishes.

```

1242 \def\XINT_fadd_D_a #1#2%
1243 {%
1244     \expandafter\XINT_fadd_D_b
1245     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1246 }%
1247 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1248 \def\XINT_fadd_D_c #1#2\Z
1249 {%
1250     \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1251 }%
1252 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1253 {%
1254     \expandafter\XINT_fadd_E
1255     \romannumeral0\xintiipro {#3}{#2}.{#2}%
1256 }%
1257 \def\XINT_fadd_E #1.#2#3%
1258 {%
1259     \expandafter\XINT_fadd_F
1260     \romannumeral0\xintiimul{#1}{#3}.{\xintiipro{#3}{#2}}{#1}%
1261 }%
1262 \def\XINT_fadd_F #1.#2#3#4#5%
1263 {%
1264     \expandafter\XINT_fadd_G
1265     \romannumeral0\xintiiaadd{\xintiimul{#2}{#4}}{\xintiimul{#3}{#5}}/#1%
1266 }%
1267 \def\XINT_fadd_G #1{%

```



```
1268 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1269 }\XINT_fadd_G{ }%
```

8.35 \xintSub

Since 1.3 will use least common multiple of denominators.

```
1270 \def\xintSub {\romannumeral0\xintsub }%
1271 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra {#1}}%
1272 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1273 \def\XINT_fsub_Azero #1{\xintopp }%
1274 \def\XINT_fsub_a #1/#2[#3]#4%
1275     {\expandafter\XINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1276 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1277     #1-\XINT_fadd_Bzero
1278     0#1\XINT_fadd_c
1279     0-\XINT_fadd_c -#1}%
1280 \krof }%
```

8.36 \xintSum

There was (not documented anymore since 1.09d, 2013/10/22) a macro \xintSumExpr, but it has been deleted at 1.2l.

Empty items are not accepted by this macro.

```
1281 \def\xintSum {\romannumeral0\xintsum }%
1282 \def\xintsum #1{\expandafter\XINT_fsumexpr\romannumeral`&&#1\xint:}%
1283 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1284 \def\XINT_fsum_loop_a #1#2%
1285 {%
1286     \expandafter\XINT_fsum_loop_b \romannumeral`&&#2\xint:{#1}%
1287 }%
1288 \def\XINT_fsum_loop_b #1%
1289 {%
1290     \xint_gob_til_xint: #1\XINT_fsum_finished\xint:\XINT_fsum_loop_c #1%
1291 }%
1292 \def\XINT_fsum_loop_c #1\xint:#2%
1293 {%
1294     \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1295 }%
1296 \def\XINT_fsum_finished #1\xint:\xint:#2{ #2}%
```

8.37 \xintMul

```
1297 \def\xintMul {\romannumeral0\xintmul }%
1298 \def\xintmul #1{\expandafter\XINT_fmula\romannumeral0\xintra {#1}.}%
1299 \def\XINT_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%
1300 \def\XINT_fmula_a #1[#2].#3%
1301     {\expandafter\XINT_fmula_b\romannumeral0\xintra {#3}#1[#2.]}%
1302 \def\XINT_fmula_b #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_c #1}%
1303 \def\XINT_fmula_c #1/#2[#3]#4/#5[#6.]%
1304 {%
1305     \expandafter\XINT_fmula_d
```

```

1306 \expandafter{\the\numexpr #3+#6\expandafter}%
1307 \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1308 {\romannumeral0\xintiimul {#4}{#1}}%
1309 }%
1310 \def\XINT_fmulo_d #1#2#3%
1311 {%
1312 \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1313 }%
1314 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1315 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

8.38 \xintSqr

1.1 modifs comme xintMul.

```

1316 \def\xintSqr {\romannumeral0\xintsqr }%
1317 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
1318 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1319 \def\XINT_fsqr_a #1/#2[#3]%
1320 {%
1321 \expandafter\XINT_fsqr_b
1322 \expandafter{\the\numexpr #3+#3\expandafter}%
1323 \expandafter{\romannumeral0\xintiisqr {#2}}%
1324 {\romannumeral0\xintiisqr {#1}}%
1325 }%
1326 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1327 \def\XINT_fsqr_zero #1{ 0/1[0]}%

```

8.39 \xintPow

1.2f: to be coherent with the "i" convention \xintiPow should parse also its exponent via \xintNum when xintfrac.sty is loaded. This was not the case so far. Cependant le problème est que le fait d'appliquer \xintNum rend impossible certains inputs qui auraient pu être gérés par \numexpr. Le \numexpr externe est ici pour intercepter trop grand input.

```

1328 \def\xintipow #1#2%
1329 {%
1330 \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1331 .\romannumeral0\xintnum{#1}\xint:
1332 }%
1333 \def\xintPow {\romannumeral0\xintpow }%
1334 \def\xintpow #1%
1335 {%
1336 \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1337 }%
1338 \def\XINT_fpow #1#2%
1339 {%
1340 \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1341 }%
1342 \def\XINT_fpow_fork #1#2\Z
1343 {%
1344 \xint_UDzerominusfork
1345 #1-\XINT_fpow_zero

```

```

1346      0#1\XINT_fpow_neg
1347      0-{\XINT_fpow_pos #1}%
1348      \krof
1349      {#2}%
1350 }%
1351 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1352 \def\XINT_fpow_pos #1#2#3#4#5%
1353 {%
1354     \expandafter\XINT_fpow_pos_A\expandafter
1355     {\the\numexpr #1#2*#3\expandafter}\expandafter
1356     {\romannumeral0\xintiipow {#5}{#1#2}}%
1357     {\romannumeral0\xintiipow {#4}{#1#2}}%
1358 }%
1359 \def\XINT_fpow_neg #1#2#3#4%
1360 {%
1361     \expandafter\XINT_fpow_pos_A\expandafter
1362     {\the\numexpr -#1*#2\expandafter}\expandafter
1363     {\romannumeral0\xintiipow {#3}{#1}}%
1364     {\romannumeral0\xintiipow {#4}{#1}}%
1365 }%
1366 \def\XINT_fpow_pos_A #1#2#3%
1367 {%
1368     \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1369 }%
1370 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

8.40 \xintFac

Factorial coefficients: variant which can be chained with other `xintfrac` macros. `\xintiFac` deprecated at 1.2o and removed at 1.3; `\xintFac` used by `xintexpr.sty`.

```

1371 \def\xintFac {\romannumeral0\xintfac}%
1372 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

8.41 \xintBinomial

1.2f. Binomial coefficients. `\xintiBinomial` deprecated at 1.2o and removed at 1.3; `\xintBinomial` needed by `xintexpr.sty`.

```

1373 \def\xintBinomial {\romannumeral0\xintbinomial}%
1374 \def\xintbinomial #1#2%
1375 {%
1376     \expandafter\XINT_binom_pre
1377     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1378 }%

```

8.42 \xintPFactorial

1.2f. Partial factorial. For needs of `xintexpr.sty`.

```

1379 \def\xintipfactorial #1#2%
1380 {%
1381     \expandafter\XINT_pfac_fork

```

```

1382 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1383 }%
1384 \def\xintPfactorial {\romannumeral0\xintpfactorial}%
1385 \def\xintpfactorial #1#2%
1386 {%
1387 \expandafter\XINT_pfac_fork
1388 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1389 }%

```

8.43 \xintPrd

There was (not documented anymore since 1.09d, 2013/10/22) a macro \xintPrdExpr, but it has been deleted at 1.21

```

1390 \def\xintPrd {\romannumeral0\xintprd}%
1391 \def\xintprd #1{\expandafter\XINT_fprdexpr \romannumeral`&&@#1\xint:}%
1392 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1393 \def\XINT_fprod_loop_a #1#2%
1394 {%
1395 \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\xint:{#1}%
1396 }%
1397 \def\XINT_fprod_loop_b #1%
1398 {%
1399 \xint_gob_til_xint: #1\XINT_fprod_finished\xint:\XINT_fprod_loop_c #1%
1400 }%
1401 \def\XINT_fprod_loop_c #1\xint:#2%
1402 {%
1403 \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1404 }%
1405 \def\XINT_fprod_finished#1\xint:\xint:#2{ #2}%

```

8.44 \xintDiv

```

1406 \def\xintDiv {\romannumeral0\xintdiv}%
1407 \def\xintdiv #1%
1408 {%
1409 \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1410 }%
1411 \def\XINT_fdiv #1#2%
1412 {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1413 \def\XINT_fdiv_A #1#2#3#4#5#6%
1414 {%
1415 \expandafter\XINT_fdiv_B
1416 \expandafter{\the\numexpr #4-#1\expandafter}%
1417 \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1418 {\romannumeral0\xintiimul {#3}{#5}}%
1419 }%
1420 \def\XINT_fdiv_B #1#2#3%
1421 {%
1422 \expandafter\XINT_fdiv_C
1423 \expandafter{#3}{#1}{#2}%
1424 }%
1425 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

8.45 \xintDivFloor

1.1. Changed at 1.2p to not append /1[0] ending but rather output a big integer in strict format, like \xintDivTrunc and \xintDivRound.

```
1426 \def\xintDivFloor      {\romannumeral0\xintdivfloor }%
1427 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%
```

8.46 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```
1428 \def\xintDivTrunc      {\romannumeral0\xintdivtrunc }%
1429 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

8.47 \xintDivRound

1.1

```
1430 \def\xintDivRound      {\romannumeral0\xintdivround }%
1431 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

8.48 \xintModTrunc

1.1. \xintModTrunc {q1}{q2} computes $q_1 - q_2 * t(q_1/q_2)$ with $t(q_1/q_2)$ equal to the truncated division of two fractions q_1 and q_2 .

Its former name, prior to 1.2p, was \xintMod.

At 1.3, uses least common multiple denominator, like \xintMod (next).

```
1432 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1433 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1434 \def\XINT_modtrunc_a #1#2.#3%
1435   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1436 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1437 {%
1438   \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1439   \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1440   \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1441   \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1442 }%
1443 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1444 {%
1445   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{0/1[0]}%
1446 }%
1447 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
1448 \def\XINT_modtrunc_bneg #1%
1449 {%
1450   \xint_UDsignfork
1451     #1{\xintiopp\XINT_modtrunc_pos {}}%
1452     -{\XINT_modtrunc_pos #1}%
1453   \krof
1454 }%
1455 \def\XINT_modtrunc_bpos #1%
```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

```

1456 {%
1457   \xint_UDsignfork
1458       #1{\xintiiopp\XINT_modtrunc_pos {}}%
1459       -{\XINT_modtrunc_pos #1}%
1460   \krof
1461 }%

    Attention. This crucially uses that xint's \xintiiE{x}{e} is defined to return x unchanged if e
    is negative (and x extended by e zeroes if e >= 0).

1462 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1463 {%
1464   \expandafter\XINT_modtrunc_pos_a
1465   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1466   \romannumeral0\expandafter\XINT_mod_D_b
1467   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1468   {#1#5}{#7-#4}{#2}{#4-#7}%
1469 }%
1470 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

8.49 \xintDivMod

1.2p. `\xintDivMod{q1}{q2}` outputs `{floor(q1/q2)}{q1 - q2*floor(q1/q2)}`. Attention that it relies on `\xintiiE{x}{e}` returning x if e < 0.

Modified (like `\xintAdd` and `\xintSub`) at 1.3 to use a l.c.m for final denominator of the "mod" part.

```

1471 \def\xintDivMod {\romannumeral0\xintdivmod }%
1472 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1473 \def\XINT_divmod_a #1#2.#3%
1474   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1475 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1476 {%
1477   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1478   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1479   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1480   \xint_orthat{\XINT_divmod_bpos #1#2}%
1481 }%
1482 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1483 {%
1484   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
1485   {{0}{0/1[0]}}% à revoir...
1486 }%
1487 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1488 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)
1489 {%
1490   \expandafter\XINT_divmod_bneg_finish
1491   \romannumeral0\xint_UDsignfork
1492       #1{\XINT_divmod_bpos {}}%
1493       -{\XINT_divmod_bpos {-#1}}%
1494   \krof
1495 }%
1496 \def\XINT_divmod_bneg_finish#1#2%

```

```

1497 {%
1498   \expandafter\xint_exchangetwo_keepbraces\expandafter
1499   {\romannumeral0\xintiioopp#2}{#1}%
1500 }%
1501 \def\xINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1502 {%
1503   \expandafter\xINT_divmod_bpos_a
1504   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1505   \romannumeral0\expandafter\xINT_mod_D_b
1506   \romannumeral0\xINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1507   {#1#5}{#7-#4}{#2}{#4-#7}%
1508 }%
1509 \def\xINT_divmod_bpos_a #1.#2#3#4%
1510 {%
1511   \expandafter\xINT_divmod_bpos_finish
1512   \romannumeral0\xintiidiivision{#3}{#4}{/#2[#1]}%
1513 }%
1514 \def\xINT_divmod_bpos_finish #1#2#3{#1}{#2#3}%

```

8.50 \xintMod

1.2p. `\xintMod{q1}{q2}` computes $q1 - q2 \cdot \text{floor}(q1/q2)$. Attention that it relies on `\xintiiE{x}{e}` returning x if $e < 0$.

Prior to 1.2p, that macro had the meaning now attributed to `\xintModTrunc`.

Modified (like `\xintAdd` and `\xintSub`) at 1.3 to use a l.c.m for final denominator.

```

1515 \def\xintMod {\romannumeral0\xintmod }%
1516 \def\xintmod #1{\expandafter\xINT_mod_a\romannumeral0\xintra{#1}.}%
1517 \def\xINT_mod_a #1#2.#3%
1518   {\expandafter\xINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1519 \def\xINT_mod_b #1#2% #1 de A, #2 de B.
1520 {%
1521   \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1522   \if0#1\xint_dothis\xINT_mod_aiszero\fi
1523   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1524   \xint_orthat{\XINT_mod_bpos #1#2}%
1525 }%

```

Attention to not move `ModTrunc` code beyond that point.

```

1526 \let\xINT_mod_divbyzero\xINT_modtrunc_divbyzero
1527 \let\xINT_mod_aiszero \XINT_modtrunc_aiszero
1528 \def\xINT_mod_bneg #1% f % -g = -((-f) % g), for g > 0
1529 {%
1530   \xintiioopp\xint_UDsignfork
1531   #1{\XINT_mod_bpos {}}%
1532   -{\XINT_mod_bpos {-#1}}%
1533   \krof
1534 }%
1535 \def\xINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1536 {%
1537   \expandafter\xINT_mod_bpos_a
1538   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%

```

```

1539 \romannumeral0\expandafter\XINT_mod_D_b
1540 \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1541 {#1#5}{#7-#4}{#2}{#4-#7}%
1542 }%
1543 \def\XINT_mod_D_a #1#2%
1544 {%
1545 \expandafter\XINT_mod_D_b
1546 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1547 }%
1548 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1549 \def\XINT_mod_D_c #1#2\Z
1550 {%
1551 \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1552 }%
1553 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1554 {%
1555 \expandafter\XINT_mod_E
1556 \romannumeral0\xintiico {#3}{#2}.{#2}%
1557 }%
1558 \def\XINT_mod_E #1.#2#3%
1559 {%
1560 \expandafter\XINT_mod_F
1561 \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
1562 }%
1563 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1564 {%
1565 {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
1566 {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
1567 }%
1568 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiiirem {#3}{#4}/#2[#1]}%

```

8.51 \xintIsOne

New with 1.09a. Could be more efficient. For fractions with big powers of tens, it is better to use `\xintCmp{f}{1}`. Restyled in 1.09i.

```

1569 \def\xintIsOne {\romannumeral0\xintisone }%
1570 \def\xintisone #1{\expandafter\XINT_fracisone
1571 \romannumeral0\xintrawwithzeros{#1}\Z }%
1572 \def\XINT_fracisone #1/#2\Z
1573 {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

8.52 \xintGeq

```

1574 \def\xintGeq {\romannumeral0\xintgeq }%
1575 \def\xintgeq #1%
1576 {%
1577 \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1578 }%
1579 \def\XINT_fgeq #1#2%
1580 {%
1581 \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1582 }%

```



```

1583 \def\XINT_fgeq_A #1%
1584 {%
1585     \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1586     \XINT_fgeq_B #1%
1587 }%
1588 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1589 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1590 {%
1591     \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1592     \expandafter\XINT_fgeq_C\expandafter
1593     {\the\numexpr #7-#3\expandafter}\expandafter
1594     {\romannumeral0\xintiimul {#4#5}{#2}}%
1595     {\romannumeral0\xintiimul {#6}{#1}}%
1596 }%
1597 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1598 \def\XINT_fgeq_C #1#2#3%
1599 {%
1600     \expandafter\XINT_fgeq_D\expandafter
1601     {#3}{#1}{#2}%
1602 }%
1603 \def\XINT_fgeq_D #1#2#3%
1604 {%
1605     \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1606     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1607     { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1608 }%
1609 \def\XINT_fgeq_E #1%
1610 {%
1611     \xint_UDsignfork
1612     #1\XINT_fgeq_Fd
1613     -{\XINT_fgeq_Fn #1}%
1614     \krof
1615 }%

1616 \def\XINT_fgeq_Fd #1\Z #2#3%
1617 {%
1618     \expandafter\XINT_fgeq_Fe
1619     \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1620 }%
1621 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1622 \def\XINT_fgeq_Fn #1\Z #2#3%
1623 {%
1624     \expandafter\XINT_fgeq_Fo
1625     \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1626 }%
1627 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%

```

8.53 \xintMax

```

1628 \def\xintMax {\romannumeral0\xintmax }%
1629 \def\xintmax #1%
1630 {%
1631     \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1632 }%

```

```

1633 \def\XINT_fmax #1#2%
1634 {%
1635     \expandafter\XINT_fmax_A\romannumeral0\xintra {#2}#1%
1636 }%
1637 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1638 {%
1639     \xint_UDsignsfork
1640     #1#5\XINT_fmax_minusminus
1641     -#5\XINT_fmax_firstneg
1642     #1-\XINT_fmax_secondneg
1643     --\XINT_fmax_nonneg_a
1644     \krof
1645     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1646 }%
1647 \def\XINT_fmax_minusminus --%
1648     {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1649 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1650 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1651 \def\XINT_fmax_nonneg_a #1#2#3#4%
1652 {%
1653     \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1654 }%
1655 \def\XINT_fmax_nonneg_b #1#2%
1656 {%
1657     \if0\romannumeral0\XINT_fgeq_A #1#2%
1658         \xint_afterfi{ #1}%
1659     \else \xint_afterfi{ #2}%
1660     \fi
1661 }%

```

8.54 \xintMaxof

1.2l protects \xintMaxof against items with non terminated \the\numexpr expressions.
The macro is not compatible with an empty list.

```

1662 \def\xintMaxof {\romannumeral0\xintmaxof }%
1663 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\xint:}%
1664 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}!}%
1665 \def\XINT_maxof_b #1!#2%
1666     {\expandafter\XINT_maxof_c\romannumeral`&&@#2!{#1}!}%
1667 \def\XINT_maxof_c #1%
1668     {\xint_gob_til_xint: #1\XINT_maxof_e\xint:\XINT_maxof_d #1}%
1669 \def\XINT_maxof_d #1!%
1670     {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1671 \def\XINT_maxof_e #1!#2!{ #2}%

```

8.55 \xintMin

```

1672 \def\xintMin {\romannumeral0\xintmin }%
1673 \def\xintmin #1%
1674 {%
1675     \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra {#1}}%
1676 }%
1677 \def\XINT_fmin #1#2%

```

```

1678 {%
1679   \expandafter\XINT_fmin_A\romannumeral0\xintra {#2}#1%
1680 }%
1681 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1682 {%
1683   \xint_UDsignsfork
1684     #1#5\XINT_fmin_minusminus
1685     -#5\XINT_fmin_firstneg
1686     #1-\XINT_fmin_secondneg
1687     --\XINT_fmin_nonneg_a
1688   \krof
1689   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1690 }%
1691 \def\XINT_fmin_minusminus --%
1692   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1693 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1694 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1695 \def\XINT_fmin_nonneg_a #1#2#3#4%
1696 {%
1697   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1698 }%
1699 \def\XINT_fmin_nonneg_b #1#2%
1700 {%
1701   \if0\romannumeral0\XINT_fgeq_A #1#2%
1702     \xint_afterfi{ #2}%
1703   \else \xint_afterfi{ #1}%
1704   \fi
1705 }%

```

8.56 \xintMinof

1.21 protects \xintMinof against items with non terminated \the\numexpr expressions.
The macro is not compatible with an empty list.

```

1706 \def\xintMinof      {\romannumeral0\xintminof }%
1707 \def\xintminof      #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\xint:}%
1708 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}!}%
1709 \def\XINT_minof_b #1!#2%
1710   {\expandafter\XINT_minof_c\romannumeral`&&@#2!{#1}!}%
1711 \def\XINT_minof_c #1%
1712   {\xint_gob_til_xint: #1\XINT_minof_e\xint:\XINT_minof_d #1}%
1713 \def\XINT_minof_d #1!%
1714   {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1715 \def\XINT_minof_e #1!#2!{ #2}%

```

8.57 \xintCmp

```

1716 \def\xintCmp {\romannumeral0\xintcmp }%
1717 \def\xintcmp #1%
1718 {%
1719   \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintra {#1}}%
1720 }%
1721 \def\XINT_fcmp #1#2%
1722 {%

```

```

1723 \expandafter\XINT_fcmp_A\romannumeral0\xintra {#2}#1%
1724 }%
1725 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1726 {%
1727 \xint_UDsignsfork
1728 #1#5\XINT_fcmp_minusminus
1729 -#5\XINT_fcmp_firstneg
1730 #1-\XINT_fcmp_secondneg
1731 --\XINT_fcmp_nonneg_a
1732 \krof
1733 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1734 }%
1735 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1736 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1737 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1738 \def\XINT_fcmp_nonneg_a #1#2%
1739 {%
1740 \xint_UDzerosfork
1741 #1#2\XINT_fcmp_zerozero
1742 0#2\XINT_fcmp_firstzero
1743 #10\XINT_fcmp_secondzero
1744 00\XINT_fcmp_pos
1745 \krof
1746 #1#2%
1747 }%
1748 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1749 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1750 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1751 \def\XINT_fcmp_pos #1#2#3#4%
1752 {%
1753 \XINT_fcmp_B #1#3#2#4%
1754 }%
1755 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1756 {%
1757 \expandafter\XINT_fcmp_C\expandafter
1758 {\the\numexpr #6-#3\expandafter}\expandafter
1759 {\romannumeral0\xintiimul {#4}{#2}}%
1760 {\romannumeral0\xintiimul {#5}{#1}}%
1761 }%
1762 \def\XINT_fcmp_C #1#2#3%
1763 {%
1764 \expandafter\XINT_fcmp_D\expandafter
1765 {#3}{#1}{#2}%
1766 }%
1767 \def\XINT_fcmp_D #1#2#3%
1768 {%
1769 \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1770 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1771 { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1772 }%
1773 \def\XINT_fcmp_E #1%
1774 {%

```

```

1775 \xint_UDsignfork
1776 #1\XINT_fcmp_Fd
1777 -{\XINT_fcmp_Fn #1}%
1778 \krof
1779 }%

1780 \def\XINT_fcmp_Fd #1\Z #2#3%
1781 {%
1782 \expandafter\XINT_fcmp_Fe
1783 \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1784 }%
1785 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1786 \def\XINT_fcmp_Fn #1\Z #2#3%
1787 {%
1788 \expandafter\XINT_fcmp_Fo
1789 \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1790 }%
1791 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

8.58 \xintAbs

```

1792 \def\xintAbs {\romannumeral0\xintabs }%
1793 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintra {#1}}%

```

8.59 \xintOpp

```

1794 \def\xintOpp {\romannumeral0\xintopp }%
1795 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintra {#1}}%

```

8.60 \xintSgn

```

1796 \def\xintSgn {\romannumeral0\xintsqn }%
1797 \def\xintsqn #1{\expandafter\XINT_sgn\romannumeral0\xintra {#1}\xint:}%

```

8.61 Floating point macros

For a long time the float routines dating back to releases 1.07/1.08a (May-June 2013) were not modified.

Since 1.2f (March 2016) the four operations first round their arguments to `\xinttheDigits-floats` (or `P-floats`), not `(\xinttheDigits+2)-floats` or `(P+2)-floats` as was the case with earlier releases.

The four operations addition, subtraction, multiplication, division have always produced the correct rounding of the theoretical exact value to `P` or `\xinttheDigits` digits when the inputs are decimal numbers with at most `P` digits, and arbitrary decimal exponent part.

From 1.08a to 1.2j, `\xintFloat` (and `\XINTinFloat` which is used to parse inputs to other float macros) handled a fractional input `A/B` via an initial replacement to `A'/B'` where `A'` and `B'` were `A` and `B` truncated to `Q+2` digits (where asked-for precision is `Q`), and then they correctly rounded `AQ/B'` to `Q` digits. But this meant that this rounding of the input could differ (by up to one unit in the last place) from the correct rounding of the original `A/B` to the asked-for number of digits (which until 1.2f in uses as auxiliary to the macros for the basic operations was 2 more than the prevailing precision).

Since 1.2k all inputs are correctly rounded to the asked-for number of digits (this was, I think, the case in the 1.07 release -- there are no code comments -- but was, afaicr, not very efficiently done, and this is why the 1.08a release opted for truncation of the numerator and denominator.)

Notice that in float expressions, the `/` is treated as operator, hence the above discussion makes a difference only for the special input form `qfloat(A/B)` or for an `\xintexpr A/B\relax` embedded in the float expression, with `A` or `B` having more digits than the prevailing float precision.

Internally there is no inner representation of P-floats as such !!!!!

The input parser will again compute the length of the mantissa on each use !!! This is obviously something that must be improved upon before implementation of higher functions.

Currently, special tricks are used to quickly recognize inputs having no denominators, or fractions whose numerators and denominators are not too long compared to the target precision P, and in particular P-floats or quotients of two such.

Another long-standing issue is that float multiplication will first compute the 2P or 2P-1 digits of the exact product, and then round it to P digits. This is sub-optimal for large P particularly as the multiplication algorithm is basically the schoolbook one, hence worse than quadratic in the \TeX implementation which has extra cost of fetching long sequences of tokens.

8.62 \xintFloat

1.2f and 1.2g brought some refactoring which resulted in faster treatment of decimal inputs. 1.2i dropped use of some old routines dating back to pre 1.2 era in favor of more modern \xintDSRr for rounding. Then 1.2k improves again the handling of denominators B with few digits.

But the main change with 1.2k is a complete rewrite of the B>1 case in order to achieve again correct rounding in all cases.

The original version from 1.07 (May 2013) computed the exact rounding to P digits for all inputs. But from 1.08 on (June 2013), the macro handled A/B input by first truncating both A and B to at most P+2 digits. This meant that decimal input (arbitrarily long, with scientific part) was correctly rounded, but in case of fractional input there could be up to 0.6 unit in the last place difference of the produced rounding to the input, hence the output could differ from the correct rounding.

Example with 16 digits (the default): \xintFloat {1/17597472569900621233}

with xintfrac 1.07: 5.682634230727187e-20

with xintfrac 1.08b--1.2j: 5.682634230727188e-20

with xintfrac 1.2k: 5.682634230727187e-20

The exact value is 5.682634230727187499924124...e-20, showing that 1.07 and 1.2k produce the correct rounding.

Currently the code ends in a more costly branch in about 1 case among 500, where it does some extra operations (a multiplication in particular). There is a free parameter delta (here set at 4), I have yet to make some numerical explorations, to see if it could be favorable to set it to a higher value (with delta=5, there is only 1 exceptional case in 5000, etc...).

I have always hesitated about the policy of printing 10.00...0 in case of rounding upwards to the next power of ten. Already since 1.2f \XINTinFloat always produced a mantissa with exactly P digits (except for the zero value). Starting with 1.2k, \xintFloat drops this habit of printing 10.00...0 in such cases. Side note: the rounding-up detection worked when the input A/B was with numerator A and denominator B having each less than P+2 digits, or with B=1, else, it could happen that the output was a power of ten but not detected to be a rounding up of the original fraction. The value was ok, but printed 1.0...0eN with P-1 zeroes, not 10.0...0e(N-1).

I decided it was not worth the effort to enhance the algorithm to detect with 100% fiability all cases of rounding up to next power of ten, hence 1.2k dropped this.

To avoid duplication of code, and any extra burden on \XINTinFloat, which is the macro used internally by the float macros for parsing their inputs, we simply make now \xintFloat a wrapper of \XINTinFloat.

```
1798 \def\xintFloat    {\romannumeral0\xintfloat }%
1799 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
1800 \def\XINT_float_chkopt #1%
1801 {%
```

```

1802    \ifx [#1\expandafter\XINT_float_opt
1803        \else\expandafter\XINT_float_noopt
1804    \fi #1%
1805 }%
1806 \def\XINT_float_noopt #1\xint:%
1807 {%
1808     \expandafter\XINT_float_post
1809     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
1810 }%
1811 \def\XINT_float_opt [\xint:#1]%
1812 {%
1813     \expandafter\XINT_float_opt_a\the\numexpr #1.%
1814 }%
1815 \def\XINT_float_opt_a #1.#2%
1816 {%
1817     \expandafter\XINT_float_post
1818     \romannumeral0\XINTinfloat[#1]{#2}#1.%
1819 }%
1820 \def\XINT_float_post #1%
1821 {%
1822     \xint_UDzerominusfork
1823     #1-\XINT_float_zero
1824     0#1\XINT_float_neg
1825     0-\XINT_float_pos
1826     \krof #1%
1827 }%[
1828 \def\XINT_float_zero #1]#2.{ 0.e0}%
1829 \def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
1830 \def\XINT_float_pos #1#2[#3]#4.%
1831 {%
1832     \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
1833 }%
1834 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

8.63 \XINTinFloat, \XINTinFloatS

This routine is like `\xintFloat` but produces an output of the shape `A[N]` which is then parsed faster as input to other float macros. Float operations in `\xintfloatexpr...` relax use internally this format.

It must be used in form `\XINTinFloat[P]{f}`: the optional `[P]` is mandatory.

Since 1.2f, the mantissa always has exactly `P` digits even in case of rounding up to next power of ten. This simplifies other routines.

1.2g added a variant `\XINTinFloatS` which, in case of decimal input with less than the asked for precision `P` will not add extra zeros to the mantissa. For example it may output `2[0]` even if `P=500`, rather than the canonical representation `200...000[-499]`. This is how `\xintFloatMul` and `\xintFloatDiv` parse their inputs, which speeds-up follow-up processing. But `\xintFloatAdd` and `\xintFloatSub` still use `\XINTinFloat` for parsing their inputs; anyway this will have to be changed again when inner structure will carry upfront at least the length of mantissa as data.

Each time `\XINTinFloat` is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of...

something like `<letterP><length of mantissa>.mantissa.exponent`, etc... not yet.

Since 1.2k, `\XINTinFloat` always correctly rounds its argument, even if it is a fraction with

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfac](#), [xintexpr](#), [indices](#)

very big numerator and denominator. See the discussion of `\xintFloat`.

```
1835 \def\xINTinFloat {\romannumeral0\xINTinfloat}%
1836 \def\xINTinfloat
1837   {\expandafter\xINT_infloat_clean\romannumeral0\xINT_infloat}%
1838 \def\xINT_infloat_clean #1%
1839   {\if #1!\xint_dothis\xINT_infloat_clean_a\fi\xint_orthat{ }#1}%
```

Ici on ajoute les zeros pour faire exactement avec P chiffres. Car le #1 = P - L avec L la longueur de #2, (ou de abs(#2), ici le #2 peut avoir un signe) qui est < P

```
1840 \def\xINT_infloat_clean_a !#1.#2[#3]%
1841 {%
1842   \expandafter\xINT_infloat_done
1843   \the\numexpr #3-#1\expandafter.%
1844   \romannumeral0\xINT_dsx_addzeros {#1}#2;;%
1845}%
1846 \def\xINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
1847 \def\xINTinFloatS {\romannumeral0\xINTinfloatS}%
1848 \def\xINTinfloatS
1849   {\expandafter\xINT_infloatS_clean\romannumeral0\xINT_infloat}%
1850 \def\xINT_infloatS_clean #1%
1851   {\if #1!\xint_dothis\xINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
1852 \def\xINT_infloatS_clean_a !#1.{ }%
```

début de la routine proprement dite, l'argument optionnel est obligatoire.

```
1853 \def\xINT_infloat [#1]#2%
1854 {%
1855   \expandafter\xINT_infloat_a\the\numexpr #1\expandafter.%
1856   \romannumeral0\xINT_infrac {#2}%
1857}%

```

#1=P, #2=n, #3=A, #4=B.

```
1858 \def\xINT_infloat_a #1.#2#3#4%
1859 {%
```

micro boost au lieu d'utiliser `\XINT_isOne{#4}`, mais pas bon style.

```
1860   \if1\xINT_is_One#4XY%
1861     \expandafter\xINT_infloat_sp
1862   \else\expandafter\xINT_infloat_fork
1863     \fi #3.{#1}{#2}{#4}%
1864}%

```

Special quick treatment of B=1 case (1.2f then again 1.2g.)
maintenant: A.{P}{N}{1} Il est possible que A soit nul.

```
1865 \def\xINT_infloat_sp #1%
1866 {%
1867   \xint_UDzerominusfork
1868   #1-\xINT_infloat_spzero
```



```
1869      0#1\XINT_infloat_spneg
1870      0-\XINT_infloat_sppos
1871      \krof #1%
1872 }%
```

Attention surtout pas 0/1[0] ici.

```
1873 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
1874 \def\XINT_infloat_spneg-%
1875      {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
1876 \def\XINT_infloat_spnegend #1%
1877      {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
1878 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%
```

in: A.{P}{N}{1}
out: P-L.A.P.N.

```
1879 \def\XINT_infloat_sppos #1.#2#3#4%
1880 {%
1881      \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1882 }%
```

#1= P-L. Si c'est positif ou nul il faut retrancher #1 à l'exposant, et ajouter autant de zéros. On regarde premier token. P-L.A.P.N.

```
1883 \def\XINT_infloat_sp_b #1%
1884 {%
1885      \xint_UDzerominusfork
1886      #1-\XINT_infloat_sp_quick
1887      0#1\XINT_infloat_sp_c
1888      0-\XINT_infloat_sp_needzeros
1889      \krof #1%
1890 }%
```

Ici P=L. Le cas usuel dans \xintfloatexpr.

```
1891 \def\XINT_infloat_sp_quick 0.#1.#2.#3.{ #1[#3]}%
```

Ici #1=P-L est >0. L'exposant sera N-(P-L). #2=A. #3=P. #4=N.
18 mars 2016. En fait dans certains contextes il est sous-optimal d'ajouter les zéros. Par exemple quand c'est appelé par la multiplication ou la division, c'est idiot de convertir 2 en 200000...00000[-499]. Donc je redéfinit addzeros en needzeroes. Si on appelle sous la forme \XINTinFloatS, on ne fait pas l'addition de zeros.

```
1892 \def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

L-P=#1.A=#2#3.P=#4.N=#5.

Ici P<L. Il va falloir arrondir. Attention si on va à la puissance de 10 suivante. En #1 on a L-P qui est >0. L'exposant final sera N+L-P, sauf dans le cas spécial, il sera alors N+L-P+1. L'ajustement final est fait par \XINT_infloat_Y.

```
1893 \def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
1894 {%
1895      \expandafter\XINT_infloat_Y
1896      \the\numexpr #5+#1\expandafter.%
1897      \romannumeral0\expandafter\XINT_infloat_sp_round
```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

1898 \romannumeral0\XINT_split_fromleft
1899 (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
1900 }%
1901 \def\XINT_infloat_sp_round #1.#2.%
1902 {%
1903 \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
1904 }%

```

General branch for A/B with B>1 inputs. It achieves correct rounding always since 1.2k (done January 2, 2017.) This branch is never taken for A=0 because \XINT_infrac will have returned B=1 then.

```

1905 \def\XINT_infloat_fork #1%
1906 {%
1907 \xint_UDsignfork
1908 #1\XINT_infloat_J
1909 -\XINT_infloat_K
1910 \krof #1%
1911 }%
1912 \def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%

```

A.{P}{n}{B} avec B>1.

```

1913 \def\XINT_infloat_K #1.#2%
1914 {%
1915 \expandafter\XINT_infloat_L
1916 \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
1917 }%

```

|A|.P+4.{A}{P}{n}{B}. We check if A already has length <= P+4.

```

1918 \def\XINT_infloat_L #1.#2.%
1919 {%
1920 \ifnum #1>#2
1921 \expandafter\XINT_infloat_Ma
1922 \else
1923 \expandafter\XINT_infloat_Mb
1924 \fi #1.#2.%
1925 }%

```

|A|.P+4.{A}{P}{n}{B}. We will keep only the first P+4 digits of A, denoted A'' in what follows.
output: u=-0.A''.junk.P+4.|A|.A{P}{n}{B}

```

1926 \def\XINT_infloat_Ma #1.#2.#3%
1927 {%
1928 \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
1929 \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1930 #2.#1.{#3}%
1931 }%

```

|A|.P+4.{A}{P}{n}{B}.

Here A is short. We set u = P+4 - |A|, and A'' = A (A' = 10^u A)

output: u.A''..P+4.|A|.A{P}{n}{B}

```

1932 \def\XINT_infloat_Mb #1.#2.#3%

```

```

1933 {%
1934     \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
1935     #3..#2.#1.{#3}%
1936 }%

    input u.A''.junk.P+4.|A|. {A}{P}{n}{B}
    output |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}

1937 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
1938 {%
1939     \expandafter\XINT_infloat_N
1940     \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
1941 }%
1942 \def\XINT_infloat_N #1.#2.%
1943 {%
1944     \ifnum #1>#2
1945         \expandafter\XINT_infloat_Oa
1946     \else
1947         \expandafter\XINT_infloat_Ob
1948     \fi #1.#2.%
1949 }%

    input |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}
    output v=-0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}

1950 \def\XINT_infloat_Oa #1.#2.#3%
1951 {%
1952     \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
1953     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1954     #1.%
1955 }%

    output v=P+4-|B|>=0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}

1956 \def\XINT_infloat_Ob #1.#2.#3%
1957 {%
1958     \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
1959 }%

    input v.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
    output Q1.P.|B|. |A|.n.{A}{B}
    Q1 = division euclidienne de A''.10^{u-v+P+3} par B''.
    Special detection of cases with A and B both having length at most P+4: this will happen when
    called from \xintFloatDiv as A and B (produced then via \XINTinFloatS) will have at most P digits.
    We then only need integer division with P+1 extra zeros, not P+3.

1960 \def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
1961 {%
1962     \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
1963     \romannumeral0\xintiique
1964     {\romannumeral0\XINT_dsx_addzerosnofuss
1965         {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
1966     {#3}.#9.#5.%
1967 }%

```

«quick» branch.

```

1968 \def\XINT_infloat_Qq #1.#2.%
1969 {%
1970     \expandafter\XINT_infloat_Rq
1971     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1972 }%
1973 \def\XINT_infloat_Rq #1.#2#3.%
1974 {%
1975     \ifnum#2<\xint_c_v
1976         \expandafter\XINT_infloat_SEq
1977     \else\expandafter\XINT_infloat_SUP
1978     \fi
1979     {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
1980 }%

```

standard branch which will have to handle undecided rounding, if too close to a mid-value.

```

1981 \def\XINT_infloat_Q #1.#2.%
1982 {%
1983     \expandafter\XINT_infloat_R
1984     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1985 }%
1986 \def\XINT_infloat_R #1.#2#3#4#5.%
1987 {%
1988     \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
1989     #2#3#4#5.#1.%
1990 }%

```

trailing digits.Q.P.|B|. |A|.n.{A}{B}

#1=trailing digits (they may have leading zeros.)

```

1991 \def\XINT_infloat_Sa #1.%
1992 {%
1993     \ifnum#1>500 \xint_dothis\XINT_infloat_SUP\fi
1994     \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
1995     \xint_orthat\XINT_infloat_X\xint_c_
1996 }%
1997 \def\XINT_infloat_Sb #1.%
1998 {%
1999     \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
2000     \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2001     \xint_orthat\XINT_infloat_X\xint_c_i
2002 }%

```

epsilon #2=Q.#3=P.#4=|B|. #5=|A|. #6=n.{A}{B}

exposant final est n+|A|-|B|-P+epsilon

```

2003 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2004 {%
2005     \expandafter\XINT_infloat_SY
2006     \the\numexpr #6+#5-#4-#3+#1.#2.%
2007 }%
2008 \def\XINT_infloat_SY #1.#2.{ #2[#1]}%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

initial digit #2 put aside to check for case of rounding up to next power of ten, which will need adjustment of mantissa and exponent.

```
2009 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2010 {%
2011   \expandafter\XINT_infloat_Y
2012   \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2013   \romannumeral0\xintinc{#2#3}.#2%
2014 }%
```

$\epsilon Q.P.|B|. |A|.n.\{A\}\{B\}$

$\backslash\text{xintDSH}\{-x\}\{U\}$ multiplies U by 10^x . When x is negative, this means it truncates (i.e. it drops the last -x digits).

We don't try to optimize too much macro calls here, the odds are 2 per 1000 for this branch to be taken. Perhaps in future I will use higher free parameter d, which currently is set at 4.

#1= ϵ , #2#3=Q, #4=P, #5=|B|, #6=|A|, #7=n, #8=A, #9=B

```
2015 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2016 {%
2017   \expandafter\XINT_infloat_Y
2018   \the\numexpr #7+#6-#5-#4+#1\expandafter.%
2019   \romannumeral`&&\romannumeral0\xintiiflt
2020   {\xintDSH{#6-#5-#4+#1}\xintDouble{#8}}}%
2021   {\xintiiMul{\xintInc{\xintDouble{#2#3}}}\{#9}}}%
2022   \xint_firstofone
2023   \xintinc{#2#3}.#2%
2024 }%
```

check for rounding up to next power of ten.

```
2025 \def\XINT_infloat_Y #1{%
2026 \def\XINT_infloat_Y ##1.##2##3.##4%
2027 {%
2028   \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
2029   #1##2##3[##1]%
2030 }}\XINT_infloat_Y{ }%
```

#1=1, #2=0.

```
2031 \def\XINT_infloat_Z #1#2#3[#4]%
2032 {%
2033   \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2034 }%
2035 \def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%
```

8.64 $\backslash\text{xintPFloat}$

1.1. This is a prettifying printing macro for floats.

The macro applies one simple rule: x.yz...eN will drop scientific notation in favor of pure decimal notation if $-5 \leq N \leq 5$. This is the default behaviour of Maple. The N here is as produced on output by $\backslash\text{xintFloat}$.

Special case: the zero value is printed 0. (with a dot)

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfraction, xintexpr, indices

The coding got simpler with 1.2k as its `\xintFloat` always produces a mantissa with exactly P digits (no more 10.0...0eN annoying exception).

```

2036 \def\xintPFloat    {\romannumeral0\xintpfloat }%
2037 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2038 \def\XINT_pfloat_chkopt #1%
2039 {%
2040     \ifx [#1\expandafter\XINT_pfloat_opt
2041         \else\expandafter\XINT_pfloat_noopt
2042     \fi #1%
2043 }%
2044 \def\XINT_pfloat_noopt #1\xint:%
2045 {%
2046     \expandafter\XINT_pfloat_a
2047     \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2048 }%

2049 \def\XINT_pfloat_opt [\xint:#1]%
2050 {%
2051     \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
2052 }%
2053 \def\XINT_pfloat_opt_a #1.#2%
2054 {%
2055     \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2056 }%
2057 \def\XINT_pfloat_a #1%
2058 {%
2059     \xint_UDzerominusfork
2060     #1-\XINT_pfloat_zero
2061     0#1\XINT_pfloat_neg
2062     0-\XINT_pfloat_pos
2063     \krof #1%
2064 }%

2065 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2066 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%

2067 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2068 {%
2069     \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2070     \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2071     \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2072     \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2073     \xint_orthat\XINT_pfloat_P #1#2e#3;%
2074 }%
2075 \def\XINT_pfloat_no #1#2;{ #1.#2}%

This is all simpler coded, now that 1.2k's \xintFloat always outputs a mantissa with exactly one
digits before decimal mark always.

2076 \def\XINT_pfloat_N #1e-#2;%
2077 {%

```

```

2078 \csname XINT_pfloat_N_\romannumeral#2\endcsname #1%
2079 }%
2080 \def\XINT_pfloat_N_i { 0.}%
2081 \def\XINT_pfloat_N_ii { 0.0}%
2082 \def\XINT_pfloat_N_iii{ 0.00}%
2083 \def\XINT_pfloat_N_iv { 0.000}%
2084 \def\XINT_pfloat_N_v { 0.0000}%

2085 \def\XINT_pfloat_P #1e#2;%
2086 {%
2087 \csname XINT_pfloat_P_\romannumeral#2\endcsname #1%
2088 }%
2089 \def\XINT_pfloat_P_ #1{ #1.}%
2090 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2091 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2092 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2093 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2094 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

2095 \def\XINT_pfloat_Ps #1e#2;%
2096 {%
2097 \csname XINT_pfloat_Ps_\romannumeral#2\endcsname #100000;%
2098 }%
2099 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.}%
2100 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.}%
2101 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.}%
2102 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.}%
2103 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.}%

```

8.65 \XINTinFloatFracdigits

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

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

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

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

```

2104 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2105 \def\XINTinfloatfracdigits #1%
2106 {%
2107 \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xinttfrac{#1}}%
2108 }%
2109 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

8.66 \xintFloatAdd, \XINTinFloatAdd

First included in release 1.07.

1.09ka improved a bit the efficiency. However the add, sub, mul, div routines were provisory and supposed to be revised soon.

Which didn't happen until 1.2f. Now, the inputs are first rounded to P digits, not P+2 as earlier.

```

2110 \def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
2111 \def\xintfloatadd      #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2112 \def\XINTinFloatAdd    {\romannumeral0\XINTinfloatadd }%
2113 \def\XINTinfloatadd    #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2114 \def\XINT_fladd_chkopt #1#2%
2115 {%
2116   \ifx [#2\expandafter\XINT_fladd_opt
2117     \else\expandafter\XINT_fladd_noopt
2118   \fi #1#2%
2119 }%
2120 \def\XINT_fladd_noopt #1#2\xint:#3%
2121 {%
2122   #1[\XINTdigits]%
2123   {\expandafter\XINT_FL_add_a
2124     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2125 }%
2126 \def\XINT_fladd_opt #1[\xint:#2]%#3#4%
2127 {%
2128   \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2129 }%
2130 \def\XINT_fladd_opt_a #1.#2#3#4%
2131 {%
2132   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2133 }%
2134 \def\XINT_FL_add_a #1%
2135 {%
2136   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2137 }%

2138 \def\XINT_FL_add_zero #1.#2{#2}%[
2139 \def\XINT_FL_add_b #1]#2.#3%
2140 {%
2141   \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1%
2142 }%

2143 \def\XINT_FL_add_c #1%
2144 {%
2145   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2146 }%

2147 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2148 {%
2149   \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2150   \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2151   \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2152 }%

```


8.67 \xintFloatSub, \XINTinFloatSub

First done 1.07.

Starting with 1.2f the arguments undergo an initial rounding to the target precision P not P+2.

```

2153 \def\xintFloatSub      {\romannumeral0\xintfloatsub }%
2154 \def\xintfloatsub      #1{\XINT_flsb_chkopt \xintfloat #1\xint:}%
2155 \def\XINTinFloatSub    {\romannumeral0\XINTinfloatsub }%
2156 \def\XINTinfloatsub    #1{\XINT_flsb_chkopt \XINTinfloatS #1\xint:}%
2157 \def\XINT_flsb_chkopt #1#2%
2158 {%
2159   \ifx [#2\expandafter\XINT_flsb_opt
2160     \else\expandafter\XINT_flsb_noopt
2161   \fi #1#2%
2162 }%
2163 \def\XINT_flsb_noopt #1#2\xint:#3%
2164 {%
2165   #1[\XINTdigits]%
2166   {\expandafter\XINT_FL_add_a
2167     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2168 }%
2169 \def\XINT_flsb_opt #1[\xint:#2]%#3#4%
2170 {%
2171   \expandafter\XINT_flsb_opt_a\the\numexpr #2.#1%
2172 }%
2173 \def\XINT_flsb_opt_a #1.#2#3#4%
2174 {%
2175   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2176 }%

```

8.68 \xintFloatMul, \XINTinFloatMul

1.07.

Starting with 1.2f the arguments are rounded to the target precision P not P+2.

1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

1.2k does a micro improvement to the way the macro passes over control to its output routine (former version used a higher level \xintE causing some extra un-needed processing with two calls to \XINT_infrac where one was amply enough).

```

2177 \def\xintFloatMul      {\romannumeral0\xintfloatmul }%
2178 \def\xintfloatmul      #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2179 \def\XINTinFloatMul    {\romannumeral0\XINTinfloatmul }%
2180 \def\XINTinfloatmul    #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2181 \def\XINT_flmul_chkopt #1#2%
2182 {%
2183   \ifx [#2\expandafter\XINT_flmul_opt
2184     \else\expandafter\XINT_flmul_noopt
2185   \fi #1#2%
2186 }%
2187 \def\XINT_flmul_noopt #1#2\xint:#3%
2188 {%
2189   #1[\XINTdigits]%

```

```

2190     {\expandafter\XINT_FL_mul_a
2191       \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2192 }%
2193 \def\XINT_flmul_opt #1[\xint:#2]%#3#4%
2194 {%
2195   \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2196 }%
2197 \def\XINT_flmul_opt_a #1.#2#3#4%
2198 {%
2199   #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
2200 }%
2201 \def\XINT_FL_mul_a #1[#2]#3.#4%
2202 {%
2203   \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2204 }%

2205 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

8.69 \xintFloatDiv, \XINTinFloatDiv

1.07.

Starting with 1.2f the arguments are rounded to the target precision P not P+2.

1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

The actual rounding of the quotient is handled via \xintfloat (or \XINTinfloatS).

1.2k does the same kind of improvement in \XINT_FL_div_b as for multiplication: earlier code was unnecessarily high level.

```

2206 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2207 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint:%}
2208 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2209 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:%}
2210 \def\XINT_fldiv_chkopt #1#2%
2211 {%
2212   \ifx [#2\expandafter\XINT_fldiv_opt
2213     \else\expandafter\XINT_fldiv_noopt
2214   \fi #1#2%
2215 }%

2216 \def\XINT_fldiv_noopt #1#2\xint:#3%
2217 {%
2218   #1[\XINTdigits]%
2219   {\expandafter\XINT_FL_div_a
2220     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2221 }%
2222 \def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
2223 {%
2224   \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2225 }%

2226 \def\XINT_fldiv_opt_a #1.#2#3#4%

```

```

2227 {%
2228     #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.#3}}%
2229 }%
2230 \def\XINT_FL_div_a #1[#2]#3.#4%
2231 {%
2232     \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2233 }%

2234 \def\XINT_FL_div_b #1[#2]{#1e#2}%

```

8.70 \xintFloatPow, \XINTinFloatPow

1.07: initial version. 1.09j has re-organized the core loop.

2015/12/07. I have hesitated to map $^$ in expressions to \xintFloatPow rather than \xintFloatPower. But for 1.234567890123456 to the power 2145678912 with P=16, using Pow rather than Power seems to bring only about 5% gain.

This routine requires the exponent x to be compatible with \numexpr parsing.

1.2f has rewritten the code for better efficiency. Also, now the argument A for A^x is first rounded to P digits before switching to the increased working precision (which depends upon x).

```

2235 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2236 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2237 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2238 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2239 \def\XINT_flpow_chkopt #1#2%
2240 {%
2241     \ifx [#2\expandafter\XINT_flpow_opt
2242     \else\expandafter\XINT_flpow_noopt
2243     \fi
2244     #1#2%
2245 }%
2246 \def\XINT_flpow_noopt #1#2\xint:#3%
2247 {%
2248     \expandafter\XINT_flpow_checkB_a
2249     \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}}%
2250 }%
2251 \def\XINT_flpow_opt #1[\xint:#2]%
2252 {%
2253     \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2254 }%
2255 \def\XINT_flpow_opt_a #1.#2#3#4%
2256 {%
2257     \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}}%
2258 }%
2259 \def\XINT_flpow_checkB_a #1%
2260 {%
2261     \xint_UDzerominusfork
2262     #1-\XINT_flpow_BisZero
2263     0#1{\XINT_flpow_checkB_b -}%
2264     0-{\XINT_flpow_checkB_b }{#1}%
2265     \krof
2266 }%
2267 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

```

```

2268 \def\XINT_flpow_checkB_b #1#2.#3.%
2269 {%
2270     \expandafter\XINT_flpow_checkB_c
2271     \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2272 }%

2273 \def\XINT_flpow_checkB_c #1.#2.%
2274 {%
2275     \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2276 }%

    1.2f rounds input to P digits, first.

2277 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2278 {%
2279     \expandafter \XINT_flpow_aa
2280     \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2281 }%

2282 \def\XINT_flpow_aa #1[#2]#3%
2283 {%
2284     \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2285     \romannumeral\XINT_rep #3\endcsname0.#1.%
2286 }%

2287 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2288 \def\XINT_flpow_a #1%
2289 {%
2290     \xint_UDzerominusfork
2291     #1-\XINT_flpow_zero
2292     0#1{\XINT_flpow_b \iftrue}%
2293     0-{\XINT_flpow_b \iffalse#1}%
2294     \krof
2295 }%
2296 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2297 {%
2298     #6{\if 1#51\xint_dothis {0[0]}\fi
2299         \xint_orthat
2300         {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2301     }%
2302 }%

2303 \def\XINT_flpow_b #1#2[#3]#4#5%
2304 {%
2305     \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2306 }%

2307 \def\XINT_flpow_truncate #1.#2.#3.%
2308 {%
2309     \expandafter\XINT_flpow_truncate_a
2310     \romannumeral0\XINT_split_fromleft
2311     #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2312 }%

```

```

2313 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2314 \def\XINT_flpow_loopI #1.%
2315 {%
2316     \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2317     \ifodd #1
2318         \expandafter\XINT_flpow_loopI_odd
2319     \else
2320         \expandafter\XINT_flpow_loopI_even
2321     \fi
2322     #1.%
2323 }%

2324 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2325 {%
2326     \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2327 }%

2328 \def\XINT_flpow_loopI_even #1.#2.#3.##4.%
2329 {%
2330     \expandafter\XINT_flpow_loopI
2331     \the\numexpr #1/\xint_c_ii\expandafter.%
2332     \the\numexpr\expandafter\XINT_flpow_truncate
2333     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2334 }%
2335 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2336 {%
2337     \expandafter\XINT_flpow_loopII
2338     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2339     \the\numexpr\expandafter\XINT_flpow_truncate
2340     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2341 }%
2342 \def\XINT_flpow_loopII #1.%
2343 {%
2344     \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2345     \ifodd #1
2346         \expandafter\XINT_flpow_loopII_odd
2347     \else
2348         \expandafter\XINT_flpow_loopII_even
2349     \fi
2350     #1.%
2351 }%
2352 \def\XINT_flpow_loopII_even #1.#2.#3.##4.%
2353 {%
2354     \expandafter\XINT_flpow_loopII
2355     \the\numexpr #1/\xint_c_ii\expandafter.%
2356     \the\numexpr\expandafter\XINT_flpow_truncate
2357     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2358 }%
2359 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2360 {%
2361     \expandafter\XINT_flpow_loopII_odda
2362     \the\numexpr\expandafter\XINT_flpow_truncate

```

```

2363 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2364 #1.#2.#3.%
2365 }%
2366 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2367 {%
2368 \expandafter\XINT_flpow_loopII
2369 \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2370 \the\numexpr\expandafter\XINT_flpow_truncate
2371 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2372 #1.#2.%
2373 }%

2374 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2375 {%
2376 \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2377 \the\numexpr\expandafter\XINT_flpow_truncate
2378 \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2379 }%

```

This ending is common with `\xintFloatPower`.

In the case of negative exponent we need to inverse the Q-digits mantissa. This requires no special attention now as 1.2k's `\xintFloat` does correct rounding of fractions hence it is easy to bound the total error. It can be checked that the algorithm after final rounding to the target precision computes a value Z whose distance to the exact theoretical will be less than 0.52 ulp(Z) (and worst cases can only be slightly worse than 0.51 ulp(Z)).

In the case of the half-integer exponent (only via the expression interface,) the computation (which proceeds via `\XINTinFloatPowerH`) ends with a square root. This square root extraction is done with 3 guard digits (the power operations were done with more.) Then the value is rounded to the target precision. There is thus this rounding to 3 guard digits (in the case of negative exponent the reciprocal is computed before the square-root), then the square root is (computed with exact rounding for these 3 guard digits), and then there is the final rounding of this to the target precision. The total error (for positive as well as negative exponent) has been estimated to at worst possibly exceed slightly 0.5125 ulp(Z), and at any rate it is less than 0.52 ulp(Z).

```

2380 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2381 {%
2382 \expandafter\XINT_flpow_IIIend
2383 \xint_UDsignfork
2384 #5{{1/#3[-#2]}}}%
2385 -{{#3[#2]}}}%
2386 \krof #1%
2387 }%

2388 \def\XINT_flpow_IIIend #1#2#3%
2389 {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&}\fi#1}}%

```

8.71 `\xintFloatPower`, `\XINTinFloatPower`

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to `\xintNum`. The `^` in expressions is mapped to this routine.

Same modifications as in `\xintFloatPow` for 1.2f.

1.2f adds a special private macro for allowing half-integral exponents for use with `^` within `\xintfloatexpr`. The exponent will be first truncated to either an integer or an half-integer. The macro is not for general use.

1.2k does anew this 1.2f handling of half-integer exponents for the `\xintfloatexpr` parser: with 1.2f's code the final square-root extraction was applied to a value already rounded to the target precision, unneedlessly losing precision.

```
2390 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2391 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2392 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
2393 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint:}%
```

First the special macro for use by the expression parser which checks if one raises to an half-integer exponent. This is always with `\XINTdigits` precision. Rewritten for 1.2k in order for the final square root to keep three guard digits.

We have to be careful that exponent #2 is not constrained by TeX bound. And we must allow fractions. The 1.2k variant does a rounding to nearest integer of half-integer, 1.2f did a truncation rather (this is done after truncation of #2 to fixed point with one digit after mark.) We try to recognize quickly the case of integer exponent, for speed, but there is overhead of going through `\xintiTrunc1`.

```
2394 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerh }%

2395 \def\XINTinfloatpowerh #1#2%
2396 {%
2397   \expandafter\XINT_flpowerh_a\romannumeral0\xintiTrunc1{#2};%
2398   \XINTdigits.{#1}\XINTinfloatS[\XINTdigits]}%
2399 }%

2400 \def\XINT_flpowerh_a #1;%
2401 {%
2402   \if0\xintLDg{#1}\expandafter\XINT_flpowerh_int
2403   \else\expandafter\XINT_flpowerh_b
2404   \fi #1.%
2405 }%
2406 \def\XINT_flpowerh_int #1%
2407 {%
2408   \if0#1\expandafter\XINT_flpower_BisZero
2409   \else\expandafter\XINT_flpowerh_i
2410   \fi #1%
2411 }%
2412 \def\XINT_flpowerh_i #10.{\expandafter\XINT_flpower_checkB_a#1.}%
2413 \def\XINT_flpowerh_b #1.%
2414 {%
2415   \expandafter\XINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2416 }%
2417 \def\XINT_flpowerh_c #1.%
2418 {%
2419   \ifodd\xintLDg{#1} %<- intentional space
2420   \expandafter\XINT_flpowerh_d\else\expandafter\XINT_flpowerh_e
2421   \fi #1.%
2422 }%
```

```

2423 \def\XINT_flpowerh_d #1.\XINTdigits.#2#3%
2424 {%
2425   \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2426 }%
2427 \def\XINT_flpowerh_finish #1%
2428   {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%
2429 \def\XINT_flpowerh_e #1.%
2430   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xinthal{#1}.}%

```

Start of macro. Check for optional argument.

```

2431 \def\XINT_flpower_chkopt #1#2%
2432 {%
2433   \ifx [#2\expandafter\XINT_flpower_opt
2434     \else\expandafter\XINT_flpower_noopt
2435   \fi
2436   #1#2%
2437 }%
2438 \def\XINT_flpower_noopt #1#2\xint:#3%
2439 {%
2440   \expandafter\XINT_flpower_checkB_a
2441   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2442 }%
2443 \def\XINT_flpower_opt #1[\xint:#2]%
2444 {%
2445   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2446 }%
2447 \def\XINT_flpower_opt_a #1.#2#3#4%
2448 {%
2449   \expandafter\XINT_flpower_checkB_a
2450   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2451 }%
2452 \def\XINT_flpower_checkB_a #1%
2453 {%
2454   \xint_UDzerominusfork
2455     #1-{\XINT_flpower_BisZero 0}%
2456     0#1{\XINT_flpower_checkB_b -}%
2457     0-{\XINT_flpower_checkB_b {}#1}%
2458   \krof
2459 }%
2460 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2461 \def\XINT_flpower_checkB_b #1#2.#3.%
2462 {%
2463   \expandafter\XINT_flpower_checkB_c
2464   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2465 }%

2466 \def\XINT_flpower_checkB_c #1.#2.%
2467 {%
2468   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2469 }%

```



```

2470 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2471 {%
2472     \expandafter \XINT_flpower_aa
2473     \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2474 }%

2475 \def\XINT_flpower_aa #1[#2]#3%
2476 {%
2477     \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2478     \romannumeral\XINT_rep #3\endcsname0.#1.%
2479 }%
2480 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2481 \def\XINT_flpower_a #1%
2482 {%
2483     \xint_UDzerominusfork
2484     #1-\XINT_flpow_zero
2485     0#1{\XINT_flpower_b \iftrue}%
2486     0-{\XINT_flpower_b \iffalse#1}%
2487     \krof
2488 }%
2489 \def\XINT_flpower_b #1#2[#3]#4#5%
2490 {%
2491     \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiioOdd{#5}\fi}%
2492 }%
2493 \def\XINT_flpower_loopI #1.%
2494 {%
2495     \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2496     \ifodd\xintLDg{#1} %<- intentional space
2497     \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2498     \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2499     \romannumeral0\XINT_half
2500     #1\xint_bye\xint_Bye345678\xint_bye
2501     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2502 }%
2503 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2504 {%
2505     \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2506 }%
2507 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2508 {%
2509     \expandafter\XINT_flpower_toloopI
2510     \the\numexpr\expandafter\XINT_flpow_truncate
2511     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2512 }%
2513 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2514 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2515 {%
2516     \expandafter\XINT_flpower_toloopII
2517     \the\numexpr\expandafter\XINT_flpow_truncate
2518     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2519     #1.#2.#3.%

```

```

2520 }%
2521 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2522 \def\XINT_flpower_loopII #1.%
2523 {%
2524     \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_IItoIII\fi
2525     \ifodd\xintLDg{#1} %<- intentional space
2526         \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2527     \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

2528     \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2529     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2530 }%
2531 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2532 {%
2533     \expandafter\XINT_flpower_toloopII
2534     \the\numexpr\expandafter\XINT_flpow_truncate
2535     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2536 }%
2537 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2538 {%
2539     \expandafter\XINT_flpower_loopII_odda
2540     \the\numexpr\expandafter\XINT_flpow_truncate
2541     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2542     #1.#2.#3.%
2543 }%
2544 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2545 {%
2546     \expandafter\XINT_flpower_toloopII
2547     \the\numexpr\expandafter\XINT_flpow_truncate
2548     \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2549     #4.#1.#2.%
2550 }%
2551 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2552 {%
2553     \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2554     \the\numexpr\expandafter\XINT_flpow_truncate
2555     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2556 }%

```

8.72 \xintFloatFac, \XINTFloatFac

```

2557 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2558 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2559 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac}%
2560 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint:}%
2561 \def\XINT_flfac_chkopt #1#2%
2562 {%
2563     \ifx [#2\expandafter\XINT_flfac_opt
2564         \else\expandafter\XINT_flfac_noopt
2565     \fi
2566     #1#2%
2567 }%

```

```

2568 \def\XINT_flgfac_noopt #1#2\xint:
2569 {%
2570   \expandafter\XINT_FL_fac_fork_a
2571   \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%
2572 }%
2573 \def\XINT_flgfac_opt #1[\xint:#2]%
2574 {%
2575   \expandafter\XINT_flgfac_opt_a\the\numexpr #2.#1%
2576 }%
2577 \def\XINT_flgfac_opt_a #1.#2#3%
2578 {%
2579   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]]}%
2580 }%
2581 \def\XINT_FL_fac_fork_a #1%
2582 {%
2583   \xint_UDzerominusfork
2584   #1-\XINT_FL_fac_iszero
2585   0#1\XINT_FL_fac_isneg
2586   0-\XINT_FL_fac_fork_b #1}%
2587 \krof
2588 }%
2589 \def\XINT_FL_fac_iszero #1.#2#3#4#5{#5{1[0]}}%

```

1.2f XINT_FL_fac_isneg returns 0, earlier versions used 1 here.

```

2590 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2591 {%
2592   #5{\XINT_signalcondition{InvalidOperation}
2593     {Factorial of negative: (-#1)!}{0[0]}}%
2594 }%
2595 \def\XINT_FL_fac_fork_b #1.%
2596 {%
2597   \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2598   \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2599   \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2600   \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2601   \xint_orthat\XINT_FL_fac_small
2602   #1.%
2603 }%
2604 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2605 {%
2606   #5{\XINT_signalcondition{InvalidOperation}
2607     {Factorial of too big: (#1)!}{0[0]}}%
2608 }%

```

Computations are done with Q blocks of eight digits. When a multiplication has a carry, hence creates Q+1 blocks, the least significant one is dropped. The goal is to compute an approximate value X' to the exact value X, such that the final relative error (X-X')/X will be at most 10^{-P} with P the desired precision. Then, when we round X' to X'' with P significant digits, we can prove that the absolute error |X-X''| is bounded (strictly) by 0.6 ulp(X''). (ulp= unit in the last (significant) place). Let N be the number of such operations, the formula for Q deduces from the previous explanations is that 8Q should be at least P+9+k, with k the number of digits of N (in base 10). Note that 1.2 version used P+10+k, for 1.2f I reduced to P+9+k. Also, k should be the number of digits of the number N of multiplications done, hence for n<=10000 we can take N=n/2, or

$N/3$, or $N/4$. This is rounded above by numexpr and always an overestimate of the actual number of approximate multiplications done (the first ones are exact). (vérifier ce que je raconte, j'ai la flemme là).

We then want $\text{ceil}((P+k+n)/8)$. Using \numexpr rounding division (ARRRRRGGGHHHH), if m is a positive integer, $\text{ceil}(m/8)$ can be computed as $(m+3)/8$. Thus with $m=P+10+k$, this gives $Q<-(P+13+k)/8$. The routine actually computes $8(Q-1)$ for use in \XINT_FL_fac_addzeros.

With 1.2f the formula is $m=P+9+k$, $Q<-(P+12+k)/8$, and we use now $4=12-8$ rather than the earlier $5=13-8$. Whatever happens, the value computed in \XINT_FL_fac_increaseP is at least 8. There will always be an extra block.

Note: with Digits:=32; Maple gives for 200!:

```
> factorial(200.);
```

```
375
```

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

and this is the correct rounding because for 40 digits it computes

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to xint) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```
2609 \def\XINT_FL_fac_vbig
2610   {\expandafter\XINT_FL_fac_vbigloop_a
2611     \the\numexpr \XINT_FL_fac_increaseP \xint_c_i }%
2612 \def\XINT_FL_fac_big
2613   {\expandafter\XINT_FL_fac_bigloop_a
2614     \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii }%
2615 \def\XINT_FL_fac_med
2616   {\expandafter\XINT_FL_fac_medloop_a
2617     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2618 \def\XINT_FL_fac_small
2619   {\expandafter\XINT_FL_fac_smallloop_a
2620     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv }%
2621 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2622 {%
2623   #2\expandafter.\the\numexpr\xint_c_viii*%
2624   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2625     \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2626 }%
2627 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2628 \def\XINT_FL_fac_countdone #1#2\Z {#1}%
2629 \def\XINT_FL_fac_out #1;![#2]#3%
2630   {#3{\romannumeral0\XINT_mul_out
2631     #1;!1\R!1\R!1\R!1\R!1\R!%
2632     1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2633 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2634 {%
2635   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2636   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2637     \the\numexpr \xint_c_x^viii+#1.}%
2638 }%
2639 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2640 {%
2641   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
```

```

2642 \expandafter\XINT_FL_fac_vbigloop_loop
2643 \the\numexpr #1+\xint_c_i\expandafter.%
2644 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2645 }%
2646 \def\XINT_FL_fac_bigloop_a #1.%
2647 {%
2648 \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2649 #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2650 }%
2651 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2652 {%
2653 \expandafter\XINT_FL_fac_medloop_a
2654 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2655 }%
2656 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2657 {%
2658 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2659 \expandafter\XINT_FL_fac_bigloop_loop
2660 \the\numexpr #1+\xint_c_ii\expandafter.%
2661 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2662 }%
2663 \def\XINT_FL_fac_bigloop_mul #1!%
2664 {%
2665 \expandafter\XINT_FL_fac_mul
2666 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2667 }%
2668 \def\XINT_FL_fac_medloop_a #1.%
2669 {%
2670 \expandafter\XINT_FL_fac_medloop_b
2671 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2672 }%
2673 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2674 {%
2675 \expandafter\XINT_FL_fac_smallloop_a
2676 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2677 }%
2678 \def\XINT_FL_fac_medloop_loop #1.#2.%
2679 {%
2680 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2681 \expandafter\XINT_FL_fac_medloop_loop
2682 \the\numexpr #1+\xint_c_iii\expandafter.%
2683 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2684 }%
2685 \def\XINT_FL_fac_medloop_mul #1!%
2686 {%
2687 \expandafter\XINT_FL_fac_mul
2688 \the\numexpr
2689 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2690 }%
2691 \def\XINT_FL_fac_smallloop_a #1.%
2692 {%
2693 \csname

```

```

2694      XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2695      \endcsname #1.%
2696 }%
2697 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2698 {%
2699      \XINT_FL_fac_addzeros #2.1000000001!.{2.#1.}{#2}%
2700 }%
2701 \expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%
2702 {%
2703      \XINT_FL_fac_addzeros #2.1000000002!.{3.#1.}{#2}%
2704 }%
2705 \expandafter\def\csname XINT_FL_fac_smallloop_-1\endcsname #1.#2.%
2706 {%
2707      \XINT_FL_fac_addzeros #2.1000000006!.{4.#1.}{#2}%
2708 }%
2709 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2710 {%
2711      \XINT_FL_fac_addzeros #2.1000000024!.{5.#1.}{#2}%
2712 }%
2713 \def\XINT_FL_fac_addzeros #1.%
2714 {%
2715      \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2716      \expandafter\XINT_FL_fac_addzeros
2717      \the\numexpr #1-\xint_c_viii.1000000000!%
2718 }%

```

We will manipulate by successive *small* multiplications Q blocks 1<8d>!, terminated by 1;! We need a custom small multiplication which tells us when it has create a new block, and the least significant one should be dropped.

```

2719 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![-#4]]%
2720 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2721 {%
2722      \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2723      \expandafter\XINT_FL_fac_smallloop_loop
2724      \the\numexpr #1+\xint_c_iv\expandafter.%
2725      \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2726 }%
2727 \def\XINT_FL_fac_smallloop_mul #1!%
2728 {%
2729      \expandafter\XINT_FL_fac_mul
2730      \the\numexpr
2731          \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2732 }%[[
2733 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]]%
2734 \def\XINT_FL_fac_mul 1#1!%
2735      {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2736 \def\XINT_FL_fac_mul_a #1-#2%
2737 {%
2738      \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2739      \expandafter\space\fi #11;!%
2740 }%
2741 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2742 {%

```

```

2743 \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2744 }%
2745 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2746 {%
2747 \expandafter\XINT_FL_fac_minimulwc_c
2748 \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
2749 }%
2750 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2751 {%
2752 \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}{#7}{#6}%
2753 }%
2754 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2755 {%
2756 \expandafter\XINT_FL_fac_minimulwc_e
2757 \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{{#2}{#4}}%
2758 }%
2759 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
2760 {%
2761 1#6#9\expandafter!%
2762 \the\numexpr\expandafter\XINT_FL_fac_smallmul
2763 \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2764 }%
2765 \def\XINT_FL_fac_smallmul 1#1!#2#3!%
2766 {%
2767 \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2768 \XINT_FL_fac_minimulwc_a #2!#3!{{#1}{#2}}%
2769 }%

```

This is the crucial ending. I note that I used here an `\ifnum` test rather than the `gob_til_eightzeroes` thing. Actually for eight digits there is much less difference than for only four.

The "carry" situation is marked by a final `!-1` rather than `!-2` for no-carry. (a `\numexpr` must be stopped, and leaving a `-` as delimiter is good as it will not arise earlier.)

```

2770 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
2771 {%
2772 \ifnum #2=\xint_c_
2773 \expandafter\xint_firstoftwo\else
2774 \expandafter\xint_secondoftwo
2775 \fi
2776 {-2\relax[#4]}%
2777 {1#2\expandafter!\expandafter-\expandafter1\expandafter
2778 [\the\numexpr #4+\xint_c_viii]}%
2779 }%

```

8.73 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

2015/11/29 for 1.2f. Partial factorial `pfactorial(a,b)=(a+1)...``b`, only for non-negative integers with `a<=b<10^8`.

1.2h (2016/11/20) now avoids raising `\xintError:OutOfRangePFac` if the condition `0<=a<=b<10^8` is violated. Same as for `\xintiPFactorial`.

```

2780 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2781 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint}%
2782 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial}%

```

```

2783 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
2784 \def\XINT_flpfac_chkopt #1#2%
2785 {%
2786     \ifx [#2\expandafter\XINT_flpfac_opt
2787     \else\expandafter\XINT_flpfac_noopt
2788     \fi
2789     #1#2%
2790 }%
2791 \def\XINT_flpfac_noopt #1#2\xint:#3%
2792 {%
2793     \expandafter\XINT_FL_pfac_fork
2794     \the\numexpr \xintNum{#2}\expandafter.%
2795     \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]}}%
2796 }%
2797 \def\XINT_flpfac_opt #1[\xint:#2]%
2798 {%
2799     \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
2800 }%
2801 \def\XINT_flpfac_opt_b #1.#2#3#4%
2802 {%
2803     \expandafter\XINT_FL_pfac_fork
2804     \the\numexpr \xintNum{#3}\expandafter.%
2805     \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}}%
2806 }%
2807 \def\XINT_FL_pfac_fork #1#2.#3#4.%
2808 {%
2809     \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
2810     \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
2811     \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
2812     \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
2813     \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
2814 }%
2815 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
2816 {%
2817     #5{\XINT_signalcondition{InvalidOperation}
2818         {pfactorial second arg too big: 99999999 < #2}{0[0]}}%
2819 }%
2820 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
2821 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2822 \def\XINT_FL_pfac_neg -#1.-#2.%
2823 {%
2824     \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2825     \xint_orthat {%
2826         \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\fi
2827         \expandafter\XINT_FL_pfac_increaseP}%
2828         \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2829     }%

```

See the comments for \XINT_FL_pfac_increaseP. Case of b=a+1 should be filtered out perhaps. We only needed here to copy the \xintPFactorial macros and re-use \XINT_FL_fac_mul/\XINT_FL_fac_out. Had to modify a bit \XINT_FL_pfac_addzeroes. We can enter here directly with #3 equal to specify the precision (the calculated value before final rounding has a relative error less than #3.10^{ -#4-1}), and #5 would hold the macro doing the final rounding (or truncating, if I make a FloatTrunc

available) to a given number of digits, possibly not #4. By default the #3 is 1, but FloatBinomial calls it with #3=4.

```

2830 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2831 {%
2832   \expandafter\XINT_FL_pfac_a
2833   \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
2834     \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2835     /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2836     87654321\Z)/\xint_c_viii).#1.#2.%
2837 }%
2838 \def\XINT_FL_pfac_a #1.#2.#3.%
2839 {%
2840   \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2841   \the\numexpr#3\expandafter.%
2842   \romannumeral0\XINT_FL_pfac_addzeroes #1.10000000011;![-#1]%
2843 }%
2844 \def\XINT_FL_pfac_addzeroes #1.%
2845 {%
2846   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2847   \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.1000000000!%
2848 }%
2849 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2850 \def\XINT_FL_pfac_b #1.%
2851 {%
2852   \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2853   \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2854   \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2855   \xint_orthat\XINT_FL_pfac_smallloop #1.%
2856 }%
2857 \def\XINT_FL_pfac_smallloop #1.#2.%
2858 {%
2859   \ifcase\numexpr #2-#1\relax
2860     \expandafter\XINT_FL_pfac_end_
2861   \or \expandafter\XINT_FL_pfac_end_i
2862   \or \expandafter\XINT_FL_pfac_end_ii
2863   \or \expandafter\XINT_FL_pfac_end_iii
2864   \else\expandafter\XINT_FL_pfac_smallloop_a
2865   \fi #1.#2.%
2866 }%
2867 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2868 {%
2869   \expandafter\XINT_FL_pfac_smallloop_b
2870   \the\numexpr #1+\xint_c_iv\expandafter.%
2871   \the\numexpr #2\expandafter.%
2872   \romannumeral0\expandafter\XINT_FL_fac_mul
2873   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2874 }%
2875 \def\XINT_FL_pfac_smallloop_b #1.%
2876 {%
2877   \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2878     \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2879 }%

```

```

2880 \def\XINT_FL_pfac_medloop #1.#2.%
2881 {%
2882     \ifcase\numexpr #2-#1\relax
2883         \expandafter\XINT_FL_pfac_end_
2884     \or \expandafter\XINT_FL_pfac_end_i
2885     \or \expandafter\XINT_FL_pfac_end_ii
2886     \else\expandafter\XINT_FL_pfac_medloop_a
2887     \fi #1.#2.%
2888 }%
2889 \def\XINT_FL_pfac_medloop_a #1.#2.%
2890 {%
2891     \expandafter\XINT_FL_pfac_medloop_b
2892     \the\numexpr #1+\xint_c_iii\expandafter.%
2893     \the\numexpr #2\expandafter.%
2894     \romannumeral0\expandafter\XINT_FL_fac_mul
2895     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2896 }%
2897 \def\XINT_FL_pfac_medloop_b #1.%
2898 {%
2899     \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2900         \expandafter\XINT_FL_pfac_medloop \fi #1.%
2901 }%
2902 \def\XINT_FL_pfac_bigloop #1.#2.%
2903 {%
2904     \ifcase\numexpr #2-#1\relax
2905         \expandafter\XINT_FL_pfac_end_
2906     \or \expandafter\XINT_FL_pfac_end_i
2907     \else\expandafter\XINT_FL_pfac_bigloop_a
2908     \fi #1.#2.%
2909 }%
2910 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2911 {%
2912     \expandafter\XINT_FL_pfac_bigloop_b
2913     \the\numexpr #1+\xint_c_ii\expandafter.%
2914     \the\numexpr #2\expandafter.%
2915     \romannumeral0\expandafter\XINT_FL_fac_mul
2916     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2917 }%
2918 \def\XINT_FL_pfac_bigloop_b #1.%
2919 {%
2920     \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2921         \expandafter\XINT_FL_pfac_bigloop \fi #1.%
2922 }%
2923 \def\XINT_FL_pfac_vbigloop #1.#2.%
2924 {%
2925     \ifnum #2=#1
2926         \expandafter\XINT_FL_pfac_end_
2927     \else\expandafter\XINT_FL_pfac_vbigloop_a
2928     \fi #1.#2.%
2929 }%
2930 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
2931 {%

```

```

2932 \expandafter\XINT_FL_pfac_vbigloop
2933 \the\numexpr #1+\xint_c_i\expandafter.%
2934 \the\numexpr #2\expandafter.%
2935 \romannumeral0\expandafter\XINT_FL_fac_mul
2936 \the\numexpr\xint_c_x^viii+#1!%
2937 }%
2938 \def\XINT_FL_pfac_end_iii #1.#2.%
2939 {%
2940 \expandafter\XINT_FL_fac_out
2941 \romannumeral0\expandafter\XINT_FL_fac_mul
2942 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2943 }%
2944 \def\XINT_FL_pfac_end_ii #1.#2.%
2945 {%
2946 \expandafter\XINT_FL_fac_out
2947 \romannumeral0\expandafter\XINT_FL_fac_mul
2948 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2949 }%
2950 \def\XINT_FL_pfac_end_i #1.#2.%
2951 {%
2952 \expandafter\XINT_FL_fac_out
2953 \romannumeral0\expandafter\XINT_FL_fac_mul
2954 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2955 }%
2956 \def\XINT_FL_pfac_end_ #1.#2.%
2957 {%
2958 \expandafter\XINT_FL_fac_out
2959 \romannumeral0\expandafter\XINT_FL_fac_mul
2960 \the\numexpr \xint_c_x^viii+#1!%
2961 }%

```

8.74 \xintFloatBinomial, \XINTinFloatBinomial

1.2f. We compute $\text{binomial}(x,y)$ as $\text{pfac}(x-y,x)/y!$, where the numerator and denominator are computed with a relative error at most 4.10^{-2} , then rounded (once I have a float truncation, I will use truncation rather) to P+3 digits, and finally the quotient is correctly rounded to P digits. This will guarantee that the exact value X differs from the computed one Y by at most $0.6 \text{ ulp}(Y)$. (2015/12/01).

2016/11/19 for 1.2h. As for `\xintiiBinomial`, hard to understand why last year I coded this to raise an error if $y < 0$ or $y > x$! The question of the Gamma function is for another occasion, here x and y must be (small) integers.

```

2962 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
2963 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
2964 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial}%
2965 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
2966 \def\XINT_flbinom_chkopt #1#2%
2967 {%
2968 \ifx [#2\expandafter\XINT_flbinom_opt
2969 \else\expandafter\XINT_flbinom_noopt
2970 \fi #1#2%
2971 }%
2972 \def\XINT_flbinom_noopt #1#2\xint:#3%

```

```

2973 {%
2974     \expandafter\XINT_FL_binom_a
2975     \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
2976 }%
2977 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
2978 {%
2979     \expandafter\XINT_FL_binom_a
2980     \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
2981     \the\numexpr #2.#1%
2982 }%
2983 \def\XINT_FL_binom_a #1.#2.%
2984 {%
2985     \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
2986 }%
2987 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
2988 {%
2989     \if-#5\xint_dothis \XINT_FL_binom_neg\fi
2990     \if-#1\xint_dothis \XINT_FL_binom_zero\fi
2991     \if-#3\xint_dothis \XINT_FL_binom_zero\fi
2992     \if0#1\xint_dothis \XINT_FL_binom_one\fi
2993     \if0#3\xint_dothis \XINT_FL_binom_one\fi
2994     \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
2995     \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
2996         \xint_orthat\XINT_FL_binom_aa
2997     #1#2.#3#4.#5#6.%
2998 }%
2999 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3000 {%
3001     #5[#4]{\XINT_signalcondition{InvalidOperation}
3002         {binomial with first arg negative: #3}{\0[0]}}%
3003 }%
3004 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3005 {%
3006     #5[#4]{\XINT_signalcondition{InvalidOperation}
3007         {binomial with first arg too big: 99999999 < #3}{\0[0]}}%
3008 }%
3009 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3010 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3011 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3012 {%
3013     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3014         #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3015         {\XINT_FL_fac_fork_b
3016         #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3017 }%
3018 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3019 {%
3020     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3021         #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3022         {\XINT_FL_fac_fork_b
3023         #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3024 }%

```

8.75 \xintFloatSqrt, \XINTinFloatSqrt

First done for 1.08.

The float version was developed at the same time as the integer one and even a bit earlier. As a result the integer variant had some sub-optimal parts. Anyway, for 1.2f I have rewritten the integer variant, and the float variant delegates all preparatory work for it until the last step. In particular the very low precisions are not penalized anymore from doing computations for at least 17 or 18 digits. Both the large and small precisions give quite shorter computation times.

Also, after examining more closely the achieved precision I decided to extend the float version in order for it to obtain the correct rounding (for inputs already of at most P digits with P the precision) of the theoretical exact value.

Beyond about 500 digits of precision the efficiency decreases swiftly, as is the case generally speaking with xintcore/xint/xintfrac arithmetic macros.

Final note: with 1.2f the input is always first rounded to P significant places.

```

3025 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqrt }%
3026 \def\xintfloatsqrt      #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
3027 \def\XINTinFloatSqrt    {\romannumeral0\XINTinfloatsqrt }%
3028 \def\XINTinfloatsqrt    #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint:}%
3029 \def\XINT_flsqrt_chkopt #1#2%
3030 {%
3031   \ifx [#2\expandafter\XINT_flsqrt_opt
3032     \else\expandafter\XINT_flsqrt_noopt
3033   \fi  #1#2%
3034 }%
3035 \def\XINT_flsqrt_noopt #1#2\xint:%
3036 {%
3037   \expandafter\XINT_FL_sqrt_a
3038     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3039 }%
3040 \def\XINT_flsqrt_opt #1[\xint:#2]%#3%
3041 {%
3042   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
3043 }%
3044 \def\XINT_flsqrt_opt_a #1.#2#3%
3045 {%
3046   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3047 }%
3048 \def\XINT_FL_sqrt_a #1%
3049 {%
3050   \xint_UDzerominusfork
3051     #1-\XINT_FL_sqrt_iszero
3052     0#1\XINT_FL_sqrt_isneg
3053     0-{\XINT_FL_sqrt_pos #1}%
3054   \krof
3055 }%[
3056 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3057 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3058 {%
3059   #3[#2]{\XINT_signalcondition{InvalidOperation}
3060     {Square root of negative: -#1}}{0[0]}}%
3061 }%

```

```

3062 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3063 {%
3064     \expandafter\XINT_flsqrt
3065     \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3066     \xint_orthat {+\xint_c_ii.#2.{}#100.#3.%
3067 }%

3068 \def\XINT_flsqrt #1.#2.%
3069 {%
3070     \expandafter\XINT_flsqrt_a
3071     \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3072 }%

3073 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3074 {%
3075     \expandafter\XINT_flsqrt_b
3076     \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3077     \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3078 }%

3079 \def\XINT_flsqrt_b #1.#2#3%
3080 {%
3081     \expandafter\XINT_flsqrt_c
3082     \romannumeral0\xintiisub
3083     {\XINT_dsx_addzeros {#1}#2;}%
3084     {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3085         {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3086 }%

3087 \def\XINT_flsqrt_c #1.#2.%
3088 {%
3089     \expandafter\XINT_flsqrt_d
3090     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3091 }%

3092 \def\XINT_flsqrt_d #1.#2#3.%
3093 {%
3094     \ifnum #2=\xint_c_v
3095     \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3096     #2#3.#1.%
3097 }%

3098 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

3099 \def\XINT_flsqrt_f 5#1.%
3100     {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.%
3101 \def\XINT_flsqrt_g #1#2#3.{\ifrelax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3102     \xint_orthat{\XINT_flsqrt_finish 5.}}%
3103 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3104     \xint_orthat{\XINT_flsqrt_finish 5.}}%

```

```

3105 \def\XINT_flsqrt_again #1.#2.%
3106 {%
3107     \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3108 }%

```

```

3109 \def\XINT_flsqrt_again_a #1.#2.#3.%
3110 {%
3111     \expandafter\XINT_flsqrt_b
3112     \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3113     \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
3114                                     #1.#2000000000.#3.%
3115 }%

```

8.76 \xintFloatE, \XINTinFloatE

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

1.2k had to rewrite this since there is no more a \XINT_float_a macro. Attention about \XINTinFloatE: it is for use by xintexpr.sty, contrarily to other \XINTinFloat<foo> macros it inserts itself the [\XINTdigits] thing, and with value 0 it produces on output 0[N], not 0[0].

```

3116 \def\xintFloatE    {\romannumeral0\xintfloate }%
3117 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3118 \def\XINT_floate_chkopt #1%
3119 {%
3120     \ifx [#1\expandafter\XINT_floate_opt
3121         \else\expandafter\XINT_floate_noopt
3122     \fi #1%
3123 }%
3124 \def\XINT_floate_noopt #1\xint:%
3125 {%
3126     \expandafter\XINT_floate_post
3127     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3128 }%
3129 \def\XINT_floate_opt [\xint:#1]%
3130 {%
3131     \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3132 }%
3133 \def\XINT_floate_opt_a #1.#2%
3134 {%
3135     \expandafter\XINT_floate_post
3136     \romannumeral0\XINTinfloat[#1]{#2}#1.%
3137 }%
3138 \def\XINT_floate_post #1%
3139 {%
3140     \xint_UDzerominusfork
3141     #1-\XINT_floate_zero
3142     0#1\XINT_floate_neg
3143     0-\XINT_floate_pos
3144     \krof #1%
3145 }%[
3146 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3147 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%

```

```

3148 \def\XINT_floate_pos #1#2[#3]#4.#5%
3149 {%
3150     \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3151 }%
3152 \def\XINTinFloatE {\romannumeral0\XINTinfloate}%
3153 \def\XINTinfloate
3154     {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]]}%
3155 \def\XINT_infloate #1[#2]#3%
3156     {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}}%
3157 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

8.77 \XINTinFloatMod

1.1. Pour emploi dans `xintexpr`. Code shortened at 1.2p.

```

3158 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]}%
3159 \def\XINTinfloatmod [#1]#2#3%
3160 {%
3161     \XINTinfloat[#1]{\xintMod
3162         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3163         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3164 }%

```

8.78 \XINTinFloatDivFloor

1.2p. Formerly `//` and `/:` in `\xintfloatexpr` used `\xintDivFloor` and `\xintMod`, hence did not round their operands to float precision beforehand.

```

3165 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]]}%
3166 \def\XINTinfloatdivfloor [#1]#2#3%
3167 {%
3168     \xintdivfloor
3169         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3170         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3171 }%

```

8.79 \XINTinFloatDivMod

1.2p. Pour emploi dans `xintexpr`, donc je ne prends pas la peine de faire l'expansion du modulo, qui se produira dans le `\csname`.

Hésitation sur le quotient, faut-il l'arrondir immédiatement ? Finalement non, le produire comme un integer.

```

3172 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]]}%
3173 \def\XINTinfloatdivmod [#1]#2#3%
3174 {%
3175     \expandafter\XINT_infloatdivmod
3176     \romannumeral0\xintdivmod
3177         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3178         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3179     {#1}%
3180 }%
3181 \def\XINT_infloatdivmod #1#2#3{ #1,\XINTinFloat[#3]{#2}}%

```


8.80 \xintiffloatint

```

3182 \def\xintiffloatint {\romannumeral0\xintiffloatint}%
3183 \def\xintiffloatint #1{\expandafter\xint_iffloatint
3184         \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
3185 \def\xint_iffloatint #1#2/1[#3]%
3186 {%
3187   \if 0#1\xint_dothis\xint_firstoftwo_thenstop\fi
3188   \ifnum#3<\xint_c_\xint_dothis\xint_secondoftwo_thenstop\fi
3189   \xint_orthat\xint_firstoftwo_thenstop
3190 }%

```

8.81 (WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits

1.3b. Support for random() function.

Thus as it is a priori only for xintexpr usage, it expands inside \csname context, but as we need to get rid of initial zeros we use \xintRandomDigits not \xintXRandomDigits (\expanded would have a use case here).

And anyway as we want to be able to use random() in \xintdeffunc/\xintNewExpr, it is good to have f-expandable macros, so we add the small overhead to make it f-expandable.

We don't have to be very efficient in removing leading zeroes, as there is only 10% chance for each successive one. Besides we use (current) internal storage format of the type A[N], where A is not required to be with \xintDigits digits, so N will simply be -\xintDigits and needs no adjustment.

In case we use in future with #1 something else than \xintDigits we do the 0-(#1) construct.

I had some qualms about doing a random float like this which means that when there are leading zeros in the random digits the (virtual) mantissa ends up with trailing zeros. That did not feel right but I checked random() in Python (which of course uses radix 2), and indeed this is what happens there.

```

3191 \def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
3192 \def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3193 \def\XINTinrandomfloatS[#1]%
3194 {%
3195   \expandafter\xint_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
3196 }%
3197 \def\xint_inrandomfloatS-#1\xint:
3198 {%
3199   \expandafter\xint_inrandomfloatS_a
3200   \romannumeral0\xinrandomdigits{#1}[-#1]%
3201 }%

```

We add one macro to handle a tiny bit faster 90of cases, after all we also use one extra macro for the completely improbable all 0 case.

```

3202 \def\xint_inrandomfloatS_a#1%
3203 {%
3204   \if#10\xint_dothis{\xint_inrandomfloatS_b}\fi
3205   \xint_orthat{ #1}%
3206 }%[
3207 \def\xint_inrandomfloatS_b#1%
3208 {%
3209   \if#1[\xint_dothis{\xint_inrandomfloatS_zero}\fi% ]
3210   \if#10\xint_dothis{\xint_inrandomfloatS_b}\fi

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```
3211 \xint_orthat{ #1}%  
3212 }%[  
3213 \def\xINT_inrandomfloatS_zero#1]{ 0[0]}%
```

8.82 (WIP) \XINTinRandomFloatSixteen

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

```
3214 \def\xINTinRandomFloatSixteen%  
3215 {%  
3216 \romannumeral0\expandafter\xINT_inrandomfloatS_a  
3217 \romannumeral'&&\expandafter\xINT_eightrandomdigits  
3218 \romannumeral'&&\xINT_eightrandomdigits[-16]%  
3219 }%  
3220 \XINT_restorecatcodes_endinput%
```

9 Package [xintseries](#) implementation

.1	Catcodes, ε -TeX and reload detection . . .	259	.7	<code>\xintRationalSeries</code>	262
.2	Package identification	260	.8	<code>\xintRationalSeriesX</code>	263
.3	<code>\xintSeries</code>	260	.9	<code>\xintFxFtPowerSeries</code>	264
.4	<code>\xintiSeries</code>	260	.10	<code>\xintFxFtPowerSeriesX</code>	265
.5	<code>\xintPowerSeries</code>	261	.11	<code>\xintFloatPowerSeries</code>	265
.6	<code>\xintPowerSeriesX</code>	262	.12	<code>\xintFloatPowerSeriesX</code>	267

The commenting is currently (2018/06/17) very sparse.

9.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

38      \aftergroup\endinput % xintseries already loaded.
39      \fi
40  \fi
41  \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

9.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintseries}%
46 [2018/06/17 1.3c Expandable partial sums with xint package (JFB)]%

```

9.3 \xintSeries

```

47 \def\xintSeries {\romannumeral0\xintseries }%
48 \def\xintseries #1#2%
49 {%
50     \expandafter\XINT_series\expandafter
51     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
52 }%
53 \def\XINT_series #1#2#3%
54 {%
55     \ifnum #2<#1
56         \xint_afterfi { 0/1[0]}%
57     \else
58         \xint_afterfi {\XINT_series_loop {#1}{0}{#2}{#3}}%
59     \fi
60 }%
61 \def\XINT_series_loop #1#2#3#4%
62 {%
63     \ifnum #3>#1 \else \XINT_series_exit \fi
64     \expandafter\XINT_series_loop\expandafter
65     {\the\numexpr #1+1\expandafter }\expandafter
66     {\romannumeral0\xintadd {#2}{#4{#1}}}%
67     {#3}{#4}%
68 }%
69 \def\XINT_series_exit \fi #1#2#3#4#5#6#7#8%
70 {%
71     \fi\xint_gobble_ii #6%
72 }%

```

9.4 \xintiSeries

```

73 \def\xintiSeries {\romannumeral0\xintiseries }%
74 \def\xintiseries #1#2%
75 {%
76     \expandafter\XINT_iseries\expandafter
77     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
78 }%
79 \def\XINT_iseries #1#2#3%
80 {%
81     \ifnum #2<#1
82         \xint_afterfi { 0}%
83     \else

```

```

84     \xint_afterfi {\XINT_iseries_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%
87 \def\XINT_iseries_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_iseries_exit \fi
90   \expandafter\XINT_iseries_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\xintiadd {#2}{#4{#1}}}%
93   {#3}{#4}%
94 }%
95 \def\XINT_iseries_exit \fi #1#2#3#4#5#6#7#8%
96 {%
97   \fi\xint_gobble_ii #6%
98 }%

```

9.5 \xintPowerSeries

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

```

99 \def\xintPowerSeries {\romannumeral0\xintpowerseries }%
100 \def\xintpowerseries #1#2%
101 {%
102   \expandafter\XINT_powseries\expandafter
103   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
104 }%
105 \def\XINT_powseries #1#2#3#4%
106 {%
107   \ifnum #2<#1
108     \xint_afterfi { 0/1[0]}%
109   \else
110     \xint_afterfi
111     {\XINT_powseries_loop_i {#3{#2}}{#1}{#2}{#3}{#4}}%
112   \fi
113 }%
114 \def\XINT_powseries_loop_i #1#2#3#4#5%
115 {%
116   \ifnum #3>#2 \else\XINT_powseries_exit_i\fi
117   \expandafter\XINT_powseries_loop_ii\expandafter
118   {\the\numexpr #3-1\expandafter}\expandafter
119   {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
120 }%
121 \def\XINT_powseries_loop_ii #1#2#3#4%
122 {%
123   \expandafter\XINT_powseries_loop_i\expandafter
124   {\romannumeral0\xintadd {#4{#1}}{#2}}{#3}{#1}{#4}%
125 }%
126 \def\XINT_powseries_exit_i\fi #1#2#3#4#5#6#7#8#9%
127 {%

```

```

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

```

9.6 \xintPowerSeriesX

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

```

134 \def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
135 \def\xintpowerseriesx #1#2%
136 {%
137 \expandafter\XINT_powseriesx\expandafter
138 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
139 }%
140 \def\XINT_powseriesx #1#2#3#4%
141 {%
142 \ifnum #2<#1
143 \xint_afterfi { 0/1[0]}%
144 \else
145 \xint_afterfi
146 {\expandafter\XINT_powseriesx_pre\expandafter
147 {\romannumeral`&&@#4}{#1}{#2}{#3}%
148 }%
149 \fi
150 }%
151 \def\XINT_powseriesx_pre #1#2#3#4%
152 {%
153 \XINT_powseries_loop_i {#4{#3}}{#2}{#3}{#4}{#1}%
154 }%

```

9.7 \xintRationalSeries

This computes $F(a) + \dots + F(b)$ on the basis of the value of $F(a)$ and the ratios $F(n)/F(n-1)$. As in `\xintPowerSeries` we use an iterative scheme which has the great advantage to avoid denominator build-up. This makes exact computations possible with exponential type series, which would be completely inaccessible to `\xintSeries`. #1=a, #2=b, #3=F(a), #4=ratio function Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

155 \def\xintRationalSeries {\romannumeral0\xintratseries }%
156 \def\xintratseries #1#2%
157 {%
158 \expandafter\XINT_ratseries\expandafter
159 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
160 }%
161 \def\XINT_ratseries #1#2#3#4%

```

```

162 {%
163   \ifnum #2<#1
164     \xint_afterfi { 0/1[0]}%
165   \else
166     \xint_afterfi
167     {\XINT_ratseries_loop {#2}{1}{#1}{#4}{#3}}%
168   \fi
169}%
170 \def\XINT_ratseries_loop #1#2#3#4%
171 {%
172   \ifnum #1>#3 \else\XINT_ratseries_exit_i\fi
173   \expandafter\XINT_ratseries_loop\expandafter
174   {\the\numexpr #1-1\expandafter}\expandafter
175   {\romannumeral0\xintadd {1}{\xintMul {#2}{#4{#1}}}{#3}{#4}}%
176}%
177 \def\XINT_ratseries_exit_i\fi #1#2#3#4#5#6#7#8%
178 {%
179   \fi \XINT_ratseries_exit_ii #6%
180}%
181 \def\XINT_ratseries_exit_ii #1#2#3#4#5%
182 {%
183   \XINT_ratseries_exit_iii #5%
184}%
185 \def\XINT_ratseries_exit_iii #1#2#3#4%
186 {%
187   \xintmul{#2}{#4}%
188}%

```

9.8 \xintRationalSeriesX

a,b,initial,rationfunction,x

This computes $F(a,x)+\dots+F(b,x)$ on the basis of the value of $F(a,x)$ and the ratios $F(n,x)/F(n-1,x)$. The argument x is first expanded and it is the value resulting from this which is used then throughout. The initial term $F(a,x)$ must be defined as one-parameter macro which will be given x . Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

189 \def\xintRationalSeriesX {\romannumeral0\xintratseriesx}%
190 \def\xintratseriesx #1#2%
191 {%
192   \expandafter\XINT_ratseriesx\expandafter
193   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
194}%
195 \def\XINT_ratseriesx #1#2#3#4#5%
196 {%
197   \ifnum #2<#1
198     \xint_afterfi { 0/1[0]}%
199   \else
200     \xint_afterfi
201     {\expandafter\XINT_ratseriesx_pre\expandafter
202      {\romannumeral`&&@#5}{#2}{#1}{#4}{#3}}%
203  }%

```

```

204 \fi
205 }%
206 \def\XINT_ratseriesx_pre #1#2#3#4#5%
207 {%
208 \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
209 }%

```

9.9 \xintFxFtPowerSeries

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

```

210 \def\xintFxFtPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213 \expandafter\XINT_fppowseries\expandafter
214 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218 \ifnum #2<#1
219 \xint_afterfi { 0}%
220 \else
221 \xint_afterfi
222 {\expandafter\XINT_fppowseries_loop_pre\expandafter
223 {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224 {#1}{#4}{#2}{#3}{#5}%
225 }%
226 \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230 \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231 \expandafter\XINT_fppowseries_loop_i\expandafter
232 {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233 {\romannumeral0\xintitrunc {#6}{\xintMul {#5{#2}}{#1}}}%
234 {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237 {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241 \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242 \expandafter\XINT_fppowseries_loop_ii\expandafter
243 {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244 {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
247 {%
248 \expandafter\XINT_fppowseries_loop_i\expandafter

```



```

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

```

9.10 \xintFxFtPowerSeriesX

a,b,coeff,x,D

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

```

260 \def\xintFxFtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
261 \def\xintfxptpowerseriesx #1#2%
262 {%
263     \expandafter\XINT_fppowseriesx\expandafter
264     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
265 }%
266 \def\XINT_fppowseriesx #1#2#3#4#5%
267 {%
268     \ifnum #2<#1
269         \xint_afterfi { 0}%
270     \else
271         \xint_afterfi
272         {\expandafter \XINT_fppowseriesx_pre \expandafter
273         {\romannumeral`&&@#4}{#1}{#2}{#3}{#5}%
274         }%
275     \fi
276 }%
277 \def\XINT_fppowseriesx_pre #1#2#3#4#5%
278 {%
279     \expandafter\XINT_fppowseries_loop_pre\expandafter
280     {\romannumeral0\xinttrunc {#5}{\xintPow {#1}{#2}}}%
281     {#2}{#1}{#3}{#4}{#5}%
282 }%

```

9.11 \xintFloatPowerSeries

1.08a. I still have to re-visit \xintFxFtPowerSeries; temporarily I just adapted the code to the case of floats.

```

283 \def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
284 \def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint:}%
285 \def\XINT_flpowseries_chkopt #1%
286 {%
287     \ifx [#1\expandafter\XINT_flpowseries_opt

```

```

288     \else\expandafter\XINT_flpowseries_noopt
289     \fi
290     #1%
291 }%
292 \def\XINT_flpowseries_noopt #1\xint:#2%
293 {%
294     \expandafter\XINT_flpowseries\expandafter
295     {\the\numexpr #1\expandafter}\expandafter
296     {\the\numexpr #2}\XINTdigits
297 }%
298 \def\XINT_flpowseries_opt [\xint:#1]#2#3%
299 {%
300     \expandafter\XINT_flpowseries\expandafter
301     {\the\numexpr #2\expandafter}\expandafter
302     {\the\numexpr #3\expandafter}{\the\numexpr #1}%
303 }%
304 \def\XINT_flpowseries #1#2#3#4#5%
305 {%
306     \ifnum #2<#1
307         \xint_afterfi { 0.e0}%
308     \else
309         \xint_afterfi
310         {\expandafter\XINT_flpowseries_loop_pre\expandafter
311         {\romannumeral0\XINTinfloatpow [#3]{#5}{#1}}%
312         {#1}{#5}{#2}{#4}{#3}%
313         }%
314     \fi
315 }%
316 \def\XINT_flpowseries_loop_pre #1#2#3#4#5#6%
317 {%
318     \ifnum #4>#2 \else\XINT_flpowseries_dont_i \fi
319     \expandafter\XINT_flpowseries_loop_i\expandafter
320     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
321     {\romannumeral0\XINTinfloatmul [#6]{#5}{#2}}{#1}%
322     {#1}{#3}{#4}{#5}{#6}%
323 }%
324 \def\XINT_flpowseries_dont_i \fi\expandafter\XINT_flpowseries_loop_i
325     {\fi \expandafter\XINT_flpowseries_dont_ii }%
326 \def\XINT_flpowseries_dont_ii #1#2#3#4#5#6#7{\xintfloat [#7]{#2}}%
327 \def\XINT_flpowseries_loop_i #1#2#3#4#5#6#7%
328 {%
329     \ifnum #5>#1 \else \XINT_flpowseries_exit_i \fi
330     \expandafter\XINT_flpowseries_loop_ii\expandafter
331     {\romannumeral0\XINTinfloatmul [#7]{#3}{#4}}%
332     {#1}{#4}{#2}{#5}{#6}{#7}%
333 }%
334 \def\XINT_flpowseries_loop_ii #1#2#3#4#5#6#7%
335 {%
336     \expandafter\XINT_flpowseries_loop_i\expandafter
337     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
338     {\romannumeral0\XINTinfloatadd [#7]{#4}%
339     {\XINTinfloatmul [#7]{#6}{#2}}{#1}}}%

```

```

340      {#1}{#3}{#5}{#6}{#7}%
341 }%
342 \def\XINT_flpowseries_exit_i\fi\expandafter\XINT_flpowseries_loop_ii
343   {\fi \expandafter\XINT_flpowseries_exit_ii }%
344 \def\XINT_flpowseries_exit_ii #1#2#3#4#5#6#7%
345 {%
346   \xintfloatadd [#7]{#4}{\XINTinfloatmul [#7]{#6{#2}}{#1}}%
347 }%

```

9.12 \xintFloatPowerSeriesX

1.08a

```

348 \def\xintFloatPowerSeriesX {\romannumeral0\xintfloatpowerseriesx }%
349 \def\xintfloatpowerseriesx #1{\XINT_flpowseriesx_chkopt #1\xint:}%
350 \def\XINT_flpowseriesx_chkopt #1%
351 {%
352   \ifx [#1\expandafter\XINT_flpowseriesx_opt
353     \else\expandafter\XINT_flpowseriesx_noopt
354   \fi
355   #1%
356 }%
357 \def\XINT_flpowseriesx_noopt #1\xint:#2%
358 {%
359   \expandafter\XINT_flpowseriesx\expandafter
360   {\the\numexpr #1\expandafter}\expandafter
361   {\the\numexpr #2}\XINTdigits
362 }%
363 \def\XINT_flpowseriesx_opt [\xint:#1]#2#3%
364 {%
365   \expandafter\XINT_flpowseriesx\expandafter
366   {\the\numexpr #2\expandafter}\expandafter
367   {\the\numexpr #3\expandafter}{\the\numexpr #1}%
368 }%
369 \def\XINT_flpowseriesx #1#2#3#4#5%
370 {%
371   \ifnum #2<#1
372     \xint_afterfi { 0.e0}%
373   \else
374     \xint_afterfi
375     {\expandafter \XINT_flpowseriesx_pre \expandafter
376      {\romannumeral`&&@#5}{#1}{#2}{#4}{#3}%
377     }%
378   \fi
379 }%
380 \def\XINT_flpowseriesx_pre #1#2#3#4#5%
381 {%
382   \expandafter\XINT_flpowseries_loop_pre\expandafter
383   {\romannumeral0\XINTinfloatpow [#5]{#1}{#2}}%
384   {#2}{#1}{#3}{#4}{#5}%
385 }%
386 \XINT_restorecatcodes_endinput%

```

10 Package [xintcfrac](#) implementation

.1	Catcodes, ε -TeX and reload detection . . .	268	.16	<code>\xintiGctoF</code>	280
.2	Package identification	269	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	281
.3	<code>\xintCFrac</code>	269	.18	<code>\xintiCstoCv</code>	282
.4	<code>\xintGCFrac</code>	270	.19	<code>\xintGctoCv</code>	282
.5	<code>\xintGGCFrac</code>	272	.20	<code>\xintiGctoCv</code>	284
.6	<code>\xintGctoGCx</code>	273	.21	<code>\xintFtoCv</code>	285
.7	<code>\xintFtoCs</code>	273	.22	<code>\xintFtoCCv</code>	285
.8	<code>\xintFtoCx</code>	274	.23	<code>\xintCntoF</code>	285
.9	<code>\xintFtoC</code>	274	.24	<code>\xintGCntoF</code>	286
.10	<code>\xintFtoGC</code>	275	.25	<code>\xintCntoCs</code>	287
.11	<code>\xintFGtoC</code>	275	.26	<code>\xintCntoGC</code>	287
.12	<code>\xintFtoCC</code>	276	.27	<code>\xintGCntoGC</code>	288
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	277	.28	<code>\xintCstoGC</code>	289
.14	<code>\xintiCstoF</code>	278	.29	<code>\xintGctoGC</code>	289
.15	<code>\xintGctoF</code>	278			

The commenting is currently (2018/06/17) very sparse. Release 1.09m (2014/02/26) has modified a few things: `\xintFtoCs` and `\xintCntoCs` insert spaces after the commas, `\xintCstoF` and `\xintCstoCv` authorize spaces in the input also before the commas, `\xintCntoCs` does not brace the produced coefficients, new macros `\xintFtoC`, `\xintCtoF`, `\xintCtoCv`, `\xintFGtoC`, and `\xintGGCFrac`.

There is partial dependency on [xinttools](#) due to `\xintCstoF` and `\xintCsToCv`.

10.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

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

```

10.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcfrac}%
46 [2018/06/17 1.3c Expandable continued fractions with xint package (JFB)]%

```

10.3 \xintCFrac

```

47 \def\xintCFrac {\romannumeral0\xintcfrac}%
48 \def\xintcfrac #1%
49 {%
50   \XINT_cfrac_opt_a #1\xint:
51 }%
52 \def\XINT_cfrac_opt_a #1%
53 {%
54   \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
55 }%
56 \def\XINT_cfrac_noopt #1\xint:
57 {%
58   \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
59   \relax\relax
60 }%
61 \def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint:#1]%
62 {%
63   \fi\csname XINT_cfrac_opt#1\endcsname
64 }%
65 \def\XINT_cfrac_optl #1%
66 {%
67   \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
68   \relax\hfill
69 }%
70 \def\XINT_cfrac_optc #1%
71 {%
72   \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z

```

```

73    \relax\relax
74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
78    \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82    \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86    \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90    \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}%
94 \def\XINT_cfrac_loop_a
95 {%
96    \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100    \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104    \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108    \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111    {\XINT_cfrac_T #5#6{#2#4}\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114    \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#11#2}{#3}}%
115 }%
116 \def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
117 {%
118    \XINT_cfrac_end_b #3%
119 }%
120 \def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%

```

10.4 \xintGCFrac

```

121 \def\xintGCFrac {\romannumeral0\xintgcfrac }%
122 \def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint:}%
123 \def\XINT_gcfrac_opt_a #1%

```

```

124 {%
125   \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
126 }%
127 \def\XINT_gcfrac_noopt #1\xint:%
128 {%
129   \XINT_gcfrac #1+!\relax\relax
130 }%
131 \def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint:#1]%
132 {%
133   \fi\csname XINT_gcfrac_opt#1\endcsname
134 }%
135 \def\XINT_gcfrac_optl #1%
136 {%
137   \XINT_gcfrac #1+!\relax\hfill
138 }%
139 \def\XINT_gcfrac_optc #1%
140 {%
141   \XINT_gcfrac #1+!\relax\relax
142 }%
143 \def\XINT_gcfrac_optr #1%
144 {%
145   \XINT_gcfrac #1+!\hfill\relax
146 }%
147 \def\XINT_gcfrac
148 {%
149   \expandafter\XINT_gcfrac_enter\romannumeral`&&@%
150 }%
151 \def\XINT_gcfrac_enter {\XINT_gcfrac_loop {}}%
152 \def\XINT_gcfrac_loop #1#2+#3/%
153 {%
154   \xint_gob_til_exclam #3\XINT_gcfrac_endloop!%
155   \XINT_gcfrac_loop {{#3}{#2}{#1}}%
156 }%
157 \def\XINT_gcfrac_endloop!\XINT_gcfrac_loop #1#2#3%
158 {%
159   \XINT_gcfrac_T #2#3#1!!%
160 }%
161 \def\XINT_gcfrac_T #1#2#3#4{\XINT_gcfrac_U #1#2{\xintFrac{#4}}}%
162 \def\XINT_gcfrac_U #1#2#3#4#5%
163 {%
164   \xint_gob_til_exclam #5\XINT_gcfrac_end!\XINT_gcfrac_U
165     #1#2{\xintFrac{#5}}%
166     \ifcase\xintSgn{#4}
167     +\or+\else-\fi
168     \cfrac{#1\xintFrac{\xintAbs{#4}}#2}{#3}}%
169 }%
170 \def\XINT_gcfrac_end!\XINT_gcfrac_U #1#2#3%
171 {%
172   \XINT_gcfrac_end_b #3%
173 }%
174 \def\XINT_gcfrac_end_b #1\cfrac#2#3{ #3}%

```

10.5 \xintGGCFrac

New with 1.09m

```

175 \def\xintGGCFrac {\romannumeral0\xintggcfrac }%
176 \def\xintggcfrac #1{\XINT_ggcfrac_opt_a #1\xint:}%
177 \def\XINT_ggcfrac_opt_a #1%
178 {%
179     \ifx[#1\XINT_ggcfrac_opt_b\fi \XINT_ggcfrac_noopt #1%
180 }%
181 \def\XINT_ggcfrac_noopt #1\xint:
182 {%
183     \XINT_ggcfrac #1+!\relax\relax
184 }%
185 \def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint:#1]%
186 {%
187     \fi\csname XINT_ggcfrac_opt#1\endcsname
188 }%
189 \def\XINT_ggcfrac_optl #1%
190 {%
191     \XINT_ggcfrac #1+!\relax\hfill
192 }%
193 \def\XINT_ggcfrac_optc #1%
194 {%
195     \XINT_ggcfrac #1+!\relax\relax
196 }%
197 \def\XINT_ggcfrac_optr #1%
198 {%
199     \XINT_ggcfrac #1+!\hfill\relax
200 }%
201 \def\XINT_ggcfrac
202 {%
203     \expandafter\XINT_ggcfrac_enter\romannumeral`&&@%
204 }%
205 \def\XINT_ggcfrac_enter {\XINT_ggcfrac_loop {}}%
206 \def\XINT_ggcfrac_loop #1#2+#3/%
207 {%
208     \xint_gob_til_exclam #3\XINT_ggcfrac_endloop!%
209     \XINT_ggcfrac_loop {{#3}{#2}{#1}}%
210 }%
211 \def\XINT_ggcfrac_endloop!\XINT_ggcfrac_loop #1#2#3%
212 {%
213     \XINT_ggcfrac_T #2#3#1!!%
214 }%
215 \def\XINT_ggcfrac_T #1#2#3#4{\XINT_ggcfrac_U #1#2{#4}}%
216 \def\XINT_ggcfrac_U #1#2#3#4#5%
217 {%
218     \xint_gob_til_exclam #5\XINT_ggcfrac_end!\XINT_ggcfrac_U
219         #1#2{#5+\cfrac{#1#4#2}{#3}}%
220 }%
221 \def\XINT_ggcfrac_end!\XINT_ggcfrac_U #1#2#3%
222 {%
223     \XINT_ggcfrac_end_b #3%

```



```
224 }%
225 \def\XINT_ggcfrac_end_b #1\cfrac#2#3{ #3}%
```

10.6 \xintGctoGCx

```
226 \def\xintGctoGCx {\romannumeral0\xintgctogcx }%
227 \def\xintgctogcx #1#2#3%
228 {%
229   \expandafter\XINT_gctgcx_start\expandafter {\romannumeral`&&@#3}{#1}{#2}%
230 }%
231 \def\XINT_gctgcx_start #1#2#3{\XINT_gctgcx_loop_a {}{#2}{#3}#1+!/}%
232 \def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
233 {%
234   \xint_gob_til_exclam #5\XINT_gctgcx_end!%
235   \XINT_gctgcx_loop_b {#1{#4}}{#2{#5}#3}{#2}{#3}%
236 }%
237 \def\XINT_gctgcx_loop_b #1#2%
238 {%
239   \XINT_gctgcx_loop_a {#1#2}%
240 }%
241 \def\XINT_gctgcx_end!\XINT_gctgcx_loop_b #1#2#3#4{ #1}%
```

10.7 \xintFtoCs

Modified in 1.09m: a space is added after the inserted commas.

```
242 \def\xintFtoCs {\romannumeral0\xintftocs }%
243 \def\xintftocs #1%
244 {%
245   \expandafter\XINT_ftc_A\romannumeral0\xintrawwithzeros {#1}\Z
246 }%
247 \def\XINT_ftc_A #1/#2\Z
248 {%
249   \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
250 }%
251 \def\XINT_ftc_B #1#2%
252 {%
253   \XINT_ftc_C #2.{#1}%
254 }%
255 \def\XINT_ftc_C #1%
256 {%
257   \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
258 }%
259 \def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
260 \def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
261 \def\XINT_ftc_loop_a
262 {%
263   \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_div_prepare
264 }%
265 \def\XINT_ftc_loop_d #1#2%
266 {%
267   \XINT_ftc_loop_e #2.{#1}%
268 }%
269 \def\XINT_ftc_loop_e #1%
```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [indices](#)

```

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

```

10.8 \xintFtoCx

```

278 \def\xintFtoCx {\romannumeral0\xintftocx }%
279 \def\xintftocx #1#2%
280 {%
281   \expandafter\XINT_ftcx_A\romannumeral0\xintraewithzeros {#2}\Z {#1}%
282 }%
283 \def\XINT_ftcx_A #1/#2\Z
284 {%
285   \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
286 }%
287 \def\XINT_ftcx_B #1#2%
288 {%
289   \XINT_ftcx_C #2.{#1}%
290 }%
291 \def\XINT_ftcx_C #1%
292 {%
293   \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
294 }%
295 \def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
296 \def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{#2#4}{#4}}%
297 \def\XINT_ftcx_loop_a
298 {%
299   \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_div_prepare
300 }%
301 \def\XINT_ftcx_loop_d #1#2%
302 {%
303   \XINT_ftcx_loop_e #2.{#1}%
304 }%
305 \def\XINT_ftcx_loop_e #1%
306 {%
307   \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
308 }%
309 \def\XINT_ftcx_loop_f #1.#2#3#4#5%
310 {%
311   \XINT_ftcx_loop_a {#1}{#3}{#1}{#4#2#5}{#5}%
312 }%
313 \def\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1.#2#3#4#5{ #4#2}%

```

10.9 \xintFtoC

New in 1.09m: this is the same as \xintFtoCx with empty separator. I had temporarily during preparation of 1.09m removed braces from \xintFtoCx, but I recalled later why that was useful (see doc), thus let's just here do \xintFtoCx {}

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfraction](#), [xintexpr](#), [indices](#)

```
314 \def\xintFtoC {\romannumeral0\xintftoc}%
315 \def\xintftoc {\xintftocx {}}%
```

10.10 \xintFtoGC

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

10.11 \xintFGtoC

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

```
318 \def\xintFGtoC {\romannumeral0\xintfgtoc}%
319 \def\xintfgtoc#1%
320 {%
321   \expandafter\XINT_fgtc_a\romannumeral0\xintraewithzeros {#1}\Z
322 }%
323 \def\XINT_fgtc_a #1/#2\Z #3%
324 {%
325   \expandafter\XINT_fgtc_b\romannumeral0\xintraewithzeros {#3}\Z #1/#2\Z { }%
326 }%
327 \def\XINT_fgtc_b #1/#2\Z
328 {%
329   \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
330 }%
331 \def\XINT_fgtc_c #1#2#3#4/#5\Z
332 {%
333   \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
334     {#4}{#5}{#5}{#1}{#2}{#3}%
335 }%
336 \def\XINT_fgtc_d #1#2#3#4#5#6#7%
337 {%
338   \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
339     {\xint_thirdofthree}%
340 }%
341 \def\XINT_fgtc_da #1#2#3#4#5#6#7%
342 {%
343   \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7}{#1}}%
344 }%
345 \def\XINT_fgtc_e #1%
346 {%
347   \xintiiifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
348     {\XINT_fgtc_f {#1}}%
349 }%
350 \def\XINT_fgtc_f #1#2%
351 {%
352   \xintiiifZero {#2}{\xint_thirdofthree}{\XINT_fgtc_g {#1}{#2}}%
353 }%
354 \def\XINT_fgtc_g #1#2#3%
355 {%
356   \expandafter\XINT_fgtc_h\romannumeral0\XINT_div_prepare {#1}{#3}{#1}{#2}%
357 }%
358 \def\XINT_fgtc_h #1#2#3#4#5%
```

```

359 {%
360     \expandafter\XINT_fgtd\romannumeral0\XINT_div_prepare
361         {#4}{#5}{#4}{#1}{#2}{#3}%
362 }%

```

10.12 \xintFtoCC

```

363 \def\xintFtoCC {\romannumeral0\xintftocc }%
364 \def\xintftocc #1%
365 {%
366     \expandafter\XINT_ftcc_A\expandafter {\romannumeral0\xintrawwithzeros {#1}}%
367 }%
368 \def\XINT_ftcc_A #1%
369 {%
370     \expandafter\XINT_ftcc_B
371     \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1[0]}}\Z {#1[0]}%
372 }%
373 \def\XINT_ftcc_B #1/#2\Z
374 {%
375     \expandafter\XINT_ftcc_C\expandafter {\romannumeral0\xintiiquo {#1}{#2}}%
376 }%
377 \def\XINT_ftcc_C #1#2%
378 {%
379     \expandafter\XINT_ftcc_D\romannumeral0\xintsub {#2}{#1}\Z {#1}%
380 }%
381 \def\XINT_ftcc_D #1%
382 {%
383     \xint_UDzerominusfork
384     #1-\XINT_ftcc_integer
385     0#1\XINT_ftcc_En
386     0-{\XINT_ftcc_Ep #1}%
387     \krof
388 }%
389 \def\XINT_ftcc_Ep #1\Z #2%
390 {%
391     \expandafter\XINT_ftcc_loop_a\expandafter
392     {\romannumeral0\xintdiv {1[0]}{#1}}{#2+1/}%
393 }%
394 \def\XINT_ftcc_En #1\Z #2%
395 {%
396     \expandafter\XINT_ftcc_loop_a\expandafter
397     {\romannumeral0\xintdiv {1[0]}{#1}}{#2+-1/}%
398 }%
399 \def\XINT_ftcc_integer #1\Z #2{ #2}%
400 \def\XINT_ftcc_loop_a #1%
401 {%
402     \expandafter\XINT_ftcc_loop_b
403     \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1}}\Z {#1}%
404 }%
405 \def\XINT_ftcc_loop_b #1/#2\Z
406 {%
407     \expandafter\XINT_ftcc_loop_c\expandafter
408     {\romannumeral0\xintiiquo {#1}{#2}}%

```

```

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

```

10.13 \xintCtoF, \xintCstoF

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

```

434 \def\xintCstoF {\romannumeral0\xintcstof }%
435 \def\xintcstof #1%
436 {%
437   \expandafter\XINT_ctf_prep \romannumeral0\xintcsvtolist{#1}!%
438 }%
439 \def\xintCtoF {\romannumeral0\xintctof }%
440 \def\xintctof #1%
441 {%
442   \expandafter\XINT_ctf_prep \romannumeral`&&@#1!%
443 }%
444 \def\XINT_ctf_prep
445 {%
446   \XINT_ctf_loop_a 1001%
447 }%
448 \def\XINT_ctf_loop_a #1#2#3#4#5%
449 {%
450   \xint_gob_til_exclam #5\XINT_ctf_end!%
451   \expandafter\XINT_ctf_loop_b
452   \romannumeral0\xintraewithzeros {#5}.\{#1}\{#2}\{#3}\{#4}%
453 }%
454 \def\XINT_ctf_loop_b #1/#2.#3#4#5#6%
455 {%

```

```

456 \expandafter\XINT_ctf_loop_c\expandafter
457 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
458 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
459 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
460 {\XINT_mul_fork #1\xint:#4\xint:}}%
461 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
462 {\XINT_mul_fork #1\xint:#3\xint:}}%
463 }%
464 \def\XINT_ctf_loop_c #1#2%
465 {%
466 \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
467 }%
468 \def\XINT_ctf_loop_d #1#2%
469 {%
470 \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}#1}%
471 }%
472 \def\XINT_ctf_loop_e #1#2%
473 {%
474 \expandafter\XINT_ctf_loop_a\expandafter{#2}#1%
475 }%
476 \def\XINT_ctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.14 \xintiCstoF

```

477 \def\xintiCstoF {\romannumeral0\xinticstof }%
478 \def\xinticstof #1%
479 {%
480 \expandafter\XINT_icstf_prep \romannumeral`&&@#1,!,%
481 }%
482 \def\XINT_icstf_prep
483 {%
484 \XINT_icstf_loop_a 1001%
485 }%
486 \def\XINT_icstf_loop_a #1#2#3#4#5,%
487 {%
488 \xint_gob_til_exclam #5\XINT_icstf_end!%
489 \expandafter
490 \XINT_icstf_loop_b \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
491 }%
492 \def\XINT_icstf_loop_b #1.#2#3#4#5%
493 {%
494 \expandafter\XINT_icstf_loop_c\expandafter
495 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
496 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
497 {#2}{#3}%
498 }%
499 \def\XINT_icstf_loop_c #1#2%
500 {%
501 \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
502 }%
503 \def\XINT_icstf_end#1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.15 \xintGctoF

```

504 \def\xintGctoF {\romannumeral0\xintgctof }%
505 \def\xintgctof #1%
506 {%
507     \expandafter\xINT_gctf_prep \romannumeral`&&@#1+!/ %
508 }%
509 \def\xINT_gctf_prep
510 {%
511     \XINT_gctf_loop_a 1001%
512 }%
513 \def\xINT_gctf_loop_a #1#2#3#4#5+%
514 {%
515     \expandafter\xINT_gctf_loop_b
516     \romannumeral0\xintraewithzeros {#5}. {#1}{#2}{#3}{#4}%
517 }%
518 \def\xINT_gctf_loop_b #1/#2.#3#4#5#6%
519 {%
520     \expandafter\xINT_gctf_loop_c\expandafter
521     {\romannumeral0\xINT_mul_fork #2\xint:#4\xint:}%
522     {\romannumeral0\xINT_mul_fork #2\xint:#3\xint:}%
523     {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
524         {\XINT_mul_fork #1\xint:#4\xint:}}%
525     {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
526         {\XINT_mul_fork #1\xint:#3\xint:}}%
527 }%
528 \def\xINT_gctf_loop_c #1#2%
529 {%
530     \expandafter\xINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
531 }%
532 \def\xINT_gctf_loop_d #1#2%
533 {%
534     \expandafter\xINT_gctf_loop_e\expandafter {\expandafter{#2}{#1}}%
535 }%
536 \def\xINT_gctf_loop_e #1#2%
537 {%
538     \expandafter\xINT_gctf_loop_f\expandafter {\expandafter{#2}{#1}}%
539 }%
540 \def\xINT_gctf_loop_f #1#2/%
541 {%
542     \xint_gob_til_exclam #2\xINT_gctf_end!%
543     \expandafter\xINT_gctf_loop_g
544     \romannumeral0\xintraewithzeros {#2}.#1%
545 }%
546 \def\xINT_gctf_loop_g #1/#2.#3#4#5#6%
547 {%
548     \expandafter\xINT_gctf_loop_h\expandafter
549     {\romannumeral0\xINT_mul_fork #1\xint:#6\xint:}%
550     {\romannumeral0\xINT_mul_fork #1\xint:#5\xint:}%
551     {\romannumeral0\xINT_mul_fork #2\xint:#4\xint:}%
552     {\romannumeral0\xINT_mul_fork #2\xint:#3\xint:}%
553 }%
554 \def\xINT_gctf_loop_h #1#2%
555 {%

```

```

556 \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
557 }%
558 \def\XINT_gctf_loop_i #1#2%
559 {%
560 \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}{#1}}%
561 }%
562 \def\XINT_gctf_loop_j #1#2%
563 {%
564 \expandafter\XINT_gctf_loop_a\expandafter {#2}{#1}%
565 }%
566 \def\XINT_gctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.16 \xintiGctoF

```

567 \def\xintiGctoF {\romannumeral0\xintigctoF}%
568 \def\xintigctoF #1%
569 {%
570 \expandafter\XINT_igctf_prep \romannumeral`&&@#1+!/ %
571 }%
572 \def\XINT_igctf_prep
573 {%
574 \XINT_igctf_loop_a 1001%
575 }%
576 \def\XINT_igctf_loop_a #1#2#3#4#5+%
577 {%
578 \expandafter\XINT_igctf_loop_b
579 \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
580 }%
581 \def\XINT_igctf_loop_b #1.#2#3#4#5%
582 {%
583 \expandafter\XINT_igctf_loop_c\expandafter
584 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
585 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
586 {#2}{#3}%
587 }%
588 \def\XINT_igctf_loop_c #1#2%
589 {%
590 \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
591 }%
592 \def\XINT_igctf_loop_f #1#2#3#4/%
593 {%
594 \xint_gob_til_exclam #4\XINT_igctf_end!%
595 \expandafter\XINT_igctf_loop_g
596 \romannumeral`&&@#4.{#2}{#3}{#1}%
597 }%
598 \def\XINT_igctf_loop_g #1.#2#3%
599 {%
600 \expandafter\XINT_igctf_loop_h\expandafter
601 {\romannumeral0\XINT_mul_fork #1\xint:#3\xint:}%
602 {\romannumeral0\XINT_mul_fork #1\xint:#2\xint:}%
603 }%
604 \def\XINT_igctf_loop_h #1#2%
605 {%
606 \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%

```



```

607 }%
608 \def\XINT_igctf_loop_i #1#2#3#4%
609 {%
610   \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%
611 }%
612 \def\XINT_igctf_end #1.#2#3#4#5{\xintrawwithzeros {#4/#5}}% 1.09b removes [0]

```

10.17 \xintCtoCv, \xintCstoCv

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

```

613 \def\xintCstoCv {\romannumeral0\xintcstocv }%
614 \def\xintcstocv #1%
615 {%
616   \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}!%
617 }%
618 \def\xintCtoCv {\romannumeral0\xintctocv }%
619 \def\xintctocv #1%
620 {%
621   \expandafter\XINT_ctcv_prep\romannumeral`&&@#1!%
622 }%
623 \def\XINT_ctcv_prep
624 {%
625   \XINT_ctcv_loop_a {}1001%
626 }%
627 \def\XINT_ctcv_loop_a #1#2#3#4#5#6%
628 {%
629   \xint_gob_til_exclam #6\XINT_ctcv_end!%
630   \expandafter\XINT_ctcv_loop_b
631   \romannumeral0\xintrawwithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
632 }%
633 \def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
634 {%
635   \expandafter\XINT_ctcv_loop_c\expandafter
636   {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
637   {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
638   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
639     {\XINT_mul_fork #1\xint:#4\xint:}}%
640   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
641     {\XINT_mul_fork #1\xint:#3\xint:}}%
642 }%
643 \def\XINT_ctcv_loop_c #1#2%
644 {%
645   \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
646 }%
647 \def\XINT_ctcv_loop_d #1#2%
648 {%
649   \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}#1}%
650 }%
651 \def\XINT_ctcv_loop_e #1#2%
652 {%
653   \expandafter\XINT_ctcv_loop_f\expandafter{#2}#1%

```

```

654 }%
655 \def\XINT_ctcv_loop_f #1#2#3#4#5%
656 {%
657     \expandafter\XINT_ctcv_loop_g\expandafter
658     {\romannumeral0\xintraawithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
659 }%
660 \def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
661 \def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%

```

10.18 \xintiCstoCv

```

662 \def\xintiCstoCv {\romannumeral0\xinticstocv }%
663 \def\xinticstocv #1%
664 {%
665     \expandafter\XINT_icstcv_prep \romannumeral`&&@#1,!,%
666 }%
667 \def\XINT_icstcv_prep
668 {%
669     \XINT_icstcv_loop_a {}1001%
670 }%
671 \def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
672 {%
673     \xint_gob_til_exclam #6\XINT_icstcv_end!%
674     \expandafter
675     \XINT_icstcv_loop_b \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
676 }%
677 \def\XINT_icstcv_loop_b #1.#2#3#4#5%
678 {%
679     \expandafter\XINT_icstcv_loop_c\expandafter
680     {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
681     {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
682     {{#2}{#3}}%
683 }%
684 \def\XINT_icstcv_loop_c #1#2%
685 {%
686     \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
687 }%
688 \def\XINT_icstcv_loop_d #1#2%
689 {%
690     \expandafter\XINT_icstcv_loop_e\expandafter
691     {\romannumeral0\xintraawithzeros {#1/#2}}{#1}{#2}}%
692 }%
693 \def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
694 \def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}% 1.09b removes [0]

```

10.19 \xintGctoCv

```

695 \def\xintGctoCv {\romannumeral0\xintgctocv }%
696 \def\xintgctocv #1%
697 {%
698     \expandafter\XINT_gctcv_prep \romannumeral`&&@#1+!/,%
699 }%
700 \def\XINT_gctcv_prep
701 {%

```

```

702 \XINT_gctcv_loop_a {}1001%
703 }%
704 \def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
705 {%
706 \expandafter\XINT_gctcv_loop_b
707 \romannumeral0\xintraawithzeros {#6}. {#2}{#3}{#4}{#5}{#1}%
708 }%
709 \def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
710 {%
711 \expandafter\XINT_gctcv_loop_c\expandafter
712 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
713 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
714 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
715 {\XINT_mul_fork #1\xint:#4\xint:}}%
716 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
717 {\XINT_mul_fork #1\xint:#3\xint:}}%
718 }%
719 \def\XINT_gctcv_loop_c #1#2%
720 {%
721 \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
722 }%
723 \def\XINT_gctcv_loop_d #1#2%
724 {%
725 \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
726 }%
727 \def\XINT_gctcv_loop_e #1#2%
728 {%
729 \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
730 }%
731 \def\XINT_gctcv_loop_f #1#2%
732 {%
733 \expandafter\XINT_gctcv_loop_g\expandafter
734 {\romannumeral0\xintraawithzeros {#1/#2}}{#1}{#2}}%
735 }%
736 \def\XINT_gctcv_loop_g #1#2#3#4%
737 {%
738 \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
739 }%
740 \def\XINT_gctcv_loop_h #1#2#3/%
741 {%
742 \xint_gob_til_exclam #3\XINT_gctcv_end!%
743 \expandafter\XINT_gctcv_loop_i
744 \romannumeral0\xintraawithzeros {#3}.#2{#1}%
745 }%
746 \def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
747 {%
748 \expandafter\XINT_gctcv_loop_j\expandafter
749 {\romannumeral0\XINT_mul_fork #1\xint:#6\xint:}%
750 {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
751 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
752 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
753 }%

```

```

754 \def\XINT_gctcv_loop_j #1#2%
755 {%
756   \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
757 }%
758 \def\XINT_gctcv_loop_k #1#2%
759 {%
760   \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}{#1}}%
761 }%
762 \def\XINT_gctcv_loop_l #1#2%
763 {%
764   \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}{#1}}%
765 }%
766 \def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}{#1}}%
767 \def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%

```

10.20 \xintiGtoCv

```

768 \def\xintiGtoCv {\romannumeral0\xintigctocv}%
769 \def\xintigctocv #1%
770 {%
771   \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+!/ %
772 }%
773 \def\XINT_igctcv_prep
774 {%
775   \XINT_igctcv_loop_a {}1001%
776 }%
777 \def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
778 {%
779   \expandafter\XINT_igctcv_loop_b
780   \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
781 }%
782 \def\XINT_igctcv_loop_b #1.#2#3#4#5%
783 {%
784   \expandafter\XINT_igctcv_loop_c\expandafter
785   {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
786   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
787   {{#2}{#3}}%
788 }%
789 \def\XINT_igctcv_loop_c #1#2%
790 {%
791   \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
792 }%
793 \def\XINT_igctcv_loop_f #1#2#3#4/%
794 {%
795   \xint_gob_til_exclam #4\XINT_igctcv_end_a!%
796   \expandafter\XINT_igctcv_loop_g
797   \romannumeral`&&@#4.#1#2{#3}%
798 }%
799 \def\XINT_igctcv_loop_g #1.#2#3#4#5%
800 {%
801   \expandafter\XINT_igctcv_loop_h\expandafter
802   {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
803   {\romannumeral0\XINT_mul_fork #1\xint:#4\xint:}%
804   {{#2}{#3}}%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

805 }%
806 \def\XINT_igctcv_loop_h #1#2%
807 {%
808     \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
809 }%
810 \def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
811 \def\XINT_igctcv_loop_k #1#2%
812 {%
813     \expandafter\XINT_igctcv_loop_l\expandafter
814     {\romannumeral0\xintraawithzeros {#1/#2}}%
815 }%
816 \def\XINT_igctcv_loop_l #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
817 \def\XINT_igctcv_end_a #1.#2#3#4#5%
818 {%
819     \expandafter\XINT_igctcv_end_b\expandafter
820     {\romannumeral0\xintraawithzeros {#2/#3}}%
821 }%
822 \def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]

```

10.21 \xintFtoCv

Still uses \xinticstocv \xintFtoCs rather than \xintctocv \xintFtoC.

```

823 \def\xintFtoCv {\romannumeral0\xintftocv }%
824 \def\xintftocv #1%
825 {%
826     \xinticstocv {\xintFtoCs {#1}}%
827 }%

```

10.22 \xintFtoCCv

```

828 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
829 \def\xintftoccv #1%
830 {%
831     \xintigctocv {\xintFtoCC {#1}}%
832 }%

```

10.23 \xintCntoF

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

833 \def\xintCntoF {\romannumeral0\xintcntof }%
834 \def\xintcntof #1%
835 {%
836     \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
837 }%
838 \def\XINT_cntf #1#2%
839 {%
840     \ifnum #1>\xint_c_
841         \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
842             {\the\numexpr #1-1\expandafter}\expandafter
843             {\romannumeral`&&#2{#1}}{#2}}%
844     \else

```

```

845     \xint_afterfi
846     {\ifnum #1=\xint_c_
847       \xint_afterfi {\expandafter\space \romannumeral`&&#2{0}}}%
848       \else \xint_afterfi { }% 1.09m now returns nothing.
849     \fi}%
850 \fi
851 }%
852 \def\xint_cntf_loop #1#2#3%
853 {%
854   \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
855   \expandafter\xint_cntf_loop\expandafter
856   {\the\numexpr #1-1\expandafter }\expandafter
857   {\romannumeral0\xintadd {\xintDiv {1[0]}\{#2}\{#3{#1}}}%
858   {#3}%
859 }%
860 \def\xint_cntf_exit \fi
861   \expandafter\xint_cntf_loop\expandafter
862   #1\expandafter #2#3%
863 {%
864   \fi\xint_gobble_ii #2%
865 }%

```

10.24 \xintGCntoF

Modified in 1.06 to give the N argument first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

866 \def\xintGCntoF {\romannumeral0\xintgcntof }%
867 \def\xintgcntof #1%
868 {%
869   \expandafter\xint_gcntf\expandafter {\the\numexpr #1}%
870 }%
871 \def\xint_gcntf #1#2#3%
872 {%
873   \ifnum #1>\xint_c_
874     \xint_afterfi {\expandafter\xint_gcntf_loop\expandafter
875       {\the\numexpr #1-1\expandafter}\expandafter
876       {\romannumeral`&&#2{#1}\{#2}\{#3}}}%
877   \else
878     \xint_afterfi
879     {\ifnum #1=\xint_c_
880       \xint_afterfi {\expandafter\space\romannumeral`&&#2{0}}}%
881       \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
882     \fi}%
883 \fi
884 }%
885 \def\xint_gcntf_loop #1#2#3#4%
886 {%
887   \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
888   \expandafter\xint_gcntf_loop\expandafter
889   {\the\numexpr #1-1\expandafter }\expandafter
890   {\romannumeral0\xintadd {\xintDiv {#4{#1}\{#2}\{#3{#1}}}%
891   {#3}\{#4}%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```

892 }%
893 \def\XINT_gcntf_exit \fi
894   \expandafter\XINT_gcntf_loop\expandafter
895   #1\expandafter #2#3#4%
896 {%
897   \fi\xint_gobble_ii #2%
898 }%

```

10.25 \xintCntoCs

Modified in 1.09m: added spaces after the commas in the produced list. Moreover the coefficients are not braced anymore. A slight induced limitation is that the macro argument should not contain some explicit comma (cf. `\XINT_cntcs_exit_b`), hence `\xintCntoCs {\macro,}` with `\def\macro,#1{<stuff>}` would crash. Not a very serious limitation, I believe.

```

899 \def\xintCntoCs {\romannumeral0\xintcntocs }%
900 \def\xintcntocs #1%
901 {%
902   \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
903 }%
904 \def\XINT_cntcs #1#2%
905 {%
906   \ifnum #1<0
907     \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
908   \else
909     \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
910                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
911                   {\romannumeral`&&@#2{#1}}{#2}}% produced coeff not braced
912   \fi
913 }%
914 \def\XINT_cntcs_loop #1#2#3%
915 {%
916   \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
917   \expandafter\XINT_cntcs_loop\expandafter
918   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
919   {\romannumeral`&&@#3{#1}, #2}{#3}% space added, 1.09m
920 }%
921 \def\XINT_cntcs_exit \fi
922   \expandafter\XINT_cntcs_loop\expandafter
923   #1\expandafter #2#3%
924 {%
925   \fi\XINT_cntcs_exit_b #2%
926 }%
927 \def\XINT_cntcs_exit_b #1,{}% romannumeral stopping space already there

```

10.26 \xintCntoGC

Modified in 1.06 to give the N first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that.

1.09m maintains the braces, as the coeff are allowed to be fraction and the slash can not be naked in the GC format, contrarily to what happens in `\xintCntoCs`. Also the separators given to `\xintGctoGCx` may then fetch the coefficients as argument, as they are braced.

```

928 \def\xintCntoGC {\romannumeral0\xintcntogc }%
929 \def\xintcntogc #1%
930 {%
931     \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
932 }%
933 \def\XINT_cntgc #1#2%
934 {%
935     \ifnum #1<0
936         \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
937     \else
938         \xint_afterfi {\expandafter\XINT_cntgc_loop\expandafter
939             {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
940             {\expandafter{\romannumeral`&&@#2{#1}}{#2}}}%
941     \fi
942 }%
943 \def\XINT_cntgc_loop #1#2#3%
944 {%
945     \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
946     \expandafter\XINT_cntgc_loop\expandafter
947     {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
948     {\expandafter{\romannumeral`&&@#3{#1}}+1/#2}{#3}%
949 }%
950 \def\XINT_cntgc_exit \fi
951     \expandafter\XINT_cntgc_loop\expandafter
952     #1\expandafter #2#3%
953 {%
954     \fi\XINT_cntgc_exit_b #2%
955 }%
956 \def\XINT_cntgc_exit_b #1+1/{ }%

```

10.27 \xintGCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

957 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
958 \def\xintgcntogc #1%
959 {%
960     \expandafter\XINT_gcntgc\expandafter {\the\numexpr #1}%
961 }%
962 \def\XINT_gcntgc #1#2#3%
963 {%
964     \ifnum #1<0
965         \xint_afterfi { }% 1.09i now returns nothing
966     \else
967         \xint_afterfi {\expandafter\XINT_gcntgc_loop\expandafter
968             {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
969             {\expandafter{\romannumeral`&&@#2{#1}}{#2}{#3}}}%
970     \fi
971 }%
972 \def\XINT_gcntgc_loop #1#2#3#4%
973 {%
974     \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi

```



```

975 \expandafter\XINT_gcntgc_loop_b\expandafter
976 {\expandafter{\romannumeral`&&@#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
977 }%
978 \def\XINT_gcntgc_loop_b #1#2#3%
979 {%
980 \expandafter\XINT_gcntgc_loop\expandafter
981 {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
982 {\expandafter{\romannumeral`&&@#2}+#1}%
983 }%
984 \def\XINT_gcntgc_exit \fi
985 \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
986 {%
987 \fi\XINT_gcntgc_exit_b #1%
988 }%
989 \def\XINT_gcntgc_exit_b #1/{ }%

```

10.28 \xintCstoGC

```

990 \def\xintCstoGC {\romannumeral0\xintcstogc }%
991 \def\xintcstogc #1%
992 {%
993 \expandafter\XINT_cstc_prep \romannumeral`&&@#1,!,%
994 }%
995 \def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
996 \def\XINT_cstc_loop_a #1#2,%
997 {%
998 \xint_gob_til_exclam #2\XINT_cstc_end!%
999 \XINT_cstc_loop_b {#1}{#2}%
1000 }%
1001 \def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
1002 \def\XINT_cstc_end!\XINT_cstc_loop_b #1#2{ #1}%

```

10.29 \xintGctoGC

```

1003 \def\xintGctoGC {\romannumeral0\xintgctogc }%
1004 \def\xintgctogc #1%
1005 {%
1006 \expandafter\XINT_gctgc_start \romannumeral`&&@#1+!/,%
1007 }%
1008 \def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
1009 \def\XINT_gctgc_loop_a #1#2+#3/%
1010 {%
1011 \xint_gob_til_exclam #3\XINT_gctgc_end!%
1012 \expandafter\XINT_gctgc_loop_b\expandafter
1013 {\romannumeral`&&@#2}{#3}{#1}%
1014 }%
1015 \def\XINT_gctgc_loop_b #1#2%
1016 {%
1017 \expandafter\XINT_gctgc_loop_c\expandafter
1018 {\romannumeral`&&@#2}{#1}%
1019 }%
1020 \def\XINT_gctgc_loop_c #1#2#3%
1021 {%
1022 \XINT_gctgc_loop_a {#3{#2}+{#1}}/%

```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices

```
1023 }%
1024 \def\XINT_gctgc_end!\expandafter\XINT_gctgc_loop_b
1025 {%
1026     \expandafter\XINT_gctgc_end_b
1027 }%
1028 \def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
1029 \XINT_restorecatcodes_endinput%
```

11 Package [xintexpr](#) implementation

Contents

11.1	Old comments	292
11.2	Catcodes, ε -TeX and reload detection	293
11.3	Package identification	294
11.4	Locking and unlocking	294
11.5	<code>\XINT_expr_wrap</code> , <code>\XINT_iexpr_wrap</code>	295
11.6	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	295
11.7	<code>\XINT_expr_print</code> , <code>\XINT_iexpr_print</code> , <code>\XINT_boolexpr_print</code>	295
11.8	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiexpr</code>	295
11.9	<code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiexpr</code>	295
11.10	<code>\xintthe</code>	295
11.11	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiexpr</code>	295
11.12	<code>\xintthecoords</code>	295
11.13	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	296
11.14	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	296
11.15	<code>\xinteval</code> , <code>\xintiieval</code>	296
11.16	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	296
11.17	<code>\xintfloateval</code> , <code>\XINT_flexexpr_wrap</code> , <code>\XINT_flexexpr_print</code>	297
11.18	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	297
11.19	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliexpr</code>	297
11.20	Hooks for the functioning of <code>\xintNewExpr</code> and <code>\xintdeffunc</code>	297
11.21	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	298
11.22	<code>\XINT_expr_getnext</code> : fetching some number then an operator	299
11.23	<code>\XINT_expr_scan_nbr_or_func</code> : the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	300
11.24	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	307
11.25	Expansion spanning; opening and closing parentheses	309
11.26	<code> </code> , <code> </code> , <code>&</code> , <code>&&</code> , <code><</code> , <code>></code> , <code>=</code> , <code>==</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>..</code> , <code>..[</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , and <code>++</code> operators	310
11.27	Macros for list selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	317
11.28	Macros for a..b list generation	322
11.29	Macros for a..[d]..b list generation	324
11.30	The comma as binary operator	326
11.31	The minus as prefix operator of variable precedence level	326
11.32	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches	327
11.33	<code>!</code> as postfix factorial operator	328
11.34	The A/B[N] mechanism	328
11.35	<code>\XINT_expr_op_`</code> for recognizing functions	328
11.36	The <code>\bool()</code> , <code>\togl()</code> , <code>\protect()</code> pseudo “functions”	329
11.37	The <code>\break()</code> function	329
11.38	The <code>\qraw()</code> , <code>\qint()</code> , <code>\qfrac()</code> , and <code>\qfloat()</code> “functions”	329
11.39	The <code>\random()</code> and <code>\grand()</code> “functions”	330
11.40	<code>\XINT_expr_op__</code> for recognizing variables	330
11.41	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	330
11.42	<code>\xintunassignvar</code>	332
11.43	<code>seq</code> and the implementation of dummy variables	333
11.44	<code>\add()</code> , <code>\mul()</code>	339
11.45	<code>\subs()</code>	340
11.46	<code>\rseq()</code>	340

11.47	<code>\iter()</code>	342
11.48	<code>\rrseq()</code>	343
11.49	<code>\iterr()</code>	345
11.50	Macros handling csv lists for functions with multiple comma separated arguments in expressions	347
11.51	Auxiliary wrappers for function macros	350
11.52	The <code>num()</code> , <code>reduce()</code> , <code>preduce()</code> , <code>abs()</code> , <code>sgn()</code> , <code>frac()</code> , <code>floor()</code> , <code>ceil()</code> , <code>sqr()</code> , <code>sqrt()</code> , <code>sqrtr()</code> , <code>float()</code> , <code>round()</code> , <code>trunc()</code> , <code>mod()</code> , <code>quo()</code> , <code>rem()</code> , <code>divmod()</code> , <code>gcd()</code> , <code>lcm()</code> , <code>max()</code> , <code>min()</code> , <code>`+`()</code> , <code>`*`()</code> , <code>?()</code> , <code>!()</code> , <code>not()</code> , <code>all()</code> , <code>any()</code> , <code>xor()</code> , <code>if()</code> , <code>ifsgn()</code> , <code>ifint()</code> , <code>ifone()</code> , <code>even()</code> , <code>odd()</code> , <code>first()</code> , <code>last()</code> , <code>len()</code> , <code>reversed()</code> , <code>factorial()</code> , <code>binomial()</code> and <code>randrange()</code> functions	351
11.53	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	362
11.54	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	364
11.55	<code>\xintNewFunction</code>	365
11.56	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	366

This is release 1.3c of [2018/06/17].

11.1 Old comments

These general comments were last updated at the end of the 1.09x series in 2014. The principles remain in place to this day but refer to [CHANGES.html](#) for some significant evolutions since.

The first version was released in June 2013. I was greatly helped in this task of writing an expandable parser of infix operations by the comments provided in [l3fp-parse.dtx](#) (in its version as available in April-May 2013). One will recognize in particular the idea of the 'until' macros; I have not looked into the actual [l3fp](#) code beyond the very useful comments provided in its documentation.

A main worry was that my data has no a priori bound on its size; to keep the code reasonably efficient, I experimented with a technique of storing and retrieving data expandably as *names* of control sequences. Intermediate computation results are stored as control sequences `\.=a/b[n]`.

Roughly speaking, the parser mechanism is as follows: at any given time the last found 'operator' has its associated `until` macro awaiting some news from the token flow; first `getnext` expands forward in the hope to construct some number, which may come from a parenthesized sub-expression, from some braced material, or from a digit by digit scan. After this number has been formed the next operator is looked for by the `getop` macro. Once `getop` has finished its job, `until` is presented with three tokens: the first one is the precedence level of the new found operator (which may be an end of expression marker), the second is the operator character token (earlier versions had here already some macro name, but in order to keep as much common code to `expr` and `floatexpr` common as possible, this was modified) of the new found operator, and the third one is the newly found number (which was encountered just before the new operator).

The `until` macro of the earlier operator examines the precedence level of the new found one, and either executes the earlier operator (in the case of a binary operation, with the found number and a previously stored one) or it delays execution, giving the hand to the `until` macro of the operator having been found of higher precedence.

A minus sign acting as prefix gets converted into a (unary) operator inheriting the precedence level of the previous operator.

Once the end of the expression is found (it has to be marked by a `\relax`) the final result is output as four tokens (five tokens since 1.09j) the first one a catcode 11 exclamation mark, the second one an error generating macro, the third one is a protection mechanism, the fourth one a printing macro and the fifth is `\.=a/b[n]`. The prefix `\xintthe` makes the output printable by killing the first three tokens.

11.2 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

50      \fi
51      \fi
52      \fi
53 \z%
54 \XINTsetupcatcodes%

```

11.3 Package identification

```

55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2018/06/17 1.3c Expandable expression parser (JFB)]%
58 \catcode\! 11
59 \let\XINT_Cmp \xintiiCmp

```

11.4 Locking and unlocking

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannu-
meral-`0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed expan-
sion. In this way no more limit at 5000 digits, and besides this is a logical move because the
`\xintexpr` parser is already based on `\csname...\endcsname` storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch
computations with thousands of digits... such computations are still slow with 1.2 but less so now.
Chains or `\romannu-meral` are still used for the gathering of function names and other stuff which
I have half-forgotten because the parser does many things.

In the earlier versions we used the `lockscan` macro after a chain of `\romannu-meral-`0` had ended
gathering digits; this uses has been replaced by direct processing inside a `\csname...\endcsname`
and the macro is kept only for matters of dummy variables.

Currently, the parsing of hexadecimal numbers needs two nested `\csname...\endcsname`, first to
gather the letters (possibly with a hexadecimal fractional part), and in a second stage to apply
`\xintHexToDec` to do the actual conversion. This should be faster than updating on the fly the
number (which would be hard for the fraction part...).

```

60 \def\xint_gob_til_! #1!{}% ! with catcode 11
61 \def\XINT_expr_lockscan#1{% not used for decimal numbers in xintexpr 1.2
62 \def\XINT_expr_lockscan##1!{\expandafter#1\csname .##1\endcsname}%
63 }\XINT_expr_lockscan{ }%
64 \def\XINT_expr_lockit#1{%
65 \def\XINT_expr_lockit##1{\expandafter#1\csname .##1\endcsname}%
66 }\XINT_expr_lockit{ }%
67 \def\XINT_expr_unlock_hex_in #1% expanded inside \csname..\endcsname
68 {\expandafter\XINT_expr_inhex\romannu-meral`&&\XINT_expr_unlock#1;}%
69 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
70 {%
71      \if#2>%
72      \xintHexToDec{#1}%
73      \else
74      \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
75      [\the\numexpr-4*\xintLength{#3}]%
76      \fi
77 }%
78 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string}%
79 \def\XINT_expr_unlock_a #1.={}%
80 \def\XINT_expr_unexpectedtoken {\xintError:ignored}%

```

```
81 \let\XINT_expr_done\space
```

11.5 \XINT_expr_wrap, \XINT_iiexpr_wrap

```
82 \def\XINT_expr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
83 \def\XINT_iiexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%
```

11.6 \XINT_protectii, \XINT_expr_usethe

```
84 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
85 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%
```

11.7 \XINT_expr_print, \XINT_iiexpr_print, \XINT_boolexpr_print

See also the `\XINT_flexpr_print` which is special, below.

```
86 \def\XINT_expr_print #1{\xintSPraw::csv {\XINT_expr_unlock #1}}%
87 \def\XINT_iiexpr_print #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
88 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
```

11.8 \xintexpr, \xintiexpr, \xintfloatexpr, \xintiiepr

```
89 \def\xintexpr {\romannumeral0\xinteval }%
90 \def\xintiexpr {\romannumeral0\xintieval }%
91 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
92 \def\xintiiepr {\romannumeral0\xintiieval }%
```

11.9 \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xinttheiiepr

```
93 \def\xinttheexpr
94 {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
95 \def\xinttheiexpr {\romannumeral`&&\xintthe\xintiexpr }%
96 \def\xintthefloatexpr {\romannumeral`&&\xintthe\xintfloatexpr }%
97 \def\xinttheiiepr
98 {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%
```

11.10 \xintthe

```
99 \def\xintthe #1{\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&@#1}%
```

11.11 \thexintexpr, \thexintiexpr, \thexintfloatexpr, \thexintiiepr

New with 1.2h. I have been three years long very strict in terms of prefixing macros, but well.

```
100 \let\thexintexpr \xinttheexpr
101 \let\thexintiexpr \xinttheiexpr
102 \let\thexintfloatexpr\xintthefloatexpr
103 \let\thexintiiepr \xinttheiiepr
```

11.12 \xintthecoords

1.1 Wraps up an even number of comma separated items into pairs of TikZ coordinates; for use in the following way:

```
coordinates {\xintthecoords\xintfloatexpr ... \relax}
```

The crazyness with the `\csname` and `unlock` is due to TikZ somewhat STRANGE control of the TOTAL number of expansions which should not exceed the very low value of 100 !! As we implemented `\XINT_thecoords_b` in an "inline" style for efficiency, we need to hide its expansions.

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

Not to be used as `\xintthecoords\xintthefloatexpr`, only as `\xintthecoords\xintfloatexpr` (or `\xintiexpr` etc...). Perhaps `\xintthecoords` could make an extra check, but one should not accustom users to too loose requirements!

```

104 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
105     \XINT_thecoords_a
106     \expandafter\xint_gobble_iii\romannumeral0#1}%
107 \def\XINT_thecoords_a #1#2 #1=print macro, indispensable for scientific notation
108     {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
109         \romannumeral`&&@#1#2,!,!,^\endcsname }%
110 \def\XINT_thecoords_b #1#2,#3#4,%
111     {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
112 \def\XINT_thecoords_c #1^{}%

```

11.13 `\xintbareeval`, `\xintbarefloateval`, `\xintbareieval`

```

113 \def\xintbareeval
114     {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
115 \def\xintbarefloateval
116     {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
117 \def\xintbareieval
118     {\expandafter\XINT_iexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%

```

11.14 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareieval`

```

119 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
120 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
121 \def\xintthebareieval     {\expandafter\XINT_expr_unlock\romannumeral0\xintbareieval}%

```

11.15 `\xinteval`, `\xintiieval`

```

122 \def\xinteval      {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
123 \def\xintiieval    {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareieval }%

```

11.16 `\xintieval`, `\XINT_iexpr_wrap`

Optional argument since 1.1.

```

124 \def\xintieval #1%
125     {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
126 \def\XINT_iexpr_noopt
127     {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
128 \def\XINT_iexpr_withopt [#1]%
129 {%
130     \expandafter\XINT_iexpr_wrap\expandafter
131     {\the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter}%
132     \romannumeral0\xintbareeval
133 }%
134 \def\XINT_iexpr_wrap #1#2%
135 {%
136     \expandafter\XINT_expr_wrap
137     \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
138 }%

```


11.17 `\xintfloateval`, `\XINT_flexpr_wrap`, `\XINT_flexpr_print`

Optional argument since 1.1

```

139 \def\xintfloateval #1%
140 {%
141     \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
142     \fi #1%
143 }%
144 \def\XINT_flexpr_noopt
145 {%
146     \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
147     \romannumeral0\xintbarefloateval
148 }%
149 \def\XINT_flexpr_withopt_a [#1]%
150 {%
151     \expandafter\XINT_flexpr_withopt_b\expandafter
152     {\the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter}%
153     \romannumeral0\xintbarefloateval
154 }%
155 \def\XINT_flexpr_withopt_b #1#2%
156 {%
157     \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
158     \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
159 }%
160 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
161 \def\XINT_flexpr_print #1%
162 {%
163     \expandafter\xintPFloat::csv
164     \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%
165 }%
166 \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%

```

11.18 `\xintboolexpr`, `\xinttheboolexpr`, `\thexintboolexpr`

```

167 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
168     \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
169 \def\xinttheboolexpr   {\romannumeral`&&\expandafter\expandafter\expandafter
170     \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
171 \let\thexintboolexpr\xinttheboolexpr
172 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%

```

11.19 `\xintifboolexpr`, `\xintifboolfloatexpr`, `\xintifboolliexpr`

Do not work with comma separated expressions.

```

173 \def\xintifboolexpr    #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
174 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
175 \def\xintifboolliexpr  #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%

```

11.20 Hooks for the functioning of `\xintNewExpr` and `\xintdeffunc`

This is new with 1.3. See `\XINT_expr_redefinemacros`.

```
176 \let\XINT:NEhook:two\empty
177 \let\XINT:NEhook:one\empty
178 \let\XINT:NEhook:csv\empty
```

11.21 Macros handling csv lists on output (for \XINT_expr_print et al. routines)

11.21.1	\XINT::_end	298
11.21.2	\xintCSV::csv	298
11.21.3	\xintSPRaw, \xintSPRaw::csv	298
11.21.4	\xintIsTrue::csv	298
11.21.5	\xintRound::csv	299
11.21.6	\XINTinFloat::csv	299
11.21.7	\xintPFloat::csv	299

Changed completely for 1.1, which adds the optional arguments to \xintexpr and \xintfloatexpr.

11.21.1 \XINT::_end

Le mécanisme est le suivant, #2 est dans des accolades et commence par ,<sp>. Donc le gobble se débarrasse du, et le <sp> après brace stripping arrête un \romannumeral0 ou \romannumeral-`0

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

11.21.2 \xintCSV::csv

```
180 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^,%}
181 \def\XINT_csv::_a {\XINT_csv::_b {}}%
182 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
183 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
184 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

11.21.3 \xintSPRaw, \xintSPRaw::csv

```
185 \def\xintSPRaw {\romannumeral0\xintspraw}%
186 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
187 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
188 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
189 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
190 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^,%}
191 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
192 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
193 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
194 \if ^#1\xint_dothis\XINT::_end\fi
195 \xint_orthat\XINT_spraw::_d #1}%
196 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
197 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

11.21.4 \xintIsTrue::csv

```
198 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^,%}
199 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
200 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
201 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
202 \if ^#1\xint_dothis\XINT::_end\fi
203 \xint_orthat\XINT_istrue::_d #1}%

```

```
204 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
205 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

11.21.5 \xintRound::csv

```
206 \def\XINT_::_end #1,#2#3{\xint_gobble_i #3}%
207 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
208   {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
209 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
210 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
211   \if ^#1\xint_dothis\XINT_::_end\fi
212   \xint_orthat\XINT_round::_d #1}%
213 \def\XINT_round::_d #1,#2{%
214   \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
215   \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
216 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%
```

11.21.6 \XINTinFloat::csv

```
217 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
218   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
219 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
220 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
221   \if ^#1\xint_dothis\XINT_::_end\fi
222   \xint_orthat\XINT_infloat::_d #1}%
223 \def\XINT_infloat::_d #1,#2{%
224   {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
225 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%
```

11.21.7 \xintPFloat::csv

```
226 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
227   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
228 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
229 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
230   \if ^#1\xint_dothis\XINT_::_end\fi
231   \xint_orthat\XINT_pfloat::_d #1}%
232 \def\XINT_pfloat::_d #1,#2{%
233   {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint:#2]{#1},{#2}}%
234 \def\XINT_pfloat::_e #1,#2#3{\XINT_pfloat::_b {#2}{#3, #1}}%
```

11.22 \XINT_expr_getnext: fetching some number then an operator

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then \romannumeral-0 expansion.

```
235 \def\XINT_expr_getnext #1%
236 {%
237   \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
238 }%
239 \def\XINT_expr_getnext_a #1%
240 {% screens out sub-expressions and \count or \dimen registers/variables
241   \xint_gob_til! #1\XINT_expr_subexpr !% recall this ! has catcode 11
242   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
243     \expandafter\XINT_expr_countetc
244   \else
```

```

245 \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
246 \fi
247 #1%
248 }%
249 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

1.2 adds \ht, \dp, \wd and the eTeX font things.

```

250 \def\XINT_expr_countetc #1%
251 {%
252 \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
253 \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
254 \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
255 \ifx\fontcharhp#1\else\ifx\fontcharic#1\else
256 \XINT_expr_unpackvar
257 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
258 \expandafter\XINT_expr_getnext\number #1%
259 }%
260 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
261 \expandafter\XINT_expr_getnext\number #1%
262 {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
263 \expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%
264 \begingroup
265 \lccode`*=`#
266 \lowercase{\endgroup
267 \def\XINT_expr_getnextfork #1{%
268 \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
269 \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
270 \if#1+\xint_dothis \XINT_expr_getnext \fi
271 \if#1.\xint_dothis {\XINT_expr_startdec}\fi
272 \if#1-\xint_dothis -\fi
273 \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
274 \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
275 }%
276 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

11.23 \XINT_expr_scan_nbr_or_func: the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser

11.23.1	Integral part (skipping zeroes)	301
11.23.2	Fractional part	303
11.23.3	Scientific notation	304
11.23.4	Hexadecimal numbers	305
11.23.5	\XINT_expr_scanfunc: parsing names of functions and variables	306

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with \xintiexpr which should never be given a [n] inside its \csname.<=<digits>\endcsname storage of numbers (because its arithmetic uses the ii macros which

know nothing about the [N] notation). Hence if the parser has only seen digits when hitting something else than the dot or e (or E), it will not insert a [0]. Thus we very slightly compromise the efficiency of `\xintexpr` and `\xintfloatexpr` in order to be able to share the same code with `\xintiexpr`.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside `\xintexpr` or `\xintiexpr`. On the other hand if a dot or a e (or E) is met, then the (common) parser has no scruples ending this number with a [n], this will provoke an error later if that was within an `\xintiexpr`, as soon as an arithmetic macro is used.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the `\xintHexToDec` macro.

With 1.2, `\xinttheexpr . \relax` does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both `\xinttheexpr 1.\relax` and `\xinttheexpr .1\relax` are legal.

The ``` syntax is here used for special constructs like ``+`(..)`, ``*`(..)` where + or * will be treated as functions. Current implementation pick only one token (could have been braced stuff), thus here it will be + or *, and via `\XINT_expr_op`` this into becomes a suitable `\XINT_{expr|iexpr|fexpr}_func_+` (or *). Documentation of 1.1 said to use ``+`(...)`, but ``+`(...)` is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```

277 \catcode96 11 % `
278 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
279 {%
280   \if #1\xint_dothis \XINT_expr_gotnil \fi
281   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
282   \if `#1\xint_dothis {\XINT_expr_onliteral`}\fi
283   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
284   \xint_orthat \XINT_expr_scanfunc #1%
285 }%
286 \def\XINT_expr_gotnil{\expandafter\XINT_expr_getop\csname.= \endcsname}%
287 \def\XINT_expr_onliteral` #1#2#3({\xint_c_xviii `{#2}}%
288 \catcode96 12 % `
289 \def\XINT_expr_startint #1%
290 {%
291   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
292 }%
293 \def\XINT_expr_scanint_a #1#2%
294   {\expandafter\XINT_expr_getop\csname.=#1%
295   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
296 \def\XINT_expr_gobz_a #1%
297   {\expandafter\XINT_expr_getop\csname.=
298   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
299 \def\XINT_expr_startdec #1%
300   {\expandafter\XINT_expr_getop\csname.=
301   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

11.23.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligeable. I don't think the doubled `\string` is a serious penalty.

```

302 \def\XINT_expr_scanint_b #1%
303 {%

```

```

304 \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
305 \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
306 \string#1\XINT_expr_scanint_d
307 }%
308 \def\XINT_expr_scanint_d #1%
309 {%
310 \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
311 }%
312 \def\XINT_expr_scanint_endbycs#1#2\XINT_expr_scanint_d{\endcsname #1}%

```

With 1.2d the tacit multiplication in front of a variable name or function name is now done with a higher precedence, intermediate between the common one of $*$ and $/$ and the one of $^$. Thus $x/2y$ is like $x/(2y)$, but x^2y is like x^2*y and $2y!$ is not $(2y)!$ but $2*y!$.

Finally, 1.2d has moved away from the `_scan` macros all the business of the tacit multiplication in one unique place via `\XINT_expr_getop`. For this, the ending token is not first given to `\string` as was done earlier before handing over back control to `\XINT_expr_getop`. Earlier we had to identify the catcode 11 ! signaling a sub-expression here. With no `\string` applied we can do it in `\XINT_expr_getop`. As a corollary of this displacement, parsing of big numbers should be a tiny bit faster now.

Extended for 1.2l to ignore underscore character `_` if encountered within digits; so it can serve as separator for better readability.

```

313 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
314 {%
315 \if _#1\xint_dothis\XINT_expr_scanint_d\fi
316 \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
317 \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
318 \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
319 \xint_orthat {\endcsname #1}%
320 }%
321 \def\XINT_expr_startdec_a .#1%
322 {%
323 \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
324 }%
325 \def\XINT_expr_scandec_a #1%
326 {%
327 \if .#1\xint_dothis{\endcsname .}\fi
328 \xint_orthat {\XINT_expr_scandec_b 0.#1}%
329 }%
330 \def\XINT_expr_gobz_scanint_b #1%
331 {%
332 \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
333 \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
334 \string#1\XINT_expr_scanint_d
335 }%
336 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
337 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
338 {%
339 \if _#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
340 \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
341 \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
342 \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
343 \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi

```

```

344 \xint_orthat {0\endcsname #1}%
345 }%
346 \def\xINT_expr_gobz_scanint_d #1%
347 {%
348 \expandafter\xINT_expr_gobz_scanint_b\romannumeral`&&@#1%
349 }%
350 \def\xINT_expr_gobz_startdec_a .#1%
351 {%
352 \expandafter\xINT_expr_gobz_scandec_a\romannumeral`&&@#1%
353 }%
354 \def\xINT_expr_gobz_scandec_a #1%
355 {%
356 \if .#1\xint_dothis{0\endcsname.}\fi
357 \xint_orthat {\xINT_expr_gobz_scandec_b 0.#1}%
358 }%

```

11.23.2 Fractional part

Annoying duplication of code to allow 0. as input.

1.2a corrects a very bad bug in 1.2 \XINT_expr_gobz_scandec_b which should have stripped leading zeroes in the fractional part but didn't; as a result \xinttheexpr 0.01\relax returned 0 =:-(((Thanks to Kroum Tzanev who reported the issue. Does it improve things if I say the bug was introduced in 1.2, it wasn't present before ?

```

359 \def\xINT_expr_scandec_b #1.#2%
360 {%
361 \ifcat \relax #2\expandafter\xINT_expr_scandec_endbycs\expandafter#2\fi
362 \ifnum\xint_c_ix<1\string#2 \else\expandafter\xINT_expr_scandec_c\fi
363 \string#2\expandafter\xINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
364 }%
365 \def\xINT_expr_scandec_endbycs #1#2\xINT_expr_scandec_d
366 \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
367 \def\xINT_expr_scandec_d #1.#2%
368 {%
369 \expandafter\xINT_expr_scandec_b
370 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
371 }%
372 \def\xINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
373 {%
374 \if _#1\xint_dothis{\xINT_expr_scandec_d#3.}\fi
375 \if e#1\xint_dothis{[\the\numexpr#3\xINT_expr_scanexp_a +]\fi
376 \if E#1\xint_dothis{[\the\numexpr#3\xINT_expr_scanexp_a +]\fi
377 \xint_orthat {[#3]\endcsname #1}%
378 }%
379 \def\xINT_expr_gobz_scandec_b #1.#2%
380 {%
381 \ifcat \relax #2\expandafter\xINT_expr_gobz_scandec_endbycs\expandafter#2\fi
382 \ifnum\xint_c_ix<1\string#2 \else\expandafter\xINT_expr_gobz_scandec_c\fi
383 \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
384 {\expandafter\xINT_expr_gobz_scandec_b}%
385 {\string#2\expandafter\xINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
386 }%

```

```

387 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
388 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
389 {%
390   \if _#1\xint_dothis{\XINT_expr_gobz_scandec_b #4.}\fi
391   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
392   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
393   \xint_orthat {0[0]\endcsname #1}%
394 }%

```

11.23.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

```

395 \def\XINT_expr_scanexp_a #1#2%
396 {%
397   #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
398 }%
399 \def\XINT_expr_scanexp_b #1%
400 {%
401   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
402   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
403   \string#1\XINT_expr_scanexp_d
404 }%
405 \def\XINT_expr_scanexp_endbycs#1#2\XINT_expr_scanexp_d []\endcsname #1}%
406 \def\XINT_expr_scanexp_d #1%
407 {%
408   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
409 }%
410 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
411 {%
412   \if _#1\xint_dothis \XINT_expr_scanexp_d \fi
413   \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
414   \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
415   \xint_orthat []\endcsname #1}%
416 }%
417 \def\XINT_expr_scanexp_bb #1%
418 {%
419   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
420   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
421   \string#1\XINT_expr_scanexp_db
422 }%
423 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db []\endcsname #1}%
424 \def\XINT_expr_scanexp_db #1%
425 {%
426   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
427 }%
428 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db
429 {%
430   \if _#1\xint_dothis\XINT_expr_scanexp_d\fi
431   \xint_orthat[]\endcsname #1}%
432 }%

```


11.23.4 Hexadecimal numbers

1.2d has moved most of the handling of tacit multiplication to `\XINT_expr_getop`, but we have to do some of it here, because we apply `\string` before calling `\XINT_expr_scanhexI_aa`. I do not insert the `*` in `\XINT_expr_scanhexI_a`, because it is its higher precedence variant which will be expected, to do the same as when a non-hexadecimal number prefixes a sub-expression. Tacit multiplication in front of variable or function names will not work (because of this `\string`).

Extended for 1.2l to ignore underscore character `_` if encountered within digits.

```

433 \def\XINT_expr_scanhex_I #1% #1="
434 {%
435     \expandafter\XINT_expr_getop\csname.=\expandafter
436     \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
437 }%
438 \def\XINT_expr_scanhexI_a #1%
439 {%
440     \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
441     \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
442     \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
443 }%
444 \def\XINT_expr_scanhexI_aa #1%
445 {%
446     \if\ifnum`#1>`/
447         \ifnum`#1>`9
448         \ifnum`#1>`@
449         \ifnum`#1>`F
450         0\else1\fi\else0\fi\else1\fi\else0\fi 1%
451         \expandafter\XINT_expr_scanhexI_b
452     \else
453         \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
454         \if.#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
455         \xint_orthat % gather what we got so far, leave catcode 12 #1 in stream
456         {\xint_afterfi {.>\endcsname\endcsname}}%
457     \fi
458     #1%
459 }%
460 \def\XINT_expr_scanhexI_b #1#2%
461 {%
462     #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
463 }%
464 \def\XINT_expr_scanhexI_bgob #1#2%
465 {%
466     \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
467 }%
468 \def\XINT_expr_scanhex_transition .#1%
469 {%
470     \expandafter.\expandafter.\expandafter
471     \XINT_expr_scanhexII_a\romannumeral`&&@#1%
472 }%
473 \def\XINT_expr_scanhexII_a #1%
474 {%
475     \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
476     \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi

```

```

477 \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
478 }%
479 \def\XINT_expr_scanhexII_aa #1%
480 {%
481 \if\ifnum`#1>`/
482 \ifnum`#1>`9
483 \ifnum`#1>`@
484 \ifnum`#1>`F
485 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
486 \expandafter\XINT_expr_scanhexII_b
487 \else
488 \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
489 \xint_orthat{\xint_afterfi {\endcsname\endcsname}}%
490 \fi
491 #1%
492 }%
493 \def\XINT_expr_scanhexII_b #1#2%
494 {%
495 #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
496 }%
497 \def\XINT_expr_scanhexII_bgob #1#2%
498 {%
499 \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
500 }%

```

11.23.5 \XINT_expr_scanfunc: parsing names of functions and variables

```

501 \def\XINT_expr_scanfunc
502 {%
503 \expandafter\XINT_expr_func\romannumeral`&&\XINT_expr_scanfunc_a
504 }%
505 \def\XINT_expr_scanfunc_a #1#2%
506 {%
507 \expandafter #1\romannumeral`&&\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
508 }%

```

This handles: 1) (indirectly) tacit multiplication by a variable in front of a sub-expression, 2) (indirectly) tacit multiplication in front of a \count etc..., 3) functions which are recognized via an encountered opening parenthesis (but later this must be disambiguated from variables with tacit multiplication) 4) 5) 6) 7) acceptable components of a variable or function names: @, underscore, digits, letters (or chars of category code letter.)

The short lived 1.2d which followed the even shorter lived 1.2c managed to introduce a bug here as it removed the check for catcode 11 !, which must be recognized if ! is not to be taken as part of a variable name. Don't know what I was thinking, it was the time when I was moving the handling of tacit multiplication entirely to the \XINT_expr_getop side. Fixed in 1.2e.

I almost decided to remove the \ifcat\relax test whose rôle is to avoid the \string#1 to do something bad is the escape char is a digit! Perhaps I will remove it at some point ! I truly almost did it, but also the case of no escape char is a problem (\string\0, if \0 is a count ...)

The (indirectly) above means that via \XINT_expr_func then \XINT_expr_op__ one goes back to \XINT_expr_getop then \XINT_expr_getop_b which is the location where tacit multiplication is now centralized. This makes the treatment of tacit multiplication for situations such as <variable>\count or <variable>\xintexpr..\relax, perhaps a bit sub-optimal, but first the variable name must be gathered, second the variable must expand to its value.

```

509 \def\XINT_expr_scanfunc_b #1%
510 {%
511   \ifx !#1\xint_dothis{(_)\fi
512   \ifcat \relax#1\xint_dothis{(_)\fi
513   \if (#1\xint_dothis{\xint_firstoftwo{(`)\fi
514   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
515   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
516   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
517   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
518   \xint_orthat {(_)%
519     #1%
520 }%

```

Comments written 2015/11/12: earlier there was an `\ifcsname` test for checking if we had a variable in front of a `(`, for tacit multiplication for example in `x(y+z(x+w))` to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in `expr`, `iiexpr` or `floatexpr`. The `\xint_c_xviii` causes all fetching operations to stop and control is handed over to the routines which will be `expr`, `iiexpr` ou `floatexpr` specific, i.e. the `\XINT_{expr|iiexpr|flexpr}_op_{`|_}` which are invoked by the `until_<op>_b` macros earlier in the stream. Functions may exist for one but not the two other parsers. Variables are declared via one parser and usable in the others, but naturally `\xintiexpr` has its restrictions.

Thinking about this again I decided to treat a priori cases such as `x(...)` as functions, after having assigned to each variable a low-weight macro which will convert this into `_getop\.=<value of x>*(...)`. To activate that macro at the right time I could for this exploit the "onliteral" intercept, which is parser independent (1.2c).

This led to me necessarily to rewrite partially the `seq`, `add`, `mul`, `subs`, `iter` ... routines as now the variables fetch only one token. I think the thing is more efficient.

1.2c had `\def\XINT_expr_func #1(#2{\xint_c_xviii #2{#1}}`

In `\XINT_expr_func` the `#2` is `_` if `#1` must be a variable name, or `#2=` if `#1` must be either a function name or possibly a variable name which will then have to be followed by tacit multiplication before the opening parenthesis.

The `\xint_c_xviii` is there because `_op_` must know in which parser it works. Dispendious for `_`. Hence I modify for 1.2d.

```

521 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
522   \xint_orthat{\xint_c_xviii #2}{#1}}%

```

11.24 `\XINT_expr_getop`: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit multiplication also in front of variable or functions names starting with a letter, not only a `@` or a `_` as was already the case. This is for `(x+y)z` situations. It also applies higher precedence in cases like `x/2y` or `x/2@`, or `x/2max(3,5)`, or `x/2\xintexpr 3\relax`.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if `\XINT_expr_getop` as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like `(a+b)/(c+d)(e+f)` will first multiply the last two parenthesized terms.

The `!` starting a sub-expression must be distinguished from the post-fix `!` for factorial, thus we must not do a too early `\string`. In versions `< 1.2c`, the catcode 11 `!` had to be identified in all

branches of the number or function scans. Here it is simply treated as a special case of a letter.

1.2q adds tacit multiplication in cases such as (1+1)3 or 5!7!

```

523 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
524 {%
525     \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
526 }%

527 \catcode`* 11
528 \def\XINT_expr_getop_a #1#2%
529 {%
530     \ifx \relax #2\xint_dothis\xint_firstofthree\fi
531     \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
532     \ifnum\xint_c_ix<1\string#2 \xint_dothis\xint_secondofthree\fi
533     \if _#2\xint_dothis \xint_secondofthree\fi
534     \if @#2\xint_dothis \xint_secondofthree\fi
535     \if (#2\xint_dothis \xint_secondofthree\fi
536     \ifcat a#2\xint_dothis \xint_secondofthree\fi
537     \xint_orthat \xint_thirdofthree
538     {\XINT_expr_foundend #1}%
539     {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
540     {\expandafter\XINT_expr_getop_b \string#2#1}%
541 }%
542 \catcode`* 12
543 \def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.

```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```

544 \def\XINT_expr_getop_b #1%
545 {%
546     \if '#1\xint_dothis{\XINT_expr_binopwrdrd }\fi
547     \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
548     \xint_orthat {\XINT_expr_scanop_a #1}%
549 }%
550 \def\XINT_expr_binopwrdrd #1#2'\{\expandafter\XINT_expr_foundop_a
551     \csname XINT_expr_itself_\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
552 \def\XINT_expr_scanop_a #1#2#3%
553     {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
554 \def\XINT_expr_scanop_b #1#2#3%
555 {%
556     \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
557     \ifcsname XINT_expr_itself_#1#3\endcsname
558     \xint_dothis
559         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
560     \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
561 }%
562 \def\XINT_expr_scanop_c #1#2#3%
563 {%
564     \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
565 }%
566 \def\XINT_expr_scanop_d #1#2#3%
567 {%

```

```

568 \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
569 \ifcsname XINT_expr_itself_#1#3\endcsname
570 \xint_dothis
571     {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
572 \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
573 }%
574 \def\XINT_expr_foundop_a #1%
575 {%
576     \ifcsname XINT_expr_precedence_#1\endcsname
577         \csname XINT_expr_precedence_#1\expandafter\endcsname
578         \expandafter #1%
579     \else
580         \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
581     \fi
582 }%
583 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
584 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

11.25 Expansion spanning; opening and closing parentheses

Version 1.1 had a hack inside the until macros for handling the omit and abort in iterations over dummy variables. This has been removed by 1.2c, see the subsection where omit and abort are discussed.

```

585 \catcode`) 11
586 \def\XINT_tmpa #1#2#3#4%
587 {%
588     \def#1##1%
589     {%
590         \xint_UDsignfork
591             ##1{\expandafter#1\romannumeral`&&@#3}%
592             -{#2##1}%
593         \krof
594     }%
595     \def#2##1##2%
596     {%
597         \ifcase ##1\expandafter\XINT_expr_done
598         \or\xint_afterfi{\XINT_expr_extra_)
599             \expandafter #1\romannumeral`&&\XINT_expr_getop }%
600         \else
601             \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
602         \fi
603     }%
604 }%
605 \def\XINT_expr_extra_) {\xintError:removed }%
606 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
607     \expandafter\XINT_tmpa
608     \csname XINT_#1_until_end_a\expandafter\endcsname
609     \csname XINT_#1_until_end_b\expandafter\endcsname
610     \csname XINT_#1_op_-vi\endcsname
611     {#1}%
612 }%
613 \def\XINT_tmpa #1#2#3#4#5#6%

```

```

614 {%
615   \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
616   \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
617   \def #3##1{\xint_UDsignfork
618     ##1{\expandafter #3\romannumeral`&&#5}%
619     -{#4##1}%
620     \krof }%
621   \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_
622     \or \csname XINT_#6_op_#2\expandafter\endcsname
623     \else
624     \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_#2\endcsname }%
625     \fi
626   }%
627 }%
628 \def\XINT_expr_missing_ {\xintError:inserted \xint_c_ \XINT_expr_done }%

```

We should be using `until_` notation to stay synchronous with `until_+`, `until_*` etc..., but I found that `until_` was more telling.

```

629 \catcode` ) 12
630 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
631   \expandafter\XINT_tmpa
632   \csname XINT_#1_op_(\expandafter\endcsname
633   \csname XINT_#1_oparen\expandafter\endcsname
634   \csname XINT_#1_until_)_a\expandafter\endcsname
635   \csname XINT_#1_until_)_b\expandafter\endcsname
636   \csname XINT_#1_op_-vi\endcsname
637   {#1}%
638 }%
639 \expandafter\let\csname XINT_expr_precedence_\endcsname\xint_c_i

```

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

- 11.26.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct 310
- 11.26.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iiexpr operators 312
- 11.26.3 The]+,]-,]*,]/,]^, +[, -[, *[, /[, and ^[list operators 314
- 11.26.4 The 'and', 'or', 'xor', and 'mod' as infix operator words 316
- 11.26.5 The ||, &&, **, **[,]** operators as synonyms 316

11.26.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct

This is all very clever and only need setting some suitable precedence levels, if only I could understand what I did in 2014... just joking. Notice that `op_` macros are defined here in the `\xintFor` loop.

There is some clever business going on here with the letter `a` for handling constructs such as `[3..5]*2` (I think...).

1.2c has replaced 1.1's private dealings with "`^C`" (which was done before dummy variables got implemented) by use of "`!?`". See discussion of omit and abort.

```

640 \expandafter\let\csname XINT_expr_precedence_\endcsname\xint_c_i
641 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i

```

```
642 \let\XINT_expr_precedence_a \xint_c_xviii
643 \let\XINT_expr_precedence_!? \xint_c_ii
644 \expandafter\let\csname XINT_expr_precedence_++\endcsname \xint_c_i
```

Comments added 2015/11/13 Here we have in particular the mechanism for post action on lists via op_] The precedence_] is the one of a closing parenthesis. We need the closing parenthesis to do its job, hence we can not define a op_]+ operator for example, as we want to assign it the precedence of addition not the one of closing parenthesis. The trick I used in 1.1 was to let the op_] insert the letter a, this letter exceptionnally also being a legitimate operator, launch the _getop and let it find a a*, a+, a/, a-, a^, a** operator standing for],*,],+,],/,],^,]** postfix item by item list operator. I thought I had in mind an example to show that having defined op_a and precedence_a for the letter a caused a reduction in syntax for this letter, but it seems I am lacking now an example.

2015/11/18: for 1.2d I accelerate \XINT_expr_op_] to jump over the \XINT_expr_getop_a which now does tacit multiplications also in front of letters, for reasons of things like, (x+y)z, hence it must not see the "a". I could have used a catcode12 a possibly, but anyhow jumping straight to \XINT_expr_scanop_a skips a few expansion steps (up to the potential price of less conceptual programming if I change things in the future.)

```
645 \catcode`. 11 \catcode`= 11 \catcode`+ 11
646 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
647   \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
648   \expandafter\let\csname XINT_#1_op_;\endcsname \space
649   \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
650   \expandafter\let\csname XINT_#1_op_a)\endcsname \XINT_expr_getop
```

1.1 2014/10/29 did \expandafter\.=+\xintiCeil which transformed it into \romannumeral0\xinticeil, which seems a bit weird. This exploited the fact that dummy variables macros could back then pick braced material (which in the case at hand here ended being {\romannumeral0\xinticeil...} and were submitted to two expansions. The result of this was to provide a not value which got expanded only in the first loop of the :_A and following macros of seq, iter, rseq, etc...

Anyhow with 1.2c I have changed the implementation of dummy variables which now need to fetch a single locked token, which they do not expand.

The \xintiCeil appears a bit dispendious, but I need the starting value in a \numexpr compatible form in the iteration loops.

```
651   \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
652   {\expandafter\XINT_expr_foundend \expandafter
653     {\expandafter\.=+\csname .=\XINT:NEhook:one\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
654 }%
655 \catcode`. 12 \catcode`= 12 \catcode`+ 12
```

1.2d adds the *** for tying via tacit multiplication, for example x/2y. Actually I don't need the _itself mechanism for ***, only a precedence.

```
656 \catcode`& 12
657 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{.[]{}..}%
658   {+[]{-[]*[]{/[]{**[]{^[]{a+}{a-}{a*}{a/}{a**}{a^}%
659   [][][:]{:}[]{![]{++}{++}}}%{***}}
660   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
661 \catcode`& 7
662 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii
```

11.26.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iexpr operators

1.2d needed some room between /, * and ^. Hence precedence for ^ is now at 9

```

663 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
664 {%
665   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iexpr
666   {% keep value, get next number and operator, then do until
667     \expandafter #2\expandafter ##1%
668     \romannumeral`&&\expandafter\XINT_expr_getnext }%
669   \def #2#1#2% \XINT_expr_until_<op>_a ou flexpr ou iexpr
670   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
671     -{#3##1#2}%
672     \krof }%
673   \def #3#1#2#3#4% \XINT_expr_until_<op>_b ou flexpr ou iexpr
674   {% either execute next operation now, or first do next (possibly unary)
675     \ifnum ##2>#7%
676     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
677       \csname XINT_#8_op_##3\endcsname {##4}}%
678     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
679       \csname .=#9#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
680     \fi }%
681   \let #7#5%
682 }%
683 \def\XINT_expr_defbin_b #1#2#3#4#5%
684 {%
685   \expandafter\XINT_expr_defbin_c
686   \csname XINT_#1_op_#2\expandafter\endcsname
687   \csname XINT_#1_until_#2_a\expandafter\endcsname
688   \csname XINT_#1_until_#2_b\expandafter\endcsname
689   \csname XINT_#1_op_#4\expandafter\endcsname
690   \csname xint_c_#3\expandafter\endcsname
691   \csname #5\expandafter\endcsname
692   \csname XINT_expr_precedence_#2\endcsname {#1}\XINT:NEhook:two
693 }%
694 \XINT_expr_defbin_b {expr} | {iii}{vi} {xintOR}%
695 \XINT_expr_defbin_b {flexpr} | {iii}{vi} {xintOR}%
696 \XINT_expr_defbin_b {iexpr} | {iii}{vi} {xintOR}%
697 \XINT_expr_defbin_b {expr} & {iv}{vi} {xintAND}%
698 \XINT_expr_defbin_b {flexpr} & {iv}{vi} {xintAND}%
699 \XINT_expr_defbin_b {iexpr} & {iv}{vi} {xintAND}%
700 \XINT_expr_defbin_b {expr} {xor}{iii}{vi} {xintXOR}%
701 \XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
702 \XINT_expr_defbin_b {iexpr}{xor}{iii}{vi} {xintXOR}%
703 \XINT_expr_defbin_b {expr} < {v}{vi} {xintLt}%
704 \XINT_expr_defbin_b {flexpr} < {v}{vi} {xintLt}%
705 \XINT_expr_defbin_b {iexpr} < {v}{vi} {xintiiLt}%
706 \XINT_expr_defbin_b {expr} > {v}{vi} {xintGt}%
707 \XINT_expr_defbin_b {flexpr} > {v}{vi} {xintGt}%
708 \XINT_expr_defbin_b {iexpr} > {v}{vi} {xintiiGt}%
709 \XINT_expr_defbin_b {expr} = {v}{vi} {xintEq}%
710 \XINT_expr_defbin_b {flexpr} = {v}{vi} {xintEq}%

```



```

711 \XINT_expr_defbin_b {iiexpr} = {v}{vi} {xintiiEq}%
712 \XINT_expr_defbin_b {expr} {<=} {v}{vi} {xintLtorEq}%
713 \XINT_expr_defbin_b {flexpr} {<=} {v}{vi} {xintLtorEq}%
714 \XINT_expr_defbin_b {iiexpr} {<=} {v}{vi} {xintiiLtorEq}%
715 \XINT_expr_defbin_b {expr} {>=} {v}{vi} {xintGtorEq}%
716 \XINT_expr_defbin_b {flexpr} {>=} {v}{vi} {xintGtorEq}%
717 \XINT_expr_defbin_b {iiexpr} {>=} {v}{vi} {xintiiGtorEq}%
718 \XINT_expr_defbin_b {expr} {!=} {v}{vi} {xintNotEq}%
719 \XINT_expr_defbin_b {flexpr} {!=} {v}{vi} {xintNotEq}%
720 \XINT_expr_defbin_b {iiexpr} {!=} {v}{vi} {xintiiNotEq}%
721 \XINT_expr_defbin_b {expr} {//} {vii}{vii}{xintDivFloor}% CHANGED IN 1.2p!
722 \XINT_expr_defbin_b {flexpr} {//} {vii}{vii}{XINTinFloatDivFloor}% "
723 \XINT_expr_defbin_b {iiexpr} {//} {vii}{vii}{xintiiDivFloor}% "
724 \XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}% "
725 \XINT_expr_defbin_b {flexpr} {/:} {vii}{vii}{XINTinFloatMod}% "
726 \XINT_expr_defbin_b {iiexpr} {/:} {vii}{vii}{xintiiMod}% "
727 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
728 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
729 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
730 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%
731 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
732 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
733 \XINT_expr_defbin_b {expr} * {vii}{vii}{xintMul}%
734 \XINT_expr_defbin_b {flexpr} * {vii}{vii}{XINTinFloatMul}%
735 \XINT_expr_defbin_b {iiexpr} * {vii}{vii}{xintiiMul}%
736 \XINT_expr_defbin_b {expr} / {vii}{vii}{xintDiv}%
737 \XINT_expr_defbin_b {flexpr} / {vii}{vii}{XINTinFloatDiv}%
738 \XINT_expr_defbin_b {iiexpr} / {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
739 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
740 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
741 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
742 \XINT_expr_defbin_b {expr} {...} {iii}{vi} {xintSeqA::csv}%
743 \XINT_expr_defbin_b {flexpr} {...} {iii}{vi} {XINTinFloatSeqA::csv}%
744 \XINT_expr_defbin_b {iiexpr} {...} {iii}{vi} {xintiiSeqA::csv}%
745 \def\XINT_expr_defbin_b #1#2#3#4#5%
746 {%
747 \expandafter\XINT_expr_defbin_c
748 \csname XINT_#1_op_#2\expandafter\endcsname
749 \csname XINT_#1_until_#2_a\expandafter\endcsname
750 \csname XINT_#1_until_#2_b\expandafter\endcsname
751 \csname XINT_#1_op_#4\expandafter\endcsname
752 \csname xint_c_#3\expandafter\endcsname
753 \csname #5\expandafter\endcsname
754 \csname XINT_expr_precedence_#2\endcsname {#1}{}%
755 }%
756 \XINT_expr_defbin_b {expr} {...} {iii}{vi} {xintSeq::csv}%
757 \XINT_expr_defbin_b {flexpr} {...} {iii}{vi} {xintSeq::csv}%
758 \XINT_expr_defbin_b {iiexpr} {...} {iii}{vi} {xintiiSeq::csv}%
759 \XINT_expr_defbin_b {expr} {][...} {iii}{vi} {xintSeqB::csv}%
760 \XINT_expr_defbin_b {flexpr} {][...} {iii}{vi} {XINTinFloatSeqB::csv}%
761 \XINT_expr_defbin_b {iiexpr} {][...} {iii}{vi} {xintiiSeqB::csv}%

```

11.26.3 The], -,], *,], ^, +[, -[, *, /[, and ^[list operators

\XINT_expr_binop_inline_b This handles acting on comma separated values (no need to bother about spaces in this context; expansion in a \csname...\endcsname.

```

762 \def\XINT_expr_binop_inline#1%
763   {\XINT_expr_binop_inline_a{\expandafter\XINT:NEhook:two\expandafter#1}}%
764 \def\XINT_expr_binop_inline_a
765   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b}%
766 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
767 \def\XINT_expr_binop_inline_c #1{%
768   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
769   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
770   \xint_orthat\XINT_expr_binop_inline_d #1}%
771 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
772 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
773 \def\XINT_expr_binop_inline_end #1,#2{%
774 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
775 {%
776   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
777   {% keep value, get next number and operator, then do until
778     \expandafter #2\expandafter ##1%
779     \romannumeral`&&\expandafter\XINT_expr_getnext}%
780   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
781   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
782    -{#3##1##2}%
783    \krof}%
784   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
785   {% either execute next operation now, or first do next (possibly unary)
786     \ifnum ##2>#7%
787       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
788         \csname XINT_#8_op_##3\endcsname {##4}}%
789       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
790         \csname .=\expandafter\XINT_expr_binop_inline\expandafter
791           {\expandafter#6\expandafter\xint_exchangetwo_keepbraces\expandafter
792             {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
793           \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname}%
794       \fi}%
795   \let #7#5%
796  }%
797 \def\XINT_expr_deflistopr_b #1#2#3#4%
798 {%
799   \expandafter\XINT_expr_deflistopr_c
800   \csname XINT_#1_op_#2\expandafter\endcsname
801   \csname XINT_#1_until_#2_a\expandafter\endcsname
802   \csname XINT_#1_until_#2_b\expandafter\endcsname
803   \csname XINT_#1_op_-#3\expandafter\endcsname
804   \csname xint_c_#3\expandafter\endcsname
805   \csname #4\expandafter\endcsname
806   \csname XINT_expr_precedence_#2\endcsname {#1}%
807  }%

```

This is for [x..y]*z syntax etc.... Attention that with 1.2d, precedence level of ^ raised to ix to make room for ***.

```

808 \XINT_expr_deflistopr_b {expr} {a+}{vi} {xintAdd}%
809 \XINT_expr_deflistopr_b {expr} {a-}{vi} {xintSub}%
810 \XINT_expr_deflistopr_b {expr} {a*}{vii} {xintMul}%
811 \XINT_expr_deflistopr_b {expr} {a/}{vii} {xintDiv}%
812 \XINT_expr_deflistopr_b {expr} {a^}{ix} {xintPow}%
813 \XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
814 \XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
815 \XINT_expr_deflistopr_b {iiexpr}{a*}{vii} {xintiiMul}%
816 \XINT_expr_deflistopr_b {iiexpr}{a/}{vii} {xintiiDivRound}%
817 \XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
818 \XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
819 \XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
820 \XINT_expr_deflistopr_b {flexpr}{a*}{vii} {XINTinFloatMul}%
821 \XINT_expr_deflistopr_b {flexpr}{a/}{vii} {XINTinFloatDiv}%
822 \XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%
823 \def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
824 {%
825   \def #1#1{\expandafter#2\expandafter##1\romannumeral`&&%
826     \expandafter #3\romannumeral`&&\XINT_expr_getnext }%
827   \def #2##1##2##3##4%
828   {% either execute next operation now, or first do next (possibly unary)
829     \ifnum ##2>#6%
830       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
831         \csname XINT_#7_op_##3\endcsname {##4}}%
832       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
833         \csname .=\expandafter\XINT_expr_binop_inline\expandafter
834           {\expandafter#5\expandafter
835             {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
836           \romannumeral`&&\XINT_expr_unlock ##4,^,\endcsname }%
837       \fi }%
838   \let #6#4%
839 }%
840 \def\XINT_expr_deflistopl_b #1#2#3#4%
841 {%
842   \expandafter\XINT_expr_deflistopl_c
843   \csname XINT_#1_op_#2\expandafter\endcsname
844   \csname XINT_#1_until_#2\expandafter\endcsname
845   \csname XINT_#1_until_)_a\expandafter\endcsname
846   \csname xint_c_#3\expandafter\endcsname
847   \csname #4\expandafter\endcsname
848   \csname XINT_expr_precedence_#2\endcsname {#1}%
849 }%

```

This is for $z*[x..y]$ syntax etc...

```

850 \XINT_expr_deflistopl_b {expr} {+[]}{vi} {xintAdd}%
851 \XINT_expr_deflistopl_b {expr} {-[]}{vi} {xintSub}%
852 \XINT_expr_deflistopl_b {expr} {*[]}{vii} {xintMul}%
853 \XINT_expr_deflistopl_b {expr} {/[]}{vii} {xintDiv}%
854 \XINT_expr_deflistopl_b {expr} {^[]}{ix} {xintPow}%
855 \XINT_expr_deflistopl_b {iiexpr}{+[]}{vi} {xintiiAdd}%
856 \XINT_expr_deflistopl_b {iiexpr}{-[]}{vi} {xintiiSub}%
857 \XINT_expr_deflistopl_b {iiexpr}{*[]}{vii} {xintiiMul}%

```

```

858 \XINT_expr_deflistopl_b {iiexpr}{/[]{}{vii}{xintiiDivRound}%
859 \XINT_expr_deflistopl_b {iiexpr}{^[]{}{ix} {xintiiPow}%
860 \XINT_expr_deflistopl_b {fexpr}{+[]{}{vi} {XINTinFloatAdd}%
861 \XINT_expr_deflistopl_b {fexpr}{-[]{}{vi} {XINTinFloatSub}%
862 \XINT_expr_deflistopl_b {fexpr}{*[]{}{vii}{XINTinFloatMul}%
863 \XINT_expr_deflistopl_b {fexpr}{/[]{}{vii}{XINTinFloatDiv}%
864 \XINT_expr_deflistopl_b {fexpr}{^[]{}{ix} {XINTinFloatPowerH}%

```

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

```

865 \xintFor #1 in {and,or,xor,mod} \do {%
866   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
867 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
868   \csname XINT_expr_precedence_&\endcsname
869 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
870   \csname XINT_expr_precedence_| \endcsname
871 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
872   \csname XINT_expr_precedence_/: \endcsname
873 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
874   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
875     \csname XINT_#1_op_&\endcsname
876   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
877     \csname XINT_#1_op_| \endcsname
878   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
879     \csname XINT_#1_op_/: \endcsname
880 }%

```

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

```

881 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
882   \csname XINT_expr_precedence_=\endcsname
883 \expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
884   \csname XINT_expr_precedence_&\endcsname
885 \expandafter\let\csname XINT_expr_precedence_| |\expandafter\endcsname
886   \csname XINT_expr_precedence_| \endcsname
887 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
888   \csname XINT_expr_precedence_^\endcsname
889 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
890   \csname XINT_expr_precedence_a^\endcsname
891 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
892   \csname XINT_expr_precedence_^[ \endcsname
893 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
894   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
895     \csname XINT_#1_op_=\endcsname
896   \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
897     \csname XINT_#1_op_&\endcsname
898   \expandafter\let\csname XINT_#1_op_| |\expandafter\endcsname
899     \csname XINT_#1_op_| \endcsname
900   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
901     \csname XINT_#1_op_^\endcsname
902   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
903     \csname XINT_#1_op_a^\endcsname
904   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
905     \csname XINT_#1_op_^[ \endcsname

```

906 }%

11.27 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

11.27.1	\xintListSel:x:csv	319
11.27.2	\xintListSel:f:csv	320
11.27.3	\xintKeep:x:csv	321
11.27.4	\xintKeep:f:csv	322
11.27.5	\xintTrim:f:csv	322
11.27.6	\xintNthEltPy:f:csv	322
11.27.7	\xintLength:f:csv	322
11.27.8	\xintReverse:f:csv	322

Python slicing was first implemented for 1.1 (27 octobre 2014). But it used \xintCSVtoList and \xintListWithSep{,} to convert back and forth to token lists for use of \xintKeep, \xintTrim, \xintNthElt. Not very efficient! Also [list][a:b] was Python like but not [list][N] which counted items starting at one, and returned the length for N=0.

Release 1.2g changed this so [list][N] now counts starting at zero and len(list) computes the number of items. Also 1.2g had its own f-expandable macros handling directly the comma separated lists. They are located into [xinttools.sty](#).

1.2j improved the [xinttools.sty](#) macros and furthermore it made the Python slicing in expressions a bit more efficient still by exploiting in some cases that expansion happens in \csname...\endcsname and does not have to be f-expandable. But the f-expandable variants must be kept for use by \xintNewExpr and \xintdeffunc.

```

907 \def\xINT_tmpa #1#2#3#4#5#6%
908 {%
909   \def #1##1% \XINT_expr_op_][
910   {%
911     \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
912   }%
913   \def #2##1##2% \XINT_expr_until_][_a
914   {\xint_UDsignfork
915     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
916     -{#3##1##2}%
917   \krof }%
918   \def #3##1##2##3##4% \XINT_expr_until_][_b
919   {%
920     \ifnum ##2>#5%
921       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
922         \csname XINT_#6_op_##3\endcsname {##4}}%
923     \else
924       \xint_afterfi
925       {\expandafter ##2\expandafter ##3\csname
926         .=\expandafter\xintListSel:x:csv % will be \xintListSel:f:csv in \xintNewExpr output
927         \romannumeral`&&@\XINT_expr_unlock ##4;% selector
928         \XINT_expr_unlock ##1;\endcsname % unlock already pre-positioned for \xintNewExpr
929       }%
930     \fi
931   }%
932   \let #5\xint_c_ii
933 }%
934 \xintFor #1 in {expr,flexpr,iiexpr} \do {%

```

```

935 \expandafter\XINT_tmpa
936   \csname XINT_#1_op_][\expandafter\endcsname
937   \csname XINT_#1_until_][_a\expandafter\endcsname
938   \csname XINT_#1_until_][_b\expandafter\endcsname
939   \csname XINT_#1_op_-vi\expandafter\endcsname
940   \csname XINT_expr_precedence_][\endcsname {#1}%
941 }%
942 \def\XINT_tmpa #1#2#3#4#5#6%
943 {%
944   \def #1##1% \XINT_expr_op_:
945   {%
946     \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
947   }%
948   \def #2##1##2% \XINT_expr_until_:_a
949   {\xint_UDsignfork
950     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
951     -{#3##1##2}%
952   \krof }%
953   \def #3##1##2##3##4% \XINT_expr_until_:_b
954   {%
955     \ifnum ##2>#5%
956     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
957                   \csname XINT_#6_op_##3\endcsname {##4}}%
958   \else
959     \xint_afterfi
960     {\expandafter ##2\expandafter ##3\csname
961       .:=\XINT:NEhook:one\xintNum{\XINT_expr_unlock ##1};%
962       \XINT:NEhook:one\xintNum{\XINT_expr_unlock ##4}%
963       \endcsname
964     }%
965   \fi
966   }%
967   \let #5\xint_c_iii
968 }%
969 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
970 \expandafter\XINT_tmpa
971   \csname XINT_#1_op_:\expandafter\endcsname
972   \csname XINT_#1_until_:_a\expandafter\endcsname
973   \csname XINT_#1_until_:_b\expandafter\endcsname
974   \csname XINT_#1_op_-vi\expandafter\endcsname
975   \csname XINT_expr_precedence_:\endcsname {#1}%
976 }%
977 \catcode`[ 11 \catcode`] 11
978 \let\XINT_expr_precedence_:] \xint_c_iii
979 \def\XINT_expr_op_:] #1%
980 {%
981   \expandafter\xint_c_i\expandafter )%
982   \csname .:=\XINT:NEhook:one\xintNum{\XINT_expr_unlock #1}\endcsname
983 }%
984 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
985 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
986 \let\XINT_expr_precedence_:[: \xint_c_iii

```

At the end of the replacement text of `\XINT_expr_op_`[:], the : after index 0 must be catcode 12, else will be mistaken for the start of variable by expression parser (as `<digits><variable>` is allowed by the syntax and does tacit multiplication).

```
987 \edef\XINT_expr_op_[: #1{\xint_c_ii\noexpand\XINT_expr_itself_][#10\string :}%
988 \let\XINT_flexpr_op_[: \XINT_expr_op_[:
989 \let\XINT_iiexpr_op_[: \XINT_expr_op_[:
990 \catcode`[ 12 \catcode`] 12
```

11.27.1 `\xintListSel:x:csv`

1.2j. Because there is `\xintKeep:x:csv` which is faster than `\xintKeep:f:csv`.

```
991 \def\xintListSel:x:csv #1%
992 {%
993   \if ]\noexpand#1\xint_dothis\XINT_listsel:_s\fi
994   \if : \noexpand#1\xint_dothis\XINT_listxsel:_:\fi
995   \xint_orthat {\XINT_listsel:_nth #1}%
996 }%
997 \def\XINT_listsel:_s #1#2;#3;%
998 {%
999   \if-#1\expandafter\xintKeep:f:csv\else\expandafter\xintTrim:f:csv\fi
1000   {#1#2}{#3}%
1001 }%
1002 \def\XINT_listsel:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%
```

`\XINT_listsel:_nth` and `\XINT_listsel:_s` located in `\xintListSel:f:csv`.

```
1003 \def\XINT_listxsel:_: #1#2;#3#4;%
1004 {%
1005   \xint_UDsignsfork
1006     #1#3\XINT_listxsel:_N:N
1007     #1-\XINT_listxsel:_N:P
1008     -#3\XINT_listxsel:_P:N
1009     --\XINT_listxsel:_P:P
1010   \krof #1#2;#3#4;%
1011 }%
1012 \def\XINT_listxsel:_P:P #1;#2;#3;%
1013 {%
1014   \unless\ifnum #1<#2 \expandafter\xint_gobble_iii\fi
1015   \xintKeep:x:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1016 }%
1017 \def\XINT_listxsel:_N:N #1;#2;#3;%
1018 {%
1019   \expandafter\XINT_listxsel:_N:N_a
1020   \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength:f:csv{#3};#3;%
1021 }%
1022 \def\XINT_listxsel:_N:N_a #1;#2;#3;%
1023 {%
1024   \unless\ifnum #1>\xint_c_ \expandafter\xint_gobble_iii\fi
1025   \xintKeep:x:csv{#1}{\xintTrim:f:csv{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1026 }%
1027 \def\XINT_listxsel:_N:P #1;#2;#3;{\expandafter\XINT_listxsel:_N:P_a
```



```

1028 \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
1029 \def\xINT_listxsel:_N:P_a #1#2;%
1030 { \if -#1\expandafter\xINT_listxsel:_O:P\fi\xINT_listxsel:_P:P #1#2;}%
1031 \def\xINT_listxsel:_O:P\xINT_listxsel:_P:P #1;{\xINT_listxsel:_P:P 0;}%
1032 \def\xINT_listxsel:_P:N #1;#2;#3;{\expandafter\xINT_listxsel:_P:N_a
1033 \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
1034 \def\xINT_listxsel:_P:N_a #1#2;#3;%
1035 { \if -#1\expandafter\xINT_listxsel:_P:O\fi\xINT_listxsel:_P:P #3;#1#2;}%
1036 \def\xINT_listxsel:_P:O\xINT_listxsel:_P:P #1;#2;{\xINT_listxsel:_P:P #1;0;}%

```

11.27.2 \xintListSel:f:csv

1.2g. Since 1.2j this is needed only for \xintNewExpr and user defined functions. Some extras compared to \xintListSel:x:csv because things may not yet have been expanded in the \xintNewExpr context.

```

1037 \def\xintListSel:f:csv #1%
1038 {%
1039 \if ]\noexpand#1\xint_dothis{\expandafter\xINT_listsel:_s\romannumeral`&&@\fi
1040 \if :\noexpand#1\xint_dothis{\xINT_listsel:_:}\fi
1041 \xint_orthat {\xINT_listsel:_nth #1}%
1042 }%
1043 \def\xINT_listsel:_: #1;#2;%
1044 {%
1045 \expandafter\xINT_listsel:_:a
1046 \the\numexpr #1\expandafter;\the\numexpr #2\expandafter;\romannumeral`&&@%
1047 }%
1048 \def\xINT_listsel:_:a #1#2;#3#4;%
1049 {%
1050 \xint_UDsignsfork
1051 #1#3\xINT_listsel:_N:N
1052 #1-\xINT_listsel:_N:P
1053 -#3\xINT_listsel:_P:N
1054 --\xINT_listsel:_P:P
1055 \krof #1#2;#3#4;%
1056 }%
1057 \def\xINT_listsel:_P:P #1;#2;#3;%
1058 {%
1059 \unless\ifnum #1<#2 \xint_afterfi{\expandafter\space\xint_gobble_iii}\fi
1060 \xintKeep:f:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1061 }%
1062 \def\xINT_listsel:_N:N #1;#2;#3;%
1063 {%
1064 \unless\ifnum #1<#2 \expandafter\xINT_listsel:_N:N_abort\fi
1065 \expandafter\xINT_listsel:_N:N_a
1066 \the\numexpr#1+\xintLength:f:csv{#3}\expandafter;\the\numexpr#2-#1;#3;%
1067 }%
1068 \def\xINT_listsel:_N:N_abort #1;#2;#3;{ }%
1069 \def\xINT_listsel:_N:N_a #1;#2;#3;%
1070 {%
1071 \xintKeep:f:csv{#2}{\xintTrim:f:csv{\ifnum#1<\xint_c_\xint_c_\else#1\fi}{#3}}%
1072 }%
1073 \def\xINT_listsel:_N:P #1;#2;#3;{\expandafter\xINT_listsel:_N:P_a

```



```

1074 \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
1075 \def\xINT_listsel:_N:P_a #1#2;%
1076 { \if -#1\expandafter\xINT_listsel:_O:P\fi\xINT_listsel:_P:P #1#2;}%
1077 \def\xINT_listsel:_O:P\xINT_listsel:_P:P #1;{\xINT_listsel:_P:P 0;}%
1078 \def\xINT_listsel:_P:N #1;#2;#3;{\expandafter\xINT_listsel:_P:N_a
1079 \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
1080 \def\xINT_listsel:_P:N_a #1#2;#3;%
1081 { \if -#1\expandafter\xINT_listsel:_P:O\fi\xINT_listsel:_P:P #3;#1#2;}%
1082 \def\xINT_listsel:_P:O\xINT_listsel:_P:P #1;#2;{\xINT_listsel:_P:P #1;0;}%

```

11.27.3 \xintKeep:x:csv

1.2j. This macro is used only with positive first argument.

```

1083 \def\xintKeep:x:csv #1#2%
1084 {%
1085 \expandafter\xint_gobble_i
1086 \romannumeral0\expandafter\xINT_keep:x:csv_pos
1087 \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1088 }%
1089 \def\xINT_keep:x:csv_pos #1.#2%
1090 {%
1091 \expandafter\xINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1092 #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1093 \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1094 }%
1095 \def\xINT_keep:x:csv_loop #1%
1096 {%
1097 \xint_gob_til_minus#1\xINT_keep:x:csv_finish-%
1098 \XINT_keep:x:csv_loop_pickeight #1%
1099 }%
1100 \def\xINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1101 {%
1102 ,#2,#3,#4,#5,#6,#7,#8,#9%
1103 \expandafter\xINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1104 }%
1105 \def\xINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1106 {%
1107 \csname XINT_keep:x:csv_finish#1\endcsname
1108 }%
1109 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
1110 #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1111 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1112 #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
1113 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1114 #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
1115 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1116 #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
1117 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1118 #1,#2,#3,{,#1,#2,#3\xint_Bye}%
1119 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1120 #1,#2,{,#1,#2\xint_Bye}%
1121 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname

```

```

1122 #1,{,#1\xint_Bye}%
1123 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye

```

11.27.4 \xintKeep:f:csv

1.2g. moved to [xinttools](#).

11.27.5 \xintTrim:f:csv

1.2g. moved to [xinttools](#).

11.27.6 \xintNthEltPy:f:csv

1.2g. moved to [xinttools](#).

11.27.7 \xintLength:f:csv

1.2g. moved to [xinttools](#).

11.27.8 \xintReverse:f:csv

1.2g. moved to [xinttools](#).

11.28 Macros for a..b list generation

11.28.1 \xintSeq::csv	322
11.28.2 \xintiSeq::csv	323

Ne produit que des listes d'entiers inférieurs à la borne de TeX ! mais sous la forme N/1[0] en ce qui concerne \xintSeq::csv.

11.28.1 \xintSeq::csv

Commence par remplacer a par ceil(a) et b par floor(b) et renvoie ensuite les entiers entre les deux, possiblement en décroissant, et extrémités comprises. Si a=b est non entier en obtient donc ceil(a) et floor(a). Ne renvoie jamais une liste vide.

Note: le a..b dans \xintfloatexpr utilise cette routine.

```

1124 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
1125 \def\xintseq::csv #1#2%
1126 {%
1127   \expandafter\XINT_seq::csv\expandafter
1128     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
1129     {\the\numexpr \xintiFloor{#2}}}%
1130}%
1131 \def\XINT_seq::csv #1#2%
1132 {%
1133   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1134   \expandafter\XINT_seq::csv_z
1135   \or
1136   \expandafter\XINT_seq::csv_p
1137   \else

```

```

1138     \expandafter\XINT_seq::csv_n
1139   \fi
1140   {#2}{#1}%
1141 }%
1142 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1143 \def\XINT_seq::csv_p #1#2%
1144 {%
1145   \ifnum #1>#2
1146     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1147   \else
1148     \expandafter\XINT_seq::csv_e
1149   \fi
1150   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1151 }%
1152 \def\XINT_seq::csv_n #1#2%
1153 {%
1154   \ifnum #1<#2
1155     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1156   \else
1157     \expandafter\XINT_seq::csv_e
1158   \fi
1159   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1160 }%
1161 \def\XINT_seq::csv_e #1,{ }%

```

11.28.2 \xintiiSeq::csv

```

1162 \def\xintiiSeq::csv {\romannumeral0\xintiiSeq::csv }%
1163 \def\xintiiSeq::csv #1#2%
1164 {%
1165   \expandafter\XINT_iiseq::csv\expandafter
1166   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1167 }%
1168 \def\XINT_iiseq::csv #1#2%
1169 {%
1170   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1171   \expandafter\XINT_iiseq::csv_z
1172   \or
1173   \expandafter\XINT_iiseq::csv_p
1174   \else
1175   \expandafter\XINT_iiseq::csv_n
1176   \fi
1177   {#2}{#1}%
1178 }%
1179 \def\XINT_iiseq::csv_z #1#2{ #1}%
1180 \def\XINT_iiseq::csv_p #1#2%
1181 {%
1182   \ifnum #1>#2
1183     \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1184   \else
1185     \expandafter\XINT_seq::csv_e
1186   \fi
1187   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%

```

```

1188 }%
1189 \def\XINT_iiseq::csv_n #1#2%
1190 {%
1191     \ifnum #1<#2
1192         \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1193     \else
1194         \expandafter\XINT_seq::csv_e
1195     \fi
1196     \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1197 }%
1198 \def\XINT_seq::csv_e #1,{ }%

```

11.29 Macros for a..[d].b list generation

11.29.1	\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv	324
11.29.2	\xintSeqB::csv	324
11.29.3	\xintiiSeqB::csv	325
11.29.4	\XINTinFloatSeqB::csv	325

Contrarily to a..b which is limited to small integers, this works with a, b, and d (big) fractions. It will produce a «nil» list, if a>b and d<0 or a<b and d>0.

11.29.1 \xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv

```

1199 \def\xintSeqA::csv #1%
1200     {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%
1201 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1;}%
1202 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseq::csv\expandafter{\romannumeral`&&@#1}}%
1203 \def\XINT_iiseq::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1;}%
1204 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1205     {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1206 \def\XINT_flseqa::csv #1#2%
1207     {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1208 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1209     #1-{z}%
1210     0#1{n}%
1211     0-{p}%
1212     \krof #1}%

```

11.29.2 \xintSeqB::csv

With one year late documentation, let's just say, the #1 is \XINT_expr_unlock\.=Ua;b; with U=z or n or p, a=step, b=start.

```

1213 \def\xintSeqB::csv #1#2%
1214     {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1215 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1216 \def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1217     \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1218 \def\XINT_seqb::csv_p #1#2#3%
1219 {%
1220     \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1221     {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

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

```
1222   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1223 }%
1224 \def\XINT_seqb::csv_n #1#2#3%
1225 {%
1226   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1227   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1228   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1229 }%
1230 \def\XINT_seqb::csv_z #1#2#3{, #1}%
```

11.29.3 `\xintiiSeqB::csv`

```
1231 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1232 \def\XINT_iiseqb::csv #1#2#3#4%
1233   {\expandafter\XINT_iiseqb::csv_a
1234    \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1235    \romannumeral`&&\XINT_expr_unlock #4!}%
1236 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1237   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1238 \def\XINT_iiseqb::csv_p #1#2#3%
1239 {%
1240   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1241   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1242   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1243 }%
1244 \def\XINT_iiseqb::csv_n #1#2#3%
1245 {%
1246   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1247   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1248   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1249 }%
1250 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%
```

11.29.4 `\XINTinFloatSeqB::csv`

```
1251 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1252   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1253 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&#2#1!}%
1254 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1255   \romannumeral`&&\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1256 \def\XINT_flseqb::csv_p #1#2#3%
1257 {%
1258   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1259   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1260   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1261 }%
1262 \def\XINT_flseqb::csv_n #1#2#3%
1263 {%
1264   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1265   {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1266   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1267 }%
```

```
1268 \def\XINT_flseqb::csv_z #1#2#3{,#1}%
```

11.30 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```
1269 \def\XINT_tmpa #1#2#3#4#5#6%
1270 {%
1271   \def #1##1% \XINT_expr_op_,
1272   {%
1273     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
1274   }%
1275   \def #2##1##2% \XINT_expr_until_,_a
1276   {\xint_UDsignfork
1277     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
1278     -{##1##2}%
1279   \krof }%
1280   \def #3##1##2##3##4% \XINT_expr_until_,_b
1281   {%
1282     \ifnum ##2>\xint_c_ii
1283       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
1284                     \csname XINT_#6_op_##3\endcsname {##4}}%
1285     \else
1286       \xint_afterfi
1287       {\expandafter ##2\expandafter ##3%
1288        \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1289     \fi
1290   }%
1291   \let #5\xint_c_ii
1292 }%
1293 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1294 \expandafter\XINT_tmpa
1295   \csname XINT_#1_op_,\expandafter\endcsname
1296   \csname XINT_#1_until_,_a\expandafter\endcsname
1297   \csname XINT_#1_until_,_b\expandafter\endcsname
1298   \csname XINT_#1_op_-vi\expandafter\endcsname
1299   \csname XINT_expr_precedence_,\endcsname {#1}%
1300 }%
```

11.31 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```
1301 \def\XINT_tmpa #1#2#3%
1302 {%
1303   \expandafter\XINT_tmppb
1304   \csname XINT_#1_op_-#3\expandafter\endcsname
1305   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1306   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1307   \csname xint_c_#3\endcsname {#1}#2%
1308 }%
1309 \def\XINT_tmppb #1#2#3#4#5#6%
1310 {%
```

```

1311 \def #1% \XINT_expr_op_-<level>
1312 {% get next number+operator then switch to _until macro
1313 \expandafter #2\romannumeral`&&\XINT_expr_getnext
1314 }%
1315 \def #2##1% \XINT_expr_until_-<l>_a
1316 {\xint_UDsignfork
1317 ##1{\expandafter #2\romannumeral`&&@#1}%
1318 -{#3##1}%
1319 \krof }%
1320 \def #3##1##2##3% \XINT_expr_until_-<l>_b
1321 {% _until tests precedence level with next op, executes now or postpones
1322 \ifnum ##1>#4%
1323 \xint_afterfi {\expandafter #2\romannumeral`&&@%
1324 \csname XINT_#5_op_##2\endcsname {##3}}%
1325 \else
1326 \xint_afterfi {\expandafter ##1\expandafter ##2%
1327 \csname .=%
1328 \XINT:NEhook:one#6{\XINT_expr_unlock ##3}\endcsname }%
1329 \fi
1330 }%
1331 }%

```

1.2d needs precedence 8 for *** and 9 for ^. Earlier, precedence level for ^ was only 8 but nevertheless the code did also "ix" here, which I think was unneeded back then.

```

1332 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1333 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1334 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{\vi}{vii}{viii}{ix}}%

```

11.32 ? as two-way and ?? as three-way conditionals with braced branches

In 1.1, I overload ? with ??, as : will be used for list extraction, problem with (stuff)?{?(1)}{0} for example, one should put a space (stuff)?{?(1)}{0} will work. Small idiosyncrasy. (which has been removed in 1.2h, there is no problem anymore with (test)?{?(1)}{0}, however (test)?{?}{!(x)} is not accepted; but (test)?{?(x)}{!(x)} is or even with {?(x)}{!(x)}.)

syntax: ?{yes}{no} and ??{<0}{=0}{>0}.

The difficulty is to recognize the second ? without removing braces as would be the case with standard parsing of operators. Hence the ? operator is intercepted in \XINT_expr_getop_b.

1.2h corrects a bug in \XINT_expr_op_? which in context like (test)?{\foo}{bar} would provoke expansion of \foo, or also with (test)?{}{bar} would result in an error. The fix also solves the (test)?{?(1)}{0} issue mentioned above.

```

1335 \let\XINT_expr_precedence_? \xint_c_x
1336 \def\XINT_expr_op_? #1#2%
1337 {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
1338 \def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
1339 \def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
1340 \else\expandafter\xint_bye\fi }%
1341 \def\XINT_expr_op_?checkc #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1342 \def\XINT_expr_op_?a #1#2#3%
1343 {%
1344 \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1345 }%

```

```

1346 \let\XINT_flexpr_op_?\XINT_expr_op_?
1347 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1348 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
1349 {%
1350     \xintiiifSgn {\XINT_expr_unlock #1}%
1351     {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1352 }%

```

11.33 ! as postfix factorial operator

```

1353 \let\XINT_expr_precedence_! \xint_c_x
1354 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1355     \csname .=\XINT:NEhook:one\xintFac{\XINT_expr_unlock #1}\endcsname }%
1356 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1357     \csname .=\XINT:NEhook:one\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1358 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1359     \csname .=\XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #1}\endcsname }%

```

11.34 The A/B[N] mechanism

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

```

1360 \catcode`[ 11
1361 \let\XINT_expr_precedence_[ \xint_c_vii
1362 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1363     \csname .=\xintE{\XINT_expr_unlock #1}%
1364     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1365 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1366     \csname .=\xintiiE{\XINT_expr_unlock #1}%
1367     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1368 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1369     \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1370     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1371 \catcode`[ 12

```

11.35 \XINT_expr_op_` for recognizing functions

The "onliteral" intercepts is for bool, tog1, protect, ... but also for add, mul, seq, etc... Genuine functions have expr, iiexpr and flexpr versions (or only one or two of the three).

With 1.2c "onliteral" is also used to disambiguate variables from functions. However as I use only a \ifcsname test, in order to be able to re-define a variable as function, I move the check for being a function first. Each variable name now has its onliteral_<name> associated macro which is the new way tacit multiplication in front of a parenthesis is implemented. This used to be decided much earlier at the time of \XINT_expr_func.

The advantage of our choices for 1.2c is that the same name can be used for a variable or a function, the parser will apply the correct interpretation which is decided by the presence or not of an opening parenthesis next.


```

1372 \def\XINT_tmpa #1#2#3{%
1373   \def #1##1%
1374   {%
1375     \ifcsname XINT_#3_func_##1\endcsname
1376     \xint_dothis{\expandafter\expandafter
1377               \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1378     \ifcsname XINT_expr_onliteral_##1\endcsname
1379     \xint_dothis{\csname XINT_expr_onliteral_##1\endcsname}\fi
1380     \xint_orthat{\XINT_expr_unknown_function {##1}%
1381               \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1382   }%
1383 }%
1384 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1385 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1386   \expandafter\XINT_tmpa
1387     \csname XINT_#1_op_`\expandafter\endcsname
1388     \csname XINT_#1_oparen\endcsname
1389     {#1}%
1390 }%
1391 \def\XINT_expr_func_unknown #1#2#3%
1392   {\expandafter #1\expandafter #2\csname .=#0\endcsname }%

```

11.36 The `\bool()`, `\togl()`, `\protect()` pseudo “functions”

`bool`, `togl` and `protect` use delimited macros. They are not true functions, they turn off the parser to gather their “variable”.

```

1393 \def\XINT_expr_onliteral_bool #1)%
1394   {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1395 \def\XINT_expr_onliteral_togl #1)%
1396   {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1397 \def\XINT_expr_onliteral_protect #1)%
1398   {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%

```

11.37 The `\break()` function

`break` is a true function, the parsing via expansion of the succeeding material proceeded via `_oparen` macros as with any other function.

```

1399 \def\XINT_expr_func_break #1#2#3%
1400   {\expandafter #1\expandafter #2\csname .=?\romannumeral`&&\XINT_expr_unlock #3\endcsname }%
1401 \let\XINT_flexpr_func_break \XINT_expr_func_break
1402 \let\XINT_iiexpr_func_break \XINT_expr_func_break

```

11.38 The `\qraw()`, `\qint()`, `\qfrac()`, and `\qfloat()` “functions”

1.2. adds `\qint()`, `\qfrac()`, `\qfloat()`.

1.3c. adds `\qraw()`. Useful to limit impact on TeX memory from abuse of `\csname`'s storage when generating many comma separated values from a loop.

They allow the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general. The `\qraw()` does no post-processing at all apart complete

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

expansion, useful for comma-separated values, but must be obedient to (non really documented) expected format. Each uses a delimited macro, the closing parenthesis can not emerge from expansion.

```

1403 \def\XINT_expr_onliteral_qint #1)%
1404     {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1405 \def\XINT_expr_onliteral_qfrac #1)%
1406     {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1407 \def\XINT_expr_onliteral_qfloat #1)%
1408     {\expandafter\XINT_expr_getop\csname .=\XINTinFloatdigits{#1}\endcsname }%
1409 \def\XINT_expr_onliteral_qraw #1)%
1410     {\expandafter\XINT_expr_getop\csname .=#1\endcsname }%

```

11.39 The \random() and \grand() “functions”

1.3b. Function-like syntax but with no argument currently, so let's use fast parsing which requires though the closing parenthesis to be explicit.

```

1411 \def\XINT_expr_onliteral_random #1)%
1412     {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSdigits\endcsname }%
1413 \def\XINT_expr_onliteral_grand #1)%
1414     {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSixteen\endcsname }%

```

11.40 \XINT_expr_op__ for recognizing variables

The 1.1 mechanism for \XINT_expr_var_<varname> has been modified in 1.2c. The <varname> associated macro is now only expanded once, not twice. We arrive here via \XINT_expr_func.

```

1415 \def\XINT_expr_op__ #1% op__ with two _'s
1416     {%
1417         \ifcsname XINT_expr_var_#1\endcsname
1418             \expandafter\xint_firstoftwo
1419         \else
1420             \expandafter\xint_secondoftwo
1421         \fi
1422         {\expandafter\expandafter\expandafter
1423           \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1424         {\XINT_expr_unknown_variable {#1}%
1425           \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1426     }%
1427 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1428 \let\XINT_flexpr_op__ \XINT_expr_op__
1429 \let\XINT_iiexpr_op__ \XINT_expr_op__

```

11.41 User defined variables: \xintdefvar, \xintdefiivar, \xintdeffloatvar

1.1.

1.2p (2017/12/01). extends \xintdefvar et. al. to accept simultaneous assignments to multiple variables.

1.3c (2018/06/17). Use \xintexprSafeCatcodes (to palliate issue with active semi-colon from Babel+French if in body of a \TeX document).

And allow usage with both syntaxes `name:=expr`; or `name=expr`; . Also the colon may have catcode 11, 12, or 13 with no issue.

Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with @ or an underscore are reserved.

```

1430 \catcode`* 11
1431 \def\XINT_expr_defvar_one #1#2%
1432 {%
1433   \XINT_global
1434   \expandafter\edef\csname XINT_expr_var_#1\endcsname
1435     {\expandafter\noexpand#2}%
1436   \XINT_global
1437   \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1438     {\XINT_expr_precedence_*** *\expandafter\noexpand#2}%
1439   \ifxintverbose\xintMessage{xintexpr}{Info}
1440     {Variable "#1" \ifxintglobaldefs globally \fi
1441     defined with value \expandafter\XINT_expr_unlock#2.}%
1442   \fi
1443 }%
1444 \catcode`* 12

1445 \catcode`~\active
1446 \catcode`: 12
1447 \def\XINT_expr_defvar_getname #1:#2~{\endgroup
1448   \def\XINT_expr_tmpa{#1}\edef\XINT_expr_tmppc{\xintCSVLength{#1}}}%
1449 \def\XINT_expr_defvar #1#2#3;%
1450 {%
1451   \xintexprRestoreCatcodes

```

Maybe SafeCatcodes was without effect because the colon and the rest are from some earlier macro definition. Give a safe definition to active colon (even if in math mode with a math active colon..).

```

1452   \beginngroup\lccode`~: \lowercase{\let~}\empty
1453   \edef\XINT_expr_tmpa{#2}%
1454   \edef\XINT_expr_tmpa{\xint_zapspaces_o\XINT_expr_tmpa}%
1455   \expandafter\XINT_expr_defvar_getname
1456     \detokenize\expandafter{\XINT_expr_tmpa}:~%
1457   \ifcase\XINT_expr_tmppc
1458     \xintMessage {xintexpr}{Warning}
1459     {Aborting: impossible to declare variable with empty name.}%
1460   \or
1461     \edef\XINT_expr_tmppb{\romannumeral0#1#3\relax}%
1462     \XINT_expr_defvar_one\XINT_expr_tmpa\XINT_expr_tmppb
1463   \else
1464     \edef\XINT_expr_tmppb
1465       {\expandafter\XINT_expr_unlock\romannumeral0#1#3\relax}%
1466     \edef\XINT_expr_tmppd{\xintCSVLength{\XINT_expr_tmppb}}%
1467     \ifnum\XINT_expr_tmppc=\XINT_expr_tmppd\space
1468       \xintAssignArray\xintCSVtoList\XINT_expr_tmpa\to\XINT_expr_tmppvar
1469       \xintAssignArray
1470       \xintApply\XINT_expr_lockit{\xintCSVtoList\XINT_expr_tmppb}%
1471       \to\XINT_expr_tmppval
1472     \def\XINT_expr_tmppd{1}%
1473     \xintloop
1474       \expandafter\XINT_expr_defvar_one

```

```

1475      \csname XINT_expr_tmpvar\XINT_expr_tmpd\expandafter\endcsname
1476      \csname XINT_expr_tmpval\XINT_expr_tmpd\endcsname
1477      \ifnum\XINT_expr_tmpd<\XINT_expr_tmpc\space
1478        \edef\XINT_expr_tmpd{\the\numexpr\XINT_expr_tmpd+1}%
1479      \repeat
1480      \xintRelaxArray\XINT_expr_tmpvar
1481      \xintRelaxArray\XINT_expr_tmpval
1482    \else
1483      \xintMessage {xintexpr}{Warning}
1484      {Aborting: mismatch between number of variables (\XINT_expr_tmpc)
1485        and number of values (\XINT_expr_tmpd).}%
1486    \fi
1487  \fi
1488 }%
1489 \catcode`~ 3
1490 \catcode`: 11

```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

```

1491 \def\xintdefvar      {\xintexprSafeCatcodes\xintdefvar_a}%
1492 \def\xintdefiivar    {\xintexprSafeCatcodes\xintdefiivar_a}%
1493 \def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
1494 \def\xintdefvar_a    #1={\XINT_expr_defvar\xintbareeval      {#1}}%
1495 \def\xintdefiivar_a  #1={\XINT_expr_defvar\xintbareiieval    {#1}}%
1496 \def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintbarefloateval {#1}}%

```

11.42 \xintunassignvar

1.2e. Currently not possible to genuinely ``undefine'' a variable, all we can do is to let it stand for zero and generate an error. The reason is that I chose to use \ifcsname tests in \XINT_expr_op__ and \XINT_expr_op_`.

```

1497 \def\xintunassignvar #1{%
1498   \edef\XINT_expr_tmpa{#1}%
1499   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
1500   \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1501     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1502       \expandafter\xintnewdummy\XINT_expr_tmpa
1503     \else
1504       \XINT_global
1505       \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1506       {\csname .:=0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
1507       \XINT_global
1508       \expandafter\edef\csname XINT_expr_onliteral_\XINT_expr_tmpa\endcsname
1509       {\csname .:=0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
1510       \ifxintverbose\xintMessage {xintexpr}{Info}
1511       {Variable \XINT_expr_tmpa\space has been
1512        \ifxintglobaldefs globally \fi ``unassigned''.}%
1513     \fi
1514   \fi
1515 \else
1516   \xintMessage {xintexpr}{Warning}

```

```

1517         {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1518     \fi
1519 }%
1520 \def\XINT_expr_undefined #1{\xintError:replaced_by_zero\xint_gobble_i {#1}}%

```

11.43 seq and the implementation of dummy variables

11.43.1	All letters usable as dummy variables	333
11.43.2	\omit() and \abort()	334
11.43.3	The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	335
11.43.4	\XINT_expr_onliteral_seq	336
11.43.5	\XINT_expr_onliteral_seq_a	336
11.43.6	\XINT_isbalanced_a for \XINT_expr_onliteral_seq_a	336
11.43.7	\XINT_allexpr_func_seqx	337
11.43.8	Evaluation over list, \XINT_expr_seq:_a with break, abort, omit	337
11.43.9	Evaluation over ++ generated lists with \XINT_expr_seq:_A	338

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain \xintNew-Expr !)

The \XINT_expr_onliteral_seq_a parses: "expression, variable=list)" (when it is called the opening (has been swallowed, and it looks for the ending one.) Both expression and list may themselves contain parentheses and commas, we allow nesting. For example "x^2,x=1..10)", at the end of seq_a we have {variable{expression}}{list}, in this example {x{x^2}}{1..10}, or more complicated "seq(add(y,y=1..x),x=1..10)" will work too. The variable is a single lowercase Latin letter.

The complications with \xint_c_xviii in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iexpr.

11.43.1 All letters usable as dummy variables

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..[d]..b, or for seq with dummy variable where omit has omitted everyting may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of \XINT_expr_onliteral_<name>.

In 1.1 a letter variable say X was acting as a delimited macro looking for !X{stuff} and then would expand the stuff inside a \csname=...\endcsname. I don't think I used the possibilities this opened and the 1.2c version has stuff_already_ encapsulated thus a single token. Only one expansion, not two is then needed in \XINT_expr_op__.

I had to accordingly modify seq, add, mul and subs, but fortunately realized that the @, @1, etc... variables for rseq, rrseq and iter already had been defined in the way now also followed by the Latin letters as dummy variables.

The 1.2e \XINT_expr_makedummy was adjoined \xintnewdummy by 1.2k for a public interface. It should not be used with multi-letter argument. The add, mul, seq, etc... can only work with one-letter long dummy variable. And this will almost certainly not change.

Also 1.2e does the tacit multiplication `x(stuff)->x*(stuff)` in its higher precedence form. Things are easy now that variables always fetch a single already locked value `\.=<number>`.

The tacit multiplication in case of the ``nil'` variable doesn't make much sense but we do it anyhow.

```

1521 \catcode`* 11
1522 \def\XINT_expr_makedummy #1%
1523 {%
1524   \XINT_global
1525   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1526   {##2##1\relax !#1##2}%
1527   \XINT_global
1528   \expandafter\def\csname XINT_expr_onliteral_#1\endcsname ##1\relax !#1##2%
1529   {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
1530 }%
1531 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1532 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1533 \def\xintnewdummy #1{%
1534   \XINT_expr_makedummy{#1}%
1535   \ifxintverbose\xintMessage {xintexpr}{Info}%
1536   {#1 (with letter catcode) now
1537   \ifxintglobaldefs globally \fi usable as dummy variable.}%
1538   \fi
1539 }%
1540 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1541 \edef\XINT_expr_onliteral_nil
1542   {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname {}%
1543 \catcode`* 12

```

11.43.2 \omit() and \abort()

June 24 and 25, 2014.

Added comments 2015/11/13:

Et la documentation ? on n'y comprend plus rien. Trop rusé.

```
\def\XINT_expr_var_omit #1\relax !{1^C!}{}}{}}\.=!\relax !}
```

```
\def\XINT_expr_var_abort #1\relax !{1^C!}{}}{}}\.=^\relax !}
```

C'était accompagné de `\XINT_expr_precedence_^C=0` et d'un hack au sein même des macros until de plus bas niveau.

Le mécanisme sioux était le suivant: `^C` est déclaré comme un opérateur de précédence nulle. Lorsque le parseur trouve un "omit" dans un seq ou autre, il va insérer dans le stream `\XINT_expr_getop` suivi du texte de remplacement. Donc ici on avait un 1 comme place holder, puis l'opérateur `^C`. Celui-ci étant de précédence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le `until_end_b` (et le `until_)_b`) qui confronté à `^C`, va se relancer à zéro, le `getnext` va trouver le `!{}}{}}\.=!` et ensuite il y aura `\relax`, et le résultat sera `\.=!` pour omit ou `\.=^` pour abort. Les routines des boucles seq, iter, etc... peuvent alors repérer le `!` ou `^` et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais `^C` a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car `seq(2^C, C=1..5)` est alors impossible. De toute façon ce `^C` était à usage interne uniquement.

Il me faut un symbole d'opérateur qui ne rentre pas en conflit. Bon je vais prendre `!?`. Ensuite au lieu de hacker `until_end`, il vaut mieux lui donner précédence 2 (mais ça ne pourra pas marcher à l'intérieur de parenthèses il faut d'abord les fermer manuellement) et lui associer un simplement

un op spécial. Je n'avais pas fait cela peut-être pour éviter d'avoir à définir plusieurs macros. Le #1 dans la définition de \XINT_expr_op_!? est le résultat de l'évaluation forcée précédente.

Attention que les premier ! doivent être de catcode 12 sinon ils signalent une sous-expression qui déclenche une multiplication tacite.

```
1544 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
1545 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
1546 \def\XINT_expr_op_!? #1#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
1547 \let\XINT_iexpr_op_!? \XINT_expr_op_!?
1548 \let\XINT_flexpr_op_!? \XINT_expr_op_!?
```

11.43.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

October 2014: I had completely forgotten what the @@@ etc... stuff were supposed to do: this is for nesting recursions! (I was mad back in June). @@(N) gives the Nth back, @@@(N) gives the Nth back of the higher recursion!

1.2c adds the needed "onliteral" now that tacit multiplication between a variable and a (has a new mechanism. 1.2e does this tacit multiplication with higher precedence.

For the record, the ~ has catcode 3 in this code.

```
1549 \catcode`? 3 \catcode`* 11
1550 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1551 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1552 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1553 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1554 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1555 \def\XINT_expr_onliteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2)%
1556 \expandafter\let\csname XINT_expr_onliteral_@1\endcsname \XINT_expr_onliteral_@
1557 \expandafter\def\csname XINT_expr_onliteral_@2\endcsname #1~#2#3%
1558     {\XINT_expr_precedence_*** *#3(#1~#2#3)%
1559 \expandafter\def\csname XINT_expr_onliteral_@3\endcsname #1~#2#3#4%
1560     {\XINT_expr_precedence_*** *#4(#1~#2#3#4)%
1561 \expandafter\def\csname XINT_expr_onliteral_@4\endcsname #1~#2#3#4#5%
1562     {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5)%
1563 \catcode`* 12
1564 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1565 {%
1566     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1567         {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1568}%
1569 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1570 {%
1571     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1572     {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1573}%
1574 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1575 {%
1576     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1577     {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1578}%
1579 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1580 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1581 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
```

```

1582 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1583 {%
1584     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1585     {\XINT_expr_unlock#3}{#5}#4~#5?%
1586 }%
1587 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1588 {%
1589     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1590     {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1591 }%
1592 \def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1593 {%
1594     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1595     {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1596 }%
1597 \catcode`? 11

```

11.43.4 \XINT_expr_onliteral_seq

```

1598 \def\XINT_expr_onliteral_seq
1599 {\expandafter\XINT_expr_onliteral_seq_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1600 \def\XINT_expr_onliteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

11.43.5 \XINT_expr_onliteral_seq_a

```

1601 \def\XINT_expr_onliteral_seq_a #1#2,%
1602 {%
1603     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1604         \expandafter\XINT_expr_onliteral_seq_c
1605         \or\expandafter\XINT_expr_onliteral_seq_b
1606         \else\expandafter\xintError:we_are_doomed
1607     \fi {#1#2},%
1608 }%
1609 \def\XINT_expr_onliteral_seq_b #1,{\XINT_expr_onliteral_seq_a {#1},}%
1610 \def\XINT_expr_onliteral_seq_c #1,#2#3% #3 pour absorber le =
1611 {%
1612     \XINT_expr_onliteral_seq_d {#2{#1}}}%
1613 }%
1614 \def\XINT_expr_onliteral_seq_d #1#2#3)%
1615 {%
1616     \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1617         \or\expandafter\XINT_expr_onliteral_seq_e
1618         \else\expandafter\xintError:we_are_doomed
1619     \fi
1620     {#1}{#2#3}%
1621 }%
1622 \def\XINT_expr_onliteral_seq_e #1#2{\XINT_expr_onliteral_seq_d {#1}{#2}}%

```

11.43.6 \XINT_isbalanced_a for \XINT_expr_onliteral_seq_a

Expands to \xint_c_mone in case a closing) had no opening (matching it, to \@ne if opening) had no closing) matching it, to \z@ if expression was balanced.

```

1623 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1624 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%

```



```

1625 \def\XINT_isbalanced_b #1)#2%
1626   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%

    if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1

1627 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%

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

1628 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1629   {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%

    #1 is \xint_bye, there was never ( nor ) in original #1, hence OK.

1630 \def\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d\xint_bye )\xint_bye {\xint_c_ }%

    #1 is not \xint_bye, there was indeed a ( in original #1. We check if we see a ). If we do, we then
    loop until no ( nor ) is to be found.

1631 \def\XINT_isbalanced_d #1)#2%
1632   {\xint_bye #2\XINT_isbalanced_no\xint_bye\XINT_isbalanced_a #1#2}%

    #2 was \xint_bye, we did not find a closing ) in original #1. Error.

1633 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

11.43.7 \XINT_allexpr_func_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT_expr_seq:_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT_allexpr_seqx, #2 is the list, evaluated and encapsulated, #3 is the dummy variable, #4 is the expression to evaluate repeatedly.

A special case is a list generated by <variable>+: then #2 is {\.=+\.=<start>}.

```

1634 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareeval }%
1635 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1636 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
1637 \def\XINT_allexpr_seqx #1#2#3#4%
1638 {%
1639   \expandafter \XINT_expr_getop
1640   \csname .=\expandafter\XINT_expr_seq:_aa
1641     \romannumeral`&&\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1642 }%
1643 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1644   \expandafter\XINT_expr_seq:_a\fi #1}%

```

11.43.8 Evaluation over list, \XINT_expr_seq:_a with break, abort, omit

The #2 here is \...bareeval <expression>\relax !<variable name>. The #1 is a comma separated list of values to assign to the dummy variable. The \XINT_expr_seq_empty? intervenes immediately after handling of firstvalue.

1.2c has rewritten to a large extent this and other similar loops because the dummy variables now fetch a single encapsulated token (apart from a good means to lose a few hours needlessly -- as I have had to rewrite and review most everything, this change could make the thing more efficient

if the same variable is used many times in an expression, but we are talking micro-seconds here anyhow.)

```

1645 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1646         \romannumeral0\XINT_expr_seq:_b {#2}#1,^,}%
1647 \def\XINT_expr_seq:_b #1#2#3,{%
1648     \if ,#2\xint_dothis\XINT_expr_seq:_noop\fi
1649     \if ^#2\xint_dothis\XINT_expr_seq:_end\fi
1650     \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1651 }%
1652 \def\XINT_expr_seq:_noop\csname.=,#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
1653 \def\XINT_expr_seq:_end \csname.=^ \endcsname #1{%
1654 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1655 \def\XINT_expr_seq:_d #1{\if #1^ \xint_dothis\XINT_expr_seq:_abort\fi
1656     \if #1? \xint_dothis\XINT_expr_seq:_break\fi
1657     \if #1! \xint_dothis\XINT_expr_seq:_omit\fi
1658     \xint_orthat{\XINT_expr_seq:_goon #1}}%
1659 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1660 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1661 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1662 \def\XINT_expr_seq:_goon #1!#2#3#4{,#1\XINT_expr_seq:_b {#4}}%

```

If all is omitted or list is empty, `_empty?` will fetch within the `##1` a `\endcsname` token and construct "nil" via `<space>\endcsname`, if not `##1` will be a comma and the gobble will swallow the space token and the extra `\endcsname`.

```

1663 \def\XINT_expr_seq_empty? #1{%
1664 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1665 \XINT_expr_seq_empty? { }%

```

11.43.9 Evaluation over ++ generated lists with `\XINT_expr_seq:_A`

This is for index lists generated by `n++`. The starting point will have been replaced by its ceil (added: in fact with version 1.1. the ceil was not yet evaluated, but `_var<letter>` did an expansion of what they fetch). We use `\numexpr` rather than `\xintInc`, hence the indexing is limited to small integers.

The 1.2c version of `n++` produces a `#1` here which is already a single `\.=<value>` token.

```

1666 \def\XINT_expr_seq:_A +#1!%
1667     {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1668 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1669 \def\XINT_expr_seq:_E #1{\if #1^ \xint_dothis\XINT_expr_seq:_Abort\fi
1670     \if #1? \xint_dothis\XINT_expr_seq:_Break\fi
1671     \if #1! \xint_dothis\XINT_expr_seq:_Omit\fi
1672     \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1673 \def\XINT_expr_seq:_Abort #1!#2#3#4{%
1674 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1675 \def\XINT_expr_seq:_Omit #1!#2#3%
1676     {\expandafter\XINT_expr_seq:_D
1677         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1678 \def\XINT_expr_seq:_Goon #1!#2#3%
1679     {,#1\expandafter\XINT_expr_seq:_D
1680         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%

```

11.44 \add(), \mul()

1.2c uses more directly the \xintiiAdd etc... macros and has opxadd/opxmul rather than a single opx. This is less conceptual as I use explicitly the associated macro names for +, * but this makes other things more efficient, and the code more readable.

```
1681 \def\XINT_expr_onliteral_add
1682 {\expandafter\XINT_expr_onliteral_add_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1683 \def\XINT_expr_onliteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1684 \def\XINT_expr_onliteral_mul
1685 {\expandafter\XINT_expr_onliteral_mul_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1686 \def\XINT_expr_onliteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%

```

11.44.1 \XINT_expr_func_opxadd, \XINT_flexpr_func_opxadd, \XINT_iiexpr_func_opxadd and same for mul

modified 1.2c.

```
1687 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1688 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1689 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1690 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1691 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1692 \def\XINT_iiexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiMul 1}}%

```

#1=bareval etc, #2={Add0} ou {Mul1}, #3=liste encapsulée, #4=la variable, #5=expression

```
1693 \def\XINT_allexpr_opx #1#2#3#4#5%
1694 {%
1695   \expandafter\XINT_expr_getop
1696   \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1697   \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1698 }%
1699 \def\XINT_expr_op:_a #1!#2#3{\XINT_expr_op:_b #3{#2}#1,^,%}

```

#2 in \XINT_expr_op:_b is the partial result of computation so far, not locked. A noop with have #4=, and #5 the next item which we need to recover. No need to be very efficient for that in op:_noop. In op:_d, #4 is \xintAdd or similar.

```
1700 \def\XINT_expr_op:_b #1#2#3#4#5,{%
1701   \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1702   \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1703   \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1704 }%
1705 \def\XINT_expr_op:_c #1#2#3#4%
1706   {\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1707 \def\XINT_expr_op:_d #1!#2#3#4#5%
1708   {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1709   {\romannumeral`&&\XINT:NEhook:two#4{\XINT_expr_unlock#1}{#5}}}%

```

The replacement text had expr_seq:_b rather than expr_op:_b due to a left-over from copy-paste. This made add and mul fail with an empty range for the variable (or "nil" in the list of values). Fixed in 1.2h.

```
1710 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,%}
1711 \def\XINT_expr_op:_end \csname.=^ \endcsname #1#2#3{#3}%

```

11.45 \subs()

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

```

1712 \def\XINT_expr_onliteral_subs
1713 {\expandafter\XINT_expr_onliteral_subs_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1714 \def\XINT_expr_onliteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1715 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval}%
1716 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1717 \def\XINT_iexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval}%
1718 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1719 {% #3 is the dummy variable, #4 is the expression to evaluate
1720   \expandafter\expandafter\expandafter\XINT_expr_getop
1721   \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1722 }%
1723 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

11.46 \rseq()

11.46.1	\XINT_expr_rseqx	340
11.46.2	\XINT_expr_rseqy	341
11.46.3	\XINT_expr_rseq:_a etc...	341
11.46.4	\XINT_expr_rseq:_A etc...	341

When func_rseq has its turn, initial segment has been scanned by oparen, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion. Notice that the ; is discovered during standard parsing mode, it may be for example {;} or arise from expansion as rseq does not use a delimited macro to locate it.

Here and in rrseq and iter, 1.2c adds also use of \xintthebareeval, etc...

```

1724 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval}%
1725 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval}%
1726 \def\XINT_iexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval}%
1727 \def\XINT_allexpr_rseq #1#2#3%
1728 {%
1729   \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1730   #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1731 }%

```

11.46.1 \XINT_expr_rseqx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1732 \def\XINT_expr_rseqx #1#2#3#4#5%
1733 {%
1734   \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1735 }%

```

11.46.2 \XINT_expr_rseq

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```
1736 \def\XINT_expr_rseq #1#2#3#4#5%
1737 {%
1738     \expandafter \XINT_expr_getop
1739     \csname .=\XINT_expr_unlock #2%
1740     \expandafter\XINT_expr_rseq:_aa
1741         \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1742 }%
1743 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1744     \expandafter\XINT_expr_rseq:_a\fi #1}%

```

11.46.3 \XINT_expr_rseq:_a etc...

```
1745 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1746 \def\XINT_expr_rseq:_b #1#2#3#4,%{
1747     \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1748     \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1749     \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname .=#3#4\endcsname
1750     {#1}{#2}%
1751 }%
1752 \def\XINT_expr_rseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1753 \def\XINT_expr_rseq:_end \csname .=#1\endcsname #1#2{%}
1754 \def\XINT_expr_rseq:_c #1#2#3%
1755     {\expandafter\XINT_expr_rseq:_d\romannumeral`&&#3#1~#2{#3}}%
1756 \def\XINT_expr_rseq:_d #1{%
1757     \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1758     \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1759     \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1760     \xint_orthat{\XINT_expr_rseq:_goon #1}%}
1761 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1762     \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1763 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1764 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{,%}
1765 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%

```

11.46.4 \XINT_expr_rseq:_A etc...

n++ for rseq. With 1.2c dummy variables pick a single token.

```
1766 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1767 \def\XINT_expr_rseq:_D #1#2#3%
1768     {\expandafter\XINT_expr_rseq:_E\romannumeral`&&#3#1~#2{#3}}%
1769 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1770     \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1771     \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1772     \xint_orthat{\XINT_expr_rseq:_Goon #1}%}
1773 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1774     {,#1\expandafter\XINT_expr_rseq:_D
1775     \csname .=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1776     \romannumeral0\XINT_expr_lockit{#1}{#5}}%

```

```

1777 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1778     {\expandafter\XINT_expr_rseq:_D
1779         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1780 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{ }%
1781 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%

```

11.47 \iter()

11.47.1	\XINT_expr_iterx	342
11.47.2	\XINT_expr_itory	342
11.47.3	\XINT_expr_iter:_a etc...	343
11.47.4	\XINT_expr_iter:_A etc...	343

Prior to 1.2g, the iter keyword was what is now called iterr, analogous with rrseq. Somehow I forgot an iter functioning like rseq with the sole difference of printing only the last iteration. Both rseq and iter work well with list selectors, as @ refers to the whole comma separated sequence of the initial values. I have thus deliberately done the backwards incompatible renaming of iter to iterr, and the new iter.

```

1782 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval \xintthebareeval }%
1783 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1784 \def\XINT_iexpr_func_iter {\XINT_allexpr_iter \xintbareiieval \xintthebareiieval }%
1785 \def\XINT_allexpr_iter #1#2#3%
1786 {%
1787     \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
1788     #3\romannumeral`&&\XINT_expr_onliteral_seq_a { }%
1789 }%

```

11.47.1 \XINT_expr_iterx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1790 \def\XINT_expr_iterx #1#2#3#4#5%
1791 {%
1792     \expandafter\XINT_expr_itory\romannumeral0#1(#5)\relax #3#4#2%
1793 }%

```

11.47.2 \XINT_expr_itory

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```

1794 \def\XINT_expr_itory #1#2#3#4#5%
1795 {%
1796     \expandafter \XINT_expr_getop
1797     \csname .=%
1798     \expandafter\XINT_expr_iter:_aa
1799     \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1800 }%
1801 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1802     \expandafter\XINT_expr_iter:_a\fi #1}%

```

11.47.3 \XINT_expr_iter:_a etc...

```

1803 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%}
1804 \def\XINT_expr_iter:_b #1#2#3#4,{%
1805     \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1806     \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1807     \xint_orthat{\expandafter\XINT_expr_iter:_c}%
1808     \csname.=#3#4\endcsname {#1}{#2}%
1809 }%
1810 \def\XINT_expr_iter:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,%}
1811 \def\XINT_expr_iter:_end \csname.=^ \endcsname #1#2{\XINT_expr:_unlock #1}%
1812 \def\XINT_expr_iter:_c #1#2#3%
1813     {\expandafter\XINT_expr_iter:_d\romannumeral`&&@#3#1~#2{#3}}%
1814 \def\XINT_expr_iter:_d #1{%
1815     \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1816     \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1817     \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1818     \xint_orthat{\XINT_expr_iter:_goon #1}%
1819 \def\XINT_expr_iter:_goon #1!#2#3~#4#5%
1820     {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
1821 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1822 \def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
1823 \def\XINT_expr_iter:_break #1!#2#3~#4#5#6^,{#1}%

```

11.47.4 \XINT_expr_iter:_A etc...

n++ for iter. With 1.2c dummy variables pick a single token.

```

1824 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
1825 \def\XINT_expr_iter:_D #1#2#3%
1826     {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
1827 \def\XINT_expr_iter:_E #1{\if #1^ \xint_dothis\XINT_expr_iter:_Abort\fi
1828     \if #1? \xint_dothis\XINT_expr_iter:_Break\fi
1829     \if #1! \xint_dothis\XINT_expr_iter:_Omit\fi
1830     \xint_orthat{\XINT_expr_iter:_Goon #1}%
1831 \def\XINT_expr_iter:_Goon #1!#2#3~#4#5%
1832     {\expandafter\XINT_expr_iter:_D
1833     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1834     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1835 \def\XINT_expr_iter:_Omit #1!#2#3~%#4#5%
1836     {\expandafter\XINT_expr_iter:_D
1837     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1838 \def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
1839 \def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%

```

11.48 \rrseq()

11.48.1	\XINT_expr_rrseqx	344
11.48.2	\XINT_expr_rrseqy	344
11.48.3	\XINT_expr_rrseq:_a etc...	344
11.48.4	\XINT_expr_rrseq:_A etc...	345

When `func_rrseq` has its turn, initial segment has been scanned by `oparen`, the `;` mimicking the rôle of a closing parenthesis, and stopping further expansion.


```

1840 \def\XINT_expr_func_rrseq  {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1841 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1842 \def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1843 \def\XINT_allexpr_rrseq #1#2#3%
1844 {%
1845   \expandafter\XINT_expr_rrseq\expandafter #1\expandafter#2\expandafter
1846   #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1847 }%

```

11.48.1 \XINT_expr_rrseqx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1848 \def\XINT_expr_rrseqx #1#2#3#4#5%
1849 {%
1850   \expandafter\XINT_expr_rrseq\romannumeral0#1(#5)\expandafter\relax
1851   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1852     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1853   #3#4#2%
1854 }%

```

11.48.2 \XINT_expr_rrseqy

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

```

1855 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
1856 {%
1857   \expandafter \XINT_expr_getop
1858   \csname .=\XINT_expr_unlock #3%
1859   \expandafter\XINT_expr_rrseq:_aa
1860   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1861 }%
1862 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1863   \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

11.48.3 \XINT_expr_rrseq:_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1864 \catcode`? 3
1865 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1866 \def\XINT_expr_rrseq:_b #1#2#3#4,%{
1867   \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1868   \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1869   \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname .=#3#4\endcsname
1870   {#1}{#2}%
1871 }%
1872 \def\XINT_expr_rrseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1873 \def\XINT_expr_rrseq:_end \csname .=#1\endcsname #1#2{%}
1874 \def\XINT_expr_rrseq:_c #1#2#3%
1875   {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&#3#1~#2?{#3}}%

```



```

1876 \def\XINT_expr_rrseq:_d #1{%
1877   \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1878   \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1879   \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1880   \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1881 }%
1882 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1883   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1884 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1885 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{ }%
1886 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%

```

11.48.4 \XINT_expr_rrseq:_A etc...

n++ for rrseq. With 1.2C, the #1 in \XINT_expr_rrseq:_A is a single token.

```

1887 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1888 \def\XINT_expr_rrseq:_D #1#2#3%
1889   {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1890 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1891   {,#1\expandafter\XINT_expr_rrseq:_D
1892     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1893     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1894 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1895   {\expandafter\XINT_expr_rrseq:_D
1896     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1897 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5}%
1898 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1899 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1900   \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1901   \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1902   \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

11.49 \iterr()

11.49.1	\XINT_expr_iterrx	345
11.49.2	\XINT_expr_iterry	346
11.49.3	\XINT_expr_iterr:_a etc.	346
11.49.4	\XINT_expr_iterr:_A etc.	347

```

1903 \def\XINT_expr_func_iterr {\XINT_allexpr_iterr \xintbareeval \xintthebareeval }%
1904 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval \xintthebarefloateval }%
1905 \def\XINT_iexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval \xintthebareiieval }%
1906 \def\XINT_allexpr_iterr #1#2#3%
1907 {%
1908   \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
1909   #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
1910 }%

```

11.49.1 \XINT_expr_iterrx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1911 \def\XINT_expr_iterrx #1#2#3#4#5%
1912 {%
1913   \expandafter\XINT_expr_itterry\romannumeral0#1(#5)\expandafter\relax
1914   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1915     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1916   #3#4#2%
1917 }%

```

11.49.2 \XINT_expr_itterry

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

```

1918 \def\XINT_expr_itterry #1#2#3#4#5#6%
1919 {%
1920   \expandafter \XINT_expr_getop
1921   \csname .=%
1922   \expandafter\XINT_expr_iterr:_aa
1923   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1924 }%
1925 \def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
1926   \expandafter\XINT_expr_iterr:_a\fi #1}%

```

11.49.3 \XINT_expr_iterr:_a etc...

```

1927 \def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,%}
1928 \def\XINT_expr_iterr:_b #1#2#3#4,{%
1929   \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
1930   \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
1931   \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
1932   \csname .=#3#4\endcsname {#1}{#2}%
1933 }%
1934 \def\XINT_expr_iterr:_noop\csname .=#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,%}
1935 \def\XINT_expr_iterr:_end \csname .=#1\endcsname #1#2%
1936   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1937     {\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1938 \def\XINT_expr_iterr:_c #1#2#3%
1939   {\expandafter\XINT_expr_iterr:_d\romannumeral`&&#3#1~#2?{#3}}%
1940 \def\XINT_expr_iterr:_d #1{%
1941   \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
1942   \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
1943   \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
1944   \xint_orthat{\XINT_expr_iterr:_goon #1}%
1945 }%
1946 \def\XINT_expr_iterr:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
1947   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1948 \def\XINT_expr_iterr:_omit #1!#2#3~{\XINT_expr_iterr:_b }%
1949 \def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
1950   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1951     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1952 \def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
1953   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced

```

```

1954      {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1955 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

11.49.4 \XINT_expr_iterr:_A etc...

n++ for iterr. ? is of catcode 3 here.

1956 \def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
1957 \def\XINT_expr_iterr:_D #1#2#3%
1958   {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
1959 \def\XINT_expr_iterr:_Goon #1!#2#3~#4?#5%
1960   {\expandafter\XINT_expr_iterr:_D
1961     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1962     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}}%
1963 \def\XINT_expr_iterr:_Omit #1!#2#3~#4?#5%
1964   {\expandafter\XINT_expr_iterr:_D
1965     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1966 \def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
1967   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1968     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1969 \def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
1970   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1971     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1972 \def\XINT_expr_iterr:_E #1{\if #1^\xint_dothis\XINT_expr_iterr:_Abort\fi
1973     \if #1?\xint_dothis\XINT_expr_iterr:_Break\fi
1974     \if #1!\xint_dothis\XINT_expr_iterr:_Omit\fi
1975     \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
1976 \catcode`? 11

```

11.50 Macros handling csv lists for functions with multiple comma separated arguments in expressions

11.50.1	\xintANDof:csv	348
11.50.2	\xintORof:csv	348
11.50.3	\xintXORof:csv	348
11.50.4	Generic csv routine (\XINT_oncsv:_a)	348
11.50.5	\xintMaxof:csv, \xintiiMaxof:csv	349
11.50.6	\xintMinof:csv, \xintiiMinof:csv	349
11.50.7	\xintSum:csv, \xintiiSum:csv	349
11.50.8	\xintPrd:csv, \xintiiPrd:csv	349
11.50.9	\xintGCDof:csv, \xintLCMof:csv	349
11.50.10	\xintiiGCDof:csv, \xintiiLCMof:csv	350
11.50.11	\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits	350
11.50.12	\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv	350
11.50.13	\XINTinFloatSum:csv, \XINTinFloatPrd:csv	350

These macros are used inside \csname...\endcsname. These things are not initiated by a \romannumeral in general, but in some cases they are, especially when involved in an \xintNewExpr. They will then be protected against expansion and expand only later in contexts governed by an initial \romannumeral-`0. There each new item may need to be expanded, which would not be the case in the use for the _func_ things.

1.2g adds (to be continued)

11.50.1 \xintANDof:csv

1.09a. For use by \xintexpr inside \csname. 1.1, je remplace ifTrueAelseB par iiNotZero pour des raisons d'optimisations.

```
1977 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,,^}%
1978 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1979         \else\expandafter\XINT_andof:_c\fi #1}%
1980 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1981 \def\XINT_andof:_no #1^{0}%
1982 \def\XINT_andof:_e #1^{1}% works with empty list
```

11.50.2 \xintORof:csv

1.09a. For use by \xintexpr.

```
1983 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
1984 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1985         \else\expandafter\XINT_orof:_c\fi #1}%
1986 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1987 \def\XINT_orof:_yes #1^{1}%
1988 \def\XINT_orof:_e #1^{0}% works with empty list
```

11.50.3 \xintXORof:csv

1.09a. For use by \xintexpr (inside a \csname..\endcsname).

```
1989 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
1990 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1991 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
1992         \else\expandafter\XINT_xorof:_c\fi #1}%
1993 \def\XINT_xorof:_c #1,#2%
1994         {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1995                 \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1996                 {\XINT_xorof:_a #2}%
1997         }%
1998 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)
```

11.50.4 Generic csv routine (\XINT_oncsv:_a)

1.1. generic routine. up to the loss of some efficiency, especially for Sum:csv and Prod:csv, where \XINTinFloat will be done twice for each argument.

```
1999 \def\XINT_oncsv:_empty #1,^,#2{#2}%
2000 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
2001 \def\XINT_oncsv:_a #1#2#3%
2002     {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
2003 \def\XINT_oncsv:_b #1#2#3,%
2004     {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
2005 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}{#2#3}%
2006 \def\XINT_oncsv:_d #1%
2007     {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
2008 \def\XINT_oncsv:_e #1,#2#3#4%
2009     {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%

```

11.50.5 \xintMaxof:csv, \xintiiMaxof:csv

1.09i. Rewritten for 1.1. Compatible avec liste vide donnant valeur par défaut. Pas compatible avec items manquants. ah je m'aperçois au dernier moment que je n'ai pas en effet de \xintiiMax. Je devrais le rajouter. En tout cas ici c'est uniquement pour xintiiexpr, dans il faut bien sûr ne pas faire de xintNum, donc il faut un iimax.

```
2010 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
2011 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
2012 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimax
2013 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

11.50.6 \xintMinof:csv, \xintiiMinof:csv

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

```
2014 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
2015 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
2016 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimin
2017 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

11.50.7 \xintSum:csv, \xintiiSum:csv

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

```
2018 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
2019 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
2020 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiaadd
2021 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

11.50.8 \xintPrd:csv, \xintiiPrd:csv

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

```
2022 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
2023 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1/1[0]}}%
2024 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
2025 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1}%
```

11.50.9 \xintGCDof:csv, \xintLCMof:csv

1.09a. Rewritten for 1.1. For use by \xintexpr. Expansion réinstaurée pour besoins de xintNewExpr de version 1.1

```
2026 \def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
2027 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1}%
2028 \def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
2029 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

11.50.10 \xintiigCDof:csv, \xintiilCMof:csv

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

```
2030 \def\xintiigCDof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiigcd
2031      \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
2032 \def\xintiilCMof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiilcm
2033      \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

11.50.11 \XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits

for \xintNewExpr matters, mainly.

```
2034 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]]}%
2035 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]]}%
2036 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]]}%
```

11.50.12 \XINTinFloatMaxof:csv, \XINTinFloatMinof:csv

1.09a. Rewritten for 1.1. For use by \xintfloatexpr. Name changed in 1.09h

```
2037 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmax
2038      \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
2039 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmin
2040      \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
```

11.50.13 \XINTinFloatSum:csv, \XINTinFloatPrd:csv

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

```
2041 \def\XINTinFloatSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatadd
2042      \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
2043 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatmul
2044      \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{1[0]}}}%
```

11.51 Auxiliary wrappers for function macros

```
2045 \def\XINT:expr:one:and:opt #1,#2,#3!#4#5%
2046 {%
2047   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2048     \expandafter\xint_secondoftwo\fi
2049   {#4}{#5[\xintNum{#2}]}{#1}%
2050 }%
2051 \def\XINT:expr:tacitzeroifonearg #1,#2,#3!#4#5%
2052 {%
2053   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2054     \expandafter\xint_secondoftwo\fi
2055   {#4{0}}{#5[\xintNum{#2}]}{#1}%
2056 }%
2057 \def\XINT:iiexpr:tacitzeroifonearg #1,#2,#3!#4%
2058 {%
2059   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2060     \expandafter\xint_secondoftwo\fi
2061   {#4{0}}{#4{#2}}{#1}%
```

```

2062 }%
2063 \def\XINT:expr:totwo #1#2{#1,#2}%
2064 \def\XINT:expr:two:to:two #1,#2,!#3%
2065 {%
2066     \expandafter\XINT:expr:totwo\romannumeral`&&@%
2067     #3{#1}{#2}%
2068 }%
2069 \def\XINT:expr:two:to:one #1,#2,!#3%
2070 {%
2071     #3{#1}{#2}%
2072 }%
2073 \def\XINT:flexpr:two:to:one #1,#2,!#3%
2074 {%
2075     #3{#1}{#2}%
2076 }%
2077 \let\XINT:flexpr:two:to:two\XINT:flexpr:two:to:one

```

11.52 The num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), sqrt(), sqrtr(), float(), round(), trunc(), mod(), quo(), rem(), divmod(), gcd(), lcm(), max(), min(), `+`(), `*`(), ?(), !(), not(), all(), any(), xor(), if(), ifsgn(), ifint(), ifone(), even(), odd(), first(), last(), len(), reversed(), factorial(), binomial() and randrange() functions

```

2078 \def\XINT_expr_func_num #1#2#3%
2079 {%
2080     \expandafter #1\expandafter #2\csname.=%
2081     \XINT:NEhook:one\xintNum{\XINT_expr_unlock #3}\endcsname
2082 }%
2083 \let\XINT_flexpr_func_num\XINT_expr_func_num
2084 \let\XINT_iiexpr_func_num\XINT_expr_func_num
2085 \def\XINT_expr_func_reduce #1#2#3%
2086 {%
2087     \expandafter #1\expandafter #2\csname.=%
2088     \XINT:NEhook:one\xintIrr{\XINT_expr_unlock #3}[0]\endcsname
2089 }%
2090 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
2091 \def\XINT_expr_func_preduce #1#2#3%
2092 {%
2093     \expandafter #1\expandafter #2\csname.=%
2094     \XINT:NEhook:one\xintPIrr{\XINT_expr_unlock #3}\endcsname
2095 }%
2096 \let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
2097 \def\XINT_expr_func_abs #1#2#3%
2098 {%
2099     \expandafter #1\expandafter #2\csname.=%
2100     \XINT:NEhook:one\xintAbs{\XINT_expr_unlock #3}\endcsname
2101 }%
2102 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
2103 \def\XINT_iiexpr_func_abs #1#2#3%
2104 {%
2105     \expandafter #1\expandafter #2\csname.=%
2106     \XINT:NEhook:one\xintiiAbs{\XINT_expr_unlock #3}\endcsname

```

```

2107 }%
2108 \def\XINT_expr_func_sgn #1#2#3%
2109 {%
2110     \expandafter #1\expandafter #2\csname.=%
2111     \XINT:NEhook:one\xintSgn{\XINT_expr_unlock #3}\endcsname
2112 }%
2113 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
2114 \def\XINT_iiexpr_func_sgn #1#2#3%
2115 {%
2116     \expandafter #1\expandafter #2\csname.=%
2117     \XINT:NEhook:one\xintiSgn{\XINT_expr_unlock #3}\endcsname
2118 }%
2119 \def\XINT_expr_func_frac #1#2#3%
2120 {%
2121     \expandafter #1\expandafter #2\csname.=%
2122     \XINT:NEhook:one\xintTFrac{\XINT_expr_unlock #3}\endcsname
2123 }%
2124 \def\XINT_flexpr_func_frac #1#2#3%
2125 {%
2126     \expandafter #1\expandafter #2\csname.=%
2127     \XINT:NEhook:one\XINTinFloatFracdigits{\XINT_expr_unlock #3}\endcsname
2128 }%

```

no \XINT_iiexpr_func_frac

```

2129 \def\XINT_expr_func_floor #1#2#3%
2130 {%
2131     \expandafter #1\expandafter #2\csname.=%
2132     \XINT:NEhook:one\xintFloor{\XINT_expr_unlock #3}\endcsname
2133 }%
2134 \let\XINT_flexpr_func_floor\XINT_expr_func_floor

```

The floor and ceil functions in \xintiexpr require protect(a/b) or, better, \qfrac(a/b); else the / will be executed first and do an integer rounded division.

```

2135 \def\XINT_iiexpr_func_floor #1#2#3%
2136 {%
2137     \expandafter #1\expandafter #2\csname.=%
2138     \XINT:NEhook:one\xintiFloor{\XINT_expr_unlock #3}\endcsname
2139 }%
2140 \def\XINT_expr_func_ceil #1#2#3%
2141 {%
2142     \expandafter #1\expandafter #2\csname.=%
2143     \XINT:NEhook:one\xintCeil{\XINT_expr_unlock #3}\endcsname
2144 }%
2145 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2146 \def\XINT_iiexpr_func_ceil #1#2#3%
2147 {%
2148     \expandafter #1\expandafter #2\csname.=%
2149     \XINT:NEhook:one\xintiCeil{\XINT_expr_unlock #3}\endcsname
2150 }%
2151 \def\XINT_expr_func_sqr #1#2#3%
2152 {%
2153     \expandafter #1\expandafter #2\csname.=%
2154     \XINT:NEhook:one\xintSqr{\XINT_expr_unlock #3}\endcsname

```



```

2155 }%
2156 \def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}% revoir après
2157 \def\XINT_flexpr_func_sqr #1#2#3%
2158 {%
2159     \expandafter #1\expandafter #2\csname.=%
2160     \XINT:NEhook:one\XINTinFloatSqr{\XINT_expr_unlock #3}\endcsname
2161 }%
2162 \def\XINT_iiexpr_func_sqr #1#2#3%
2163 {%
2164     \expandafter #1\expandafter #2\csname.=%
2165     \XINT:NEhook:one\xintiiSqr{\XINT_expr_unlock #3}\endcsname
2166 }%
2167 \def\XINT_expr_func_? #1#2#3%
2168 {%
2169     \expandafter #1\expandafter #2\csname.=%
2170     \XINT:NEhook:one\xintiiIsNotZero{\XINT_expr_unlock #3}\endcsname
2171 }%
2172 \let\XINT_flexpr_func_? \XINT_expr_func_?
2173 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2174 \def\XINT_expr_func_! #1#2#3%
2175 {%
2176     \expandafter #1\expandafter #2\csname.=%
2177     \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2178 }%
2179 \let\XINT_flexpr_func_! \XINT_expr_func_!
2180 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2181 \def\XINT_expr_func_not #1#2#3%
2182 {%
2183     \expandafter #1\expandafter #2\csname.=%
2184     \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2185 }%
2186 \let\XINT_flexpr_func_not \XINT_expr_func_not
2187 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2188 \def\XINT_expr_func_odd #1#2#3%
2189 {%
2190     \expandafter #1\expandafter #2\csname.=%
2191     \XINT:NEhook:one\xintOdd{\XINT_expr_unlock #3}\endcsname
2192 }%
2193 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2194 \def\XINT_iiexpr_func_odd #1#2#3%
2195 {%
2196     \expandafter #1\expandafter #2\csname.=%
2197     \XINT:NEhook:one\xintiiOdd{\XINT_expr_unlock #3}\endcsname
2198 }%
2199 \def\XINT_expr_func_even #1#2#3%
2200 {%
2201     \expandafter #1\expandafter #2\csname.=%
2202     \XINT:NEhook:one\xintEven{\XINT_expr_unlock #3}\endcsname
2203 }%
2204 \let\XINT_flexpr_func_even\XINT_expr_func_even
2205 \def\XINT_iiexpr_func_even #1#2#3%
2206 {%

```

```

2207 \expandafter #1\expandafter #2\csname.=%
2208 \XINT:NEhook:one\xintiiEven{\XINT_expr_unlock #3}\endcsname
2209 }%
2210 % REVOIR nuple
2211 \def\XINT_expr_func_nuple #1#2#3%
2212 {\expandafter #1\expandafter #2\csname.\XINT_expr_unlock #3\endcsname }%
2213 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2214 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
2215 \def\XINT_expr_func_factorial #1#2#3%
2216 {%
2217 \expandafter #1\expandafter #2\csname.=%
2218 \expandafter\XINT:expr:one:and:opt
2219 \romannumeral`&&\XINT_expr_unlock#3,,!\xintFac\XINTinFloatFac
2220 \endcsname
2221 }%
2222 \def\XINT_flexpr_func_factorial #1#2#3%
2223 {%
2224 \expandafter #1\expandafter #2\csname.=%
2225 \expandafter\XINT:expr:one:and:opt
2226 \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatFacdigits\XINTinFloatFac
2227 \endcsname
2228 }%
2229 \def\XINT_iiexpr_func_factorial #1#2#3%
2230 {%
2231 \expandafter #1\expandafter #2\csname.=%
2232 \XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #3}\endcsname
2233 }%
2234 \def\XINT_expr_func_sqrt #1#2#3%
2235 {%
2236 \expandafter #1\expandafter #2\csname.=%
2237 \expandafter\XINT:expr:one:and:opt
2238 \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatSqrtdigits\XINTinFloatSqrt
2239 \endcsname
2240 }%
2241 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2242 \def\XINT_iiexpr_func_sqrt #1#2#3%
2243 {%
2244 \expandafter #1\expandafter #2\csname.=%
2245 \XINT:NEhook:one\xintiiSqrt{\XINT_expr_unlock #3}\endcsname
2246 }%
2247 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2248 {%
2249 \expandafter #1\expandafter #2\csname.=%
2250 \XINT:NEhook:one\xintiiSqrtR{\XINT_expr_unlock #3}\endcsname
2251 }%
2252 \def\XINT_expr_func_round #1#2#3%
2253 {%
2254 \expandafter #1\expandafter #2\csname.=%
2255 \expandafter\XINT:expr:tacitzeroifonearg
2256 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound\xintRound
2257 \endcsname
2258 }%

```

```

2259 \let\XINT_flexpr_func_round\XINT_expr_func_round
2260 \def\XINT_iiexpr_func_round #1#2#3%
2261 {%
2262     \expandafter #1\expandafter #2\csname.=%
2263     \expandafter\XINT:iiexpr:tacitzeroifonearg
2264     \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound
2265     \endcsname
2266 }%
2267 \def\XINT_expr_func_trunc #1#2#3%
2268 {%
2269     \expandafter #1\expandafter #2\csname.=%
2270     \expandafter\XINT:expr:tacitzeroifonearg
2271     \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc\xintTrunc
2272     \endcsname
2273 }%
2274 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2275 \def\XINT_iiexpr_func_trunc #1#2#3%
2276 {%
2277     \expandafter #1\expandafter #2\csname.=%
2278     \expandafter\XINT:iiexpr:tacitzeroifonearg
2279     \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc
2280     \endcsname
2281 }%
2282 \def\XINT_expr_func_float #1#2#3%
2283 {%
2284     \expandafter #1\expandafter #2\csname.=%
2285     \expandafter\XINT:expr:one:and:opt
2286     \romannumeral`&&\XINT_expr_unlock #3,,!\XINTinFloatdigits\XINTinFloat
2287     \endcsname
2288 }%
2289 \let\XINT_flexpr_func_float\XINT_expr_func_float
2290 % \XINT_iiexpr_func_float not defined
2291 \def\XINT_expr_func_divmod #1#2#3%
2292 {%
2293     \expandafter #1\expandafter #2\csname.=%
2294     \expandafter\XINT:expr:two:to:two
2295     \romannumeral`&&\XINT_expr_unlock #3,! \xintDivMod
2296     \endcsname
2297 }%
2298 \def\XINT_flexpr_func_divmod #1#2#3%
2299 {%
2300     \expandafter #1\expandafter #2\csname.=%
2301     \expandafter\XINT:flexpr:two:to:two
2302     \romannumeral`&&\XINT_expr_unlock #3,! \XINTinFloatDivMod
2303     \endcsname
2304 }%
2305 \def\XINT_iiexpr_func_divmod #1#2#3%
2306 {%
2307     \expandafter #1\expandafter #2\csname.=%
2308     \expandafter\XINT:expr:two:to:two
2309     \romannumeral`&&\XINT_expr_unlock #3,! \xintiiDivMod
2310     \endcsname

```

```

2311 }%
2312 \def\XINT_expr_func_mod #1#2#3%
2313 {%
2314     \expandafter #1\expandafter #2\csname.=%
2315     \expandafter\XINT:expr:two:to:one
2316     \romannumeral`&&\XINT_expr_unlock #3,! \xintMod
2317     \endcsname
2318 }%
2319 \def\XINT_flexpr_func_mod #1#2#3%
2320 {%
2321     \expandafter #1\expandafter #2\csname.=%
2322     \expandafter\XINT:flexpr:two:to:one
2323     \romannumeral`&&\XINT_expr_unlock #3,! \XINTinFloatMod
2324     \endcsname
2325 }%
2326 \def\XINT_iiexpr_func_mod #1#2#3%
2327 {%
2328     \expandafter #1\expandafter #2\csname.=%
2329     \expandafter\XINT:expr:two:to:one
2330     \romannumeral`&&\XINT_expr_unlock #3,! \xintiMod
2331     \endcsname
2332 }%
2333 \def\XINT_expr_func_binomial #1#2#3%
2334 {%
2335     \expandafter #1\expandafter #2\csname.=%
2336     \expandafter\XINT:expr:two:to:one
2337     \romannumeral`&&\XINT_expr_unlock #3,! \xintBinomial
2338     \endcsname
2339 }%
2340 \def\XINT_flexpr_func_binomial #1#2#3%
2341 {%
2342     \expandafter #1\expandafter #2\csname.=%
2343     \expandafter\XINT:flexpr:two:to:one
2344     \romannumeral`&&\XINT_expr_unlock #3,! \XINTinFloatBinomial
2345     \endcsname
2346 }%
2347 \def\XINT_iiexpr_func_binomial #1#2#3%
2348 {%
2349     \expandafter #1\expandafter #2\csname.=%
2350     \expandafter\XINT:expr:two:to:one
2351     \romannumeral`&&\XINT_expr_unlock #3,! \xintiiBinomial
2352     \endcsname
2353 }%
2354 \def\XINT_expr_func_pfactorial #1#2#3%
2355 {%
2356     \expandafter #1\expandafter #2\csname.=%
2357     \expandafter\XINT:expr:two:to:one
2358     \romannumeral`&&\XINT_expr_unlock #3,! \xintPFactorial
2359     \endcsname
2360 }%
2361 \def\XINT_flexpr_func_pfactorial #1#2#3%
2362 {%

```

```

2363 \expandafter #1\expandafter #2\csname.=%
2364 \expandafter\XINT:flexpr:two:to:one
2365 \romannumeral`&&\XINT_expr_unlock #3,! \XINTinFloatPFactorial
2366 \endcsname
2367 }%
2368 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2369 {%
2370 \expandafter #1\expandafter #2\csname.=%
2371 \expandafter\XINT:expr:two:to:one
2372 \romannumeral`&&\XINT_expr_unlock #3,! \xintiPFactorial
2373 \endcsname
2374 }%
2375 \def\XINT_expr_func_randrange #1#2#3%
2376 {%
2377 \expandafter #1\expandafter #2\csname.=%
2378 \expandafter\XINT:expr:randrange
2379 \romannumeral`&&\XINT_expr_unlock #3,!,%
2380 \endcsname
2381 }%
2382 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
2383 \def\XINT_iiexpr_func_randrange #1#2#3%
2384 {%
2385 \expandafter #1\expandafter #2\csname.=%
2386 \expandafter\XINT:iiexpr:randrange
2387 \romannumeral`&&\XINT_expr_unlock #3,!,%
2388 \endcsname
2389 }%
2390 \def\XINT:expr:randrange #1,#2,#3!%
2391 {%
2392 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2393 \expandafter\xint_secondoftwo\fi
2394 {\xintiiRandRange{\XINT:NEhook:one\xintNum{#1}}}%
2395 {\xintiiRandRangeAtoB{\XINT:NEhook:one\xintNum{#1}}%
2396 {\XINT:NEhook:one\xintNum{#2}}}%
2397 }%
2398 \def\XINT:iiexpr:randrange #1,#2,#3!%
2399 {%
2400 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2401 \expandafter\xint_secondoftwo\fi
2402 {\xintiiRandRange{#1}}{\xintiiRandRangeAtoB{#1}{#2}}%
2403 }%
2404 \def\XINT_expr_func_quo #1#2#3%
2405 {%
2406 \expandafter #1\expandafter #2\csname.=%
2407 \expandafter\XINT:expr:two:to:one
2408 \romannumeral`&&\XINT_expr_unlock #3,! \xintiQuo
2409 \endcsname
2410 }%
2411 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
2412 \def\XINT_iiexpr_func_quo #1#2#3%
2413 {%
2414 \expandafter #1\expandafter #2\csname.=%

```

```

2415 \expandafter\XINT:expr:two:to:one
2416 \romannumeral`&&\XINT_expr_unlock #3,! \xintiiQuo
2417 \endcsname
2418 }%
2419 \def\XINT_expr_func_rem #1#2#3%
2420 {%
2421 \expandafter #1\expandafter #2\csname.=%
2422 \expandafter\XINT:expr:two:to:one
2423 \romannumeral`&&\XINT_expr_unlock #3,! \xintiRem
2424 \endcsname
2425 }%
2426 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2427 \def\XINT_iiexpr_func_rem #1#2#3%
2428 {%
2429 \expandafter #1\expandafter #2\csname.=%
2430 \expandafter\XINT:expr:two:to:one
2431 \romannumeral`&&\XINT_expr_unlock #3,! \xintiiRem
2432 \endcsname
2433 }%
2434 \def\XINT_expr_func_gcd #1#2#3%
2435 {%
2436 \expandafter #1\expandafter #2\csname.=%
2437 \XINT:NEhook:csv\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname
2438 }%
2439 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
2440 \def\XINT_iiexpr_func_gcd #1#2#3%
2441 {%
2442 \expandafter #1\expandafter #2\csname.=%
2443 \XINT:NEhook:csv\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname
2444 }%
2445 \def\XINT_expr_func_lcm #1#2#3%
2446 {%
2447 \expandafter #1\expandafter #2\csname.=%
2448 \XINT:NEhook:csv\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname
2449 }%
2450 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2451 \def\XINT_iiexpr_func_lcm #1#2#3%
2452 {%
2453 \expandafter #1\expandafter #2\csname.=%
2454 \XINT:NEhook:csv\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname
2455 }%
2456 \def\XINT_expr_func_max #1#2#3%
2457 {%
2458 \expandafter #1\expandafter #2\csname.=%
2459 \XINT:NEhook:csv\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname
2460 }%
2461 \def\XINT_iiexpr_func_max #1#2#3%
2462 {%
2463 \expandafter #1\expandafter #2\csname.=%
2464 \XINT:NEhook:csv\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname
2465 }%
2466 \def\XINT_flexpr_func_max #1#2#3%

```

```

2467 {%
2468     \expandafter #1\expandafter #2\csname.=%
2469     \XINT:NEhook:csv\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname
2470 }%
2471 \def\XINT_expr_func_min #1#2#3%
2472 {%
2473     \expandafter #1\expandafter #2\csname.=%
2474     \XINT:NEhook:csv\xintMinof:csv{\XINT_expr_unlock #3}\endcsname
2475 }%
2476 \def\XINT_iiexpr_func_min #1#2#3%
2477 {%
2478     \expandafter #1\expandafter #2\csname.=%
2479     \XINT:NEhook:csv\xintiMinof:csv{\XINT_expr_unlock #3}\endcsname
2480 }%
2481 \def\XINT_flexpr_func_min #1#2#3%
2482 {%
2483     \expandafter #1\expandafter #2\csname.=%
2484     \XINT:NEhook:csv\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname
2485 }%
2486 \expandafter
2487 \def\csname XINT_expr_func_+\endcsname #1#2#3%
2488 {%
2489     \expandafter #1\expandafter #2\csname.=%
2490     \XINT:NEhook:csv\xintSum:csv{\XINT_expr_unlock #3}\endcsname
2491 }%
2492 \expandafter
2493 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2494 {%
2495     \expandafter #1\expandafter #2\csname.=%
2496     \XINT:NEhook:csv\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname
2497 }%
2498 \expandafter
2499 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2500 {%
2501     \expandafter #1\expandafter #2\csname.=%
2502     \XINT:NEhook:csv\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname
2503 }%
2504 \expandafter
2505 \def\csname XINT_expr_func_*\endcsname #1#2#3%
2506 {%
2507     \expandafter #1\expandafter #2\csname.=%
2508     \XINT:NEhook:csv\xintPrd:csv{\XINT_expr_unlock #3}\endcsname
2509 }%
2510 \expandafter
2511 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2512 {%
2513     \expandafter #1\expandafter #2\csname.=%
2514     \XINT:NEhook:csv\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname
2515 }%
2516 \expandafter
2517 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2518 {%

```

```

2519 \expandafter #1\expandafter #2\csname.=%
2520 \XINT:NEhook:csv\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname
2521 }%
2522 \def\XINT_expr_func_all #1#2#3%
2523 {%
2524 \expandafter #1\expandafter #2\csname.=%
2525 \XINT:NEhook:csv\xintANDof:csv{\XINT_expr_unlock #3}\endcsname
2526 }%
2527 \let\XINT_flexpr_func_all\XINT_expr_func_all
2528 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2529 \def\XINT_expr_func_any #1#2#3%
2530 {%
2531 \expandafter #1\expandafter #2\csname.=%
2532 \XINT:NEhook:csv\xintORof:csv{\XINT_expr_unlock #3}\endcsname
2533 }%
2534 \let\XINT_flexpr_func_any\XINT_expr_func_any
2535 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2536 \def\XINT_expr_func_xor #1#2#3%
2537 {%
2538 \expandafter #1\expandafter #2\csname.=%
2539 \XINT:NEhook:csv\xintXORof:csv{\XINT_expr_unlock #3}\endcsname
2540 }%
2541 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2542 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2543 \def\XINT_expr_func_len #1#2#3%
2544 {%
2545 \expandafter#1\expandafter#2\csname.=%
2546 \XINT:NEhook:csv\xintLength:f:csv{\XINT_expr_unlock#3}\endcsname
2547 }%
2548 \let\XINT_flexpr_func_len \XINT_expr_func_len
2549 \let\XINT_iiexpr_func_len \XINT_expr_func_len

```

1.2k has `\xintFirstItem:f:csv` for improved `\xintNewExpr` compatibility.

```

2550 \def\XINT_expr_func_first #1#2#3%
2551 {%
2552 \expandafter #1\expandafter #2\csname.=%
2553 \XINT:NEhook:csv\xintFirstItem:f:csv{\XINT_expr_unlock #3}\endcsname
2554 }%
2555 \let\XINT_flexpr_func_first\XINT_expr_func_first
2556 \let\XINT_iiexpr_func_first\XINT_expr_func_first

```

1.2k has `\xintLastItem:f:csv` for efficiency and improved `\xintNewExpr` compatibility.

```

2557 \def\XINT_expr_func_last #1#2#3%
2558 {%
2559 \expandafter #1\expandafter #2\csname.=%
2560 \XINT:NEhook:csv\xintLastItem:f:csv{\XINT_expr_unlock #3}\endcsname
2561 }%
2562 \let\XINT_flexpr_func_last\XINT_expr_func_last
2563 \let\XINT_iiexpr_func_last\XINT_expr_func_last

```

1.2c I hesitated but left the function "reversed" from 1.1 with this name, not "reverse". But the inner not public macro got renamed into `\xintReverse::csv`. 1.2g opts for the name `\xintReverse:f:csv`, and rewrites it for direct handling of csv lists. 2016/03/17.


```

2564 \def\XINT_expr_func_reversed #1#2#3%
2565 {%
2566     \expandafter #1\expandafter #2\csname.=%
2567     \XINT:NEhook:csv\xintReverse:f:csv{\XINT_expr_unlock #3}\endcsname
2568 }%
2569 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2570 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2571 \def\xintiifNotZero: #1,#2,#3,{\xintiifNotZero{#1}{#2}{#3}}%
2572 \def\XINT_expr_func_if #1#2#3%
2573 {%
2574     \expandafter #1\expandafter #2\csname.=%
2575     \expandafter\xintiifNotZero:%
2576     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2577 }%
2578 \let\XINT_flexpr_func_if\XINT_expr_func_if
2579 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2580 \def\xintifInt: #1,#2,#3,{\xintifInt{#1}{#2}{#3}}%
2581 \def\XINT_expr_func_ifint #1#2#3%
2582 {%
2583     \expandafter #1\expandafter #2\csname.=%
2584     \expandafter\xintifInt:%
2585     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2586 }%
2587 \let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint
2588 \def\xintifFloatInt: #1,#2,#3,{\xintifFloatInt{#1}{#2}{#3}}%
2589 \def\XINT_flexpr_func_ifint #1#2#3%
2590 {%
2591     \expandafter #1\expandafter #2\csname.=%
2592     \expandafter\xintifFloatInt:%
2593     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2594 }%
2595 \def\xintifOne: #1,#2,#3,{\xintifOne{#1}{#2}{#3}}%
2596 \def\XINT_expr_func_ifone #1#2#3%
2597 {%
2598     \expandafter #1\expandafter #2\csname.=%
2599     \expandafter\xintifOne:%
2600     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2601 }%
2602 \let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
2603 \def\xintiifOne: #1,#2,#3,{\xintiifOne{#1}{#2}{#3}}%
2604 \def\XINT_iiexpr_func_ifone #1#2#3%
2605 {%
2606     \expandafter #1\expandafter #2\csname.=%
2607     \expandafter\xintiifOne:%
2608     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2609 }%
2610 \def\xintiifSgn: #1,#2,#3,#4,{\xintiifSgn{#1}{#2}{#3}{#4}}%
2611 \def\XINT_expr_func_ifsgn #1#2#3%
2612 {%
2613     \expandafter #1\expandafter #2\csname.=%
2614     \expandafter\xintiifSgn:%
2615     \romannumeral`&&\XINT_expr_unlock #3,\endcsname

```

```
2616 }%
2617 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2618 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
```

11.53 f-expandable versions of the \xintSeqB::csv and alike routines, for \xintNewExpr

11.53.1	\xintSeqB:f:csv	362
11.53.2	\xintiiSeqB:f:csv	362
11.53.3	\XINTinFloatSeqB:f:csv	363

11.53.1 \xintSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```
2619 \def\xintSeqB:f:csv #1#2%
2620   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2621 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%
2622 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2623   \expandafter\xint_gobble_i\romannumeral`&&@%
2624   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2625   #1{#3}{#4}{#2}}%
2626 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2627 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2628   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2629 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2630   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2631 \def\XINT_seqb:f:csv_p #1#2%
2632 {%
2633   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2634   {#1}{#2}%
2635 }%
2636 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2637 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2638 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2639   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2640 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2641   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2642 \def\XINT_seqb:f:csv_n #1#2%
2643 {%
2644   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2645   {#1}{#2}%
2646 }%
2647 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2648 \def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%
```

11.53.2 \xintiiSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

2015/11/11. I correct a typo dating back to release 1.1 (2014/10/29): the macro name had a "b" rather than "B", hence was not functional (causing \xintNewIIExpr to fail on inputs such as #1..[1]..#2).

```

2649 \def\xintiiSeqB:f:csv #1#2%
2650   {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%
2651 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&@#2#1!}%
2652 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2653   \expandafter\xint_gobble_i\romannumeral`&&@%
2654   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2655     \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2656     #1{#3}{#4}{#2}}%
2657 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2658   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2659 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2660   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2661 \def\XINT_iiseqb:f:csv_p #1#2%
2662 {%
2663   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2664   \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2665 }%
2666 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2667 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2668 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2669   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2670 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2671   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2672 \def\XINT_iiseqb:f:csv_n #1#2%
2673 {%
2674   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2675   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2676 }%

```

11.53.3 \XINTinFloatSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide. This is all for \xintNewExpr.

```

2677 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2678   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2679 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2680 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2681   \expandafter\xint_gobble_i\romannumeral`&&@%
2682   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2683   #1{#3}{#4}{#2}}%
2684 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2685   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2686 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2687   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2688 \def\XINT_flseqb:f:csv_p #1#2%
2689 {%
2690   \xintifCmp {#1}{#2}%
2691   \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2692 }%
2693 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2694 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2695 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else

```

```

2696 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi}%
2697 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2698     {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2699 \def\XINT_flseqb:f:csv_n #1#2%
2700 {%
2701     \xintifCmp {#1}{#2}%
2702     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2703}%

```

11.54 User defined functions: \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc

1.2c (2015/11/12).

Note: it is possible to have same name assigned both to a variable and a function: things such as add(f(f), f=1..10) are possible.

1.2f (2016/03/08).

Comma separated expressions allowed (formerly this required using parenthesis \xintdeffunc foo(x,...):=(., ., .));

1.3c (2018/06/17).

Usage of \xintexprSafeCatcodes to be compatible with an active semi-colon at time of use; the colon was not a problem (see ##3) already.

```

2704 \def\XINT_tmpa #1#2#3#4%
2705 {%
2706     \def #1##1(##2)##3=##4;%
2707     \edef\XINT_expr_tmpa {##1}%
2708     \edef\XINT_expr_tmpa {\xint_zapspace_o \XINT_expr_tmpa}%
2709     \def\XINT_expr_tmppb {0}%
2710     \def\XINT_expr_tmppc {(##4)}%
2711     \xintFor ####1 in {##2} \do
2712         {\edef\XINT_expr_tmppb {\the\numexpr\XINT_expr_tmppb+\xint_c_i}%
2713          \edef\XINT_expr_tmppc {subs(\unexpanded\expandafter{\XINT_expr_tmppc},%
2714              ####1=#####\XINT_expr_tmppb)}}%
2715    }%
2716     \expandafter\XINT_expr_defuserfunc
2717     \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
2718     \expandafter{\XINT_expr_tmpa}{#2}%
2719     \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2720         [\XINT_expr_tmppb]{\XINT_expr_tmppc}%
2721     \ifxintverbose\xintMessage {xintexpr}{Info}
2722         {Function \XINT_expr_tmpa\space for \string\xint #4 parser
2723         associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
2724         with \ifxintglobaldefs global \fi meaning \expandafter\meaning
2725         \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
2726     \fi
2727     \xintexprRestoreCatcodes
2728}%
2729}%
2730 \def\xintdeffunc      {\xintexprSafeCatcodes\xintdeffunc_a}%
2731 \def\xintdefiifunc    {\xintexprSafeCatcodes\xintdefiifunc_a}%
2732 \def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
2733 \XINT_tmpa\xintdeffunc_a {expr} \XINT_NewFunc {expr}%

```

```

2734 \XINT_tmpa\xintdefiifunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
2735 \XINT_tmpa\xintdeffloatfunc_a{fexpr}\XINT_NewFloatFunc{floatexpr}%
2736 \def\XINT_expr_defuserfunc #1#2#3%
2737 {%
2738   \XINT_global
2739   \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2740     \csname .=\XINT:expr:userfunc{#3}{#2}\XINT_expr_unlock ##3}\endcsname
2741  }%
2742}%
2743 \def\XINT:expr:userfunc #1#2#3%
2744   {\csname XINT_#1_userfunc_#2\expandafter\endcsname
2745     \romannumeral0\xintcsvtolistnonstripped{#3}}%
2746 \catcode`~ 12
2747 \def\XINT:newexpr:userfunc #1#2#3%
2748   {\xintExpandArgs{XINT_#1_userfunc_#2}{\xintCSVtoListNonStripped{#3}}}%
2749 \catcode`~ 3

```

11.55 \xintNewFunction

1.2h (2016/11/20). Syntax is `\xintNewFunction{<name>}[nb of arguments]{expression with #1, #2,... as in \xintNewExpr}`. This defines a function for all three parsers but the expression parsing is delayed until function execution. Hence the expression admits all constructs, contrarily to `\xintNewExpr` or `\xintdefunc`.

```

2750 \def\XINT_expr_wrapit #1{\expandafter\XINT_expr_wrap\csname.=#1\endcsname}%
2751 \def\xintNewFunction #1#2[#3]#4%
2752 {%
2753   \edef\XINT_expr_tmpa {#1}%
2754   \edef\XINT_expr_tmpa {\xint_zapspaces_o \XINT_expr_tmpa}%
2755   \def\XINT_expr_tmppb ##1##2##3##4##5##6##7##8##9{#4}%
2756   \begingroup
2757     \ifcase #3\relax
2758       \toks0{}%
2759     \or \toks0{##1}%
2760     \or \toks0{##1##2}%
2761     \or \toks0{##1##2##3}%
2762     \or \toks0{##1##2##3##4}%
2763     \or \toks0{##1##2##3##4##5}%
2764     \or \toks0{##1##2##3##4##5##6}%
2765     \or \toks0{##1##2##3##4##5##6##7}%
2766     \or \toks0{##1##2##3##4##5##6##7##8}%
2767     \else \toks0{##1##2##3##4##5##6##7##8##9}%
2768     \fi
2769     \expandafter
2770   \endgroup\expandafter
2771   \XINT_global\expandafter
2772   \def\csname XINT_expr_macrofunc_\XINT_expr_tmpa\expandafter\endcsname
2773   \the\toks0\expandafter{\XINT_expr_tmppb
2774     {\XINT_expr_wrapit{##1}}{\XINT_expr_wrapit{##2}}{\XINT_expr_wrapit{##3}}%
2775     {\XINT_expr_wrapit{##4}}{\XINT_expr_wrapit{##5}}{\XINT_expr_wrapit{##6}}%
2776     {\XINT_expr_wrapit{##7}}{\XINT_expr_wrapit{##8}}{\XINT_expr_wrapit{##9}}}%
2777   \expandafter\XINT_expr_newfunction
2778   \csname XINT_expr_func_\XINT_expr_tmpa\expandafter\endcsname

```

```

2779 \expandafter{\XINT_expr_tmpa}{eval}\xintbareeval
2780 \expandafter\XINT_expr_newfunction
2781 \csname XINT_iiexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2782 \expandafter{\XINT_expr_tmpa}{iieval}\xintbareiieval
2783 \expandafter\XINT_expr_newfunction
2784 \csname XINT_flexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2785 \expandafter{\XINT_expr_tmpa}{floateval}\xintbarefloateval
2786 \ifxintverbose
2787 \xintMessage {xintexpr}{Info}
2788 {Function \XINT_expr_tmpa\space for the expression parsers is
2789 associated to \string\XINT_expr_macrofunc_\XINT_expr_tmpa\space
2790 with \ifxintglobaldefs global \fi meaning \expandafter\meaning
2791 \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname}%
2792 \fi
2793 }%
2794 \def\XINT_expr_newfunction #1#2#3#4%
2795 {%
2796 \XINT_global
2797 \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral0%
2798 \XINT:expr:macrofunc{#4}{#3}{#2}{\XINT_expr_unlock##3}}%
2799 }%
2800 \def\XINT:expr:macrofunc #1#2#3#4%
2801 {%
2802 #1\csname XINT_expr_macrofunc_#3\expandafter\endcsname
2803 \romannumeral0\xintcsvtolistnonstripped{#4}\relax
2804 }%
2805 \catcode~ 12
2806 \def\XINT:newexpr:macrofunc #1{%
2807 \def\XINT:newexpr:macrofunc ##1##2##3##4%
2808 {%
2809 \expandafter#1\csname.=~XINT:newexpr:macrofunc:a{##2}{##3}%
2810 {\xintCSVtoListNonStripped{##4}}\endcsname
2811 }%
2812 }\XINT:newexpr:macrofunc { }%
2813 \catcode~ 3
2814 \def\XINT:newexpr:macrofunc:a #1#2#3%
2815 {%
2816 \expandafter\XINT_expr_unlock\romannumeral0\csname xintbare#1\endcsname
2817 \csname XINT_expr_macrofunc_#2\endcsname#3\relax
2818 }%

```

11.56 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

11.56.1	\xintApply::csv and \xintApply:::csv	368
11.56.2	Mysterious stuff	368
11.56.3	\XINT_expr_redefinemacros	371
11.56.4	\XINT_expr_redefineprints	372
11.56.5	\xintNewExpr, ..., at last.	372
11.56.6	\ifxintsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes	374

There was an \xintNewExpr already in 1.07 from May 2013, which was modified in September 2013 to work with the # macro parameter character, and then refactored into a more powerful version in

June 2014 for 1.1 release of 2014/10/28. List handling causes special challenges, addressed by `\xintApply::csv`, `\xintApply:::csv`, ... next.

Comments finally added 2015/12/11 (with later edits):

The whole point is to expand completely macros when they have only numerical arguments and to inhibit this expansion if not. This is done in a recursive way: the catcode 12 `~` is used to register a macro name whose expansion must be inhibited. Any argument itself starting with such a `~` will force use of `~` for the macro which receives it.

In this context the catcode 12 `$` is used to signal a "virtual list" argument. It triggers insertion of `\xintApply::csv` or `\xintApply:::csv` for delayed handling later. This succeeds into handling inputs such as `[#1..[#2]..#3][#4:#5]...`

A final `\scantokens` converts the `"~"` prefixed names into real control sequences.

For this whole mechanism we need to have everything expressed using exclusively f-expandable macros. We avoid `\csname... \endcsname` like construct, but if absolutely needed perhaps we will do it ultimately.

For the iterating loops `seq`, `iter`, etc..., and dummy variables, we have no macros to our disposal to handle the case where the list of indices is not explicit. Moreover `omit`, `abort`, `break` can not work with non numerical data. Thus the whole mechanism is currently not applicable to them. It does work when the macro parameters (or variables for `\xintdeffunc`) do not intervene in the list of values to iterate over. But we can not delay expansion of dummy variables.

Comments added 2018/02/28:

At 1.3 of February 2018, there was important refactoring. Earlier, `\XINT_expr_redefinemacros` was a very big macro which made aliases of the dozens of macros (most from `xintfrac` and some defined especially by `xintexpr` for acting on csv lists primarily) involved in the expression rendering and then redefined them all to expand to their original selves only when applied to purely numeric arguments. At 1.3 only very few such re-definitions are made, as what is redefined are a limited number of core wrapper macros.

Only when the original macros have one or two arguments is it examined if they can expand immediately (this includes case of function having possibly only one, or possibly two arguments). For macros applying to three or more or an undefined number of arguments, we don't complicate matters into checking if expansion is possible, and we delay that expansion automatically (but `if()` and `ifsgn()` do check if first argument is numeric and expand to suitable branch in that case).

In particular any function defined by `\xintdeffunc` or `\xintNewFunction` (it is then basically only a macro abstraction) when used in new function definitions will never be expanded immediately, because the detection of whether they are applied to only numerical data has not yet been added. (this might be added in future).

Some aspects of the 1.3 refactoring have made recursive definition via `\xintdeffunc` possible (of course they always were via `\xintNewFunction` as the latter is but a wrapper of a standard TeX macro definition, where `\xintexpr` parsing is not at all involved).

A somewhat complicated layer (not modified at 1.3) is devoted to making possible the parsing of constructs such as `[#1..[#2]..#3][#4:#5]` or `[#1..#2]*#3` and it seems to work. At 1.3, even esoteric construct such as `[divmod(#1,#2)]*#3` is parsable by `\xintNewExpr`. (In `\xintNewFloatExpr`, don't forget `\empty` token so that square brackets are not mistaken for optional argument of `\xintthefloatexpr`; same for `\xintdeffloatfunc`.)

Side note: I wonder if I really had a good idea to define these list operations `[..]*foo` or `foo^[...]` which do not seem to occur in other languages with the meanings I used. And they caused me lots of efforts for support at `\xintNewExpr` level...

The catcode 12 dollar sign is used to signal when a macro can not be expanded but would produce a csv list. Furthermore some cases require f-expandable macros as the original code expanding in `\xintexpr` is in `\csname` context and did not need f-expandability.

As mentioned above, currently syntax with dummy variables can not go through where the values iterated over are not explicit; and `omit`, `abort`, `break` mechanisms are not parsable with non purely numerical data, in part because they are not implemented internally via pure f-expansion.

11.56.1 \xintApply::csv and \xintApply:::csv

```

2819 \def\xintApply::csv #1#2%
2820   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
2821 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},,%}
2822 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
2823 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2824   \else\expandafter\XINT_applyon::_d\fi #1}%
2825 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
2826 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}}%
2827 \def\XINT_applyon::_end #1,#2#3{\xint_seconddoftwo #3}%
2828 \def\xintApply:::csv #1#2#3%
2829   {\expandafter\XINT_applyon:::a\expandafter{\romannumeral`&&@#2}{#1}{#3}}%
2830 \def\XINT_applyon:::a #1#2#3{\XINT_applyon:::b {#2}{#3}{#1},,%}
2831 \def\XINT_applyon:::b #1#2#3#4,%
2832   {\expandafter\XINT_applyon:::c \romannumeral`&&@#4,{#1}{#2}{#3}}%
2833 \def\XINT_applyon:::c #1{\if #1,\expandafter\XINT_applyon:::end
2834   \else\expandafter\XINT_applyon:::d\fi #1}%
2835 \def\XINT_applyon:::d #1,#2#3%
2836   {\expandafter\XINT_applyon:::e\expandafter
2837     {\romannumeral`&&@#4\xintApply:::csv {#2}{#1}{#3}},{#2}{#3}}%
2838 \def\XINT_applyon:::e #1,#2#3#4{\XINT_applyon:::b {#2}{#3}{#4}, #1}}%
2839 \def\XINT_applyon:::end #1,#2#3#4{\xint_seconddoftwo #4}%

```

11.56.2 Mysterious stuff

~ and \$ of catcode 12 in what follows.

```

2840 \catcode~ 12
2841 \catcode$ 12 % $
2842 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2843 \def\XINT:NE:RApply::csv #1#2#3#4%
2844     {~xintApply::csv{~expandafter #2~xint_exchangetwo_keepbraces{#4}}{#3}}%
2845 \def\XINT:NE:LApply::csv #1#2#3{~xintApply::csv{#2{#3}}}%
2846 \def\XINT:NE:RLApply:::csv #1{~xintApply:::csv}%
2847 \def\XINT:NE:two#1{\XINT:NE:two_{#1}{\detokenize{#1}}}%
2848 \def\XINT:NE:two_#1#2#3#4%
2849     {\expandafter\XINT:NE:two_a\romannumeral`&&@#4!{#3}{#1}{#2}}%
2850 \def\XINT:NE:two_a#1#2!#3#4#5%
2851     {\expandafter\XINT:NE:two_b\romannumeral`&&@#3!#1{#4}{#5}{#1#2}}%
2852 \def\XINT:NE:two_b#1#2!#3#4#5{\XINT:NE:two_fork_dd#1#3{#4}{#5}{#1#2}}%
2853 \def\XINT:NE:two_fork_dd #1#2{%
2854     \xint_ddfork
2855     #1#2\XINT:NE:RLApply:::csv
2856     #1$\XINT:NE:RApply:::csv% $
2857     $#2\XINT:NE:LApply:::csv% $
2858     $$\{\XINT:NE:two_fork_nn #1#2}% $$
2859     \krof
2860 }%
2861 \def\XINT:NE:two_fork_nn #1#2#3#4{%
2862     \if #1##\xint_dothis{#4}\fi
2863     \if #1~\xint_dothis{#4}\fi
2864     \if #2##\xint_dothis{#4}\fi
2865     \if #2~\xint_dothis{#4}\fi

```



```

2866 \xint_orthat{#3}%
2867 }%
2868 \def\xint:NE:one#1#2{\expandafter\xint:NE:one_a\romannumeral`&&@#2!#1}%
2869 \def\xint:NE:one_a#1#2!#3{%
2870 \if ##1\xint_dothis {\detokenize{#3}}\fi
2871 \if ~#1\xint_dothis {\detokenize{#3}}\fi
2872 \if $#1\xint_dothis {\xintApply::csv{\detokenize{#3}}}\fi %$
2873 \xint_orthat #3{#1#2}%
2874 }%
2875 \def\xint:NE:oneopt#1[#2]#3%
2876 {\expandafter\xint:NE:oneopt_a\romannumeral`&&@#3!{#2}#1}%
2877 \def\xint:NE:oneopt_a#1#2!#3#4%
2878 {\expandafter\xint:NE:oneopt_b\romannumeral`&&@#3!#1#4{#1#2}}%
2879 \def\xint:NE:oneopt_b#1#2!#3#4%
2880 {\expandafter\xint:NE:oneopt_fork#1#3#4{#1#2}}%
2881 \def\xint:NE:oneopt_fork#1#2#3#4{%
2882 \if1\if###11\else\if~#11\else\if###21\else\if~#21\else0\fi\fi\fi\fi
2883 \xint_dothis {\detokenize{#3}[#4]}\fi
2884 \if $#2\xint_dothis {\xintApply::csv{\detokenize{#3}[#4]}}\fi %$
2885 \xint_orthat{#3[#4]}%
2886 }% pas complètement général, mais bon
2887 \def\xint:NE:csv #1{\detokenize{#1}}% radicalement fainéant
2888 \def\xint:newexpr:one:and:opt #1,#2,#3!#4#5%
2889 {%
2890 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2891 \expandafter\xint_secondoftwo\fi
2892 {\XINT:NE:one#4}{\XINT:NE:oneopt#5[\XINT:NE:one\xintNum{#2}]}{#1}%
2893 }%
2894 \def\xint:newexpr:tacitzeroifonearg #1,#2,#3!#4#5%
2895 {%
2896 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2897 \expandafter\xint_secondoftwo\fi
2898 {\XINT:NE:two#4{0}}{\XINT:NE:two#5{\XINT:NE:one\xintNum{#2}}}{#1}%
2899 }%
2900 \def\xint:newiexpr:tacitzeroifonearg #1,#2,#3!#4%
2901 {%
2902 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2903 \expandafter\xint_secondoftwo\fi
2904 {\XINT:NE:two#4{0}}{\XINT:NE:two#4{#2}}{#1}%
2905 }%
2906 \def\xint:newexpr:insertdollar~{$noexpand$}%
2907 \def\xint:newexpr:two:to:two #1,#2,!#3%
2908 {%
2909 \XINT:NE:two_
2910 {\expandafter\xint:expr:totwo\romannumeral`&&@#3}%
2911 {$noexpand$\expandafter~XINT:expr:totwo~romannumeral-`0\detokenize{#3}}%
2912 {#1}{#2}%
2913 }%
2914 \def\xint:newflexpr:two:to:two #1,#2,!#3%
2915 {%
2916 \XINT:NE:two_
2917 {#3}%

```

```

2918     {\expandafter\XINT:newexpr:insertdollar\detokenize{#3}}%
2919     {#1}{#2}%
2920 }%
2921 \def\XINT:newexpr:two:to:one #1,#2,!#3%
2922 {%
2923     \XINT:NE:two#3{#1}{#2}%
2924 }%
2925 \let\XINT:newflexpr:two:to:one\XINT:newexpr:two:to:one
2926 \def\xintiiifNotZeroNE:#1#2,#3,#4,%
2927 {%
2928     \if1\if###11\else\if~#11\else\if$#11\else0%$
2929         \fi\fi\fi
2930     \xint_dothis{~xintiiifNotZero}\fi
2931     \xint_orthat\xintiiifNotZero
2932     {#1#2}{#3}{#4}%
2933 }%
2934 \def\xintifIntNE:#1#2,#3,#4,%
2935 {%
2936     \if1\if###11\else\if~#11\else\if$#11\else0%$
2937         \fi\fi\fi
2938     \xint_dothis{~xintifInt}\fi
2939     \xint_orthat\xintifInt
2940     {#1#2}{#3}{#4}%
2941 }%
2942 \def\xintifFloatIntNE:#1#2,#3,#4,%
2943 {%
2944     \if1\if###11\else\if~#11\else\if$#11\else0%$
2945         \fi\fi\fi
2946     \xint_dothis{~xintifFloatInt}\fi
2947     \xint_orthat\xintifFloatInt
2948     {#1#2}{#3}{#4}%
2949 }%
2950 \def\xintiiifOneNE:#1#2,#3,#4,%
2951 {%
2952     \if1\if###11\else\if~#11\else\if$#11\else0%$
2953         \fi\fi\fi
2954     \xint_dothis{~xintiiifOne}\fi
2955     \xint_orthat\xintiiifOne
2956     {#1#2}{#3}{#4}%
2957 }%
2958 \def\xintifOneNE:#1#2,#3,#4,%
2959 {%
2960     \if1\if###11\else\if~#11\else\if$#11\else0%$
2961         \fi\fi\fi
2962     \xint_dothis{~xintifOne}\fi
2963     \xint_orthat\xintifOne
2964     {#1#2}{#3}{#4}%
2965 }%
2966 \def\xintiiifSgnNE:#1#2,#3,#4,#5,%
2967 {%
2968     \if1\if###11\else\if~#11\else\if$#11\else0%$
2969         \fi\fi\fi

```

```

2970 \xint_dothis{~xintiiifSgn}\fi
2971 \xint_orthat\xintiiifSgn
2972 {#1#2}{#3}{#4}{#5}%
2973 }%

```

11.56.3 \XINT_expr_redefinemacros

Completely refactored at 1.3.

```

2974 \def\xint_expr_redefinemacros {%
2975 \let\xint_NEhook:one\xint_NE:one
2976 \let\xint_NEhook:two\xint_NE:two
2977 \let\xint_NEhook:csv\xint_NE:csv
2978 \let\xint_expr:one:and:opt \xint_newexpr:one:and:opt
2979 \let\xint_expr:one:or:two:nums \xint_newexpr:one:or:two:nums
2980 \let\xint_iiexpr:one:or:two: \xint_newiiexpr:one:or:two:
2981 \let\xint_expr:tacitzeroifonearg \xint_newexpr:tacitzeroifonearg
2982 \let\xint_iiexpr:tacitzeroifonearg \xint_newiiexpr:tacitzeroifonearg
2983 \let\xint_expr:two:to:two \xint_newexpr:two:to:two
2984 \let\xint_expr:two:to:one \xint_newexpr:two:to:one
2985 \let\xint_flexpr:two:to:one \xint_newflexpr:two:to:one
2986 \let\xint_expr:two:to:one \xint_newexpr:two:to:one
2987 \let\xint_flexpr:two:to:two \xint_newflexpr:two:to:two
2988 \let\xint_flexpr:two:to:one \xint_newflexpr:two:to:one
2989 \let\xintiiifNotZero: \xintiiifNotZeroNE:
2990 \let\xintifInt: \xintifIntNE:
2991 \let\xintifFloatInt: \xintifFloatIntNE:
2992 \let\xintiiifOne: \xintiiifOneNE:
2993 \let\xintifOne: \xintifOneNE:
2994 \let\xintiiifSgn: \xintiiifSgnNE:
2995 \let\xintSeqNumeric::csv \xintSeq::csv
2996 \let\xintiiSeqNumeric::csv \xintiiSeq::csv
2997 \let\xintinFloatSeqNumeric::csv \xintinFloatSeq::csv
2998 \let\xintSeqBNumeric::csv \xintSeqB::csv
2999 \let\xintiiSeqBNumeric::csv \xintiiSeqB::csv
3000 \let\xintinFloatSeqBNumeric::csv\xintinFloatSeqB::csv
3001 \def\xintSeq::csv
3002 {\xint_NE:two_\xintSeqNumeric::csv{$noexpand\xintSeq::csv}}%
3003 \def\xintiiSeq::csv
3004 {\xint_NE:two_\xintiiSeqNumeric::csv{$noexpand\xintiiSeq::csv}}%
3005 \def\xintinFloatSeq::csv
3006 {\xint_NE:two_\xintinFloatSeqNumeric::csv{$noexpand\xintinFloatSeq::csv}}%
3007 \def\xintSeqB::csv
3008 {\xint_NE:two_\xintSeqBNumeric::csv{$noexpand\xintSeqB:f:csv}}%
3009 \def\xintiiSeqB::csv
3010 {\xint_NE:two_\xintiiSeqBNumeric::csv{$noexpand\xintiiSeqB:f:csv}}%
3011 \def\xintinFloatSeqB::csv
3012 {\xint_NE:two_\xintinFloatSeqBNumeric::csv{$noexpand\xintinFloatSeqB:f:csv}}%
3013 \def\xintListSel:x:csv {~xintListSel:f:csv}%
3014 \let\xint_expr:userfunc \xint_newexpr:userfunc
3015 \let\xint_expr:macrofunc\xint_newexpr:macrofunc
3016 \def\xintinRandomFloatSdigits{~xintinRandomFloatSdigits}%
3017 \def\xintinRandomFloatSixteen{~xintinRandomFloatSixteen}%

```

```
3018 \def\xintiiRandRange{~xintiiRandRange }%
3019 \def\xintiiRandRangeAtoB{~xintiiRandRangeAtoB }%
3020 }%
```

11.56.4 \XINT_expr_redefineprints

This is used by `\xintNewExpr` but not by `\xintdeffunc`.

```
3021 \def\XINT_expr_redefineprints
3022 {%
3023   \def\XINT_flexpr_noopt
3024     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
3025   \def\XINT_flexpr_withopt_b ##1##2%
3026     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
3027   \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!%
3028     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
3029     \XINTdigits{##2##3}{##4}}%
3030   \def\XINT_expr_print ##1%
3031     {\expandafter\xintSPRaw::csv\expandafter
3032       {\romannumeral`&&\XINT_expr_unlock ##1}}%
3033   \def\XINT_iexpr_print ##1%
3034     {\expandafter\xintCSV::csv\expandafter
3035       {\romannumeral`&&\XINT_expr_unlock ##1}}%
3036   \def\XINT_boolexpr_print ##1%
3037     {\expandafter\xintIsTrue::csv\expandafter
3038       {\romannumeral`&&\XINT_expr_unlock ##1}}%
3039   \def\xintCSV::csv {\xintCSV::csv }%
3040   \def\xintSPRaw::csv {\xintSPRaw::csv }%
3041   \def\xintPFloat::csv {\xintPFloat::csv }%
3042   \def\xintIsTrue::csv {\xintIsTrue::csv }%
3043   \def\xintRound::csv {\xintRound::csv }%
3044 }%
```

11.56.5 \xintNewExpr, ..., at last.

1.2c modifications to accomodate `\XINT_expr_deffunc_newexpr` etc..

1.2f adds token `\XINT_newexpr_clean` to be able to have a different `\XINT_newfunc_clean`.

```
3045 \def\xintNewExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3046                  \xinttheexpr\XINT_newexpr_clean}%
3047 \def\xintNewFloatExpr{\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3048                  \xintthefloatexpr\XINT_newexpr_clean}%
3049 \def\xintNewIExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3050                  \xinttheiexpr\XINT_newexpr_clean}%
3051 \def\xintNewIIExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3052                  \xinttheiexpr\XINT_newexpr_clean}%
3053 \def\xintNewBoolExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3054                  \xinttheboolexpr\XINT_newexpr_clean}%
3055 \def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&}%
```

1.2c for `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`.

At 1.3, NewFunc does not use a comma delimited pattern anymore.

```

3056 \def\XINT_NewFunc
3057 {\XINT_NewExpr{\xint_gobble_i\xintthebareeval\XINT_newfunc_clean}%
3058 \def\XINT_NewFloatFunc
3059 {\XINT_NewExpr{\xint_gobble_i\xintthebarefloateval\XINT_newfunc_clean}%
3060 \def\XINT_NewIIFunc
3061 {\XINT_NewExpr{\xint_gobble_i\xintthebareiieval\XINT_newfunc_clean}%
3062 \def\XINT_newfunc_clean #1>{%

```

1.2c adds optional logging. For this needed to pass to `_NewExpr_a` the macro name as parameter.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

Modified at 1.3c so that `\XINT_NewFunc` et. al. do not execute the `\xintexprSafeCatcodes`, as it is now already done earlier by `\xintdeffunc`: and as already #2 was either `\xint_firstofone` (for `\xintNewExpr` et. al.) or `\xint_gobble_i` (for `\XINT_NewFunc` et. al.) we can use that #2. This is only to avoid doing twice the catcodes, as anyhow there is an `\endgroup` coming later, so external `\xintexprRestoreCatcodes` would not have been compromised.

```

3063 \def\XINT_NewExpr #1#2#3#4#5#6[#7]%
3064 {%
3065 \begingroup
3066 \ifcase #7\relax
3067 \toks0 {\endgroup\XINT_global\def#5}%
3068 \or \toks0 {\endgroup\XINT_global\def#5##1}%
3069 \or \toks0 {\endgroup\XINT_global\def#5##1##2}%
3070 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3}%
3071 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4}%
3072 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5}%
3073 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6}%
3074 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7}%
3075 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7##8}%
3076 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7##8##9}%
3077 \fi
3078 #2\xintexprSafeCatcodes
3079 \XINT_expr_redefinemacros
3080 #1%
3081 \XINT_NewExpr_a #2#3#4#5%
3082 }%

```

1.2d's `\xintNewExpr` makes a local definition. In earlier releases, the definition was global.

`\the\toks0` inserts the `\endgroup`, but this will happen after `\XINT_tmpa` has already been expanded...

The 1 is `\xint_firstofone` for `\xintNewExpr`, `\xint_gobble_i` for `\xintdeffunc`.

```

3083 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
3084 \def\XINT_NewExpr_a %1%2%3%4%5@
3085 {@
3086 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
3087 \def~{$\noexpand$}@
3088 \catcode` : 11 \catcode`_ 11
3089 \catcode`# 12 \catcode`~ 13 \escapechar 126
3090 \endlinechar -1 \everyeof {\noexpand }@
3091 \edef\XINT_tmppb
3092 {\scantokens\expandafter{\romannumeral`&&\expandafter
3093 %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
3094 }@

```

```

3095 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
3096 \edef\XINT_tmpa %1%2%3%4%5%6%7%8%9@
3097 {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
3098 \the\toks0\expandafter
3099 {\XINT_tmpa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
3100 %1{\ifxintverbose
3101 \xintMessage{xintexpr}{Info}@
3102 {\string%4\space now with @
3103 \ifxintglobaldefs global \fi meaning \meaning%4}@
3104 \fi}@
3105 }@
3106 \catcode`% 14

```

11.56.6 \ifxintsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes

1.3c (2018/06/17).

Added \ifxintsafecatcodes to allow nesting

```

3107 \newif\ifxintexprsafecatcodes
3108 \let\xintexprRestoreCatcodes\empty
3109 \def\xintexprSafeCatcodes
3110 {%
3111 \unless\ifxintexprsafecatcodes
3112 \edef\xintexprRestoreCatcodes {%
3113 \catcode59=\the\catcode59 % ;
3114 \catcode34=\the\catcode34 % "
3115 \catcode63=\the\catcode63 % ?
3116 \catcode124=\the\catcode124 % |
3117 \catcode38=\the\catcode38 % &
3118 \catcode33=\the\catcode33 % !
3119 \catcode93=\the\catcode93 % ]
3120 \catcode91=\the\catcode91 % [
3121 \catcode94=\the\catcode94 % ^
3122 \catcode95=\the\catcode95 % _
3123 \catcode47=\the\catcode47 % /
3124 \catcode41=\the\catcode41 % )
3125 \catcode40=\the\catcode40 % (
3126 \catcode42=\the\catcode42 % *
3127 \catcode43=\the\catcode43 % +
3128 \catcode62=\the\catcode62 % >
3129 \catcode60=\the\catcode60 % <
3130 \catcode58=\the\catcode58 % :
3131 \catcode46=\the\catcode46 % .
3132 \catcode45=\the\catcode45 % -
3133 \catcode44=\the\catcode44 % ,
3134 \catcode61=\the\catcode61 % =
3135 \catcode96=\the\catcode96 % `
3136 \catcode32=\the\catcode32\relax % space
3137 \noexpand\xintexprsafecatcodesfalse
3138 }%
3139 \fi
3140 \xintexprsafecatcodestrue
3141 \catcode59=12 % ;

```

```

3142      \catcode34=12 % "
3143      \catcode63=12 % ?
3144      \catcode124=12 % |
3145      \catcode38=4  % &
3146      \catcode33=12 % !
3147      \catcode93=12 % ]
3148      \catcode91=12 % [
3149      \catcode94=7  % ^
3150      \catcode95=8  % _
3151      \catcode47=12 % /
3152      \catcode41=12 % )
3153      \catcode40=12 % (
3154      \catcode42=12 % *
3155      \catcode43=12 % +
3156      \catcode62=12 % >
3157      \catcode60=12 % <
3158      \catcode58=12 % :
3159      \catcode46=12 % .
3160      \catcode45=12 % -
3161      \catcode44=12 % ,
3162      \catcode61=12 % =
3163      \catcode96=12 % `
3164      \catcode32=10 % space
3165 }%
3166 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
3167 \XINT_restorecatcodes_endinput%

```

12 Per package indices of control sequences

12.1 Index of [xintkernel](#)

<code>_!_/</code>	18		
C			
<code>\catcode</code>	5--7, 18		
<code>\char</code>	10		
<code>\chardef</code>	8		
<code>\csname</code>	5, 6, 8, 14--17		
E			
<code>\edef</code>	6, 7, 15--17		
<code>\endcsname</code>	5, 6, 8, 14--17		
<code>\endinput</code>	5--7		
<code>\endlinechar</code>	5--7		
F			
<code>\fdef</code>	11		
I			
<code>\if</code>	17, 18		
<code>\ifdefined</code>	8, 9, 11, 17		
<code>\ifxintglobaldefs</code>	18		
<code>\ifxintverbose</code>	18		
<code>\immediate</code>	5, 8, 9, 18		
<code>\inputlineno</code>	18		
K			
<code>\krof</code>	10, 13, 14		
L			
<code>\lccode</code>	18		
M			
<code>\mathchardef</code>	8		
N			
<code>\newif</code>	18		
<code>\noexpand</code>	6, 7, 15--17		
<code>\numexpr</code>	5, 12--15, 17, 18		
O			
<code>\odef</code>	11		
<code>\oodef</code>	11		
P			
<code>\pdfuniformdeviate</code>	8		
<code>\PrepareCatcodes</code>	6, 7		
<code>\ProvidesPackage</code>	8		
R			
<code>\R</code>	9		
<code>\romannumeral</code>	7, 11--13		
U			
<code>\uniformdeviate</code>	8		
W			
<code>\W</code>	9, 10		
<code>\write</code>	5, 8, 9, 18		
X			
<code>\xint:</code>	10--13, 17, 18		
<code>\xint_afterfi</code>	10		
<code>\xint_bracedstopper</code>	10		
<code>\xint_Bye</code>	10		
<code>\xint_bye</code>	10--13		
<code>\xint_bye_thenstop</code>	10		
<code>\xint_c_</code>	8, 12--15		
<code>\xint_c_i</code>	8, 12, 13, 15		
<code>\xint_c_ii</code>	8, 12, 13		
<code>\xint_c_ii^v</code>	8		
<code>\xint_c_ii^vi</code>	8		
<code>\xint_c_ii^vii</code>	8, 17		
<code>\xint_c_ii^viii</code>	8		
<code>\xint_c_ii^xii</code>	8		
<code>\xint_c_ii^xiv</code>	8, 17		
<code>\xint_c_ii^xxi</code>	8, 17		
<code>\xint_c_iii</code>	8, 12, 13		
<code>\xint_c_iv</code>	8, 12, 13		
<code>\xint_c_ix</code>	8, 12, 15		
<code>\xint_c_v</code>	8, 12, 13, 15		
<code>\xint_c_vi</code>	8, 12, 13		
<code>\xint_c_vii</code>	8, 12, 13		
<code>\xint_c_viii</code>	8, 12, 13		
<code>\xint_c_x</code>	8		
<code>\xint_c_x^iv</code>	8		
<code>\xint_c_xii</code>	8		
<code>\xint_c_xiv</code>	8		
<code>\xint_c_xvi</code>	8		
<code>\xint_c_xviii</code>	8		
<code>\xint_c_xxii</code>	8		
<code>\xint_dothis</code>	10		
<code>\xint_exchangetwo_keepbraces</code>	9		
<code>\XINT_expandableerror</code>	18		
<code>\XINT_expandableerror_continue</code>	18		
<code>\xint_firstofone</code>	9, 10, 18		
<code>\xint_firstofone_thenstop</code>	9		
<code>\xint_firstoftwo</code>	9, 17		
<code>\xint_firstoftwo_thenstop</code>	9		
<code>\XINT_global</code>	18		
<code>\xint_gob_andstop_</code>	9		
<code>\xint_gob_andstop_i</code>	9		
<code>\xint_gob_andstop_ii</code>	9		
<code>\xint_gob_andstop_iii</code>	9		
<code>\xint_gob_andstop_iv</code>	9		

\xint_gob_andstop_v	9	\XINT_rep_neg	14
\xint_gob_andstop_vi	9	\XINT_replicate	14
\xint_gob_andstop_vii	9	\XINT_restorecatcodes	6, 7
\xint_gob_andstop_viii	9	\XINT_restorecatcodes_endinput	6, 7, 18
\xint_gob_til_dot	10	\XINT_rord_cleanup	12
\xint_gob_til_eightzeroes	10	\XINT_rord_main	11, 12
\xint_gob_til_exclam	10	\xint_secondoftwo	9, 17
\xint_gob_til_G	10	\xint_secondoftwo_thenstop	9
\xint_gob_til_minus	10	\XINT_setcatcodes	7
\xint_gob_til_one	10	\xint_texuniformdeviate	8, 17
\xint_gob_til_R	9	\XINT_tmpa	9
\xint_gob_til_sc	10	\XINT_tmpb	9
\xint_gob_til_W	9	\xint_UDonezerofork	10
\xint_gob_til_xint:	10, 12, 13	\xint_UDsignfork	10, 13, 14
\xint_gob_til_Z	10	\xint_UDsignsfork	10
\xint_gob_til_zero	10, 15	\xint_UDwfork	10
\xint_gob_til_zeros_iii	10	\xint_UDXINTWfork	10
\xint_gob_til_zeros_iv	10	\xint_UDzerofork	10
\XINT_gobble	15	\xint_UDzerominusfork	10
\xint_gobble_	9	\xint_UDzerosfork	10
\XINT_gobble_a	15	\xint_undefined	8
\XINT_gobble_b	15	\XINT_uniformdeviate	17
\XINT_gobble_c	15	\XINT_uniformdeviate_a	17
\XINT_gobble_d	15	\XINT_uniformdeviate_b	17, 18
\xint_gobble_i	9, 11	\XINT_uniformdeviate_neg	17
\xint_gobble_ii	9	\XINT_uniformdeviate_sgnfork	17
\xint_gobble_iii	9	\XINT_W	10
\xint_gobble_iv	9	\xint_zapspaces	11
\xint_gobble_thenstop	9	\xint_zapspaces_o	11
\xint_gobble_v	9	\xintdothis	10
\xint_gobble_vi	9	\xintfdef	11
\xint_gobble_vii	9	\xintgobble	15
\xint_gobble_viii	9	\xintLastItem	12
\XINT_last_loop	12, 13	\xintlastitem	12
\XINT_last_loop_enda	12, 13	\xintLength	12
\XINT_last_loop_endb	12, 13	\xintlength	12
\XINT_last_loop_endc	12, 13	\xintLengthUpTo	13
\XINT_last_loop_endd	12, 13	\xintlengthupto	13
\XINT_last_loop_ende	12, 13	\xintMessage	18
\XINT_last_loop_endf	12, 13	\xintodef	11
\XINT_last_loop_endg	12, 13	\xintoodef	11
\XINT_last_loop_endh	12, 13	\xintorthat	10
\XINT_length_finish_a	12	\xintreplicate	14
\XINT_length_loop	12	\xintReverseOrder	11
\XINT_lengthupto_finish_a	13	\xintreverseorder	11
\XINT_lengthupto_finish_b	13	\XINTsetupcatcodes	7
\XINT_lengthupto_gt	13	\xintUniformDeviate	17, 18
\XINT_lengthupto_loop	13		
\XINT_lengthupto_loop_a	13		
\xint_orthat	10		
\XINT_providespackage	8		
\XINT_rep	14		
\XINT_rep_	14		
\XINT_rep_a	14		
		Y	
		\y	5, 6
		Z	
		\Z	10
		\z	5, 6

12.2 Index of [xinttools](#)

A	
<code>\aftergroup</code>	20
B	
<code>\bgroup</code>	43
C	
<code>\catcode</code>	19, 37--40, 42, 43
<code>\csname</code>	19, 20, 28, 29, 31, 38--46, 49--52
D	
<code>\dimexpr</code>	40, 41
E	
<code>\edef</code>	38, 39, 41--45
<code>\endcsname</code>	19, 20, 28, 29, 31, 38--46, 49--52
<code>\endinput</code>	20
<code>\endlinechar</code>	19
<code>\escapechar</code>	44, 45
F	
<code>\futurelet</code>	36, 38, 39
I	
<code>\if</code>	52
<code>\ifcsname</code>	43, 44
<code>\immediate</code>	19
K	
<code>\krof</code>	26--32, 48--52
N	
<code>\newtoks</code>	20
<code>\noexpand</code>	39, 41--43
<code>\numexpr</code>	20, 26--36, 38--43, 45, 47--52
P	
<code>\ProvidesPackage</code>	20
R	
<code>\R</code>	21, 24
<code>\repeat</code>	35, 44
<code>\romannumeral</code>	20--33, 36, 38, 40--43, 45--54
T	
<code>\to</code>	43--45
W	
<code>\w</code>	19, 20
<code>\write</code>	19
X	
<code>\x</code>	19, 20
<code>\xint:</code> ...	21--23, 26--31, 43--45, 47--49, 51--54
<code>\XINT?expr_D</code>	41
<code>\XINT?expr_Da</code>	40
<code>\XINT?expr_Di</code>	41
<code>\XINT?expr_U</code>	41
<code>\XINT?expr_Ua</code>	40
<code>\XINT?expr_Ui</code>	41
<code>\XINT?expr_V</code>	41
<code>\XINT?expr_Va</code>	40
<code>\XINT?expr_Vb</code>	40
<code>\XINT?expr_Vc</code>	40
<code>\XINT?expr_Vd</code>	40, 41
<code>\XINT?expr_Ve</code>	41
<code>\XINT?expr_Vf</code>	40, 41
<code>\XINT?expr_Vi</code>	41
<code>\XINT?expr_Vx</code>	41
<code>\XINT?expr_Vy</code>	41
<code>\xint_afterfi</code>	34, 38, 45
<code>\XINT_apply</code>	32
<code>\XINT_apply_end</code>	32
<code>\XINT_apply_loop_a</code>	32
<code>\XINT_apply_loop_b</code>	32
<code>\XINT_applyunbr</code>	32
<code>\XINT_applyunbr_end</code>	32, 33
<code>\XINT_applyunbr_loop_a</code>	32, 33
<code>\XINT_applyunbr_loop_b</code>	32, 33
<code>\xint_arrayname</code>	44, 45
<code>\XINT_assign_a</code>	43
<code>\XINT_assign_b</code>	43
<code>\XINT_assign_c</code>	43, 44
<code>\XINT_assign_d</code>	44
<code>\XINT_assign_def</code>	43, 44
<code>\XINT_assign_e</code>	44
<code>\XINT_assign_f</code>	43, 44
<code>\XINT_assign_fork</code>	43
<code>\XINT_assign_opt</code>	43
<code>\XINT_assignarray</code>	44, 45
<code>\XINT_assignarray_def</code>	44, 45
<code>\XINT_assignarray_end</code>	45
<code>\XINT_assignarray_fork</code>	44
<code>\XINT_assignarray_loop</code>	45
<code>\XINT_assignarray_opt</code>	44
<code>\xint_bracedstopper</code>	44, 45
<code>\xint_bye</code>	21--34, 47--54
<code>\xint_bye_thenstop</code>	26, 52
<code>\xint_c_</code>	26--31, 34, 44, 45, 47, 51
<code>\xint_c_i</code>	26--31, 33, 42, 45, 47, 49, 51, 52
<code>\xint_c_ii</code>	26--31, 42, 47, 49, 51, 52
<code>\xint_c_iii</code>	26--31, 43, 47, 49, 51, 52
<code>\xint_c_iv</code>	26--31, 47, 49, 51, 52
<code>\xint_c_ix</code>	30, 31, 47, 49, 51, 52
<code>\xint_c_v</code>	26--31, 47, 49, 51, 52
<code>\xint_c_vi</code>	26--31, 47, 49, 51, 52
<code>\xint_c_vii</code>	26--31, 47, 49, 51, 52
<code>\xint_c_viii</code>	26--32, 47--52
<code>\xint_c_x</code>	27
<code>\XINT_csvtol_finish_a</code>	24

\XINT_csvtol_finish_b	24	\XINT_inline_f	37
\XINT_csvtol_finish_di	24, 25	\XINT_inline_g	37
\XINT_csvtol_finish_dii	24, 25	\XINT_inline_macro	37
\XINT_csvtol_finish_diii	24	\XINT_inline_w	37
\XINT_csvtol_finish_div	24	\XINT_item	37
\XINT_csvtol_finish_dv	24	\xint_itemcount	45
\XINT_csvtol_finish_dvi	24	\XINT_keep:f:csv_a	48
\XINT_csvtol_finish_dvii	24	\XINT_keep:f:csv_keepall	49
\XINT_csvtol_finish_dviii	24	\XINT_keep:f:csv_keepnone	48, 49
\XINT_csvtol_loop_a	24	\XINT_keep:f:csv_loop	49--51
\XINT_csvtol_loop_b	24	\XINT_keep:f:csv_loop_end	50
\XINT_expandableerror	45	\XINT_keep:f:csv_loop_pickeight	50
\XINT_first:f:csv_a	53	\XINT_keep:f:csv_neg	49
\xint_firstofone	20, 22, 23, 46, 53	\XINT_keep:f:csv_neg_a	49
\xint_firstoftwo	38, 39, 42	\XINT_keep:f:csv_neg_done	49
\xint_firstoftwo_thenstop	33	\XINT_keep:f:csv_pos	49
\XINT_flet_macro	38, 43, 44	\XINT_keep:f:csv_pos_fork	49
\XINT_flet_sp?	38	\XINT_keep:f:csv_pos_keepall	49, 50
\XINT_flet_zapsp	38, 43, 44	\XINT_keep:f:csv_trimloop	49, 52
\XINT_for	38	\XINT_keep:f:csv_trimloop_finish	49
\XINT_for_d	38, 39	\XINT_keep:f:csv_trimloop_trimmnine	49
\XINT_for_forever?	38	\XINT_keep_a	27
\XINT_for_ifstar	38	\XINT_keep_keepall	28
\XINT_for_last?	39	\XINT_keep_keepnone	27, 29
\XINT_for_last?yes	39, 42, 43	\XINT_keep_loop	28, 30
\XINT_forever	39, 40	\XINT_keep_loop_end	28
\XINT_forever_a	40, 41	\XINT_keep_loop_pickeight	28
\XINT_forever_b	41	\XINT_keep_neg	27, 29
\XINT_forever_c	41	\XINT_keep_neg_a	27
\XINT_forever_d	41	\XINT_keep_pos	27, 28
\XINT_forever_opt_a	41	\XINT_keepunbr_a	29
\XINT_forever_opt_c	41	\XINT_keepunbr_loop	29, 32
\XINT_forfour_d	42, 43	\XINT_keepunbr_loop_end	29
\XINT_forpair_d	42	\XINT_keepunbr_loop_pickeight	29
\XINT_forthree_d	42	\XINT_keepunbr_pos	29
\XINT_forx	38, 39	\XINT_last:f:csv_loop	54
\XINT_forx_d	39	\XINT_last_loop_enda	54
\XINT_forx_empty?	39	\XINT_last_loop_endb	54
\XINT_forx_forever?	39	\XINT_last_loop_endc	54
\xint_gob_til_minus	28, 29, 31, 49--51	\XINT_last_loop_endd	54
\xint_gob_til_R	21, 24	\XINT_last_loop_ende	54
\xint_gob_til_xint:	21, 23, 27, 31, 47, 48, 51, 52, 54	\XINT_last_loop_endf	54
\xint_gob_til_Z	24	\XINT_last_loop_endg	54
\XINT_gobble	26, 28	\XINT_last_loop_endh	54
\xint_gobble_i	21, 35, 37, 40	\XINT_length:f:csv_a	47, 49, 51, 52
\xint_gobble_ii	21, 27	\XINT_length:f:csv_finish	47
\xint_gobble_iii	21, 35, 36	\XINT_length:f:csv_loop	47
\xint_gobble_iv	21, 34	\XINT_length_loop	26, 27, 30, 31
\xint_gobble_thenstop	48, 50	\XINT_lengthupto:f:csv_a	47--49
\xint_gobble_v	21	\XINT_lengthupto:f:csv_empty	48
\xint_gobble_vi	21	\XINT_lengthupto:f:csv_finish_a	48
\xint_gobble_vii	21	\XINT_lengthupto:f:csv_finish_b	48
\xint_gobble_viii	21	\XINT_lengthupto:f:csv_gt	48
\XINT_inline_b	37	\XINT_lengthupto:f:csv_loop_a	48
\XINT_inline_d	37	\XINT_lengthupto:f:csv_loop_b	48
\XINT_inline_e	37	\XINT_lengthupto_loop	28, 29
		\XINT_lws	25

\XINT_lws_e	25, 26	\XINT_to_forever	38, 39
\XINT_lws_e_i	25, 26	\XINT_to_forxever	39
\XINT_lws_e_ii	25, 26	\XINT_token	36--39, 43, 44
\XINT_lws_e_iii	25, 26	\XINT_tokenB	36
\XINT_lws_e_iv	25	\XINT_toks	20, 38, 39, 41--43
\XINT_lws_e_v	25	\XINT_trim:f:csv_a	50
\XINT_lws_e_vi	25	\XINT_trim:f:csv_finish	51
\XINT_lws_loop_a	25	\XINT_trim:f:csv_loop	51, 52
\XINT_lws_loop_b	25, 26	\XINT_trim:f:csv_loop_trimnine	51
\XINT_nthelt:f:csv_a	52	\XINT_trim:f:csv_neg	50
\XINT_nthelt:f:csv_neg	52	\XINT_trim:f:csv_neg_a	51
\XINT_nthelt:f:csv_neg_done	52	\XINT_trim:f:csv_pos	50, 51
\XINT_nthelt:f:csv_neg_fork	52	\XINT_trim:f:csv_pos_done	51, 52
\XINT_nthelt:f:csv_pos	52	\XINT_trim:f:csv_toofew	51
\XINT_nthelt:f:csv_pos_cleanup	52, 53	\XINT_trim:f:csv_trimall	51
\XINT_nthelt:f:csv_pos_done	52, 53	\XINT_trim:f:csv_trimnone	50
\XINT_nthelt_a	26	\XINT_trim_a	30
\XINT_nthelt_neg	26	\XINT_trim_finish	31
\XINT_nthelt_neg_a	26	\XINT_trim_loop	27, 30, 31
\XINT_nthelt_neg_b	26	\XINT_trim_loop_trimnine	31
\XINT_nthelt_pos	26, 27	\XINT_trim_neg	30
\XINT_nthelt_pos_done	27	\XINT_trim_neg_a	30
\XINT_nthelt_zero	26	\XINT_trim_pos	30, 31
\XINT_providespackage	20	\XINT_trim_pos_done	30, 31
\XINT_restorecatcodes_endinput	54	\XINT_trim_toofew	31
\XINT_restoreescapechar	44, 45	\XINT_trim_trimall	30, 32
\XINT_reverse:f:csv_cleanup	53	\XINT_trim_trimnone	30, 31
\XINT_reverse:f:csv_finish	53	\XINT_trimunbr_a	31
\XINT_reverse:f:csv_loop	53	\XINT_trimunbr_neg	31
\XINT_revwbr_finish_a	21	\XINT_trimunbr_neg_a	31
\XINT_revwbr_finish_b	21	\xint_UDsignfork	27, 30, 31, 48, 49, 51, 52
\XINT_revwbr_finish_c	21	\xint_UDzerominusfork	26, 27, 29--31, 48, 50
\XINT_revwbr_loop	21	\XINT_x	39--43
\xint_secondoftwo	38, 39, 41--43	\XINT_xflet	36, 37, 39
\XINT_seq	33	\XINT_xflet_eq?	36
\XINT_seq_chkopt	33	\XINT_xflet_macro	36
\XINT_seq_e	33	\XINT_xflet_sp?	36
\XINT_seq_n	33	\XINT_xflet_spB?	36
\XINT_seq_noopt	33	\XINT_xflet_zapsp	36
\XINT_seq_opt	33, 34	\XINT_xflet_zapspB	36
\XINT_seq_p	33	\XINT_y	39, 41--43
\XINT_seqo	34	\XINT_zapbsp_a	21, 22
\XINT_seqo_a	34	\XINT_zapbsp_again	22
\XINT_seqo_na	34	\XINT_zapbsp_again?	22
\XINT_seqo_nb	35	\XINT_zapbsp_b	22
\XINT_seqo_nc	35	\XINT_zapesp_a	22, 23
\XINT_seqo_nd	35	\XINT_zapesp_b	22
\XINT_seqo_o	34, 35	\XINT_zapesp_e	22
\XINT_seqo_pa	34	\XINT_zapesp_end	22
\XINT_seqo_pb	34	\XINT_zapesp_end?	22
\XINT_seqo_pc	34	\XINT_zapsp_a	23
\XINT_seqo_pd	34	\XINT_zapsp_again	23
\XINT_sptoken	20, 36, 38	\XINT_zapsp_again?	23
\xint_temp	44, 45	\XINT_zapsp_b	23
\XINT_tmpa	38, 54	\XINT_zapsp_c	23
\XINT_tmpb	38, 54	\XINT_zapspb_bracedorone	23
\XINT_tmppc	38, 54	\XINT_zapspb_one?	23

\XINT_zapspb_onlyspaces	23	\xinttiloop_again_b	35, 36
\xintApply	23, 24, 32	\xinttiloopindex	35, 44
\xintapply	32, 46	\xinttiloopskipandredo	36
\xintApplyInline	37, 38	\xinttiloopskiptonext	36
\xintApplyNoExpand	32	\xintintegers	40
\xintapplynoexpand	32	\xintKeep	27
\xintApplyUnbraced	32, 38	\xintkeep	27
\xintapplyunbraced	32	\xintKeep:f:csv	48, 54
\xintApplyUnbracedNoExpand	32	\xintkeep:f:csv	48
\xintapplyunbracednoexpand	32	\xintKeepNoExpand	27
\xintAssign	43	\xintkeepnoexpand	27
\xintAssignArray	44, 45	\xintKeepUnbraced	28
\xintbracediloopindex	35	\xintkeepunbraced	28
\xintbracedouteriloopindex	36	\xintKeepUnbracedNoExpand	28
\xintBreakFor	38, 39	\xintkeepunbracednoexpand	28, 29
\xintBreakForAndDo	38, 40	\xintLastItem:f:csv	54
\xintbreakiloop	35	\xintlastitem:f:csv	54
\xintbreakiloopanddo	35	\xintlength	26
\xintbreakloop	35	\xintLength:f:csv	47, 54
\xintbreakloopanddo	35	\xintlength:f:csv	47
\xintCSVFirstItem	54	\xintLengthUpTo:f:csv	47
\xintCSVKeep	54	\xintlengthupto:f:csv	47
\xintCSVLastItem	54	\xintListWithSep	25
\xintCSVLLength	54	\xintlistwithsep	25
\xintCSVNthEltPy	54	\xintListWithSepNoExpand	25
\xintCSVReverse	54	\xintlistwithsepnoexpand	25
\xintCSVtoList	23	\xintloop	35
\xintcsvtolist	23, 38, 39, 42, 43	\xintloop_again	35
\xintCSVtoListNoExpand	24	\xintloopskiptonext	35
\xintcsvtolistnoexpand	24	\xintNthElt	26
\xintCSVtoListNonStripped	24	\xintnthelt	26
\xintcsvtolistnonstripped	23, 24	\xintNthEltNoExpand	26
\xintCSVtoListNonStrippedNoExpand	24	\xintntheltnoexpand	26
\xintcsvtolistnonstrippednoexpand	24	\xintNthEltPy:f:csv	52, 54
\xintCSVTrim	54	\xintntheltpy:f:csv	52
\xintDigitsOf	45	\xintodef	20
\xintdimensions	40	\xintodef	20
\xintegers	40	\xintouteriloopindex	36
\xintExpandArgs	46	\xinrationals	40
\xintfdef	20	\xintrawwithzeros	40
\xintFirstItem:f:csv	53, 54	\xintRelaxArray	44
\xintfirstitem:f:csv	53	\xintReverse:f:csv	53, 54
\xintFor	38	\xintreverse:f:csv	53
\xintForfour	42	\xintRevWithBraces	20
\xintForpair	42	\xintrevwithbraces	20, 21
\xintForthree	42	\xintRevWithBracesNoExpand	21
\xintgfdef	20	\xintrevwithbracesnoexpand	21
\xintgodef	20	\xintSeq	33
\xintgoodef	20	\xintseq	33
\xintifEq	40	\XINTsetupcatcodes	20
\xintifForFirst	38, 39, 41--43	\xintTrim	30
\xintifForLast	38, 39, 41--43	\xinttrim	30
\xintiiadd	41	\xintTrim:f:csv	50, 54
\xintiimul	40, 41	\xinttrim:f:csv	50
\xintiloop	35, 44	\xintTrimNoExpand	30
\xintiloop_a	35	\xinttrimnoexpand	30
\xintiloop_again	35, 36	\xintTrimUnbraced	31

Y

Z

12.3 Index of **xintcore**

Symbols	
\@backslashchar	112
A	
\aftergroup	56
C	
\catcode	55, 57, 58
\csname	56--59, 92, 111, 112
E	
\endcsname	56--59, 92, 112
\endinput	56
\endlinechar	55
I	
\if	63, 64, 75, 87, 88, 91, 93, 98, 100, 102--104
\ifcsname	57, 58
\ifdefined	58, 59
\immediate	56
K	
\krof	60--65, 72, 75, 76, 78, 81--83, 87, 99, 102--105, 107, 110
N	
\numexpr	56, 59, 60, 62--71, 73--76, 78--87, 89--101, 106, 108--112
P	
\ProvidesPackage	56
R	
\R	66, 68, 70--73, 76, 78, 79, 82--84, 88--90, 93--96, 98, 105, 106, 108, 110
\romannumeral	57, 59--65, 71--73, 75--79, 82--85, 87--96, 99, 102--108, 110
U	
\unless	57, 64, 86
W	
\W	68--85, 88--90, 93--96, 98, 99, 105, 106, 108--111
\w	56
\write	56
X	
\X	73, 78, 90
\x	56
\xint:	59--61, 69--109
\XINT_abs	61
\XINT_add_A	73
\XINT_add_a	73--75, 85
\XINT_add_aa	73
\XINT_add_aa_small	73
\XINT_add_b	74
\XINT_add_bi	74
\XINT_add_c	74
\XINT_add_checklengths	73
\XINT_add_d	74
\XINT_add_di	74
\XINT_add_e	74
\XINT_add_exchange	73
\XINT_add_f	74
\XINT_add_fi	74
\XINT_add_firstiszero	72
\XINT_add_fork	72
\XINT_add_g	74
\XINT_add_h	74
\XINT_add_hi	74, 75
\XINT_add_i	75
\XINT_add_k	74, 75
\XINT_add_ke	75
\XINT_add_kf	75
\XINT_add_l	75
\XINT_add_lf	75
\XINT_add_m	75
\XINT_add_minusminus	72
\XINT_add_minusplus	72
\XINT_add_n	75
\XINT_add_nfork	72
\XINT_add_o	75
\XINT_add_out	73
\XINT_add_plusminus	72
\XINT_add_plusplus	72, 73
\XINT_add_pp_a	72, 73, 78
\XINT_add_pp_b	72, 73
\XINT_add_secondiszero	72
\xint_afterfi	100, 107
\xint_Bye	63--65, 89, 103
\xint_bye	62--65, 67, 82, 89, 91, 94, 99, 103
\xint_c	60, 62, 66, 69, 73, 76, 78, 84, 90, 93, 99, 100, 105
\xint_c_i	60, 63, 64, 67, 70--73, 75, 76, 78--81, 83, 84, 89--91, 93, 94, 96, 98--101, 105, 106, 108, 109, 111, 112
\xint_c_ii	62--64, 72--76, 78, 83--85, 89, 90, 105, 108, 109, 111, 112
\xint_c_iii	76, 90, 111, 112
\xint_c_iv	73, 76, 78, 84, 90, 105, 112
\xint_c_ix	93
\xint_c_mone	58, 60
\xint_c_v	63, 64, 76, 89, 90
\xint_c_vi	72, 73, 76, 78, 84, 90, 92, 105
\xint_c_vii	76, 90, 92
\xint_c_viii	69, 72, 73, 78, 84, 90, 105
\xint_c_x	63--65, 72, 73, 78, 84, 89, 90, 103, 105
\xint_c_x^iv	110
\xint_c_x^ix	59, 85, 86, 97

\xint_c_x^viii	58, 86, 87, 89, 97, 100, 101, 111, 112
\xint_c_x^viii_mone	59, 101, 102
\xint_c_xi_e_viii_mone	59, 83, 100, 101
\xint_c_xii	72, 73, 78, 84, 90, 105
\xint_c_xii_e_viii	59
\xint_c_xiv	71
\xint_c_xxii	84
\XINT_Clamped.handler	58
\XINT_cmp_a	76, 77
\XINT_cmp_checklengths	76
\XINT_cmp_distinctlengths	76
\XINT_cmp_equal	77
\XINT_cmp_firstiszero	75, 76
\XINT_cmp_gt	77
\XINT_cmp_lt	77
\XINT_cmp_minusminus	75, 76
\XINT_cmp_minusplus	75, 76
\XINT_cmp_nfork	75
\XINT_cmp_plusminus	75, 76
\XINT_cmp_plusplus	75, 76
\XINT_cmp_pp	76
\XINT_cmp_secondiszero	75, 76
\XINT_cntSgn	60, 107
\XINT_ConversionSyntax-signal	57
\XINT_ConversionSyntax.handler	58
\XINT_cuz	66, 106
\XINT_cuz_byviii	66
\XINT_cuz_byviii_done	66
\XINT_cuz_byviii_e	66
\XINT_cuz_byviii_z	66
\XINT_cuz_cleantotend	66
\XINT_cuz_hitend	66
\XINT_cuz_loop	66, 68
\XINT_cuz_small	59, 71, 73, 82, 85
\XINT_dbl	62
\XINT_dbl_a	62
\XINT_dbl_e	62, 63
\XINT_dbl_fork	62
\XINT_dbl_neg	62
\XINT_dec	63, 64
\XINT_dec_a	64
\XINT_dec_bye	63, 64
\XINT_dec_e	64
\XINT_dec_fork	64
\XINT_dec_neg	64
\XINT_div_BisOne	88
\XINT_div_BisTwo	88
\XINT_div_BisTwo_a	89
\XINT_div_cleanR	91, 92
\XINT_div_dosmalldiv	89, 90
\XINT_div_dosmallsmall	89
\XINT_div_exittotofinish	94
\XINT_div_finish	91
\XINT_div_finish_a	91
\XINT_div_finish_b	91
\XINT_div_finish_bRpos	91
\XINT_div_finish_bRzero	91
\XINT_div_I_a	92, 97
\XINT_div_I_b	92, 93
\XINT_div_I_c	92, 93
\XINT_div_I_czero	92, 93
\XINT_div_I_da	93
\XINT_div_I_db	93
\XINT_div_I_dc	93
\XINT_div_I_dd	93
\XINT_div_I_de	93, 94
\XINT_div_I_dN	93
\XINT_div_I_dP	93, 94
\XINT_div_I_dz	93
\XINT_div_I_g	93, 94
\XINT_div_I_h	94
\XINT_div_II_b	94
\XINT_div_II_c	95
\XINT_div_II_d	95
\XINT_div_II_e	95
\XINT_div_II_f	95, 96
\XINT_div_II_fa	96
\XINT_div_II_g	96
\XINT_div_II_h	96
\XINT_div_II_k	95, 96
\XINT_div_II_l	96
\XINT_div_II_m	96, 97
\XINT_div_II_skipc	95
\XINT_div_II_skipf	95, 96
\XINT_div_mini	95, 101
\XINT_div_mini_a	101
\XINT_div_mini_b	101
\XINT_div_mini_c	101
\XINT_div_mini_d	101, 102
\XINT_div_mini_w	101
\XINT_div_minimulwc_a	97
\XINT_div_minimulwc_b	97
\XINT_div_minimulwc_c	97
\XINT_div_prepare	88, 103, 104
\XINT_div_prepare_a	88
\XINT_div_prepare_b	88, 90
\XINT_div_prepare_c	90
\XINT_div_prepare_d	90
\XINT_div_prepare_e	90
\XINT_div_prepare_f	90
\XINT_div_prepare_g	90
\XINT_div_prepare_h	90, 91
\XINT_div_prepare_small	88
\XINT_div_small_a	88, 89
\XINT_div_small_b	89
\XINT_div_small_ba	89
\XINT_div_smallmul_a	95--97
\XINT_div_smallmul_e	97
\XINT_div_smallsmall	89
\XINT_div_smallsmallend	89
\XINT_div_start_a	91
\XINT_div_start_b	91
\XINT_div_start_c	91, 92

\XINT_div_start_c	92	\XINT_dsr_fork	64
\XINT_div_start_c_i	92	\XINT_dsr_neg	65
\XINT_div_start_c_ii	92	\XINT_dsrr	65, 103
\XINT_div_start_c_iii	92	\XINT_dsrr_a	65
\XINT_div_start_c_iv	92	\XINT_dsrr_b	65
\XINT_div_start_c_v	92	\XINT_dsrr_e	65
\XINT_div_start_c_vi	92	\XINT_dsrr_fork	65
\XINT_div_start_ca	92	\XINT_dsrr_neg	65
\XINT_div_start_cb	92	\xint_exchangetwo_keepbraces	88, 92, 105
\XINT_div_sub	93--96, 98	\XINT_expandableerror	58
\XINT_div_sub_a	98	\XINT_fac_bigloop_a	110, 111
\XINT_div_sub_b	98	\XINT_fac_bigloop_b	111
\XINT_div_sub_bi	98, 99	\XINT_fac_bigloop_exit	111
\XINT_div_sub_c	98, 99	\XINT_fac_bigloop_loop	111
\XINT_div_sub_clean	98	\XINT_fac_bigloop_mul	111
\XINT_div_sub_d	98	\XINT_fac_checksize	110
\XINT_div_sub_di	98, 99	\XINT_fac_fork	110
\XINT_div_sub_e	98, 99	\XINT_fac_loop_exit	111, 112
\XINT_div_sub_f	98	\XINT_fac_medloop_a	110, 111
\XINT_div_sub_fi	98, 99	\XINT_fac_medloop_b	111
\XINT_div_sub_g	98, 99	\XINT_fac_medloop_loop	111
\XINT_div_sub_h	98	\XINT_fac_medloop_mul	111
\XINT_div_sub_hi	98, 99	\XINT_fac_neg	110
\XINT_div_sub_i	98, 99	\XINT_fac_smallloop_a	110, 111
\XINT_div_sub_l	99	\XINT_fac_smallloop_loop	112
\XINT_div_sub_neg	98	\XINT_fac_smallloop_mul	112
\XINT_div_sub_r	99	\XINT_fac_toobig	110, 111
\XINT_div_unsepQ	67, 91	\XINT_fac_zero	110
\XINT_div_unsepQ_delim	67, 94	\XINT_FDg	61
\XINT_div_unsepQ_one	67, 68	\XINT_fdg	61
\XINT_div_unsepQ_x	67	\xint_firstofone	58
\XINT_div_unsepQ_y	67, 68	\xint_firstofone_thenstop	58
\XINT_div_unsepR	68, 94	\xint_firstofthree_thenstop	58
\XINT_div_unsepR_x	68	\xint_firstoftwo	58, 76, 77, 103
\XINT_div_verysmallisone	97	\xint_firstoftwo_thenstop	102, 103, 105
\XINT_div_verysmallmul	93, 94, 97	\xint_gob_til_dot	90
\XINT_div_verysmallmul_a	97	\xint_gob_til_eightzeroes	66, 67, 86, 95
\XINT_div_verysmallmul_b	97	\xint_gob_til_exclam	96
\XINT_div_verysmallmul_e	97	\xint_gob_til_one	68, 95, 97
\XINT_div_xmini	95, 96	\xint_gob_til_R	66, 68, 70, 71, 88, 89
\XINT_div_xmini_a	95	\xint_gob_til_sc	71,
\XINT_div_xmini_b	95		74, 75, 77, 80--82, 85, 86, 97, 98, 100, 108
\XINT_div_xmini_c	95	\xint_gob_til_xint:	60, 107
\XINT_div_zeroQ	91	\xint_gob_til_Z	67
\XINT_div_zeroQ_end	91	\xint_gob_til_zero	64, 86, 92, 95, 100, 101
\XINT_DivisionByZero.handler	58	\xint_gobble_i	57, 58
\XINT_DivisionImpossible-signal	57	\xint_gobble_iv	112
\XINT_DivisionImpossible.handler	58	\XINT_half	63, 89
\XINT_DivisionUndefined-signal	57	\XINT_half_a	63
\XINT_DivisionUndefined.handler	58	\XINT_half_b	63
\xint_dothis	57, 58, 64, 87, 102--104, 110	\XINT_half_e	63
\XINT_dsl	64	\XINT_half_fork	63
\xint_dsl_zero	64	\XINT_half_neg	63
\XINT_dsr	65	\XINT_iadd	72
\XINT_dsr_a	65	\XINT_icmp	75
\XINT_dsr_b	65	\XINT_idivround	102
\XINT_dsr_e	65	\XINT_iflagRaised	58, 112

\XINT_iadd	72	\XINT_ldg_c	62
\XINT_iicmp	75	\XINT_ldg_cbye	62
\XINT_iidivision	87	\XINT_ldg_d	62
\XINT_iidivision_a	87	\XINT_ldg_fork	62
\XINT_iidivision_aiszero	87	\XINT_minimul_a	86, 100, 101, 106
\XINT_iidivision_aneg	87, 88	\XINT_minimul_b	86
\XINT_iidivision_aneg_b	88	\XINT_minimul_c	86, 87
\XINT_iidivision_aneg_end	88	\XINT_minimulwc_a	85, 86
\XINT_iidivision_aneg_rpos	88	\XINT_minimulwc_b	85
\XINT_iidivision_aneg_rzero	88	\XINT_minimulwc_c	85, 86
\XINT_iidivision_apos	87, 88	\XINT_mul_a	85
\XINT_iidivision_bneg	87	\XINT_mul_b	85
\XINT_iidivision_bpos	87	\XINT_mul_checklengths	84
\XINT_iidivision_divbyzero	87	\XINT_mul_e	85
\XINT_iidivmod	104	\XINT_mul_exchange	84
\XINT_iidivmod_a	104	\XINT_mul_fork	83
\XINT_iidivmod_aiszero	104	\XINT_mul_loop	85, 106, 108
\XINT_iidivmod_bneg	104, 105	\XINT_mul_minusminus	83
\XINT_iidivmod_bneg_finish	105	\XINT_mul_minusplus	83
\XINT_iidivmod_bpos	104, 105	\XINT_mul_nfork	83
\XINT_iidivmod_divbyzero	104	\XINT_mul_oneisone	84
\XINT_iidivround	102	\XINT_mul_out	84, 85, 106, 108--111
\XINT_iidivround_a	102	\XINT_mul_plusminus	83
\XINT_iidivround_aiszero	102--104	\XINT_mul_plusplus	83
\XINT_iidivround_bneg	102	\XINT_mul_pre_b	83, 84
\XINT_iidivround_bpos	102	\XINT_mul_smallbyfirst	84
\XINT_iidivround_divbyzero	102--104	\XINT_mul_smallbysecond	84
\XINT_iidivround_pos	102, 103	\XINT_mul_start	84, 85
\XINT_iidivtrunc	103	\XINT_mul_verysmall	84
\XINT_iidivtrunc_a	103	\XINT_mul_zero	83
\XINT_iidivtrunc_aiszero	103	\XINT_num	59
\XINT_iidivtrunc_bneg	103	\XINT_num_cleanup	59, 60, 82
\XINT_iidivtrunc_bpos	103	\XINT_num_end	60
\XINT_iidivtrunc_divbyzero	103	\XINT_num_loop	59, 60, 82
\XINT_iidivtrunc_pos	103	\XINT_Opp	61
\XINT_iimodtrunc	103	\XINT_opp	61, 72, 76, 78, 79, 87, 107
\XINT_iimodtrunc_a	103, 104	\xint_orthat	57, 58, 64, 87, 102--104, 110
\XINT_iimodtrunc_aiszero	104	\XINT_Overflow.handler	58
\XINT_iimodtrunc_bneg	104	\xint_pow	106
\XINT_iimodtrunc_bpos	104	\XINT_pow_AatleastTwo	107
\XINT_iimodtrunc_divbyzero	104	\XINT_pow_AisOne	107
\XINT_iimodtrunc_pos	104	\XINT_pow_AisZero	107
\XINT_iimul	83	\XINT_pow_Aneg	107
\XINT_iisub	77	\XINT_pow_Apos	107
\XINT_inc	63, 64	\XINT_pow_Apos_a	107
\XINT_inc_a	63	\XINT_pow_Apos_short	107
\XINT_inc_e	63, 65	\XINT_pow_BisNegative	107
\XINT_inc_fork	63	\XINT_pow_BisZero	107, 108
\XINT_inc_neg	63	\XINT_pow_I_exit	108
\XINT_Inexact.handler	58	\XINT_pow_I_in	107, 108
\XINT_InvalidContext-signal	57	\XINT_pow_I_loop	108
\XINT_InvalidContext.handler	58	\XINT_pow_I_squareit	108
\XINT_InvalidOperation.handler	58	\XINT_pow_II_even	109
\XINT_isub	78	\XINT_pow_II_exit	109
\XINT_ldg	62	\XINT_pow_II_in	108
\XINT_ldg_a	62	\XINT_pow_II_loop	108, 109
\XINT_ldg_b	62	\XINT_pow_II_odda	109

\XINT_pow_II_oddb	109	\XINT_smalldivx_b	99
\XINT_pow_mul_small	108	\XINT_smallmul	84--86, 108, 111, 112
\XINT_pow_mulbutcheckifsmall	108, 109	\XINT_smallmul_a	86
\XINT_providespackage	56	\XINT_smallmul_e	86
\XINT_RaiseFlag	58	\XINT_smallmul_f	86
\XINT_resetFlag	58	\XINT_smallunrevbyviii	71, 73
\XINT_resetFlags	58	\XINT_sqr	105
\XINT_restorecatcodes_endinput	112	\XINT_sqr_a	105, 106
\XINT_rev_nounsep	71, 93--96, 98	\XINT_sqr_small	106
\XINT_rev_nounsep_done	71	\XINT_sqr_small_out	106
\XINT_rev_nounsep_end	71	\XINT_sqr_start	106
\XINT_Rounded.handler	58	\XINT_sqr_verysmall	106
\XINT_rsepbyviii	69, 70	\XINT_sub_a	79, 80
\XINT_rsepbyviii_b	69	\XINT_sub_aa	79
\XINT_rsepbyviii_end_A		\XINT_sub_b	80
..... 70, 72, 73, 78, 83, 84, 90, 105, 108		\XINT_sub_bi	80
\XINT_rsepbyviii_end_B		\XINT_sub_c	80
..... 70, 72, 73, 78, 83, 84, 90, 105, 108		\XINT_sub_checklengths	78, 79
\XINT_sdiv_out	90, 99	\XINT_sub_comp_clean	83
\xint_secondoftwo	58, 76, 77	\XINT_sub_comp_finish	82
\xint_secondoftwo_thenstop	102, 104, 105	\XINT_sub_comp_loop	82, 83
\XINT_sepandrev	70, 108	\XINT_sub_d	80
\XINT_sepandrev_a	70	\XINT_sub_di	80, 81
\XINT_sepandrev_andcount		\XINT_sub_e	80, 81
..... 70, 72, 73, 78, 83, 84, 90, 105		\XINT_sub_exchange	79
\XINT_sepandrev_andcount_a	70	\XINT_sub_f	80
\XINT_sepandrev_andcount_b	70, 71	\XINT_sub_fi	80, 81
\XINT_sepandrev_andcount_done	71	\XINT_sub_firstiszero	78
\XINT_sepandrev_andcount_end	70, 71	\XINT_sub_fix_cuz	82
\XINT_sepandrev_b	70	\XINT_sub_fix_neg	82
\XINT_sepandrev_done	70	\XINT_sub_fix_none	81, 82
\XINT_sepandrev_end	70	\XINT_sub_g	80, 81
\XINT_sepbyviii	69	\XINT_sub_h	80
\XINT_sepbyviii_andcount	69, 76, 90	\XINT_sub_hi	80, 81
\XINT_sepbyviii_andcount_a	69	\XINT_sub_i	80, 81
\XINT_sepbyviii_andcount_b	69	\XINT_sub_k	81
\XINT_sepbyviii_andcount_end	69	\XINT_sub_l	81, 82
\XINT_sepbyviii_end	69, 76, 90	\XINT_sub_l_carry	81
\XINT_sepbyviii_Z	68, 89	\XINT_sub_l_Ia	81
\XINT_sepbyviii_Z_end	68, 89	\XINT_sub_l_Ib	81, 82
\XINT_Sgn	60, 88	\XINT_sub_l_Ic	81
\XINT_sgn	60	\XINT_sub_l_Ica	81
\XINT_signalcondition		\XINT_sub_l_Id	81, 82
..... 57, 87, 102, 104, 107, 110, 111		\XINT_sub_l_Id_a	82
\XINT_signalcondition_a	57	\XINT_sub_l_Id_b	82
\XINT_smalldiv_c	100	\XINT_sub_l_Ida	82
\XINT_smalldiv_d	100, 101	\XINT_sub_minusminus	78
\XINT_smalldiv_e	100	\XINT_sub_minusplus	78
\XINT_smalldiv_end	100	\XINT_sub_mm_a	72, 78, 88
\XINT_smalldiv_f	100	\XINT_sub_mm_b	78
\XINT_smalldiv_fz	100	\XINT_sub_nfork	77, 78
\XINT_smalldiv_g	100	\XINT_sub_out	79
\XINT_smalldiv_h	100	\XINT_sub_p	81, 82
\XINT_smalldiv_i	100	\XINT_sub_plusminus	78
\XINT_smalldiv_j	100, 101	\XINT_sub_plusplus	78
\XINT_smalldiv_k	101	\XINT_sub_secondiszero	78
\XINT_smalldivx_a	90, 99	\XINT_Subnormal.handler	58

\xint_UDsignfork	61--65, 87, 102--105
\xint_UDsignsfork	72, 75, 78, 83
\xint_UDzerofork	72, 75, 78, 81--83, 99
\xint_UDzerominusfork	60, 61, 76, 107, 110
\XINT_undefined	58
\XINT_Underflow.handler	58
\XINT_unrevbyviii	71, 73, 79, 85
\XINT_unrevbyviii_a	71
\XINT_unsep_clean	66, 67
\XINT_unsep_cuzsmall	67, 91, 99
\XINT_unsep_cuzsmall_x	67
\XINT_unsep_loop	66--68
\XINT_useiiessage	112
\XINT_verysmallmul	84, 86
\XINT_verysmallmul_a	86
\XINT_verysmallmul_bi	86
\XINT_verysmallmul_bj	86
\XINT_verysmallmul_cj	86
\XINT_verysmallmul_e	86
\XINT_verysmallmul_f	86
\XINT_zeroes_forviii	68, 72, 73, 76, 78, 83, 84, 89, 90, 105, 108
\XINT_zeroes_forviii_end	68
\xintCTRLC	58
\xintDec	64
\xintdec	64
\xintDouble	62
\xintdouble	62
\xintDSL	64
\xintdsl	64
\xintDSR	64
\xintdsr	64
\xintDSRr	65
\xintdsrr	65
\xintColor:thiscannothisappen	107
\xintFDg	61
\xintfdg	61
\xintHalf	63
\xinthalft	63
\xintiiaBs	61
\xintiiabs	61, 105
\xintiiAdd	72
\xintiiadd	72
\xintiiCmp	75
\xintiicmp	75
\xintiidiVflooR	105
\xintiidiivflooor	105
\xintiiDivision	87
\xintiidiviision	87, 102, 105
\xintiiDivMod	104
\xintiividmod	104, 105
\xintiiDivRound	102
\xintiidiivrond	102
\xintiiDivTrunc	103
\xintiidiivtrunc	103
\xintiiFac	110
\xintiifac	110
\xintiiMod	105
\xintiimod	105
\xintiiModTrunc	103
\xintiimodtrunc	103
\xintiiMul	83
\xintiimul	83
\xintiiOpp	61
\xintiioopp	61, 63, 65, 102--105
\xintiiPow	58, 106
\xintiipow	106
\xintiiQuo	102
\xintiiquoo	102
\xintiiRem	102
\xintiirem	102
\xintiiSgn	60
\xintiisgn	60
\xintiiSqR	105
\xintiisqr	105
\xintiiSub	77
\xintiisub	77
\xintInc	63
\xintinc	63, 88
\xintiNum	59
\xintinum	59
\xintLDg	62, 89
\xintlDg	62
\xintNum	59
\xintnum	59, 61, 72, 75, 78, 102
\XINTsetupcatcodes	56
\xintUse	58
Y	
y	56
Z	
Z	59--61, 67, 82, 91, 93, 94
z	56

12.4 Index of **xint**

A	
\aftergroup	114
C	
\catcode	113
\csname	113, 114, 117, 118, 135, 148, 149
E	
\endcsname	113, 114, 116--118, 120, 135, 148, 149
\endinput	114
\endlinechar	113
I	
\if	122, 123, 125--131, 134, 138--140, 145, 146
\ifdefined	114, 152
\iftoggle	148
\immediate	114
K	
\krof	115--122, 129, 130, 133, 150
N	
\numexpr	114--120, 133--151
P	
\ProvidesPackage	114
R	
\R	115, 116, 143, 146
\romannumeral	115--133, 135--139, 141--146, 148--151
U	
\unless	114, 123, 130, 131, 137, 145
W	
\W	115, 116, 143, 146
\w	113, 114
\write	114
X	
\x	113, 114
\xint:	115, 118, 122, 128--133, 139, 143, 148--152
\XINT_add_fork	132
\xint_afterfi	123, 124, 128, 129, 146
\XINT_andof_a	128
\XINT_andof_b	128
\XINT_andof_c	128
\XINT_andof_e	128
\XINT_andof_no	128
\XINT_binom_a	140
\XINT_binom_b	140
\XINT_binom_bigloop	140--142
\XINT_binom_bigloop_a	142
\XINT_binom_bigloop_b	142
\XINT_binom_div	141--144
\XINT_binom_end_	140--142, 144, 145
\XINT_binom_end_i	141, 142, 144, 145
\XINT_binom_end_ii	141, 144
\XINT_binom_end_iii	141, 143, 144
\XINT_binom_finish	143, 144
\XINT_binom_fork	139, 140
\XINT_binom_medloop	140, 141
\XINT_binom_medloop_a	141
\XINT_binom_medloop_b	141
\XINT_binom_mul	141--144
\XINT_binom_pre	139
\XINT_binom_smallloop	140, 141
\XINT_binom_smallloop_a	141, 143
\XINT_binom_smallloop_b	141
\XINT_binom_vbigloop	140, 142
\XINT_binom_vbigloop_a	142
\XINT_binom_vsmallend_	143, 145
\XINT_binom_vsmallend_b	145
\XINT_binom_vsmallend_i	143, 145
\XINT_binom_vsmallend_ib	145
\XINT_binom_vsmallend_ii	143, 144
\XINT_binom_vsmallend_iib	144
\XINT_binom_vsmallend_iii	143, 144
\XINT_binom_vsmallend_iiib	144
\XINT_binom_vsmallfinish	144, 145
\XINT_binom_vsmallloop	140, 142, 143
\XINT_binom_vsmallloop_a	143
\XINT_binom_vsmallloop_b	143
\XINT_binom_vsmallmuldiv	143--145
\xint_bye	115, 117, 118, 121, 137--139, 144
\xint_c_	115, 118, 126, 151
\xint_c_i	115, 118, 134--136, 140--148, 151
\xint_c_ii	115, 118, 133--138, 141--148
\xint_c_ii^vii	149
\xint_c_ii^xiv	149
\xint_c_ii^xxi	149
\xint_c_iii	115, 118, 141, 143, 144, 146--149
\xint_c_iv	115, 118, 136, 141, 143, 146
\xint_c_ix	135
\xint_c_nine_x^viii	114, 149
\xint_c_v	115, 118
\xint_c_vi	115, 118, 133
\xint_c_vii	115, 118
\xint_c_viii	115, 117, 118, 136, 137, 148, 149, 151
\xint_c_x	133, 150
\xint_c_x^viii	136, 143, 146--149
\xint_c_x^viii_mone	140, 146
\xint_c_xvi	137
\XINT_cmp_plusplus	122
\XINT_dbl	137, 138
\XINT_dec	139
\XINT_dec_bye	139
\xint_dothis	123, 133--135, 137, 138, 140, 145, 146, 150, 151

\XINT_dsh_fork	119	\XINT_iiminof_b	131
\XINT_dsh_xisPos	119	\XINT_iiminof_c	131
\XINT_dsh_xiszero	119	\XINT_iiminof_d	131
\XINT_dshr_fork	119	\XINT_iiminof_e	131
\XINT_dshr_xpositive	119	\XINT_is_One	123
\XINT_dshr_xzeroorneg	119	\XINT_isOne	123
\XINT_dsx_addzeros	120, 138	\XINT_isone	123
\XINT_dsx_addzerosnofuss	120, 138	\XINT_len_fork	115
\XINT_dsx_AisNeg	121	\XINT_length_loop	115, 118
\XINT_dsx_AisNeg_checkiffirstempty	121	\XINT_max_fork	129
\XINT_dsx_AisNeg_finish_notzero	121	\XINT_max_minusminus	129, 130
\XINT_dsx_AisNeg_finish_zero	121	\XINT_max_minusplus	129, 130
\XINT_dsx_AisPos	121	\XINT_max_plusminus	129, 130
\XINT_dsx_AisPos_finish	121	\XINT_max_plusplus	129, 130
\XINT_dsx_AisZero	120, 121	\XINT_max_pluszero	129, 130
\XINT_dsx_append	116, 120	\XINT_max_zeroplus	129, 130
\XINT_dsx_end	121	\XINT_max_zerozero	129
\XINT_dsx_fork	120	\XINT_microrevsep	116
\XINT_dsx_xisNeg_Azero	120	\XINT_microrevsep_end	115, 116
\XINT_dsx_xisNeg_checkA	119, 120	\XINT_min_fork	130
\XINT_dsx_xisPos	119, 120	\XINT_min_minusminus	130, 131
\XINT_dsx_xisZero	120	\XINT_min_minusplus	130, 131
\XINT_eightrandomdigits	149	\XINT_min_plusminus	130, 131
\XINT_expandableerror	150, 152	\XINT_min_plusplus	130, 131
\xint_firstofthree	114	\XINT_min_pluszero	130, 131
\xint_firstofthree_thenstop	114, 124, 125	\XINT_min_zeroplus	130, 131
\xint_firstoftwo	128	\XINT_min_zerozero	130, 131
\xint_firstoftwo_thenstop	125--127, 129--131	\XINT_mul_fork	132
\XINT_geq	122	\XINT_mul_out	148
\XINT_geq_finish	122	\XINT_num	121
\XINT_geq_firstiszero	122	\XINT_orof_a	128
\XINT_geq_fork	122	\XINT_orof_b	128
\XINT_geq_minusminus	122	\XINT_orof_c	128
\XINT_geq_minusplus	122	\XINT_orof_e	128
\XINT_geq_no	122	\XINT_orof_yes	128
\XINT_geq_plusminus	122	\xint_orthat	123, 133--135, 137, 138, 140, 146, 150, 151
\XINT_geq_plusplus	122, 130, 131	\XINT_pfac_a	146
\XINT_geq_secondiszero	122	\XINT_pfac_b	146
\XINT_geq_yes	122	\XINT_pfac_bigloop	146, 147
\xint_gob_til_dot	121	\XINT_pfac_bigloop_a	147
\xint_gob_til_R	116	\XINT_pfac_bigloop_b	147
\xint_gob_til_xint:	128, 129, 131, 132	\XINT_pfac_end_	146--148
\xint_gob_til_Z	116	\XINT_pfac_end_i	146--148
\xint_gob_til_zero	120	\XINT_pfac_end_ii	146--148
\xint_gobble_i	149	\XINT_pfac_end_iii	146, 148
\XINT_ii_e_a	116	\XINT_pfac_fork	145
\XINT_ii_e_fork	116	\XINT_pfac_medloop	146, 147
\XINT_ii_e_neg	116	\XINT_pfac_medloop_a	147
\XINT_ii_geq	122	\XINT_pfac_medloop_b	147
\xint_iimax	129	\XINT_pfac_neg	145, 146
\XINT_iimaxof_a	131	\XINT_pfac_one	145, 146
\XINT_iimaxof_b	131	\XINT_pfac_outofrange	146
\XINT_iimaxof_c	131	\XINT_pfac_smallloop	146
\XINT_iimaxof_d	131	\XINT_pfac_smallloop_a	146
\XINT_iimaxof_e	131	\XINT_pfac_smallloop_b	146
\xint_iimin	130	\XINT_pfac_vbigloop	146--148
\XINT_iiminof_a	131		

\XINT_pfac_vbigloop_a	147	\XINT_sqrt_a	133
\XINT_pfac_zero	146	\XINT_sqrt_bE	133
\XINT_prdexpr	132	\XINT_sqrt_big_a	133
\XINT_prod_finished	132	\XINT_sqrt_big_d	133, 135
\XINT_prod_loop_a	132	\XINT_sqrt_big_eE	135
\XINT_prod_loop_b	132	\XINT_sqrt_big_eE_a	135
\XINT_prod_loop_c	132	\XINT_sqrt_big_end	139
\XINT_providespackage	114	\XINT_sqrt_big_e0	135
\XINT_randomdigits	148	\XINT_sqrt_big_e0_a	135
\XINT_randomdigits_a	148	\XINT_sqrt_big_f	135, 136
\XINT_randrange	150, 152	\XINT_sqrt_big_fa	136
\XINT_randrange_A	151	\XINT_sqrt_big_ga	136
\XINT_randrange_a	150	\XINT_sqrt_big_gb	136
\XINT_randrange_again	151	\XINT_sqrt_big_gc	136
\XINT_randrange_B	151	\XINT_sqrt_big_gd	136, 137
\XINT_randrange_b	150	\XINT_sqrt_big_ge	137
\XINT_randrange_c	150, 151	\XINT_sqrt_big_gf	137
\XINT_randrange_d	151	\XINT_sqrt_big_gg	137
\XINT_randrange_E	151	\XINT_sqrt_big_gi	137
\XINT_randrange_e	151	\XINT_sqrt_big_gj	137
\XINT_randrange_err:empty	150	\XINT_sqrt_big_gk	137
\XINT_randrange_Z	151	\XINT_sqrt_big_gl	137
\XINT_randrangeAtoB_a	150	\XINT_sqrt_big_gloop	137, 138
\XINT_rdg	149	\XINT_sqrt_big_gm	137, 138
\XINT_rdg_aux	149	\XINT_sqrt_big_gn	138
\XINT_rep	116, 120	\XINT_sqrt_big_ka	136--138
\XINT_replicate	149	\XINT_sqrt_big_kb	138
\XINT_restorecatcodes_endinput	152	\XINT_sqrt_big_kc	138
\XINT_rev_nounsep	143	\XINT_sqrt_big_ke	138, 139
\XINT_revdigits	115	\XINT_sqrt_big_kend	138
\XINT_revdigits_a	115	\XINT_sqrt_big_kf	139
\XINT_revdigits_b	116	\XINT_sqrt_big_kg	139
\XINT_revdigits_end	116	\XINT_sqrt_big_kloop	138, 139
\xint_secondofthree	114	\XINT_sqrt_big_kz	138
\xint_secondofthree_thenstop	114, 124, 125	\XINT_sqrt_bigormed_f	135
\xint_secondoftwo	128	\XINT_sqrt_b0	133
\xint_secondoftwo_thenstop	119, 125--127, 129--131	\XINT_sqrt_c	133
\XINT_Sgn	139	\XINT_sqrt_checkin	132, 133
\XINT_signalcondition	133, 140, 146	\XINT_sqrt_isneg	133
\XINT_smalldivx_a	143	\XINT_sqrt_iszero	133
\XINT_smallmul	143, 146--148	\XINT_sqrt_med_fa	135, 136
\XINT_split_finish	117	\XINT_sqrt_med_fb	136
\XINT_split_fromleft	117, 118, 138	\XINT_sqrt_med_fv	135
\XINT_split_fromleft_a	117	\XINT_sqrt_med_fvi	135
\XINT_split_fromleft_b	117	\XINT_sqrt_med_fvii	135
\XINT_split_fromleft_clean	117	\XINT_sqrt_med_fviii	135
\XINT_split_fromleft_end_a	117	\XINT_sqrt_post	139
\XINT_split_fromleft_end_b	117, 118	\XINT_sqrt_small_a	133
\XINT_split_fromright	117, 118	\XINT_sqrt_small_d	133, 134
\XINT_split_fromright_a	118	\XINT_sqrt_small_e	134, 135
\XINT_split_fromright_Lempty	118	\XINT_sqrt_small_ea	134, 136
\XINT_split_xfork	117, 118, 121	\XINT_sqrt_small_eb	134
\XINT_split_zerosplit	117	\XINT_sqrt_small_ec	134
\XINT_splitl_finish	118	\XINT_sqrt_small_end	134, 135
\XINT_splitr_finish	118	\XINT_sqrt_small_ez	134
\XINT_sqrt	133	\XINT_sqrt_small_f	134, 135
		\XINT_sqrt_small_g	134

\XINT_sqrt_small_h	134, 135	\xintiiGt	122
\XINT_sqrt_start	133	\xintiiGtorEq	123
\XINT_sqrtr_post	139	\xintiiifCmp	125
\XINT_sum_finished	132	\xintiiifcmp	125
\XINT_sum_loop_a	132	\xintiiifEq	125
\XINT_sum_loop_b	132	\xintiiifeq	121, 125
\XINT_sum_loop_c	132	\xintiiifGt	125
\XINT_sumexpr	132	\xintiiifgt	122, 123, 125
\xint_texuniformdeviate	114, 149, 152	\xintiiifLt	126, 139, 151
\xint_thirdofthree	114	\xintiiiflt	122, 123, 126
\xint_thirdofthree_thenstop	114, 124, 125	\xintiiifNotZero	126
\xint_UDsignfork	115--118	\xintiiifnotzero	126, 127
\xint_UDsignsfork	122, 129, 130	\xintiiif0dd	127
\xint_UDzerofork	122	\xintiiifodd	127
\xint_UDzerominusfork ...	117, 119, 120, 133, 150	\xintiiifOne	126
\xint_UDzerosfork	129, 130	\xintiiifone	126
\XINT_uniformdeviate	150, 151	\xintiiifSgn	125
\XINT_unsep_cuzsmall	144	\xintiiifsgn	125
\XINT_XINT_rdg	149	\xintiiifZero	126
\XINT_xorof_a	128, 129	\xintiiifzero	126, 127
\XINT_xorof_b	129	\xintiiIsNotZero	123
\XINT_xorof_c	129	\xintiiisnotzero	123, 128
\XINT_xorof_e	129	\xintiiIsOne	123, 127
\XINT_xrandomdigits	149	\xintiiisone	123
\XINT_xrandomdigits_a	149	\xintiiIsZero	123, 128
\xintAND	127	\xintiiiszero	123, 127
\xintand	127, 128	\xintiiLt	122
\xintANDof	128	\xintiiLtorEq	123
\xintandof	128	\xintiiMax	129
\xintBool	148	\xintiiimax	129, 131
\xintDecSplit	117	\xintiiMaxof	131
\xintdecsplit	117	\xintiiimaxof	131
\xintDecSplitL	118	\xintiiMin	130
\xintdecsplitl	118	\xintiiimin	130, 131
\xintDecSplitR	118	\xintiiMinof	131
\xintdecsplitr	118	\xintiiiminof	131
\xintDouble	136	\xintiiMMON	124
\xintDSH	119	\xintiiimmon	124
\xintdsh	119	\xintiiMON	124
\xintDSHr	119	\xintiiimon	124
\xintdshr	119	\xintiiMul	136
\xintDSx	120	\xintiiNotEq	121
\xintdsx	120	\xintiiOdd	123, 127
\xintifFalseAelseB	127	\xintiiodd	123
\xintifTrueAelseB	127--129	\xintiiPFactorial	145
\xintiiadd	136, 137, 139, 150	\xintiipfactorial	145
\xintiiBinomial	139	\xintiiPrd	132
\xintiiibinomial	139	\xintiiiprd	132
\xintiiCmp	125, 126	\xintiiRandRange	150
\xintiidivision	137, 138	\xintiiirandrange	150
\xintiiE	116	\xintiiRandRangeAtoB	150
\xintiie	116	\xintiiirandrangeAtoB	150
\xintiiEq	121	\xintiiSgn	123, 125, 126, 128
\xintiiEven	124	\xintiiSqr	136, 137, 139
\xintiieven	124	\xintiiSqrt	139
\xintiiGeq	122	\xintiisqrt	139
\xintiigeq	122	\xintiiSqrtR	139

Y

Z

12.5 Index of xintbinhex

A	
<code>\aftergroup</code>	153, 154
C	
<code>\catcode</code>	153
<code>\csname</code>	153--155, 157--159, 162--164
E	
<code>\edef</code>	154
<code>\endcsname</code>	153--155, 157--159, 162--164
<code>\endinput</code>	153, 154
<code>\endlinechar</code>	153
I	
<code>\if</code>	158, 160, 161
<code>\immediate</code>	153
K	
<code>\krof</code>	156, 158, 159, 162--164
N	
<code>\numexpr</code>	153, 156--163
P	
<code>\ProvidesPackage</code>	154
R	
<code>\R</code>	155, 156, 158, 160, 162
<code>\romannumeral</code>	155, 156, 158--164
U	
<code>\unless</code>	158
W	
<code>\W</code>	155, 156, 158, 160, 162
<code>\w</code>	153, 154
<code>\write</code>	153
X	
<code>\x</code>	153, 154
<code>\xint:</code>	155--159, 161, 162
<code>\xint_afterfi</code>	157
<code>\XINT_btd_checkin</code>	161
<code>\XINT_btd_htd</code>	162
<code>\XINT_btd_main</code>	162
<code>\XINT_btd_N</code>	161, 162
<code>\XINT_bth_checkin</code>	162
<code>\XINT_bth_loop</code>	162, 163
<code>\XINT_bth_main</code>	162
<code>\XINT_bth_N</code>	162
<code>\xint_bye</code>	156, 158, 160--164
<code>\xint_c</code>	157
<code>\xint_c_i</code>	156--159
<code>\xint_c_ii^vii</code>	157, 158
<code>\xint_c_ii^viii</code>	157, 158
<code>\xint_c_ii^xv</code>	154, 156
<code>\xint_c_ii^xvi</code>	154, 156, 160
<code>\xint_c_viii</code>	157--159
<code>\xint_c_xvi</code>	157--159
<code>\XINT_chtb_checkin</code>	164
<code>\XINT_chtb_main</code>	164
<code>\XINT_chtb_N</code>	164
<code>\XINT_csbth_none</code>	155
<code>\XINT_cshtb_A</code>	155
<code>\XINT_cshtb_B</code>	155
<code>\XINT_cshtb_C</code>	155
<code>\XINT_cshtb_D</code>	155
<code>\XINT_cshtb_E</code>	155
<code>\XINT_cshtb_F</code>	155
<code>\XINT_cshtb_none</code>	155
<code>\xint_dothis</code>	157, 158
<code>\XINT_dtb_checkin</code>	158
<code>\XINT_dtb_finish</code>	158, 159
<code>\XINT_dtb_finish_a</code>	159
<code>\XINT_dtb_main</code>	158
<code>\XINT_dtb_neg</code>	158
<code>\XINT_dtb_tobin</code>	158, 159
<code>\XINT_dtb_tobin_a</code>	158
<code>\XINT_dth_checkin</code>	155
<code>\XINT_dth_finish</code>	156, 158
<code>\XINT_dth_main</code>	156
<code>\XINT_dth_neg</code>	155, 156
<code>\XINT_dth_tohex</code>	156--158
<code>\XINT_dth_tohex_a</code>	157
<code>\XINT_dthb_again</code>	156, 157
<code>\XINT_dthb_exclam</code>	156
<code>\XINT_dthb_lastpass</code>	156
<code>\XINT_dthb_nextfour</code>	156
<code>\XINT_dthb_small</code>	156
<code>\XINT_dthb_start</code>	156, 158
<code>\XINT_dthb_start_a</code>	156
<code>\XINT_dthb_update</code>	156, 157
<code>\XINT_dthb_update_a</code>	156
<code>\xint_firstoftwo</code>	157
<code>\xint_gob_til_R</code>	155
<code>\XINT_htb_checkin</code>	163
<code>\XINT_htb_cuz</code>	163
<code>\XINT_htb_loop</code>	163, 164
<code>\XINT_htb_main</code>	163
<code>\XINT_htb_N</code>	163
<code>\XINT_htd_A</code>	160, 161
<code>\XINT_htd_a</code>	160, 161
<code>\XINT_htd_Aa</code>	161
<code>\XINT_htd_Ab</code>	161
<code>\XINT_htd_again</code>	160, 161
<code>\XINT_htd_checkin</code>	159
<code>\XINT_htd_finish</code>	160, 161
<code>\XINT_htd_finish_cuz</code>	161
<code>\XINT_htd_main</code>	159
<code>\XINT_htd_neg</code>	159

\XINT_htd_nextfour	160, 161	\XINT_unsep_clean	161
\XINT_htd_starta	160, 162	\XINT_zeroes_foriv	155, 156, 158, 160, 162
\XINT_htd_startb	160, 162	\XINT_zeroes_foriv_done	155
\XINT_htd_startba	160	\XINT_zeroes_foriv_end	155
\XINT_htd_startbb	160	\xintBinToDec	161
\XINT_htd_unsep_loop	161	\xintbintodec	161
\XINT_htd_unsep_loop_a	161	\xintBinToHex	162
\XINT_htd_update	160	\xintbintoHex	162
\xint_orthat	157, 158	\xintCHexToBin	163
\XINT_providespackage	154	\xintchextobin	163, 164
\XINT_restorecatcodes_endinput	164	\xintDecToBin	158
\xint_secondoftwo	157	\xintdectobin	158
\XINT_tmpa	154	\xintDecToHex	155
\XINT_tofourhex	157	\xintdectohex	155
\XINT_tofourhex_a	157	\xintHexToBin	163
\XINT_tofourhex_c	157	\xinthextobin	163
\XINT_tofourhex_d	157	\xintHexToDec	159
\XINT_tofourhex_e	157	\xinthextodec	159
\XINT_tosixteenbits	158	\XINTsetupcatcodes	154
\XINT_tosixteenbits_a	158		
\XINT_tosixteenbits_c	158, 159	Y	
\XINT_tosixteenbits_d	159	\y	153
\XINT_tosixteenbits_e	159		
\xint_UDsignfork	155, 158, 159, 161--164	Z	
		\z	153, 154

12.6 Index of [xintgcd](#)

A		V	
<code>\A</code>	175--177	<code>\V</code>	177
<code>\aftergroup</code>	165, 166	<code>\vbox</code>	175, 176
B		W	
<code>\B</code>	175--177	<code>\w</code>	165, 166
<code>\BEZ</code>	176, 177	<code>\write</code>	165
<code>\break</code>	176	X	
C		<code>\x</code>	165, 166
<code>\catcode</code>	165	<code>\xint:</code>	170--175
<code>\count</code>	175--177	<code>\XINT_bezalg</code>	173
<code>\cr</code>	175, 176	<code>\XINT_bezalg_a</code>	173, 174
<code>\csname</code>	165	<code>\XINT_bezalg_AisZero</code>	173
D		<code>\XINT_bezalg_b</code>	173
<code>\D</code>	177	<code>\XINT_bezalg_BisZero</code>	173
E		<code>\XINT_bezalg_c</code>	173, 174
<code>\edef</code>	175--177	<code>\XINT_bezalg_d</code>	174
<code>\endcsname</code>	165	<code>\XINT_bezalg_e</code>	174
<code>\endinput</code>	165, 166	<code>\XINT_bezalg_end</code>	174
<code>\endlinechar</code>	165	<code>\XINT_bezalg_end_a</code>	174
<code>\errmessage</code>	175, 176	<code>\XINT_bezalg_fork</code>	173
H		<code>\XINT_bezout</code>	168
<code>\halign</code>	175, 176	<code>\XINT_bezout_botharezero</code>	168
<code>\hfill</code>	176	<code>\XINT_bezout_exitA</code>	170, 171
I		<code>\XINT_bezout_exita</code>	171
<code>\if</code>	170, 171	<code>\XINT_bezout_firstiszero</code>	168
<code>\ifdefined</code>	175, 176	<code>\XINT_bezout_fork</code>	168
<code>\immediate</code>	165	<code>\XINT_bezout_loop_B</code>	170, 171
<code>\indent</code>	176	<code>\XINT_bezout_loop_b</code>	171
K		<code>\XINT_bezout_loop_C</code>	170
<code>\krof</code>	166--169, 172, 173	<code>\XINT_bezout_loop_c</code>	171
N		<code>\XINT_bezout_loop_D</code>	170, 171
<code>\N</code>	175--177	<code>\XINT_bezout_loop_d</code>	171
<code>\numexpr</code>	165, 172, 173, 176	<code>\XINT_bezout_loop_E</code>	171
P		<code>\XINT_bezout_loop_e</code>	171
<code>\ProvidesPackage</code>	166	<code>\XINT_bezout_loop_minusminus</code>	168, 169
R		<code>\XINT_bezout_loop_minusplus</code>	168, 169
<code>\repeat</code>	176, 177	<code>\XINT_bezout_mm_post</code>	169
<code>\romannumeral</code>	166--175	<code>\XINT_bezout_mm_postb</code>	169
T		<code>\XINT_bezout_mp_post</code>	169
<code>\times</code>	176, 177	<code>\XINT_bezout_plusminus</code>	168, 169
<code>\to</code>	175, 176	<code>\XINT_bezout_plusplus</code>	168, 169
U		<code>\XINT_bezout_pm_post</code>	169
<code>\U</code>	175--177	<code>\XINT_bezout_preloop_A</code>	170
<code>\unless</code>	175, 176	<code>\XINT_bezout_preloop_a</code>	169, 170
		<code>\XINT_bezout_preloop_exchange</code>	170
		<code>\XINT_bezout_preloop_exit</code>	170
		<code>\XINT_bezout_secondiszero</code>	168
		<code>\xint_bye</code>	173, 174
		<code>\xint_c_i</code>	172, 173
		<code>\XINT_div_prepare</code>	167, 169--173
		<code>\xint_dothis</code>	170
		<code>\XINT_euc</code>	171
		<code>\XINT_euc_a</code>	172

Y

Z

12.7 Index of xintfrac

A		X	
<code>\afterassignment</code>	207	<code>\X</code>	203
<code>\aftergroup</code>	179	<code>\x</code>	179
C		<code>\xint:</code>	180--184, 190, 196, 198--200, 202--207, 209, 210, 212, 217--223, 230, 232--235, 239, 240, 242, 243, 247, 248, 251--253, 255, 257
<code>\catcode</code>	179, 191, 192	<code>\XINT_abs</code>	180, 221
<code>\csname</code>	179, 227, 231, 245, 246	<code>\xint_afterfi</code>	186, 191, 192, 194, 196, 200, 216, 218, 219, 238, 246, 248
E		<code>\XINT_binom_pre</code>	211
<code>\endcsname</code>	179, 204, 206, 207, 227, 231, 236, 241, 246	<code>\xint_Bye</code>	201, 226, 241, 242
<code>\endinput</code>	179	<code>\xint_bye</code>	180, 183, 184, 199, 201, 204, 206, 207, 226--228, 236, 241, 242, 254
<code>\endlinechar</code>	179	<code>\xint_c</code>	180, 183, 184, 187--190, 199, 200, 203, 205, 206, 228, 230, 232, 237, 238, 241, 242, 247, 257
F		<code>\xint_c_i</code>	180, 183--185, 199--205, 223, 226, 228--230, 236--238, 241--246, 248--252, 254--256
<code>\frac</code>	191	<code>\xint_c_ii</code>	180, 183, 184, 199, 202, 203, 227, 237, 238, 241, 242, 244--246, 249--251, 254, 255
I		<code>\xint_c_ii^v</code>	204, 205
<code>\if</code>	181, 185--189, 191, 192, 194--196, 198--200, 209, 213--216, 218, 219, 224, 225, 227--229, 236, 238, 239, 241, 242, 246, 248, 252, 254, 257	<code>\xint_c_ii^vi</code>	204, 205
<code>\iffalse</code>	236, 241	<code>\xint_c_iii</code>	180, 183, 184, 199, 202, 227, 236, 240, 244--246, 249--252, 254
<code>\iftrue</code>	236, 241	<code>\xint_c_iv</code>	180, 183, 184, 199, 203, 226, 244, 246, 249, 252
<code>\immediate</code>	179	<code>\xint_c_ix</code>	185
K		<code>\xint_c_v</code>	180, 183, 184, 199, 228, 230, 241, 242, 254
<code>\krof</code>	182--184, 191--199, 201, 204, 206, 208, 209, 211, 213--215, 217--221, 223, 225, 226, 230, 235, 236, 238, 240, 241, 243, 253, 255	<code>\xint_c_vi</code>	180, 183, 184, 199, 203
M		<code>\xint_c_vii</code>	180, 183, 184, 199
<code>\mathchardef</code>	207	<code>\xint_c_viii</code>	180, 183, 184, 199, 203, 244, 246, 247, 249, 255
N		<code>\xint_c_x</code>	201, 203, 226, 241, 242
<code>\numexpr</code>	179, 180, 182--185, 193, 194, 198--207, 210--212, 214, 215, 217, 220, 223--230, 232--238, 240--257	<code>\xint_c_x^iv</code>	243, 244, 249
O		<code>\xint_c_x^ix</code>	247
<code>\over</code>	192	<code>\xint_c_x^viii</code>	202, 244--251
P		<code>\xint_c_x^viii_mone</code>	243, 248, 252
<code>\ProvidesPackage</code>	179	<code>\xint_c_xii</code>	203
R		<code>\XINT_cmp_plusplus</code>	221
<code>\R</code>	201--203, 244	<code>\XINT_cntSgn</code>	180, 190, 217, 220
<code>\romannumeral</code>	180--242, 244, 246--251, 253--258	<code>\XINT_cntSgnFork</code>	180, 217, 220
U		<code>\XINT_dbl</code>	254
<code>\unless</code>	248	<code>\XINT_dectostr</code>	189
W		<code>\XINT_dectostr_a</code>	189
<code>\W</code>	201--203, 244	<code>\XINT_dectostr_b</code>	189
<code>\w</code>	179	<code>\XINT_dectostr_z</code>	189
<code>\write</code>	179	<code>\XINT_denom_A</code>	190, 191
		<code>\XINT_denom_B</code>	190, 191
		<code>\XINT_denom_fork</code>	190
		<code>\XINT_div_prepare</code>	195, 196, 199, 208, 214--216
		<code>\XINT_div_prepare_g</code>	203
		<code>\XINT_div_small_b</code>	202
		<code>\XINT_div_small_ba</code>	202

\XINT_divmod_a	214	\XINT_fcmp_D	220
\XINT_divmod_aiszero	214	\XINT_fcmp_E	220
\XINT_divmod_b	214	\XINT_fcmp_Fd	221
\XINT_divmod_bneg	214	\XINT_fcmp_Fe	221
\XINT_divmod_bneg_finish	214	\XINT_fcmp_firstneg	220
\XINT_divmod_bpos	214, 215	\XINT_fcmp_firstzero	220
\XINT_divmod_bpos_a	215	\XINT_fcmp_Fn	221
\XINT_divmod_bpos_finish	215	\XINT_fcmp_Fo	221
\XINT_divmod_divbyzero	214	\XINT_fcmp_minusminus	220
\xint_dothis		\XINT_fcmp_nonneg_a	220
185, 189, 198--200, 213--215, 224, 228, 230,		\XINT_fcmp_pos	220
232, 236, 241--243, 248, 249, 252, 254, 257		\XINT_fcmp_secondneg	220
\XINT_dsrr	201, 226	\XINT_fcmp_secondzero	220
\XINT_dsx_addzeros	189-	\XINT_fcmp_zerozero	220
-191, 198, 199, 205, 208, 217, 221, 224, 254		\XINT_fdiv	212
\XINT_dsx_addzerosnofuss	227	\XINT_fdiv_A	212
\XINT_e	194	\XINT_fdiv_B	212
\XINT_e_end	194	\XINT_fdiv_C	212
\XINT_eightrandmdigits	258	\XINT_fgeq	216
\xint_exchangetwo_keepbraces	215	\XINT_fgeq_A	216--219
\XINT_fac_fork	211	\XINT_fgeq_B	217
\XINT_factortens	185, 193	\XINT_fgeq_C	217
\XINT_factortens_a	185	\XINT_fgeq_D	217
\XINT_factortens_b	185	\XINT_fgeq_E	217
\XINT_factortens_c	185	\XINT_fgeq_Fd	217
\XINT_factortens_cc	185	\XINT_fgeq_Fe	217
\XINT_factortens_d	185	\XINT_fgeq_Fn	217
\XINT_factortens_e	185	\XINT_fgeq_Fo	217
\XINT_factortens_f	185	\XINT_fgeq_Zi	217
\XINT_factortens_g	185	\XINT_fgeq_Zii	217
\XINT_factortens_x	185	\xint_firstofone	229
\XINT_factortens_y	185	\xint_firstofthree	180
\XINT_factortens_yy	185	\xint_firstofthree_thenstop	186, 187
\XINT_factortens_z	185	\xint_firstoftwo	188, 199, 232, 247
\XINT_fadd	207	\xint_firstoftwo_thenstop	187, 188, 191, 195, 257
\XINT_fadd_a	207	\XINT_FL_add_a	232, 233
\XINT_fadd_Aa	207	\XINT_FL_add_b	232
\XINT_fadd_Azero	207	\XINT_FL_add_c	232
\XINT_fadd_B	207, 208	\XINT_FL_add_d	232
\XINT_fadd_b	207	\XINT_FL_add_zero	232
\XINT_fadd_Ba	208	\XINT_FL_binom_a	252
\XINT_fadd_Bb	208	\XINT_FL_binom_aa	252
\XINT_fadd_Bzero	207, 209	\XINT_FL_binom_ab	252
\XINT_fadd_C	208	\XINT_FL_binom_fork	252
\XINT_fadd_c	207, 209	\XINT_FL_binom_neg	252
\XINT_fadd_D_a	208	\XINT_FL_binom_one	252
\XINT_fadd_D_b	208	\XINT_FL_binom_toobig	252
\XINT_fadd_D_c	208	\XINT_FL_binom_zero	252
\XINT_fadd_D_exit	208	\XINT_FL_div_a	234, 235
\XINT_fadd_E	208	\XINT_FL_div_b	235
\XINT_fadd_F	208	\XINT_FL_fac_addzeros	246
\XINT_fadd_G	208, 209	\XINT_FL_fac_addzeros_exit	246
\XINT_fadd_iszero	208, 209	\XINT_FL_fac_big	243, 244
\XINT_fcmp	219	\XINT_FL_fac_bigloop_a	244, 245
\XINT_fcmp_A	220	\XINT_FL_fac_bigloop_b	245
\XINT_fcmp_B	220	\XINT_FL_fac_bigloop_loop	245
\XINT_fcmp_C	220	\XINT_FL_fac_bigloop_mul	245

\XINT_FL_fac_countdigits	244, 249	\XINT_FL_pfac_zero	248
\XINT_FL_fac_countdone	244	\XINT_FL_sqrt_a	253
\XINT_FL_fac_fork_a	243	\XINT_FL_sqrt_isneg	253
\XINT_FL_fac_fork_b	243, 252	\XINT_FL_sqrt_iszero	253
\XINT_FL_fac_increaseP	244	\XINT_FL_sqrt_pos	253, 254
\XINT_FL_fac_isneg	243	\XINT_fladd_chkopt	232
\XINT_FL_fac_iszero	243	\XINT_fladd_noopt	232
\XINT_FL_fac_loop_exit	244--246	\XINT_fladd_opt	232
\XINT_FL_fac_med	243, 244	\XINT_fladd_opt_a	232
\XINT_FL_fac_medloop_a	244, 245	\XINT_flbinom_chkopt	251
\XINT_FL_fac_medloop_b	245	\XINT_flbinom_noopt	251
\XINT_FL_fac_medloop_loop	245	\XINT_flbinom_opt	251, 252
\XINT_FL_fac_medloop_mul	245	\XINT_fldiv_chkopt	234
\XINT_FL_fac_minimulwc_a	246, 247	\XINT_fldiv_noopt	234
\XINT_FL_fac_minimulwc_b	247	\XINT_fldiv_opt	234
\XINT_FL_fac_minimulwc_c	247	\XINT_fldiv_opt_a	234
\XINT_FL_fac_minimulwc_d	247	\XINT_flen	180
\XINT_FL_fac_minimulwc_e	247	\XINT_flfac_chkopt	242
\XINT_FL_fac_mul	245, 246, 249--251	\XINT_flfac_noopt	242, 243
\XINT_FL_fac_mul_a	246	\XINT_flfac_opt	242, 243
\XINT_FL_fac_out	243, 244, 251, 252	\XINT_flfac_opt_a	243
\XINT_FL_fac_small	243, 244	\XINT_flmul_chkopt	233
\XINT_FL_fac_smallloop_a	244, 245	\XINT_flmul_noopt	233
\XINT_FL_fac_smallloop_loop	246	\XINT_flmul_opt	233, 234
\XINT_FL_fac_smallloop_mul	246	\XINT_flmul_opt_a	234
\XINT_FL_fac_smallmul	246, 247	\XINT_float_chkopt	222
\XINT_FL_fac_smallmul_end	247	\XINT_float_neg	223
\XINT_FL_fac_toobig	243	\XINT_float_noopt	223
\XINT_FL_fac_vbig	243, 244	\XINT_float_opt	223
\XINT_FL_fac_vbigloop_a	244	\XINT_float_opt_a	223
\XINT_FL_fac_vbigloop_loop	244, 245	\XINT_float_pos	223
\XINT_FL_mul_a	234	\XINT_float_pos_done	223, 256
\XINT_FL_mul_b	234	\XINT_float_post	223
\XINT_FL_pfac_a	249	\XINT_float_zero	223
\XINT_FL_pfac_addzeroes	249	\XINT_floate_chkopt	255
\XINT_FL_pfac_addzeroes_exit	249	\XINT_floate_neg	255
\XINT_FL_pfac_b	249	\XINT_floate_noopt	255
\XINT_FL_pfac_bigloop	249, 250	\XINT_floate_opt	255
\XINT_FL_pfac_bigloop_a	250	\XINT_floate_opt_a	255
\XINT_FL_pfac_bigloop_b	250	\XINT_floate_pos	255, 256
\XINT_FL_pfac_end_	249--251	\XINT_floate_post	255
\XINT_FL_pfac_end_i	249--251	\XINT_floate_zero	255
\XINT_FL_pfac_end_ii	249--251	\XINT_flpfac_chkopt	247, 248
\XINT_FL_pfac_end_iii	249, 251	\XINT_flpfac_noopt	248
\XINT_FL_pfac_fork	248	\XINT_flpfac_opt	248
\XINT_FL_pfac_increaseP	248, 249, 252	\XINT_flpfac_opt_b	248
\XINT_FL_pfac_medloop	249, 250	\XINT_flpow_a	236
\XINT_FL_pfac_medloop_a	250	\XINT_flpow_aa	236
\XINT_FL_pfac_medloop_b	250	\XINT_flpow_ab	236
\XINT_FL_pfac_neg	248	\XINT_flpow_b	236
\XINT_FL_pfac_one	248	\XINT_flpow_BisZero	235
\XINT_FL_pfac_outofrange	248	\XINT_flpow_checkB_a	235
\XINT_FL_pfac_smallloop	249	\XINT_flpow_checkB_b	235, 236
\XINT_FL_pfac_smallloop_a	249	\XINT_flpow_checkB_c	236
\XINT_FL_pfac_smallloop_b	249	\XINT_flpow_checkB_d	236
\XINT_FL_pfac_vbigloop	249--251	\XINT_flpow_chkopt	235
\XINT_FL_pfac_vbigloop_a	250	\XINT_flpow_III	237, 238, 241, 242

\XINT_flpow_IIIend	238	\XINT_flsqrt_finish	254
\XINT_flpow_IItoIII	237, 238	\XINT_flsqrt_g	254
\XINT_flpow_ItoIII	237	\XINT_flsqrt_h	254
\XINT_flpow_loopI	236, 237	\XINT_flsqrt_noopt	253
\XINT_flpow_loopI_even	237	\XINT_flsqrt_opt	253
\XINT_flpow_loopI_odd	237	\XINT_flsqrt_opt_a	253
\XINT_flpow_loopII	237, 238	\XINT_flsub_chkopt	233
\XINT_flpow_loopII_even	237	\XINT_flsub_noopt	233
\XINT_flpow_loopII_odd	237	\XINT_flsub_opt	233
\XINT_flpow_loopII_odda	237, 238	\XINT_flsub_opt_a	233
\XINT_flpow_noopt	235	\XINT_fmax	217, 218
\XINT_flpow_opt	235	\XINT_fmax_A	218
\XINT_flpow_opt_a	235	\XINT_fmax_firstneg	218
\XINT_flpow_truncate	236--238, 241, 242	\XINT_fmax_minusminus	218
\XINT_flpow_truncate_a	236, 237	\XINT_fmax_nonneg_a	218
\XINT_flpow_zero	236, 241	\XINT_fmax_nonneg_b	218, 219
\XINT_flpower_a	241	\XINT_fmax_secondneg	218
\XINT_flpower_aa	241	\XINT_fmin	218
\XINT_flpower_ab	241	\XINT_fmin_A	219
\XINT_flpower_b	241	\XINT_fmin_firstneg	219
\XINT_flpower_BisZero	239, 240	\XINT_fmin_minusminus	219
\XINT_flpower_checkB_a	239, 240	\XINT_fmin_nonneg_a	219
\XINT_flpower_checkB_b	240	\XINT_fmin_nonneg_b	218, 219
\XINT_flpower_checkB_c	240	\XINT_fmin_secondneg	219
\XINT_flpower_checkB_d	240, 241	\XINT_fmud	209
\XINT_flpower_chkopt	239, 240	\XINT_fmud_a	209
\XINT_flpower_IItoIII	242	\XINT_fmud_b	209
\XINT_flpower_ItoIII	241	\XINT_fmud_c	209
\XINT_flpower_loopI	241	\XINT_fmud_d	209, 210
\XINT_flpower_loopI_even	241	\XINT_fmud_e	210
\XINT_flpower_loopI_odd	241	\XINT_fmud_zero	209, 210
\XINT_flpower_loopII	242	\XINT_fpow	210
\XINT_flpower_loopII_even	242	\XINT_fpow_fork	210
\XINT_flpower_loopII_odd	242	\XINT_fpow_neg	211
\XINT_flpower_loopII_odda	242	\XINT_fpow_pos	211
\XINT_flpower_noopt	240	\XINT_fpow_pos_A	211
\XINT_flpower_opt	240	\XINT_fpow_pos_B	211
\XINT_flpower_opt_a	240	\XINT_fpow_zero	210, 211
\XINT_flpower_toloopI	241	\XINT_fprdxpr	212
\XINT_flpower_toloopII	241, 242	\XINT_fprod_finished	212
\XINT_flpowerh_a	239	\XINT_fprod_loop_a	212
\XINT_flpowerh_b	239	\XINT_fprod_loop_b	212
\XINT_flpowerh_c	239	\XINT_fprod_loop_c	212
\XINT_flpowerh_d	239, 240	\XINT_frac_gen	182, 183
\XINT_flpowerh_e	239, 240	\XINT_frac_gen_A	183
\XINT_flpowerh_finish	240	\XINT_frac_gen_B	183
\XINT_flpowerh_i	239	\XINT_frac_gen_Ba	183
\XINT_flpowerh_int	239	\XINT_frac_gen_Bb	183
\XINT_flsqrt	254	\XINT_frac_gen_Bc	183
\XINT_flsqrt_a	254	\XINT_frac_gen_Bd	183
\XINT_flsqrt_again	254, 255	\XINT_frac_gen_C	183
\XINT_flsqrt_again_a	255	\XINT_frac_gen_Ca	184
\XINT_flsqrt_b	254, 255	\XINT_frac_gen_Cb	184
\XINT_flsqrt_c	254	\XINT_frac_gen_Cc	184
\XINT_flsqrt_chkopt	253	\XINT_frac_gen_F	184
\XINT_flsqrt_d	254	\XINT_frac_gen_G	184
\XINT_flsqrt_f	254	\XINT_frac_gen_Ga	184, 185

\XINT_frac_gen_Gdivbyzero	184	\XINT_infloat_Q	228
\XINT_frac_gen_Gdivbyzero_a	184	\XINT_infloat_Qq	228
\XINT_frac_gen_zero	185	\XINT_infloat_R	228
\XINT_fracfrac_A	191, 192	\XINT_infloat_Rq	228
\XINT_fracfrac_B	191	\XINT_infloat_Sa	228
\XINT_fracfrac_C	191	\XINT_infloat_Sb	228
\XINT_fracfrac_D	191	\XINT_infloat_SEq	228
\XINT_fracfrac_E	191	\XINT_infloat_sp	224
\XINT_fracisone	216	\XINT_infloat_sp_b	225
\XINT_fsqr	210	\XINT_infloat_sp_c	225
\XINT_fsqr_a	210	\XINT_infloat_sp_needzeros	225
\XINT_fsqr_b	210	\XINT_infloat_sp_quick	225
\XINT_fsqr_zero	210	\XINT_infloat_sp_round	225, 226
\XINT_fsub	209	\XINT_infloat_spneg	225
\XINT_fsub_a	209	\XINT_infloat_spneg_needzeros	225
\XINT_fsub_Azero	209	\XINT_infloat_spnegend	225
\XINT_fsub_b	209	\XINT_infloat_sppos	225
\XINT_fsum_finished	209	\XINT_infloat_spzero	224, 225
\XINT_fsum_loop_a	209	\XINT_infloat_SUp	228, 229
\XINT_fsum_loop_b	209	\XINT_infloat_SY	228
\XINT_fsum_loop_c	209	\XINT_infloat_X	228, 229
\XINT_fsumexpr	209	\XINT_infloat_Y	225, 229
\XINT_fwover_A	192, 193	\XINT_infloat_Z	229
\XINT_fwover_B	192	\XINT_infloat_ZZ	229
\XINT_fwover_C	192	\XINT_infloatdivmod	256
\XINT_fwover_D	192	\XINT_infloate	256
\XINT_geq_plusplus	217	\XINT_infloate_end	256
\xint_gob_til_dot	203	\XINT_infloatfracdg_a	231
\xint_gob_til_exclam	246	\XINT_infloatS_clean	224
\xint_gob_til_R	201	\XINT_infloatS_clean_a	224
\xint_gob_til_sc	247	\XINT_infrac	182
\xint_gob_til_xint:	209, 212, 218, 219	\XINT_infrac	180, 182, 188--194, 198, 200, 210, 212, 224
\xint_gob_til_zero	182, 185, 191, 192, 195, 197, 207--210, 216, 217, 232	\XINT_infrac_fork	182
\xint_gobble_i	204, 206, 207	\XINT_infrac_res_a	182
\xint_gobble_ii	191	\XINT_infrac_res_b	182
\xint_gobble_vi	205	\XINT_infrac_res_ca	182
\XINT_half	241, 242	\XINT_infrac_res_cb	182
\XINT_iffloatint	257	\XINT_infrac_res_cc	182
\XINT_ifint	195	\XINT_infrac_res_zero	182
\XINT_ifloor	190	\XINT_inrandomfloatS	257
\XINT_infloat	224	\XINT_inrandomfloatS_a	257, 258
\XINT_infloat_a	224	\XINT_inrandomfloatS_b	257
\XINT_infloat_clean	224	\XINT_inrandomfloatS_zero	257, 258
\XINT_infloat_clean_a	224	\XINT_iround_A	200
\XINT_infloat_done	224	\XINT_irr_D	194
\XINT_infloat_fork	224, 226	\XINT_irr_denomisone	194
\XINT_infloat_J	226	\XINT_irr_divisionbyzero	195
\XINT_infloat_K	226	\XINT_irr_finish	195, 197
\XINT_infloat_L	226	\XINT_irr_indeterminate	195
\XINT_infloat_Ma	226	\XINT_irr_loop_a	195
\XINT_infloat_Mb	226	\XINT_irr_loop_d	195
\XINT_infloat_MtoN	226, 227	\XINT_irr_loop_e	195
\XINT_infloat_N	227	\XINT_irr_loop_exit	195
\XINT_infloat_Oa	227	\XINT_irr_loop_exitb	195
\XINT_infloat_Ob	227	\XINT_irr_negative	194
\XINT_infloat_P	227	\XINT_irr_nonneg	194

\XINT_irr_start	194	\XINT_numer_A	190
\XINT_irr_zero	195	\XINT_numer_B	190
\XINT_is_One	198, 200, 224	\XINT_opp	181, 221
\XINT_isOne 188, 189, 191, 192, 194, 196, 241, 242		\xint_orthat	185,
\XINT_itrunc_G	198--200	189, 198--200, 213--215, 224, 228, 230, 232,	
\XINT_jrr_D	196	236, 241--243, 248, 249, 252, 254, 257, 258	
\XINT_jrr_denomisone	196	\XINT_outfrac	180, 210--212
\XINT_jrr_divisionbyzero	196	\XINT_outfrac_divisionbyzero	180
\XINT_jrr_indeterminate	196	\XINT_outfrac_N	180, 181
\XINT_jrr_loop_a	196, 197	\XINT_outfrac_N_a	181
\XINT_jrr_loop_b	196	\XINT_outfrac_P	180, 181
\XINT_jrr_loop_c	196, 197	\XINT_outfrac_Zero	181
\XINT_jrr_loop_d	197	\XINT_pfac_fork	211, 212
\XINT_jrr_loop_e	197	\XINT_pfloat_a	230
\XINT_jrr_loop_exit	197	\XINT_pfloat_chkopt	230
\XINT_jrr_negative	196	\XINT_pfloat_N	230
\XINT_jrr_nonneg	196	\XINT_pfloat_N_i	231
\XINT_jrr_start	196	\XINT_pfloat_N_ii	231
\XINT_jrr_zero	196	\XINT_pfloat_N_iii	231
\XINT_len_fork	180	\XINT_pfloat_N_iv	231
\XINT_length_loop	183, 184, 199	\XINT_pfloat_N_v	231
\XINT_maxof_a	218	\XINT_pfloat_neg	230
\XINT_maxof_b	218	\XINT_pfloat_no	230
\XINT_maxof_c	218	\XINT_pfloat_noopt	230
\XINT_maxof_d	218	\XINT_pfloat_opt	230
\XINT_maxof_e	218	\XINT_pfloat_opt_a	230
\XINT_minof_a	219	\XINT_pfloat_P	230, 231
\XINT_minof_b	219	\XINT_pfloat_P_	231
\XINT_minof_c	219	\XINT_pfloat_P_i	231
\XINT_minof_d	219	\XINT_pfloat_P_ii	231
\XINT_minof_e	219	\XINT_pfloat_P_iii	231
\XINT_mod_a	215	\XINT_pfloat_P_iv	231
\XINT_mod_aiszero	215	\XINT_pfloat_P_v	231
\XINT_mod_b	215	\XINT_pfloat_pos	230
\XINT_mod_bneg	215	\XINT_pfloat_Ps	230, 231
\XINT_mod_bpos	215	\XINT_pfloat_Psi	231
\XINT_mod_bpos_a	215, 216	\XINT_pfloat_Psii	231
\XINT_mod_D_a	216	\XINT_pfloat_Psiii	231
\XINT_mod_D_b	214--216	\XINT_pfloat_Psiv	231
\XINT_mod_D_c	216	\XINT_pfloat_Psv	231
\XINT_mod_D_exit	216	\XINT_pfloat_zero	230
\XINT_mod_divbyzero	215	\XINT_pirr_start	194
\XINT_mod_E	216	\xint_pow	210
\XINT_mod_F	216	\XINT_praw	188
\XINT_modtrunc_a	213	\XINT_praw_A	188
\XINT_modtrunc_aiszero	213, 215	\XINT_praw_a	188
\XINT_modtrunc_b	213	\XINT_providespackage	179
\XINT_modtrunc_bneg	213	\XINT_raw	188
\XINT_modtrunc_bpos	213	\XINT_rawz_A	189
\XINT_modtrunc_divbyzero	213, 215	\XINT_rawz_Ba	189
\XINT_modtrunc_pos	213, 214	\XINT_rawz_Bb	189
\XINT_modtrunc_pos_a	214	\XINT_rawz_fork	189
\XINT_mul_fork	196	\XINT_rep	204, 206, 207, 236, 241
\XINT_mul_out	244	\XINT_restorecatcodes_endinput	258
\XINT_num_cleanup	184	\XINT_rez_A	193
\XINT_num_loop	184	\XINT_rez_AB	193, 197
\XINT_numer	190	\XINT_rez_B	193

\XINT_rez_C	193	\XINT_trunc_sp_Ca	198
\XINT_rez_D	193	\XINT_trunc_sp_Cb	198
\XINT_rez_E	193	\XINT_trunc_sp_Cc	198, 199
\XINT_rez_neg	193	\XINT_trunc_sp_Cd	199
\XINT_rez_zero	193	\XINT_trunc_sp_D	198, 199
\XINT_round	200	\XINT_trunc_sp_E	199
\XINT_round_A	200	\XINT_trunc_sp_F	199
\XINT_round_a	200	\XINT_trunc_zero	198, 200
\XINT_round_aa	200	\xint_UDsignfork	191, 192, 194, 196, 198, 199, 204, 206, 213--215, 217, 221, 226, 238
\XINT_round_B	200, 201	\xint_UDsignsfork	218--220
\XINT_rsepbyviii_end_A	203	\xint_UDXINTWfork	182, 183
\XINT_rsepbyviii_end_B	203	\xint_UDzerominusfork 184, 193, 197, 201, 207, 209, 210, 223-- -225, 230, 235, 236, 240, 241, 243, 253, 255
\xint_secondofthree	180	\xint_UDzerosfork	195, 196, 220
\xint_secondofthree_thenstop	186	\XINT_W	182--184
\xint_secondoftwo	188, 232, 247	\XINT_xtrunc_a	201
\xint_secondoftwo_thenstop ...	187, 188, 195, 257	\XINT_xtrunc_b	201
\XINT_sepandrev_andcount	203	\XINT_xtrunc_BisEight	202
\XINT_sepbyviii_Z_end	202	\XINT_xtrunc_BisFive	202
\XINT_Sgn	181, 198, 200	\XINT_xtrunc_BisFour	202
\XINT_sgn	221	\XINT_xtrunc_BisOne	201, 202
\XINT_sgnfrac_a	191	\XINT_xtrunc_BisSmall	202
\XINT_sgnfrac_b	191	\XINT_xtrunc_BisTwo	201, 202
\XINT_sgnfrac_N	191, 192	\XINT_xtrunc_c	201
\XINT_sgnfrac_P	191, 192	\XINT_xtrunc_d	201
\XINT_sgnfwover_a	192	\XINT_xtrunc_e	202, 203
\XINT_sgnfwover_b	192	\XINT_xtrunc_finish	205
\XINT_sgnfwover_N	192, 193	\XINT_xtrunc_finish_a	206
\XINT_sgnfwover_P	192, 193	\XINT_xtrunc_I	203
\XINT_signalcondition	181, 184, 195, 196, 213, 214, 236, 243, 248, 252, 253	\XINT_xtrunc_I_a	203
\XINT_split_fromleft 204, 206, 226--228, 236, 254		\XINT_xtrunc_I_b	203, 204
\XINT_split_fromright_a	199	\XINT_xtrunc_IA_c	204
\XINT_split_xfork	204, 207	\XINT_xtrunc_IA_d	204, 205
\XINT_sqrt_start	254, 255	\XINT_xtrunc_IA_xd	204
\XINT_T	182, 184	\XINT_xtrunc_IA_xe	204
\XINT_tfrac_fork	197	\XINT_xtrunc_IAB_e	204
\XINT_tfrac_P	197	\XINT_xtrunc_IB_c	204
\XINT_tfrac_zero	197	\XINT_xtrunc_IB_d	204, 205
\xint_thirdofthree	180	\XINT_xtrunc_II	203, 205
\xint_thirdofthree_thenstop	186, 187	\XINT_xtrunc_II_a	205
\XINT_trunc	198	\XINT_xtrunc_II_b	205
\XINT_trunc_a	198	\XINT_xtrunc_II_c	205
\XINT_trunc_B	198	\XINT_xtrunc_loop	204, 205
\XINT_trunc_b	198, 200	\XINT_xtrunc_loop_a	205
\XINT_trunc_C	198	\XINT_xtrunc_loop_b	205
\XINT_trunc_CE	198	\XINT_xtrunc_prepare_a	201
\XINT_trunc_D	198, 199	\XINT_xtrunc_prepare_b	201, 202
\XINT_trunc_E	198, 199	\XINT_xtrunc_prepare_c	203
\XINT_trunc_F	199	\XINT_xtrunc_prepare_d	203
\XINT_trunc_G	198--200	\XINT_xtrunc_prepare_e	203
\XINT_trunc_H	199	\XINT_xtrunc_prepare_f	203
\XINT_trunc_Ha	200	\XINT_xtrunc_prepare_g	203
\XINT_trunc_Haa	200	\XINT_xtrunc_prepare_small	201
\XINT_trunc_Hb	200	\XINT_xtrunc_small_a	202
\XINT_trunc_sp_B	198		
\XINT_trunc_sp_b	198, 200		
\XINT_trunc_sp_C	198		

\XINT_xtrunc_sp_e	202, 206	\xintFloatAdd	232
\XINT_xtrunc_sp_I	206	\xintfloatadd	232
\XINT_xtrunc_sp_I_a	206	\xintFloatBinomial	251
\XINT_xtrunc_sp_IA_b	206	\xintfloatbinomial	251
\XINT_xtrunc_sp_IA_c	206, 207	\xintFloatDiv	234
\XINT_xtrunc_sp_IAA	206	\xintfloatdiv	234
\XINT_xtrunc_sp_IAB	206, 207	\xintFloatE	255
\XINT_xtrunc_sp_IB_b	206, 207	\xintfloate	255
\XINT_xtrunc_sp_IB_c	207	\xintFloatFac	242
\XINT_xtrunc_sp_II	206, 207	\xintfloatfac	242
\XINT_xtrunc_transition	205	\xintFloatMul	233
\XINT_xtrunc_zero	201	\xintfloatmul	233
\XINT_Z	183	\xintFloatPFactorial	247
\XINT_zeroes_forviii	203	\xintfloatpfactorial	247
\xintAbs	221	\xintFloatPow	235
\xintabs	216, 221	\xintfloatpow	235
\xintAdd	207, 232	\xintFloatPower	239
\xintadd	207, 209	\xintfloatpower	239
\xintBinomial	211	\xintFloatSqrt	253
\xintbinomial	211	\xintfloatsqrt	253
\xintCeil	190	\xintFloatSub	233
\xintceil	190	\xintfloatsub	233
\xintCmp	186, 187, 219	\xintFloor	190
\xintcmp	219	\xintfloor	190
\xintdecsplit	200	\xintFrac	191
\xintDecToString	189	\xintfrac	191
\xintdectostring	189	\xintFwOver	192
\xintDenominator	190	\xintfwover	192
\xintdenominator	190	\xintGeq	216
\XINTdigits	207, 223, 230- -235, 239, 240, 243, 248, 252, 253, 255--257	\xintgeq	216
\xintDigits	207	\xintGt	186
\xintDiv	212, 213, 252	\xintgt	186
\xintdiv	212	\xintGtorEq	186
\xintDivFloor	213	\xinthalf	240
\xintdivfloor	213, 256	\xintiCeil	190
\xintDivMod	214	\xinticeil	190
\xintdivmod	214, 256	\xintifCmp	186
\xintDivRound	213	\xintifcmp	186
\xintdivround	213	\xintifEq	187
\xintDivTrunc	213	\xintifeq	186, 187
\xintdivtrunc	213	\xintifFloatInt	257
\xintDouble	229, 239	\xintiffloatint	257
\xintdouble	202	\xintifGt	187
\xintDSH	229	\xintifgt	186, 187
\xintdsrr	239	\xintifInt	195
\xintE	193	\xintifint	195
\xinte	193	\xintiFloor	190
\xintEq	186	\xintifloor	190, 213
\xinteq	186	\xintifLt	187
\xintEven	186	\xintiflt	186, 187
\xinteven	186	\xintifNotZero	187
\xintFac	211	\xintifnotzero	187
\xintfac	211	\xintifOdd	188
\xintFloat	222	\xintifodd	188
\xintfloat	222, 230, 232--235, 239, 242, 247, 251, 253	\xintifOne	187
		\xintifone	187
		\xintifSgn	186

<code>\xintifsgn</code>	186	<code>\XINTinfloatS</code>	224, 232--235, 239, 240
<code>\xintifZero</code>	187	<code>\XINTinFloatSqrt</code>	240, 253
<code>\xintifzero</code>	187	<code>\XINTinfloatsqrt</code>	253
<code>\xintiiadd</code>	196, 208	<code>\XINTinFloatSub</code>	233
<code>\xintiiCmp</code>	216	<code>\XINTinfloatsub</code>	233
<code>\xintiidivision</code>	215	<code>\XINTinRandomFloatS</code>	257
<code>\xintiiDivRound</code>	254	<code>\XINTinrandomfloatS</code>	257
<code>\xintiiE</code>	216	<code>\XINTinRandomFloatSdigits</code>	257
<code>\xintiiE</code>	189	<code>\XINTinRandomFloatSixteen</code>	258
<code>\xintiiiflt</code>	229	<code>\xintinum</code>	254
<code>\xintiiifZero</code>	189	<code>\xintipfactorial</code>	211
<code>\xintiiMul</code>	208, 216, 229, 234	<code>\xintipow</code>	210
<code>\xintiimul</code>		<code>\xintiRound</code>	200
	202, 208, 210, 212, 216, 217, 220, 238, 242	<code>\xintiround</code>	200, 213
<code>\xintiiOdd</code>	241	<code>\xintIrr</code>	194
<code>\xintiiopp</code>	190, 197, 213--215	<code>\xintirr</code>	194
<code>\xintiipow</code>	211	<code>\xintIsNotZero</code>	186
<code>\xintiiQuo</code>	208, 216	<code>\xintisnotzero</code>	186
<code>\xintiiquo</code>	190, 195, 208, 216, 227	<code>\xintIsOne</code>	187, 216
<code>\xintiiRem</code>	195	<code>\xintisone</code>	216
<code>\xintiiirem</code>	197, 214, 216	<code>\xintIsZero</code>	186
<code>\xintiisqr</code>	210, 237, 238, 241, 242	<code>\xintiszero</code>	186
<code>\xintiisub</code>	254	<code>\xintiTrunc</code>	198
<code>\xintInc</code>	229	<code>\xintitrunc</code>	198, 200, 239
<code>\xintinc</code>	229	<code>\xintJrr</code>	196
<code>\XINTinFloat</code>	224, 256, 257	<code>\xintjrr</code>	196
<code>\XINTinfloat</code>	223, 224, 231--	<code>\xintLDg</code>	186, 239, 241, 242
	-233, 236, 241, 242, 248, 251--253, 255, 256	<code>\xintLen</code>	180
<code>\XINTinFloatAdd</code>	232	<code>\xintlen</code>	180
<code>\XINTinfloatadd</code>	232	<code>\xintLength</code>	
<code>\XINTinFloatBinomial</code>	251		204--207, 217, 220, 225--227, 236, 237, 240
<code>\XINTinfloatbinomial</code>	251	<code>\xintlength</code>	199
<code>\XINTinFloatDiv</code>	234	<code>\xintLt</code>	186
<code>\XINTinfloatdiv</code>	234	<code>\xintlt</code>	186
<code>\XINTinFloatDivFloor</code>	256	<code>\xintLtorEq</code>	186
<code>\XINTinfloatdivfloor</code>	256	<code>\xintMax</code>	217
<code>\XINTinFloatDivMod</code>	256	<code>\xintmax</code>	217, 218
<code>\XINTinfloatdivmod</code>	256	<code>\xintMaxof</code>	218
<code>\XINTinFloatE</code>	256	<code>\xintmaxof</code>	218
<code>\XINTinfloate</code>	256	<code>\xintMin</code>	218
<code>\XINTinFloatFac</code>	242	<code>\xintmin</code>	218, 219
<code>\XINTinfloatfac</code>	242	<code>\xintMinof</code>	219
<code>\XINTinFloatFracdigits</code>	231	<code>\xintminof</code>	219
<code>\XINTinfloatfracdigits</code>	231	<code>\xintMod</code>	215, 256
<code>\XINTinFloatMod</code>	256	<code>\xintmod</code>	215
<code>\XINTinfloatmod</code>	256	<code>\xintModTrunc</code>	213
<code>\XINTinFloatMul</code>	233	<code>\xintmodtrunc</code>	213
<code>\XINTinfloatmul</code>	233	<code>\xintMul</code>	209
<code>\XINTinFloatPFFactorial</code>	247	<code>\xintmul</code>	209, 212
<code>\XINTinfloatpfactorial</code>	247, 248	<code>\xintNotEq</code>	186
<code>\XINTinFloatPow</code>	235	<code>\xintNum</code>	186, 210--212, 243, 248, 252
<code>\XINTinfloatpow</code>	235	<code>\xintnum</code>	200, 210, 240
<code>\XINTinFloatPower</code>	239	<code>\xintNumerator</code>	190
<code>\XINTinfloatpower</code>	239	<code>\xintnumerator</code>	190
<code>\XINTinFloatPowerH</code>	239	<code>\xintOdd</code>	186, 188
<code>\XINTinfloatpowerh</code>	239	<code>\xintodd</code>	186
<code>\XINTinFloatS</code>	224	<code>\xintOpp</code>	190, 221, 233

Y

Z

12.8 Index of **xintseries**

A	
\aftergroup	259, 260
C	
\catcode	259
\csname	259
E	
\endcsname	259
\endinput	259, 260
\endlinechar	259
I	
\immediate	259
N	
\numexpr	259--267
P	
\ProvidesPackage	259, 260
R	
\romannumeral	260--267
W	
\w	259
\write	259
X	
\x	259
\xint:	265--267
\xint_afterfi	260--267
\xint_c_i	264--266
\XINT_flpowseries	266
\XINT_flpowseries_chkopt	265
\XINT_flpowseries_dont_i	266
\XINT_flpowseries_dont_ii	266
\XINT_flpowseries_exit_i	266, 267
\XINT_flpowseries_exit_ii	267
\XINT_flpowseries_loop_i	266
\XINT_flpowseries_loop_ii	266, 267
\XINT_flpowseries_loop_pre	266, 267
\XINT_flpowseries_noopt	266
\XINT_flpowseries_opt	265, 266
\XINT_flpowseriesx	267
\XINT_flpowseriesx_chkopt	267
\XINT_flpowseriesx_noopt	267
\XINT_flpowseriesx_opt	267
\XINT_fppowseries	264
\XINT_fppowseries_dont_i	264
\XINT_fppowseries_dont_ii	264
\XINT_fppowseries_exit_i	264, 265
\XINT_fppowseries_exit_ii	265
\XINT_fppowseries_loop_i	264
\XINT_fppowseries_loop_ii	264, 265
\XINT_fppowseries_loop_pre	264, 265
\XINT_fppowseriesx	265
\XINT_fppowseriesx_pre	265
\xint_gobble_ii	260, 261
\XINT_iseries	260
\XINT_iseries_exit	261
\XINT_iseries_loop	261
\XINT_powseries	261
\XINT_powseries_exit_i	261
\XINT_powseries_exit_ii	262
\XINT_powseries_loop_i	261, 262
\XINT_powseries_loop_ii	261
\XINT_powseriesx	262
\XINT_powseriesx_pre	262
\XINT_providespackage	260
\XINT_ratseries	262
\XINT_ratseries_exit_i	263
\XINT_ratseries_exit_ii	263
\XINT_ratseries_exit_iii	263
\XINT_ratseries_loop	263, 264
\XINT_ratseriesx	263
\XINT_ratseriesx_pre	263, 264
\XINT_restorecatcodes_endinput	267
\XINT_series	260
\XINT_series_exit	260
\XINT_series_loop	260
\xintadd	260, 261, 263
\XINTdigits	266, 267
\xintfloat	266
\xintfloatadd	267
\xintFloatPowerSeries	265
\xintfloatpowerseries	265
\xintFloatPowerSeriesX	267
\xintfloatpowerseriesx	267
\xintFxpPtPowerSeries	264
\xintfxptpowerseries	264
\xintFxpPtPowerSeriesX	265
\xintfxptpowerseriesx	265
\xintiadd	261, 265
\XINTinfloatadd	266
\XINTinfloatmul	266, 267
\XINTinfloatpow	266, 267
\xintiSeries	260
\xintiseries	260
\xintiTrunc	265
\xintitrunc	264
\xintMul	263--265
\xintmul	261--263
\xintPow	262, 264, 265
\xintPowerSeries	261
\xintpowerseries	261
\xintPowerSeriesX	262
\xintpowerseriesx	262

<code>\xintRationalSeries</code>	262	<code>\xinttrunc</code>	264, 265
<code>\xintRationalSeriesX</code>	263		
<code>\xintratseries</code>	262	Y	
<code>\xintratseriesx</code>	263	<code>\y</code>	259
<code>\xintSeries</code>	260		
<code>\xintseries</code>	260	Z	
<code>\XINTsetupcatcodes</code>	260	<code>\z</code>	259, 260

12.9 Index of [xintcfrac](#)

A	
\aftergroup	268, 269
C	
\catcode	268
\cfrac	270--273
\csname	268, 269, 271, 272
E	
\endcsname	268, 269, 271, 272
\endinput	268, 269
\endlinechar	268
H	
\hfill	269--272
I	
\immediate	268
K	
\krof	276, 277
N	
\numexpr	268, 285--289
P	
\ProvidesPackage	269
R	
\romannumeral	269--289
W	
\w	268, 269
\write	268
X	
\x	268, 269
\xint:	269--272, 278--284
\xint_afterfi	285--288
\xint_c_	285, 286
\xint_c_i	287--289
\XINT_cfrac_A	269, 270
\XINT_cfrac_B	270
\XINT_cfrac_C	270
\XINT_cfrac_D	270
\XINT_cfrac_end	270
\XINT_cfrac_end_b	270
\XINT_cfrac_integer	270
\XINT_cfrac_loop_a	270
\XINT_cfrac_loop_d	270
\XINT_cfrac_loop_e	270
\xint_cfrac_loop_exit	270
\XINT_cfrac_loop_f	270
\XINT_cfrac_noopt	269
\XINT_cfrac_opt_a	269
\XINT_cfrac_opt_b	269
\XINT_cfrac_optc	269
\XINT_cfrac_optl	269
\XINT_cfrac_optr	270
\XINT_cfrac_T	270
\XINT_cntcs	287
\XINT_cntcs_exit	287
\XINT_cntcs_exit_b	287
\XINT_cntcs_loop	287
\XINT_cntf	285
\XINT_cntf_exit	286
\XINT_cntf_loop	285, 286
\XINT_cntgc	288
\XINT_cntgc_exit	288
\XINT_cntgc_exit_b	288
\XINT_cntgc_loop	288
\XINT_cstc_end	289
\XINT_cstc_loop_a	289
\XINT_cstc_loop_b	289
\XINT_cstc_prep	289
\XINT_ctcv_end	281, 282
\XINT_ctcv_loop_a	281, 282
\XINT_ctcv_loop_b	281
\XINT_ctcv_loop_c	281
\XINT_ctcv_loop_d	281
\XINT_ctcv_loop_e	281
\XINT_ctcv_loop_f	281, 282
\XINT_ctcv_loop_g	282
\XINT_ctcv_prep	281
\XINT_ctf_end	277, 278
\XINT_ctf_loop_a	277, 278
\XINT_ctf_loop_b	277
\XINT_ctf_loop_c	278
\XINT_ctf_loop_d	278
\XINT_ctf_loop_e	278
\XINT_ctf_prep	277
\XINT_div_prepare	270, 273--276
\XINT_fgtc_a	275
\XINT_fgtc_b	275
\XINT_fgtc_c	275
\XINT_fgtc_d	275, 276
\XINT_fgtc_da	275
\XINT_fgtc_e	275
\XINT_fgtc_f	275
\XINT_fgtc_g	275
\XINT_fgtc_h	275
\xint_firstofone	275
\XINT_ftc_A	273
\XINT_ftc_B	273
\XINT_ftc_C	273
\XINT_ftc_D	273
\XINT_ftc_integer	273
\XINT_ftc_loop_a	273, 274
\XINT_ftc_loop_d	273
\XINT_ftc_loop_e	273

<code>\xint_ftc_loop_exit</code>	274	<code>\XINT_gctcv_loop_i</code>	283
<code>\XINT_ftc_loop_f</code>	274	<code>\XINT_gctcv_loop_j</code>	283, 284
<code>\XINT_ftcc_A</code>	276	<code>\XINT_gctcv_loop_k</code>	284
<code>\XINT_ftcc_B</code>	276	<code>\XINT_gctcv_loop_l</code>	284
<code>\XINT_ftcc_C</code>	276	<code>\XINT_gctcv_loop_m</code>	284
<code>\XINT_ftcc_D</code>	276	<code>\XINT_gctcv_prep</code>	282
<code>\XINT_ftcc_En</code>	276	<code>\XINT_gctf_end</code>	279, 280
<code>\XINT_ftcc_end</code>	277	<code>\XINT_gctf_loop_a</code>	279, 280
<code>\XINT_ftcc_Ep</code>	276	<code>\XINT_gctf_loop_b</code>	279
<code>\XINT_ftcc_integer</code>	276	<code>\XINT_gctf_loop_c</code>	279
<code>\XINT_ftcc_loop_a</code>	276, 277	<code>\XINT_gctf_loop_d</code>	279
<code>\XINT_ftcc_loop_b</code>	276	<code>\XINT_gctf_loop_e</code>	279
<code>\XINT_ftcc_loop_c</code>	276, 277	<code>\XINT_gctf_loop_f</code>	279
<code>\XINT_ftcc_loop_d</code>	277	<code>\XINT_gctf_loop_g</code>	279
<code>\XINT_ftcc_loop_N</code>	277	<code>\XINT_gctf_loop_h</code>	279
<code>\XINT_ftcc_loop_P</code>	277	<code>\XINT_gctf_loop_i</code>	280
<code>\XINT_ftcx_A</code>	274	<code>\XINT_gctf_loop_j</code>	280
<code>\XINT_ftcx_B</code>	274	<code>\XINT_gctf_prep</code>	279
<code>\XINT_ftcx_C</code>	274	<code>\XINT_gctgc_end</code>	289, 290
<code>\XINT_ftcx_D</code>	274	<code>\XINT_gctgc_end_b</code>	290
<code>\XINT_ftcx_integer</code>	274	<code>\XINT_gctgc_loop_a</code>	289
<code>\XINT_ftcx_loop_a</code>	274	<code>\XINT_gctgc_loop_b</code>	289, 290
<code>\XINT_ftcx_loop_d</code>	274	<code>\XINT_gctgc_loop_c</code>	289
<code>\XINT_ftcx_loop_e</code>	274	<code>\XINT_gctgc_start</code>	289
<code>\xint_ftcx_loop_exit</code>	274	<code>\XINT_gctgcx_end</code>	273
<code>\XINT_ftcx_loop_f</code>	274	<code>\XINT_gctgcx_loop_a</code>	273
<code>\XINT_gcfrac</code>	271	<code>\XINT_gctgcx_loop_b</code>	273
<code>\XINT_gcfrac_end</code>	271	<code>\XINT_gctgcx_start</code>	273
<code>\XINT_gcfrac_end_b</code>	271	<code>\XINT_ggcfrac</code>	272
<code>\XINT_gcfrac_endloop</code>	271	<code>\XINT_ggcfrac_end</code>	272
<code>\XINT_gcfrac_enter</code>	271	<code>\XINT_ggcfrac_end_b</code>	272, 273
<code>\XINT_gcfrac_loop</code>	271	<code>\XINT_ggcfrac_endloop</code>	272
<code>\XINT_gcfrac_noopt</code>	271	<code>\XINT_ggcfrac_enter</code>	272
<code>\XINT_gcfrac_opt_a</code>	270	<code>\XINT_ggcfrac_loop</code>	272
<code>\XINT_gcfrac_opt_b</code>	271	<code>\XINT_ggcfrac_noopt</code>	272
<code>\XINT_gcfrac_optc</code>	271	<code>\XINT_ggcfrac_opt_a</code>	272
<code>\XINT_gcfrac_optl</code>	271	<code>\XINT_ggcfrac_opt_b</code>	272
<code>\XINT_gcfrac_optr</code>	271	<code>\XINT_ggcfrac_optc</code>	272
<code>\XINT_gcfrac_T</code>	271	<code>\XINT_ggcfrac_optl</code>	272
<code>\XINT_gcfrac_U</code>	271	<code>\XINT_ggcfrac_optr</code>	272
<code>\XINT_gcntf</code>	286	<code>\XINT_ggcfrac_T</code>	272
<code>\XINT_gcntf_exit</code>	286, 287	<code>\XINT_ggcfrac_U</code>	272
<code>\XINT_gcntf_loop</code>	286, 287	<code>\xint_gob_til_exclam</code>	271--273, 277--284, 289
<code>\XINT_gcntgc</code>	288	<code>\xint_gob_til_Z</code>	270
<code>\XINT_gcntgc_exit</code>	288, 289	<code>\xint_gob_til_zero</code>	270, 273, 274
<code>\XINT_gcntgc_exit_b</code>	289	<code>\xint_gobble_ii</code>	286, 287
<code>\XINT_gcntgc_loop</code>	288, 289	<code>\xint_gobble_iii</code>	275
<code>\XINT_gcntgc_loop_b</code>	289	<code>\XINT_icstcv_end</code>	282
<code>\XINT_gctcv_end</code>	283, 284	<code>\XINT_icstcv_loop_a</code>	282
<code>\XINT_gctcv_loop_a</code>	283, 284	<code>\XINT_icstcv_loop_b</code>	282
<code>\XINT_gctcv_loop_b</code>	283	<code>\XINT_icstcv_loop_c</code>	282
<code>\XINT_gctcv_loop_c</code>	283	<code>\XINT_icstcv_loop_d</code>	282
<code>\XINT_gctcv_loop_d</code>	283	<code>\XINT_icstcv_loop_e</code>	282
<code>\XINT_gctcv_loop_e</code>	283	<code>\XINT_icstcv_prep</code>	282
<code>\XINT_gctcv_loop_f</code>	283	<code>\XINT_icstf_end</code>	278
<code>\XINT_gctcv_loop_g</code>	283	<code>\XINT_icstf_loop_a</code>	278
<code>\XINT_gctcv_loop_h</code>	283	<code>\XINT_icstf_loop_b</code>	278

[illegible]

12.10 Index of xintexpr

Particularly for this package, there are quite a few macros which are defined via constructs such as `\expandafter \def \csmame ... \endcsname`, and their indexing needs some extra mark-up which is yet to be added to the commented source code.

Symbols	
<code>\%</code>	373
A	
<code>\active</code>	331
<code>\aftergroup</code>	293
C	
<code>\catcode</code>	293, 294, 301, 308--311, 318, 319, 328, 331, 332, 334--336, 344, 347, 365, 366, 368, 373--375
<code>\count</code>	299, 300
<code>\csmame</code>	293, 294, 296, 297, 300, 301, 305, 308--318, 321, 322, 324--332, 334, 335, 337--339, 341--347, 351--361, 364--366, 372
D	
<code>\detokenize</code>	327, 329, 331, 368--370
<code>\dimen</code>	299, 300
<code>\dimexpr</code>	300
<code>\do</code>	309--311, 316--318, 326, 329, 364
<code>\dp</code>	300
E	
<code>\edef</code> .	319, 331, 332, 334, 335, 364, 365, 373, 374
<code>\endcsname</code>	293, 294, 296, 297, 300--306, 308--318, 321, 322, 324--332, 334, 335, 337--339, 341--347, 351--361, 364--366, 372
<code>\endinput</code>	293
<code>\endlinechar</code>	293, 373
<code>\escapechar</code>	373, 374
<code>\everyeof</code>	373
F	
<code>\fontchardp</code>	300
<code>\fontcharht</code>	300
<code>\fontcharic</code>	300
<code>\fontcharwd</code>	300
<code>\fontdimen</code>	300
G	
<code>\glueexpr</code>	300
H	
<code>\ht</code>	300
I	
<code>\if</code> 294, 298--308, 314, 319--321, 327, 337--339, 341--348, 350, 357, 362, 363, 368--370, 372	
<code>\ifcat</code>	299, 302--305, 307--309
<code>\ifcsmame</code>	308, 309, 329, 330, 332
<code>\ifxintexprsafecatcodes</code>	374
<code>\ifxintglobaldefs</code> ...	331, 332, 334, 364, 366, 374
<code>\ifxintverbose</code>	331, 332, 334, 364, 366, 374
<code>\immediate</code>	293
K	
<code>\krof</code>	309, 310, 312, 314, 317--320, 324, 326, 327, 368
L	
<code>\lccode</code>	300, 331
<code>\lowercase</code>	300, 331
M	
<code>\meaning</code>	364, 366, 374
N	
<code>\newif</code>	374
<code>\noexpand</code> .	295, 319, 320, 331, 332, 334, 372--374
<code>\numexpr</code> .	293, 294, 296, 297, 299, 300, 302--304, 319--324, 332, 338, 341--343, 345, 347, 364
P	
<code>\protected</code>	295
<code>\ProvidesPackage</code>	293, 294
R	
<code>\repeat</code>	332
<code>\romannumeral</code> 294--299, 301--306, 308--310, 312, 314, 315, 317, 318, 320--327, 329, 331, 335--351, 354--358, 361--366, 368, 369, 372, 373	
S	
<code>\scantokens</code>	373, 374
T	
<code>\t</code>	293
<code>\thexintboolexpr</code>	297
<code>\thexintexpr</code>	295
<code>\thexintfloatexpr</code>	295
<code>\thexintiexpr</code>	295
<code>\thexintiexpr</code>	295
<code>\to</code>	331
<code>\toks</code>	365, 373, 374
U	
<code>\unexpanded</code>	364
<code>\unless</code>	319, 320, 374
W	
<code>\W</code>	298
<code>\w</code>	293
<code>\write</code>	293
X	
<code>\x</code>	293

\xint	364	\XINT_allexpr_rrseq	344
\xint:	299	\XINT_allexpr_rseq	340
\XINT:expr:macrofunc	366, 371	\XINT_allexpr_seqx	337
\XINT:expr:one:and:opt	350, 354, 355, 371	\XINT_allexpr_subx	340
\XINT:expr:one:or:two:nums	371	\XINT_andof:_a	348
\XINT:expr:randrange	357	\XINT_andof:_c	348
\XINT:expr:tacitzeroifonearg ..	350, 354, 355, 371	\XINT_andof:_e	348
\XINT:expr:totwo	351, 369	\XINT_andof:_no	348
\XINT:expr:two:to:one	351, 356--358, 371	\XINT_applyon::_a	368
\XINT:expr:two:to:two	351, 355, 371	\XINT_applyon::_b	368
\XINT:expr:userfunc	365, 371	\XINT_applyon::_c	368
\XINT:flexpr:two:to:one	351, 356, 357, 371	\XINT_applyon::_d	368
\XINT:flexpr:two:to:two	351, 355, 371	\XINT_applyon::_e	368
\XINT:iiexpr:one:or:two:	371	\XINT_applyon::_end	368
\XINT:iiexpr:randrange	357	\XINT_applyon::_a	368
\XINT:iiexpr:tacitzeroifonearg ...	350, 355, 371	\XINT_applyon::_b	368
\XINT:NE:csv	369, 371	\XINT_applyon::_c	368
\XINT:NE:LApply::csv	368	\XINT_applyon::_d	368
\XINT:NE:one	369, 371	\XINT_applyon::_e	368
\XINT:NE:one_a	369	\XINT_applyon::_end	368
\XINT:NE:oneopt	369	\XINT_boolexpr_done	297
\XINT:NE:oneopt_a	369	\XINT_boolexpr_print	295, 297, 372
\XINT:NE:oneopt_b	369	\xint_Bye	321, 322
\XINT:NE:oneopt_fork	369	\xint_bye	321, 327, 328, 336, 337
\XINT:NE:RApply::csv	368	\xint_c_	299, 308, 310, 319, 320, 337
\XINT:NE:RLApply::csv	368	\xint_c_i	303, 304, 310, 311, 318,
\XINT:NE:two	368--371	323, 324, 337, 338, 341--343, 345, 347, 364	
\XINT:NE:two_	368, 369, 371	\xint_c_ii	311, 317, 319, 326
\XINT:NE:two_a	368	\xint_c_iii	318
\XINT:NE:two_b	368	\xint_c_ix	301--304, 307, 308
\XINT:NE:two_fork_dd	368	\xint_c_mone	337
\XINT:NE:two_fork_nn	368	\xint_c_vii	328
\XINT:NEhook:csv	298, 358--361, 371	\xint_c_viii	311, 321
\XINT:NEhook:one	298, 311, 318, 327, 328, 351--354, 357, 371	\xint_c_x	302, 327, 328
\XINT:NEhook:two	298, 312, 314, 339, 371	\xint_c_xviii ..	300, 301, 307, 311, 336, 339, 340
\XINT:newexpr:insertdollar	369, 370	\XINT_Cmp	294, 325, 363
\XINT:newexpr:macrofunc	366, 371	\XINT_csv::_a	298
\XINT:newexpr:macrofunc:a	366	\XINT_csv::_b	298
\XINT:newexpr:one:and:opt	369, 371	\XINT_csv::_c	298
\XINT:newexpr:one:or:two:nums	371	\XINT_csv::_d	298
\XINT:newexpr:tacitzeroifonearg	369, 371	\xint_ddfork	368
\XINT:newexpr:two:to:one	370, 371	\xint_dothis	298--309, 314, 319,
\XINT:newexpr:two:to:two	369, 371	320, 329, 338, 339, 341, 343--347, 368--371	
\XINT:newexpr:userfunc	365, 371	\xint_exchangetwo_keepbraces	314
\XINT:newflexpr:two:to:one	370, 371	\XINT_expr:_unlock	343, 346, 347
\XINT:newflexpr:two:to:two	369, 371	\XINT_expr_binop_inline	314, 315
\XINT:newiiexpr:one:or:two:	371	\XINT_expr_binop_inline_a	314
\XINT:newiiexpr:tacitzeroifonearg	369, 371	\XINT_expr_binop_inline_b	314
\XINT_	364	\XINT_expr_binop_inline_c	314
\XINT::_:end	299	\XINT_expr_binop_inline_d	314
\XINT::_:end	298	\XINT_expr_binop_inline_e	314
\xint_afterfi	305, 306, 309, 310, 312, 314,	\XINT_expr_binop_inline_end	314
315, 317, 318, 320, 326, 327, 348, 362--364		\XINT_expr_binopwrđ	308
\XINT_allexpr_iter	342	\XINT_expr_countetc	299, 300
\XINT_allexpr_iterr	345	\XINT_expr_defbin_b	312, 313
\XINT_allexpr_opx	339	\XINT_expr_defbin_c	312, 313
		\XINT_expr_deflistopl_b	315, 316

\XINT_expr_deflistopl_c	315	\XINT_expr_func_round	354, 355
\XINT_expr_deflistopr_b	314, 315	\XINT_expr_func_rrseq	344
\XINT_expr_deflistopr_c	314	\XINT_expr_func_rseq	340
\XINT_expr_defuserfunc	364, 365	\XINT_expr_func_seqx	337
\XINT_expr_defvar	331, 332	\XINT_expr_func_sgn	352
\XINT_expr_defvar_getname	331	\XINT_expr_func_sqr	352
\XINT_expr_defvar_one	331	\XINT_expr_func_sqrt	354
\XINT_expr_done	295, 309, 310	\XINT_expr_func_subx	340
\XINT_expr_extra_)	309	\XINT_expr_func_trunc	355
\XINT_expr_foundend	308, 311, 335	\XINT_expr_func_unknown	329
\XINT_expr_foundop	309	\XINT_expr_func_xor	360
\XINT_expr_foundop_a	308, 309	\XINT_expr_getnext	296, 299, 300, 310, 312, 314, 315, 317, 318, 326--328
\XINT_expr_func	306, 307	\XINT_expr_getnext_a	299
\XINT_expr_func_!	353	\XINT_expr_getnextfork	300
\XINT_expr_func_?	353	\XINT_expr_getop	300, 301, 305, 308, 309, 311, 328--330, 337, 339--342, 344, 346
\XINT_expr_func_@@	335	\XINT_expr_getop_a	308
\XINT_expr_func_@@@	335	\XINT_expr_getop_b	308
\XINT_expr_func_@@@	335	\XINT_expr_gobz_a	301
\XINT_expr_func_abs	351	\XINT_expr_gobz_scandec_a	303
\XINT_expr_func_all	360	\XINT_expr_gobz_scandec_b	303, 304
\XINT_expr_func_any	360	\XINT_expr_gobz_scandec_c	303, 304
\XINT_expr_func_binomial	356	\XINT_expr_gobz_scandec_endbycs	303, 304
\XINT_expr_func_break	329	\XINT_expr_gobz_scanint_b	301--303
\XINT_expr_func_ceil	352	\XINT_expr_gobz_scanint_c	302
\XINT_expr_func_divmod	355	\XINT_expr_gobz_scanint_d	302, 303
\XINT_expr_func_even	353	\XINT_expr_gobz_scanint_endbycs	302
\XINT_expr_func_factorial	354	\XINT_expr_gobz_startdec_a	302, 303
\XINT_expr_func_first	360	\XINT_expr_gotnil	301
\XINT_expr_func_float	355	\XINT_expr_inhex	294
\XINT_expr_func_floor	352	\XINT_expr_iter:A	342, 343
\XINT_expr_func_frac	352	\XINT_expr_iter:a	342, 343
\XINT_expr_func_gcd	358	\XINT_expr_iter:aa	342
\XINT_expr_func_if	361	\XINT_expr_iter:Abort	343
\XINT_expr_func_ifint	361	\XINT_expr_iter:abort	343
\XINT_expr_func_ifone	361	\XINT_expr_iter:b	343
\XINT_expr_func_ifsgn	361, 362	\XINT_expr_iter:Break	343
\XINT_expr_func_iter	342	\XINT_expr_iter:break	343
\XINT_expr_func_iterr	345	\XINT_expr_iter:c	343
\XINT_expr_func_last	360	\XINT_expr_iter:D	343
\XINT_expr_func_lcm	358	\XINT_expr_iter:d	343
\XINT_expr_func_len	360	\XINT_expr_iter:E	343
\XINT_expr_func_max	358	\XINT_expr_iter:end	343
\XINT_expr_func_min	359	\XINT_expr_iter:Goon	343
\XINT_expr_func_mod	356	\XINT_expr_iter:goon	343
\XINT_expr_func_not	353	\XINT_expr_iter:noop	343
\XINT_expr_func_num	351	\XINT_expr_iter:Omit	343
\XINT_expr_func_nuple	354	\XINT_expr_iter:omit	343
\XINT_expr_func_odd	353	\XINT_expr_iterr:A	346, 347
\XINT_expr_func_opxadd	339	\XINT_expr_iterr:a	346
\XINT_expr_func_opxmul	339	\XINT_expr_iterr:aa	346
\XINT_expr_func_pfactorial	356	\XINT_expr_iterr:Abort	347
\XINT_expr_func_preduce	351	\XINT_expr_iterr:abort	346
\XINT_expr_func_quo	357	\XINT_expr_iterr:b	346
\XINT_expr_func_randrange	357	\XINT_expr_iterr:Break	347
\XINT_expr_func_reduce	351	\XINT_expr_iterr:break	346
\XINT_expr_func_rem	358		
\XINT_expr_func_reversed	361		

<code>\XINT_expr_iterr:_c</code>	346	<code>\XINT_expr_op_:]</code>	318
<code>\XINT_expr_iterr:_D</code>	347	<code>\XINT_expr_op_?</code>	327, 328
<code>\XINT_expr_iterr:_d</code>	346	<code>\XINT_expr_op_??</code>	327, 328
<code>\XINT_expr_iterr:_E</code>	347	<code>\XINT_expr_op_?a</code>	327, 328
<code>\XINT_expr_iterr:_end</code>	346	<code>\XINT_expr_op_?checka</code>	327
<code>\XINT_expr_iterr:_Goon</code>	347	<code>\XINT_expr_op_?checkb</code>	327
<code>\XINT_expr_iterr:_goon</code>	346	<code>\XINT_expr_op_?checkc</code>	327
<code>\XINT_expr_iterr:_noop</code>	346	<code>\XINT_expr_op_[</code>	328
<code>\XINT_expr_iterr:_Omit</code>	347	<code>\XINT_expr_op_[[:</code>	319
<code>\XINT_expr_iterr:_omit</code>	346	<code>\XINT_expr_op__</code>	307, 330
<code>\XINT_expr_iterrx</code>	345, 346	<code>\XINT_expr_precedence_</code>	308, 331, 334, 335
<code>\XINT_expr_iterrx</code>	346	<code>\XINT_expr_precedence_!</code>	328
<code>\XINT_expr_iterx</code>	342	<code>\XINT_expr_precedence_!?</code>	311
<code>\XINT_expr_itory</code>	342	<code>\XINT_expr_precedence_:]</code>	318
<code>\XINT_expr_itself_[</code>	319	<code>\XINT_expr_precedence_?</code>	308, 327
<code>\XINT_expr_lockit</code>	294, 331, 341, 343--347	<code>\XINT_expr_precedence_[</code>	328
<code>\XINT_expr_lockscan</code>	294	<code>\XINT_expr_precedence_[[:</code>	318
<code>\XINT_expr_macrofunc_</code>	366	<code>\XINT_expr_precedence_a</code>	311
<code>\XINT_expr_makedummy</code>	334	<code>\XINT_expr_print</code>	295, 372
<code>\XINT_expr_missing_)</code>	310	<code>\XINT_expr_redefinemacros</code>	371, 373
<code>\XINT_expr_newfunction</code>	365, 366	<code>\XINT_expr_redefineprints</code>	372
<code>\XINT_expr_onliteral_@</code>	335	<code>\XINT_expr_rrseq:_A</code>	344, 345
<code>\XINT_expr_onliteral_`</code>	301	<code>\XINT_expr_rrseq:_a</code>	344
<code>\XINT_expr_onliteral_add</code>	339	<code>\XINT_expr_rrseq:_aa</code>	344
<code>\XINT_expr_onliteral_add_f</code>	339	<code>\XINT_expr_rrseq:_Abort</code>	345
<code>\XINT_expr_onliteral_bool</code>	329	<code>\XINT_expr_rrseq:_abort</code>	345
<code>\XINT_expr_onliteral_mul</code>	339	<code>\XINT_expr_rrseq:_b</code>	344, 345
<code>\XINT_expr_onliteral_mul_f</code>	339	<code>\XINT_expr_rrseq:_Break</code>	345
<code>\XINT_expr_onliteral_nil</code>	334	<code>\XINT_expr_rrseq:_break</code>	345
<code>\XINT_expr_onliteral_protect</code>	329	<code>\XINT_expr_rrseq:_c</code>	344
<code>\XINT_expr_onliteral_qfloat</code>	330	<code>\XINT_expr_rrseq:_D</code>	345
<code>\XINT_expr_onliteral_qfrac</code>	330	<code>\XINT_expr_rrseq:_d</code>	344, 345
<code>\XINT_expr_onliteral_qint</code>	330	<code>\XINT_expr_rrseq:_E</code>	345
<code>\XINT_expr_onliteral_qrand</code>	330	<code>\XINT_expr_rrseq:_end</code>	344
<code>\XINT_expr_onliteral_qraw</code>	330	<code>\XINT_expr_rrseq:_Goon</code>	345
<code>\XINT_expr_onliteral_random</code>	330	<code>\XINT_expr_rrseq:_goon</code>	345
<code>\XINT_expr_onliteral_seq</code>	336	<code>\XINT_expr_rrseq:_noop</code>	344
<code>\XINT_expr_onliteral_seq_a</code>	336, 339, 340, 342, 344, 345	<code>\XINT_expr_rrseq:_Omit</code>	345
<code>\XINT_expr_onliteral_seq_b</code>	336	<code>\XINT_expr_rrseq:_omit</code>	345
<code>\XINT_expr_onliteral_seq_c</code>	336	<code>\XINT_expr_rrseqx</code>	344
<code>\XINT_expr_onliteral_seq_d</code>	336	<code>\XINT_expr_rrseqy</code>	344
<code>\XINT_expr_onliteral_seq_e</code>	336	<code>\XINT_expr_rseq:_A</code>	341
<code>\XINT_expr_onliteral_seq_f</code>	336	<code>\XINT_expr_rseq:_a</code>	341
<code>\XINT_expr_onliteral_subs</code>	340	<code>\XINT_expr_rseq:_aa</code>	341
<code>\XINT_expr_onliteral_subs_f</code>	340	<code>\XINT_expr_rseq:_Abort</code>	341, 342
<code>\XINT_expr_onliteral_togl</code>	329	<code>\XINT_expr_rseq:_abort</code>	341
<code>\XINT_expr_op:_a</code>	339	<code>\XINT_expr_rseq:_b</code>	341
<code>\XINT_expr_op:_b</code>	339	<code>\XINT_expr_rseq:_Break</code>	341, 342
<code>\XINT_expr_op:_c</code>	339	<code>\XINT_expr_rseq:_break</code>	341
<code>\XINT_expr_op:_d</code>	339	<code>\XINT_expr_rseq:_c</code>	341
<code>\XINT_expr_op:_end</code>	339	<code>\XINT_expr_rseq:_D</code>	341, 342
<code>\XINT_expr_op:_noop</code>	339	<code>\XINT_expr_rseq:_d</code>	341
<code>\XINT_expr_op_</code>	312, 314, 317, 326, 327	<code>\XINT_expr_rseq:_E</code>	341
<code>\XINT_expr_op_!</code>	328	<code>\XINT_expr_rseq:_end</code>	341
<code>\XINT_expr_op_!?</code>	335	<code>\XINT_expr_rseq:_Goon</code>	341
<code>\XINT_expr_op:_</code>	318	<code>\XINT_expr_rseq:_goon</code>	341
		<code>\XINT_expr_rseq:_noop</code>	341

\XINT_expr_rseq:_Omit	341, 342	\XINT_expr_seq:_goon	338
\XINT_expr_rseq:_omit	341	\XINT_expr_seq:_noop	338
\XINT_expr_rseqx	340	\XINT_expr_seq:_Omit	338
\XINT_expr_rseqy	340, 341	\XINT_expr_seq:_omit	338
\XINT_expr_scan_macropar	300	\XINT_expr_seq_empty?	324, 325, 338
\XINT_expr_scan_nbr_or_func	300, 301	\XINT_expr_startdec	300, 301
\XINT_expr_scandec_a	301, 302	\XINT_expr_startdec_a	302
\XINT_expr_scandec_b	302, 303	\XINT_expr_startint	301
\XINT_expr_scandec_c	303	\XINT_expr_subexpr	299, 300
\XINT_expr_scandec_d	303	\XINT_expr_subx:_end	340
\XINT_expr_scandec_endbycs	303	\XINT_expr_tmpa	331--333, 364--366
\XINT_expr_scanexp_a	302--304	\XINT_expr_tmpb	331, 364, 365
\XINT_expr_scanexp_b	304	\XINT_expr_tmpc	331, 332, 364
\XINT_expr_scanexp_bb	304	\XINT_expr_tmpd	331, 332
\XINT_expr_scanexp_c	304	\XINT_expr_tmpval	331, 332
\XINT_expr_scanexp_cb	304	\XINT_expr_tmpvar	331, 332
\XINT_expr_scanexp_d	304	\XINT_expr_undefined	332, 333
\XINT_expr_scanexp_db	304	\XINT_expr_unexpectedtoken	294
\XINT_expr_scanexp_endbycs	304	\XINT_expr_unknown_function	329
\XINT_expr_scanexp_endbycs_b	304	\XINT_expr_unknown_operator	309
\XINT_expr_scanexpr_endbycs	304	\XINT_expr_unknown_variable	330
\XINT_expr_scanfunc	301, 306	\XINT_expr_unlock	294--297, 311, 312, 314, 315, 317, 318, 325--329, 331, 335--339, 341--347, 351--361, 365, 366, 372
\XINT_expr_scanfunc_a	306, 307	\XINT_expr_unlock_a	294
\XINT_expr_scanfunc_b	306, 307	\XINT_expr_unlock_hex_in	294, 305
\XINT_expr_scanhex_I	301, 305	\XINT_expr_unlock_sp	297, 372
\XINT_expr_scanhex_transition	305	\XINT_expr_unpackvar	300
\XINT_expr_scanhexI_a	305	\XINT_expr_until	312, 314, 317, 326, 327
\XINT_expr_scanhexI_aa	305	\XINT_expr_until:_a	318
\XINT_expr_scanhexI_b	305	\XINT_expr_until:_b	318
\XINT_expr_scanhexI_bgob	305	\XINT_expr_until_end_a	296
\XINT_expr_scanhexII_a	305, 306	\XINT_expr_usethe	295, 297
\XINT_expr_scanhexII_aa	306	\XINT_expr_var_@	335
\XINT_expr_scanhexII_b	306	\XINT_expr_var_abort	335
\XINT_expr_scanhexII_bgob	306	\XINT_expr_var_nil	334
\XINT_expr_scanint_a	301	\XINT_expr_var_omit	335
\XINT_expr_scanint_b	301, 302	\XINT_expr_wrap	295, 296, 365
\XINT_expr_scanint_c	302	\XINT_expr_wrapit	365
\XINT_expr_scanint_d	302	\xint_firstofone	349, 350, 372
\XINT_expr_scanint_endbycs	302	\xint_firstofthree	308
\XINT_expr_scanop_a	308, 311	\xint_firstoftwo	303, 307, 330, 350, 357, 369, 372
\XINT_expr_scanop_b	308	\XINT_flexpr_func_!	353
\XINT_expr_scanop_c	308, 309	\XINT_flexpr_func_?	353
\XINT_expr_scanop_d	308	\XINT_flexpr_func_@@	335
\XINT_expr_seq:_A	337, 338	\XINT_flexpr_func_@@@	335
\XINT_expr_seq:_a	337, 338	\XINT_flexpr_func_@@@	335
\XINT_expr_seq:_aa	337	\XINT_flexpr_func_abs	351
\XINT_expr_seq:_Abort	338	\XINT_flexpr_func_all	360
\XINT_expr_seq:_abort	338	\XINT_flexpr_func_any	360
\XINT_expr_seq:_b	338	\XINT_flexpr_func_binomial	356
\XINT_expr_seq:_Break	338	\XINT_flexpr_func_break	329
\XINT_expr_seq:_break	338	\XINT_flexpr_func_ceil	352
\XINT_expr_seq:_c	338	\XINT_flexpr_func_divmod	355
\XINT_expr_seq:_D	338	\XINT_flexpr_func_even	353
\XINT_expr_seq:_d	338	\XINT_flexpr_func_factorial	354
\XINT_expr_seq:_E	338	\XINT_flexpr_func_first	360
\XINT_expr_seq:_end	338		
\XINT_expr_seq:_Goon	338		

\XINT_flexpr_func_float	355	\XINT_flseqb::csv_z	326
\XINT_flexpr_func_floor	352	\XINT_flseqb:f:csv	363
\XINT_flexpr_func_frac	352	\XINT_flseqb:f:csv_a	363
\XINT_flexpr_func_gcd	358	\XINT_flseqb:f:csv_bg	363
\XINT_flexpr_func_if	361	\XINT_flseqb:f:csv_bl	363
\XINT_flexpr_func_ifint	361	\XINT_flseqb:f:csv_n	364
\XINT_flexpr_func_ifone	361	\XINT_flseqb:f:csv_na	363, 364
\XINT_flexpr_func_ifsgn	362	\XINT_flseqb:f:csv_p	363
\XINT_flexpr_func_iter	342	\XINT_flseqb:f:csv_pa	363
\XINT_flexpr_func_iterr	345	\XINT_flseqb:f:csv_pb	363
\XINT_flexpr_func_last	360	\XINT_flseqb:f:csv_pc	363
\XINT_flexpr_func_lcm	358	\XINT_global	331, 332, 334, 365, 366, 373
\XINT_flexpr_func_len	360	\xint_gob_til_!	294, 296, 299, 327
\XINT_flexpr_func_max	358	\xint_gob_til_minus	321
\XINT_flexpr_func_min	359	\xint_gob_til_W	298
\XINT_flexpr_func_mod	356	\xint_gobble_i	296--299, 308, 309, 314, 321, 328--330, 333, 338, 346, 347, 362, 363, 373
\XINT_flexpr_func_not	353	\xint_gobble_iii	295, 296, 300, 319, 320, 324, 325
\XINT_flexpr_func_num	351	\xint_gobble_iv	297, 346, 347, 362--364
\XINT_flexpr_func_nuple	354	\XINT_iexpr_noopt	296
\XINT_flexpr_func_odd	353	\XINT_iexpr_withopt	296
\XINT_flexpr_func_opxadd	339	\XINT_iexpr_wrap	296
\XINT_flexpr_func_opxmud	339	\XINT_iiexpr_func_!	353
\XINT_flexpr_func_pfactorial	356	\XINT_iiexpr_func_?	353
\XINT_flexpr_func_preduce	351	\XINT_iiexpr_func_@@	336
\XINT_flexpr_func_quo	357	\XINT_iiexpr_func_@@@	336
\XINT_flexpr_func_randrange	357	\XINT_iiexpr_func_@@@	336
\XINT_flexpr_func_reduce	351	\XINT_iiexpr_func_abs	351
\XINT_flexpr_func_rem	358	\XINT_iiexpr_func_all	360
\XINT_flexpr_func_reversed	361	\XINT_iiexpr_func_any	360
\XINT_flexpr_func_round	355	\XINT_iiexpr_func_binomial	356
\XINT_flexpr_func_rrseq	344	\XINT_iiexpr_func_break	329
\XINT_flexpr_func_rseq	340	\XINT_iiexpr_func_ceil	352
\XINT_flexpr_func_seqx	337	\XINT_iiexpr_func_divmod	355
\XINT_flexpr_func_sgn	352	\XINT_iiexpr_func_even	353
\XINT_flexpr_func_sqr	353	\XINT_iiexpr_func_factorial	354
\XINT_flexpr_func_sqrt	354	\XINT_iiexpr_func_first	360
\XINT_flexpr_func_subx	340	\XINT_iiexpr_func_float	355
\XINT_flexpr_func_trunc	355	\XINT_iiexpr_func_floor	352
\XINT_flexpr_func_xor	360	\XINT_iiexpr_func_gcd	358
\XINT_flexpr_noopt	297, 372	\XINT_iiexpr_func_if	361
\XINT_flexpr_op_!	328	\XINT_iiexpr_func_ifint	361
\XINT_flexpr_op_! ?	335	\XINT_iiexpr_func_ifone	361
\XINT_flexpr_op_:]	318	\XINT_iiexpr_func_ifsgn	362
\XINT_flexpr_op_?	328	\XINT_iiexpr_func_iter	342
\XINT_flexpr_op_[.....	328	\XINT_iiexpr_func_iterr	345
\XINT_flexpr_op_][:	319	\XINT_iiexpr_func_last	360
\XINT_flexpr_op__	330	\XINT_iiexpr_func_lcm	358
\XINT_flexpr_print	297	\XINT_iiexpr_func_len	360
\XINT_flexpr_until_end_a	296	\XINT_iiexpr_func_max	358
\XINT_flexpr_withopt_a	297	\XINT_iiexpr_func_min	359
\XINT_flexpr_withopt_b	297, 372	\XINT_iiexpr_func_mod	356
\XINT_flexpr_wrap	297, 372	\XINT_iiexpr_func_not	353
\XINT_flseqa::csv	324	\XINT_iiexpr_func_num	351
\XINT_flseqb::csv	325	\XINT_iiexpr_func_nuple	354
\XINT_flseqb::csv_a	325	\XINT_iiexpr_func_odd	353
\XINT_flseqb::csv_n	325	\XINT_iiexpr_func_opxadd	339
\XINT_flseqb::csv_p	325		

\XINT_iiexpr_func_opxmuls	339	\XINT_isbalanced_yes	337
\XINT_iiexpr_func_pfactorial	357	\XINT_istrue::_a	298
\XINT_iiexpr_func_quo	357	\XINT_istrue::_b	298, 299
\XINT_iiexpr_func_randrange	357	\XINT_istrue::_c	298
\XINT_iiexpr_func_rem	358	\XINT_istrue::_d	298, 299
\XINT_iiexpr_func_reversed	361	\XINT_istrue::_e	298, 299
\XINT_iiexpr_func_round	355	\XINT_keep:x:csv_finish	321
\XINT_iiexpr_func_rrseq	344	\XINT_keep:x:csv_loop	321
\XINT_iiexpr_func_rseq	340	\XINT_keep:x:csv_loop_pickeight	321
\XINT_iiexpr_func_seqx	337	\XINT_keep:x:csv_pos	321
\XINT_iiexpr_func_sgn	352	\XINT_listsel::_:	320
\XINT_iiexpr_func_sqr	353	\XINT_listsel::_a	320
\XINT_iiexpr_func_sqrt	354	\XINT_listsel:_N:N	320
\XINT_iiexpr_func_sqrtr	354	\XINT_listsel:_N:N_a	320
\XINT_iiexpr_func_subx	340	\XINT_listsel:_N:N_abort	320
\XINT_iiexpr_func_trunc	355	\XINT_listsel:_N:P	320
\XINT_iiexpr_func_xor	360	\XINT_listsel:_N:P_a	320, 321
\XINT_iiexpr_op_!	328	\XINT_listsel:_nth	319, 320
\XINT_iiexpr_op_!?	335	\XINT_listsel:_O:P	321
\XINT_iiexpr_op:_]	318	\XINT_listsel:_P:N	320, 321
\XINT_iiexpr_op_?	328	\XINT_listsel:_P:N_a	321
\XINT_iiexpr_op_[328	\XINT_listsel:_P:O	321
\XINT_iiexpr_op_][:	319	\XINT_listsel:_P:P	320, 321
\XINT_iiexpr_op__	330	\XINT_listsel:_s	319, 320
\XINT_iiexpr_print	295, 372	\XINT_listxsel::_:	319
\XINT_iiexpr_until_end_a	296	\XINT_listxsel:_N:N	319
\XINT_iiexpr_wrap	295, 296	\XINT_listxsel:_N:N_a	319
\XINT_iiseq::csv	323	\XINT_listxsel:_N:P	319
\XINT_iiseq::csv_n	323, 324	\XINT_listxsel:_N:P_a	319, 320
\XINT_iiseq::csv_p	323	\XINT_listxsel:_O:P	320
\XINT_iiseq::csv_z	323	\XINT_listxsel:_P:N	319, 320
\XINT_iiseqa::csv	324	\XINT_listxsel:_P:N_a	320
\XINT_iiseqb::csv	325	\XINT_listxsel:_P:O	320
\XINT_iiseqb::csv_a	325	\XINT_listxsel:_P:P	319, 320
\XINT_iiseqb::csv_n	325	\XINT_NewExpr	372, 373
\XINT_iiseqb::csv_p	325	\XINT_NewExpr_a	373
\XINT_iiseqb::csv_z	325	\XINT_newexpr_clean	372
\XINT_iiseqb:f:csv	363	\XINT_NewFloatFunc	365, 373
\XINT_iiseqb:f:csv_a	363	\XINT_NewFunc	364, 373
\XINT_iiseqb:f:csv_bg	363	\XINT_newfunc_clean	373
\XINT_iiseqb:f:csv_bl	363	\XINT_NewIIFunc	365, 373
\XINT_iiseqb:f:csv_n	363	\XINT_oncsv:_a	348--350
\XINT_iiseqb:f:csv_na	363	\XINT_oncsv:_b	348
\XINT_iiseqb:f:csv_p	363	\XINT_oncsv:_c	348
\XINT_iiseqb:f:csv_pa	363	\XINT_oncsv:_d	348
\XINT_iiseqb:f:csv_pb	363	\XINT_oncsv:_e	348
\XINT_iiseqb:f:csv_pc	363	\XINT_oncsv:_empty	348
\XINT_infloat::_b	299	\XINT_oncsv:_end	348
\XINT_infloat::_c	299	\XINT_orof:_a	348
\XINT_infloat::_d	299	\XINT_orof:_c	348
\XINT_infloat::_e	299	\XINT_orof:_e	348
\XINT_isbalanced_a	336, 337	\XINT_orof:_yes	348
\XINT_isbalanced_b	336, 337	\xint_orthat	298--309, 314, 319, 320, 329, 338, 339, 341, 343--347, 369--371
\XINT_isbalanced_c	337	\XINT_pfloat::_b	299
\XINT_isbalanced_d	337	\XINT_pfloat::_c	299
\XINT_isbalanced_error	337	\XINT_pfloat::_d	299
\XINT_isbalanced_no	337		

\XINT_pfloat::_e	299	\XINT_xorof:_b	348
\XINT_pfloat_opt	299	\XINT_xorof:_c	348
\XINT_protectii	295, 297	\XINT_xorof:_e	348
\XINT_providespackage	294	\xint_zapspaces	296, 297, 308, 328
\XINT_restorecatcodes_endinput	375	\xint_zapspaces_o	331, 332, 364, 365
\XINT_round::_b	299	\xintAbs	351
\XINT_round::_c	299	\xintAdd	339
\XINT_round::_d	299	\xintadd	325, 349, 362
\XINT_round::_e	299	\xintANDof:csv	348, 360
\xint_secondofthree	308	\xintApply	331
\xint_secondoftwo 303, 330, 350, 357, 368, 369, 372		\xintapply	344, 346
\XINT_seq::csv	322	\xintApply::csv	368
\XINT_seq::csv_e	323, 324	\xintApply::csv	368
\XINT_seq::csv_n	323	\xintApplyInline	327
\XINT_seq::csv_p	322, 323	\xintApplyUnbraced	334
\XINT_seq::csv_z	322, 323	\xintapplyunbraced	346, 347
\XINT_seqa::csv	324	\xintAssignArray	331
\XINT_seqa::csv_a	324	\xintbareeval	295, 296, 332, 339, 340, 342, 344, 345, 366
\XINT_seqb::csv	324	\xintbarefloateval	296,
\XINT_seqb::csv_a	324	297, 332, 339, 340, 342, 344, 345, 366, 372	
\XINT_seqb::csv_n	325	\xintbareieval	295, 296, 332, 339, 340, 342, 344, 345, 366
\XINT_seqb::csv_p	324	\xintBinomial	356
\XINT_seqb::csv_z	325	\xintBool	329
\XINT_seqb:f:csv	362	\xintboolexpr	297
\XINT_seqb:f:csv_a	362	\xintCeil	352
\XINT_seqb:f:csv_be	362, 363	\xintCSV::csv	295, 298, 372
\XINT_seqb:f:csv_bg	362	\xintCSVLength	331
\XINT_seqb:f:csv_bl	362	\xintCSVtoList	331
\XINT_seqb:f:csv_n	362	\xintCSVtoListNonStripped	344, 346, 365, 366
\XINT_seqb:f:csv_na	362	\xintcsvtolistnonstripped	365, 366
\XINT_seqb:f:csv_nb	362--364	\xintdeffloatfunc	364
\XINT_seqb:f:csv_nc	362--364	\xintdeffloatfunc_a	364, 365
\XINT_seqb:f:csv_p	362	\xintdeffloatvar	332
\XINT_seqb:f:csv_pa	362	\xintdeffloatvar_a	332
\XINT_seqb:f:csv_pb	362	\xintdeffunc	364
\XINT_seqb:f:csv_pc	362	\xintdeffunc_a	364
\XINT_spraw	298	\xintdefiifunc	364
\XINT_spraw::_a	298	\xintdefiifunc_a	364, 365
\XINT_spraw::_b	298	\xintdefiivar	332
\XINT_spraw::_c	298	\xintdefiivar_a	332
\XINT_spraw::_d	298	\xintdefvar	332
\XINT_spraw::_e	298	\xintdefvar_a	332
\XINT_spraw_a	298	\XINTdigits	324, 325, 350, 363, 372
\XINT_spraw_p	298	\xintDivMod	355
\XINT_thecoords_a	296	\xintE	328
\XINT_thecoords_b	296	\xintError:ignored	294
\XINT_thecoords_c	296	\xintError:inserted	310
\xint_thirdofthree	308	\xintError:missing_xintthe!	295
\XINT_tmpa	309, 310,	\xintError:removed	309, 329, 330
317, 318, 326, 327, 329, 364, 365, 373--375		\xintError:replaced_by_zero	333
\XINT_tmpb	326, 373--375	\xintError:we_are_doomed	336
\XINT_tmpc	375	\xinteval	295--297
\xint_UDsignfork		\xintEven	353
.... 309, 310, 312, 314, 317, 318, 326, 327		\xintexpr	295
\xint_UDsignsfork	319, 320	\xintexprRestoreCatcodes	331, 364, 374
\xint_UDzerominusfork	324		
\XINT_xorof:_a	348		

<code>\xintexprSafeCatcodes</code>	332, 364, 373, 374	<code>\xintiMaxof:csv</code>	349, 358
<code>\xintexprsafecatcodesfalse</code>	374	<code>\xintiimin</code>	349
<code>\xintexprsafecatcodestrue</code>	374	<code>\xintiMinof:csv</code>	349, 359
<code>\xintFac</code>	328, 354	<code>\xintiMod</code>	356
<code>\xintFirstItem:f:csv</code>	360	<code>\xintiMul</code>	294, 339
<code>\xintfloateval</code>	295, 297	<code>\xintiimul</code>	349
<code>\xintfloatexpr</code>	295	<code>\xintiOdd</code>	353
<code>\xintFloor</code>	352	<code>\xintiOpp</code>	327
<code>\xintFor</code>	309--311, 316--318, 326, 329, 364	<code>\xintiPFactorial</code>	357
<code>\xintgcd</code>	349	<code>\xintiPow</code>	294
<code>\xintGCDof:csv</code>	349, 358	<code>\xintiPrd:csv</code>	349, 360
<code>\xintHexToDec</code>	294	<code>\xintiQuo</code>	358
<code>\xintiCeil</code>	311, 322, 352	<code>\xintiRandRange</code>	357, 372
<code>\xintieval</code>	295, 296	<code>\xintiRandRangeAtoB</code>	357, 372
<code>\xintiexpr</code>	295	<code>\xintiRem</code>	358
<code>\xintifboolexpr</code>	297	<code>\xintiSeq::csv</code>	323, 371
<code>\xintifboolfloatexpr</code>	297	<code>\xintiiseq::csv</code>	323
<code>\xintifboolliexpr</code>	297	<code>\xintiSeqA::csv</code>	324
<code>\xintifCmp</code>	324, 325, 362--364	<code>\xintiSeqB::csv</code>	325, 371
<code>\xintifFloatInt</code>	361, 370	<code>\xintiSeqB:f:csv</code>	363
<code>\xintifFloatInt:</code>	361, 371	<code>\xintiSeqBNumeric::csv</code>	371
<code>\xintifFloatIntNE:</code>	370, 371	<code>\xintiSeqNumeric::csv</code>	371
<code>\xintifInt</code>	361, 370	<code>\xintiSgn</code>	352
<code>\xintifInt:</code>	361, 371	<code>\xintiSqr</code>	353
<code>\xintifIntNE:</code>	370, 371	<code>\xintiSqrt</code>	354
<code>\xintiFloor</code>	322, 352	<code>\xintiSqrtR</code>	354
<code>\xintifnotzero</code>	297	<code>\xintiSum:csv</code>	349, 359
<code>\xintifOne</code>	361, 370	<code>\XINTinFloat</code>	350, 355
<code>\xintifOne:</code>	361, 371	<code>\XINTinfloat</code>	299, 324, 325, 363
<code>\xintifOneNE:</code>	370, 371	<code>\XINTinFloat::csv</code>	297, 299
<code>\xintiiAbs</code>	351	<code>\XINTinFloatAdd</code>	339
<code>\xintiiAdd</code>	339	<code>\XINTinfloatadd</code>	325, 350, 363, 364
<code>\xintiiadd</code>	325, 349, 363	<code>\XINTinFloatBinomial</code>	356
<code>\xintiiBinomial</code>	356	<code>\XINTinFloatdigits</code>	330, 350, 355
<code>\xintiiCmp</code>	294	<code>\XINTinFloatDivMod</code>	355
<code>\xintiiDivMod</code>	355	<code>\XINTinFloatE</code>	328
<code>\xintiiE</code>	328	<code>\XINTinFloatFac</code>	328, 350, 354
<code>\xintiiEval</code>	295, 296	<code>\XINTinFloatFacdigits</code>	350, 354
<code>\xintiiEven</code>	354	<code>\XINTinFloatFracdigits</code>	352
<code>\xintiiexpr</code>	295	<code>\XINTinFloatMaxof:csv</code>	350, 359
<code>\xintiiFac</code>	328, 354	<code>\XINTinFloatMinof:csv</code>	350, 359
<code>\xintiigcd</code>	350	<code>\XINTinFloatMod</code>	356
<code>\xintiiGCDof:csv</code>	350, 358	<code>\XINTinFloatMul</code>	339, 353
<code>\xintiiifNotZero</code>	327, 348, 361, 370	<code>\XINTinfloatmul</code>	350
<code>\xintiiifNotZero:</code>	361, 371	<code>\XINTinFloatPFactorial</code>	357
<code>\xintiiifNotZeroNE:</code>	370, 371	<code>\XINTinFloatPrd:csv</code>	350, 359
<code>\xintiiifOne</code>	361, 370	<code>\XINTinFloatSeq::csv</code>	371
<code>\xintiiifOne:</code>	361, 371	<code>\XINTinFloatSeqA::csv</code>	324
<code>\xintiiifOneNE:</code>	370, 371	<code>\XINTinFloatSeqB::csv</code>	325, 371
<code>\xintiiifSgn</code>	328, 361, 371	<code>\XINTinFloatSeqB:f:csv</code>	363
<code>\xintiiifSgn:</code>	361, 371	<code>\XINTinFloatSeqBNumeric::csv</code>	371
<code>\xintiiifSgnNE:</code>	370, 371	<code>\XINTinFloatSeqNumeric::csv</code>	371
<code>\xintiiIsNotZero</code>	353	<code>\XINTinFloatSqr</code>	353
<code>\xintiiIsZero</code>	353	<code>\XINTinFloatSqrt</code>	350, 354
<code>\xintiilcm</code>	350	<code>\XINTinFloatSqrtdigits</code>	350, 354
<code>\xintiilCMof:csv</code>	350, 358	<code>\XINTinFloatSum:csv</code>	350, 359
<code>\xintiimax</code>	349	<code>\XINTinRandomFloatSdigits</code>	330, 371

\XINTinRandomFloatSixteen	330, 371	\xintRelaxArray	361
\xintiNum	330	\xintReverse:f:csv	361
\xintiQuo	357	\xintReverseOrder	346, 347
\xintiRem	358	\xintRevWithBraces	344, 346
\xintiRound	354, 355	\xintRound	354
\xintiround	299	\xintround	299
\xintIrr	351	\xintRound::csv	296, 299, 372
\xintisnotzero	299	\xintSeq::csv	322, 371
\xintIsTrue::csv	295, 298, 372	\xintseq::csv	322
\xintiTrunc	355	\xintSeqA::csv	324
\xintKeep:f:csv	319, 320	\xintSeqB::csv	324, 371
\xintKeep:x:csv	319, 321	\xintSeqB:f:csv	362
\xintLastItem:f:csv	360	\xintSeqBNumeric::csv	371
\xintlcm	349	\xintSeqNumeric::csv	371
\xintLCMof:csv	349, 358	\XINTsetupcatcodes	294
\xintLength	294, 332	\xintSgn	352
\xintLength:f:csv	319--321, 360	\xintSgnFork	325, 363
\xintListSel:f:csv	317, 320	\xintSPraw	298
\xintListSel:x:csv	317, 319, 371	\xintspraw	298
\xintloop	331	\xintSPraw::csv	295, 298, 372
\xintmax	349, 350	\xintSqr	352
\xintMaxof:csv	349, 358	\xintSum:csv	349, 359
\xintMessage	331, 332, 334, 364, 366, 374	\xintTFrac	352
\xintmin	349, 350	\xintthe	295
\xintMinof:csv	349, 359	\xintthebareeval	296, 337, 340, 342, 344, 345, 373
\xintMod	356	\xintthebarefloateval	296, 337, 340, 342, 344, 345, 373
\xintMul	339	\xintthebareiieval	296, 337, 340, 342, 344, 345, 373
\xintmul	349	\xinttheboolexpr	297, 372
\xintNewBoolExpr	372	\xintthecoords	296
\xintnewdummy	332, 334	\xinttheDigits	297
\xintNewExpr	317, 372	\xinttheexpr	295, 297, 372
\xintNewFloatExpr	372	\xintthefloatexpr	295, 297, 372
\xintNewFunction	365	\xinttheiexpr	295, 372
\xintNewIExpr	372	\xinttheiiexpr	295, 297, 372
\xintNewIIExpr	372	\xintToggle	329
\xintntheltnoexpand	335, 336	\xinttrim	345--347
\xintNthEltPy:f:csv	319	\xintTrim:f:csv	319, 320
\xintNum	318, 319, 335, 350, 351, 357, 369	\xintTrunc	355
\xintOdd	353	\xintunassignvar	332
\xintOpp	327	\xintXORof:csv	348, 360
\xintORof:csv	348, 360		
\xintPFactorial	356		
\xintPFloat::csv	297, 299, 372		
\xintPIrr	351		
\xintpraw	298		
\xintPrd:csv	349, 359		
\xintRaw	330		
\xintraw	324, 362		

Z

13 Cumulative index

Currently there are some macros which did not make it to the indices because they are defined via constructs such as `\expandafter\def\csname ... \endcsname`, thus their indexing needs some extra mark-up which is yet to be added. This is particularly true for some definitions done by `xintexpr`.

The first indicated page number is the one where the macro is first encountered, hence this is most likely also the page where it gets defined (if it is not one of those which are only used as delimiters).

Symbols	
<code>\%</code>	373
<code>\@backslashchar</code>	112
<code>_!_/</code>	18
A	
<code>\A</code>	175--177
<code>\active</code>	331
<code>\afterassignment</code>	207
<code>\aftergroup</code>	20, 56, 114, 153, 154, 165, 166, 179, 259, 260, 268, 269, 293
B	
<code>\B</code>	175--177
<code>\BEZ</code>	176, 177
<code>\bgroup</code>	43
<code>\break</code>	176
C	
<code>\catcode</code> 5--7, 18, 19, 37--40, 42, 43, 55, 57, 58, 113, 153, 165, 179, 191, 192, 259, 268, 293, 294, 301, 308--311, 318, 319, 328, 331, 332, 334--336, 344, 347, 365, 366, 368, 373--375	
<code>\cfrac</code>	270--273
<code>\char</code>	10
<code>\chardef</code>	8
<code>\count</code>	175--177, 299, 300
<code>\cr</code>	175, 176
<code>\csname</code>	5, 6, 8, 14--17, 19, 20, 28, 29, 31, 38--46, 49--52, 56--59, 92, 111--114, 117, 118, 135, 148, 149, 153--155, 157--159, 162--165, 179, 227, 231, 245, 246, 259, 268, 269, 271, 272, 293, 294, 296, 297, 300, 301, 305, 308--318, 321, 322, 324--332, 334, 335, 337--339, 341--347, 351--361, 364--366, 372
D	
<code>\D</code>	177
<code>\detokenize</code>	327, 329, 331, 368--370
<code>\dimen</code>	299, 300
<code>\dimexpr</code>	40, 41, 300
<code>\do</code>	309--311, 316--318, 326, 329, 364
<code>\dp</code>	300
E	
<code>\edef</code> 6, 7, 15--17, 38, 39, 41--45, 154, 175--177, 319, 331, 332, 334, 335, 364, 365, 373, 374	
<code>\endcsname</code>	5, 6, 8, 14--17, 19, 20, 28, 29, 31, 38--46, 49-- 52, 56--59, 92, 112--114, 116--118, 120, 135, 148, 149, 153--155, 157--159, 162--165, 179, 204, 206, 207, 227, 231, 236, 241, 246, 259, 268, 269, 271, 272, 293, 294, 296, 297, 300-- 306, 308--318, 321, 322, 324--332, 334, 335, 337--339, 341--347, 351--361, 364--366, 372
<code>\endinput</code>	5--7, 20, 56, 114, 153, 154, 165, 166, 179, 259, 260, 268, 269, 293
<code>\endlinechar</code>	5--7, 19, 55, 113, 153, 165, 179, 259, 268, 293, 373
<code>\errmessage</code>	175, 176
<code>\escapechar</code>	44, 45, 373, 374
<code>\everyeof</code>	373
F	
<code>\fdef</code>	11
<code>\fontchardp</code>	300
<code>\fontcharht</code>	300
<code>\fontcharic</code>	300
<code>\fontcharwd</code>	300
<code>\fontdimen</code>	300
<code>\frac</code>	191
<code>\futurelet</code>	36, 38, 39
G	
<code>\glueexpr</code>	300
H	
<code>\halign</code>	175, 176
<code>\hfill</code>	176, 269--272
<code>\ht</code>	300
I	
<code>\if</code> ... 17, 18, 52, 63, 64, 75, 87, 88, 91, 93, 98, 100, 102--104, 122, 123, 125--131, 134, 138-- 140, 145, 146, 158, 160, 161, 170, 171, 181, 185--189, 191, 192, 194--196, 198--200, 209, 213--216, 218, 219, 224, 225, 227--229, 236, 238, 239, 241, 242, 246, 248, 252, 254, 257, 294, 298--308, 314, 319--321, 327, 337--339, 341--348, 350, 357, 362, 363, 368--370, 372	
<code>\ifcat</code>	299, 302--305, 307--309
<code>\ifcsname</code> 43, 44, 57, 58, 308, 309, 329, 330, 332	
<code>\ifdefined</code> 8, 9, 11, 17, 58, 59, 114, 152, 175, 176	
<code>\iffalse</code>	236, 241
<code>\iftoggle</code>	148

\iftrue	236, 241
\ifxintexprsafecatcodes	374
\ifxintglobaldefs	18, 331, 332, 334, 364, 366, 374
\ifxintverbose	18, 331, 332, 334, 364, 366, 374
\immediate	5, 8, 9, 18, 19, 56, 114, 153, 165, 179, 259, 268, 293
\indent	176
\inputlineno	18
K	
\krof	10, 13, 14, 26--32, 48--52, 60--65, 72, 75, 76, 78, 81--83, 87, 99, 102--105, 107, 110, 115--122, 129, 130, 133, 150, 156, 158, 159, 162--164, 166--169, 172, 173, 182--184, 191--199, 201, 204, 206, 208, 209, 211, 213-- 215, 217--221, 223, 225, 226, 230, 235, 236, 238, 240, 241, 243, 253, 255, 276, 277, 309, 310, 312, 314, 317--320, 324, 326, 327, 368
L	
\lccode	18, 300, 331
\lowercase	300, 331
M	
\mathchardef	8, 207
\meaning	364, 366, 374
N	
\N	175--177
\newif	18, 374
\newtoks	20
\noexpand	6, 7, 15--17, 39, 41- 43, 295, 319, 320, 331, 332, 334, 372--374
\numexpr	5, 12--15, 17, 18, 20, 26--36, 38--43, 45, 47--52, 56, 59, 60, 62--71, 73--76, 78--87, 89--101, 106, 108--112, 114--120, 133--151, 153, 156--163, 165, 172, 173, 176, 179, 180, 182--185, 193, 194, 198--207, 210--212, 214, 215, 217, 220, 223--230, 232--238, 240--257, 259--268, 285- 289, 293, 294, 296, 297, 299, 300, 302--304, 319--324, 332, 338, 341--343, 345, 347, 364
O	
\odef	11
\oodef	11
\over	192
P	
\pdfuniformdeviate	8
\PrepareCatCodes	6, 7
\protected	295
\ProvidesPackage	8, 20, 56, 114, 154, 166, 179, 259, 260, 269, 293, 294
R	
\R	9, 21, 24, 66, 68, 70--73, 76, 78, 79, 82--84, 88--90, 93- 96, 98, 105, 106, 108, 110, 115, 116, 143, 146, 155, 156, 158, 160, 162, 201--203, 244
\repeat	35, 44, 176, 177, 332
\romannumeral	. 7, 11--13, 20--33, 36, 38, 40--43, 45--54, 57, 59--65, 71--73, 75--79, 82--85, 87--96, 99, 102--108, 110, 115--133, 135--139, 141-- 146, 148--151, 155, 156, 158--164, 166--175, 180--242, 244, 246--251, 253--258, 260--267, 269--289, 294--299, 301--306, 308--310, 312, 314, 315, 317, 318, 320--327, 329, 331, 335- 351, 354--358, 361--366, 368, 369, 372, 373
S	
\scantokens	373, 374
T	
\t	293
\thexintboolexpr	297
\thexintexpr	295
\thexintfloatexpr	295
\thexintiexpr	295
\thexintiiexpr	295
\times	176, 177
\to	43--45, 175, 176, 331
\toks	365, 373, 374
U	
\U	175--177
\unexpanded	364
\uniformdeviate	8
\unless	57, 64, 86, 114, 123, 130, 131, 137, 145, 158, 175, 176, 248, 319, 320, 374
V	
\V	177
\vbox	175, 176
W	
\W	9, 10, 68--85, 88--90, 93--96, 98, 99, 105, 106, 108--111, 115, 116, 143, 146, 155, 156, 158, 160, 162, 201--203, 244, 298
\w	19, 20, 56, 113, 114, 153, 154, 165, 166, 179, 259, 268, 269, 293
\write	5, 8, 9, 18, 19, 56, 114, 153, 165, 179, 259, 268, 293
X	
\X	73, 78, 90, 203
\x	19, 20, 56, 113, 114, 153, 154, 165, 166, 179, 259, 268, 269, 293
\xint	364
\xint:	10--13, 17, 18, 21--23, 26--31, 43--45, 47- 49, 51--54, 59--61, 69--109, 115, 118, 122, 128--133, 139, 143, 148--152, 155--159, 161, 162, 170--175, 180--184, 190, 196, 198--200, 202--207, 209, 210, 212, 217--223, 230, 232- 235, 239, 240, 242, 243, 247, 248, 251--253, 255, 257, 265--267, 269--272, 278--284, 299
\XINT:expr:macrofunc	366, 371
\XINT:expr:one:and:opt	350, 354, 355, 371

\XINT:expr:one:or:two:nums	371	\XINT_?expr_Vc	40
\XINT:expr:randrange	357	\XINT_?expr_Vd	40, 41
\XINT:expr:tacitzeroifonearg	350, 354, 355, 371	\XINT_?expr_Ve	41
\XINT:expr:totwo	351, 369	\XINT_?expr_Vf	40, 41
\XINT:expr:two:to:one	351, 356--358, 371	\XINT_?expr_Vi	41
\XINT:expr:two:to:two	351, 355, 371	\XINT_?expr_Vx	41
\XINT:expr:userfunc	365, 371	\XINT_?expr_Vy	41
\XINT:flexpr:two:to:one	351, 356, 357, 371	\XINT_abs	61, 180, 221
\XINT:flexpr:two:to:two	351, 355, 371	\XINT_add_A	73
\XINT:iiexpr:one:or:two:	371	\XINT_add_a	73--75, 85
\XINT:iiexpr:randrange	357	\XINT_add_aa	73
\XINT:iiexpr:tacitzeroifonearg	350, 355, 371	\XINT_add_aa_small	73
\XINT:NE:csv	369, 371	\XINT_add_b	74
\XINT:NE:LApply::csv	368	\XINT_add_bi	74
\XINT:NE:one	369, 371	\XINT_add_c	74
\XINT:NE:one_a	369	\XINT_add_checklengths	73
\XINT:NE:oneopt	369	\XINT_add_d	74
\XINT:NE:oneopt_a	369	\XINT_add_di	74
\XINT:NE:oneopt_b	369	\XINT_add_e	74
\XINT:NE:oneopt_fork	369	\XINT_add_exchange	73
\XINT:NE:RApply::csv	368	\XINT_add_f	74
\XINT:NE:RLApply::csv	368	\XINT_add_fi	74
\XINT:NE:two	368--371	\XINT_add_firstiszero	72
\XINT:NE:two_	368, 369, 371	\XINT_add_fork	72, 132
\XINT:NE:two_a	368	\XINT_add_g	74
\XINT:NE:two_b	368	\XINT_add_h	74
\XINT:NE:two_fork_dd	368	\XINT_add_hi	74, 75
\XINT:NE:two_fork_nn	368	\XINT_add_i	75
\XINT:NEhook:csv	298, 358--361, 371	\XINT_add_k	74, 75
\XINT:NEhook:one	298, 311, 318, 327, 328, 351--354, 357, 371	\XINT_add_ke	75
\XINT:NEhook:two	298, 312, 314, 339, 371	\XINT_add_kf	75
\XINT:newexpr:insertdollar	369, 370	\XINT_add_l	75
\XINT:newexpr:macrofunc	366, 371	\XINT_add_lf	75
\XINT:newexpr:macrofunc:a	366	\XINT_add_m	75
\XINT:newexpr:one:and:opt	369, 371	\XINT_add_minusminus	72
\XINT:newexpr:one:or:two:nums	371	\XINT_add_minusplus	72
\XINT:newexpr:tacitzeroifonearg	369, 371	\XINT_add_n	75
\XINT:newexpr:two:to:one	370, 371	\XINT_add_nfork	72
\XINT:newexpr:two:to:two	369, 371	\XINT_add_o	75
\XINT:newexpr:userfunc	365, 371	\XINT_add_out	73
\XINT:newflexpr:two:to:one	370, 371	\XINT_add_plusminus	72
\XINT:newflexpr:two:to:two	369, 371	\XINT_add_plusplus	72, 73
\XINT:newiiexpr:one:or:two:	371	\XINT_add_pp_a	72, 73, 78
\XINT:newiiexpr:tacitzeroifonearg	369, 371	\XINT_add_pp_b	72, 73
\XINT_	364	\XINT_add_secondiszero	72
\XINT_::_end	299	\xint_afterfi	10, 34, 38, 45, 100, 107, 123, 124, 128, 129, 146, 157, 186, 191, 192, 194, 196, 200, 216, 218, 219, 238, 246, 248, 260--267, 285--288, 305, 306, 309, 310, 312, 314, 315, 317, 318, 320, 326, 327, 348, 362--364
\XINT_::_end	298	\XINT_allexpr_iter	342
\XINT_?expr_D	41	\XINT_allexpr_iterr	345
\XINT_?expr_Da	40	\XINT_allexpr_opx	339
\XINT_?expr_Di	41	\XINT_allexpr_rrseq	344
\XINT_?expr_U	41	\XINT_allexpr_rseq	340
\XINT_?expr_Ua	40	\XINT_allexpr_seqx	337
\XINT_?expr_Ui	41	\XINT_allexpr_subx	340
\XINT_?expr_V	41		
\XINT_?expr_Va	40		
\XINT_?expr_Vb	40		

<code>\XINT_andof:_a</code>	348	<code>\XINT_bezout_botharezero</code>	168
<code>\XINT_andof:_c</code>	348	<code>\XINT_bezout_exitA</code>	170, 171
<code>\XINT_andof:_e</code>	348	<code>\XINT_bezout_exita</code>	171
<code>\XINT_andof:_no</code>	348	<code>\XINT_bezout_firstiszero</code>	168
<code>\XINT_andof_a</code>	128	<code>\XINT_bezout_fork</code>	168
<code>\XINT_andof_b</code>	128	<code>\XINT_bezout_loop_B</code>	170, 171
<code>\XINT_andof_c</code>	128	<code>\XINT_bezout_loop_b</code>	171
<code>\XINT_andof_e</code>	128	<code>\XINT_bezout_loop_C</code>	170
<code>\XINT_andof_no</code>	128	<code>\XINT_bezout_loop_c</code>	171
<code>\XINT_apply</code>	32	<code>\XINT_bezout_loop_D</code>	170, 171
<code>\XINT_apply_end</code>	32	<code>\XINT_bezout_loop_d</code>	171
<code>\XINT_apply_loop_a</code>	32	<code>\XINT_bezout_loop_E</code>	171
<code>\XINT_apply_loop_b</code>	32	<code>\XINT_bezout_loop_e</code>	171
<code>\XINT_applyon:::_a</code>	368	<code>\XINT_bezout_minusminus</code>	168, 169
<code>\XINT_applyon:::_b</code>	368	<code>\XINT_bezout_minusplus</code>	168, 169
<code>\XINT_applyon:::_c</code>	368	<code>\XINT_bezout_mm_post</code>	169
<code>\XINT_applyon:::_d</code>	368	<code>\XINT_bezout_mm_postb</code>	169
<code>\XINT_applyon:::_e</code>	368	<code>\XINT_bezout_mp_post</code>	169
<code>\XINT_applyon:::_end</code>	368	<code>\XINT_bezout_plusminus</code>	168, 169
<code>\XINT_applyon::_a</code>	368	<code>\XINT_bezout_plusplus</code>	168, 169
<code>\XINT_applyon::_b</code>	368	<code>\XINT_bezout_pm_post</code>	169
<code>\XINT_applyon::_c</code>	368	<code>\XINT_bezout_preloop_A</code>	170
<code>\XINT_applyon::_d</code>	368	<code>\XINT_bezout_preloop_a</code>	169, 170
<code>\XINT_applyon::_e</code>	368	<code>\XINT_bezout_preloop_exchange</code>	170
<code>\XINT_applyon::_end</code>	368	<code>\XINT_bezout_preloop_exit</code>	170
<code>\XINT_applyunbr</code>	32	<code>\XINT_bezout_secondiszero</code>	168
<code>\XINT_applyunbr_end</code>	32, 33	<code>\XINT_binom_a</code>	140
<code>\XINT_applyunbr_loop_a</code>	32, 33	<code>\XINT_binom_b</code>	140
<code>\XINT_applyunbr_loop_b</code>	32, 33	<code>\XINT_binom_bigloop</code>	140--142
<code>\xint_arrayname</code>	44, 45	<code>\XINT_binom_bigloop_a</code>	142
<code>\XINT_assign_a</code>	43	<code>\XINT_binom_bigloop_b</code>	142
<code>\XINT_assign_b</code>	43	<code>\XINT_binom_div</code>	141--144
<code>\XINT_assign_c</code>	43, 44	<code>\XINT_binom_end_</code>	140--142, 144, 145
<code>\XINT_assign_d</code>	44	<code>\XINT_binom_end_i</code>	141, 142, 144, 145
<code>\XINT_assign_def</code>	43, 44	<code>\XINT_binom_end_ii</code>	141, 144
<code>\XINT_assign_e</code>	44	<code>\XINT_binom_end_iii</code>	141, 143, 144
<code>\XINT_assign_f</code>	43, 44	<code>\XINT_binom_finish</code>	143, 144
<code>\XINT_assign_fork</code>	43	<code>\XINT_binom_fork</code>	139, 140
<code>\XINT_assign_opt</code>	43	<code>\XINT_binom_medloop</code>	140, 141
<code>\XINT_assignarray</code>	44, 45	<code>\XINT_binom_medloop_a</code>	141
<code>\XINT_assignarray_def</code>	44, 45	<code>\XINT_binom_medloop_b</code>	141
<code>\XINT_assignarray_end</code>	45	<code>\XINT_binom_mul</code>	141--144
<code>\XINT_assignarray_fork</code>	44	<code>\XINT_binom_pre</code>	139, 211
<code>\XINT_assignarray_loop</code>	45	<code>\XINT_binom_smallloop</code>	140, 141
<code>\XINT_assignarray_opt</code>	44	<code>\XINT_binom_smallloop_a</code>	141, 143
<code>\XINT_bezalg</code>	173	<code>\XINT_binom_smallloop_b</code>	141
<code>\XINT_bezalg_a</code>	173, 174	<code>\XINT_binom_vbigloop</code>	140, 142
<code>\XINT_bezalg_AisZero</code>	173	<code>\XINT_binom_vbigloop_a</code>	142
<code>\XINT_bezalg_b</code>	173	<code>\XINT_binom_vsmallend_</code>	143, 145
<code>\XINT_bezalg_BisZero</code>	173	<code>\XINT_binom_vsmallend_b</code>	145
<code>\XINT_bezalg_c</code>	173, 174	<code>\XINT_binom_vsmallend_i</code>	143, 145
<code>\XINT_bezalg_d</code>	174	<code>\XINT_binom_vsmallend_ib</code>	145
<code>\XINT_bezalg_e</code>	174	<code>\XINT_binom_vsmallend_ii</code>	143, 144
<code>\XINT_bezalg_end</code>	174	<code>\XINT_binom_vsmallend_iib</code>	144
<code>\XINT_bezalg_end_a</code>	174	<code>\XINT_binom_vsmallend_iii</code>	143, 144
<code>\XINT_bezalg_fork</code>	173	<code>\XINT_binom_vsmallend_iiib</code>	144
<code>\XINT_bezout</code>	168	<code>\XINT_binom_vsmallfinish</code>	144, 145

<code>\XINT_binom_vsmallloop</code>	140, 142, 143	<code>\xint_c_iv</code>	8, 12, 13, 26--31, 47, 49, 51, 52, 73, 76, 78, 84, 90, 105, 112, 115, 118, 136, 141, 143, 146, 180, 183, 184, 199, 203, 226, 244, 246, 249, 252
<code>\XINT_binom_vsmallloop_a</code>	143	<code>\xint_c_ix</code>	8, 12, 15, 30, 31, 47, 49, 51, 52, 93, 135, 185, 301--304, 307, 308
<code>\XINT_binom_vsmallloop_b</code>	143	<code>\xint_c_mone</code>	58, 60, 337
<code>\XINT_binom_vsmallmuldiv</code>	143--145	<code>\xint_c_nine_x^viii</code>	114, 149
<code>\XINT_boolexpr_done</code>	297	<code>\xint_c_v</code>	8, 12, 13, 15, 26--31, 47, 49, 51, 52, 63, 64, 76, 89, 90, 115, 118, 180, 183, 184, 199, 228, 230, 241, 242, 254
<code>\XINT_boolexpr_print</code>	295, 297, 372	<code>\xint_c_vi</code>	8, 12, 13, 26--31, 47, 49, 51, 52, 72, 73, 76, 78, 84, 90, 92, 105, 115, 118, 133, 180, 183, 184, 199, 203
<code>\xint_bracedstopper</code>	10, 44, 45	<code>\xint_c_vii</code> ...	8, 12, 13, 26--31, 47, 49, 51, 52, 76, 90, 92, 115, 118, 180, 183, 184, 199, 328
<code>\XINT_btd_checkin</code>	161	<code>\xint_c_viii</code>	8, 12, 13, 26--32, 47--52, 69, 72, 73, 78, 84, 90, 105, 115, 117, 118, 136, 137, 148, 149, 151, 157--159, 180, 183, 184, 199, 203, 244, 246, 247, 249, 255, 311, 321
<code>\XINT_btd_htd</code>	162	<code>\xint_c_x</code>	8, 27, 63-- 65, 72, 73, 78, 84, 89, 90, 103, 105, 133, 150, 201, 203, 226, 241, 242, 302, 327, 328
<code>\XINT_btd_main</code>	162	<code>\xint_c_x^iv</code>	8, 110, 243, 244, 249
<code>\XINT_btd_N</code>	161, 162	<code>\xint_c_x^ix</code>	59, 85, 86, 97, 247
<code>\XINT_bth_checkin</code>	162	<code>\xint_c_x^viii</code>	58, 86, 87, 89, 97, 100, 101, 111, 112, 136, 143, 146--149, 202, 244--251
<code>\XINT_bth_loop</code>	162, 163	<code>\xint_c_x^viii_mone</code> 59, 101, 102, 140, 146, 243, 248, 252
<code>\XINT_bth_main</code>	162	<code>\xint_c_xi_e_viii_mone</code>	59, 83, 100, 101
<code>\XINT_bth_N</code>	162	<code>\xint_c_xii</code>	8, 72, 73, 78, 84, 90, 105, 203
<code>\xint_Bye</code>	10, 63--65, 89, 103, 201, 226, 241, 242, 321, 322	<code>\xint_c_xii_e_viii</code>	59
<code>\xint_bye</code> 10--13, 21--34, 47--54, 62--65, 67, 82, 89, 91, 94, 99, 103, 115, 117, 118, 121, 137-- 139, 144, 156, 158, 160--164, 173, 174, 180, 183, 184, 199, 201, 204, 206, 207, 226--228, 236, 241, 242, 254, 321, 327, 328, 336, 337		<code>\xint_c_xiv</code>	8, 71
<code>\xint_bye_thenstop</code>	10, 26, 52	<code>\xint_c_xvi</code>	8, 137, 157--159
<code>\xint_c_</code>	8, 12--15, 26--31, 34, 44, 45, 47, 51, 60, 62, 66, 69, 73, 76, 78, 84, 90, 93, 99, 100, 105, 115, 118, 126, 151, 157, 180, 183, 184, 187--190, 199, 200, 203, 205, 206, 228, 230, 232, 237, 238, 241, 242, 247, 257, 285, 286, 299, 308, 310, 319, 320, 337	<code>\xint_c_xviii</code> 8, 300, 301, 307, 311, 336, 339, 340	
<code>\xint_c_i</code>	8, 12, 13, 15, 26--31, 33, 42, 45, 47, 49, 51, 52, 60, 63, 64, 67, 70--73, 75, 76, 78--81, 83, 84, 89--91, 93, 94, 96, 98--101, 105, 106, 108, 109, 111, 112, 115, 118, 134--136, 140--148, 151, 156--159, 172, 173, 180, 183--185, 199--205, 223, 226, 228-- 230, 236--238, 241--246, 248--252, 254--256, 264--266, 287--289, 303, 304, 310, 311, 318, 323, 324, 337, 338, 341--343, 345, 347, 364	<code>\xint_c_xxii</code>	8, 84
<code>\xint_c_ii</code>	8, 12, 13, 26--31, 42, 47, 49, 51, 52, 62--64, 72--76, 78, 83--85, 89, 90, 105, 108, 109, 111, 112, 115, 118, 133--138, 141--148, 180, 183, 184, 199, 202, 203, 227, 237, 238, 241, 242, 244-- 246, 249--251, 254, 255, 311, 317, 319, 326	<code>\XINT_cfrac_A</code>	269, 270
<code>\xint_c_ii^v</code>	8, 204, 205	<code>\XINT_cfrac_B</code>	270
<code>\xint_c_ii^vi</code>	8, 204, 205	<code>\XINT_cfrac_C</code>	270
<code>\xint_c_ii^vii</code>	8, 17, 149, 157, 158	<code>\XINT_cfrac_D</code>	270
<code>\xint_c_ii^viii</code>	8, 157, 158	<code>\XINT_cfrac_end</code>	270
<code>\xint_c_ii^xii</code>	8	<code>\XINT_cfrac_end_b</code>	270
<code>\xint_c_ii^xiv</code>	8, 17, 149	<code>\XINT_cfrac_integer</code>	270
<code>\xint_c_ii^xv</code>	154, 156	<code>\XINT_cfrac_loop_a</code>	270
<code>\xint_c_ii^xvi</code>	154, 156, 160	<code>\XINT_cfrac_loop_d</code>	270
<code>\xint_c_ii^xxi</code>	8, 17, 149	<code>\XINT_cfrac_loop_e</code>	270
<code>\xint_c_iii</code>	8, 12, 13, 26--31, 43, 47, 49, 51, 52, 76, 90, 111, 112, 115, 118, 141, 143, 144, 146--149, 180, 183, 184, 199, 202, 227, 236, 240, 244--246, 249--252, 254, 318	<code>\xint_cfrac_loop_exit</code>	270
		<code>\XINT_cfrac_loop_f</code>	270
		<code>\XINT_cfrac_noopt</code>	269
		<code>\XINT_cfrac_opt_a</code>	269
		<code>\XINT_cfrac_opt_b</code>	269
		<code>\XINT_cfrac_optc</code>	269
		<code>\XINT_cfrac_optl</code>	269
		<code>\XINT_cfrac_optr</code>	270
		<code>\XINT_cfrac_T</code>	270
		<code>\XINT_chtb_checkin</code>	164
		<code>\XINT_chtb_main</code>	164

\XINT_chtb_N	164	\XINT_csvtol_finish_dviii	24
\XINT_Clamped.handler	58	\XINT_csvtol_loop_a	24
\XINT_Cmp	294, 325, 363	\XINT_csvtol_loop_b	24
\XINT_cmp_a	76, 77	\XINT_ctcv_end	281, 282
\XINT_cmp_checklengths	76	\XINT_ctcv_loop_a	281, 282
\XINT_cmp_distinctlengths	76	\XINT_ctcv_loop_b	281
\XINT_cmp_equal	77	\XINT_ctcv_loop_c	281
\XINT_cmp_firstiszero	75, 76	\XINT_ctcv_loop_d	281
\XINT_cmp_gt	77	\XINT_ctcv_loop_e	281
\XINT_cmp_lt	77	\XINT_ctcv_loop_f	281, 282
\XINT_cmp_minusminus	75, 76	\XINT_ctcv_loop_g	282
\XINT_cmp_minusplus	75, 76	\XINT_ctcv_prep	281
\XINT_cmp_nfork	75	\XINT_ctf_end	277, 278
\XINT_cmp_plusminus	75, 76	\XINT_ctf_loop_a	277, 278
\XINT_cmp_plusplus	75, 76, 122, 221	\XINT_ctf_loop_b	277
\XINT_cmp_pp	76	\XINT_ctf_loop_c	278
\XINT_cmp_secondiszero	75, 76	\XINT_ctf_loop_d	278
\XINT_cntcs	287	\XINT_ctf_loop_e	278
\XINT_cntcs_exit	287	\XINT_ctf_prep	277
\XINT_cntcs_exit_b	287	\XINT_cuz	66, 106
\XINT_cntcs_loop	287	\XINT_cuz_byviii	66
\XINT_cntf	285	\XINT_cuz_byviii_done	66
\XINT_cntf_exit	286	\XINT_cuz_byviii_e	66
\XINT_cntf_loop	285, 286	\XINT_cuz_byviii_z	66
\XINT_cntgc	288	\XINT_cuz_cleantoend	66
\XINT_cntgc_exit	288	\XINT_cuz_hitend	66
\XINT_cntgc_exit_b	288	\XINT_cuz_loop	66, 68
\XINT_cntgc_loop	288	\XINT_cuz_small	59, 71, 73, 82, 85
\XINT_cntSgn	60, 107, 180, 190, 217, 220	\XINT_dbl	62, 137, 138, 254
\XINT_cntSgnFork	180, 217, 220	\XINT_dbl_a	62
\XINT_ConversionSyntax-signal	57	\XINT_dbl_e	62, 63
\XINT_ConversionSyntax.handler	58	\XINT_dbl_fork	62
\XINT_csbth_none	155	\XINT_dbl_neg	62
\XINT_cshtb_A	155	\xint_ddfork	368
\XINT_cshtb_B	155	\XINT_dec	63, 64, 139
\XINT_cshtb_C	155	\XINT_dec_a	64
\XINT_cshtb_D	155	\XINT_dec_bye	63, 64, 139
\XINT_cshtb_E	155	\XINT_dec_e	64
\XINT_cshtb_F	155	\XINT_dec_fork	64
\XINT_cshtb_none	155	\XINT_dec_neg	64
\XINT_cstc_end	289	\XINT_dectostr	189
\XINT_cstc_loop_a	289	\XINT_dectostr_a	189
\XINT_cstc_loop_b	289	\XINT_dectostr_b	189
\XINT_cstc_prep	289	\XINT_dectostr_z	189
\XINT_csv:::a	298	\XINT_denom_A	190, 191
\XINT_csv:::b	298	\XINT_denom_B	190, 191
\XINT_csv:::c	298	\XINT_denom_fork	190
\XINT_csv:::d	298	\XINT_div_BisOne	88
\XINT_csvtol_finish_a	24	\XINT_div_BisTwo	88
\XINT_csvtol_finish_b	24	\XINT_div_BisTwo_a	89
\XINT_csvtol_finish_di	24, 25	\XINT_div_cleanR	91, 92
\XINT_csvtol_finish_dii	24, 25	\XINT_div_dosmalldiv	89, 90
\XINT_csvtol_finish_diii	24	\XINT_div_dosmallsmall	89
\XINT_csvtol_finish_div	24	\XINT_div_exittofinish	94
\XINT_csvtol_finish_dv	24	\XINT_div_finish	91
\XINT_csvtol_finish_dvi	24	\XINT_div_finish_a	91
\XINT_csvtol_finish_dvii	24	\XINT_div_finish_b	91

\XINT_div_finish_bRpos	91	\XINT_div_start_b	91
\XINT_div_finish_bRzero	91	\XINT_div_start_c	91, 92
\XINT_div_I_a	92, 97	\XINT_div_start_c_	92
\XINT_div_I_b	92, 93	\XINT_div_start_c_i	92
\XINT_div_I_c	92, 93	\XINT_div_start_c_ii	92
\XINT_div_I_czero	92, 93	\XINT_div_start_c_iii	92
\XINT_div_I_da	93	\XINT_div_start_c_iv	92
\XINT_div_I_db	93	\XINT_div_start_c_v	92
\XINT_div_I_dc	93	\XINT_div_start_c_vi	92
\XINT_div_I_dd	93	\XINT_div_start_ca	92
\XINT_div_I_de	93, 94	\XINT_div_start_cb	92
\XINT_div_I_dN	93	\XINT_div_sub	93--96, 98
\XINT_div_I_dP	93, 94	\XINT_div_sub_a	98
\XINT_div_I_dz	93	\XINT_div_sub_b	98
\XINT_div_I_g	93, 94	\XINT_div_sub_bi	98, 99
\XINT_div_I_h	94	\XINT_div_sub_c	98, 99
\XINT_div_II_b	94	\XINT_div_sub_clean	98
\XINT_div_II_c	95	\XINT_div_sub_d	98
\XINT_div_II_d	95	\XINT_div_sub_di	98, 99
\XINT_div_II_e	95	\XINT_div_sub_e	98, 99
\XINT_div_II_f	95, 96	\XINT_div_sub_f	98
\XINT_div_II_fa	96	\XINT_div_sub_fi	98, 99
\XINT_div_II_g	96	\XINT_div_sub_g	98, 99
\XINT_div_II_h	96	\XINT_div_sub_h	98
\XINT_div_II_k	95, 96	\XINT_div_sub_hi	98, 99
\XINT_div_II_l	96	\XINT_div_sub_i	98, 99
\XINT_div_II_m	96, 97	\XINT_div_sub_l	99
\XINT_div_II_skipc	95	\XINT_div_sub_neg	98
\XINT_div_II_skipf	95, 96	\XINT_div_sub_r	99
\XINT_div_mini	95, 101	\XINT_div_unsepQ	67, 91
\XINT_div_mini_a	101	\XINT_div_unsepQ_delim	67, 94
\XINT_div_mini_b	101	\XINT_div_unsepQ_one	67, 68
\XINT_div_mini_c	101	\XINT_div_unsepQ_x	67
\XINT_div_mini_d	101, 102	\XINT_div_unsepQ_y	67, 68
\XINT_div_mini_w	101	\XINT_div_unsepR	68, 94
\XINT_div_minimulwc_a	97	\XINT_div_unsepR_x	68
\XINT_div_minimulwc_b	97	\XINT_div_verysmallisone	97
\XINT_div_minimulwc_c	97	\XINT_div_verysmallmul	93, 94, 97
\XINT_div_prepare .. 88, 103, 104, 167, 169--173,		\XINT_div_verysmallmul_a	97
195, 196, 199, 208, 214--216, 270, 273--276		\XINT_div_verysmallmul_b	97
\XINT_div_prepare_a	88	\XINT_div_verysmallmul_e	97
\XINT_div_prepare_b	88, 90	\XINT_div_xmini	95, 96
\XINT_div_prepare_c	90	\XINT_div_xmini_a	95
\XINT_div_prepare_d	90	\XINT_div_xmini_b	95
\XINT_div_prepare_e	90	\XINT_div_xmini_c	95
\XINT_div_prepare_f	90	\XINT_div_zeroQ	91
\XINT_div_prepare_g	90, 203	\XINT_div_zeroQ_end	91
\XINT_div_prepare_h	90, 91	\XINT_DivisionByZero.handler	58
\XINT_div_prepare_small	88	\XINT_DivisionImpossible-signal	57
\XINT_div_small_a	88, 89	\XINT_DivisionImpossible.handler	58
\XINT_div_small_b	89, 202	\XINT_DivisionUndefined-signal	57
\XINT_div_small_ba	89, 202	\XINT_DivisionUndefined.handler	58
\XINT_div_smallmul_a	95--97	\XINT_divmod_a	214
\XINT_div_smallmul_e	97	\XINT_divmod_aiszero	214
\XINT_div_smallsmall	89	\XINT_divmod_b	214
\XINT_div_smallsmallend	89	\XINT_divmod_bneg	214
\XINT_div_start_a	91	\XINT_divmod_bneg_finish	214

<code>\XINT_divmod_bpos</code>	214, 215	<code>\XINT_dth_neg</code>	155, 156
<code>\XINT_divmod_bpos_a</code>	215	<code>\XINT_dth_tohex</code>	156--158
<code>\XINT_divmod_bpos_finish</code>	215	<code>\XINT_dth_tohex_a</code>	157
<code>\XINT_divmod_divbyzero</code>	214	<code>\XINT_dthb_again</code>	156, 157
<code>\xint_dothis</code>	10, 57, 58, 64, 87, 102--104, 110, 123, 133--135, 137, 138, 140, 145, 146, 150, 151, 157, 158, 170, 185, 189, 198--200, 213--215, 224, 228, 230, 232, 236, 241--243, 248, 249, 252, 254, 257, 298--309, 314, 319, 320, 329, 338, 339, 341, 343--347, 368--371	<code>\XINT_dthb_exclam</code>	156
<code>\XINT_dsh_fork</code>	119	<code>\XINT_dthb_lastpass</code>	156
<code>\XINT_dsh_xisPos</code>	119	<code>\XINT_dthb_nextfour</code>	156
<code>\XINT_dsh_xiszero</code>	119	<code>\XINT_dthb_small</code>	156
<code>\XINT_dshr_fork</code>	119	<code>\XINT_dthb_start</code>	156, 158
<code>\XINT_dshr_xpositive</code>	119	<code>\XINT_dthb_start_a</code>	156
<code>\XINT_dshr_xzeroroneg</code>	119	<code>\XINT_dthb_update</code>	156, 157
<code>\XINT_dsl</code>	64	<code>\XINT_dthb_update_a</code>	156
<code>\xint_dsl_zero</code>	64	<code>\XINT_e</code>	194
<code>\XINT_dsr</code>	65	<code>\XINT_e_end</code>	194
<code>\XINT_dsr_a</code>	65	<code>\XINT_eightrandmdigits</code>	149, 258
<code>\XINT_dsr_b</code>	65	<code>\XINT_euc</code>	171
<code>\XINT_dsr_e</code>	65	<code>\XINT_euc_a</code>	172
<code>\XINT_dsr_fork</code>	64	<code>\XINT_euc_AisZero</code>	172
<code>\XINT_dsr_neg</code>	65	<code>\XINT_euc_b</code>	172
<code>\XINT_dsrr</code>	65, 103, 201, 226	<code>\XINT_euc_BisZero</code>	172
<code>\XINT_dsrr_a</code>	65	<code>\XINT_euc_c</code>	172
<code>\XINT_dsrr_b</code>	65	<code>\XINT_euc_end</code>	172
<code>\XINT_dsrr_e</code>	65	<code>\XINT_euc_end_a</code>	172, 173
<code>\XINT_dsrr_fork</code>	65	<code>\XINT_euc_fork</code>	172
<code>\XINT_dsrr_neg</code>	65	<code>\xint_exchangetwo_keepbraces</code>	
<code>\XINT_dsx_addzeros</code>	120, 138, 189-- 191, 198, 199, 205, 208, 217, 221, 224, 254	9, 88, 92, 105, 169, 170, 215, 314
<code>\XINT_dsx_addzerosnofuss</code>	120, 138, 227	<code>\XINT_expandableerror</code>	18, 45, 58, 150, 152
<code>\XINT_dsx_AisNeg</code>	121	<code>\XINT_expandableerror_continue</code>	18
<code>\XINT_dsx_AisNeg_checkiffirstempty</code>	121	<code>\XINT_expr_unlock</code>	343, 346, 347
<code>\XINT_dsx_AisNeg_finish_notzero</code>	121	<code>\XINT_expr_binop_inline</code>	314, 315
<code>\XINT_dsx_AisNeg_finish_zero</code>	121	<code>\XINT_expr_binop_inline_a</code>	314
<code>\XINT_dsx_AisPos</code>	121	<code>\XINT_expr_binop_inline_b</code>	314
<code>\XINT_dsx_AisPos_finish</code>	121	<code>\XINT_expr_binop_inline_c</code>	314
<code>\XINT_dsx_AisZero</code>	120, 121	<code>\XINT_expr_binop_inline_d</code>	314
<code>\XINT_dsx_append</code>	116, 120	<code>\XINT_expr_binop_inline_e</code>	314
<code>\XINT_dsx_end</code>	121	<code>\XINT_expr_binop_inline_end</code>	314
<code>\XINT_dsx_fork</code>	120	<code>\XINT_expr_binop_pwd</code>	308
<code>\XINT_dsx_xisNeg_Azero</code>	120	<code>\XINT_expr_countetc</code>	299, 300
<code>\XINT_dsx_xisNeg_checkA</code>	119, 120	<code>\XINT_expr_defbin_b</code>	312, 313
<code>\XINT_dsx_xisPos</code>	119, 120	<code>\XINT_expr_defbin_c</code>	312, 313
<code>\XINT_dsx_xisZero</code>	120	<code>\XINT_expr_deflistopl_b</code>	315, 316
<code>\XINT_dtb_checkin</code>	158	<code>\XINT_expr_deflistopl_c</code>	315
<code>\XINT_dtb_finish</code>	158, 159	<code>\XINT_expr_deflistopr_b</code>	314, 315
<code>\XINT_dtb_finish_a</code>	159	<code>\XINT_expr_deflistopr_c</code>	314
<code>\XINT_dtb_main</code>	158	<code>\XINT_expr_defuserfunc</code>	364, 365
<code>\XINT_dtb_neg</code>	158	<code>\XINT_expr_defvar</code>	331, 332
<code>\XINT_dtb_tobin</code>	158, 159	<code>\XINT_expr_defvar_getname</code>	331
<code>\XINT_dtb_tobin_a</code>	158	<code>\XINT_expr_defvar_one</code>	331
<code>\XINT_dth_checkin</code>	155	<code>\XINT_expr_done</code>	295, 309, 310
<code>\XINT_dth_finish</code>	156, 158	<code>\XINT_expr_extra_)</code>	309
<code>\XINT_dth_main</code>	156	<code>\XINT_expr_foundend</code>	308, 311, 335
		<code>\XINT_expr_foundop</code>	309
		<code>\XINT_expr_foundop_a</code>	308, 309
		<code>\XINT_expr_func</code>	306, 307
		<code>\XINT_expr_func_!</code>	353
		<code>\XINT_expr_func_?</code>	353
		<code>\XINT_expr_func_@@</code>	335

\XINT_expr_func_@@@	335	\XINT_expr_getop	300, 301, 305, 308, 309, 311, 328--330, 337, 339--342, 344, 346
\XINT_expr_func_@@@	335	\XINT_expr_getop_a	308
\XINT_expr_func_abs	351	\XINT_expr_getop_b	308
\XINT_expr_func_all	360	\XINT_expr_gobz_a	301
\XINT_expr_func_any	360	\XINT_expr_gobz_scandec_a	303
\XINT_expr_func_binomial	356	\XINT_expr_gobz_scandec_b	303, 304
\XINT_expr_func_break	329	\XINT_expr_gobz_scandec_c	303, 304
\XINT_expr_func_ceil	352	\XINT_expr_gobz_scandec_endbycs	303, 304
\XINT_expr_func_divmod	355	\XINT_expr_gobz_scanint_b	301--303
\XINT_expr_func_even	353	\XINT_expr_gobz_scanint_c	302
\XINT_expr_func_factorial	354	\XINT_expr_gobz_scanint_d	302, 303
\XINT_expr_func_first	360	\XINT_expr_gobz_scanint_endbycs	302
\XINT_expr_func_float	355	\XINT_expr_gobz_startdec_a	302, 303
\XINT_expr_func_floor	352	\XINT_expr_gotnil	301
\XINT_expr_func_frac	352	\XINT_expr_inhex	294
\XINT_expr_func_gcd	358	\XINT_expr_iter:A	342, 343
\XINT_expr_func_if	361	\XINT_expr_iter:a	342, 343
\XINT_expr_func_ifint	361	\XINT_expr_iter:aa	342
\XINT_expr_func_ifone	361	\XINT_expr_iter:Abort	343
\XINT_expr_func_ifsgn	361, 362	\XINT_expr_iter:abort	343
\XINT_expr_func_iter	342	\XINT_expr_iter:b	343
\XINT_expr_func_iterr	345	\XINT_expr_iter:Break	343
\XINT_expr_func_last	360	\XINT_expr_iter:break	343
\XINT_expr_func_lcm	358	\XINT_expr_iter:c	343
\XINT_expr_func_len	360	\XINT_expr_iter:D	343
\XINT_expr_func_max	358	\XINT_expr_iter:d	343
\XINT_expr_func_min	359	\XINT_expr_iter:E	343
\XINT_expr_func_mod	356	\XINT_expr_iter:end	343
\XINT_expr_func_not	353	\XINT_expr_iter:Goon	343
\XINT_expr_func_num	351	\XINT_expr_iter:goon	343
\XINT_expr_func_nuple	354	\XINT_expr_iter:noop	343
\XINT_expr_func_odd	353	\XINT_expr_iter:Omit	343
\XINT_expr_func_opxadd	339	\XINT_expr_iter:omit	343
\XINT_expr_func_opxmud	339	\XINT_expr_iterr:A	346, 347
\XINT_expr_func_pfactorial	356	\XINT_expr_iterr:a	346
\XINT_expr_func_preduce	351	\XINT_expr_iterr:aa	346
\XINT_expr_func_quo	357	\XINT_expr_iterr:Abort	347
\XINT_expr_func_randrange	357	\XINT_expr_iterr:abort	346
\XINT_expr_func_reduce	351	\XINT_expr_iterr:b	346
\XINT_expr_func_rem	358	\XINT_expr_iterr:Break	347
\XINT_expr_func_reversed	361	\XINT_expr_iterr:break	346
\XINT_expr_func_round	354, 355	\XINT_expr_iterr:c	346
\XINT_expr_func_rrseq	344	\XINT_expr_iterr:D	347
\XINT_expr_func_rseq	340	\XINT_expr_iterr:d	346
\XINT_expr_func_seqx	337	\XINT_expr_iterr:E	347
\XINT_expr_func_sgn	352	\XINT_expr_iterr:end	346
\XINT_expr_func_sqr	352	\XINT_expr_iterr:Goon	347
\XINT_expr_func_sqrt	354	\XINT_expr_iterr:goon	346
\XINT_expr_func_subx	340	\XINT_expr_iterr:noop	346
\XINT_expr_func_trunc	355	\XINT_expr_iterr:Omit	347
\XINT_expr_func_unknown	329	\XINT_expr_iterr:omit	346
\XINT_expr_func_xor	360	\XINT_expr_iterrx	345, 346
\XINT_expr_getnext	296, 299, 300, 310, 312, 314, 315, 317, 318, 326--328	\XINT_expr_iterrry	346
\XINT_expr_getnext_a	299	\XINT_expr_iterx	342
\XINT_expr_getnextfork	300	\XINT_expr_itory	342
		\XINT_expr_itself_[]	319

\XINT_expr_lockit	294, 331, 341, 343--347	\XINT_expr_precedence_[.....	328
\XINT_expr_lockscan	294	\XINT_expr_precedence_[[:	318
\XINT_expr_macrofunc_	366	\XINT_expr_precedence_a	311
\XINT_expr_makedummy	334	\XINT_expr_print	295, 372
\XINT_expr_missing_)	310	\XINT_expr_redefinemacros	371, 373
\XINT_expr_newfunction	365, 366	\XINT_expr_redefineprints	372
\XINT_expr_onliteral_@	335	\XINT_expr_rrseq:_A	344, 345
\XINT_expr_onliteral_`	301	\XINT_expr_rrseq:_a	344
\XINT_expr_onliteral_add	339	\XINT_expr_rrseq:_aa	344
\XINT_expr_onliteral_add_f	339	\XINT_expr_rrseq:_Abort	345
\XINT_expr_onliteral_bool	329	\XINT_expr_rrseq:_abort	345
\XINT_expr_onliteral_mul	339	\XINT_expr_rrseq:_b	344, 345
\XINT_expr_onliteral_mul_f	339	\XINT_expr_rrseq:_Break	345
\XINT_expr_onliteral_nil	334	\XINT_expr_rrseq:_break	345
\XINT_expr_onliteral_protect	329	\XINT_expr_rrseq:_c	344
\XINT_expr_onliteral_qfloat	330	\XINT_expr_rrseq:_D	345
\XINT_expr_onliteral_qfrac	330	\XINT_expr_rrseq:_d	344, 345
\XINT_expr_onliteral_qint	330	\XINT_expr_rrseq:_E	345
\XINT_expr_onliteral_qrand	330	\XINT_expr_rrseq:_end	344
\XINT_expr_onliteral_qraw	330	\XINT_expr_rrseq:_Goon	345
\XINT_expr_onliteral_random	330	\XINT_expr_rrseq:_goon	345
\XINT_expr_onliteral_seq	336	\XINT_expr_rrseq:_noop	344
\XINT_expr_onliteral_seq_a	336, 339, 340, 342, 344, 345	\XINT_expr_rrseq:_Omit	345
\XINT_expr_onliteral_seq_b	336	\XINT_expr_rrseq:_omit	345
\XINT_expr_onliteral_seq_c	336	\XINT_expr_rrseqx	344
\XINT_expr_onliteral_seq_d	336	\XINT_expr_rseqy	344
\XINT_expr_onliteral_seq_e	336	\XINT_expr_rseq:_A	341
\XINT_expr_onliteral_seq_f	336	\XINT_expr_rseq:_a	341
\XINT_expr_onliteral_subs	340	\XINT_expr_rseq:_aa	341
\XINT_expr_onliteral_subs_f	340	\XINT_expr_rseq:_Abort	341, 342
\XINT_expr_onliteral_togl	329	\XINT_expr_rseq:_abort	341
\XINT_expr_op:_a	339	\XINT_expr_rseq:_b	341
\XINT_expr_op:_b	339	\XINT_expr_rseq:_Break	341, 342
\XINT_expr_op:_c	339	\XINT_expr_rseq:_break	341
\XINT_expr_op:_d	339	\XINT_expr_rseq:_c	341
\XINT_expr_op:_end	339	\XINT_expr_rseq:_D	341, 342
\XINT_expr_op:_noop	339	\XINT_expr_rseq:_d	341
\XINT_expr_op_	312, 314, 317, 326, 327	\XINT_expr_rseq:_E	341
\XINT_expr_op_!	328	\XINT_expr_rseq:_end	341
\XINT_expr_op_!?.	335	\XINT_expr_rseq:_Goon	341
\XINT_expr_op:_	318	\XINT_expr_rseq:_goon	341
\XINT_expr_op:_]	318	\XINT_expr_rseq:_noop	341
\XINT_expr_op_?	327, 328	\XINT_expr_rseq:_Omit	341, 342
\XINT_expr_op_??	327, 328	\XINT_expr_rseq:_omit	341
\XINT_expr_op_?a	327, 328	\XINT_expr_rseqx	340
\XINT_expr_op_?checka	327	\XINT_expr_rseqy	340, 341
\XINT_expr_op_?checkb	327	\XINT_expr_scan_macropar	300
\XINT_expr_op_?checkc	327	\XINT_expr_scan_nbr_or_func	300, 301
\XINT_expr_op_[.....	328	\XINT_expr_scandec_a	301, 302
\XINT_expr_op_[[:	319	\XINT_expr_scandec_b	302, 303
\XINT_expr_op__	307, 330	\XINT_expr_scandec_c	303
\XINT_expr_precedence_	308, 331, 334, 335	\XINT_expr_scandec_d	303
\XINT_expr_precedence_!	328	\XINT_expr_scandec_endbycs	303
\XINT_expr_precedence_!?.	311	\XINT_expr_scanexp_a	302--304
\XINT_expr_precedence:_]	318	\XINT_expr_scanexp_b	304
\XINT_expr_precedence_?	308, 327	\XINT_expr_scanexp_bb	304
		\XINT_expr_scanexp_c	304

\XINT_expr_scanexp_cb	304	\XINT_expr_tmpvar	331, 332
\XINT_expr_scanexp_d	304	\XINT_expr_undefined	332, 333
\XINT_expr_scanexp_db	304	\XINT_expr_unexpectedtoken	294
\XINT_expr_scanexp_endbycs	304	\XINT_expr_unknown_function	329
\XINT_expr_scanexp_endbycs_b	304	\XINT_expr_unknown_operator	309
\XINT_expr_scanexpr_endbycs	304	\XINT_expr_unknown_variable	330
\XINT_expr_scanfunc	301, 306	\XINT_expr_unlock	294--297, 311, 312, 314, 315, 317, 318, 325--329, 331, 335--339, 341--347, 351--361, 365, 366, 372
\XINT_expr_scanfunc_a	306, 307	\XINT_expr_unlock_a	294
\XINT_expr_scanfunc_b	306, 307	\XINT_expr_unlock_hex_in	294, 305
\XINT_expr_scanhex_I	301, 305	\XINT_expr_unlock_sp	297, 372
\XINT_expr_scanhex_transition	305	\XINT_expr_unpackvar	300
\XINT_expr_scanhexI_a	305	\XINT_expr_until	312, 314, 317, 326, 327
\XINT_expr_scanhexI_aa	305	\XINT_expr_until:_a	318
\XINT_expr_scanhexI_b	305	\XINT_expr_until:_b	318
\XINT_expr_scanhexI_bgob	305	\XINT_expr_until_end_a	296
\XINT_expr_scanhexII_a	305, 306	\XINT_expr_usethe	295, 297
\XINT_expr_scanhexII_aa	306	\XINT_expr_var_@	335
\XINT_expr_scanhexII_b	306	\XINT_expr_var_abort	335
\XINT_expr_scanhexII_bgob	306	\XINT_expr_var_nil	334
\XINT_expr_scanint_a	301	\XINT_expr_var_omit	335
\XINT_expr_scanint_b	301, 302	\XINT_expr_wrap	295, 296, 365
\XINT_expr_scanint_c	302	\XINT_expr_wrapit	365
\XINT_expr_scanint_d	302	\XINT_fac_bigloop_a	110, 111
\XINT_expr_scanint_endbycs	302	\XINT_fac_bigloop_b	111
\XINT_expr_scanop_a	308, 311	\XINT_fac_bigloop_exit	111
\XINT_expr_scanop_b	308	\XINT_fac_bigloop_loop	111
\XINT_expr_scanop_c	308, 309	\XINT_fac_bigloop_mul	111
\XINT_expr_scanop_d	308	\XINT_fac_checksize	110
\XINT_expr_seq:_A	337, 338	\XINT_fac_fork	110, 211
\XINT_expr_seq:_a	337, 338	\XINT_fac_loop_exit	111, 112
\XINT_expr_seq:_aa	337	\XINT_fac_medloop_a	110, 111
\XINT_expr_seq:_Abort	338	\XINT_fac_medloop_b	111
\XINT_expr_seq:_abort	338	\XINT_fac_medloop_loop	111
\XINT_expr_seq:_b	338	\XINT_fac_medloop_mul	111
\XINT_expr_seq:_Break	338	\XINT_fac_neg	110
\XINT_expr_seq:_break	338	\XINT_fac_smallloop_a	110, 111
\XINT_expr_seq:_c	338	\XINT_fac_smallloop_loop	112
\XINT_expr_seq:_D	338	\XINT_fac_smallloop_mul	112
\XINT_expr_seq:_d	338	\XINT_fac_toobig	110, 111
\XINT_expr_seq:_E	338	\XINT_fac_zero	110
\XINT_expr_seq:_end	338	\XINT_factortens	185, 193
\XINT_expr_seq:_Goon	338	\XINT_factortens_a	185
\XINT_expr_seq:_goon	338	\XINT_factortens_b	185
\XINT_expr_seq:_noop	338	\XINT_factortens_c	185
\XINT_expr_seq:_Omit	338	\XINT_factortens_cc	185
\XINT_expr_seq:_omit	338	\XINT_factortens_d	185
\XINT_expr_seq:_empty?	324, 325, 338	\XINT_factortens_e	185
\XINT_expr_startdec	300, 301	\XINT_factortens_f	185
\XINT_expr_startdec_a	302	\XINT_factortens_g	185
\XINT_expr_startint	301	\XINT_factortens_x	185
\XINT_expr_subexpr	299, 300	\XINT_factortens_y	185
\XINT_expr_subx:_end	340	\XINT_factortens_yy	185
\XINT_expr_tmpa	331--333, 364--366	\XINT_factortens_z	185
\XINT_expr_tmpb	331, 364, 365	\XINT_fadd	207
\XINT_expr_tmpc	331, 332, 364	\XINT_fadd_a	207
\XINT_expr_tmpd	331, 332		
\XINT_expr_tmpval	331, 332		

\XINT_fadd_Aa	207	\XINT_fgtda	275
\XINT_fadd_Azero	207	\XINT_fgtd_e	275
\XINT_fadd_B	207, 208	\XINT_fgtd_f	275
\XINT_fadd_b	207	\XINT_fgtd_g	275
\XINT_fadd_Ba	208	\XINT_fgtd_h	275
\XINT_fadd_Bb	208	\XINT_first:f:csv_a	53
\XINT_fadd_Bzero	207, 209	\xint_firstofone	9, 10, 18, 20, 22, 23, 46, 53, 58, 229, 275, 349, 350, 372
\XINT_fadd_C	208	\xint_firstofone_thenstop	9, 58
\XINT_fadd_c	207, 209	\xint_firstofthree	114, 180, 308
\XINT_fadd_D_a	208	\xint_firstofthree_thenstop	58, 114, 124, 125, 186, 187
\XINT_fadd_D_b	208	\xint_firstoftwo	9, 17, 38, 39, 42, 58, 76, 77, 103, 128, 157, 188, 199, 232, 247, 303, 307, 330, 350, 357, 369, 372
\XINT_fadd_D_c	208	\xint_firstoftwo_thenstop	9, 33, 102, 103, 105, 125--127, 129--131, 187, 188, 191, 195, 257
\XINT_fadd_D_exit	208	\XINT_FL_add_a	232, 233
\XINT_fadd_E	208	\XINT_FL_add_b	232
\XINT_fadd_F	208	\XINT_FL_add_c	232
\XINT_fadd_G	208, 209	\XINT_FL_add_d	232
\XINT_fadd_iszero	208, 209	\XINT_FL_add_zero	232
\XINT_fcmp	219	\XINT_FL_binom_a	252
\XINT_fcmp_A	220	\XINT_FL_binom_aa	252
\XINT_fcmp_B	220	\XINT_FL_binom_ab	252
\XINT_fcmp_C	220	\XINT_FL_binom_fork	252
\XINT_fcmp_D	220	\XINT_FL_binom_neg	252
\XINT_fcmp_E	220	\XINT_FL_binom_one	252
\XINT_fcmp_Fd	221	\XINT_FL_binom_toobig	252
\XINT_fcmp_Fe	221	\XINT_FL_binom_zero	252
\XINT_fcmp_firstneg	220	\XINT_FL_div_a	234, 235
\XINT_fcmp_firstzero	220	\XINT_FL_div_b	235
\XINT_fcmp_Fn	221	\XINT_FL_fac_addzeros	246
\XINT_fcmp_Fo	221	\XINT_FL_fac_addzeros_exit	246
\XINT_fcmp_minusminus	220	\XINT_FL_fac_big	243, 244
\XINT_fcmp_nonneg_a	220	\XINT_FL_fac_bigloop_a	244, 245
\XINT_fcmp_pos	220	\XINT_FL_fac_bigloop_b	245
\XINT_fcmp_secondneg	220	\XINT_FL_fac_bigloop_loop	245
\XINT_fcmp_secondzero	220	\XINT_FL_fac_bigloop_mul	245
\XINT_fcmp_zerozero	220	\XINT_FL_fac_countdigits	244, 249
\XINT_FDg	61	\XINT_FL_fac_countdone	244
\XINT_fdg	61	\XINT_FL_fac_fork_a	243
\XINT_fdiv	212	\XINT_FL_fac_fork_b	243, 252
\XINT_fdiv_A	212	\XINT_FL_fac_increaseP	244
\XINT_fdiv_B	212	\XINT_FL_fac_isneg	243
\XINT_fdiv_C	212	\XINT_FL_fac_iszero	243
\XINT_fgeq	216	\XINT_FL_fac_loop_exit	244--246
\XINT_fgeq_A	216--219	\XINT_FL_fac_med	243, 244
\XINT_fgeq_B	217	\XINT_FL_fac_medloop_a	244, 245
\XINT_fgeq_C	217	\XINT_FL_fac_medloop_b	245
\XINT_fgeq_D	217	\XINT_FL_fac_medloop_loop	245
\XINT_fgeq_E	217	\XINT_FL_fac_medloop_mul	245
\XINT_fgeq_Fd	217	\XINT_FL_fac_minimulwc_a	246, 247
\XINT_fgeq_Fe	217	\XINT_FL_fac_minimulwc_b	247
\XINT_fgeq_Fn	217	\XINT_FL_fac_minimulwc_c	247
\XINT_fgeq_Fo	217	\XINT_FL_fac_minimulwc_d	247
\XINT_fgeq_Zi	217	\XINT_FL_fac_minimulwc_e	247
\XINT_fgeq_Zii	217		
\XINT_fgtd_a	275		
\XINT_fgtd_b	275		
\XINT_fgtd_c	275		
\XINT_fgtd_d	275, 276		

\XINT_FL_fac_mul	245, 246, 249--251	\XINT_flet_sp?	38
\XINT_FL_fac_mul_a	246	\XINT_flet_zapsp	38, 43, 44
\XINT_FL_fac_out	243, 244, 251, 252	\XINT_flexpr_func_!	353
\XINT_FL_fac_small	243, 244	\XINT_flexpr_func_?	353
\XINT_FL_fac_smallloop_a	244, 245	\XINT_flexpr_func_@@	335
\XINT_FL_fac_smallloop_loop	246	\XINT_flexpr_func_@@@	335
\XINT_FL_fac_smallloop_mul	246	\XINT_flexpr_func_@@@	335
\XINT_FL_fac_smallmul	246, 247	\XINT_flexpr_func_abs	351
\XINT_FL_fac_smallmul_end	247	\XINT_flexpr_func_all	360
\XINT_FL_fac_toobig	243	\XINT_flexpr_func_any	360
\XINT_FL_fac_vbig	243, 244	\XINT_flexpr_func_binomial	356
\XINT_FL_fac_vbigloop_a	244	\XINT_flexpr_func_break	329
\XINT_FL_fac_vbigloop_loop	244, 245	\XINT_flexpr_func_ceil	352
\XINT_FL_mul_a	234	\XINT_flexpr_func_divmod	355
\XINT_FL_mul_b	234	\XINT_flexpr_func_even	353
\XINT_FL_pfac_a	249	\XINT_flexpr_func_factorial	354
\XINT_FL_pfac_addzeroes	249	\XINT_flexpr_func_first	360
\XINT_FL_pfac_addzeroes_exit	249	\XINT_flexpr_func_float	355
\XINT_FL_pfac_b	249	\XINT_flexpr_func_floor	352
\XINT_FL_pfac_bigloop	249, 250	\XINT_flexpr_func_frac	352
\XINT_FL_pfac_bigloop_a	250	\XINT_flexpr_func_gcd	358
\XINT_FL_pfac_bigloop_b	250	\XINT_flexpr_func_if	361
\XINT_FL_pfac_end	249--251	\XINT_flexpr_func_ifint	361
\XINT_FL_pfac_end_i	249--251	\XINT_flexpr_func_ifone	361
\XINT_FL_pfac_end_ii	249--251	\XINT_flexpr_func_ifsgn	362
\XINT_FL_pfac_end_iii	249, 251	\XINT_flexpr_func_iter	342
\XINT_FL_pfac_fork	248	\XINT_flexpr_func_iterr	345
\XINT_FL_pfac_increaseP	248, 249, 252	\XINT_flexpr_func_last	360
\XINT_FL_pfac_medloop	249, 250	\XINT_flexpr_func_lcm	358
\XINT_FL_pfac_medloop_a	250	\XINT_flexpr_func_len	360
\XINT_FL_pfac_medloop_b	250	\XINT_flexpr_func_max	358
\XINT_FL_pfac_neg	248	\XINT_flexpr_func_min	359
\XINT_FL_pfac_one	248	\XINT_flexpr_func_mod	356
\XINT_FL_pfac_outofrange	248	\XINT_flexpr_func_not	353
\XINT_FL_pfac_smallloop	249	\XINT_flexpr_func_num	351
\XINT_FL_pfac_smallloop_a	249	\XINT_flexpr_func_nuple	354
\XINT_FL_pfac_smallloop_b	249	\XINT_flexpr_func_odd	353
\XINT_FL_pfac_vbigloop	249--251	\XINT_flexpr_func_opxadd	339
\XINT_FL_pfac_vbigloop_a	250	\XINT_flexpr_func_opxmuls	339
\XINT_FL_pfac_zero	248	\XINT_flexpr_func_pfactorial	356
\XINT_FL_sqrt_a	253	\XINT_flexpr_func_preduce	351
\XINT_FL_sqrt_isneg	253	\XINT_flexpr_func_quo	357
\XINT_FL_sqrt_iszero	253	\XINT_flexpr_func_randrange	357
\XINT_FL_sqrt_pos	253, 254	\XINT_flexpr_func_reduce	351
\XINT_fladd_chkopt	232	\XINT_flexpr_func_rem	358
\XINT_fladd_noopt	232	\XINT_flexpr_func_reversed	361
\XINT_fladd_opt	232	\XINT_flexpr_func_round	355
\XINT_fladd_opt_a	232	\XINT_flexpr_func_rrseq	344
\XINT_flbinom_chkopt	251	\XINT_flexpr_func_rseq	340
\XINT_flbinom_noopt	251	\XINT_flexpr_func_seq	337
\XINT_flbinom_opt	251, 252	\XINT_flexpr_func_sgn	352
\XINT_fldiv_chkopt	234	\XINT_flexpr_func_sqr	353
\XINT_fldiv_noopt	234	\XINT_flexpr_func_sqrt	354
\XINT_fldiv_opt	234	\XINT_flexpr_func_subx	340
\XINT_fldiv_opt_a	234	\XINT_flexpr_func_trunc	355
\XINT_flen	180	\XINT_flexpr_func_xor	360
\XINT_flet_macro	38, 43, 44	\XINT_flexpr_noopt	297, 372

<code>\XINT_flexpr_op_!</code>	328	<code>\XINT_flpow_loopI_odd</code>	237
<code>\XINT_flexpr_op_!?</code>	335	<code>\XINT_flpow_loopII</code>	237, 238
<code>\XINT_flexpr_op:]</code>	318	<code>\XINT_flpow_loopII_even</code>	237
<code>\XINT_flexpr_op_?</code>	328	<code>\XINT_flpow_loopII_odd</code>	237
<code>\XINT_flexpr_op_["</code>	328	<code>\XINT_flpow_loopII_odda</code>	237, 238
<code>\XINT_flexpr_op_][:</code>	319	<code>\XINT_flpow_noopt</code>	235
<code>\XINT_flexpr_op_</code>	330	<code>\XINT_flpow_opt</code>	235
<code>\XINT_flexpr_print</code>	297	<code>\XINT_flpow_opt_a</code>	235
<code>\XINT_flexpr_until_end_a</code>	296	<code>\XINT_flpow_truncate</code>	236--238, 241, 242
<code>\XINT_flexpr_withopt_a</code>	297	<code>\XINT_flpow_truncate_a</code>	236, 237
<code>\XINT_flexpr_withopt_b</code>	297, 372	<code>\XINT_flpow_zero</code>	236, 241
<code>\XINT_flexpr_wrap</code>	297, 372	<code>\XINT_flpower_a</code>	241
<code>\XINT_flfac_chkopt</code>	242	<code>\XINT_flpower_aa</code>	241
<code>\XINT_flfac_noopt</code>	242, 243	<code>\XINT_flpower_ab</code>	241
<code>\XINT_flfac_opt</code>	242, 243	<code>\XINT_flpower_b</code>	241
<code>\XINT_flfac_opt_a</code>	243	<code>\XINT_flpower_BisZero</code>	239, 240
<code>\XINT_flmul_chkopt</code>	233	<code>\XINT_flpower_checkB_a</code>	239, 240
<code>\XINT_flmul_noopt</code>	233	<code>\XINT_flpower_checkB_b</code>	240
<code>\XINT_flmul_opt</code>	233, 234	<code>\XINT_flpower_checkB_c</code>	240
<code>\XINT_flmul_opt_a</code>	234	<code>\XINT_flpower_checkB_d</code>	240, 241
<code>\XINT_float_chkopt</code>	222	<code>\XINT_flpower_chkopt</code>	239, 240
<code>\XINT_float_neg</code>	223	<code>\XINT_flpower_IItoIII</code>	242
<code>\XINT_float_noopt</code>	223	<code>\XINT_flpower_ItoIII</code>	241
<code>\XINT_float_opt</code>	223	<code>\XINT_flpower_loopI</code>	241
<code>\XINT_float_opt_a</code>	223	<code>\XINT_flpower_loopI_even</code>	241
<code>\XINT_float_pos</code>	223	<code>\XINT_flpower_loopI_odd</code>	241
<code>\XINT_float_pos_done</code>	223, 256	<code>\XINT_flpower_loopII</code>	242
<code>\XINT_float_post</code>	223	<code>\XINT_flpower_loopII_even</code>	242
<code>\XINT_float_zero</code>	223	<code>\XINT_flpower_loopII_odd</code>	242
<code>\XINT_floate_chkopt</code>	255	<code>\XINT_flpower_loopII_odda</code>	242
<code>\XINT_floate_neg</code>	255	<code>\XINT_flpower_noopt</code>	240
<code>\XINT_floate_noopt</code>	255	<code>\XINT_flpower_opt</code>	240
<code>\XINT_floate_opt</code>	255	<code>\XINT_flpower_opt_a</code>	240
<code>\XINT_floate_opt_a</code>	255	<code>\XINT_flpower_toloopI</code>	241
<code>\XINT_floate_pos</code>	255, 256	<code>\XINT_flpower_toloopII</code>	241, 242
<code>\XINT_floate_post</code>	255	<code>\XINT_flpowerh_a</code>	239
<code>\XINT_floate_zero</code>	255	<code>\XINT_flpowerh_b</code>	239
<code>\XINT_flpfac_chkopt</code>	247, 248	<code>\XINT_flpowerh_c</code>	239
<code>\XINT_flpfac_noopt</code>	248	<code>\XINT_flpowerh_d</code>	239, 240
<code>\XINT_flpfac_opt</code>	248	<code>\XINT_flpowerh_e</code>	239, 240
<code>\XINT_flpfac_opt_b</code>	248	<code>\XINT_flpowerh_finish</code>	240
<code>\XINT_flpow_a</code>	236	<code>\XINT_flpowerh_i</code>	239
<code>\XINT_flpow_aa</code>	236	<code>\XINT_flpowerh_int</code>	239
<code>\XINT_flpow_ab</code>	236	<code>\XINT_flpowseries</code>	266
<code>\XINT_flpow_b</code>	236	<code>\XINT_flpowseries_chkopt</code>	265
<code>\XINT_flpow_BisZero</code>	235	<code>\XINT_flpowseries_dont_i</code>	266
<code>\XINT_flpow_checkB_a</code>	235	<code>\XINT_flpowseries_dont_ii</code>	266
<code>\XINT_flpow_checkB_b</code>	235, 236	<code>\XINT_flpowseries_exit_i</code>	266, 267
<code>\XINT_flpow_checkB_c</code>	236	<code>\XINT_flpowseries_exit_ii</code>	267
<code>\XINT_flpow_checkB_d</code>	236	<code>\XINT_flpowseries_loop_i</code>	266
<code>\XINT_flpow_chkopt</code>	235	<code>\XINT_flpowseries_loop_ii</code>	266, 267
<code>\XINT_flpow_III</code>	237, 238, 241, 242	<code>\XINT_flpowseries_loop_pre</code>	266, 267
<code>\XINT_flpow_IIIend</code>	238	<code>\XINT_flpowseries_noopt</code>	266
<code>\XINT_flpow_IItoIII</code>	237, 238	<code>\XINT_flpowseries_opt</code>	265, 266
<code>\XINT_flpow_ItoIII</code>	237	<code>\XINT_flpowseriesx</code>	267
<code>\XINT_flpow_loopI</code>	236, 237	<code>\XINT_flpowseriesx_chkopt</code>	267
<code>\XINT_flpow_loopI_even</code>	237	<code>\XINT_flpowseriesx_noopt</code>	267

\XINT_flpowseriesx_opt	267	\XINT_fmul_zero	209, 210
\XINT_flpowseriesx_pre	267	\XINT_for	38
\XINT_flseqa::csv	324	\XINT_for_d	38, 39
\XINT_flseqb::csv	325	\XINT_for_forever?	38
\XINT_flseqb::csv_a	325	\XINT_for_ifstar	38
\XINT_flseqb::csv_n	325	\XINT_for_last?	39
\XINT_flseqb::csv_p	325	\XINT_for_last?yes	39, 42, 43
\XINT_flseqb::csv_z	326	\XINT_forever	39, 40
\XINT_flseqb:f:csv	363	\XINT_forever_a	40, 41
\XINT_flseqb:f:csv_a	363	\XINT_forever_b	41
\XINT_flseqb:f:csv_bg	363	\XINT_forever_c	41
\XINT_flseqb:f:csv_bl	363	\XINT_forever_d	41
\XINT_flseqb:f:csv_n	364	\XINT_forever_opt_a	41
\XINT_flseqb:f:csv_na	363, 364	\XINT_forever_opt_c	41
\XINT_flseqb:f:csv_p	363	\XINT_forfour_d	42, 43
\XINT_flseqb:f:csv_pa	363	\XINT_forpair_d	42
\XINT_flseqb:f:csv_pb	363	\XINT_forthree_d	42
\XINT_flseqb:f:csv_pc	363	\XINT_forx	38, 39
\XINT_flsqrt	254	\XINT_forx_d	39
\XINT_flsqrt_a	254	\XINT_forx_empty?	39
\XINT_flsqrt_again	254, 255	\XINT_forx_forever?	39
\XINT_flsqrt_again_a	255	\XINT_fpow	210
\XINT_flsqrt_b	254, 255	\XINT_fpow_fork	210
\XINT_flsqrt_c	254	\XINT_fpow_neg	211
\XINT_flsqrt_chkopt	253	\XINT_fpow_pos	211
\XINT_flsqrt_d	254	\XINT_fpow_pos_A	211
\XINT_flsqrt_f	254	\XINT_fpow_pos_B	211
\XINT_flsqrt_finish	254	\XINT_fpow_zero	210, 211
\XINT_flsqrt_g	254	\XINT_fppowseries	264
\XINT_flsqrt_h	254	\XINT_fppowseries_dont_i	264
\XINT_flsqrt_noopt	253	\XINT_fppowseries_dont_ii	264
\XINT_flsqrt_opt	253	\XINT_fppowseries_exit_i	264, 265
\XINT_flsqrt_opt_a	253	\XINT_fppowseries_exit_ii	265
\XINT_flsub_chkopt	233	\XINT_fppowseries_loop_i	264
\XINT_flsub_noopt	233	\XINT_fppowseries_loop_ii	264, 265
\XINT_flsub_opt	233	\XINT_fppowseries_loop_pre	264, 265
\XINT_flsub_opt_a	233	\XINT_fppowseriesx	265
\XINT_fmax	217, 218	\XINT_fppowseriesx_pre	265
\XINT_fmax_A	218	\XINT_fprdxpr	212
\XINT_fmax_firstneg	218	\XINT_fprod_finished	212
\XINT_fmax_minusminus	218	\XINT_fprod_loop_a	212
\XINT_fmax_nonneg_a	218	\XINT_fprod_loop_b	212
\XINT_fmax_nonneg_b	218, 219	\XINT_fprod_loop_c	212
\XINT_fmax_secondneg	218	\XINT_frac_gen	182, 183
\XINT_fmin	218	\XINT_frac_gen_A	183
\XINT_fmin_A	219	\XINT_frac_gen_B	183
\XINT_fmin_firstneg	219	\XINT_frac_gen_Ba	183
\XINT_fmin_minusminus	219	\XINT_frac_gen_Bb	183
\XINT_fmin_nonneg_a	219	\XINT_frac_gen_Bc	183
\XINT_fmin_nonneg_b	218, 219	\XINT_frac_gen_Bd	183
\XINT_fmin_secondneg	219	\XINT_frac_gen_C	183
\XINT_fmul	209	\XINT_frac_gen_Ca	184
\XINT_fmul_a	209	\XINT_frac_gen_Cb	184
\XINT_fmul_b	209	\XINT_frac_gen_Cc	184
\XINT_fmul_c	209	\XINT_frac_gen_F	184
\XINT_fmul_d	209, 210	\XINT_frac_gen_G	184
\XINT_fmul_e	210	\XINT_frac_gen_Ga	184, 185

\XINT_frac_gen_Gdivbyzero	184	\XINT_fwover_B	192
\XINT_frac_gen_Gdivbyzero_a	184	\XINT_fwover_C	192
\XINT_frac_gen_zero	185	\XINT_fwover_D	192
\XINT_fracfrac_A	191, 192	\XINT_gcd_AisZero	166
\XINT_fracfrac_B	191	\XINT_gcd_BisZero	166
\XINT_fracfrac_C	191	\XINT_gcd_CheckRem	167
\XINT_fracfrac_D	191	\XINT_gcd_end	167
\XINT_fracfrac_E	191	\XINT_gcd_fork	166
\XINT_fracisone	216	\XINT_gcd_loop	166, 167
\XINT_fsqr	210	\XINT_gcdof_a	175
\XINT_fsqr_a	210	\XINT_gcdof_b	175
\XINT_fsqr_b	210	\XINT_gcdof_c	175
\XINT_fsqr_zero	210	\XINT_gcdof_d	175
\XINT_fsub	209	\XINT_gcdof_e	175
\XINT_fsub_a	209	\XINT_gcfrac	271
\XINT_fsub_Azero	209	\XINT_gcfrac_end	271
\XINT_fsub_b	209	\XINT_gcfrac_end_b	271
\XINT_fsum_finished	209	\XINT_gcfrac_endloop	271
\XINT_fsum_loop_a	209	\XINT_gcfrac_enter	271
\XINT_fsum_loop_b	209	\XINT_gcfrac_loop	271
\XINT_fsum_loop_c	209	\XINT_gcfrac_noopt	271
\XINT_fsumexpr	209	\XINT_gcfrac_opt_a	270
\XINT_ftc_A	273	\XINT_gcfrac_opt_b	271
\XINT_ftc_B	273	\XINT_gcfrac_optc	271
\XINT_ftc_C	273	\XINT_gcfrac_optl	271
\XINT_ftc_D	273	\XINT_gcfrac_optr	271
\XINT_ftc_integer	273	\XINT_gcfrac_T	271
\XINT_ftc_loop_a	273, 274	\XINT_gcfrac_U	271
\XINT_ftc_loop_d	273	\XINT_gcntf	286
\XINT_ftc_loop_e	273	\XINT_gcntf_exit	286, 287
\xint_ftc_loop_exit	274	\XINT_gcntf_loop	286, 287
\XINT_ftc_loop_f	274	\XINT_gcntgc	288
\XINT_ftcc_A	276	\XINT_gcntgc_exit	288, 289
\XINT_ftcc_B	276	\XINT_gcntgc_exit_b	289
\XINT_ftcc_C	276	\XINT_gcntgc_loop	288, 289
\XINT_ftcc_D	276	\XINT_gcntgc_loop_b	289
\XINT_ftcc_En	276	\XINT_gctcv_end	283, 284
\XINT_ftcc_end	277	\XINT_gctcv_loop_a	283, 284
\XINT_ftcc_Ep	276	\XINT_gctcv_loop_b	283
\XINT_ftcc_integer	276	\XINT_gctcv_loop_c	283
\XINT_ftcc_loop_a	276, 277	\XINT_gctcv_loop_d	283
\XINT_ftcc_loop_b	276	\XINT_gctcv_loop_e	283
\XINT_ftcc_loop_c	276, 277	\XINT_gctcv_loop_f	283
\XINT_ftcc_loop_d	277	\XINT_gctcv_loop_g	283
\XINT_ftcc_loop_N	277	\XINT_gctcv_loop_h	283
\XINT_ftcc_loop_P	277	\XINT_gctcv_loop_i	283
\XINT_ftcx_A	274	\XINT_gctcv_loop_j	283, 284
\XINT_ftcx_B	274	\XINT_gctcv_loop_k	284
\XINT_ftcx_C	274	\XINT_gctcv_loop_l	284
\XINT_ftcx_D	274	\XINT_gctcv_loop_m	284
\XINT_ftcx_integer	274	\XINT_gctcv_prep	282
\XINT_ftcx_loop_a	274	\XINT_gctf_end	279, 280
\XINT_ftcx_loop_d	274	\XINT_gctf_loop_a	279, 280
\XINT_ftcx_loop_e	274	\XINT_gctf_loop_b	279
\xint_ftcx_loop_exit	274	\XINT_gctf_loop_c	279
\XINT_ftcx_loop_f	274	\XINT_gctf_loop_d	279
\XINT_fwover_A	192, 193	\XINT_gctf_loop_e	279

\XINT_gctf_loop_f	279	\xint_gob_til_minus ..	10, 28, 29, 31, 49--51, 321
\XINT_gctf_loop_g	279	\xint_gob_til_one	10, 68, 95, 97
\XINT_gctf_loop_h	279	\xint_gob_til_R	9,
\XINT_gctf_loop_i	280	21, 24, 66, 68, 70, 71, 88, 89, 116, 155, 201	
\XINT_gctf_loop_j	280	\xint_gob_til_sc	10, 71, 74,
\XINT_gctf_prep	279	75, 77, 80--82, 85, 86, 97, 98, 100, 108, 247	
\XINT_gctgc_end	289, 290	\xint_gob_til_W	9, 298
\XINT_gctgc_end_b	290	\xint_gob_til_xint:	10, 12, 13,
\XINT_gctgc_loop_a	289	21, 23, 27, 31, 47, 48, 51, 52, 54, 60, 107,	
\XINT_gctgc_loop_b	289, 290	128, 129, 131, 132, 175, 209, 212, 218, 219	
\XINT_gctgc_loop_c	289	\xint_gob_til_Z	10, 24, 67, 116, 270
\XINT_gctgc_start	289	\xint_gob_til_zero ..	10, 15, 64, 86, 92, 95, 100, 101,
\XINT_gctgcx_end	273	120, 167, 172, 174, 182, 185, 191, 192, 195,	
\XINT_gctgcx_loop_a	273	197, 207--210, 216, 217, 232, 270, 273, 274	
\XINT_gctgcx_loop_b	273	\xint_gob_til_zeros_iii	10
\XINT_gctgcx_start	273	\xint_gob_til_zeros_iv	10
\XINT_geq	122	\XINT_gobble	15, 26, 28
\XINT_geq_finish	122	\xint_gobble_	9
\XINT_geq_firstiszero	122	\XINT_gobble_a	15
\XINT_geq_fork	122	\XINT_gobble_b	15
\XINT_geq_minusminus	122	\XINT_gobble_c	15
\XINT_geq_minusplus	122	\XINT_gobble_d	15
\XINT_geq_no	122	\xint_gobble_i ..	9, 11, 21, 35, 37, 40, 57, 58, 149,
\XINT_geq_plusminus	122	204, 206, 207, 296--299, 308, 309, 314, 321,	
\XINT_geq_plusplus	122, 130, 131, 217	328--330, 333, 338, 346, 347, 362, 363, 373	
\XINT_geq_secondiszero	122	\xint_gobble_ii ..	9, 21, 27, 191, 260, 261, 286, 287
\XINT_geq_yes	122	\xint_gobble_iii	9, 21, 35, 36, 175,
\XINT_ggcfrac	272	176, 275, 295, 296, 300, 319, 320, 324, 325	
\XINT_ggcfrac_end	272	\xint_gobble_iv	
\XINT_ggcfrac_end_b	272, 273	9, 21, 34, 112, 297, 346, 347, 362--364
\XINT_ggcfrac_endloop	272	\xint_gobble_thenstop	9, 48, 50
\XINT_ggcfrac_enter	272	\xint_gobble_v	9, 21
\XINT_ggcfrac_loop	272	\xint_gobble_vi	9, 21, 205
\XINT_ggcfrac_noopt	272	\xint_gobble_vii	9, 21
\XINT_ggcfrac_opt_a	272	\xint_gobble_viii	9, 21
\XINT_ggcfrac_opt_b	272	\XINT_half	63, 89, 241, 242
\XINT_ggcfrac_optc	272	\XINT_half_a	63
\XINT_ggcfrac_optl	272	\XINT_half_b	63
\XINT_ggcfrac_optr	272	\XINT_half_e	63
\XINT_ggcfrac_T	272	\XINT_half_fork	63
\XINT_ggcfrac_U	272	\XINT_half_neg	63
\XINT_global	18, 331, 332, 334, 365, 366, 373	\XINT_htb_checkin	163
\xint_gob_andstop_	9	\XINT_htb_cuz	163
\xint_gob_andstop_i	9	\XINT_htb_loop	163, 164
\xint_gob_andstop_ii	9	\XINT_htb_main	163
\xint_gob_andstop_iii	9	\XINT_htb_N	163
\xint_gob_andstop_iv	9	\XINT_htd_A	160, 161
\xint_gob_andstop_v	9	\XINT_htd_a	160, 161
\xint_gob_andstop_vi	9	\XINT_htd_Aa	161
\xint_gob_andstop_vii	9	\XINT_htd_Ab	161
\xint_gob_andstop_viii	9	\XINT_htd_again	160, 161
\xint_gob_til_!	294, 296, 299, 327	\XINT_htd_checkin	159
\xint_gob_til_dot	10, 90, 121, 203	\XINT_htd_finish	160, 161
\xint_gob_til_eightzeroes	10, 66, 67, 86, 95	\XINT_htd_finish_cuz	161
\xint_gob_til_exclam		\XINT_htd_main	159
.....	10, 96, 246, 271--273, 277--284, 289	\XINT_htd_neg	159
\xint_gob_til_G	10	\XINT_htd_nextfour	160, 161

\XINT_htd_starta	160, 162	\XINT_iidivision_aneg_end	88
\XINT_htd_starttb	160, 162	\XINT_iidivision_aneg_rpos	88
\XINT_htd_startba	160	\XINT_iidivision_aneg_rzero	88
\XINT_htd_startbb	160	\XINT_iidivision_apos	87, 88
\XINT_htd_unsep_loop	161	\XINT_iidivision_bneg	87
\XINT_htd_unsep_loop_a	161	\XINT_iidivision_bpos	87
\XINT_htd_update	160	\XINT_iidivision_divbyzero	87
\XINT_iadd	72	\XINT_iidivmod	104
\XINT_icmp	75	\XINT_iidivmod_a	104
\XINT_icstcv_end	282	\XINT_iidivmod_aiszero	104
\XINT_icstcv_loop_a	282	\XINT_iidivmod_bneg	104, 105
\XINT_icstcv_loop_b	282	\XINT_iidivmod_bneg_finish	105
\XINT_icstcv_loop_c	282	\XINT_iidivmod_bpos	104, 105
\XINT_icstcv_loop_d	282	\XINT_iidivmod_divbyzero	104
\XINT_icstcv_loop_e	282	\XINT_iidivround	102
\XINT_icstcv_prep	282	\XINT_iidivround_a	102
\XINT_icstf_end	278	\XINT_iidivround_aiszero	102--104
\XINT_icstf_loop_a	278	\XINT_iidivround_bneg	102
\XINT_icstf_loop_b	278	\XINT_iidivround_bpos	102
\XINT_icstf_loop_c	278	\XINT_iidivround_divbyzero	102--104
\XINT_icstf_prep	278	\XINT_iidivround_pos	102, 103
\XINT_idivround	102	\XINT_iidivtrunc	103
\XINT_iexpr_noopt	296	\XINT_iidivtrunc_a	103
\XINT_iexpr_withopt	296	\XINT_iidivtrunc_aiszero	103
\XINT_iexpr_wrap	296	\XINT_iidivtrunc_bneg	103
\XINT_ifFlagRaised	58, 112	\XINT_iidivtrunc_bpos	103
\XINT_iffloatint	257	\XINT_iidivtrunc_divbyzero	103
\XINT_ifint	195	\XINT_iidivtrunc_pos	103
\XINT_ifloor	190	\XINT_ii_e_a	116
\XINT_igctcv_end_a	284, 285	\XINT_ii_e_fork	116
\XINT_igctcv_end_b	285	\XINT_ii_e_neg	116
\XINT_igctcv_loop_a	284, 285	\XINT_iiexpr_func_!	353
\XINT_igctcv_loop_b	284	\XINT_iiexpr_func_?	353
\XINT_igctcv_loop_c	284	\XINT_iiexpr_func_@@	336
\XINT_igctcv_loop_f	284	\XINT_iiexpr_func_@@@	336
\XINT_igctcv_loop_g	284	\XINT_iiexpr_func_@@@	336
\XINT_igctcv_loop_h	284, 285	\XINT_iiexpr_func_abs	351
\XINT_igctcv_loop_i	285	\XINT_iiexpr_func_all	360
\XINT_igctcv_loop_k	285	\XINT_iiexpr_func_any	360
\XINT_igctcv_loop_l	285	\XINT_iiexpr_func_binomial	356
\XINT_igctcv_prep	284	\XINT_iiexpr_func_break	329
\XINT_igctf_end	280, 281	\XINT_iiexpr_func_ceil	352
\XINT_igctf_loop_a	280, 281	\XINT_iiexpr_func_divmod	355
\XINT_igctf_loop_b	280	\XINT_iiexpr_func_even	353
\XINT_igctf_loop_c	280	\XINT_iiexpr_func_factorial	354
\XINT_igctf_loop_f	280	\XINT_iiexpr_func_first	360
\XINT_igctf_loop_g	280	\XINT_iiexpr_func_float	355
\XINT_igctf_loop_h	280	\XINT_iiexpr_func_floor	352
\XINT_igctf_loop_i	280, 281	\XINT_iiexpr_func_gcd	358
\XINT_igctf_prep	280	\XINT_iiexpr_func_if	361
\XINT_iiadd	72	\XINT_iiexpr_func_ifint	361
\XINT_iicmp	75	\XINT_iiexpr_func_ifone	361
\XINT_iidivision	87	\XINT_iiexpr_func_ifsgn	362
\XINT_iidivision_a	87	\XINT_iiexpr_func_iter	342
\XINT_iidivision_aiszero	87	\XINT_iiexpr_func_iterr	345
\XINT_iidivision_aneg	87, 88	\XINT_iiexpr_func_last	360
\XINT_iidivision_aneg_b	88	\XINT_iiexpr_func_lcm	358

\XINT_iiexpr_func_len	360	\XINT_iimodtrunc_pos	104
\XINT_iiexpr_func_max	358	\XINT_iimul	83
\XINT_iiexpr_func_min	359	\XINT_iiseq::csv	323
\XINT_iiexpr_func_mod	356	\XINT_iiseq::csv_n	323, 324
\XINT_iiexpr_func_not	353	\XINT_iiseq::csv_p	323
\XINT_iiexpr_func_num	351	\XINT_iiseq::csv_z	323
\XINT_iiexpr_func_nuple	354	\XINT_iiseqa::csv	324
\XINT_iiexpr_func_odd	353	\XINT_iiseqb::csv	325
\XINT_iiexpr_func_opxadd	339	\XINT_iiseqb::csv_a	325
\XINT_iiexpr_func_opxmul	339	\XINT_iiseqb::csv_n	325
\XINT_iiexpr_func_pfactorial	357	\XINT_iiseqb::csv_p	325
\XINT_iiexpr_func_quo	357	\XINT_iiseqb::csv_z	325
\XINT_iiexpr_func_randrange	357	\XINT_iiseqb:f:csv	363
\XINT_iiexpr_func_rem	358	\XINT_iiseqb:f:csv_a	363
\XINT_iiexpr_func_reversed	361	\XINT_iiseqb:f:csv_bg	363
\XINT_iiexpr_func_round	355	\XINT_iiseqb:f:csv_bl	363
\XINT_iiexpr_func_rrseq	344	\XINT_iiseqb:f:csv_n	363
\XINT_iiexpr_func_rseq	340	\XINT_iiseqb:f:csv_na	363
\XINT_iiexpr_func_seqx	337	\XINT_iiseqb:f:csv_p	363
\XINT_iiexpr_func_sgn	352	\XINT_iiseqb:f:csv_pa	363
\XINT_iiexpr_func_sqr	353	\XINT_iiseqb:f:csv_pb	363
\XINT_iiexpr_func_sqrt	354	\XINT_iiseqb:f:csv_pc	363
\XINT_iiexpr_func_sqrttr	354	\XINT_iisub	77
\XINT_iiexpr_func_subx	340	\XINT_inc	63, 64
\XINT_iiexpr_func_trunc	355	\XINT_inc_a	63
\XINT_iiexpr_func_xor	360	\XINT_inc_e	63, 65
\XINT_iiexpr_op_!	328	\XINT_inc_fork	63
\XINT_iiexpr_op_!?	335	\XINT_inc_neg	63
\XINT_iiexpr_op:]	318	\XINT_Inexact.handler	58
\XINT_iiexpr_op_?	328	\XINT_infloat	224
\XINT_iiexpr_op_[]	328	\XINT_infloat::_b	299
\XINT_iiexpr_op_][:	319	\XINT_infloat::_c	299
\XINT_iiexpr_op_~	330	\XINT_infloat::_d	299
\XINT_iiexpr_print	295, 372	\XINT_infloat::_e	299
\XINT_iiexpr_until_end_a	296	\XINT_infloat_a	224
\XINT_iiexpr_wrap	295, 296	\XINT_infloat_clean	224
\XINT_iigcd	166	\XINT_infloat_clean_a	224
\XINT_iigeq	122	\XINT_infloat_done	224
\XINT_iilcm	167	\XINT_infloat_fork	224, 226
\xint_iimax	129	\XINT_infloat_J	226
\XINT_iimaxof_a	131	\XINT_infloat_K	226
\XINT_iimaxof_b	131	\XINT_infloat_L	226
\XINT_iimaxof_c	131	\XINT_infloat_Ma	226
\XINT_iimaxof_d	131	\XINT_infloat_Mb	226
\XINT_iimaxof_e	131	\XINT_infloat_MtoN	226, 227
\xint_iimin	130	\XINT_infloat_N	227
\XINT_iiminof_a	131	\XINT_infloat_Oa	227
\XINT_iiminof_b	131	\XINT_infloat_Ob	227
\XINT_iiminof_c	131	\XINT_infloat_P	227
\XINT_iiminof_d	131	\XINT_infloat_Q	228
\XINT_iiminof_e	131	\XINT_infloat_Qq	228
\XINT_iimodtrunc	103	\XINT_infloat_R	228
\XINT_iimodtrunc_a	103, 104	\XINT_infloat_Rq	228
\XINT_iimodtrunc_aiszero	104	\XINT_infloat_Sa	228
\XINT_iimodtrunc_bneg	104	\XINT_infloat_Sb	228
\XINT_iimodtrunc_bpos	104	\XINT_infloat_SEq	228
\XINT_iimodtrunc_divbyzero	104	\XINT_infloat_sp	224

\XINT_infloat_sp_b	225	\XINT_irr_negative	194
\XINT_infloat_sp_c	225	\XINT_irr_nonneg	194
\XINT_infloat_sp_needzeros	225	\XINT_irr_start	194
\XINT_infloat_sp_quick	225	\XINT_irr_zero	195
\XINT_infloat_sp_round	225, 226	\XINT_is_One	123, 198, 200, 224
\XINT_infloat_spneg	225	\XINT_isbalanced_a	336, 337
\XINT_infloat_spneg_needzeros	225	\XINT_isbalanced_b	336, 337
\XINT_infloat_spnegend	225	\XINT_isbalanced_c	337
\XINT_infloat_sppos	225	\XINT_isbalanced_d	337
\XINT_infloat_spzero	224, 225	\XINT_isbalanced_error	337
\XINT_infloat_SUp	228, 229	\XINT_isbalanced_no	337
\XINT_infloat_SY	228	\XINT_isbalanced_yes	337
\XINT_infloat_X	228, 229	\XINT_iseries	260
\XINT_infloat_Y	225, 229	\XINT_iseries_exit	261
\XINT_infloat_Z	229	\XINT_iseries_loop	261
\XINT_infloat_ZZ	229	\XINT_isOne	123, 188, 189, 191, 192, 194, 196, 241, 242
\XINT_infloatdivmod	256	\XINT_isone	123
\XINT_infloate	256	\XINT_istrue::a	298
\XINT_infloate_end	256	\XINT_istrue::b	298, 299
\XINT_infloatfracdg_a	231	\XINT_istrue::c	298
\XINT_infloatS_clean	224	\XINT_istrue::d	298, 299
\XINT_infloatS_clean_a	224	\XINT_istrue::e	298, 299
\XINT_inFrac	182	\XINT_isub	78
\XINT_infrac	180, 182, 188--194, 198, 200, 210, 212, 224	\XINT_item	37
\XINT_infrac_fork	182	\xint_itemcount	45
\XINT_infrac_res_a	182	\XINT_itrunc_G	198--200
\XINT_infrac_res_b	182	\XINT_jrr_D	196
\XINT_infrac_res_ca	182	\XINT_jrr_denomisone	196
\XINT_infrac_res_cb	182	\XINT_jrr_divisionbyzero	196
\XINT_infrac_res_cc	182	\XINT_jrr_indeterminate	196
\XINT_infrac_res_zero	182	\XINT_jrr_loop_a	196, 197
\XINT_inline_b	37	\XINT_jrr_loop_b	196
\XINT_inline_d	37	\XINT_jrr_loop_c	196, 197
\XINT_inline_e	37	\XINT_jrr_loop_d	197
\XINT_inline_f	37	\XINT_jrr_loop_e	197
\XINT_inline_g	37	\XINT_jrr_loop_exit	197
\XINT_inline_macro	37	\XINT_jrr_negative	196
\XINT_inline_w	37	\XINT_jrr_nonneg	196
\XINT_inrandomfloatS	257	\XINT_jrr_start	196
\XINT_inrandomfloatS_a	257, 258	\XINT_jrr_zero	196
\XINT_inrandomfloatS_b	257	\XINT_keep:f:csv_a	48
\XINT_inrandomfloatS_zero	257, 258	\XINT_keep:f:csv_keepall	49
\XINT_InvalidContext-signal	57	\XINT_keep:f:csv_keepnone	48, 49
\XINT_InvalidContext.handler	58	\XINT_keep:f:csv_loop	49--51
\XINT_InvalidOperation.handler	58	\XINT_keep:f:csv_loop_end	50
\XINT_iround_A	200	\XINT_keep:f:csv_loop_pickeight	50
\XINT_irr_D	194	\XINT_keep:f:csv_neg	49
\XINT_irr_denomisone	194	\XINT_keep:f:csv_neg_a	49
\XINT_irr_divisionbyzero	195	\XINT_keep:f:csv_neg_done	49
\XINT_irr_finish	195, 197	\XINT_keep:f:csv_pos	49
\XINT_irr_indeterminate	195	\XINT_keep:f:csv_pos_fork	49
\XINT_irr_loop_a	195	\XINT_keep:f:csv_pos_keepall	49, 50
\XINT_irr_loop_d	195	\XINT_keep:f:csv_trimloop	49, 52
\XINT_irr_loop_e	195	\XINT_keep:f:csv_trimloop_finish	49
\XINT_irr_loop_exit	195	\XINT_keep:f:csv_trimloop_trimmnine	49
\XINT_irr_loop_exitb	195	\XINT_keep:x:csv_finish	321

\XINT_keep:x:csv_loop	321	\XINT_lengthupto_finish_a	13
\XINT_keep:x:csv_loop_pickeight	321	\XINT_lengthupto_finish_b	13
\XINT_keep:x:csv_pos	321	\XINT_lengthupto_gt	13
\XINT_keep_a	27	\XINT_lengthupto_loop	13, 28, 29
\XINT_keep_keepall	28	\XINT_lengthupto_loop_a	13
\XINT_keep_keepnone	27, 29	\XINT_listsel:_:	320
\XINT_keep_loop	28, 30	\XINT_listsel:_:a	320
\XINT_keep_loop_end	28	\XINT_listsel:_N:N	320
\XINT_keep_loop_pickeight	28	\XINT_listsel:_N:N_a	320
\XINT_keep_neg	27, 29	\XINT_listsel:_N:N_abort	320
\XINT_keep_neg_a	27	\XINT_listsel:_N:P	320
\XINT_keep_pos	27, 28	\XINT_listsel:_N:P_a	320, 321
\XINT_keepunbr_a	29	\XINT_listsel:_nth	319, 320
\XINT_keepunbr_loop	29, 32	\XINT_listsel:_O:P	321
\XINT_keepunbr_loop_end	29	\XINT_listsel:_P:N	320, 321
\XINT_keepunbr_loop_pickeight	29	\XINT_listsel:_P:N_a	321
\XINT_keepunbr_pos	29	\XINT_listsel:_P:O	321
\XINT_last:f:csv_loop	54	\XINT_listsel:_P:P	320, 321
\XINT_last_loop	12, 13	\XINT_listsel:_s	319, 320
\XINT_last_loop_enda	12, 13, 54	\XINT_listxsel:_:	319
\XINT_last_loop_endb	12, 13, 54	\XINT_listxsel:_N:N	319
\XINT_last_loop_endc	12, 13, 54	\XINT_listxsel:_N:N_a	319
\XINT_last_loop_endd	12, 13, 54	\XINT_listxsel:_N:P	319
\XINT_last_loop_ende	12, 13, 54	\XINT_listxsel:_N:P_a	319, 320
\XINT_last_loop_endf	12, 13, 54	\XINT_listxsel:_O:P	320
\XINT_last_loop_endg	12, 13, 54	\XINT_listxsel:_P:N	319, 320
\XINT_last_loop_endh	12, 13, 54	\XINT_listxsel:_P:N_a	320
\XINT_lcm_AisZero	167	\XINT_listxsel:_P:O	320
\XINT_lcm_BisZero	167	\XINT_listxsel:_P:P	319, 320
\XINT_lcm_fork	167	\XINT_lws	25
\XINT_lcm_notzero	167	\XINT_lws_e	25, 26
\XINT_lcmof_a	175	\XINT_lws_e_i	25, 26
\XINT_lcmof_b	175	\XINT_lws_e_ii	25, 26
\XINT_lcmof_c	175	\XINT_lws_e_iii	25, 26
\XINT_lcmof_d	175	\XINT_lws_e_iv	25
\XINT_lcmof_e	175	\XINT_lws_e_v	25
\XINT_ldg	62	\XINT_lws_e_vi	25
\XINT_ldg_a	62	\XINT_lws_loop_a	25
\XINT_ldg_b	62	\XINT_lws_loop_b	25, 26
\XINT_ldg_c	62	\XINT_max_fork	129
\XINT_ldg_cbye	62	\XINT_max_minusminus	129, 130
\XINT_ldg_d	62	\XINT_max_minusplus	129, 130
\XINT_ldg_fork	62	\XINT_max_plusminus	129, 130
\XINT_len_fork	115, 180	\XINT_max_plusplus	129, 130
\XINT_length:f:csv_a	47, 49, 51, 52	\XINT_max_pluszero	129, 130
\XINT_length:f:csv_finish	47	\XINT_max_zeroplus	129, 130
\XINT_length:f:csv_loop	47	\XINT_max_zerozero	129
\XINT_length_finish_a	12	\XINT_maxof_a	218
\XINT_length_loop	12, 26, 27, 30, 31, 115, 118, 183, 184, 199	\XINT_maxof_b	218
\XINT_lengthupto:f:csv_a	47--49	\XINT_maxof_c	218
\XINT_lengthupto:f:csv_empty	48	\XINT_maxof_d	218
\XINT_lengthupto:f:csv_finish_a	48	\XINT_maxof_e	218
\XINT_lengthupto:f:csv_finish_b	48	\XINT_microrevsep	116
\XINT_lengthupto:f:csv_finish_c	48	\XINT_microrevsep_end	115, 116
\XINT_lengthupto:f:csv_gt	48	\XINT_min_fork	130
\XINT_lengthupto:f:csv_loop_a	48	\XINT_min_minusminus	130, 131
\XINT_lengthupto:f:csv_loop_b	48	\XINT_min_minusplus	130, 131

\XINT_min_plusminus	130, 131	\XINT_NewExpr	372, 373
\XINT_min_plusplus	130, 131	\XINT_NewExpr_a	373
\XINT_min_pluszero	130, 131	\XINT_newexpr_clean	372
\XINT_min_zeroplus	130, 131	\XINT_NewFloatFunc	365, 373
\XINT_min_zerozero	130, 131	\XINT_NewFunc	364, 373
\XINT_minimul_a	86, 100, 101, 106	\XINT_newfunc_clean	373
\XINT_minimul_b	86	\XINT_NewIIFunc	365, 373
\XINT_minimul_c	86, 87	\XINT_nthelt:f:csv_a	52
\XINT_minimulwc_a	85, 86	\XINT_nthelt:f:csv_neg	52
\XINT_minimulwc_b	85	\XINT_nthelt:f:csv_neg_done	52
\XINT_minimulwc_c	85, 86	\XINT_nthelt:f:csv_neg_fork	52
\XINT_minof_a	219	\XINT_nthelt:f:csv_pos	52
\XINT_minof_b	219	\XINT_nthelt:f:csv_pos_cleanup	52, 53
\XINT_minof_c	219	\XINT_nthelt:f:csv_pos_done	52, 53
\XINT_minof_d	219	\XINT_nthelt_a	26
\XINT_minof_e	219	\XINT_nthelt_neg	26
\XINT_mod_a	215	\XINT_nthelt_neg_a	26
\XINT_mod_aiszero	215	\XINT_nthelt_neg_b	26
\XINT_mod_b	215	\XINT_nthelt_pos	26, 27
\XINT_mod_bneg	215	\XINT_nthelt_pos_done	27
\XINT_mod_bpos	215	\XINT_nthelt_zero	26
\XINT_mod_bpos_a	215, 216	\XINT_num	59, 121
\XINT_mod_D_a	216	\XINT_num_cleanup	59, 60, 82, 184
\XINT_mod_D_b	214--216	\XINT_num_end	60
\XINT_mod_D_c	216	\XINT_num_loop	59, 60, 82, 184
\XINT_mod_D_exit	216	\XINT_numer	190
\XINT_mod_divbyzero	215	\XINT_numer_A	190
\XINT_mod_E	216	\XINT_numer_B	190
\XINT_mod_F	216	\XINT_oncsv:_a	348--350
\XINT_modtrunc_a	213	\XINT_oncsv:_b	348
\XINT_modtrunc_aiszero	213, 215	\XINT_oncsv:_c	348
\XINT_modtrunc_b	213	\XINT_oncsv:_d	348
\XINT_modtrunc_bneg	213	\XINT_oncsv:_e	348
\XINT_modtrunc_bpos	213	\XINT_oncsv:_empty	348
\XINT_modtrunc_divbyzero	213, 215	\XINT_oncsv:_end	348
\XINT_modtrunc_pos	213, 214	\XINT_Opp	61
\XINT_modtrunc_pos_a	214	\XINT_opp	61, 72, 76, 78, 79, 87, 107, 181, 221
\XINT_mul_a	85	\XINT_orof:_a	348
\XINT_mul_b	85	\XINT_orof:_c	348
\XINT_mul_checklengths	84	\XINT_orof:_e	348
\XINT_mul_e	85	\XINT_orof:_yes	348
\XINT_mul_exchange	84	\XINT_orof_a	128
\XINT_mul_fork	83, 132, 196, 278--284	\XINT_orof_b	128
\XINT_mul_loop	85, 106, 108	\XINT_orof_c	128
\XINT_mul_minusminus	83	\XINT_orof_e	128
\XINT_mul_minusplus	83	\XINT_orof_yes	128
\XINT_mul_nfork	83	\xint_orthat	10, 57, 58, 64, 87, 102--104, 110, 123, 133--135, 137, 138, 140, 146, 150, 151, 157, 158, 170, 185, 189, 198--200, 213--215, 224, 228, 230, 232, 236, 241--243, 248, 249, 252, 254, 257, 258, 298--309, 314, 319, 320, 329, 338, 339, 341, 343--347, 369--371
\XINT_mul_oneisone	84	\XINT_outfrac	180, 210--212
\XINT_mul_out	84, 85, 106, 108--111, 148, 244	\XINT_outfrac_divisionbyzero	180
\XINT_mul_plusminus	83	\XINT_outfrac_N	180, 181
\XINT_mul_plusplus	83, 170, 171	\XINT_outfrac_N_a	181
\XINT_mul_pre_b	83, 84	\XINT_outfrac_P	180, 181
\XINT_mul_smallbyfirst	84		
\XINT_mul_smallbysecond	84		
\XINT_mul_start	84, 85		
\XINT_mul_verysmall	84		
\XINT_mul_zero	83		

\XINT_outfrac_Zero	181	\xint_pow	106, 210
\XINT_Overflow.handler	58	\XINT_pow_AatleastTwo	107
\XINT_pfac_a	146	\XINT_pow_AisOne	107
\XINT_pfac_b	146	\XINT_pow_AisZero	107
\XINT_pfac_bigloop	146, 147	\XINT_pow_Aneg	107
\XINT_pfac_bigloop_a	147	\XINT_pow_Apos	107
\XINT_pfac_bigloop_b	147	\XINT_pow_Apos_a	107
\XINT_pfac_end_	146--148	\XINT_pow_Apos_short	107
\XINT_pfac_end_i	146--148	\XINT_pow_BisNegative	107
\XINT_pfac_end_ii	146--148	\XINT_pow_BisZero	107, 108
\XINT_pfac_end_iii	146, 148	\XINT_pow_I_exit	108
\XINT_pfac_fork	145, 211, 212	\XINT_pow_I_in	107, 108
\XINT_pfac_medloop	146, 147	\XINT_pow_I_loop	108
\XINT_pfac_medloop_a	147	\XINT_pow_I_squareit	108
\XINT_pfac_medloop_b	147	\XINT_pow_II_even	109
\XINT_pfac_neg	145, 146	\XINT_pow_II_exit	109
\XINT_pfac_one	145, 146	\XINT_pow_II_in	108
\XINT_pfac_outofrange	146	\XINT_pow_II_loop	108, 109
\XINT_pfac_smallloop	146	\XINT_pow_II_odda	109
\XINT_pfac_smallloop_a	146	\XINT_pow_II_oddb	109
\XINT_pfac_smallloop_b	146	\XINT_pow_mul_small	108
\XINT_pfac_vbigloop	146--148	\XINT_pow_mulbutcheckifsmall	108, 109
\XINT_pfac_vbigloop_a	147	\XINT_powseries	261
\XINT_pfac_zero	146	\XINT_powseries_exit_i	261
\XINT_pfloat::_b	299	\XINT_powseries_exit_ii	262
\XINT_pfloat::_c	299	\XINT_powseries_loop_i	261, 262
\XINT_pfloat::_d	299	\XINT_powseries_loop_ii	261
\XINT_pfloat::_e	299	\XINT_powseriesx	262
\XINT_pfloat_a	230	\XINT_powseriesx_pre	262
\XINT_pfloat_chkopt	230	\XINT_praw	188
\XINT_pfloat_N	230	\XINT_praw_A	188
\XINT_pfloat_N_i	231	\XINT_praw_a	188
\XINT_pfloat_N_ii	231	\XINT_prdexpr	132
\XINT_pfloat_N_iii	231	\XINT_prod_finished	132
\XINT_pfloat_N_iv	231	\XINT_prod_loop_a	132
\XINT_pfloat_N_v	231	\XINT_prod_loop_b	132
\XINT_pfloat_neg	230	\XINT_prod_loop_c	132
\XINT_pfloat_no	230	\XINT_protectii	295, 297
\XINT_pfloat_noopt	230	\XINT_providespackage	8,
\XINT_pfloat_opt	230, 299	20, 56, 114, 154, 166, 179, 260, 269, 294	
\XINT_pfloat_opt_a	230	\XINT_RaiseFlag	58
\XINT_pfloat_P	230, 231	\XINT_randomdigits	148
\XINT_pfloat_P_	231	\XINT_randomdigits_a	148
\XINT_pfloat_P_i	231	\XINT_randrange	150, 152
\XINT_pfloat_P_ii	231	\XINT_randrange_A	151
\XINT_pfloat_P_iii	231	\XINT_randrange_a	150
\XINT_pfloat_P_iv	231	\XINT_randrange_again	151
\XINT_pfloat_P_v	231	\XINT_randrange_B	151
\XINT_pfloat_pos	230	\XINT_randrange_b	150
\XINT_pfloat_Ps	230, 231	\XINT_randrange_c	150, 151
\XINT_pfloat_Psi	231	\XINT_randrange_d	151
\XINT_pfloat_Psii	231	\XINT_randrange_E	151
\XINT_pfloat_Psiii	231	\XINT_randrange_e	151
\XINT_pfloat_Psiv	231	\XINT_randrange_err:empty	150
\XINT_pfloat_Psv	231	\XINT_randrange_Z	151
\XINT_pfloat_zero	230	\XINT_randrangeAtoB_a	150
\XINT_pirr_start	194	\XINT_ratseries	262

\XINT_ratseries_exit_i	263	\XINT_Rounded.handler	58
\XINT_ratseries_exit_ii	263	\XINT_rsepbyviii	69, 70
\XINT_ratseries_exit_iii	263	\XINT_rsepbyviii_b	69
\XINT_ratseries_loop	263, 264	\XINT_rsepbyviii_end_A	
\XINT_ratseriesx	263	.. 70, 72, 73, 78, 83, 84, 90, 105, 108, 203	
\XINT_ratseriesx_pre	263, 264	\XINT_rsepbyviii_end_B	
\XINT_raw	188	.. 70, 72, 73, 78, 83, 84, 90, 105, 108, 203	
\XINT_rawz_A	189	\XINT_sdiv_out	90, 99
\XINT_rawz_Ba	189	\xint_secondofthree	114, 180, 308
\XINT_rawz_Bb	189	\xint_secondofthree_thenstop ..	114, 124, 125, 186
\XINT_rawz_fork	189	\xint_secondoftwo	9, 17, 38,
\XINT_rdg	149	39, 41--43, 58, 76, 77, 128, 157, 167, 188,	
\XINT_rdg_aux	149	232, 247, 303, 330, 350, 357, 368, 369, 372	
\XINT_rep .. 14, 116, 120, 204, 206, 207, 236, 241		\xint_secondoftwo_thenstop ...	9, 102, 104, 105,
\XINT_rep_	14	119, 125--127, 129--131, 187, 188, 195, 257	
\XINT_rep_a	14	\XINT_sepandrev	70, 108
\XINT_rep_neg	14	\XINT_sepandrev_a	70
\XINT_replicate	14, 149	\XINT_sepandrev_andcount	
\XINT_resetFlag	58 70, 72, 73, 78, 83, 84, 90, 105, 203	
\XINT_resetFlags	58	\XINT_sepandrev_andcount_a	70
\XINT_restorecatcodes	6, 7	\XINT_sepandrev_andcount_b	70, 71
\XINT_restorecatcodes_endinput	6, 7,	\XINT_sepandrev_andcount_done	71
18, 54, 112, 152, 164, 177, 258, 267, 290, 375		\XINT_sepandrev_andcount_end	70, 71
\XINT_restoreescapechar	44, 45	\XINT_sepandrev_b	70
\XINT_rev_nounsep	71, 93--96, 98, 143	\XINT_sepandrev_done	70
\XINT_rev_nounsep_done	71	\XINT_sepandrev_end	70
\XINT_rev_nounsep_end	71	\XINT_sepbyviii	69
\XINT_revdigits	115	\XINT_sepbyviii_andcount	69, 76, 90
\XINT_revdigits_a	115	\XINT_sepbyviii_andcount_a	69
\XINT_revdigits_b	116	\XINT_sepbyviii_andcount_b	69
\XINT_revdigits_end	116	\XINT_sepbyviii_andcount_end	69
\XINT_reverse:f:csv_cleanup	53	\XINT_sepbyviii_end	69, 76, 90
\XINT_reverse:f:csv_finish	53	\XINT_sepbyviii_Z	68, 89
\XINT_reverse:f:csv_loop	53	\XINT_sepbyviii_Z_end	68, 89, 202
\XINT_revwbr_finish_a	21	\XINT_seq	33
\XINT_revwbr_finish_b	21	\XINT_seq::csv	322
\XINT_revwbr_finish_c	21	\XINT_seq::csv_e	323, 324
\XINT_revwbr_loop	21	\XINT_seq::csv_n	323
\XINT_rez_A	193	\XINT_seq::csv_p	322, 323
\XINT_rez_AB	193, 197	\XINT_seq::csv_z	322, 323
\XINT_rez_B	193	\XINT_seq_chkopt	33
\XINT_rez_C	193	\XINT_seq_e	33
\XINT_rez_D	193	\XINT_seq_n	33
\XINT_rez_E	193	\XINT_seq_noopt	33
\XINT_rez_neg	193	\XINT_seq_opt	33, 34
\XINT_rez_zero	193	\XINT_seq_p	33
\XINT_rord_cleanup	12	\XINT_seqa::csv	324
\XINT_rord_main	11, 12, 172, 174	\XINT_seqa::csv_a	324
\XINT_round	200	\XINT_seqb::csv	324
\XINT_round::b	299	\XINT_seqb::csv_a	324
\XINT_round::c	299	\XINT_seqb::csv_n	325
\XINT_round::d	299	\XINT_seqb::csv_p	324
\XINT_round::e	299	\XINT_seqb::csv_z	325
\XINT_round_A	200	\XINT_seqb:f:csv	362
\XINT_round_a	200	\XINT_seqb:f:csv_a	362
\XINT_round_aa	200	\XINT_seqb:f:csv_be	362, 363
\XINT_round_B	200, 201	\XINT_seqb:f:csv_bg	362

\XINT_seqb:f:csv_bl	362	\XINT_split_fromleft	117, 118, 138, 204, 206, 226--228, 236, 254
\XINT_seqb:f:csv_n	362	\XINT_split_fromleft_a	117
\XINT_seqb:f:csv_na	362	\XINT_split_fromleft_b	117
\XINT_seqb:f:csv_nb	362--364	\XINT_split_fromleft_clean	117
\XINT_seqb:f:csv_nc	362--364	\XINT_split_fromleft_end_a	117
\XINT_seqb:f:csv_p	362	\XINT_split_fromleft_end_b	117, 118
\XINT_seqb:f:csv_pa	362	\XINT_split_fromright	117, 118
\XINT_seqb:f:csv_pb	362	\XINT_split_fromright_a	118, 199
\XINT_seqb:f:csv_pc	362	\XINT_split_fromright_Empty	118
\XINT_seqo	34	\XINT_split_xfork	117, 118, 121, 204, 207
\XINT_seqo_a	34	\XINT_split_zerosplit	117
\XINT_seqo_na	34	\XINT_splitl_finish	118
\XINT_seqo_nb	35	\XINT_splitr_finish	118
\XINT_seqo_nc	35	\XINT_spraw	298
\XINT_seqo_nd	35	\XINT_spraw::a	298
\XINT_seqo_o	34, 35	\XINT_spraw::b	298
\XINT_seqo_pa	34	\XINT_spraw::c	298
\XINT_seqo_pb	34	\XINT_spraw::d	298
\XINT_seqo_pc	34	\XINT_spraw::e	298
\XINT_seqo_pd	34	\XINT_spraw_a	298
\XINT_series	260	\XINT_spraw_p	298
\XINT_series_exit	260	\XINT_sptoken	20, 36, 38
\XINT_series_loop	260	\XINT_sqr	105
\XINT_setcatcodes	7	\XINT_sqr_a	105, 106
\XINT_Sgn	60, 88, 139, 181, 198, 200	\XINT_sqr_small	106
\XINT_sgn	60, 221	\XINT_sqr_small_out	106
\XINT_sgnfrac_a	191	\XINT_sqr_start	106
\XINT_sgnfrac_b	191	\XINT_sqr_verysmall	106
\XINT_sgnfrac_N	191, 192	\XINT_sqrt	133
\XINT_sgnfrac_P	191, 192	\XINT_sqrt_a	133
\XINT_sgnfwover_a	192	\XINT_sqrt_bE	133
\XINT_sgnfwover_b	192	\XINT_sqrt_big_a	133
\XINT_sgnfwover_N	192, 193	\XINT_sqrt_big_d	133, 135
\XINT_sgnfwover_P	192, 193	\XINT_sqrt_big_eE	135
\XINT_signalcondition	57, 87, 102, 104, 107, 110, 111, 133, 140, 146, 181, 184, 195, 196, 213, 214, 236, 243, 248, 252, 253	\XINT_sqrt_big_eE_a	135
\XINT_signalcondition_a	57	\XINT_sqrt_big_end	139
\XINT_smallldiv_c	100	\XINT_sqrt_big_e0	135
\XINT_smallldiv_d	100, 101	\XINT_sqrt_big_e0_a	135
\XINT_smallldiv_e	100	\XINT_sqrt_big_f	135, 136
\XINT_smallldiv_end	100	\XINT_sqrt_big_fa	136
\XINT_smallldiv_f	100	\XINT_sqrt_big_ga	136
\XINT_smallldiv_fz	100	\XINT_sqrt_big_gb	136
\XINT_smallldiv_g	100	\XINT_sqrt_big_gc	136
\XINT_smallldiv_h	100	\XINT_sqrt_big_gd	136, 137
\XINT_smallldiv_i	100	\XINT_sqrt_big_ge	137
\XINT_smallldiv_j	100, 101	\XINT_sqrt_big_gf	137
\XINT_smallldiv_k	101	\XINT_sqrt_big_gg	137
\XINT_smallldivx_a	90, 99, 143	\XINT_sqrt_big_gi	137
\XINT_smallldivx_b	99	\XINT_sqrt_big_gj	137
\XINT_smallmul	84--86, 108, 111, 112, 143, 146--148	\XINT_sqrt_big_gk	137
\XINT_smallmul_a	86	\XINT_sqrt_big_gl	137
\XINT_smallmul_e	86	\XINT_sqrt_big_gloop	137, 138
\XINT_smallmul_f	86	\XINT_sqrt_big_gm	137, 138
\XINT_smallunrevbyviii	71, 73	\XINT_sqrt_big_gn	138
\XINT_split_finish	117	\XINT_sqrt_big_ka	136--138
		\XINT_sqrt_big_kb	138

\XINT_sqrt_big_kc	138	\XINT_sub_l	81, 82
\XINT_sqrt_big_ke	138, 139	\XINT_sub_l_carry	81
\XINT_sqrt_big_kend	138	\XINT_sub_l_Ia	81
\XINT_sqrt_big_kf	139	\XINT_sub_l_Ib	81, 82
\XINT_sqrt_big_kg	139	\XINT_sub_l_Ic	81
\XINT_sqrt_big_kloop	138, 139	\XINT_sub_l_Ica	81
\XINT_sqrt_big_kz	138	\XINT_sub_l_Id	81, 82
\XINT_sqrt_bigormed_f	135	\XINT_sub_l_Id_a	82
\XINT_sqrt_b0	133	\XINT_sub_l_Id_b	82
\XINT_sqrt_c	133	\XINT_sub_l_Ida	82
\XINT_sqrt_checkin	132, 133	\XINT_sub_minusminus	78
\XINT_sqrt_isneg	133	\XINT_sub_minusplus	78
\XINT_sqrt_iszero	133	\XINT_sub_mm_a	72, 78, 88
\XINT_sqrt_med_fa	135, 136	\XINT_sub_mm_b	78
\XINT_sqrt_med_fb	136	\XINT_sub_nfork	77, 78
\XINT_sqrt_med_fv	135	\XINT_sub_out	79
\XINT_sqrt_med_fvi	135	\XINT_sub_p	81, 82
\XINT_sqrt_med_fvii	135	\XINT_sub_plusminus	78
\XINT_sqrt_med_fviii	135	\XINT_sub_plusplus	78
\XINT_sqrt_post	139	\XINT_sub_secondiszero	78
\XINT_sqrt_small_a	133	\XINT_Subnormal.handler	58
\XINT_sqrt_small_d	133, 134	\XINT_sum_finished	132
\XINT_sqrt_small_e	134, 135	\XINT_sum_loop_a	132
\XINT_sqrt_small_ea	134, 136	\XINT_sum_loop_b	132
\XINT_sqrt_small_eb	134	\XINT_sum_loop_c	132
\XINT_sqrt_small_ec	134	\XINT_sumexpr	132
\XINT_sqrt_small_end	134, 135	\XINT_T	182, 184
\XINT_sqrt_small_ez	134	\xint_temp	44, 45
\XINT_sqrt_small_f	134, 135	\xint_texuniformdeviate	8, 17, 114, 149, 152
\XINT_sqrt_small_g	134	\XINT_tfrac_fork	197
\XINT_sqrt_small_h	134, 135	\XINT_tfrac_P	197
\XINT_sqrt_start	133, 254, 255	\XINT_tfrac_zero	197
\XINT_sqrtr_post	139	\XINT_thecoords_a	296
\XINT_sub_a	79, 80	\XINT_thecoords_b	296
\XINT_sub_aa	79	\XINT_thecoords_c	296
\XINT_sub_b	80	\xint_thir dofthree	114, 180, 275, 308
\XINT_sub_bi	80	\xint_thir dofthree_thenstop	114, 124, 125, 186, 187
\XINT_sub_c	80	\XINT_tmpa	9, 38, 54, 154, 309, 310, 317, 318, 326, 327, 329, 364, 365, 373--375
\XINT_sub_checklengths	78, 79	\XINT_tmpb	9, 38, 54, 326, 373--375
\XINT_sub_comp_clean	83	\XINT_tmpe	38, 54, 375
\XINT_sub_comp_finish	82	\XINT_to_forever	38, 39
\XINT_sub_comp_loop	82, 83	\XINT_to_forxever	39
\XINT_sub_d	80	\XINT_tofourhex	157
\XINT_sub_di	80, 81	\XINT_tofourhex_a	157
\XINT_sub_e	80, 81	\XINT_tofourhex_c	157
\XINT_sub_exchange	79	\XINT_tofourhex_d	157
\XINT_sub_f	80	\XINT_tofourhex_e	157
\XINT_sub_fi	80, 81	\XINT_token	36--39, 43, 44
\XINT_sub_firstiszero	78	\XINT_tokenB	36
\XINT_sub_fix_cuz	82	\XINT_toks	20, 38, 39, 41--43
\XINT_sub_fix_neg	82	\XINT_tosixteenbits	158
\XINT_sub_fix_none	81, 82	\XINT_tosixteenbits_a	158
\XINT_sub_g	80, 81	\XINT_tosixteenbits_c	158, 159
\XINT_sub_h	80	\XINT_tosixteenbits_d	159
\XINT_sub_hi	80, 81	\XINT_tosixteenbits_e	159
\XINT_sub_i	80, 81		
\XINT_sub_k	81		

\XINT_trim:f:csv_a	50	198, 199, 204, 206, 213--215, 217, 221, 226,
\XINT_trim:f:csv_finish	51	238, 309, 310, 312, 314, 317, 318, 326, 327
\XINT_trim:f:csv_loop	51, 52	\xint_UDsignsfork
\XINT_trim:f:csv_loop_trimmine	51	10, 72, 75,
\XINT_trim:f:csv_neg	50	78, 83, 122, 129, 130, 168, 218--220, 319, 320
\XINT_trim:f:csv_neg_a	51	\xint_UDwfork
\XINT_trim:f:csv_pos	50, 51	10
\XINT_trim:f:csv_pos_done	51, 52	\xint_UDXINTWfork
\XINT_trim:f:csv_toofew	51	10, 182, 183
\XINT_trim:f:csv_trimall	51	\xint_UDzerofork
\XINT_trim:f:csv_trimnone	50	10, 72,
\XINT_trim_a	30	75, 78, 81--83, 99, 122, 166, 167, 172, 173
\XINT_trim_finish	31	\xint_UDzerominusfork
\XINT_trim_loop	27, 30, 31 10, 26, 27, 29--31, 48, 50, 60, 61, 76,
\XINT_trim_loop_trimmine	31	107, 110, 117, 119, 120, 133, 150, 184, 193,
\XINT_trim_neg	30	197, 201, 207, 209, 210, 223--225, 230, 235,
\XINT_trim_neg_a	30	236, 240, 241, 243, 253, 255, 276, 277, 324
\XINT_trim_pos	30, 31	\xint_UDzerosfork
\XINT_trim_pos_done	30, 31	10, 129, 130, 168, 195, 196, 220
\XINT_trim_toofew	31	\XINT_undefined
\XINT_trim_trimall	30, 32	\xint_undefined
\XINT_trim_trimnone	30, 31	8
\XINT_trimunbr_a	31	\XINT_Underflow.handler
\XINT_trimunbr_neg	31	58
\XINT_trimunbr_neg_a	31	\XINT_uniformdeviate
\XINT_trunc	198	17, 150, 151
\XINT_trunc_a	198	\XINT_uniformdeviate_a
\XINT_trunc_B	198	17
\XINT_trunc_b	198, 200	\XINT_uniformdeviate_b
\XINT_trunc_C	198	17, 18
\XINT_trunc_CE	198	\XINT_uniformdeviate_neg
\XINT_trunc_D	198, 199	17
\XINT_trunc_E	198, 199	\XINT_uniformdeviate_sgnfork
\XINT_trunc_F	199	17
\XINT_trunc_G	198--200	\XINT_unrevbyviii
\XINT_trunc_H	199	71, 73, 79, 85
\XINT_trunc_Ha	200	\XINT_unrevbyviii_a
\XINT_trunc_Haa	200	71
\XINT_trunc_Hb	200	\XINT_unsep_clean
\XINT_trunc_sp_B	198	66, 67, 161
\XINT_trunc_sp_b	198, 200	\XINT_unsep_cuzsmall
\XINT_trunc_sp_C	198	67, 91, 99, 144
\XINT_trunc_sp_Ca	198	\XINT_unsep_cuzsmall_x
\XINT_trunc_sp_Cb	198	67
\XINT_trunc_sp_Cc	198, 199	\XINT_unsep_loop
\XINT_trunc_sp_Cd	199	66--68
\XINT_trunc_sp_D	198, 199	\XINT_useiimessage
\XINT_trunc_sp_E	199	112
\XINT_trunc_sp_F	199	\XINT_verysmallmul
\XINT_trunc_zero	198, 200	84, 86
\XINT_TypesetBezoutAlgorithm	176	\XINT_verysmallmul_a
\XINT_TypesetEuclideanAlgorithm	175	86
\xint_UDonezerofork	10	\XINT_verysmallmul_bi
\xint_UDsignfork	10, 13, 14, 27, 30, 31, 48, 49,	86
	51, 52, 61--65, 87, 102--105, 115--118, 155,	\XINT_verysmallmul_bj
	158, 159, 161--164, 168, 191, 192, 194, 196,	86
		\XINT_verysmallmul_cj
		86
		\XINT_verysmallmul_e
		86
		\XINT_verysmallmul_f
		86
		\XINT_W
		10, 182--184
		\XINT_x
		39--43
		\XINT_xflet
		36, 37, 39
		\XINT_xflet_eq?
		36
		\XINT_xflet_macro
		36
		\XINT_xflet_sp?
		36
		\XINT_xflet_spB?
		36
		\XINT_xflet_zapsp
		36
		\XINT_xflet_zapspB
		36
		\XINT_XINT_rdg
		149
		\XINT_xorof:_a
		348
		\XINT_xorof:_b
		348
		\XINT_xorof:_c
		348
		\XINT_xorof:_e
		348
		\XINT_xorof_a
		128, 129
		\XINT_xorof_b
		129
		\XINT_xorof_c
		129
		\XINT_xorof_e
		129
		\XINT_xrandomdigits
		149
		\XINT_xrandomdigits_a
		149
		\XINT_xtrunc_a
		201

<code>\XINT_xtrunc_b</code>	201	<code>\XINT_zapbsp_a</code>	22, 23
<code>\XINT_xtrunc_BisEight</code>	202	<code>\XINT_zapbsp_b</code>	22
<code>\XINT_xtrunc_BisFive</code>	202	<code>\XINT_zapbsp_e</code>	22
<code>\XINT_xtrunc_BisFour</code>	202	<code>\XINT_zapbsp_end</code>	22
<code>\XINT_xtrunc_BisOne</code>	201, 202	<code>\XINT_zapbsp_end?</code>	22
<code>\XINT_xtrunc_BisSmall</code>	202	<code>\XINT_zapbsp_a</code>	23
<code>\XINT_xtrunc_BisTwo</code>	201, 202	<code>\XINT_zapbsp_again</code>	23
<code>\XINT_xtrunc_c</code>	201	<code>\XINT_zapbsp_again?</code>	23
<code>\XINT_xtrunc_d</code>	201	<code>\XINT_zapbsp_b</code>	23
<code>\XINT_xtrunc_e</code>	202, 203	<code>\XINT_zapbsp_c</code>	23
<code>\XINT_xtrunc_finish</code>	205	<code>\xint_zapspaces</code>	11, 296, 297, 308, 328
<code>\XINT_xtrunc_finish_a</code>	206	<code>\xint_zapspaces_o</code>	11, 331, 332, 364, 365
<code>\XINT_xtrunc_I</code>	203	<code>\XINT_zapspb_bracedorone</code>	23
<code>\XINT_xtrunc_I_a</code>	203	<code>\XINT_zapspb_one?</code>	23
<code>\XINT_xtrunc_I_b</code>	203, 204	<code>\XINT_zapspb_onlyspaces</code>	23
<code>\XINT_xtrunc_IA_c</code>	204	<code>\XINT_zeroes_foriv</code>	155, 156, 158, 160, 162
<code>\XINT_xtrunc_IA_d</code>	204, 205	<code>\XINT_zeroes_foriv_done</code>	155
<code>\XINT_xtrunc_IA_xd</code>	204	<code>\XINT_zeroes_foriv_end</code>	155
<code>\XINT_xtrunc_IA_xe</code>	204	<code>\XINT_zeroes_forviii</code>	68,
<code>\XINT_xtrunc_IAA_e</code>	204		72, 73, 76, 78, 83, 84, 89, 90, 105, 108, 203
<code>\XINT_xtrunc_IAB_e</code>	204	<code>\XINT_zeroes_forviii_end</code>	68
<code>\XINT_xtrunc_IB_c</code>	204	<code>\xintAbs</code>	221, 271, 351
<code>\XINT_xtrunc_IB_d</code>	204, 205	<code>\xintabs</code>	216, 221
<code>\XINT_xtrunc_II</code>	203, 205	<code>\xintAdd</code>	207, 232, 276, 339
<code>\XINT_xtrunc_II_a</code>	205	<code>\xintadd</code>	207, 209, 260, 261, 263, 286, 325, 349, 362
<code>\XINT_xtrunc_II_b</code>	205	<code>\xintAND</code>	127
<code>\XINT_xtrunc_II_c</code>	205	<code>\xintand</code>	127, 128
<code>\XINT_xtrunc_loop</code>	204, 205	<code>\xintANDof</code>	128
<code>\XINT_xtrunc_loop_a</code>	205	<code>\xintandof</code>	128
<code>\XINT_xtrunc_loop_b</code>	205	<code>\xintANDof:csv</code>	348, 360
<code>\XINT_xtrunc_prepare_a</code>	201	<code>\xintApply</code>	23, 24, 32, 331
<code>\XINT_xtrunc_prepare_b</code>	201, 202	<code>\xintapply</code>	32, 46, 344, 346
<code>\XINT_xtrunc_prepare_c</code>	203	<code>\xintApply::csv</code>	368
<code>\XINT_xtrunc_prepare_d</code>	203	<code>\xintApply::csv</code>	368
<code>\XINT_xtrunc_prepare_e</code>	203	<code>\xintApplyInline</code>	37, 38, 327
<code>\XINT_xtrunc_prepare_f</code>	203	<code>\xintApplyNoExpand</code>	32
<code>\XINT_xtrunc_prepare_g</code>	203	<code>\xintapplynoexpand</code>	32
<code>\XINT_xtrunc_prepare_small</code>	201	<code>\xintApplyUnbraced</code>	32, 38, 334
<code>\XINT_xtrunc_small_a</code>	202	<code>\xintapplyunbraced</code>	32, 346, 347
<code>\XINT_xtrunc_sp_e</code>	202, 206	<code>\xintApplyUnbracedNoExpand</code>	32
<code>\XINT_xtrunc_sp_I</code>	206	<code>\xintapplyunbracednoexpand</code>	32
<code>\XINT_xtrunc_sp_I_a</code>	206	<code>\xintAssign</code>	43
<code>\XINT_xtrunc_sp_IA_b</code>	206	<code>\xintAssignArray</code>	44, 45, 175, 176, 331
<code>\XINT_xtrunc_sp_IA_c</code>	206, 207	<code>\xintbareeval</code>	
<code>\XINT_xtrunc_sp_IAA</code>	206		295, 296, 332, 339, 340, 342, 344, 345, 366
<code>\XINT_xtrunc_sp_IAB</code>	206, 207	<code>\xintbarefloateval</code>	296,
<code>\XINT_xtrunc_sp_IB_b</code>	206, 207		297, 332, 339, 340, 342, 344, 345, 366, 372
<code>\XINT_xtrunc_sp_IB_c</code>	207	<code>\xintbareiieval</code>	
<code>\XINT_xtrunc_sp_II</code>	206, 207		295, 296, 332, 339, 340, 342, 344, 345, 366
<code>\XINT_xtrunc_transition</code>	205	<code>\xintBezout</code>	168
<code>\XINT_xtrunc_zero</code>	201	<code>\xintbezout</code>	168
<code>\XINT_y</code>	39, 41--43	<code>\xintBezoutAlgorithm</code>	173, 176
<code>\XINT_Z</code>	183	<code>\xintbezoutalgorithm</code>	173
<code>\XINT_zapbsp_a</code>	21, 22	<code>\xintBinomial</code>	211, 356
<code>\XINT_zapbsp_again</code>	22	<code>\xintbinomial</code>	211
<code>\XINT_zapbsp_again?</code>	22	<code>\xintBinToDec</code>	161
<code>\XINT_zapbsp_b</code>	22	<code>\xintbintodec</code>	161

\xintBinToHex	162	\xintDecSplitL	118
\xintbintoohex	162	\xintdecsplitl	118
\xintBool	148, 329	\xintDecSplitR	118
\xintboolexpr	297	\xintdecsplitr	118
\xintbracediloopindex	35	\xintDecToBin	158
\xintbracedouteriloopindex	36	\xintdectobin	158
\xintBreakFor	38, 39	\xintDecToHex	155
\xintBreakForAndDo	38, 40	\xintdectoohex	155
\xintbreakiloop	35	\xintDecToString	189
\xintbreakiloopanddo	35	\xintdectostring	189
\xintbreakloop	35	\xintdeffloatfunc	364
\xintbreakloopanddo	35	\xintdeffloatfunc_a	364, 365
\xintCeil	190, 352	\xintdeffloatvar	332
\xintceil	190	\xintdeffloatvar_a	332
\xintCFrac	269	\xintdeffunc	364
\xintcfrac	269	\xintdeffunc_a	364
\xintCHexToBin	163	\xintdefiifunc	364
\xintchextobin	163, 164	\xintdefiifunc_a	364, 365
\xintCmp	186, 187, 219	\xintdefiivar	332
\xintcmp	219	\xintdefiivar_a	332
\xintCnToCs	287	\xintdefvar	332
\xintcntocs	287	\xintdefvar_a	332
\xintCnToF	285	\xintDenominator	190
\xintcntof	285	\xintdenominator	190
\xintCnToGC	288	\XINTdigits	207,
\xintcntogc	288		223, 230--235, 239, 240, 243, 248, 252, 253,
\xintCstoCv	281		255--257, 266, 267, 324, 325, 350, 363, 372
\xintcstocv	281	\xintDigits	207
\xintCstoF	277	\xintDigitsOf	45
\xintcstof	277	\xintdimensions	40
\xintCstoGC	289	\xintDiv	212, 213, 252, 286
\xintcstogc	289	\xintdiv	212, 276, 277
\xintCSV::csv	295, 298, 372	\xintDivFloor	213
\xintCSVFirstItem	54	\xintdivfloor	213, 256
\xintCSVKeep	54	\xintDivMod	214, 355
\xintCSVLastItem	54	\xintdivmod	214, 256
\xintCSVLength	54, 331	\xintDivRound	213
\xintCSVNthEltPy	54	\xintdivround	213
\xintCSVReverse	54	\xintDivTrunc	213
\xintCSVtoList	23, 331	\xintdivtrunc	213
\xintcsvtolist	23, 38, 39, 42, 43, 277, 281	\xintdothis	10
\xintCSVtoListNoExpand	24	\xintDouble	62, 136, 229, 239
\xintcsvtolistnoexpand	24	\xintdouble	62, 202
\xintCSVtoListNonStripped	24, 344, 346, 365, 366	\xintDSH	119, 229
\xintcsvtolistnonstripped	23, 24, 365, 366	\xintdsh	119
\xintCSVtoListNonStrippedNoExpand	24	\xintDSHr	119
\xintcsvtolistnonstrippednoexpand	24	\xintdshr	119
\xintCSVTrim	54	\xintDSL	64
\xintCtoCv	281	\xintdsl	64
\xintctocv	281	\xintDSR	64
\xintCtoF	277	\xintdsr	64
\xintctof	277	\xintDSRr	65
\xintCTRLC	58	\xintdsrr	65, 239
\xintDec	64	\xintDSx	120
\xintdec	64	\xintdsx	120
\xintDecSplit	117	\xintE	193, 328
\xintdecsplit	117, 200	\xinte	193

\xintegers	40	\xintFloatSqrt	253
\xintEq	186	\xintfloatsqr	253
\xinteq	186	\xintFloatSub	233
\xintError:ignored	294	\xintfloatsub	233
\xintError:inserted	310	\xintFloor	190, 352
\xintError:missing_xintthe!	295	\xintfloor	190
\xintError:removed	309, 329, 330	\xintFor	38, 309--311, 316--318, 326, 329, 364
\xintError:replaced_by_zero	333	\xintForfour	42
\xintError:thiscannothappen	107	\xintForpair	42
\xintError:we_are_doomed	336	\xintForthree	42
\xintEuclideanAlgorithm	171, 175	\xintFrac	191, 271
\xinteuclideanAlgorithm	171	\xintfrac	191
\xinteval	295--297	\xintFtoC	275
\xintEven	186, 353	\xintftoc	275
\xinteven	186	\xintFtoCC	276, 285
\xintExpandArgs	46	\xintftocc	276
\xintexpr	295	\xintFtoCCv	285
\xintexprRestoreCatcodes	331, 364, 374	\xintftoccv	285
\xintexprSafeCatcodes	332, 364, 373, 374	\xintFtoCs	273, 285
\xintexprsafecatcodesfalse	374	\xintftocs	273
\xintexprsafecatcodestrue	374	\xintFtoCv	285
\xintFac	211, 328, 354	\xintftocv	285
\xintfac	211	\xintFtoCx	274
\xintfdef	11, 20	\xintftocx	274, 275
\xintFDg	61	\xintFtoGC	275
\xintfdg	61	\xintftogc	275
\xintFGtoC	275	\xintFwOver	192
\xintfgtoc	275	\xintfwover	192
\xintFirstItem:f:csv	53, 54, 360	\xintFxFtPowerSeries	264
\xintfirstitem:f:csv	53	\xintfxptpowerseries	264
\xintFloat	222	\xintFxFtPowerSeriesX	265
\xintfloat	222, 230, 232--235, 239, 242, 247, 251, 253, 266	\xintfxptpowerseriesx	265
\xintFloatAdd	232	\xintGCD	166
\xintfloatadd	232, 267	\xintgcd	166, 175, 349
\xintFloatBinomial	251	\xintGCdof	175
\xintfloatbinomial	251	\xintgcdof	175
\xintFloatDiv	234	\xintGCdof:csv	349, 358
\xintfloatdiv	234	\xintGCFrac	270
\xintFloatE	255	\xintgcfrac	270
\xintfloate	255	\xintGCntoF	286
\xintfloateval	295, 297	\xintgcntof	286
\xintfloatexpr	295	\xintGCntoGC	288
\xintFloatFac	242	\xintgcntogc	288
\xintfloatfac	242	\xintGCtoCv	282
\xintFloatMul	233	\xintgctocv	282
\xintfloatmul	233	\xintGCtoF	279
\xintFloatPFactorial	247	\xintgctof	279
\xintfloatpfactorial	247	\xintGCtoGC	289
\xintFloatPow	235	\xintgctogc	289
\xintfloatpow	235	\xintGCtoGCx	273
\xintFloatPower	239	\xintgctogcx	273
\xintfloatpower	239	\xintGeq	216
\xintFloatPowerSeries	265	\xintgeq	216
\xintfloatpowerseries	265	\xintgfdef	20
\xintFloatPowerSeriesX	267	\xintGGCFrac	272
\xintfloatpowerseriesx	267	\xintggcfrac	272
		\xintgobble	15

<code>\xintgodef</code>	20	<code>\xintigctocv</code>	284, 285
<code>\xintgoodef</code>	20	<code>\xintiGctoF</code>	280
<code>\xintGt</code>	186	<code>\xintigctof</code>	280
<code>\xintgt</code>	186	<code>\xintiiAbs</code>	61, 351
<code>\xintGtorEq</code>	186	<code>\xintiiabs</code>	61, 105, 166, 167, 171--173
<code>\xintHalf</code>	63	<code>\xintiiAdd</code>	72, 339
<code>\xinthalf</code>	63, 240	<code>\xintiiadd</code>	
<code>\xintHexToBin</code>	163		41, 72, 136, 137, 139, 150, 170, 171, 173, 196, 208, 261, 265, 278--284, 325, 349, 363
<code>\xinthextobin</code>	163	<code>\xintiiBinomial</code>	139, 356
<code>\xintHexToDec</code>	159, 294	<code>\xintiiBinomial</code>	139
<code>\xinthextodec</code>	159	<code>\xintiiCmp</code>	75, 125, 126, 216, 294
<code>\xintiCeil</code>	190, 311, 322, 352	<code>\xintiiicmp</code>	75
<code>\xinticeil</code>	190	<code>\xintiiDivFloor</code>	105
<code>\xintiCstoCv</code>	282	<code>\xintiidivfloor</code>	105
<code>\xinticstocv</code>	282, 285	<code>\xintiiDivision</code>	87
<code>\xintiCstoF</code>	278	<code>\xintiidivision</code>	
<code>\xinticstof</code>	278		87, 102, 105, 137, 138, 215, 270, 273--275
<code>\xintieval</code>	295, 296	<code>\xintiiDivMod</code>	104, 355
<code>\xintiexpr</code>	295	<code>\xintiidivmod</code>	104, 105
<code>\xintifboolexpr</code>	297	<code>\xintiiDivRound</code>	102, 254
<code>\xintifboolfloatexpr</code>	297	<code>\xintiidivround</code>	102
<code>\xintifboolliexpr</code>	297	<code>\xintiiDivTrunc</code>	103
<code>\xintifCmp</code>	186, 324, 325, 362--364	<code>\xintiidivtrunc</code>	103
<code>\xintifcmp</code>	186	<code>\xintiiE</code>	116, 216, 328
<code>\xintifEq</code>	40, 187, 275	<code>\xintiiE</code>	116, 189
<code>\xintifeq</code>	186, 187	<code>\xintiiEq</code>	121
<code>\xintifFalseAelseB</code>	127	<code>\xintiiEval</code>	295, 296
<code>\xintifFloatInt</code>	257, 361, 370	<code>\xintiiEven</code>	124, 354
<code>\xintiffloatint</code>	257	<code>\xintiiEven</code>	124
<code>\xintifFloatInt:</code>	361, 371	<code>\xintiiexpr</code>	295
<code>\xintifFloatIntNE:</code>	370, 371	<code>\xintiiFac</code>	110, 328, 354
<code>\xintifForFirst</code>	38, 39, 41--43	<code>\xintiiifac</code>	110
<code>\xintifForLast</code>	38, 39, 41--43	<code>\xintiiGCD</code>	166
<code>\xintifGt</code>	187	<code>\xintiigcd</code>	166, 350
<code>\xintifgt</code>	186, 187	<code>\xintiiGCDof:csv</code>	350, 358
<code>\xintifInt</code>	195, 361, 370	<code>\xintiiGeq</code>	122
<code>\xintifint</code>	195	<code>\xintiiGeq</code>	122
<code>\xintifInt:</code>	361, 371	<code>\xintiiGt</code>	122
<code>\xintifIntNE:</code>	370, 371	<code>\xintiiGtorEq</code>	123
<code>\xintiFloor</code>	190, 322, 352	<code>\xintiiifCmp</code>	125
<code>\xintifloor</code>	190, 213	<code>\xintiiifcmp</code>	125
<code>\xintifLt</code>	187	<code>\xintiiifEq</code>	125
<code>\xintiflt</code>	186, 187	<code>\xintiiifeq</code>	121, 125
<code>\xintifNotZero</code>	187	<code>\xintiiifGt</code>	125
<code>\xintifnotzero</code>	187, 297	<code>\xintiiifgt</code>	122, 123, 125
<code>\xintifOdd</code>	188	<code>\xintiiifLt</code>	126, 139, 151
<code>\xintifodd</code>	188	<code>\xintiiiflt</code>	122, 123, 126, 229
<code>\xintifOne</code>	187, 361, 370	<code>\xintiiifNotZero</code>	126, 327, 348, 361, 370
<code>\xintifone</code>	187	<code>\xintiiifnotzero</code>	126, 127
<code>\xintifOne:</code>	361, 371	<code>\xintiiifNotZero:</code>	361, 371
<code>\xintifOneNE:</code>	370, 371	<code>\xintiiifNotZeroNE:</code>	370, 371
<code>\xintifSgn</code>	186	<code>\xintiiifOdd</code>	127
<code>\xintifsgn</code>	186	<code>\xintiiifodd</code>	127
<code>\xintifTrueAelseB</code>	127--129	<code>\xintiiifOne</code>	126, 361, 370
<code>\xintifZero</code>	187	<code>\xintiiifone</code>	126
<code>\xintifzero</code>	187	<code>\xintiiifOne:</code>	361, 371
<code>\xintiGctoCv</code>	284		

\xintiifOneNE:	370, 371	\xintiiRandRangeAtoB	150, 357, 372
\xintiifSgn	125, 328, 361, 371	\xintiirandrangeAtoB	150
\xintiifsgn	125	\xintiiRem	102, 195, 358
\xintiifSgn:	361, 371	\xintiirem	102, 197, 214, 216
\xintiifSgnNE:	370, 371	\xintiiSeq::csv	323, 371
\xintiifZero	126, 189, 275	\xintiiseq::csv	323
\xintiifzero	126, 127	\xintiiSeqA::csv	324
\xintiiIsNotZero	123, 353	\xintiiSeqB::csv	325, 371
\xintiiisnotzero	123, 128	\xintiiSeqB:f:csv	363
\xintiiIsOne	123, 127	\xintiiSeqBNumeric::csv	371
\xintiiisone	123	\xintiiSeqNumeric::csv	371
\xintiiIsZero	123, 128, 353	\xintiiSgn	60, 123, 125, 126, 128, 352
\xintiiiszero	123, 127	\xintiiSgn	60
\xintiiLCM	167	\xintiiSqr	105, 136, 137, 139, 353
\xintiilcm	167, 350	\xintiiSqr	105, 210, 237, 238, 241, 242
\xintiiLCMof:csv	350, 358	\xintiiSqrt	139, 354
\xintiilt	122	\xintiiSqrt	139
\xintiiltorEq	123	\xintiiSqrtR	139, 354
\xintiiMax	129	\xintiiSqrtr	139
\xintiimax	129, 131, 349	\xintiiSquareRoot	132
\xintiiMaxof	131	\xintiiSquareroot	132, 139
\xintiimaxof	131	\xintiiSub	77, 136
\xintiiMaxof:csv	349, 358	\xintiiSub	77, 136--139, 150, 254
\xintiiMin	130	\xintiiSum	132
\xintiimin	130, 131, 349	\xintiiSum	132
\xintiiMinof	131	\xintiiSum:csv	349, 359
\xintiiminof	131	\xintiLen	115
\xintiiMinof:csv	349, 359	\xintilen	115
\xintiiMMON	124	\xintiloop	35, 44
\xintiimmon	124	\xintiloop_a	35
\xintiiMod	105, 356	\xintiloop_again	35, 36
\xintiimod	105	\xintiloop_again_b	35, 36
\xintiiModTrunc	103	\xintiloopindex	35, 44
\xintiimodtrunc	103	\xintiloopskipandredo	36
\xintiiMON	124	\xintiloopskiptonext	36
\xintiimon	124	\xintInc	63, 229
\xintiiMul	83, 136, 173, 208, 216, 229, 234, 294, 339	\xintinc	63, 88, 138, 229
\xintiimul	40, 41, 83, 167, 202, 208, 210, 212, 216, 217, 220, 238, 242, 349	\XINTinFloat	224, 256, 257, 350, 355
\xintiiNotEq	121	\XINTinfloat ..	223, 224, 231--233, 236, 241, 242, 248, 251--253, 255, 256, 299, 324, 325, 363
\xintiiOdd	123, 127, 241, 353	\XINTinFloat::csv	297, 299
\xintiiodd	123	\XINTinFloatAdd	232, 339
\xintiiOpp	61, 327	\XINTinfloatadd	232, 266, 325, 350, 363, 364
\xintiopp	61, 63, 65, 102--105, 169, 190, 197, 213--215	\XINTinFloatBinomial	251, 356
\xintiiPFactorial	145, 357	\XINTinfloatbinomial	251
\xintiipfactorial	145	\XINTinFloatdigits	330, 350, 355
\xintiiPow	58, 106, 294	\XINTinFloatDiv	234
\xintiipow	106, 211	\XINTinfloatdiv	234
\xintiiPrd	132	\XINTinFloatDivFloor	256
\xintiiprd	132	\XINTinfloatdivfloor	256
\xintiiPrd:csv	349, 360	\XINTinFloatDivMod	256, 355
\xintiiQuo	102, 167, 208, 216, 358	\XINTinfloatdivmod	256
\xintiiquo	102, 190, 195, 208, 216, 227, 276	\XINTinFloatE	256, 328
\xintiiRandRange	150, 357, 372	\XINTinfloate	256
\xintiirandrange	150	\XINTinFloatFac	242, 328, 350, 354
		\XINTinfloatfac	242
		\XINTinFloatFacdigits	350, 354

\XINTinFloatFracdigits	231, 352	\xintiszero	186
\XINTinfloatfracdigits	231	\xintiTrunc	198, 265, 355
\XINTinFloatMaxof:csv	350, 359	\xintitrunc	198, 200, 239, 264
\XINTinFloatMinof:csv	350, 359	\xintJrr	196
\XINTinFloatMod	256, 356	\xintjrr	196
\XINTinfloatmod	256	\xintKeep	27
\XINTinFloatMul	233, 339, 353	\xintkeep	27
\XINTinfloatmul	233, 266, 267, 350	\xintKeep:f:csv	48, 54, 319, 320
\XINTinFloatPFactorial	247, 357	\xintkeep:f:csv	48
\XINTinfloatpfactorial	247, 248	\xintKeep:x:csv	319, 321
\XINTinFloatPow	235	\xintKeepNoExpand	27
\XINTinfloatpow	235, 266, 267	\xintkeepnoexpand	27
\XINTinFloatPower	239	\xintKeepUnbraced	28
\XINTinfloatpower	239	\xintkeepunbraced	28
\XINTinFloatPowerH	239	\xintKeepUnbracedNoExpand	28
\XINTinfloatpowerh	239	\xintkeepunbracednoexpand	28, 29
\XINTinFloatPrd:csv	350, 359	\xintLastItem	12
\XINTinFloatS	224	\xintlastitem	12
\XINTinfloatS	224, 232--235, 239, 240	\xintLastItem:f:csv	54, 360
\XINTinFloatSeq::csv	371	\xintlastitem:f:csv	54
\XINTinFloatSeqA::csv	324	\xintLCM	167
\XINTinFloatSeqB::csv	325, 371	\xintlcm	167, 175, 349
\XINTinFloatSeqB:f:csv	363	\xintLCMof	175
\XINTinFloatSeqBNumeric::csv	371	\xintlcmof	175
\XINTinFloatSqr	353	\xintLCMof:csv	349, 358
\XINTinFloatSqrt	240, 253, 350, 354	\xintLDg	62, 89, 123, 124, 186, 239, 241, 242
\XINTinfloatsqrt	253	\xintldg	62
\XINTinFloatSqrtdigits	350, 354	\xintLen	115, 180
\XINTinFloatSub	233	\xintlen	115, 180
\XINTinfloatsub	233	\xintLength	12, 204--207, 217, 220, 225--227, 236, 237, 240, 294, 332
\XINTinFloatSum:csv	350, 359	\xintlength	12, 26, 133, 150, 199
\XINTinRandomFloatS	257	\xintLength:f:csv	47, 54, 319--321, 360
\XINTinrandomfloatS	257	\xintlength:f:csv	47
\XINTinRandomFloatSdigits	257, 330, 371	\xintLengthUpTo	13
\XINTinRandomFloatSixteen	258, 330, 371	\xintlengthupto	13
\xintintegers	40	\xintLengthUpTo:f:csv	47
\xintiNum	59, 137, 138, 330	\xintlengthupto:f:csv	47
\xintinum	59, 115, 151, 254	\xintListSel:f:csv	317, 320
\xintipfactorial	211	\xintListSel:x:csv	317, 319, 371
\xintipow	210	\xintListWithSep	25
\xintiQuo	357	\xintlistwithsep	25
\xintiRem	358	\xintListWithSepNoExpand	25
\xintiRound	200, 354, 355	\xintlistwithsepnoexpand	25
\xintiround	200, 213, 299	\xintloop	35, 176, 331
\xintIrr	194, 351	\xintloop_again	35
\xintirr	194	\xintloopskiptonext	35
\xintiSeries	260	\xintLt	186
\xintiseries	260	\xintlt	186
\xintIsFalse	127	\xintLtorEq	186
\xintIsNotZero	127, 186	\xintMax	217
\xintisnotzero	186, 299	\xintmax	217, 218, 349, 350
\xintIsOne	187, 216	\xintMaxof	218
\xintisone	216	\xintmaxof	218
\xintIsTrue	127	\xintMaxof:csv	349, 358
\xintIsTrue::csv	295, 298, 372	\xintMessage	18, 331, 332, 334, 364, 366, 374
\xintIsZero	127, 186	\xintMin	218

<code>\xintmin</code>	218, 219, 349, 350	<code>\xintpowerseriesx</code>	262
<code>\xintMinof</code>	219	<code>\xintPRaw</code>	188
<code>\xintminof</code>	219	<code>\xintpraw</code>	188, 298
<code>\xintMinof:csv</code>	349, 359	<code>\xintPrd</code>	212
<code>\xintMod</code>	215, 256, 356	<code>\xintprd</code>	212
<code>\xintmod</code>	215	<code>\xintPrd:csv</code>	349, 359
<code>\xintModTrunc</code>	213	<code>\xintRandomDigits</code>	148, 151, 152
<code>\xintmodtrunc</code>	213	<code>\xintrandomdigits</code>	148, 151, 152, 257
<code>\xintMul</code>	209, 263--265, 339	<code>\xintrationals</code>	40
<code>\xintmul</code>	209, 212, 261--263, 349	<code>\xintRationalSeries</code>	262
<code>\xintNewBoolExpr</code>	372	<code>\xintRationalSeriesX</code>	263
<code>\xintnewdummy</code>	332, 334	<code>\xintratseries</code>	262
<code>\xintNewExpr</code>	317, 372	<code>\xintratseriesx</code>	263
<code>\xintNewFloatExpr</code>	372	<code>\xintRaw</code>	188, 330
<code>\xintNewFunction</code>	365	<code>\xintraw</code>	188, 189, 194, 201, 207, 209, 210, 213--215, 217--221, 324, 362
<code>\xintNewIExpr</code>	372	<code>\xintRawWithZeros</code>	189
<code>\xintNewIIExpr</code>	372	<code>\xintrawwithzeros</code>	40, 189, 190, 194--197, 216, 269, 270, 273--283, 285
<code>\xintNOT</code>	127	<code>\xintRelaxArray</code>	44, 332
<code>\xintNotEq</code>	186	<code>\xintreplicate</code>	14, 120, 137, 200, 201, 205, 206
<code>\xintNthElt</code>	26	<code>\xintRev</code>	116
<code>\xintnthelt</code>	26	<code>\xintReverse:f:csv</code>	53, 54, 361
<code>\xintNthEltNoExpand</code>	26	<code>\xintreverse:f:csv</code>	53
<code>\xintntheltnoexpand</code>	26, 335, 336	<code>\xintReverseDigits</code>	115, 116
<code>\xintNthEltPy:f:csv</code>	52, 54, 319	<code>\xintreversedigits</code>	115
<code>\xintntheltpy:f:csv</code>	52	<code>\xintReverseOrder</code>	11, 346, 347
<code>\xintNum</code>	59, 166, 167, 171--173, 186, 210--212, 243, 248, 252, 318, 319, 335, 350, 351, 357, 369	<code>\xintreverseorder</code>	11
<code>\xintnum</code>	59, 61, 72, 75, 78, 102, 122, 168, 200, 210, 240	<code>\xintRevWithBraces</code>	20, 344, 346
<code>\xintNumerator</code>	190	<code>\xintrevwithbraces</code>	20, 21
<code>\xintnumerator</code>	190	<code>\xintRevWithBracesNoExpand</code>	21
<code>\xintOdd</code>	186, 188, 353	<code>\xintrevwithbracesnoexpand</code>	21
<code>\xintodd</code>	186	<code>\xintREZ</code>	193
<code>\xintodef</code>	11, 20	<code>\xintrez</code>	193, 257
<code>\xintoodef</code>	11, 20	<code>\xintREZ?</code>	190
<code>\xintOpp</code>	190, 221, 233, 327	<code>\xintRound</code>	200, 354
<code>\xintopp</code>	209, 221	<code>\xinround</code>	200, 299
<code>\xintOR</code>	128	<code>\xintRound:csv</code>	296, 299, 372
<code>\xintor</code>	128	<code>\xintSeq</code>	33
<code>\xintORof</code>	128	<code>\xintseq</code>	33
<code>\xintorof</code>	128	<code>\xintSeq::csv</code>	322, 371
<code>\xintORof:csv</code>	348, 360	<code>\xintseq::csv</code>	322
<code>\xintorthat</code>	10	<code>\xintSeqA::csv</code>	324
<code>\xintouteriloopindex</code>	36	<code>\xintSeqB::csv</code>	324, 371
<code>\xintPFactorial</code>	145, 212, 356	<code>\xintSeqB:f:csv</code>	362
<code>\xintpfactorial</code>	145, 212	<code>\xintSeqBNumeric::csv</code>	371
<code>\xintPFloat</code>	230	<code>\xintSeqNumeric::csv</code>	371
<code>\xintpfloat</code>	230	<code>\xintSeries</code>	260
<code>\xintPFloat::csv</code>	297, 299, 372	<code>\xintseries</code>	260
<code>\xintPIrr</code>	194, 351	<code>\XINTsetupcatcodes</code>	7, 20, 56, 114, 154, 166, 179, 260, 269, 294
<code>\xintpirr</code>	194	<code>\xintSgn</code>	186, 187, 221, 271, 352
<code>\xintPow</code>	210, 262, 264, 265	<code>\xintsgn</code>	221
<code>\xintpow</code>	210	<code>\xintSgnFork</code>	124, 325, 363
<code>\xintPowerSeries</code>	261	<code>\xintsgnfork</code>	124
<code>\xintpowerseries</code>	261	<code>\xintSignedFrac</code>	191
<code>\xintPowerSeriesX</code>	262	<code>\xintsignedfrac</code>	191

\xintSignedFwOver	192	\xinttrimunbracednoexpand	31
\xintsignedfwover	192	\xintTrunc	197, 355
\xintSPRaw	298	\xintttrunc	189, 197, 198, 264, 265
\xintspraw	298	\xintTTrunc	200
\xintSPRaw::csv	295, 298, 372	\xinttttrunc	200, 213
\xintSqr	210, 352	\xintTypesetBezoutAlgorithm	176
\xintsqr	210	\xintTypesetEuclideAlgorithm	175
\xintSub	209	\xintunassignvar	332
\xintsub	209, 276, 277	\xintUniformDeviate	17, 18
\xintSum	209	\xintUse	58
\xintsum	209	\xintXOR	128
\xintSum:csv	349, 359	\xintxor	128
\xintTFrac	197, 352	\xintXORof	128
\xinttfrac	197, 231	\xintxorof	128
\xintthe	295	\xintXORof:csv	348, 360
\xintthebareeval 296, 337, 340, 342, 344, 345, 373		\xintXRANDOMdigits	149, 152
\xintthebarefloateval		\xintXTTrunc	201
..... 296, 337, 340, 342, 344, 345, 373		\xintZapFirstSpaces	21, 23
\xintthebareiieval		\xintzapfirstspaces	21
..... 296, 337, 340, 342, 344, 345, 373		\xintZapLastSpaces	22
\xinttheboolexpr	297, 372	\xintzaplastspaces	22
\xintthecoords	296	\xintZapSpaces	23
\xinttheDigits	207, 297	\xintzapspace	23
\xinttheexpr	295, 297, 372	\xintZapSpacesB	23, 40
\xintthefloatexpr	295, 297, 372	\xintzapspacebs	23, 24
\xinttheiexpr	295, 372		
\xinttheiiexpr	295, 297, 372		
\xintToggle	148, 329		
\xintTrim	30		
\xinttrim	30, 345--347		
\xintTrim:f:csv	50, 54, 319, 320		
\xinttrim:f:csv	50		
\xintTrimNoExpand	30		
\xinttrimnoexpand	30		
\xintTrimUnbraced	31		
\xinttrimunbraced	31		
\xintTrimUnbracedNoExpand	31		
\xinttrimunbracednoexpand	31		

Z

14 Cumulative line count

xintkernel: 554. Total number of code lines: 14394. (but 3325 lines among them
xinttools:1454. start either with {% or with }%.)
xintcore:2182. Each package starts with circa 50 lines dealing with cat-
xint:1496. codes, package identification and reloading management,
xintbinhex: 472. also for Plain TeX. Version 1.3c of 2018/06/17.
xintgcd: 434.
xintfrac:3220.
xintseries: 386.
xintcfrac:1029.
xintexpr:3167.