

The `bitset` package

Heiko Oberdiek*

2016/05/16 v1.2

Abstract

This package defines and implements the data type bit set, a vector of bits. The size of the vector may grow dynamically. Individual bits can be manipulated.

Contents

| | | |
|----------|---|----------|
| 1 | Documentation | 3 |
| 1.1 | Introduction | 3 |
| 1.2 | Glossary | 3 |
| 1.3 | Design principles | 4 |
| 1.4 | Operator overview | 5 |
| 1.5 | Package loading | 5 |
| 1.6 | Operators | 5 |
| 1.6.1 | Miscellaneous | 6 |
| 1.6.2 | Import | 6 |
| 1.6.3 | Export | 7 |
| 1.6.4 | Logical operators | 7 |
| 1.6.5 | Shifting | 7 |
| 1.6.6 | Bit manipulation | 8 |
| 1.6.7 | Bit retrieval | 8 |
| 1.6.8 | Bit set properties | 9 |
| 1.6.9 | Queries | 9 |
| 2 | Implementation | 9 |
| 2.1 | Reload check and package identification | 10 |
| 2.2 | Catcodes | 11 |
| 2.3 | Package loading | 12 |
| 2.4 | Help macros | 12 |
| 2.4.1 | Number constant | 12 |
| 2.4.2 | General basic macros | 12 |
| 2.4.3 | Tail recursion | 13 |
| 2.4.4 | Check macros | 13 |
| 2.5 | Miscellaneous | 14 |
| 2.6 | Import | 14 |
| 2.6.1 | From binary number | 14 |
| 2.6.2 | From octal/hex number | 15 |
| 2.6.3 | From decimal number | 17 |
| 2.7 | Export | 18 |

*Please report any issues at <https://github.com/ho-tex/oberdiek/issues>

| | | |
|-------------------|--|-----------|
| 2.7.1 | To binary number | 18 |
| 2.7.2 | To octal/hexadecimal number | 19 |
| 2.7.3 | To decimal number | 22 |
| 2.8 | Logical operators | 24 |
| 2.8.1 | \bitsetAnd | 24 |
| 2.8.2 | \bitsetAndNot | 25 |
| 2.8.3 | \bitsetOr | 26 |
| 2.8.4 | \bitsetXor | 27 |
| 2.8.5 | Shifting | 28 |
| 2.8.6 | \bitsetShiftLeft | 28 |
| 2.8.7 | \bitsetShiftRight | 28 |
| 2.9 | Bit manipulation | 29 |
| 2.9.1 | Clear operation | 30 |
| 2.9.2 | Set operation | 31 |
| 2.9.3 | Flip operation | 31 |
| 2.9.4 | Range operators | 32 |
| 2.10 | Bit retrieval | 34 |
| 2.10.1 | \bitsetGet | 34 |
| 2.10.2 | \bitsetNextClearBit, \bitsetNextSetBit | 35 |
| 2.10.3 | \bitsetGetSetBitList | 37 |
| 2.11 | Bit set properties | 38 |
| 2.12 | Queries | 39 |
| 3 | Installation | 41 |
| 3.1 | Download | 41 |
| 3.2 | Bundle installation | 41 |
| 3.3 | Package installation | 41 |
| 3.4 | Refresh file name databases | 41 |
| 3.5 | Some details for the interested | 41 |
| 4 | History | 42 |
| [2007/09/28 v1.0] | | 42 |
| [2011/01/30 v1.1] | | 42 |
| [2016/05/16 v1.2] | | 42 |
| 5 | Index | 42 |

1 Documentation

1.1 Introduction

Annotations in the PDF format know entries whose values are integers. This numbers are interpreted as set of flags specifying properties. For example, annotation dictionaries can have a key `/F`. The bits of its integer value are interpreted the following way:

| Bit position | Property name |
|--------------|---------------|
| 1 | Invisible |
| 2 | Hidden |
| 3 | Print |
| 4 | NoZoom |
| 5 | NoRotate |
| 6 | NoView |
| 7 | ReadOnly |
| ... | ... |

Now, let's see how these values are set in package `hyperref` before it uses this package (before v6.77a):

```
\ifFld@hidden /F 6\else /F 4\fi
```

Where are the other flags? The following example for key `/Ff` in a widget annotation supports at least three properties:

```
\ifFld@multiline
  \ifFld@readonly /Ff 4097\else /Ff 4096\fi
\else
  \ifFld@password
    \ifFld@readonly /Ff 8193\else /Ff 8192\fi
  \else
    \ifFld@readonly /Ff 1\fi
  \fi
\fi
```

But you see the point. It would be a nightmare to continue this way in supporting the missing flag settings. This kind of integers may have up to 32 bits.

Therefore I wanted a data structure for setting and clearing individual bits. Also it should provide an export as decimal number. The snippets above are executed in expansion contexts without \TeX 's stomach commands. It would be convenient to have an expandable conversion from the data structure to the integer that gets written to the PDF file.

This package `bitset` implements such a data structure. The interface is quite close to Java's class `BitSet` in order not to learn to many interfaces for the same kind of data structure.

1.2 Glossary

Bit set: A bit set is a vector of bits or flags. The vector size is unlimited and grows dynamically. An undefined bit set is treated as bit set where all bits are cleared.

Bit sets are addressed by name. A name should consists of letters or digits. Technically it must survive `\csname`, see \LaTeX 's environment names for other names with such a constraint. Package `babel`'s shorthands are not supported due to technical reasons. Shorthand support would break expandable operations.

Size: A size of a bit set is the number of bits in use. It's the number of the highest index, incremented by one. Sizes are in the range 0 up to 2147483647, the highest number supported by \TeX .

Index: Bit positions in a bit set are addressed by an index number. The bit vector is zero based. The first and least significant bit is addressed by index 0 and the highest possible bit by 2147483646.

Bit: A bit is encoded as 0 for cleared/disabled or 1 for set/enabled.

1.3 Design principles

Name conventions: To avoid conflicts with existing macro names, the operations are prefixed by the package name.

Zero based indexes: The first bit is addressed by zero. (Convention of array indexing in C, Java, ...)

Unlimited size: There is no restriction on the size of a bit set other than usual memory limitations. `\bitsetSetDec` and `\bitsetGetDec` transparently switch to package `bigintcalc` if the numbers get too large for \TeX 's number limit.

Expandability: Any operation that does not change the bit set is expandable. And all operations that extract or calculate some result do this in exact two expansion steps. For example, a macro `\Macro` wants a bit set as decimal number. But the argument must be a plain number without macros. Thus you could prefix `\bitsetGetDec` with `\number`. However this won't work for bit sets with 31 or more bits because of \TeX 's number limit of $2^{31} - 1$. then just hit the operator with two `\expandafter`:

```
\expandafter\expandafter\expandafter
\Macro\bitsetGetDec{foo}
```

`\bitsetGetDec` is hit first by the third `\expandafter` and then by the second one.

Format independence: This package is written as \LaTeX package, but it does not depend on \LaTeX . It will also work for other formats such as plain \TeX .

Independence from \TeX engines: Vanilla \TeX is all you need. Calculations are delegated to packages `intcalc` and `bigintcalc`. They don't need any special features, but they will switch to a little more efficient implementation if features such as `\numexpr` are available.

Numeric arguments: Anything that is accepted by `\number`. If $\varepsilon\text{-TeX}$ is detected, also expressions for `\numexpr` are supported. The only exception so far is the number for `\bitsetSetDec`. The number might be too large for `\number` or `\numexpr`.

Error messages: In expandable contexts, only a limited set of \TeX primitive commands work as expected. So called stomach commands behave like `\relax` and don't get expanded or executed. Unhappily also the error commands belong to this category. The expandable operations will throw an unknown control sequence instead to get \TeX 's and user's attention. The name of these control sequences starts with `\BitSetError:` with the type of error after the colon.

1.4 Operator overview

Miscellaneous (section 1.6.1)

| | |
|---------------------------|---|
| <code>\bitsetReset</code> | $\langle BitSet \rangle$ |
| <code>\bitsetLet</code> | $\langle BitSet A \rangle \langle BitSet B \rangle$ |

Import (section 1.6.2)

| | |
|--|--|
| <code>\bitsetSetBin</code> , <code>\bitsetSetOct</code> , <code>\bitsetSetHex</code> | $\langle BitSet \rangle \langle Value \rangle$ |
| <code>\bitsetSetDec</code> | $\langle BitSet \rangle \langle Value \rangle$ |

Export^a (section 1.6.3)

| | |
|--|--|
| <code>\bitsetGetBin</code> , <code>\bitsetGetOct</code> , <code>\bitsetGetHex</code> | $\langle BitSet \rangle \langle MinSize \rangle$ |
| <code>\bitsetGetDec</code> | $\langle BitSet \rangle$ |

Logical operators (section 1.6.4)

| | |
|--|---|
| <code>\bitsetAnd</code> , <code>\bitsetAndNot</code> | $\langle BitSet A \rangle \langle BitSet B \rangle$ |
| <code>\bitsetOr</code> , <code>\bitsetXor</code> | $\langle BitSet A \rangle \langle BitSet B \rangle$ |

Shifting (section 1.6.5)

| | |
|--|--|
| <code>\bitsetShiftLeft</code> , <code>\bitsetShiftRight</code> | $\langle BitSet \rangle \langle ShiftAmount \rangle$ |
|--|--|

Bit manipulation (section 1.6.6)

| | |
|---|--|
| <code>\bitsetClear</code> , <code>\bitsetSet</code> , <code>\bitsetFlip</code> | $\langle BitSet \rangle \langle Index \rangle$ |
| <code>\bitsetSetValue</code> | $\langle BitSet \rangle \langle Index \rangle \langle Value \rangle$ |
| <code>\bitsetClearRange</code> , <code>\bitsetSetRange</code> , <code>\bitsetFlipRange</code> | $\langle BitSet \rangle \langle IndexFrom \rangle \langle IndexTo \rangle$ |
| <code>\bitsetSetValueRange</code> | $\langle BitSet \rangle \langle IndexFrom \rangle \langle IndexTo \rangle$ |

Bit retrieval^a (section 1.6.7)

| | |
|---|--|
| <code>\bitsetGet</code> | $\langle BitSet \rangle \langle Index \rangle$ |
| <code>\bitsetNextClearBit</code> , <code>\bitsetNextSetBit</code> | $\langle BitSet \rangle \langle Index \rangle$ |
| <code>\bitsetGetSetBitList</code> | $\langle BitSet \rangle$ |

Bit set properties (section 1.6.8)

| | |
|--|--------------------------|
| <code>\bitsetSize</code> , <code>\bitsetCardinality</code> | $\langle BitSet \rangle$ |
|--|--------------------------|

Queries^b (section 1.6.9)

| | |
|---|---|
| <code>\bitsetIsDefined</code> , <code>\bitsetIsEmpty</code> | $\langle BitSet \rangle \langle Then \rangle \langle Else \rangle$ |
| <code>\bitsetEquals</code> , <code>\bitsetIntersects</code> | $\langle BitSet A \rangle \langle BitSet B \rangle \langle Then \rangle \langle Else \rangle$ |
| <code>\bitsetQuery</code> | $\langle BitSet \rangle \langle Index \rangle \langle Then \rangle \langle Else \rangle$ |

^aMacros are expandable, full expansion by two steps.

^bMacros are expandable.

1.5 Package loading

The package can be used as normal L^AT_EX package:

```
\usepackage{bitset}
```

Also plain T_EX is supported:

```
\input bitset.sty\relax
```

1.6 Operators

The following macros work on and with bit sets. A bit set $\langle BitSet \rangle$ is represented by a name. The should consist of letters and digits. Technically it must survive

`\csname`. It is the same constraint that must be satisfied by label or environment names in L^AT_EX.

However active characters that are shorthands of package `babel` are not supported. Support for shorthands works by an assignment. But many operators such as `\bitsetGetDec` must be usable in expandable contexts. These assignments will not be executed in the best case or they will cause errors.

The bits in a bit set are addressed by non-negative integers starting from zero. Thus negative index numbers cause an error message. Because index numbers are T_EX numbers. The largest index is 2147483647. But in practice memory limits and patience limits will be very likely reached much before.

1.6.1 Miscellaneous

There isn't a separate operation for bit set creation. For simplicity an undefined bit set is treated as bit set with all bits cleared.

`\bitsetReset {⟨BitSet⟩}`

Macro `\bitsetReset` clears all bits. The result is an empty bit set. It may also be used as replacement for an operation “new”, because an undefined bit set is defined afterwards.

`\bitsetLet {⟨BitSet A⟩} {⟨BitSet B⟩}`

Macro `\bitsetLet` performs a simple assignment similar to T_EX's `\let`. After the operation `⟨BitSet A⟩` has the same value as `⟨BitSet B⟩`. If `⟨BitSet B⟩` is undefined, then `⟨BitSet A⟩` will be the empty bit set.

Note: If `⟨BitSet A⟩` exists, it will be overwritten.

1.6.2 Import

`\bitsetSetBin {⟨BitSet⟩} {⟨BinaryNumber⟩}`
`\bitsetSetOct {⟨BitSet⟩} {⟨OctalNumber⟩}`
`\bitsetSetHex {⟨BitSet⟩} {⟨HexadecimalNumber⟩}`

The numbers are interpreted as bit vectors and the flags in the bit `⟨BitSet⟩` set are set accordingly. These numeric arguments are the only arguments where spaces are allowed. Then the numbers are easier to read.

`\bitsetSetDec {⟨BitSet⟩} {⟨DecimalNumber⟩}`

Macro `\bitsetSetDec` uses `⟨DecimalNumber⟩` to set the bit set `⟨BitSet⟩`. The numeric argument must expand to a plain number consisting of decimal digits without command tokens or spaces. Internally this argument is expanded only. It cannot be passed to `\number` or `\numexpr`, because the number may be too large for them. However `\number` or `\the\numexpr` may be used explicitly. This also helps for unexpandable number command tokens or registers (`\z@`, `\@ne`, `\count@`, ...). Also L^AT_EX' `\value` needs prefixing:

```
\bitsetSetDec{foo}{\number\value{bar}}
```

1.6.3 Export

| |
|---|
| <code>\bitsetGetBin {⟨BitSet⟩} {⟨MinSize⟩}</code> |
| <code>\bitsetGetOct {⟨BitSet⟩} {⟨MinSize⟩}</code> |
| <code>\bitsetGetHex {⟨BitSet⟩} {⟨MinSize⟩}</code> |

These macros returns the bit set as binary, octal or hexadecimal number. If the bit size is smaller than $\langle MinSize \rangle$ the gap is filled with leading zeros. Example:

```
\bitsetReset{abc}
\bitsetSet{abc}{2}
\bitsetGetBin{abc}{8} → 00000100
\bitsetSet{abc}{5}\bitsetSet{abc}{7}
\bitsetGetHex{abc}{16} → 00A2
```

Macro `\bitsetGetHex` uses the uppercase letters A to F. The catcode of the letters is one of 11 (letter) or 12 (other).

| |
|---------------------------------------|
| <code>\bitsetGetDec {⟨BitSet⟩}</code> |
|---------------------------------------|

Macro `\bitsetGetDec` returns the bit set $\langle BitSet \rangle$ as decimal number. The returned number can be larger than T_EX's number limit of $2^{31} - 1$.

1.6.4 Logical operators

| |
|---|
| <code>\bitsetAnd {⟨BitSet A⟩} {⟨BitSet B⟩}</code> |
|---|

$$A_{\text{new}} := A_{\text{old}} \text{ and } B \quad (\forall \text{ bits})$$

| |
|--|
| <code>\bitsetAndNot {⟨BitSet A⟩} {⟨BitSet B⟩}</code> |
|--|

$$A_{\text{new}} := A_{\text{old}} \text{ and (not } B) \quad (\forall \text{ bits})$$

| |
|--|
| <code>\bitsetOr {⟨BitSet A⟩} {⟨BitSet B⟩}</code> |
|--|

$$A_{\text{new}} := A_{\text{old}} \text{ or } B \quad (\forall \text{ bits})$$

| |
|---|
| <code>\bitsetXor {⟨BitSet A⟩} {⟨BitSet B⟩}</code> |
|---|

$$A_{\text{new}} := A_{\text{old}} \text{ xor } B \quad (\forall \text{ bits})$$

1.6.5 Shifting

| |
|---|
| <code>\bitsetShiftLeft {⟨BitSet⟩} {⟨ShiftAmount⟩}</code> |
| <code>\bitsetShiftRight {⟨BitSet⟩} {⟨ShiftAmount⟩}</code> |

A left shift by one is a multiplication by two, thus left shifting moves the flags to higher positions. The new created low positions are filled by zeros.

A right shift is the opposite, dividing by two, moving the bits to lower positions. The number will become smaller, the lowest bits are lost.

If the $\langle ShiftAmount \rangle$ is negative, it reverts the meaning of the shift operation. A left shift becomes a right shift. A $\langle ShiftAmount \rangle$ of zero is ignored.

1.6.6 Bit manipulation

| |
|---|
| $\backslash\text{bitsetClear } \{\langle BitSet \rangle\} \{\langle Index \rangle\}$ $\backslash\text{bitsetSet } \{\langle BitSet \rangle\} \{\langle Index \rangle\}$ $\backslash\text{bitsetFlip } \{\langle BitSet \rangle\} \{\langle Index \rangle\}$ |
|---|

This macros manipulate a single bit in $\langle BitSet \rangle$ addressed by $\backslash Index$. Macro $\backslash\text{bitsetClear}$ disables the bit, $\backslash\text{bitsetSet}$ enables it and $\backslash\text{bitsetFlip}$ reverts the current setting of the bit.

| |
|---|
| $\backslash\text{bitsetSetValue } \{\langle BitSet \rangle\} \{\langle Index \rangle\} \{\langle Bit \rangle\}$ |
|---|

Macro $\backslash\text{bitsetSetValue}$ puts bit $\langle Bit \rangle$ at position $\langle Index \rangle$ in bit set $\langle BitSet \rangle$. $\langle Bit \rangle$ must be a valid TeX number equals to zero (disabled/cleared) or one (enabled/set).

1.6.7 Bit retrieval

| |
|--|
| $\backslash\text{bitsetGet } \{\langle BitSet \rangle\} \{\langle Index \rangle\}$ |
|--|

Macro $\backslash\text{bitsetGet}$ extracts the status of the bit at position $\langle Index \rangle$ in bit set $\langle BitSet \rangle$. Digit 1 is returned if the bit is set/enabled. If the bit is cleared/disabled and in cases of an undefined bitset or an index number out of range the return value is 0.

| |
|---|
| $\backslash\text{bitsetNextClearBit } \{\langle BitSet \rangle\} \{\langle Index \rangle\}$ |
|---|

Starting at position $\langle Index \rangle$ (inclusive) the bits are inspected. The first position without a set bit is returned. Possible results are decimal numbers: $\langle Index \rangle$, $\langle Index \rangle + 1, \dots, (\infty)$

| |
|---|
| $\backslash\text{bitsetNextSetBit } \{\langle BitSet \rangle\} \{\langle Index \rangle\}$ |
|---|

Starting at position $\langle Index \rangle$ (inclusive) the bits are inspected and the index position of the first found set bit is returned. If there isn't such a bit, then the result is -1. In summary possible results are decimal numbers: -1, $\langle Index \rangle$, $\langle Index \rangle + 1, \dots, (\infty)$

| |
|--|
| $\backslash\text{bitsetGetSetBitList } \{\langle BitSet \rangle\}$ |
|--|

Macro $\backslash\text{bitsetGetSetBitList}$ is an application for $\backslash\text{bitsetNextSetBit}$. The set bits are iterated and returned as comma separated list of index positions in increasing order. The list is empty in case of an empty bit set.

1.6.8 Bit set properties

`\bitsetSize {<BitSet>}`

Macro `\bitsetSize` returns number of bits in use. It is the same as the index number of the highest set/enabled bit incremented by one.

`\bitsetCardinality {<BitSet>}`

Macro `\bitsetCardinality` counts the number of set/enabled bits.

1.6.9 Queries

Also the query procedures are expandable. They ask for a piece of information about a bit set and execute code depending on the answer.

`\bitsetIsDefined {<BitSet>} {<Then>} {<Else>}`

If the bit set with the name `<BitSet>` exists the code given in `<Then>` is executed, otherwise `<Else>` is used.

`\bitsetIsEmpty {<BitSet>} {<Then>} {<Else>}`

If the bit set `<BitSet>` exists and at least one bit is set/enabled, the code in `<Then>` is executed, `<Else>` otherwise.

`\bitsetEquals {<BitSet A>} {<BitSet B>} {<Then>} {<Else>}`

Both bit sets are equal if and only if either both are undefined or both are defined and represents the same bit values at the same positions. Thus this definition is reflexive, symmetric, and transitive, enough for an equivalent relation.

`\bitsetIntersects {<BitSet A>} {<BitSet B>} {<Then>} {<Else>}`

If and only if `<BitSet A>` and `<BitSet B>` have at least one bit at the same position that is set, then code part `<Then>` is executed.

`\bitsetQuery {<BitSet>} {<Index>} {<Then>} {<Else>}`

It's just a wrapper for `\bitsetGet`. If the bit at position `<Index>` is enabled, code `<Then>` is called.

2 Implementation

The internal format of a bit set is quite simple, a sequence of digits 0 and 1. The least significant bit is left. A bit set without any flag set is encoded by 0. Also undefined bit sets are treated that way. After the highest bit that is set there are no further zeroes. A regular expression of valid bit sets values:

`0|[01]*1`

```
1 <*package>
```

2.1 Reload check and package identification

Reload check, especially if the package is not used with L^AT_EX.

```
2 \begingroup\catcode61\catcode48\catcode32=10\relax%
3   \catcode13=5 % ^~M
4   \endlinechar=13 %
5   \catcode35=6 % #
6   \catcode39=12 % '
7   \catcode44=12 % ,
8   \catcode45=12 % -
9   \catcode46=12 % .
10  \catcode58=12 % :
11  \catcode64=11 % @
12  \catcode123=1 % {
13  \catcode125=2 % }
14  \expandafter\let\expandafter\x\csname ver@bitset.sty\endcsname
15  \ifx\x\relax % plain-TEX, first loading
16  \else
17    \def\empty{}%
18    \ifx\x\empty % LATEX, first loading,
19    % variable is initialized, but \ProvidesPackage not yet seen
20    \else
21      \expandafter\ifx\csname PackageInfo\endcsname\relax
22        \def\x#1#2{%
23          \immediate\write-1{Package #1 Info: #2.}%
24        }%
25      \else
26        \def\x#1#2{\PackageInfo{#1}{#2, stopped}}%
27      \fi
28      \x{bitset}{The package is already loaded}%
29      \aftergroup\endinput
30    \fi
31  \fi
32 \endgroup%
```

Package identification:

```
33 \begingroup\catcode61\catcode48\catcode32=10\relax%
34   \catcode13=5 % ^~M
35   \endlinechar=13 %
36   \catcode35=6 % #
37   \catcode39=12 % '
38   \catcode40=12 % (
39   \catcode41=12 % )
40   \catcode44=12 % ,
41   \catcode45=12 % -
42   \catcode46=12 % .
43   \catcode47=12 % /
44   \catcode58=12 % :
45   \catcode64=11 % @
46   \catcode91=12 % [
47   \catcode93=12 % ]
48   \catcode123=1 % {
49   \catcode125=2 % }
50  \expandafter\ifx\csname ProvidesPackage\endcsname\relax
51    \def\x#1#2#3[#4]{\endgroup
52      \immediate\write-1{Package: #3 #4}%
```

```

53     \xdef#1{#4}%
54 }%
55 \else
56   \def\x#1#2[#3]{\endgroup
57     #2[#3]}%
58   \ifx#1\@undefined
59     \xdef#1{#3}%
60   \fi
61   \ifx#1\relax
62     \xdef#1{#3}%
63   \fi
64 }%
65 \fi
66 \expandafter\x\csname ver@bitset.sty\endcsname
67 \ProvidesPackage{bitset}%
68 [2016/05/16 v1.2 Handle bit-vector datatype (H0)]%

```

2.2 Catcodes

```

69 \begingroup\catcode61\catcode48\catcode32=10\relax%
70 \catcode13=5 % ^~M
71 \endlinechar=13 %
72 \catcode123=1 % {
73 \catcode125=2 % }
74 \catcode64=11 % @
75 \def\x{\endgroup
76   \expandafter\edef\csname BitSet@AtEnd\endcsname{%
77     \endlinechar=\the\endlinechar\relax
78     \catcode13=\the\catcode13\relax
79     \catcode32=\the\catcode32\relax
80     \catcode35=\the\catcode35\relax
81     \catcode61=\the\catcode61\relax
82     \catcode64=\the\catcode64\relax
83     \catcode123=\the\catcode123\relax
84     \catcode125=\the\catcode125\relax
85   }%
86 }%
87 \x\catcode61\catcode48\catcode32=10\relax%
88 \catcode13=5 % ^~M
89 \endlinechar=13 %
90 \catcode35=6 % #
91 \catcode64=11 % @
92 \catcode123=1 % {
93 \catcode125=2 % }
94 \def\TMP@EnsureCode#1#2{%
95   \edef\BitSet@AtEnd{%
96     \BitSet@AtEnd
97     \catcode#1=\the\catcode#1\relax
98   }%
99   \catcode#1=#2\relax
100 }
101 \TMP@EnsureCode{33}{12}% !
102 \TMP@EnsureCode{39}{12}% '
103 \TMP@EnsureCode{40}{12}% (
104 \TMP@EnsureCode{41}{12}% )
105 \TMP@EnsureCode{42}{12}% *
106 \TMP@EnsureCode{43}{12}% +
107 \TMP@EnsureCode{44}{12}% ,

```

```

108 \TMP@EnsureCode{45}{12}% -
109 \TMP@EnsureCode{46}{12}% .
110 \TMP@EnsureCode{47}{12}% /
111 \TMP@EnsureCode{58}{11}% : (letter!)
112 \TMP@EnsureCode{60}{12}% <
113 \TMP@EnsureCode{62}{12}% >
114 \TMP@EnsureCode{63}{14}% ? (comment!)
115 \TMP@EnsureCode{91}{12}% [
116 \TMP@EnsureCode{93}{12}% ]
117 \TMP@EnsureCode{96}{12}% '
118 \edef\BitSet@AtEnd{\BitSet@AtEnd\noexpand\endinput}
119 \begingroup\expandafter\expandafter\expandafter\endgroup
120 \expandafter\ifx\csname BitSet@TestMode\endcsname\relax
121 \else
122 \catcode63=9 % ? (ignore)
123 \fi
124 ? \let\BitSet@@TestMode\BitSet@TestMode

```

2.3 Package loading

```

125 \begingroup\expandafter\expandafter\expandafter\endgroup
126 \expandafter\ifx\csname RequirePackage\endcsname\relax
127 \def\TMP@RequirePackage#1[#2]{%
128 \begingroup\expandafter\expandafter\expandafter\endgroup
129 \expandafter\ifx\csname ver@#1.sty\endcsname\relax
130 \input #1.sty\relax
131 \fi
132 }%
133 \TMP@RequirePackage{infwarerr}[2007/09/09]%
134 \TMP@RequirePackage{intcalc}[2007/09/27]%
135 \TMP@RequirePackage{bigintcalc}[2007/09/27]%
136 \else
137 \RequirePackage{infwarerr}[2007/09/09]%
138 \RequirePackage{intcalc}[2007/09/27]%
139 \RequirePackage{bigintcalc}[2007/09/27]%
140 \fi

```

2.4 Help macros

2.4.1 Number constant

```

\BitSet@MaxSize
141 \def\BitSet@MaxSize{2147483647}%

```

2.4.2 General basic macros

```

\BitSet@Empty
142 \def\BitSet@Empty{}

\BitSet@FirstOfOne
143 \def\BitSet@FirstOfOne#1{#1}

\BitSet@Gobble
144 \def\BitSet@Gobble#1{}

\BitSet@FirstOfTwo
145 \def\BitSet@FirstOfTwo#1#2{#1}

\BitSet@SecondOfTwo
146 \def\BitSet@SecondOfTwo#1#2{#2}

```

\BitSet@Space

```
147 \def\BitSet@Space{ }
```

\BitSet@ZapSpace

```
148 \def\BitSet@ZapSpace#1 #2{%
149   #1%
150   \ifx\BitSet@Empty#2%
151   \else
152     \expandafter\BitSet@ZapSpace
153   \fi
154   #2%
155 }
```

2.4.3 Tail recursion

\BitSet@Fi

```
156 \let\BitSet@Fi\fi
```

\BitSet@AfterFi

```
157 \def\BitSet@AfterFi#1#2\BitSet@Fi{\fi#1}
```

\BitSet@AfterFiFi

```
158 \def\BitSet@AfterFiFi#1#2\BitSet@Fi{\fi\fi#1}%
```

\BitSet@AfterFiFiFi

```
159 \def\BitSet@AfterFiFiFi#1#2\BitSet@Fi{\fi\fi\fi#1}%
```

2.4.4 Check macros

\BitSet@IfUndefined

```
160 \def\BitSet@IfUndefined#1{%
161   \expandafter\ifx\csname BS@#1\endcsname\relax
162     \expandafter\BitSet@FirstOfTwo
163   \else
164     \expandafter\BitSet@SecondOfTwo
165   \fi
166 }
```

\BitSet@CheckIndex #1: continuation code
#2: BitSet
#3: Index

```
167 \def\BitSet@CheckIndex#1#2#3{%
168   \BitSet@IfUndefined{#2}{\bitsetReset{#2}}{ }%
169   \expandafter\expandafter\expandafter\BitSet@@CheckIndex
170   \intcalcNum{#3}!%
171   {#2}{#1}%
172 }
```

\BitSet@@CheckIndex #1: plain Index
#2: BitSet
#3: continuation code

```
173 \def\BitSet@@CheckIndex#1!#2#3{%
174   \ifnum#1<0 %
175     \BitSet@AfterFi{%
176       \@PackageError{bitset}{%
177         Invalid negative index (#1)%
```

```

178     }\@ehc
179   }%
180 \else
181   \BitSet@AfterFi{%
182     #3{#2}{#1}%
183   }%
184 \BitSet@Fi
185 }

```

2.5 Miscellaneous

\bitsetReset

```

186 \def\bitsetReset#1{%
187   \expandafter\def\csname BS@#1\endcsname{0}%
188 }

```

\bitsetLet

```

189 \def\bitsetLet#1#2{%
190   \BitSet@IfUndefined{#2}{%
191     \bitsetReset{#1}%
192   }{%
193     \expandafter\let\csname BS@#1\expandafter\endcsname
194       \csname BS@#2\endcsname
195   }%
196 }

```

2.6 Import

2.6.1 From binary number

\bitsetSetBin

```

197 \def\bitsetSetBin#1#2{%
198   \edef\BitSet@Temp{#2}%
199   \edef\BitSet@Temp{%
200     \expandafter\expandafter\expandafter\BitSet@ZapSpace
201     \expandafter\BitSet@Temp\BitSet@Space\BitSet@Empty
202   }%
203   \edef\BitSet@Temp{%
204     \expandafter\BitSet@KillZeros\BitSet@Temp\BitSet@Empty
205   }%
206   \ifx\BitSet@Temp\BitSet@Empty
207     \expandafter\let\csname BS@#1\endcsname\BitSet@Zero
208   \else
209     \expandafter\edef\csname BS@#1\endcsname{%
210       \expandafter\BitSet@Reverse\BitSet@Temp!%
211     }%
212   \fi
213 }

```

\BitSet@KillZeros

```

214 \def\BitSet@KillZeros#1{%
215   \ifx#10%
216     \expandafter\BitSet@KillZeros
217   \else
218     #1%
219   \fi
220 }

```

\BitSet@Reverse

```
221 \def\BitSet@Reverse#1#2!{%
222   \ifx\#2\%
223     #1%
224   \else
225     \BitSet@AfterFi{%
226       \BitSet@Reverse#2!#1%
227     }%
228   \BitSet@Fi
229 }
```

2.6.2 From octal/hex number

\bitsetSetOct

```
230 \def\bitsetSetOct{%
231   \BitSet@SetOctHex\BitSet@FromFirstOct
232 }
```

\bitsetSetHex

```
233 \def\bitsetSetHex{%
234   \BitSet@SetOctHex\BitSet@FromFirstHex
235 }
```

\BitSet@SetOctHex

```
236 \def\BitSet@SetOctHex#1#2#3{%
237   \edef\BitSet@Temp{#3}%
238   \edef\BitSet@Temp{%
239     \expandafter\expandafter\expandafter\BitSet@ZapSpace
240     \expandafter\BitSet@Temp\BitSet@Space\BitSet@Empty
241   }%
242   \edef\BitSet@Temp{%
243     \expandafter\BitSet@KillZeros\BitSet@Temp\BitSet@Empty
244   }%
245   \ifx\BitSet@Temp\BitSet@Empty
246     \expandafter\let\csname BS@#2\endcsname\BitSet@Zero
247   \else
248     \edef\BitSet@Temp{%
249       \expandafter#1\BitSet@Temp!%
250     }%
251     \ifx\BitSet@Temp\BitSet@Empty
252       \expandafter\let\csname BS@#2\endcsname\BitSet@Zero
253     \else
254       \expandafter\edef\csname BS@#2\endcsname{%
255         \expandafter\BitSet@Reverse\BitSet@Temp!%
256       }%
257     \fi
258   \fi
259 }
```

\BitSet@FromFirstOct

```
260 \def\BitSet@FromFirstOct#1{%
261   \ifx#1!%
262   \else
263     \ifcase#1 \BitSet@AfterFiFi\BitSet@FromFirstOct
264     \or 1%
265     \or 10%
266     \or 11%
```

```

267     \or 100%
268     \or 101%
269     \or 110%
270     \or 111%
271     \else \BitSetError:WrongOctalDigit%
272     \fi
273     \expandafter\BitSet@FromOct
274     \BitSet@Fi
275 }

\BitSet@FromOct
276 \def\BitSet@FromOct#1{%
277     \ifx#1!%
278     \else
279         \ifcase#1 000%
280         \or 001%
281         \or 010%
282         \or 011%
283         \or 100%
284         \or 101%
285         \or 110%
286         \or 111%
287         \else \BitSetError:WrongOctalDigit%
288         \fi
289         \expandafter\BitSet@FromOct
290     \fi
291 }

\BitSet@FromFirstHex
292 \def\BitSet@FromFirstHex#1{%
293     \ifx#1!%
294     \else
295         \ifx#10%
296             \BitSet@AfterFiFi\BitSet@FromFirstHex
297         \fi
298         \expandafter\ifx\csname BitSet@Hex#1\endcsname\relax
299             \BitSetError:InvalidHexDigit%
300         \else
301             \expandafter\expandafter\expandafter\BitSet@KillZeros
302             \csname BitSet@Hex#1\endcsname
303         \fi
304         \expandafter\BitSet@FromHex
305     \BitSet@Fi
306 }

\BitSet@FromHex
307 \def\BitSet@FromHex#1{%
308     \ifx#1!%
309     \else
310         \expandafter\ifx\csname BitSet@Hex#1\endcsname\relax
311             \BitSetError:InvalidHexDigit%
312         \else
313             \csname BitSet@Hex#1\endcsname
314         \fi
315         \expandafter\BitSet@FromHex
316     \fi
317 }

```


\BitSet@Hex[0..F]

```
318 \def\BitSet@Temp#1{%
319   \expandafter\def\csname BitSet@Hex#1\endcsname
320 }
321 \BitSet@Temp 0{0000}%
322 \BitSet@Temp 1{0001}%
323 \BitSet@Temp 2{0010}%
324 \BitSet@Temp 3{0011}%
325 \BitSet@Temp 4{0100}%
326 \BitSet@Temp 5{0101}%
327 \BitSet@Temp 6{0110}%
328 \BitSet@Temp 7{0111}%
329 \BitSet@Temp 8{1000}%
330 \BitSet@Temp 9{1001}%
331 \BitSet@Temp A{1010}%
332 \BitSet@Temp B{1011}%
333 \BitSet@Temp C{1100}%
334 \BitSet@Temp D{1101}%
335 \BitSet@Temp E{1110}%
336 \BitSet@Temp F{1111}%
337 \BitSet@Temp a{1010}%
338 \BitSet@Temp b{1011}%
339 \BitSet@Temp c{1100}%
340 \BitSet@Temp d{1101}%
341 \BitSet@Temp e{1110}%
342 \BitSet@Temp f{1111}%
```

2.6.3 From decimal number

\bitsetSetDec

```
343 \def\bitsetSetDec#1#2{%
344   \edef\BitSet@Temp{#2}%
345   \edef\BitSet@Temp{%
346     \expandafter\expandafter\expandafter\BitSet@ZapSpace
347     \expandafter\BitSet@Temp\BitSet@Space\BitSet@Empty
348   }%
349   \edef\BitSet@Temp{%
350     \expandafter\BitSet@KillZeros\BitSet@Temp\BitSet@Empty
351   }%
352   \ifx\BitSet@Temp\BitSet@Empty
353     \expandafter\let\csname BS@#1\endcsname\BitSet@Zero
354   \else
355     \ifcase\bigintcalcSgn{\BitSet@Temp} %
356       \expandafter\let\csname BS@#1\endcsname\BitSet@Zero
357     \or
358       \ifnum\bigintcalcCmp\BitSet@Temp\BitSet@MaxSize>0 %
359         \expandafter\edef\csname BS@#1\endcsname{%
360           \expandafter\BitSet@SetDecBig\BitSet@Temp!%
361         }%
362       \else
363         \expandafter\edef\csname BS@#1\endcsname{%
364           \expandafter\BitSet@SetDec\BitSet@Temp!%
365         }%
366       \fi
367     \else
368       \@PackageError{bitset}{%
369         Bit sets cannot be negative%
370       }\@ehc
```

```

371     \fi
372 \fi
373 }

\BitSet@SetDecBig
374 \def\BitSet@SetDecBig#1#2#3#4#5#6#7#8#9!{%
375     \ifx\#9\%
376         \BitSet@SetDec#1#2#3#4#5#6#7#8!%
377     \else
378         \ifcase\BigIntCalcOdd#1#2#4#5#6#7#8#9! %
379             0%
380         \or
381             1%
382 ?     \else\BitSetError:ThisCannotHappen%
383         \fi
384         \BitSet@AfterFi{%
385             \expandafter\expandafter\expandafter\BitSet@SetDecBig
386             \BigIntCalcShr#1#2#3#4#5#6#7#8#9!!%
387         }%
388     \BitSet@Fi
389 }

\BitSet@SetDec
390 \def\BitSet@SetDec#1!{%
391     \ifcase#1 %
392     \or 1%
393     \else
394         \ifodd#1 %
395             1%
396         \else
397             0%
398         \fi
399         \BitSet@AfterFi{%
400             \expandafter\expandafter\expandafter\BitSet@SetDec
401             \IntCalcShr#1!!%
402         }%
403     \BitSet@Fi
404 }

```

2.7 Export

2.7.1 To binary number

```

\bitsetGetBin
405 \def\bitsetGetBin#1#2{%
406     \romannumeral0%
407     \expandafter\expandafter\expandafter\BitSet@@GetBin
408     \intcalcNum{#2}{#1}%
409 }

\BitSet@@GetBin
410 \def\BitSet@@GetBin#1#2{%
411     \BitSet@IfUndefined{#2}{%
412         \ifnum#1>1 %
413             \BitSet@AfterFi{%
414                 \expandafter\expandafter\expandafter\BitSet@Fill
415                 \IntCalcDec#1!!0%
416             }%

```

```

417     \else
418         \BitSet@AfterFi{ 0}%
419     \BitSet@Fi
420 }{%
421     \expandafter\expandafter\expandafter\BitSet@NumBinRev
422     \expandafter\expandafter\expandafter1%
423     \expandafter\expandafter\expandafter!%
424     \csname BS@#2\endcsname!!#1!%
425 }%
426 }

\BitSet@Fill #1: number of leading digits 0
#2: result
427 \def\BitSet@Fill#1!{%
428     \ifnum#1>0 %
429         \BitSet@AfterFi{%
430             \expandafter\expandafter\expandafter\BitSet@Fill
431             \IntCalcDec#1!!0%
432         }%
433     \else
434         \BitSet@AfterFi{ }%
435     \BitSet@Fi
436 }

\BitSet@NumBinRev #1: bit counter (including #2)
#2#3: reverted number
#4: result
#5: min size
437 \def\BitSet@NumBinRev#1!#2#3!{%
438     \ifx\#3\%
439         \BitSet@AfterFi{%
440             \BitSet@NumBinFill#1!#2%
441         }%
442     \else
443         \BitSet@AfterFi{%
444             \expandafter\expandafter\expandafter\BitSet@NumBinRev
445             \IntCalcInc#1!!#3!#2%
446         }%
447     \BitSet@Fi
448 }

\BitSet@NumBinFill
449 \def\BitSet@NumBinFill#1!#2!#3!{%
450     \ifnum#3>#1 %
451         \BitSet@AfterFi{%
452             \expandafter\expandafter\expandafter\BitSet@Fill
453             \IntCalcSub#3!#1!!#2%
454         }%
455     \else
456         \BitSet@AfterFi{ #2}%
457     \BitSet@Fi
458 }

```

2.7.2 To octal/hexadecimal number

```

\bitsetGetOct
459 \def\bitsetGetOct#1#2{%

```

```

460 \romannumeral0%
461 \bitsetIsEmpty{#1}{%
462   \expandafter\expandafter\expandafter\BitSet@@GetOctHex
463   \intcalcNum{#2}!3!230%
464 }{%
465   \expandafter\expandafter\expandafter\BitSet@@GetOct
466   \expandafter\expandafter\expandafter1%
467   \expandafter\expandafter\expandafter!%
468   \expandafter\expandafter\expandafter!%
469   \csname BS@#1\endcsname00%
470   \BitSet@Empty\BitSet@Empty\BitSet@Empty!{#2}%
471 }%
472 }

```

\bitsetGetHex

```

473 \def\bitsetGetHex#1#2{%
474   \romannumeral0%
475   \bitsetIsEmpty{#1}{%
476     \expandafter\expandafter\expandafter\BitSet@@GetOctHex
477     \intcalcNum{#2}!4!340%
478   }{%
479     \expandafter\expandafter\expandafter\BitSet@@GetHex
480     \expandafter\expandafter\expandafter1%
481     \expandafter\expandafter\expandafter!%
482     \expandafter\expandafter\expandafter!%
483     \csname BS@#1\endcsname000%
484     \BitSet@Empty\BitSet@Empty\BitSet@Empty\BitSet@Empty!{#2}%
485   }%
486 }

```

\BitSet@@GetOct #1: number of digits

#2: result

#3#4#5: bits

```

487 \def\BitSet@@GetOct#1!#2!#3#4#5{%
488   \ifx#5\BitSet@Empty
489     \BitSet@AfterFi{%
490       \expandafter\expandafter\expandafter\BitSet@GetOctHex
491       \IntCalcDec#1!!#2!23%
492     }%
493   \else
494     \BitSet@AfterFi{%
495       \expandafter\expandafter\expandafter\BitSet@@GetOct
496       \number\IntCalcInc#1!\expandafter\expandafter\expandafter!%
497       \csname BitSet@Oct#5#4#3\endcsname#2!%
498     }%
499   \BitSet@Fi
500 }

```

\BitSet@Oct[000..111]

```

501 \def\BitSet@Temp#1#2#3#4{%
502   \expandafter\def\csname BitSet@Oct#1#2#3\endcsname{#4}%
503 }
504 \BitSet@Temp0000%
505 \BitSet@Temp0011%
506 \BitSet@Temp0102%
507 \BitSet@Temp0113%
508 \BitSet@Temp1004%
509 \BitSet@Temp1015%

```

```

510 \BitSet@Temp1106%
511 \BitSet@Temp1117%

\BitSet@@GetHex #1: number of digits
#2: result
#3#4#5#6: bits
512 \def\BitSet@@GetHex#1!#2!#3#4#5#6{%
513   \ifx#6\BitSet@Empty
514     \BitSet@AfterFi{%
515       \expandafter\expandafter\expandafter\BitSet@GetOctHex
516       \IntCalcDec#1!!#2!34%
517     }%
518   \else
519     \BitSet@AfterFi{%
520       \expandafter\expandafter\expandafter\BitSet@@GetHex
521       \number\IntCalcInc#1!\expandafter\expandafter\expandafter!%
522       \csname BitSet@Hex#6#5#4#3\endcsname#2!%
523     }%
524   \BitSet@Fi
525 }

\BitSet@Hex[0000..1111]

526 \def\BitSet@Temp#1#2#3#4#5{%
527   \expandafter\def\csname BitSet@Hex#1#2#3#4\endcsname{#5}%
528 }
529 \BitSet@Temp00000%
530 \BitSet@Temp00011%
531 \BitSet@Temp00102%
532 \BitSet@Temp00113%
533 \BitSet@Temp01004%
534 \BitSet@Temp01015%
535 \BitSet@Temp01106%
536 \BitSet@Temp01117%
537 \BitSet@Temp10008%
538 \BitSet@Temp10019%
539 \BitSet@Temp1010A%
540 \BitSet@Temp1011B%
541 \BitSet@Temp1100C%
542 \BitSet@Temp1101D%
543 \BitSet@Temp1110E%
544 \BitSet@Temp1111F%

\BitSet@GetOctHex Leading zeros  $(\#4 - \#1 * 3 + 2)/3$  if  $\#4 > \#1 * 3$ 
#1: digit size
#2: result
#3: bits per digit - 1
#4: bits per digit #5: garbage
#6: min size
545 \def\BitSet@GetOctHex#1!#2!#3#4#5!#6{%
546   \expandafter\BitSet@@GetOctHex
547   \number\IntCalcNum{#6}\expandafter\expandafter\expandafter!%
548   \IntCalcMul#1!#4!!#3#4#2%
549 }

\BitSet@@GetOctHex #1: plain min size
#2: digits * (bits per digit)

```

```

#3: bits per digit - 1
#4: bits per digit
550 \def\BitSet@@GetOctHex#1!#2!#3#4{%
551   \ifnum#1>#2 %
552     \BitSet@AfterFi{%
553       \expandafter\expandafter\expandafter\expandafter
554       \expandafter\expandafter\expandafter\BitSet@Fill
555       \expandafter\IntCalcDiv\number
556       \expandafter\expandafter\expandafter\IntCalcAdd
557       \IntCalcSub#1!#2!#3!#4!%
558     }%
559   \else
560     \BitSet@AfterFi{ }%
561   \BitSet@Fi
562 }

```

2.7.3 To decimal number

\bitsetGetDec

```

563 \def\bitsetGetDec#1{%
564   \romannumeral0%
565   \BitSet@IfUndefined{#1}{ 0}{%
566     \expandafter\expandafter\expandafter\BitSet@GetDec
567     \csname BS@#1\endcsname!%
568   }%
569 }

```

\BitSet@GetDec

```

570 \def\BitSet@GetDec#1#2!{%
571   \ifx\\#2\\%
572     \BitSet@AfterFi{ #1}%
573   \else
574     \BitSet@AfterFi{%
575       \BitSet@@GetDec2!#1!#2!%
576     }%
577   \BitSet@Fi
578 }

```

\BitSet@@GetDec #1: power of two
#2: result
#3#4: number

```

579 \def\BitSet@@GetDec#1!#2!#3#4!{%
580   \ifx\\#4\\%
581     \ifx#31%
582       \BitSet@AfterFiFi{%
583         \expandafter\expandafter\expandafter\BitSet@Space
584         \IntCalcAdd#1!#2!%
585       }%
586     \else
587       \BitSet@AfterFiFi{ #2}%
588     \fi
589   \else
590     \ifx#31%
591       \BitSet@AfterFiFi{%
592         \csname BitSet@N#1%
593         \expandafter\expandafter\expandafter\endcsname
594         \IntCalcAdd#1!#2!#4!%

```

```

595     }%
596     \else
597         \BitSet@AfterFiFi{%
598             \csname BitSet@N#1\endcsname#2!#4!%
599         }%
600     \fi
601     \BitSet@Fi
602 }

\BitSet@N[1,2,4,...]

603 \def\BitSet@Temp#1#2{%
604     \expandafter\def\csname BitSet@N#1\endcsname{%
605         \BitSet@@GetDec#2!%
606     }%
607 }
608 \BitSet@Temp{1}{2}
609 \BitSet@Temp{2}{4}
610 \BitSet@Temp{4}{8}
611 \BitSet@Temp{8}{16}
612 \BitSet@Temp{16}{32}
613 \BitSet@Temp{32}{64}
614 \BitSet@Temp{64}{128}
615 \BitSet@Temp{128}{256}
616 \BitSet@Temp{256}{512}
617 \BitSet@Temp{512}{1024}
618 \BitSet@Temp{1024}{2048}
619 \BitSet@Temp{2048}{4096}
620 \BitSet@Temp{4096}{8192}
621 \BitSet@Temp{8192}{16384}
622 \BitSet@Temp{16384}{32768}
623 \BitSet@Temp{32768}{65536}
624 \BitSet@Temp{65536}{131072}
625 \BitSet@Temp{131072}{262144}
626 \BitSet@Temp{262144}{524288}
627 \BitSet@Temp{524288}{1048576}
628 \BitSet@Temp{1048576}{2097152}
629 \BitSet@Temp{2097152}{4194304}
630 \BitSet@Temp{4194304}{8388608}
631 \BitSet@Temp{8388608}{16777216}
632 \BitSet@Temp{16777216}{33554432}
633 \BitSet@Temp{33554432}{67108864}
634 \BitSet@Temp{67108864}{134217728}
635 \BitSet@Temp{134217728}{268435456}
636 \BitSet@Temp{268435456}{536870912}
637 \BitSet@Temp{536870912}{1073741824}

\BitSet@N1073741824

638 \expandafter\def\csname BitSet@N1073741824\endcsname{%
639     \BitSet@@GetDecBig2147483648!%
640 }%

\BitSet@@GetDecBig #1: current power of two
#2: result
#3#4: number

641 \def\BitSet@@GetDecBig#1!#2!#3#4!{%
642     \ifx\#4\%
643         \ifx#31%
644             \BitSet@AfterFiFi{%

```

```

645     \expandafter\expandafter\expandafter\BitSet@Space
646     \BigIntCalcAdd#1!#2!%
647 }%
648 \else
649     \BitSet@AfterFiFi{ #2}%
650 \fi
651 \else
652     \ifx#31%
653         \BitSet@AfterFiFi{%
654             \expandafter\expandafter\expandafter\BitSet@@GetDecBig
655             \BigIntCalcAdd#1!#2!!#1!#4!%
656         }%
657     \else
658         \BitSet@AfterFiFi{%
659             \expandafter\expandafter\expandafter\BitSet@GetDecBig
660             \BigIntCalcShl#1!!#2!#4!%
661         }%
662     \fi
663 \BitSet@Fi
664 }

\BitSet@@GetDecBig #1: result
#2: power of two
#3#4: number
665 \def\BitSet@@GetDecBig#1!#2!{%
666     \expandafter\expandafter\expandafter\BitSet@GetDecBig
667     \BigIntCalcShl#2!!#1!%
668 }

```

2.8 Logical operators

2.8.1 \bitsetAnd

\bitsetAnd Decision table for \bitsetAnd:

| | undef(B) | empty(B) | cardinality(B)>0 |
|------------------|------------|------------|------------------|
| undef(A) | A := empty | A := empty | A := empty |
| empty(A) | | | |
| cardinality(A)>0 | A := empty | A := empty | A &= B |

```

669 \def\bitsetAnd#1#2{%
670     \bitsetIsEmpty{#1}{%
671         \bitsetReset{#1}%
672     }{%
673         \bitsetIsEmpty{#2}{%
674             \bitsetReset{#1}%
675         }{%
676             \expandafter\edef\csname BS@#1\endcsname{%
677                 \expandafter\expandafter\expandafter\BitSet@And
678                 \csname BS@#1\expandafter\expandafter\expandafter\endcsname
679                 \expandafter\expandafter\expandafter!%
680                 \csname BS@#2\endcsname!%!%
681             }%
682             \expandafter\ifx\csname BS@#1\endcsname\BitSet@Empty
683                 \bitsetReset{#1}%
684             \fi
685         }%
686     }%
687 }

```


\BitSet@And

```

688 \def\BitSet@And#1#2!#3#4!#5!{%
689   \ifx\#2\%
690     \ifnum#1#3=11 #51\fi
691   \else
692     \ifx\#4\%
693       \ifnum#1#3=11 #51\fi
694     \else
695       \ifnum#1#3=11 %
696         #51%
697         \BitSet@AfterFiFiFi{%
698           \BitSet@And#2!#4!!%
699         }%
700     \else
701       \BitSet@AfterFiFiFi{%
702         \BitSet@And#2!#4!#50!%
703       }%
704     \fi
705   \fi
706 \BitSet@Fi
707 }

```

2.8.2 \bitsetAndNot

\bitsetAndNot Decision table for \bitsetAndNot:

| | undef(B) | empty(B) | cardinality(B)>0 |
|------------------|------------|------------|------------------|
| undef(A) | A := empty | A := empty | A := empty |
| empty(A) | | | |
| cardinality(A)>0 | | | A &= !B |

```

708 \def\bitsetAndNot#1#2{%
709   \bitsetIsEmpty{#1}{%
710     \bitsetReset{#1}%
711   }{%
712     \bitsetIsEmpty{#2}{%
713     }{%
714       \expandafter\edef\csname BS@#1\endcsname{%
715         \expandafter\expandafter\expandafter\BitSet@AndNot
716         \csname BS@#1\expandafter\expandafter\expandafter\endcsname
717         \expandafter\expandafter\expandafter!%
718         \csname BS@#2\endcsname!!%
719       }%
720       \expandafter\ifx\csname BS@#1\endcsname\BitSet@Empty
721         \bitsetReset{#1}%
722       \fi
723     }%
724   }%
725 }

```

\BitSet@AndNot

```

726 \def\BitSet@AndNot#1#2!#3#4!#5!{%
727   \ifx\#2\%
728     \ifnum#1#3=10 #51\fi
729   \else
730     \ifx\#4\%
731       #5%
732     \ifnum#1#3=10 1\else 0\fi

```

```

733     #2%
734   \else
735     \ifnum#1#3=10 %
736       #51%
737       \BitSet@AfterFiFiFi{%
738         \BitSet@AndNot#2!#4!!%
739       }%
740   \else
741     \BitSet@AfterFiFiFi{%
742       \BitSet@AndNot#2!#4!#50!%
743     }%
744   \fi
745 \fi
746 \BitSet@Fi
747 }

```

2.8.3 \bitsetOr

\bitsetOr Decision table for \bitsetOr:

| | undef(B) | empty(B) | cardinality(B)>0 |
|------------------|------------|------------|------------------|
| undef(A) | A := empty | A := empty | A := B |
| empty(A) | | | A := B |
| cardinality(A)>0 | | | A = B |

```

748 \def\bitsetOr#1#2{%
749   \bitsetIsEmpty{#2}{%
750     \BitSet@IfUndefined{#1}{\bitsetReset{#1}}{%
751     }{%
752       \bitsetIsEmpty{#1}{%
753         \expandafter\let\csname BS@#1\expandafter\endcsname
754           \csname BS@#2\endcsname
755       }{%
756         \expandafter\edef\csname BS@#1\endcsname{%
757           \expandafter\expandafter\expandafter\BitSet@Or
758             \csname BS@#1\expandafter\expandafter\expandafter\endcsname
759           \expandafter\expandafter\expandafter!%
760             \csname BS@#2\endcsname!%
761         }%
762       }%
763     }%
764 }

```

\BitSet@Or

```

765 \def\BitSet@Or#1#2!#3#4!{%
766   \ifnum#1#3>0 1\else 0\fi
767   \ifx\#2\%
768     #4%
769   \else
770     \ifx\#4\%
771       #2%
772     \else
773       \BitSet@AfterFiFi{%
774         \BitSet@Or#2!#4!%
775       }%
776     \fi
777   \BitSet@Fi
778 }

```

2.8.4 \bitsetXor

\bitsetXor Decision table for \bitsetXor:

| | undef(B) | empty(B) | cardinality(B)>0 |
|------------------|------------|------------|------------------|
| undef(A) | A := empty | A := empty | A := B |
| empty(A) | | | A := B |
| cardinality(A)>0 | | | A $\hat{=}$ B |

```

779 \def\bitsetXor#1#2{%
780   \bitsetIsEmpty{#2}{%
781     \BitSet@IfUndefined{#1}{\bitsetReset{#1}}{}%
782   }{%
783     \bitsetIsEmpty{#1}{%
784       \expandafter\let\csname BS@#1\expandafter\endcsname
785         \csname BS@#2\endcsname
786     }{%
787       \expandafter\edef\csname BS@#1\endcsname{%
788         \expandafter\expandafter\expandafter\BitSet@Xor
789         \csname BS@#1\expandafter\expandafter\expandafter\endcsname
790         \expandafter\expandafter\expandafter!%
791         \csname BS@#2\endcsname!!%
792       }%
793       \expandafter\ifx\csname BS@#1\endcsname\BitSet@Empty
794         \bitsetReset{#1}%
795       \fi
796     }%
797   }%
798 }
```

\BitSet@Xor

```

799 \def\BitSet@Xor#1#2!#3#4!#5!{%
800   \ifx\#2\%
801     \ifx#1#3%
802       \ifx\#4\%
803         \else
804           #50#4%
805         \fi
806       \else
807         #51#4%
808       \fi
809     \else
810       \ifx\#4\%
811         #5%
812         \ifx#1#30\else 1\fi
813         #2%
814       \else
815         \ifx#1#3%
816           \BitSet@AfterFiFiFi{%
817             \BitSet@Xor#2!#4!#50!%
818           }%
819         \else
820           #51%
821           \BitSet@AfterFiFiFi{%
822             \BitSet@Xor#2!#4!!%
823           }%
824         \fi
825       \fi
826     \BitSet@Fi
```

827 }

2.8.5 Shifting

2.8.6 \bitsetShiftLeft

\bitsetShiftLeft

```
828 \def\bitsetShiftLeft#1#2{%
829   \BitSet@IfUndefined{#1}{%
830     \bitsetReset{#1}%
831   }{%
832     \bitsetIsEmpty{#1}{%
833     }{%
834       \expandafter\expandafter\expandafter\BitSet@ShiftLeft
835       \intcalcNum{#2}!{#1}%
836     }%
837   }%
838 }
```

\BitSet@ShiftLeft

```
839 \def\BitSet@ShiftLeft#1!#2{%
840   \ifcase\intcalcSgn{#1} %
841   \or
842     \begingroup
843     \uccode'm='0 %
844     \uppercase\expandafter{\expandafter\endgroup
845     \expandafter\edef\csname BS@#2\expandafter\endcsname
846     \expandafter{%
847       \romannumeral#1000\expandafter\BitSet@Space
848       \csname BS@#2\endcsname
849     }%
850   }%
851   \else
852     \expandafter\BitSet@ShiftRight\BitSet@Gobble#1!{#2}%
853   \fi
854 }
```

2.8.7 \bitsetShiftRight

\bitsetShiftRight

```
855 \def\bitsetShiftRight#1#2{%
856   \BitSet@IfUndefined{#1}{%
857     \bitsetReset{#1}%
858   }{%
859     \bitsetIsEmpty{#1}{%
860     }{%
861       \expandafter\expandafter\expandafter\BitSet@ShiftRight
862       \intcalcNum{#2}!{#1}%
863     }%
864   }%
865 }
```

\BitSet@ShiftRight

```
866 \def\BitSet@ShiftRight#1!#2{%
867   \ifcase\intcalcSgn{#1} %
868   \or
869     \expandafter\edef\csname BS@#2\endcsname{%
870     \expandafter\expandafter\expandafter\BitSet@Kill
```

```

871     \csname BS@#2\expandafter\endcsname\expandafter\BitSet@Empty
872     \expandafter=%
873     \expandafter{\expandafter}\expandafter{\expandafter}%
874     \romannumeral#1000!%
875   }%
876 \else
877   \expandafter\BitSet@ShiftLeft\BitSet@Gobble#1!{#2}%
878 \fi
879 }

\BitSet@Kill
880 \def\BitSet@Kill#1#2=#3#4#5{%
881   #3#4%
882   \ifx#5!%
883     \ifx#1\BitSet@Empty
884       0%
885     \else
886       #1#2%
887     \fi
888   \else
889     \ifx#1\BitSet@Empty
890       0%
891       \BitSet@AfterFiFi\BitSet@Cleanup
892     \else
893       \BitSet@Kill#2=%
894     \fi
895   \BitSet@Fi
896 }

```

2.9 Bit manipulation

```

\bitsetClear
897 \def\bitsetClear{%
898   \BitSet@CheckIndex\BitSet@Clear
899 }

\bitsetSet
900 \def\bitsetSet{%
901   \BitSet@CheckIndex\BitSet@Set
902 }

\bitsetFlip
903 \def\bitsetFlip{%
904   \BitSet@CheckIndex\BitSet@Flip
905 }

\bitsetSetValue
906 \def\bitsetSetValue#1#2#3{%
907   \expandafter\expandafter\expandafter\BitSet@SetValue
908   \intcalculus{#3}!{#1}{#2}%
909 }

\BitSet@SetValue #1: plain value
#2: BitSet
#3: Index
910 \def\BitSet@SetValue#1!{%
911   \BitSet@CheckIndex{%

```

```

912 \ifcase#1 %
913 \expandafter\BitSet@Clear
914 \or
915 \expandafter\BitSet@Set
916 \else
917 \BitSet@ErrorInvalidBitValue{#1}%
918 \expandafter\expandafter\expandafter\BitSet@Gobble
919 \expandafter\BitSet@Gobble
920 \fi
921 }%
922 }

```

\BitSet@ErrorInvalidBitValue #1: Wrong bit value

```

923 \def\BitSet@ErrorInvalidBitValue#1{%
924 \@PackageError{bitset}{%
925 Invalid bit value (#1) not in range 0..1%
926 }\@ehc
927 }

```

2.9.1 Clear operation

\BitSet@Clear #1: BitSet

#2: plain and checked index

```

928 \def\BitSet@Clear#1#2{%
929 \edef\BitSet@Temp{%
930 \expandafter\expandafter\expandafter\BitSet@@Clear
931 \csname BS@#1\expandafter\endcsname
932 \expandafter\BitSet@Empty\expandafter=\expandafter!%
933 \romannumeral#2000!%
934 }%
935 \expandafter\let\csname BS@#1\expandafter\endcsname
936 \ifx\BitSet@Temp\BitSet@Empty
937 \BitSet@Zero
938 \else
939 \BitSet@Temp
940 \fi
941 }

```

\BitSet@@Clear

```

942 \def\BitSet@@Clear#1#2=#3!#4{%
943 \ifx#4!%
944 \ifx#1\BitSet@Empty
945 \else
946 \ifx\BitSet@Empty#2%
947 \else
948 #30#2%
949 \fi
950 \fi
951 \else
952 \ifx#1\BitSet@Empty
953 \BitSet@AfterFiFi\BitSet@Cleanup
954 \else
955 \ifx#10%
956 \BitSet@AfterFiFiFi{%
957 \BitSet@@Clear#2=#30!%
958 }%
959 \else
960 #31%

```

```

961         \BitSet@AfterFiFiFi{%
962         \BitSet@@Clear#2=!%
963     }%
964     \fi
965     \fi
966 \BitSet@Fi
967 }

```

2.9.2 Set operation

```

\BitSet@Set #1: BitSet
#2: plain and checked Index
968 \def\BitSet@Set#1#2{%
969     \expandafter\edef\csname BS@#1\endcsname{%
970         \expandafter\expandafter\expandafter\BitSet@@Set
971         \csname BS@#1\expandafter\endcsname
972         \expandafter\BitSet@Empty\expandafter=%
973         \expandafter{\expandafter}\expandafter{\expandafter}%
974         \romannumeral#2000!%
975     }%
976 }

\BitSet@@Set
977 \def\BitSet@@Set#1#2=#3#4#5{%
978     #3#4%
979     \ifx#5!%
980         1#2%
981     \else
982         \ifx#1\BitSet@Empty
983             0%
984             \BitSet@AfterFiFi\BitSet@@@Set
985         \else
986             #1%
987             \BitSet@@Set#2=%
988         \fi
989     \BitSet@Fi
990 }

\BitSet@@@Set
991 \def\BitSet@@@Set#1{%
992     \ifx#1!%
993         1%
994     \else
995         0%
996         \expandafter\BitSet@@@Set
997     \fi
998 }

```

2.9.3 Flip operation

```

\BitSet@Flip #1: BitSet
#2: plain and checked Index
999 \def\BitSet@Flip#1#2{%
1000     \edef\BitSet@Temp{%
1001         \expandafter\expandafter\expandafter\BitSet@@Flip
1002         \csname BS@#1\expandafter\endcsname
1003         \expandafter\BitSet@Empty\expandafter=\expandafter!%

```

```

1004     \romannumeral#2000!%
1005 }%
1006 \expandafter\let\csname BS@#1\expandafter\endcsname
1007 \ifx\BitSet@Temp\BitSet@Empty
1008     \BitSet@Zero
1009 \else
1010     \BitSet@Temp
1011 \fi
1012 }

```

\BitSet@@Flip

```

1013 \def\BitSet@@Flip#1#2=#3!#4{%
1014     \ifx#4!%
1015         \ifx#11%
1016             \ifx\BitSet@Empty#2%
1017                 \else
1018                     #30#2%
1019                 \fi
1020             \else
1021                 #31#2%
1022             \fi
1023         \else
1024             \ifx#1\BitSet@Empty
1025                 #30%
1026                 \BitSet@AfterFiFi\BitSet@@@Set
1027             \else
1028                 \ifx#10%
1029                     \BitSet@AfterFiFiFi{%
1030                         \BitSet@@Flip#2=#30!%
1031                     }%
1032                 \else
1033                     #31%
1034                     \BitSet@AfterFiFiFi{%
1035                         \BitSet@@Flip#2=!%
1036                     }%
1037                 \fi
1038             \fi
1039         \BitSet@Fi
1040 }

```

2.9.4 Range operators

\bitsetClearRange

```

1041 \def\bitsetClearRange{%
1042     \BitSet@Range\BitSet@Clear
1043 }

```

\bitsetSetRange

```

1044 \def\bitsetSetRange{%
1045     \BitSet@Range\BitSet@Set
1046 }

```

\bitsetFlipRange

```

1047 \def\bitsetFlipRange{%
1048     \BitSet@Range\BitSet@Flip
1049 }

```


\bitsetSetValueRange

```
1050 \def\bitsetSetValueRange#1#2#3#4{%
1051   \expandafter\expandafter\expandafter\BitSet@SetValueRange
1052   \intcalcNum{#4}!\{#1\}{#2\}{#3}%
1053 }
```

\BitSet@SetValueRange

```
1054 \def\BitSet@SetValueRange#1!#2#3#4{%
1055   \ifcase#1 %
1056     \BitSet@Range\BitSet@Clear{#2}{#3}{#4}%
1057   \or
1058     \BitSet@Range\BitSet@Set{#2}{#3}{#4}%
1059   \else
1060     \BitSet@ErrorInvalidBitValue{#1}%
1061   \fi
1062 }
```

\BitSet@Range #1: clear/set/flip macro

#2: BitSet

#3: Index from

#4: Index to

```
1063 \def\BitSet@Range#1#2#3#4{%
1064   \edef\BitSet@Temp{%
1065     \noexpand\BitSet@@Range\noexpand#1{#2}%
1066     \intcalcNum{#3}!\intcalcNum{#4}!%
1067   }%
1068   \BitSet@Temp
1069 }
```

\BitSet@@Range #1: clear/set/flip macro

#2: BitSet

#3: Index from

#4: Index to

```
1070 \def\BitSet@@Range#1#2#3!#4!{%
1071   \ifnum#3<0 %
1072     \BitSet@NegativeIndex#1{#2}#3!#4!0!#4!%
1073   \else
1074     \ifnum#4<0 %
1075       \BitSet@NegativeIndex#1{#2}#3!#4!#3!0!%
1076     \else
1077       \ifcase\intcalcCmp{#3}{#4} %
1078       \or
1079         \@PackageError{bitset}{%
1080           Wrong index numbers in range [#3..#4]\MessageBreak% hash-ok
1081           for clear/set/flip on bit set '#2'.\MessageBreak
1082           The lower index exceeds the upper index.\MessageBreak
1083           Canceling the operation as error recovery%
1084         }\@ehc
1085       \else
1086         \BitSet@@@Range#3!#4!#1{#2}%
1087       \fi
1088     \fi
1089   \fi
1090 }
```

\BitSet@NegativeIndex

```
1091 \def\BitSet@NegativeIndex#1#2#3!#4!#5!#6!{%
```

```

1092 \@PackageError{bitset}{%
1093     Negative index in range [#3..#4]\MessageBreak % hash-ok
1094     for \string\bitset
1095     \ifx#1\BitSet@Clear
1096         Clear%
1097     \else
1098         \ifx#1\BitSet@Set
1099             Set%
1100         \else
1101             Flip%
1102         \fi
1103     \fi
1104     Range on bit set '#2'.\MessageBreak
1105     Using [#5..#6] as error recovery% hash-ok
1106 } \@ehc
1107 \BitSet@@Range#1{#2}#5!#6!%
1108 }

```

\BitSet@@Range

```

1109 \def\BitSet@@Range#1!#2!#3#4{%
1110     \ifnum#1<#2 %
1111         #3{#4}{#1}%
1112     \BitSet@AfterFi{%
1113         \expandafter\expandafter\expandafter\BitSet@@Range
1114         \IntCalcInc#1!!#2!#3{#4}%
1115     }%
1116     \BitSet@Fi
1117 }

```

2.10 Bit retrieval

2.10.1 \bitsetGet

\bitsetGet

```

1118 \def\bitsetGet#1#2{%
1119     \number
1120     \expandafter\expandafter\expandafter\BitSet@Get
1121     \intcalcNum{#2}!{#1}%
1122 }

```

\BitSet@Get #1: plain index
#2: BitSet

```

1123 \def\BitSet@Get#1!#2{%
1124     \ifnum#1<0 %
1125         \BitSet@AfterFi{%
1126             0 \BitSetError:NegativeIndex%
1127         }%
1128     \else
1129         \BitSet@IfUndefined{#2}{0}{%
1130             \expandafter\expandafter\expandafter\BitSet@@Get
1131             \csname BS@#2\expandafter\endcsname
1132             \expandafter!\expandafter=%
1133             \expandafter{\expandafter}\expandafter{\expandafter}%
1134             \romannumeral\intcalcNum{#1}000!%
1135         }%
1136         \expandafter\BitSet@Space
1137     \BitSet@Fi
1138 }

```

\BitSet@@Get

```
1139 \def\BitSet@@Get#1#2=#3#4#5{%
1140   #3#4%
1141   \ifx#5!%
1142     \ifx#1!%
1143       0%
1144     \else
1145       #1%
1146     \fi
1147   \else
1148     \ifx#1!%
1149       0%
1150     \BitSet@AfterFiFi\BitSet@Cleanup
1151   \else
1152     \BitSet@@Get#2=%
1153   \fi
1154   \BitSet@Fi
1155 }
```

2.10.2 \bitsetNextClearBit, \bitsetNextSetBit

\bitsetNextClearBit

```
1156 \def\bitsetNextClearBit#1#2{%
1157   \number
1158   \expandafter\expandafter\expandafter\BitSet@NextClearBit
1159   \intcalculus{#2}!{#1} %
1160 }
```

\BitSet@NextClearBit

```
#1: Index
#2: BitSet
1161 \def\BitSet@NextClearBit#1!#2{%
1162   \ifnum#1<0 %
1163     \BitSet@NextClearBit0!{#2}%
1164     \BitSet@AfterFi{%
1165       \expandafter\BitSet@Space
1166       \expandafter\BitSetError:NegativeIndex\romannumeral0%
1167     }%
1168   \else
1169     \bitsetIsEmpty{#2}{#1}{%
1170       \expandafter\BitSet@Skip
1171       \number#1\expandafter\expandafter\expandafter!%
1172       \csname BS@#2\endcsname!!!!!!!!!=%
1173       {\BitSet@@NextClearBit#1!}%
1174     }%
1175     \BitSet@Fi
1176 }
```

\BitSet@@NextClearBit

```
#1: index for next bit in #2
#2: next bit
1177 \def\BitSet@@NextClearBit#1!#2{%
1178   \ifx#2!%
1179     #1%
1180   \else
1181     \ifx#20%
1182       #1%
1183     \BitSet@AfterFiFi\BitSet@Cleanup
1184   \else
```

```

1185     \BitSet@AfterFiFi{%
1186     \expandafter\expandafter\expandafter\BitSet@@NextClearBit
1187     \IntCalcInc#1!!%
1188     }%
1189     \fi
1190 \BitSet@Fi
1191 }

\bitsetNextSetBit
1192 \def\bitsetNextSetBit#1#2{%
1193   \number
1194   \expandafter\expandafter\expandafter\BitSet@NextSetBit
1195   \intcalcNum{#2}!{#1} %
1196 }

\BitSet@NextSetBit #1: Index
#2: BitSet
1197 \def\BitSet@NextSetBit#1!#2{%
1198   \ifnum#1<0 %
1199     \BitSet@NextSetBit0!{#2}%
1200     \BitSet@AfterFiFi{%
1201       \expandafter\BitSet@Space
1202       \expandafter\BitSetError:NegativeIndex\romannumeral0%
1203     }%
1204   \else
1205     \bitsetIsEmpty{#2}{-1}{%
1206       \expandafter\BitSet@Skip
1207       \number#1\expandafter\expandafter\expandafter!%
1208       \csname BS@#2\endcsname!!!!!!!!!=%
1209       {\BitSet@@NextSetBit#1!}%
1210     }%
1211     \BitSet@Fi
1212 }

\BitSet@@NextSetBit #1: index for next bit in #2
#2: next bit
1213 \def\BitSet@@NextSetBit#1!#2{%
1214   \ifx#2!%
1215     -1%
1216   \else
1217     \ifx#21%
1218       #1%
1219       \BitSet@AfterFiFi\BitSet@Cleanup
1220     \else
1221       \BitSet@AfterFiFi{%
1222         \expandafter\expandafter\expandafter\BitSet@@NextSetBit
1223         \IntCalcInc#1!!%
1224       }%
1225     \fi
1226     \BitSet@Fi
1227 }

\BitSet@Cleanup
1228 \def\BitSet@Cleanup#1!{ }

\BitSet@Skip #1: number of bits to skip
#2: bits
#3: continuation code

```

```

1229 \def\BitSet@Skip#1!#2{%
1230   \ifx#2!%
1231     \BitSet@AfterFi{%
1232       \BitSet@SkipContinue%
1233     }%
1234   \else
1235     \ifcase#1 %
1236       \BitSet@AfterFiFi{%
1237         \BitSet@SkipContinue#2%
1238       }%
1239     \or
1240       \BitSet@AfterFiFi\BitSet@SkipContinue
1241     \or
1242       \BitSet@AfterFiFi{%
1243         \expandafter\BitSet@SkipContinue\BitSet@Gobble
1244       }%
1245     \else
1246       \ifnum#1>8 %
1247         \BitSet@AfterFiFiFiFi{%
1248           \expandafter\BitSet@Skip
1249           \number\IntCalcSub#1!8!\expandafter!%
1250           \BitSet@GobbleSeven
1251         }%
1252       \else
1253         \BitSet@AfterFiFiFiFi{%
1254           \expandafter\expandafter\expandafter\BitSet@Skip
1255           \IntCalcDec#1!!%
1256         }%
1257       \fi
1258     \fi
1259   \BitSet@Fi
1260 }

```

\BitSet@SkipContinue #1: remaining bits
#2: continuation code

```

1261 \def\BitSet@SkipContinue#1!#2=#3{%
1262   #3#1!%
1263 }

```

\BitSet@GobbleSeven

```

1264 \def\BitSet@GobbleSeven#1#2#3#4#5#6#7{}

```

2.10.3 \bitsetGetSetBitList

\bitsetGetSetBitList It's just a wrapper for \bitsetNextSetBit.

```

1265 \def\bitsetGetSetBitList#1{%
1266   \romannumeral0%
1267   \bitsetIsEmpty{#1}{ }{%
1268     \expandafter\BitSet@GetSetBitList
1269     \number\BitSet@NextSetBit0!{#1}!{#1}{ }!%
1270   }%
1271 }

```

\BitSet@GetSetBitList #1: found index
#2: BitSet
#3: comma #4: result

```

1272 \def\BitSet@GetSetBitList#1!#2#3#4!{%

```

```

1273 \ifnum#1<0 %
1274 \BitSet@AfterFi{ #4}%
1275 \else
1276 \BitSet@AfterFi{%
1277 \expandafter\BitSet@GetSetBitList\number
1278 \expandafter\expandafter\expandafter\BitSet@NextSetBit
1279 \IntCalcInc#1!!{#2}!{#2},#4#3#1!%
1280 }%
1281 \BitSet@Fi
1282 }

```

2.11 Bit set properties

\bitsetSize

```

1283 \def\bitsetSize#1{%
1284 \number
1285 \BitSet@IfUndefined{#1}{0 }{%
1286 \expandafter\expandafter\expandafter\BitSet@Size
1287 \expandafter\expandafter\expandafter1%
1288 \expandafter\expandafter\expandafter!%
1289 \csname BS@#1\endcsname!0!%
1290 }%
1291 }

```

\BitSet@Size #1: counter
#2#3: bits
#4: result

```

1292 \def\BitSet@Size#1!#2#3!#4!{%
1293 \ifx#21%
1294 \ifx\#3\%
1295 \BitSet@AfterFiFi{#1 }%
1296 \else
1297 \BitSet@AfterFiFi{%
1298 \expandafter\expandafter\expandafter\BitSet@Size
1299 \IntCalcInc#1!!#3!#1!%
1300 }%
1301 \fi
1302 \else
1303 \ifx\#3\%
1304 \BitSet@AfterFiFi{#4 }%
1305 \else
1306 \BitSet@AfterFiFi{%
1307 \expandafter\expandafter\expandafter\BitSet@Size
1308 \IntCalcInc#1!!#3!#4!%
1309 }%
1310 \fi
1311 \fi
1312 \BitSet@Fi
1313 }

```

\bitsetCardinality

```

1314 \def\bitsetCardinality#1{%
1315 \number
1316 \BitSet@IfUndefined{#1}{0 }{%
1317 \expandafter\expandafter\expandafter\BitSet@Cardinality
1318 \expandafter\expandafter\expandafter0%
1319 \expandafter\expandafter\expandafter!%

```

```

1320     \csname BS@#1\endcsname!%
1321 }%
1322 }

```

```

\BitSet@Cardinality #1: result
#2#3: bits
1323 \def\BitSet@Cardinality#1!#2#3!{%
1324   \ifx#21%
1325     \ifx\#3\%
1326       \BitSet@AfterFiFi{\IntCalcInc#1! }%
1327     \else
1328       \BitSet@AfterFiFi{%
1329         \expandafter\expandafter\expandafter\BitSet@Cardinality
1330         \IntCalcInc#1!!#3!%
1331       }%
1332     \fi
1333   \else
1334     \ifx\#3\%
1335       \BitSet@AfterFiFi{#1 }%
1336     \else
1337       \BitSet@AfterFiFi{%
1338         \BitSet@Cardinality#1!#3!%
1339       }%
1340     \fi
1341   \fi
1342   \BitSet@Fi
1343 }

```

2.12 Queries

\bitsetIsDefined

```

1344 \def\bitsetIsDefined#1{%
1345   \BitSet@IfUndefined{#1}%
1346   \BitSet@SecondOfTwo
1347   \BitSet@FirstOfTwo
1348 }

```

\bitsetIsEmpty

```

1349 \def\bitsetIsEmpty#1{%
1350   \BitSet@IfUndefined{#1}\BitSet@FirstOfTwo{%
1351     \expandafter\ifx\csname BS@#1\endcsname\BitSet@Zero
1352     \expandafter\BitSet@FirstOfTwo
1353   \else
1354     \expandafter\BitSet@SecondOfTwo
1355   \fi
1356 }%
1357 }

```

\BitSet@Zero

```

1358 \def\BitSet@Zero{0}

```

\bitsetQuery

```

1359 \def\bitsetQuery#1#2{%
1360   \ifnum\bitsetGet{#1}{#2}=1 %
1361     \expandafter\BitSet@FirstOfTwo
1362   \else
1363     \expandafter\BitSet@SecondOfTwo

```

```

1364 \fi
1365 }

\bitsetEquals
1366 \def\bitsetEquals#1#2{%
1367 \BitSet@IfUndefined{#1}{%
1368 \BitSet@IfUndefined{#2}\BitSet@FirstOfTwo\BitSet@SecondOfTwo
1369 }{%
1370 \BitSet@IfUndefined{#2}\BitSet@SecondOfTwo{%
1371 \expandafter\ifx\csname BS@#1\expandafter\endcsname
1372 \csname BS@#2\endcsname
1373 \expandafter\BitSet@FirstOfTwo
1374 \else
1375 \expandafter\BitSet@SecondOfTwo
1376 \fi
1377 }%
1378 }%
1379 }

\bitsetIntersects
1380 \def\bitsetIntersects#1#2{%
1381 \bitsetIsEmpty{#1}\BitSet@SecondOfTwo{%
1382 \bitsetIsEmpty{#2}\BitSet@SecondOfTwo{%
1383 \expandafter\expandafter\expandafter\BitSet@Intersects
1384 \csname BS@#1\expandafter\expandafter\expandafter\endcsname
1385 \expandafter\expandafter\expandafter!%
1386 \csname BS@#2\endcsname!%
1387 }%
1388 }%
1389 }

\BitSet@Intersects
1390 \def\BitSet@Intersects#1#2!#3#4!{%
1391 \ifnum#1#3=11 %
1392 \BitSet@AfterFi\BitSet@FirstOfTwo
1393 \else
1394 \ifx\#2\%
1395 \BitSet@AfterFiFi\BitSet@SecondOfTwo
1396 \else
1397 \ifx\#4\%
1398 \BitSet@AfterFiFiFi\BitSet@SecondOfTwo
1399 \else
1400 \BitSet@AfterFiFiFi{%
1401 \BitSet@Intersects#2!#4!%
1402 }%
1403 \fi
1404 \fi
1405 \BitSet@Fi
1406 }

1407 \BitSet@AtEnd%
1408 \</package>

```


3 Installation

3.1 Download

Package. This package is available on CTAN¹:

[CTAN:macros/latex/contrib/oberdiek/bitset.dtx](#) The source file.

[CTAN:macros/latex/contrib/oberdiek/bitset.pdf](#) Documentation.

Bundle. All the packages of the bundle ‘oberdiek’ are also available in a TDS compliant ZIP archive. There the packages are already unpacked and the documentation files are generated. The files and directories obey the TDS standard.

[CTAN:install/macros/latex/contrib/oberdiek.tds.zip](#)

TDS refers to the standard “A Directory Structure for T_EX Files” ([CTAN:pkg/tds](#)). Directories with `texmf` in their name are usually organized this way.

3.2 Bundle installation

Unpacking. Unpack the `oberdiek.tds.zip` in the TDS tree (also known as `texmf` tree) of your choice. Example (linux):

```
unzip oberdiek.tds.zip -d ~/texmf
```

3.3 Package installation

Unpacking. The `.dtx` file is a self-extracting `docstrip` archive. The files are extracted by running the `.dtx` through plain T_EX:

```
tex bitset.dtx
```

TDS. Now the different files must be moved into the different directories in your installation TDS tree (also known as `texmf` tree):

```
bitset.sty → tex/generic/oberdiek/bitset.sty
bitset.pdf → doc/latex/oberdiek/bitset.pdf
bitset.dtx → source/latex/oberdiek/bitset.dtx
```

If you have a `docstrip.cfg` that configures and enables `docstrip`’s TDS installing feature, then some files can already be in the right place, see the documentation of `docstrip`.

3.4 Refresh file name databases

If your T_EX distribution (T_EX Live, mikT_EX, ...) relies on file name databases, you must refresh these. For example, T_EX Live users run `texhash` or `mktextlsr`.

3.5 Some details for the interested

Unpacking with L^AT_EX. The `.dtx` chooses its action depending on the format:

plain T_EX: Run `docstrip` and extract the files.

L^AT_EX: Generate the documentation.

¹[CTAN:pkg/bitset](#)

If you insist on using L^AT_EX for docstrip (really, docstrip does not need L^AT_EX), then inform the autodetect routine about your intention:

```
latex \let\install=y\input{bitset.dtx}
```

Do not forget to quote the argument according to the demands of your shell.

Generating the documentation. You can use both the .dtx or the .drv to generate the documentation. The process can be configured by the configuration file ltxdoc.cfg. For instance, put this line into this file, if you want to have A4 as paper format:

```
\PassOptionsToClass{a4paper}{article}
```

An example follows how to generate the documentation with pdfL^AT_EX:

```
pdflatex bitset.dtx
makeindex -s gind.ist bitset.idx
pdflatex bitset.dtx
makeindex -s gind.ist bitset.idx
pdflatex bitset.dtx
```

4 History

[2007/09/28 v1.0]

- First version.

[2011/01/30 v1.1]

- Already loaded package files are not input in plain T_EX.

[2016/05/16 v1.2]

- Documentation updates.

5 Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; plain numbers refer to the code lines where the entry is used.

| Symbols | | |
|----------------|------------------------------------|---|
| \@PackageError | 176, 368, 924, 1079, 1092 | \bigintcalcCmp 358 |
| \@ehc | 178, 370, 926, 1084, 1106 | \BigIntCalcOdd 378 |
| \@undefined | 58 | \bigintcalcSgn 355 |
| \\ | 222, 375, 438, | \BigIntCalcShl 660, 667 |
| | 571, 580, 642, 689, 692, 727, | \BigIntCalcShr 386 |
| | 730, 767, 770, 800, 802, 810, | \bitset 1094 |
| | 1294, 1303, 1325, 1334, 1394, 1397 | \BitSet@@@Range 1086, 1109, 1113 |
| | | \BitSet@@@Set 984, <u>991</u> , 1026 |
| | | \BitSet@@CheckIndex 169, <u>173</u> |
| | | \BitSet@@Clear 930, <u>942</u> |
| \aftergroup | 29 | \BitSet@@Flip 1001, <u>1013</u> |
| | | \BitSet@@Get 1130, <u>1139</u> |
| | | \BitSet@@GetBin 407, <u>410</u> |
| | | \BitSet@@GetDec 575, <u>579</u> , 605 |
| | | |
| A | | |
| | | |
| B | | |
| \BigIntCalcAdd | 646, 655 | |

| | | | |
|---|---|---|---|
| \BitSet@@GetDecBig | 654, <u>665</u> | \BitSet@FromOct | 273, <u>276</u> |
| \BitSet@@GetHex | 479, <u>512</u> | \BitSet@Get | 1120, <u>1123</u> |
| \BitSet@@GetOct | 465, <u>487</u> | \BitSet@GetDec | 566, <u>570</u> |
| \BitSet@@GetOctHex . | 462, 476, 546, <u>550</u> | \BitSet@GetDecBig | 639, <u>641</u> , <u>666</u> |
| \BitSet@@NextClearBit ... | 1173, <u>1177</u> | \BitSet@GetOctHex | 490, 515, <u>545</u> |
| \BitSet@@NextSetBit | 1209, <u>1213</u> | \BitSet@GetSetBitList ... | 1268, <u>1272</u> |
| \BitSet@@Range . | 1065, <u>1070</u> , 1107, <u>1109</u> | \BitSet@Gobble | |
| \BitSet@@Set | 970, <u>977</u> | | 144, 852, 877, 918, 919, 1243 |
| \BitSet@@TestMode | 124 | \BitSet@GobbleSeven | 1250, <u>1264</u> |
| \BitSet@AfterFi | | \BitSet@Hex[0..F] | <u>318</u> |
| .. | 157, 175, 181, 225, 384, 399, | \BitSet@Hex[0000..1111] | <u>526</u> |
| 413, 418, 429, 434, 439, 443, | | \BitSet@IfUndefined | |
| 451, 456, 489, 494, 514, 519, | | | 160, 168, 190, 411, 565, |
| 552, 560, 572, 574, 1112, 1125, | | | 750, 781, 829, 856, 1129, 1285, |
| 1164, 1200, 1231, 1274, 1276, 1392 | | | 1316, 1345, 1350, 1367, 1368, 1370 |
| \BitSet@AfterFiFi | | \BitSet@Intersects | 1383, <u>1390</u> |
| | 158, 263, 296, 582, 587, | \BitSet@Kill | 870, <u>880</u> |
| 591, 597, 644, 649, 653, 658, | | \BitSet@KillZeros | |
| 773, 891, 953, 984, 1026, 1150, | | | 204, <u>214</u> , 243, 301, 350 |
| 1183, 1185, 1219, 1221, 1236, | | \BitSet@MaxSize | <u>141</u> , 358 |
| 1240, 1242, 1295, 1297, 1304, | | \BitSet@N1073741824 | <u>638</u> |
| 1306, 1326, 1328, 1335, 1337, 1395 | | \BitSet@N[1,2,4,...] | <u>603</u> |
| \BitSet@AfterFiFiFi <u>159</u> , 697, 701, | | \BitSet@NegativeIndex <u>1072</u> , <u>1075</u> , <u>1091</u> | |
| 737, 741, 816, 821, 956, 961, | | \BitSet@NextClearBit | 1158, <u>1161</u> |
| 1029, 1034, 1247, 1253, 1398, 1400 | | \BitSet@NextSetBit | |
| \BitSet@And | 677, <u>688</u> | | 1194, <u>1197</u> , 1269, 1278 |
| \BitSet@AndNot | 715, <u>726</u> | \BitSet@NumBinFill | 440, <u>449</u> |
| \BitSet@AtEnd | 95, 96, 118, 1407 | \BitSet@NumBinRev | 421, <u>437</u> |
| \BitSet@Cardinality | 1317, <u>1323</u> | \BitSet@Oct[000..111] | <u>501</u> |
| \BitSet@CheckIndex | | \BitSet@Or | 757, <u>765</u> |
| | 167, 898, 901, 904, 911 | \BitSet@Range | |
| \BitSet@Cleanup | | | 1042, 1045, 1048, 1056, 1058, <u>1063</u> |
| .. | 891, 953, 1150, 1183, 1219, <u>1228</u> | \BitSet@Reverse | 210, <u>221</u> , 255 |
| \BitSet@Clear | | \BitSet@SecondOfTwo | <u>146</u> , |
| .. | 898, 913, <u>928</u> , 1042, 1056, 1095 | | 164, 1346, 1354, 1363, 1368, |
| \BitSet@Empty | <u>142</u> , 150, 201, | | 1370, 1375, 1381, 1382, 1395, 1398 |
| 204, 206, 240, 243, 245, 251, | | \BitSet@Set | |
| 347, 350, 352, 470, 484, 488, | | .. | 901, 915, <u>968</u> , 1045, 1058, 1098 |
| 513, 682, 720, 793, 871, 883, | | \BitSet@SetDec | 364, 376, <u>390</u> |
| 889, 932, 936, 944, 946, 952, | | \BitSet@SetDecBig | 360, <u>374</u> |
| 972, 982, 1003, 1007, 1016, 1024 | | \BitSet@SetOctHex | 231, 234, <u>236</u> |
| \BitSet@ErrorInvalidBitValue ... | | \BitSet@SetValue | 907, <u>910</u> |
| | 917, <u>923</u> , 1060 | \BitSet@SetValueRange ... | 1051, <u>1054</u> |
| \BitSet@Fi <u>156</u> , 157, 158, 159, 184, | | \BitSet@ShiftLeft | 834, <u>839</u> , 877 |
| 228, 274, 305, 388, 403, 419, | | \BitSet@ShiftRight | 852, 861, <u>866</u> |
| 435, 447, 457, 499, 524, 561, | | \BitSet@Size | 1286, <u>1292</u> |
| 577, 601, 663, 706, 746, 777, | | \BitSet@Skip | 1170, 1206, <u>1229</u> |
| 826, 895, 966, 989, 1039, 1116, | | \BitSet@SkipContinue | |
| 1137, 1154, 1175, 1190, 1211, | | | 1232, 1237, 1240, 1243, <u>1261</u> |
| 1226, 1259, 1281, 1312, 1342, 1405 | | \BitSet@Space | <u>147</u> , 201, 240, |
| \BitSet@Fill | 414, <u>427</u> , 452, 554 | | 347, 583, 645, 847, 1136, 1165, 1201 |
| \BitSet@FirstOfOne | <u>143</u> | \BitSet@Temp | 198, |
| \BitSet@FirstOfTwo <u>145</u> , 162, 1347, | | | 199, 201, 203, 204, 206, 210, |
| 1350, 1352, 1361, 1368, 1373, 1392 | | | 237, 238, 240, 242, 243, 245, |
| \BitSet@Flip | 904, <u>999</u> , 1048 | | 248, 249, 251, 255, 318, 321, |
| \BitSet@FromFirstHex | 234, <u>292</u> | | 322, 323, 324, 325, 326, 327, |
| \BitSet@FromFirstOct | 231, <u>260</u> | | 328, 329, 330, 331, 332, 333, |
| \BitSet@FromHex | 304, <u>307</u> | | 334, 335, 336, 337, 338, 339, |

| | |
|---|--|
| 340, 341, 342, 344, 345, 347, 349, 350, 352, 355, 358, 360, 364, 501, 504, 505, 506, 507, 508, 509, 510, 511, 526, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 603, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 929, 936, 939, 1000, 1007, 1010, 1064, 1068 | \bitsetSize 9, <u>1283</u> \bitsetXor 7, <u>779</u> |
| C | |
| \catcode 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 69, 70, 72, 73, 74, 78, 79, 80, 81, 82, 83, 84, 87, 88, 90, 91, 92, 93, 97, 99, 122 | |
| \csname 14, 21, 50, 66, 76, 120, 126, 129, 161, 187, 193, 194, 207, 209, 246, 252, 254, 298, 302, 310, 313, 319, 353, 356, 359, 363, 424, 469, 483, 497, 502, 522, 527, 567, 592, 598, 604, 638, 676, 678, 680, 682, 714, 716, 718, 720, 753, 754, 756, 758, 760, 784, 785, 787, 789, 791, 793, 845, 848, 869, 871, 931, 935, 969, 971, 1002, 1006, 1131, 1172, 1208, 1289, 1320, 1351, 1371, 1372, 1384, 1386 | |
| E | |
| \empty 17, 18 \endcsname . 14, 21, 50, 66, 76, 120, 126, 129, 161, 187, 193, 194, 207, 209, 246, 252, 254, 298, 302, 310, 313, 319, 353, 356, 359, 363, 424, 469, 483, 497, 502, 522, 527, 567, 593, 598, 604, 638, 676, 678, 680, 682, 714, 716, 718, 720, 753, 754, 756, 758, 760, 784, 785, 787, 789, 791, 793, 845, 848, 869, 871, 931, 935, 969, 971, 1002, 1006, 1131, 1172, 1208, 1289, 1320, 1351, 1371, 1372, 1384, 1386 | |
| \endinput 29, 118 \endlinechar 4, 35, 71, 77, 89 | |
| I | |
| \ifcase 263, 279, 355, 378, 391, 840, 867, 912, 1055, 1077, 1235 \ifnum . 174, 358, 412, 428, 450, 551, 690, 693, 695, 728, 732, 735, 766, 1071, 1074, 1110, 1124, 1162, 1198, 1246, 1273, 1360, 1391 \ifodd 394 \ifx . 15, 18, 21, 50, 58, 61, 120, 126, 129, 150, 161, 206, 215, 222, 245, 251, 261, 277, 293, 295, 298, 308, 310, 352, 375, 438, 488, 513, 571, 580, 581, 590, 642, 643, 652, 682, 689, 692, 720, 727, 730, 767, 770, 793, 800, 801, 802, 810, 812, 815, 882, 883, 889, 936, 943, 944, | |
| \BitSet@TestMode 124 \BitSet@Xor 788, <u>799</u> \BitSet@ZapSpace .. <u>148</u> , 200, 239, 346 \BitSet@Zero 207, 246, 252, 353, 356, 937, 1008, 1351, <u>1358</u> \bitsetAnd 7, <u>669</u> \bitsetAndNot 7, <u>708</u> \bitsetCardinality 9, <u>1314</u> \bitsetClear 8, <u>897</u> \bitsetClearRange <u>1041</u> \bitsetEquals 9, <u>1366</u> \BitSetError 271, 287, 299, 311, 382, 1126, 1166, 1202 \bitsetFlip <u>903</u> \bitsetFlipRange <u>1047</u> \bitsetGet 8, <u>1118</u> , 1360 \bitsetGetBin 7, <u>405</u> \bitsetGetDec 7, <u>563</u> \bitsetGetHex <u>473</u> \bitsetGetOct <u>459</u> \bitsetGetSetBitList 8, <u>1265</u> \bitsetIntersects 9, <u>1380</u> \bitsetIsDefined 9, <u>1344</u> \bitsetIsEmpty 9, 461, 475, 670, 673, 709, 712, 749, 752, 780, 783, 832, 859, 1169, 1205, 1267, <u>1349</u> , 1381, 1382 \bitsetLet 6, <u>189</u> \bitsetNextClearBit 8, <u>1156</u> \bitsetNextSetBit 8, <u>1192</u> \bitsetOr 7, <u>748</u> \bitsetQuery 9, <u>1359</u> \bitsetReset 6, 168, <u>186</u> , 191, 671, 674, 683, 710, 721, 750, 781, 794, 830, 857 \bitsetSet <u>900</u> \bitsetSetBin 6, <u>197</u> \bitsetSetDec 6, <u>343</u> \bitsetSetHex <u>233</u> \bitsetSetOct <u>230</u> \bitsetSetRange <u>1044</u> \bitsetSetValue 8, <u>906</u> \bitsetSetValueRange <u>1050</u> \bitsetShiftLeft 7, <u>828</u> \bitsetShiftRight <u>855</u> | |

| | |
|---|---|
| 946, 952, 955, 979, 982, 992, 1007, 1014, 1015, 1016, 1024, 1028, 1095, 1098, 1141, 1142, 1148, 1178, 1181, 1214, 1217, 1230, 1293, 1294, 1303, 1324, 1325, 1334, 1351, 1371, 1394, 1397 | 1207, 1249, 1269, 1277, 1284, 1315 |
| P | |
| \immediate | 23, 52 |
| \input | 130 |
| \IntCalcAdd | 556, 584, 594 |
| \intcalcCmp | 1077 |
| \IntCalcDec | 415, 431, 491, 516, 1255 |
| \IntCalcDiv | 555 |
| \IntCalcInc | 445, 496, 521, 1114, 1187, 1223, 1279, 1299, 1308, 1326, 1330 |
| \IntCalcMul | 548 |
| \intcalcNum | 170, 408, 463, 477, 547, 835, 862, 908, 1052, 1066, 1121, 1134, 1159, 1195 |
| \intcalcSgn | 840, 867 |
| \IntCalcShr | 401 |
| \IntCalcSub | 453, 557, 1249 |
| M | |
| \MessageBreak | |
| | 1080, 1081, 1082, 1093, 1104 |
| N | |
| \number | 496, 521, 547, 555, 1119, 1157, 1171, 1193, |
| | 1207, 1249, 1269, 1277, 1284, 1315 |
| R | |
| \RequirePackage | 137, 138, 139 |
| \romannumeral | 406, 460, 474, 564, 847, 874, 933, 974, 1004, 1134, 1166, 1202, 1266 |
| T | |
| \the | 77, 78, 79, 80, 81, 82, 83, 84, 97 |
| \TMP@EnsureCode | |
| | 94, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117 |
| \TMP@RequirePackage | 127, 133, 134, 135 |
| U | |
| \uccode | 843 |
| \uppercase | 844 |
| W | |
| \write | 23, 52 |
| X | |
| \x | 14, 15, 18, 22, 26, 28, 51, 56, 66, 75, 87 |