The **bitset** package

Heiko Oberdiek[*]
<heiko.oberdiek at googlemail.com>

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

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

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 snipsets 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 enoded 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.

Expandibility: 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 ε-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)

`\bitsetReset`	⟨*BitSet*⟩
`\bitsetLet`	⟨*BitSet A*⟩ ⟨*BitSet B*⟩

Import (section 1.6.2)

`\bitsetSetBin`, `\bitsetSetOct`, `\bitsetSetHex`	⟨*BitSet*⟩ ⟨*Value*⟩
`\bitsetSetDec`	⟨*BitSet*⟩ ⟨*Value*⟩

Export[a] (section 1.6.3)

`\bitsetGetBin`, `\bitsetGetOct`, `\bitsetGetHex`	⟨*BitSet*⟩ ⟨*MinSize*⟩
`\bitsetGetDec`	⟨*BitSet*⟩

Logical operators (section 1.6.4)

`\bitsetAnd`, `\bitsetAndNot`	⟨*BitSet A*⟩ ⟨*BitSet B*⟩
`\bitsetOr`, `\bitsetXor`	⟨*BitSet A*⟩ ⟨*BitSet B*⟩

Shifting (section 1.6.5)

`\bitsetShiftLeft`, `\bitsetShiftRight`	⟨*BitSet*⟩ ⟨*ShiftAmount*⟩

Bit manipulation (section 1.6.6)

`\bitsetClear`, `\bitsetSet`, `\bitsetFlip`	⟨*BitSet*⟩ ⟨*Index*⟩
`\bitsetSetValue`	⟨*BitSet*⟩ ⟨*Index*⟩ ⟨*Value*⟩
`\bitsetClearRange`, `\bitsetSetRange`, `\bitsetFlipRange`	⟨*BitSet*⟩ ⟨*IndexFrom*⟩ ⟨*IndexTo*⟩
`\bitsetSetValueRange`	⟨*BitSet*⟩ ⟨*IndexFrom*⟩ ⟨*IndexTo*⟩

Bit retrieval[a] (section 1.6.7)

`\bitsetGet`	⟨*BitSet*⟩ ⟨*Index*⟩
`\bitsetNextClearBit`, `\bitsetNextSetBit`	⟨*BitSet*⟩ ⟨*Index*⟩
`\bitsetGetSetBitList`	⟨*BitSet*⟩

Bit set properties (section 1.6.8)

`\bitsetSize`, `\bitsetCardinality`	⟨*BitSet*⟩

Queries[b] (section 1.6.9)

`\bitsetIsDefined`, `\bitsetIsEmpty`	⟨*BitSet*⟩ ⟨*Then*⟩ ⟨*Else*⟩
`\bitsetEquals`, `\bitsetIntersects`	⟨*BitSet A*⟩ ⟨*BitSet B*⟩ ⟨*Then*⟩ ⟨*Else*⟩
`\bitsetQuery`	⟨*BitSet*⟩ ⟨*Index*⟩ ⟨*Then*⟩ ⟨*Else*⟩

[a] Macros are expandable, full expansion by two steps.
[b] Macros are expandable.

1.5 Package loading

The package can be used as normal LaTeX package:

```
\usepackage{bitset}
```

Also plain TeX is supported:

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

1.6 Operators

The following macros work on and with bit sets. A bit set ⟨*BitSet*⟩ 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 LaTeX.

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. There 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 TeX 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 TeX'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 explicitely. This also helps for unexpandable number command tokens or registers (`\z@`, `\@ne`, `\count@`, . . .). Also LaTeX' `\value` needs prefixing:

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

1.6.3 Export

```
\bitsetGetBin {⟨BitSet⟩} {⟨MinSize⟩}
\bitsetGetOct {⟨BitSet⟩} {⟨MinSize⟩}
\bitsetGetHex {⟨BitSet⟩} {⟨MinSize⟩}
```

These macros returns the bit set as binary, octal or hexadecimal number. If the bit size is smaller than ⟨*MinSize*⟩ 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).

```
\bitsetGetDec {⟨BitSet⟩}
```

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

1.6.4 Logical operators

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

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

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

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

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

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

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

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

1.6.5 Shifting

```
\bitsetShiftLeft {⟨BitSet⟩} {⟨ShiftAmount⟩}
\bitsetShiftRight {⟨BitSet⟩} {⟨ShiftAmount⟩}
```

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, movint the bits to lower positions. The number will become smaller, the lowest bits are lost.

If the ⟨*ShiftAmount*⟩ is negative, it reverts the meaning of the shift operation. A left shift becomes a right shift. A ⟨*ShiftAmount*⟩ of zero is ignored.

1.6.6 Bit manipulation

`\bitsetClear` {⟨*BitSet*⟩} {⟨*Index*⟩}
`\bitsetSet` {⟨*BitSet*⟩} {⟨*Index*⟩}
`\bitsetFlip` {⟨*BitSet*⟩} {⟨*Index*⟩}

This macros manipulate a single bit in ⟨*BitSet*⟩ addressed by `\Index`. Macro `\bitsetClear` disables the bit, `\bitsetSet` enables it and `\bitsetFlip` reverts the current setting of the bit.

`\bitsetSetValue` {⟨*BitSet*⟩} {⟨*Index*⟩} {⟨*Bit*⟩}

Macro `\bitsetSetValue` puts bit ⟨*Bit*⟩ at position ⟨*Index*⟩ in bit set ⟨*BitSet*⟩. ⟨*Bit*⟩ must be a valid TeX number equals to zero (disabled/cleared) or one (enabled/set).

1.6.7 Bit retrieval

`\bitsetGet` {⟨*BitSet*⟩} {⟨*Index*⟩}

Macro `\bitsetGet` extracts the status of the bit at position ⟨*Index*⟩ in bit set ⟨*BitSet*⟩. 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`.

`\bitsetNextClearBit` {⟨*BitSet*⟩} {⟨*Index*⟩}

Starting at position ⟨*Index*⟩ (inclusive) the bits are inspected. The first position without a set bit is returned. Possible results are decimal numbers: ⟨*Index*⟩, ⟨*Index*⟩ + 1, . . . , (∞)

`\bitsetNextSetBit` {⟨*BitSet*⟩} {⟨*Index*⟩}

Starting at position ⟨*Index*⟩ (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`, ⟨*Index*⟩, ⟨*Index*⟩ + 1, . . . , (∞)

`\bitsetGetSetBitList` {⟨*BitSet*⟩}

Macro `\bitsetGetSetBitList` is an application for `\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, symmeric, 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
```

```
⟨*package⟩
```

2.1 Reload check and package identification

Reload check, especially if the package is not used with LaTeX.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5 % ^^M
  \endlinechar=13 %
  \catcode35=6 % #
  \catcode39=12 % '
  \catcode44=12 % ,
  \catcode45=12 % -
  \catcode46=12 % .
  \catcode58=12 % :
  \catcode64=11 % @
  \catcode123=1 % {
  \catcode125=2 % }
  \expandafter\let\expandafter\x\csname ver@bitset.sty\endcsname
  \ifx\x\relax % plain-TeX, first loading
  \else
    \def\empty{}%
    \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
    \else
      \expandafter\ifx\csname PackageInfo\endcsname\relax
        \def\x#1#2{%
          \immediate\write-1{Package #1 Info: #2.}%
        }%
      \else
        \def\x#1#2{\PackageInfo{#1}{#2, stopped}}%
      \fi
      \x{bitset}{The package is already loaded}%
      \aftergroup\endinput
    \fi
  \fi
\endgroup%
```

Package identification:

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5 % ^^M
  \endlinechar=13 %
  \catcode35=6 % #
  \catcode39=12 % '
  \catcode40=12 % (
  \catcode41=12 % )
  \catcode44=12 % ,
  \catcode45=12 % -
  \catcode46=12 % .
  \catcode47=12 % /
  \catcode58=12 % :
  \catcode64=11 % @
  \catcode91=12 % [
  \catcode93=12 % ]
  \catcode123=1 % {
  \catcode125=2 % }
  \expandafter\ifx\csname ProvidesPackage\endcsname\relax
    \def\x#1#2#3[#4]{\endgroup
      \immediate\write-1{Package: #3 #4}%
      \xdef#1{#4}%
    }%
  \else
    \def\x#1#2[#3]{\endgroup
      #2[{#3}]%
      \ifx#1\@undefined
        \xdef#1{#3}%
      \fi
      \ifx#1\relax
        \xdef#1{#3}%
      \fi
    }%
  \fi
\expandafter\x\csname ver@bitset.sty\endcsname
\ProvidesPackage{bitset}%
  [2016/05/16 v1.2 Handle bit-vector datatype (HO)]%
```

2.2 Catcodes

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5 % ^^M
  \endlinechar=13 %
  \catcode123=1 % {
  \catcode125=2 % }
  \catcode64=11 % @
  \def\x{\endgroup
    \expandafter\edef\csname BitSet@AtEnd\endcsname{%
      \endlinechar=\the\endlinechar\relax
      \catcode13=\the\catcode13\relax
      \catcode32=\the\catcode32\relax
      \catcode35=\the\catcode35\relax
      \catcode61=\the\catcode61\relax
      \catcode64=\the\catcode64\relax
      \catcode123=\the\catcode123\relax
      \catcode125=\the\catcode125\relax
    }%
  }%
\x\catcode61\catcode48\catcode32=10\relax%
\catcode13=5 % ^^M
\endlinechar=13 %
\catcode35=6 % #
\catcode64=11 % @
\catcode123=1 % {
\catcode125=2 % }
\def\TMP@EnsureCode#1#2{%
  \edef\BitSet@AtEnd{%
    \BitSet@AtEnd
    \catcode#1=\the\catcode#1\relax
  }%
  \catcode#1=#2\relax
}
\TMP@EnsureCode{33}{12}% !
\TMP@EnsureCode{39}{12}% '
\TMP@EnsureCode{40}{12}% (
\TMP@EnsureCode{41}{12}% )
\TMP@EnsureCode{42}{12}% *
\TMP@EnsureCode{43}{12}% +
\TMP@EnsureCode{44}{12}% ,
\TMP@EnsureCode{45}{12}% -
\TMP@EnsureCode{46}{12}% .
\TMP@EnsureCode{47}{12}% /
\TMP@EnsureCode{58}{11}% : (letter!)
\TMP@EnsureCode{60}{12}% <
\TMP@EnsureCode{62}{12}% >
\TMP@EnsureCode{63}{14}% ? (comment!)
\TMP@EnsureCode{91}{12}% [
\TMP@EnsureCode{93}{12}% ]
\TMP@EnsureCode{96}{12}% `
\edef\BitSet@AtEnd{\BitSet@AtEnd\noexpand\endinput}
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname BitSet@TestMode\endcsname\relax
\else
  \catcode63=9 % ? (ignore)
\fi
? \let\BitSet@@TestMode\BitSet@TestMode
```

2.3 Package loading

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \def\TMP@RequirePackage#1[#2]{%
    \begingroup\expandafter\expandafter\expandafter\endgroup
    \expandafter\ifx\csname ver@#1.sty\endcsname\relax
      \input #1.sty\relax
    \fi
  }%
  \TMP@RequirePackage{infwarerr}[2007/09/09]%
  \TMP@RequirePackage{intcalc}[2007/09/27]%
  \TMP@RequirePackage{bigintcalc}[2007/09/27]%
\else
  \RequirePackage{infwarerr}[2007/09/09]%
  \RequirePackage{intcalc}[2007/09/27]%
  \RequirePackage{bigintcalc}[2007/09/27]%
\fi
```

2.4 Help macros

2.4.1 Number constant

\BitSet@MaxSize

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

2.4.2 General basic macros

\BitSet@Empty

```
\def\BitSet@Empty{}
```

\BitSet@FirstOfOne

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

\BitSet@Gobble

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

\BitSet@FirstOfTwo

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

\BitSet@SecondOfTwo

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

\BitSet@Space

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

\BitSet@ZapSpace

```
\def\BitSet@ZapSpace#1 #2{%
  #1%
  \ifx\BitSet@Empty#2%
  \else
    \expandafter\BitSet@ZapSpace
  \fi
  #2%
}
```

2.4.3 Tail recursion

\BitSet@Fi

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

\BitSet@AfterFi

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

\BitSet@AfterFiFi

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

\BitSet@AfterFiFiFi

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

2.4.4 Check macros

\BitSet@IfUndefined

```
\def\BitSet@IfUndefined#1{%
  \expandafter\ifx\csname BS@#1\endcsname\relax
    \expandafter\BitSet@FirstOfTwo
  \else
    \expandafter\BitSet@SecondOfTwo
  \fi
}
```

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

```
\def\BitSet@CheckIndex#1#2#3{%
  \BitSet@IfUndefined{#2}{\bitsetReset{#2}}{}%
  \expandafter\expandafter\expandafter\BitSet@@CheckIndex
  \intcalcNum{#3}!%
  {#2}{#1}%
}
```

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

```
\def\BitSet@@CheckIndex#1!#2#3{%
  \ifnum#1<0 %
    \BitSet@AfterFi{%
      \@PackageError{bitset}{%
        Invalid negative index (#1)%
      }\@ehc
    }%
  \else
    \BitSet@AfterFi{%
      #3{#2}{#1}%
    }%
  \BitSet@Fi
}
```

2.5 Miscellaneous

\bitsetReset

```
\def\bitsetReset#1{%
  \expandafter\def\csname BS@#1\endcsname{0}%
}
```

\bitsetLet

```
\def\bitsetLet#1#2{%
  \BitSet@IfUndefined{#2}{%
    \bitsetReset{#1}%
  }{%
    \expandafter\let\csname BS@#1\expandafter\endcsname
                    \csname BS@#2\endcsname
  }%
}
```

2.6 Import

2.6.1 From binary number

\bitsetSetBin

```
\def\bitsetSetBin#1#2{%
  \edef\BitSet@Temp{#2}%
  \edef\BitSet@Temp{%
    \expandafter\expandafter\expandafter\BitSet@ZapSpace
    \expandafter\BitSet@Temp\BitSet@Space\BitSet@Empty
  }%
  \edef\BitSet@Temp{%
    \expandafter\BitSet@KillZeros\BitSet@Temp\BitSet@Empty
  }%
  \ifx\BitSet@Temp\BitSet@Empty
    \expandafter\let\csname BS@#1\endcsname\BitSet@Zero
  \else
    \expandafter\edef\csname BS@#1\endcsname{%
      \expandafter\BitSet@Reverse\BitSet@Temp!%
    }%
  \fi
}
```

\BitSet@KillZeros

```
\def\BitSet@KillZeros#1{%
  \ifx#10%
    \expandafter\BitSet@KillZeros
  \else
    #1%
  \fi
}
```

\BitSet@Reverse

```
\def\BitSet@Reverse#1#2!{%
  \ifx\\#2\\%
    #1%
  \else
    \BitSet@AfterFi{%
      \BitSet@Reverse#2!#1%
    }%
  \BitSet@Fi
}
```

2.6.2 From octal/hex number

\bitsetSetOct

```
\def\bitsetSetOct{%
  \BitSet@SetOctHex\BitSet@FromFirstOct
}
```

\bitsetSetHex

```
\def\bitsetSetHex{%
  \BitSet@SetOctHex\BitSet@FromFirstHex
}
```

\BitSet@SetOctHex

```
\def\BitSet@SetOctHex#1#2#3{%
  \edef\BitSet@Temp{#3}%
  \edef\BitSet@Temp{%
    \expandafter\expandafter\expandafter\BitSet@ZapSpace
    \expandafter\BitSet@Temp\BitSet@Space\BitSet@Empty
  }%
  \edef\BitSet@Temp{%
    \expandafter\BitSet@KillZeros\BitSet@Temp\BitSet@Empty
  }%
  \ifx\BitSet@Temp\BitSet@Empty
    \expandafter\let\csname BS@#2\endcsname\BitSet@Zero
  \else
    \edef\BitSet@Temp{%
      \expandafter#1\BitSet@Temp!%
    }%
    \ifx\BitSet@Temp\BitSet@Empty
      \expandafter\let\csname BS@#2\endcsname\BitSet@Zero
    \else
      \expandafter\edef\csname BS@#2\endcsname{%
        \expandafter\BitSet@Reverse\BitSet@Temp!%
      }%
    \fi
  \fi
}
```

\BitSet@FromFirstOct

```
\def\BitSet@FromFirstOct#1{%
  \ifx#1!%
  \else
    \ifcase#1 \BitSet@AfterFiFi\BitSet@FromFirstOct
    \or 1%
    \or 10%
    \or 11%
    \or 100%
    \or 101%
    \or 110%
    \or 111%
    \else \BitSetError:WrongOctalDigit%
    \fi
    \expandafter\BitSet@FromOct
  \BitSet@Fi
}
```

\BitSet@FromOct

```
\def\BitSet@FromOct#1{%
  \ifx#1!%
  \else
    \ifcase#1 000%
    \or 001%
    \or 010%
    \or 011%
    \or 100%
    \or 101%
    \or 110%
    \or 111%
    \else \BitSetError:WrongOctalDigit%
    \fi
    \expandafter\BitSet@FromOct
  \fi
}
```

\BitSet@FromFirstHex

```
\def\BitSet@FromFirstHex#1{%
  \ifx#1!%
  \else
    \ifx#10%
      \BitSet@AfterFiFi\BitSet@FromFirstHex
    \fi
    \expandafter\ifx\csname BitSet@Hex#1\endcsname\relax
      \BitSetError:InvalidHexDigit%
    \else
      \expandafter\expandafter\expandafter\BitSet@KillZeros
      \csname BitSet@Hex#1\endcsname
    \fi
    \expandafter\BitSet@FromHex
  \BitSet@Fi
}
```

\BitSet@FromHex

```
\def\BitSet@FromHex#1{%
  \ifx#1!%
  \else
    \expandafter\ifx\csname BitSet@Hex#1\endcsname\relax
      \BitSetError:InvalidHexDigit%
    \else
      \csname BitSet@Hex#1\endcsname
    \fi
    \expandafter\BitSet@FromHex
  \fi
}
```

\BitSet@Hex[0..F]

```
\def\BitSet@Temp#1{%
  \expandafter\def\csname BitSet@Hex#1\endcsname
}
\BitSet@Temp 0{0000}%
\BitSet@Temp 1{0001}%
\BitSet@Temp 2{0010}%
\BitSet@Temp 3{0011}%
\BitSet@Temp 4{0100}%
\BitSet@Temp 5{0101}%
\BitSet@Temp 6{0110}%
\BitSet@Temp 7{0111}%
\BitSet@Temp 8{1000}%
\BitSet@Temp 9{1001}%
\BitSet@Temp A{1010}%
\BitSet@Temp B{1011}%
\BitSet@Temp C{1100}%
\BitSet@Temp D{1101}%
\BitSet@Temp E{1110}%
\BitSet@Temp F{1111}%
\BitSet@Temp a{1010}%
\BitSet@Temp b{1011}%
\BitSet@Temp c{1100}%
\BitSet@Temp d{1101}%
\BitSet@Temp e{1110}%
\BitSet@Temp f{1111}%
```

2.6.3 From decimal number

\bitsetSetDec

```
\def\bitsetSetDec#1#2{%
  \edef\BitSet@Temp{#2}%
  \edef\BitSet@Temp{%
    \expandafter\expandafter\expandafter\BitSet@ZapSpace
    \expandafter\BitSet@Temp\BitSet@Space\BitSet@Empty
  }%
  \edef\BitSet@Temp{%
    \expandafter\BitSet@KillZeros\BitSet@Temp\BitSet@Empty
  }%
  \ifx\BitSet@Temp\BitSet@Empty
    \expandafter\let\csname BS@#1\endcsname\BitSet@Zero
  \else
    \ifcase\bigintcalcSgn{\BitSet@Temp} %
      \expandafter\let\csname BS@#1\endcsname\BitSet@Zero
    \or
      \ifnum\bigintcalcCmp\BitSet@Temp\BitSet@MaxSize>0 %
        \expandafter\edef\csname BS@#1\endcsname{%
          \expandafter\BitSet@SetDecBig\BitSet@Temp!%
        }%
      \else
        \expandafter\edef\csname BS@#1\endcsname{%
          \expandafter\BitSet@SetDec\BitSet@Temp!%
        }%
      \fi
    \else
      \@PackageError{bitset}{%
        Bit sets cannot be negative%
      }\@ehc
    \fi
  \fi
}
```

\BitSet@SetDecBig

```
\def\BitSet@SetDecBig#1#2#3#4#5#6#7#8#9!{%
  \ifx\\#9\\%
    \BitSet@SetDec#1#2#3#4#5#6#7#8!%
  \else
    \ifcase\BigIntCalcOdd#1#2#4#5#6#7#8#9! %
      0%
    \or
      1%
?   \else\BitSetError:ThisCannotHappen%
    \fi
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@SetDecBig
      \BigIntCalcShr#1#2#3#4#5#6#7#8#9!!%
    }%
  \BitSet@Fi
}
```

\BitSet@SetDec

```
\def\BitSet@SetDec#1!{%
  \ifcase#1 %
  \or 1%
  \else
    \ifodd#1 %
      1%
    \else
      0%
    \fi
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@SetDec
      \IntCalcShr#1!!%
    }%
  \BitSet@Fi
}
```

2.7 Export

2.7.1 To binary number

\bitsetGetBin

```
\def\bitsetGetBin#1#2{%
  \romannumeral0%
  \expandafter\expandafter\expandafter\BitSet@@GetBin
  \intcalcNum{#2}!{#1}%
}
```

\BitSet@@GetBin

```
\def\BitSet@@GetBin#1!#2{%
  \BitSet@IfUndefined{#2}{%
    \ifnum#1>1 %
      \BitSet@AfterFi{%
        \expandafter\expandafter\expandafter\BitSet@Fill
        \IntCalcDec#1!!0%
      }%
    \else
      \BitSet@AfterFi{ 0}%
    \BitSet@Fi
  }{%
    \expandafter\expandafter\expandafter\BitSet@NumBinRev
    \expandafter\expandafter\expandafter1%
    \expandafter\expandafter\expandafter!%
    \csname BS@#2\endcsname!!#1!%
  }%
}
```

\BitSet@Fill

#1: number of leading digits 0
#2: result

```
\def\BitSet@Fill#1!{%
  \ifnum#1>0 %
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@Fill
      \IntCalcDec#1!!0%
    }%
  \else
    \BitSet@AfterFi{ }%
  \BitSet@Fi
}
```

\BitSet@NumBinRev

#1: bit counter (including #2)
#2#3: reverted number
#4: result
#5: min size

```
\def\BitSet@NumBinRev#1!#2#3!{%
  \ifx\\#3\\%
    \BitSet@AfterFi{%
      \BitSet@NumBinFill#1!#2%
    }%
  \else
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@NumBinRev
      \IntCalcInc#1!!#3!#2%
    }%
  \BitSet@Fi
}
```

\BitSet@NumBinFill

```
\def\BitSet@NumBinFill#1!#2!#3!{%
  \ifnum#3>#1 %
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@Fill
      \IntCalcSub#3!#1!!#2%
    }%
  \else
    \BitSet@AfterFi{ #2}%
  \BitSet@Fi
}
```

2.7.2 To octal/hexadecimal number

\bitsetGetOct

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

```
  \romannumeral0%
  \bitsetIsEmpty{#1}{%
    \expandafter\expandafter\expandafter\BitSet@@GetOctHex
    \intcalcNum{#2}!3!230%
  }{%
    \expandafter\expandafter\expandafter\BitSet@@GetOct
    \expandafter\expandafter\expandafter1%
    \expandafter\expandafter\expandafter!%
    \expandafter\expandafter\expandafter!%
    \csname BS@#1\endcsname00%
    \BitSet@Empty\BitSet@Empty\BitSet@Empty!{#2}%
  }%
}
```

`\bitsetGetHex`

```
\def\bitsetGetHex#1#2{%
  \romannumeral0%
  \bitsetIsEmpty{#1}{%
    \expandafter\expandafter\expandafter\BitSet@@GetOctHex
    \intcalcNum{#2}!4!340%
  }{%
    \expandafter\expandafter\expandafter\BitSet@@GetHex
    \expandafter\expandafter\expandafter1%
    \expandafter\expandafter\expandafter!%
    \expandafter\expandafter\expandafter!%
    \csname BS@#1\endcsname000%
    \BitSet@Empty\BitSet@Empty\BitSet@Empty\BitSet@Empty!{#2}%
  }%
}
```

`\BitSet@@GetOct` #1: number of digits
#2: result
#3#4#5: bits

```
\def\BitSet@@GetOct#1!#2!#3#4#5{%
  \ifx#5\BitSet@Empty
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@GetOctHex
      \IntCalcDec#1!!#2!23%
    }%
  \else
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@@GetOct
      \number\IntCalcInc#1!\expandafter\expandafter\expandafter!%
      \csname BitSet@Oct#5#4#3\endcsname#2!%
    }%
  \BitSet@Fi
}
```

`\BitSet@Oct[000..111]`

```
\def\BitSet@Temp#1#2#3#4{%
  \expandafter\def\csname BitSet@Oct#1#2#3\endcsname{#4}%
}
\BitSet@Temp0000%
\BitSet@Temp0011%
\BitSet@Temp0102%
\BitSet@Temp0113%
\BitSet@Temp1004%
\BitSet@Temp1015%
```

```
\BitSet@Temp1106%
\BitSet@Temp1117%
```

`\BitSet@@GetHex`

#1: number of digits
#2: result
#3#4#5#6: bits

```
\def\BitSet@@GetHex#1!#2!#3#4#5#6{%
  \ifx#6\BitSet@Empty
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@GetOctHex
      \IntCalcDec#1!!#2!34%
    }%
  \else
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@@GetHex
      \number\IntCalcInc#1!\expandafter\expandafter\expandafter!%
      \csname BitSet@Hex#6#5#4#3\endcsname#2!%
    }%
  \BitSet@Fi
}
```

`\BitSet@Hex[0000..1111]`

```
\def\BitSet@Temp#1#2#3#4#5{%
  \expandafter\def\csname BitSet@Hex#1#2#3#4\endcsname{#5}%
}
\BitSet@Temp00000%
\BitSet@Temp00011%
\BitSet@Temp00102%
\BitSet@Temp00113%
\BitSet@Temp01004%
\BitSet@Temp01015%
\BitSet@Temp01106%
\BitSet@Temp01117%
\BitSet@Temp10008%
\BitSet@Temp10019%
\BitSet@Temp1010A%
\BitSet@Temp1011B%
\BitSet@Temp1100C%
\BitSet@Temp1101D%
\BitSet@Temp1110E%
\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

```
\def\BitSet@GetOctHex#1!#2!#3#4#5!#6{%
  \expandafter\BitSet@@GetOctHex
  \number\intcalcNum{#6}\expandafter\expandafter\expandafter!%
  \IntCalcMul#1!#4!!#3#4#2%
}
```

`\BitSet@@GetOctHex`

#1: plain min size
#2: digits * (bits per digit)

#3: bits per digit - 1
#4: bits per digit

```
\def\BitSet@@GetOctHex#1!#2!#3#4{%
  \ifnum#1>#2 %
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\BitSet@Fill
      \expandafter\IntCalcDiv\number
      \expandafter\expandafter\expandafter\IntCalcAdd
      \IntCalcSub#1!#2!!#3!!#4!!%
    }%
  \else
    \BitSet@AfterFi{ }%
  \BitSet@Fi
}
```

2.7.3 To decimal number

\bitsetGetDec

```
\def\bitsetGetDec#1{%
  \romannumeral0%
  \BitSet@IfUndefined{#1}{ 0}{%
    \expandafter\expandafter\expandafter\BitSet@GetDec
    \csname BS@#1\endcsname!%
  }%
}
```

\BitSet@GetDec

```
\def\BitSet@GetDec#1#2!{%
  \ifx\\#2\\%
    \BitSet@AfterFi{ #1}%
  \else
    \BitSet@AfterFi{%
      \BitSet@@GetDec2!#1!#2!%
    }%
  \BitSet@Fi
}
```

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

```
\def\BitSet@@GetDec#1!#2!#3#4!{%
  \ifx\\#4\\%
    \ifx#31%
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@Space
        \IntCalcAdd#1!#2!%
      }%
    \else
      \BitSet@AfterFiFi{ #2}%
    \fi
  \else
    \ifx#31%
      \BitSet@AfterFiFi{%
        \csname BitSet@N#1%
        \expandafter\expandafter\expandafter\endcsname
        \IntCalcAdd#1!#2!!#4!%
      }%
    \else
      \BitSet@AfterFiFi{%
        \csname BitSet@N#1\endcsname#2!#4!%
      }%
    \fi
  \BitSet@Fi
}
```

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

```
\def\BitSet@Temp#1#2{%
  \expandafter\def\csname BitSet@N#1\endcsname{%
    \BitSet@@GetDec#2!%
  }%
}
\BitSet@Temp{1}{2}
\BitSet@Temp{2}{4}
\BitSet@Temp{4}{8}
\BitSet@Temp{8}{16}
\BitSet@Temp{16}{32}
\BitSet@Temp{32}{64}
\BitSet@Temp{64}{128}
\BitSet@Temp{128}{256}
\BitSet@Temp{256}{512}
\BitSet@Temp{512}{1024}
\BitSet@Temp{1024}{2048}
\BitSet@Temp{2048}{4096}
\BitSet@Temp{4096}{8192}
\BitSet@Temp{8192}{16384}
\BitSet@Temp{16384}{32768}
\BitSet@Temp{32768}{65536}
\BitSet@Temp{65536}{131072}
\BitSet@Temp{131072}{262144}
\BitSet@Temp{262144}{524288}
\BitSet@Temp{524288}{1048576}
\BitSet@Temp{1048576}{2097152}
\BitSet@Temp{2097152}{4194304}
\BitSet@Temp{4194304}{8388608}
\BitSet@Temp{8388608}{16777216}
\BitSet@Temp{16777216}{33554432}
\BitSet@Temp{33554432}{67108864}
\BitSet@Temp{67108864}{134217728}
\BitSet@Temp{134217728}{268435456}
\BitSet@Temp{268435456}{536870912}
\BitSet@Temp{536870912}{1073741824}
```

\BitSet@N1073741824

```
\expandafter\def\csname BitSet@N1073741824\endcsname{%
  \BitSet@GetDecBig2147483648!%
}%
```

\BitSet@GetDecBig

#1: current power of two
#2: result
#3#4: number

```
\def\BitSet@GetDecBig#1!#2!#3#4!{%
  \ifx\\#4\\%
    \ifx#31%
      \BitSet@AfterFiFi{%
```

```
        \expandafter\expandafter\expandafter\BitSet@Space
        \BigIntCalcAdd#1!#2!%
      }%
    \else
      \BitSet@AfterFiFi{ #2}%
    \fi
  \else
    \ifx#31%
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@@GetDecBig
        \BigIntCalcAdd#1!#2!!#1!#4!%
      }%
    \else
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@GetDecBig
        \BigIntCalcShl#1!!#2!#4!%
      }%
    \fi
  \BitSet@Fi
}
```

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

```
\def\BitSet@@GetDecBig#1!#2!{%
  \expandafter\expandafter\expandafter\BitSet@GetDecBig
  \BigIntCalcShl#2!!#1!%
}
```

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

```
\def\bitsetAnd#1#2{%
  \bitsetIsEmpty{#1}{%
    \bitsetReset{#1}%
  }{%
    \bitsetIsEmpty{#2}{%
      \bitsetReset{#1}%
    }{%
      \expandafter\edef\csname BS@#1\endcsname{%
        \expandafter\expandafter\expandafter\BitSet@And
        \csname BS@#1\expandafter\expandafter\expandafter\endcsname
        \expandafter\expandafter\expandafter!%
        \csname BS@#2\endcsname!!%
      }%
      \expandafter\ifx\csname BS@#1\endcsname\BitSet@Empty
        \bitsetReset{#1}%
      \fi
    }%
  }%
}
```

\BitSet@And

```
\def\BitSet@And#1#2!#3#4!#5!{%
  \ifx\\#2\\%
    \ifnum#1#3=11 #51\fi
  \else
    \ifx\\#4\\%
      \ifnum#1#3=11 #51\fi
    \else
      \ifnum#1#3=11 %
        #51%
        \BitSet@AfterFiFiFi{%
          \BitSet@And#2!#4!!%
        }%
      \else
        \BitSet@AfterFiFiFi{%
          \BitSet@And#2!#4!#50!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

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

```
\def\bitsetAndNot#1#2{%
  \bitsetIsEmpty{#1}{%
    \bitsetReset{#1}%
  }{%
    \bitsetIsEmpty{#2}{%
    }{%
      \expandafter\edef\csname BS@#1\endcsname{%
        \expandafter\expandafter\expandafter\BitSet@AndNot
        \csname BS@#1\expandafter\expandafter\expandafter\endcsname
        \expandafter\expandafter\expandafter!%
        \csname BS@#2\endcsname!!%
      }%
      \expandafter\ifx\csname BS@#1\endcsname\BitSet@Empty
        \bitsetReset{#1}%
      \fi
    }%
  }%
}
```

\BitSet@AndNot

```
\def\BitSet@AndNot#1#2!#3#4!#5!{%
  \ifx\\#2\\%
    \ifnum#1#3=10 #51\fi
  \else
    \ifx\\#4\\%
      #5%
      \ifnum#1#3=10 1\else 0\fi
      #2%
    \else
      \ifnum#1#3=10 %
        #51%
        \BitSet@AfterFiFiFi{%
          \BitSet@AndNot#2!#4!!%
        }%
      \else
        \BitSet@AfterFiFiFi{%
          \BitSet@AndNot#2!#4!#50!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

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

```
\def\bitsetOr#1#2{%
  \bitsetIsEmpty{#2}{%
    \BitSet@IfUndefined{#1}{\bitsetReset{#1}}{}%
  }{%
    \bitsetIsEmpty{#1}{%
      \expandafter\let\csname BS@#1\expandafter\endcsname
                      \csname BS@#2\endcsname
    }{%
      \expandafter\edef\csname BS@#1\endcsname{%
        \expandafter\expandafter\expandafter\BitSet@Or
        \csname BS@#1\expandafter\expandafter\expandafter\endcsname
        \expandafter\expandafter\expandafter!%
        \csname BS@#2\endcsname!%
      }%
    }%
  }%
}
```

\BitSet@Or

```
\def\BitSet@Or#1#2!#3#4!{%
  \ifnum#1#3>0 1\else 0\fi
  \ifx\\#2\\%
    #4%
  \else
    \ifx\\#4\\%
      #2%
    \else
      \BitSet@AfterFiFi{%
        \BitSet@Or#2!#4!%
      }%
    \fi
  \BitSet@Fi
}
```

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 ^= B

```
\def\bitsetXor#1#2{%
  \bitsetIsEmpty{#2}{%
    \BitSet@IfUndefined{#1}{\bitsetReset{#1}}{}%
  }{%
    \bitsetIsEmpty{#1}{%
      \expandafter\let\csname BS@#1\expandafter\endcsname
                      \csname BS@#2\endcsname
    }{%
      \expandafter\edef\csname BS@#1\endcsname{%
        \expandafter\expandafter\expandafter\BitSet@Xor
        \csname BS@#1\expandafter\expandafter\expandafter\endcsname
        \expandafter\expandafter\expandafter!%
        \csname BS@#2\endcsname!!%
      }%
      \expandafter\ifx\csname BS@#1\endcsname\BitSet@Empty
        \bitsetReset{#1}%
      \fi
    }%
  }%
}
```

`\BitSet@Xor`

```
\def\BitSet@Xor#1#2!#3#4!#5!{%
  \ifx\\#2\\%
    \ifx#1#3%
      \ifx\\#4\\%
      \else
        #50#4%
      \fi
    \else
      #51#4%
    \fi
  \else
    \ifx\\#4\\%
      #5%
      \ifx#1#30\else 1\fi
      #2%
    \else
      \ifx#1#3%
        \BitSet@AfterFiFiFi{%
          \BitSet@Xor#2!#4!#50!%
        }%
      \else
        #51%
        \BitSet@AfterFiFiFi{%
          \BitSet@Xor#2!#4!!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

2.8.5 Shifting

2.8.6 \bitsetShiftLeft

\bitsetShiftLeft

```
\def\bitsetShiftLeft#1#2{%
  \BitSet@IfUndefined{#1}{%
    \bitsetReset{#1}%
  }{%
    \bitsetIsEmpty{#1}{%
    }{%
      \expandafter\expandafter\expandafter\BitSet@ShiftLeft
      \intcalcNum{#2}!{#1}%
    }%
  }%
}
```

\BitSet@ShiftLeft

```
\def\BitSet@ShiftLeft#1!#2{%
  \ifcase\intcalcSgn{#1} %
  \or
    \begingroup
      \uccode`m=`0 %
    \uppercase\expandafter{\expandafter\endgroup
      \expandafter\edef\csname BS@#2\expandafter\endcsname
      \expandafter{%
        \romannumeral#1000\expandafter\BitSet@Space
        \csname BS@#2\endcsname
      }%
    }%
  \else
    \expandafter\BitSet@ShiftRight\BitSet@Gobble#1!{#2}%
  \fi
}
```

2.8.7 \bitsetShiftRight

\bitsetShiftRight

```
\def\bitsetShiftRight#1#2{%
  \BitSet@IfUndefined{#1}{%
    \bitsetReset{#1}%
  }{%
    \bitsetIsEmpty{#1}{%
    }{%
      \expandafter\expandafter\expandafter\BitSet@ShiftRight
      \intcalcNum{#2}!{#1}%
    }%
  }%
}
```

\BitSet@ShiftRight

```
\def\BitSet@ShiftRight#1!#2{%
  \ifcase\intcalcSgn{#1} %
  \or
    \expandafter\edef\csname BS@#2\endcsname{%
      \expandafter\expandafter\expandafter\BitSet@Kill
      \csname BS@#2\expandafter\endcsname\expandafter\BitSet@Empty
      \expandafter=%
      \expandafter{\expandafter}\expandafter{\expandafter}%
      \romannumeral#1000!%
    }%
  \else
    \expandafter\BitSet@ShiftLeft\BitSet@Gobble#1!{#2}%
  \fi
}
```

\BitSet@Kill

```
\def\BitSet@Kill#1#2=#3#4#5{%
  #3#4%
  \ifx#5!%
    \ifx#1\BitSet@Empty
      0%
    \else
      #1#2%
    \fi
  \else
    \ifx#1\BitSet@Empty
      0%
      \BitSet@AfterFiFi\BitSet@Cleanup
    \else
      \BitSet@Kill#2=%
    \fi
  \BitSet@Fi
}
```

2.9 Bit manipulation

\bitsetClear

```
\def\bitsetClear{%
  \BitSet@CheckIndex\BitSet@Clear
}
```

\bitsetSet

```
\def\bitsetSet{%
  \BitSet@CheckIndex\BitSet@Set
}
```

\bitsetFlip

```
\def\bitsetFlip{%
  \BitSet@CheckIndex\BitSet@Flip
}
```

\bitsetSetValue

```
\def\bitsetSetValue#1#2#3{%
  \expandafter\expandafter\expandafter\BitSet@SetValue
  \intcalcNum{#3}!{#1}{#2}%
}
```

\BitSet@SetValue #1: plain value
#2: BitSet
#3: Index

```
\def\BitSet@SetValue#1!{%
  \BitSet@CheckIndex{%
```

```
    \ifcase#1 %
      \expandafter\BitSet@Clear
    \or
      \expandafter\BitSet@Set
    \else
      \BitSet@ErrorInvalidBitValue{#1}%
      \expandafter\expandafter\expandafter\BitSet@Gobble
      \expandafter\BitSet@Gobble
    \fi
  }%
}
```

`\BitSet@ErrorInvalidBitValue` #1: Wrong bit value

```
\def\BitSet@ErrorInvalidBitValue#1{%
  \@PackageError{bitset}{%
    Invalid bit value (#1) not in range 0..1%
  }\@ehc
}
```

2.9.1 Clear operation

`\BitSet@Clear` #1: BitSet
#2: plain and checked index

```
\def\BitSet@Clear#1#2{%
  \edef\BitSet@Temp{%
    \expandafter\expandafter\expandafter\BitSet@@Clear
    \csname BS@#1\expandafter\endcsname
    \expandafter\BitSet@Empty\expandafter=\expandafter!%
    \romannumeral#2000!%
  }%
  \expandafter\let\csname BS@#1\expandafter\endcsname
  \ifx\BitSet@Temp\BitSet@Empty
    \BitSet@Zero
  \else
    \BitSet@Temp
  \fi
}
```

`\BitSet@@Clear`

```
\def\BitSet@@Clear#1#2=#3!#4{%
  \ifx#4!%
    \ifx#1\BitSet@Empty
    \else
      \ifx\BitSet@Empty#2%
      \else
        #30#2%
      \fi
    \fi
  \else
    \ifx#1\BitSet@Empty
      \BitSet@AfterFiFi\BitSet@Cleanup
    \else
      \ifx#10%
        \BitSet@AfterFiFiFi{%
          \BitSet@@Clear#2=#30!%
        }%
      \else
        #31%
        \BitSet@AfterFiFiFi{%
          \BitSet@@Clear#2=!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

2.9.2 Set operation

`\BitSet@Set` #1: BitSet
#2: plain and checked Index

```
\def\BitSet@Set#1#2{%
  \expandafter\edef\csname BS@#1\endcsname{%
    \expandafter\expandafter\expandafter\BitSet@@Set
    \csname BS@#1\expandafter\endcsname
    \expandafter\BitSet@Empty\expandafter=%
    \expandafter{\expandafter}\expandafter{\expandafter}%
    \romannumeral#2000!%
  }%
}
```

`\BitSet@@Set`

```
\def\BitSet@@Set#1#2=#3#4#5{%
  #3#4%
  \ifx#5!%
    1#2%
  \else
    \ifx#1\BitSet@Empty
      0%
      \BitSet@AfterFiFi\BitSet@@@Set
    \else
      #1%
      \BitSet@@Set#2=%
    \fi
  \BitSet@Fi
}
```

`\BitSet@@@Set`

```
\def\BitSet@@@Set#1{%
  \ifx#1!%
    1%
  \else
    0%
    \expandafter\BitSet@@@Set
  \fi
}
```

2.9.3 Flip operation

`\BitSet@Flip` #1: BitSet
#2: plain and checked Index

```
\def\BitSet@Flip#1#2{%
  \edef\BitSet@Temp{%
    \expandafter\expandafter\expandafter\BitSet@@Flip
    \csname BS@#1\expandafter\endcsname
    \expandafter\BitSet@Empty\expandafter=\expandafter!%
    \romannumeral#2000!%
  }%
  \expandafter\let\csname BS@#1\expandafter\endcsname
  \ifx\BitSet@Temp\BitSet@Empty
    \BitSet@Zero
  \else
    \BitSet@Temp
  \fi
}
```

\BitSet@@Flip

```
\def\BitSet@@Flip#1#2=#3!#4{%
  \ifx#4!%
    \ifx#11%
      \ifx\BitSet@Empty#2%
      \else
        #30#2%
      \fi
    \else
      #31#2%
    \fi
  \else
    \ifx#1\BitSet@Empty
      #30%
      \BitSet@AfterFiFi\BitSet@@@Set
    \else
      \ifx#10%
        \BitSet@AfterFiFiFi{%
          \BitSet@@Flip#2=#30!%
        }%
      \else
        #31%
        \BitSet@AfterFiFiFi{%
          \BitSet@@Flip#2=!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

2.9.4 Range operators

\bitsetClearRange

```
\def\bitsetClearRange{%
  \BitSet@Range\BitSet@Clear
}
```

\bitsetSetRange

```
\def\bitsetSetRange{%
  \BitSet@Range\BitSet@Set
}
```

\bitsetFlipRange

```
\def\bitsetFlipRange{%
  \BitSet@Range\BitSet@Flip
}
```

\bitsetSetValueRange

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

\BitSet@SetValueRange

```
\def\BitSet@SetValueRange#1!#2#3#4{%
  \ifcase#1 %
    \BitSet@Range\BitSet@Clear{#2}{#3}{#4}%
  \or
    \BitSet@Range\BitSet@Set{#2}{#3}{#4}%
  \else
    \BitSet@ErrorInvalidBitValue{#1}%
  \fi
}
```

\BitSet@Range #1: clear/set/flip macro
#2: BitSet
#3: Index from
#4: Index to

```
\def\BitSet@Range#1#2#3#4{%
  \edef\BitSet@Temp{%
    \noexpand\BitSet@@Range\noexpand#1{#2}%
    \intcalcNum{#3}!\intcalcNum{#4}!%
  }%
  \BitSet@Temp
}
```

\BitSet@@Range #1: clear/set/flip macro
#2: BitSet
#3: Index from
#4: Index to

```
\def\BitSet@@Range#1#2#3!#4!{%
  \ifnum#3<0 %
    \BitSet@NegativeIndex#1{#2}#3!#4!0!#4!%
  \else
    \ifnum#4<0 %
      \BitSet@NegativeIndex#1{#2}#3!#4!#3!0!%
    \else
      \ifcase\intcalcCmp{#3}{#4} %
      \or
        \@PackageError{bitset}{%
          Wrong index numbers in range [#3..#4]\MessageBreak% hash-ok
          for clear/set/flip on bit set `#2'.\MessageBreak
          The lower index exceeds the upper index.\MessageBreak
          Canceling the operation as error recovery%
        }\@ehc
      \else
        \BitSet@@@Range#3!#4!#1{#2}%
      \fi
    \fi
  \fi
}
```

\BitSet@NegativeIndex

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

```
  \@PackageError{bitset}{%
    Negative index in range [#3..#4]\MessageBreak % hash-ok
    for \string\bitset
    \ifx#1\BitSet@Clear
      Clear%
    \else
      \ifx#1\BitSet@Set
        Set%
      \else
        Flip%
      \fi
    \fi
    Range on bit set `#2'.\MessageBreak
    Using [#5..#6] as error recovery% hash-ok
  }\@ehc
  \BitSet@@Range#1{#2}#5!#6!%
}
```

\BitSet@@Range

```
\def\BitSet@@@Range#1!#2!#3#4{%
  \ifnum#1<#2 %
    #3{#4}{#1}%
    \BitSet@AfterFi{%
      \expandafter\expandafter\expandafter\BitSet@@@Range
      \IntCalcInc#1!!#2!#3{#4}%
    }%
  \BitSet@Fi
}
```

2.10 Bit retrieval

2.10.1 \bitsetGet

\bitsetGet

```
\def\bitsetGet#1#2{%
  \number
  \expandafter\expandafter\expandafter\BitSet@Get
  \intcalcNum{#2}!{#1}%
}
```

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

```
\def\BitSet@Get#1!#2{%
  \ifnum#1<0 %
    \BitSet@AfterFi{%
      0 \BitSetError:NegativeIndex%
    }%
  \else
    \BitSet@IfUndefined{#2}{0}{%
      \expandafter\expandafter\expandafter\BitSet@@Get
      \csname BS@#2\expandafter\endcsname
      \expandafter!\expandafter=%
      \expandafter{\expandafter}\expandafter{\expandafter}%
      \romannumeral\intcalcNum{#1}000!%
    }%
    \expandafter\BitSet@Space
  \BitSet@Fi
}
```

\BitSet@@Get

```
\def\BitSet@@Get#1#2=#3#4#5{%
  #3#4%
  \ifx#5!%
    \ifx#1!%
      0%
    \else
      #1%
    \fi
  \else
    \ifx#1!%
      0%
      \BitSet@AfterFiFi\BitSet@Cleanup
    \else
      \BitSet@@Get#2=%
    \fi
  \BitSet@Fi
}
```

2.10.2 \bitsetNextClearBit, \bitsetNextSetBit

\bitsetNextClearBit

```
\def\bitsetNextClearBit#1#2{%
  \number
  \expandafter\expandafter\expandafter\BitSet@NextClearBit
  \intcalcNum{#2}!{#1} %
}
```

\BitSet@NextClearBit

#1: Index
#2: BitSet

```
\def\BitSet@NextClearBit#1!#2{%
  \ifnum#1<0 %
    \BitSet@NextClearBit0!{#2}%
    \BitSet@AfterFi{%
      \expandafter\BitSet@Space
      \expandafter\BitSetError:NegativeIndex\romannumeral0%
    }%
  \else
    \bitsetIsEmpty{#2}{#1}{%
      \expandafter\BitSet@Skip
      \number#1\expandafter\expandafter\expandafter!%
      \csname BS@#2\endcsname!!!!!!!!!=%
      {\BitSet@@NextClearBit#1!}%
    }%
  \BitSet@Fi
}
```

\BitSet@@NextClearBit

#1: index for next bit in #2
#2: next bit

```
\def\BitSet@@NextClearBit#1!#2{%
  \ifx#2!%
    #1%
  \else
    \ifx#20%
      #1%
      \BitSet@AfterFiFi\BitSet@Cleanup
    \else
```

```
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@@NextClearBit
        \IntCalcInc#1!!%
      }%
    \fi
  \BitSet@Fi
}
```

\bitsetNextSetBit

```
\def\bitsetNextSetBit#1#2{%
  \number
  \expandafter\expandafter\expandafter\BitSet@NextSetBit
  \intcalcNum{#2}!{#1} %
}
```

\BitSet@NextSetBit #1: Index
#2: BitSet

```
\def\BitSet@NextSetBit#1!#2{%
  \ifnum#1<0 %
    \BitSet@NextSetBit0!{#2}%
    \BitSet@AfterFi{%
      \expandafter\BitSet@Space
      \expandafter\BitSetError:NegativeIndex\romannumeral0%
    }%
  \else
    \bitsetIsEmpty{#2}{-1}{%
      \expandafter\BitSet@Skip
      \number#1\expandafter\expandafter\expandafter!%
      \csname BS@#2\endcsname!!!!!!!!!=%
      {\BitSet@@NextSetBit#1!}%
    }%
  \BitSet@Fi
}
```

\BitSet@@NextSetBit #1: index for next bit in #2
#2: next bit

```
\def\BitSet@@NextSetBit#1!#2{%
  \ifx#2!%
    -1%
  \else
    \ifx#21%
      #1%
      \BitSet@AfterFiFi\BitSet@Cleanup
    \else
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@@NextSetBit
        \IntCalcInc#1!!%
      }%
    \fi
  \BitSet@Fi
}
```

\BitSet@Cleanup

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

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

```
\def\BitSet@Skip#1!#2{%
  \ifx#2!%
    \BitSet@AfterFi{%
      \BitSet@SkipContinue%
    }%
  \else
    \ifcase#1 %
      \BitSet@AfterFiFi{%
        \BitSet@SkipContinue#2%
      }%
    \or
      \BitSet@AfterFiFi\BitSet@SkipContinue
    \or
      \BitSet@AfterFiFi{%
        \expandafter\BitSet@SkipContinue\BitSet@Gobble
      }%
    \else
      \ifnum#1>8 %
        \BitSet@AfterFiFiFi{%
          \expandafter\BitSet@Skip
          \number\IntCalcSub#1!8!\expandafter!%
          \BitSet@GobbleSeven
        }%
      \else
        \BitSet@AfterFiFiFi{%
          \expandafter\expandafter\expandafter\BitSet@Skip
          \IntCalcDec#1!!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

`\BitSet@SkipContinue`

`#1`: remaining bits
`#2`: continuation code

```
\def\BitSet@SkipContinue#1!#2=#3{%
  #3#1!%
}
```

`\BitSet@GobbleSeven`

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

2.10.3 `\bitsetGetSetBitList`

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

```
\def\bitsetGetSetBitList#1{%
  \romannumeral0%
  \bitsetIsEmpty{#1}{ }{%
    \expandafter\BitSet@GetSetBitList
    \number\BitSet@NextSetBit0!{#1}!{#1}{}!%
  }%
}
```

`\BitSet@GetSetBitList`

`#1`: found index
`#2`: BitSet
`#3`: comma `#4`: result

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

```
  \ifnum#1<0 %
    \BitSet@AfterFi{ #4}%
  \else
    \BitSet@AfterFi{%
      \expandafter\BitSet@GetSetBitList\number
      \expandafter\expandafter\expandafter\BitSet@NextSetBit
      \IntCalcInc#1!!{#2}!{#2},#4#3#1!%
    }%
  \BitSet@Fi
}
```

2.11 Bit set properties

\bitsetSize

```
\def\bitsetSize#1{%
  \number
  \BitSet@IfUndefined{#1}{0 }{%
    \expandafter\expandafter\expandafter\BitSet@Size
    \expandafter\expandafter\expandafter1%
    \expandafter\expandafter\expandafter!%
    \csname BS@#1\endcsname!0!%
  }%
}
```

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

```
\def\BitSet@Size#1!#2#3!#4!{%
  \ifx#21%
    \ifx\\#3\\%
      \BitSet@AfterFiFi{#1 }%
    \else
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@Size
        \IntCalcInc#1!!#3!#1!%
      }%
    \fi
  \else
    \ifx\\#3\\%
      \BitSet@AfterFiFi{#4 }%
    \else
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@Size
        \IntCalcInc#1!!#3!#4!%
      }%
    \fi
  \fi
  \BitSet@Fi
}
```

\bitsetCardinality

```
\def\bitsetCardinality#1{%
  \number
  \BitSet@IfUndefined{#1}{0 }{%
    \expandafter\expandafter\expandafter\BitSet@Cardinality
    \expandafter\expandafter\expandafter0%
    \expandafter\expandafter\expandafter!%
    \csname BS@#1\endcsname!%
  }%
}
```

\BitSet@Cardinality

#1: result

#2#3: bits

```
\def\BitSet@Cardinality#1!#2#3!{%
  \ifx#21%
    \ifx\\#3\\%
      \BitSet@AfterFiFi{\IntCalcInc#1! }%
    \else
      \BitSet@AfterFiFi{%
        \expandafter\expandafter\expandafter\BitSet@Cardinality
        \IntCalcInc#1!!#3!%
      }%
    \fi
  \else
    \ifx\\#3\\%
      \BitSet@AfterFiFi{#1 }%
    \else
      \BitSet@AfterFiFi{%
        \BitSet@Cardinality#1!#3!%
      }%
    \fi
  \fi
  \BitSet@Fi
}
```

2.12 Queries

\bitsetIsDefined

```
\def\bitsetIsDefined#1{%
  \BitSet@IfUndefined{#1}%
  \BitSet@SecondOfTwo
  \BitSet@FirstOfTwo
}
```

\bitsetIsEmpty

```
\def\bitsetIsEmpty#1{%
  \BitSet@IfUndefined{#1}\BitSet@FirstOfTwo{%
    \expandafter\ifx\csname BS@#1\endcsname\BitSet@Zero
      \expandafter\BitSet@FirstOfTwo
    \else
      \expandafter\BitSet@SecondOfTwo
    \fi
  }%
}
```

\BitSet@Zero

```
\def\BitSet@Zero{0}
```

\bitsetQuery

```
\def\bitsetQuery#1#2{%
  \ifnum\bitsetGet{#1}{#2}=1 %
    \expandafter\BitSet@FirstOfTwo
  \else
    \expandafter\BitSet@SecondOfTwo
  \fi
}
```

\bitsetEquals

```
\def\bitsetEquals#1#2{%
  \BitSet@IfUndefined{#1}{%
    \BitSet@IfUndefined{#2}\BitSet@FirstOfTwo\BitSet@SecondOfTwo
  }{%
    \BitSet@IfUndefined{#2}\BitSet@SecondOfTwo{%
      \expandafter\ifx\csname BS@#1\expandafter\endcsname
                      \csname BS@#2\endcsname
        \expandafter\BitSet@FirstOfTwo
      \else
        \expandafter\BitSet@SecondOfTwo
      \fi
    }%
  }%
}
```

\bitsetIntersects

```
\def\bitsetIntersects#1#2{%
  \bitsetIsEmpty{#1}\BitSet@SecondOfTwo{%
    \bitsetIsEmpty{#2}\BitSet@SecondOfTwo{%
      \expandafter\expandafter\expandafter\BitSet@Intersects
      \csname BS@#1\expandafter\expandafter\expandafter\endcsname
      \expandafter\expandafter\expandafter!%
      \csname BS@#2\endcsname!%
    }%
  }%
}
```

\BitSet@Intersects

```
\def\BitSet@Intersects#1#2!#3#4!{%
  \ifnum#1#3=11 %
    \BitSet@AfterFi\BitSet@FirstOfTwo
  \else
    \ifx\\#2\\%
      \BitSet@AfterFiFi\BitSet@SecondOfTwo
    \else
      \ifx\\#4\\%
        \BitSet@AfterFiFiFi\BitSet@SecondOfTwo
      \else
        \BitSet@AfterFiFiFi{%
          \BitSet@Intersects#2!#4!%
        }%
      \fi
    \fi
  \BitSet@Fi
}
```

```
\BitSet@AtEnd%
⟨/package⟩
```

3 Test

3.1 Catcode checks for loading

⟨*test1⟩

```
\catcode`\{=1 %
\catcode`\}=2 %
\catcode`\#=6 %
\catcode`\@=11 %
\expandafter\ifx\csname count@\endcsname\relax
  \countdef\count@=255 %
\fi
\expandafter\ifx\csname @gobble\endcsname\relax
  \long\def\@gobble#1{}%
\fi
\expandafter\ifx\csname @firstofone\endcsname\relax
  \long\def\@firstofone#1{#1}%
\fi
\expandafter\ifx\csname loop\endcsname\relax
  \expandafter\@firstofone
\else
  \expandafter\@gobble
\fi
{%
  \def\loop#1\repeat{%
    \def\body{#1}%
    \iterate
  }%
  \def\iterate{%
    \body
      \let\next\iterate
    \else
      \let\next\relax
    \fi
    \next
  }%
  \let\repeat=\fi
}%
\def\RestoreCatcodes{}
\count@=0 %
\loop
  \edef\RestoreCatcodes{%
    \RestoreCatcodes
    \catcode\the\count@=\the\catcode\count@\relax
  }%
\ifnum\count@<255 %
  \advance\count@ 1 %
\repeat

\def\RangeCatcodeInvalid#1#2{%
  \count@=#1\relax
  \loop
    \catcode\count@=15 %
  \ifnum\count@<#2\relax
    \advance\count@ 1 %
  \repeat
}
\def\RangeCatcodeCheck#1#2#3{%
  \count@=#1\relax
  \loop
    \ifnum#3=\catcode\count@
    \else
    \errmessage{%
      Character \the\count@\space
      with wrong catcode \the\catcode\count@\space
      instead of \number#3%
    }%
  \fi
\ifnum\count@<#2\relax
  \advance\count@ 1 %
\repeat
}
\def\space{ }
\expandafter\ifx\csname LoadCommand\endcsname\relax
  \def\LoadCommand{\input bitset.sty\relax}%
\fi
\def\Test{%
  \RangeCatcodeInvalid{0}{47}%
  \RangeCatcodeInvalid{58}{64}%
  \RangeCatcodeInvalid{91}{96}%
  \RangeCatcodeInvalid{123}{255}%
  \catcode`\@=12 %
  \catcode`\\=0 %
  \catcode`\%=14 %
  \LoadCommand
  \RangeCatcodeCheck{0}{36}{15}%
  \RangeCatcodeCheck{37}{37}{14}%
  \RangeCatcodeCheck{38}{47}{15}%
  \RangeCatcodeCheck{48}{57}{12}%
  \RangeCatcodeCheck{58}{63}{15}%
  \RangeCatcodeCheck{64}{64}{12}%
  \RangeCatcodeCheck{65}{90}{11}%
  \RangeCatcodeCheck{91}{91}{15}%
  \RangeCatcodeCheck{92}{92}{0}%
  \RangeCatcodeCheck{93}{96}{15}%
  \RangeCatcodeCheck{97}{122}{11}%
  \RangeCatcodeCheck{123}{255}{15}%
  \RestoreCatcodes
}
\Test
\csname @@end\endcsname
\end
```

⟨/test1⟩

3.2 Macro tests

3.2.1 Preamble

⟨*test2⟩

```
\NeedsTeXFormat{LaTeX2e}
\nofiles
\documentclass{article}
\makeatletter
```

⟨*noetex⟩

```
\let\SavedNumexpr\numexpr
\let\SavedIfcsname\ifcsname
\let\SavedCurrentgrouplevel\currentgrouplevel
\def\ETeXDisable{%
  \let\ifcsname\@undefined
  \let\numexpr\@undefined
  \let\currentgrouplevel\@undefined
}
\ETeXDisable
⟨/noetex⟩
\makeatletter
\chardef\BitSet@TestMode=1 %
\makeatother
\usepackage{bitset}[2016/05/16]
⟨*noetex⟩
\def\ETeXEnable{%
  \let\numexpr\SavedNumexpr
  \let\ifcsname\SavedIfcsname
  \let\currentgrouplevel\SavedCurrentgrouplevel
}
\ETeXEnable
⟨/noetex⟩
\usepackage{qstest}
\IncludeTests{*}
\LogTests{log}{*}{*}
\makeatletter
```

3.2.2 Time

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname pdfresettimer\endcsname\relax
\else
  \newcount\SummaryTime
  \newcount\TestTime
  \SummaryTime=\z@
  \newcommand*{\PrintTime}[2]{%
    \typeout{%
      [Time #1: \strip@pt\dimexpr\number#2sp\relax\space s]%
    }%
  }%
  \newcommand*{\StartTime}[1]{%
    \renewcommand*{\TimeDescription}{#1}%
    \pdfresettimer
  }%
  \newcommand*{\TimeDescription}{}%
  \newcommand*{\StopTime}{%
    \TestTime=\pdfelapsedtime
    \global\advance\SummaryTime\TestTime
    \PrintTime\TimeDescription\TestTime
  }%
  \let\saved@qstest\qstest
  \let\saved@endqstest\endqstest
  \def\qstest#1#2{%
    \saved@qstest{#1}{#2}%
    \StartTime{#1}%
  }%
  \def\endqstest{%
    \StopTime
    \saved@endqstest
  }%
  \AtEndDocument{%
    \PrintTime{summary}\SummaryTime
  }%
\fi
```

3.2.3 Detection of unwanted space

```
\let\orig@qstest\qstest
\let\orig@endqstest\endqstest
\def\qstest#1#2{%
  \orig@qstest{#1}{#2}%
  \setbox0\hbox\bgroup\begingroup\ignorespaces
}
\def\endqstest{%
  \endgroup\egroup
  \Expect*{\the\wd0}{0.0pt}%
  \orig@endqstest
}
```

3.2.4 Test macros

```
\newcounter{Test}

\def\TestError#1#2{%
  \begingroup
    \setcounter{Test}{0}%
    \sbox0{%
      \def\@PackageError##1##2##3{%
        \stepcounter{Test}%
        \begingroup
          \let\MessageBreak\relax
⟨*noetex⟩
          \ETeXEnable
⟨/noetex⟩
          \Expect{##1}{bitset}%
          \Expect*{##2}*{#1}%
        \endgroup
      }%
⟨*noetex⟩
      \ETeXDisable
⟨/noetex⟩
      #2%
    }%
    \Expect*{\theTest}{1}%
    \Expect*{\the\wd0}{0.0pt}%
  \endgroup
}

\def\TestErrorNegativeIndex#1#2{%
  \TestError{Invalid negative index (#1)}{#2}%
}

\def\TestGetterUndefined#1{%
  \CheckUndef{dummy}%
  \expandafter\expandafter\expandafter\Expect
  \expandafter\expandafter\expandafter{#1{dummy}}{0}%
}

\def\ExpectBitSet#1#2{%
  \expandafter\expandafter\expandafter\Expect
  \expandafter\expandafter\expandafter
  {\csname BS@#1\endcsname}*{#2}%
}
\def\Check#1#2{%
  \ExpectBitSet{#1}{#2}%
}
```

```
\def\CheckUndef#1{%
  \begingroup
    \Expect*{%
      \expandafter
      \ifx\csname BS@#1\endcsname\relax true\else false\fi
    }{true}%
  \endgroup
}
\def\RevCheck#1#2{%
  \ExpectBitSet{#1}{\Reverse#2!!}%
}
\def\Set#1#2{%
  \expandafter\def\csname BS@#1\endcsname{#2}%
}
\def\RevSet#1#2{%
  \expandafter\edef\csname BS@#1\endcsname{%
    \Reverse#2!!%
  }%
}
\def\Reverse#1#2!#3!{%
  \ifx\\#2\\%
    #1#3%
    \expandafter\@gobble
  \else
    \expandafter\@firstofone
  \fi
  {\Reverse#2!#1#3!}%
}
```

3.2.5 Test sets

```
\begin{qstest}{Let}{Let}
  \CheckUndef{abc}%
  \CheckUndef{xyz}%
  \bitsetLet{xyz}{abc}%
  \CheckUndef{abc}%
  \Check{xyz}{0}%
  \Set{abc}{1}%
  \Check{abc}{1}%
  \Check{xyz}{0}%
  \bitsetLet{xyz}{abc}%
  \Check{abc}{1}%
  \Check{xyz}{1}%
  \Set{xyz}{11}%
  \Check{abc}{1}%
  \Check{xyz}{11}%
\end{qstest}

\begin{qstest}{Reset}{Reset}
  \bitsetReset{xyz}%
  \Check{xyz}{0}%
  \bitsetReset{abc}%
  \Check{abc}{0}%
  \Set{abc}{10101}%
  \bitsetReset{abc}%
  \Check{abc}{0}%
\end{qstest}

\begin{qstest}{Get/Query}{Get/Query}
```

```
  \expandafter\expandafter\expandafter\Expect
  \expandafter\expandafter\expandafter{%
    \bitsetGet{dummy}{0}%
  }{0}%
  \begingroup
    \expandafter\def\csname BitSetError:NegativeIndex\endcsname{}%
    \Set{abc}{1}%
    \Expect*{\bitsetQuery{abc}{-1}{true}{false}}{false}%
  \endgroup
  \def\Test#1#2#3{%
    \Set{abc}{#1}%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{\bitsetGet{abc}{#2}}{#3}%
    \Expect*{\bitsetQuery{abc}{#2}{true}{false}}%
           *{\ifcase#3 false\or true\else error\fi}%
  }%
  \Test{1}{100}{0}%
  \Test{0}{0}{0}%
  \Test{1}{0}{1}%
  \Test{11}{1}{1}%
  \Test{111}{1}{1}%
  \Test{101}{1}{0}%
  \Test{101}{2}{1}%
  \Test{10100110011}{10}{1}%
\end{qstest}

\begin{qstest}{Size}{Size}
  \TestGetterUndefined\bitsetSize
  \def\Test#1#2{%
    \Set{abc}{#1}%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{\bitsetSize{abc}}{#2}%
  }%
  \Test{0}{0}%
  \Test{1}{1}%
  \Test{00}{0}%
  \Test{0000000}{0}%
  \Test{10}{1}%
  \Test{01}{2}%
  \Test{11}{2}%
  \Test{010}{2}%
  \Test{011}{3}%
  \Test{100110011}{9}%
  \Test{0000011111000001111100000}{20}%
  \Test{000000000000000000000000011111111111111111111}{45}%
\end{qstest}

\begin{qstest}{Cardinality}{Cardinality}
  \TestGetterUndefined\bitsetCardinality
  \def\Test#1#2{%
    \Set{abc}{#1}%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetCardinality{abc}%
    }{#2}%
  }%
  \Test{0}{0}%
  \Test{1}{1}%
  \Test{00}{0}%
  \Test{0000000}{0}%
  \Test{10}{1}%
  \Test{01}{1}%
  \Test{11}{2}%
  \Test{010}{1}%
  \Test{011}{2}%
  \Test{100110011}{5}%
  \Test{0000011111000001111100000}{10}%
  \Test{000000000000000000000000011111111111111111111}{20}%
\end{qstest}

\begin{qstest}{NextClearBit/NextSetBit}{NextClearBit/NextSetBit}
  \def\Test#1#2{%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \TestOp{abc}{#1}%
    }{#2}%
  }%
  \def\Clear{\let\TestOp\bitsetNextClearBit}%
  \def\Set{\let\TestOp\bitsetNextSetBit}%
  \begingroup
    \catcode`\:=11 %
    \bitsetSetBin{abc}{1}%
    \Clear
    \Test{-1}{1\BitSetError:NegativeIndex}%
    \Set
    \Test{-1}{0\BitSetError:NegativeIndex}%
  \endgroup
  \let\BS@abc\@undefined
  \Clear
  \Test{0}{0}%
  \Test{1}{1}%
  \Test{2}{2}%
  \Test{100}{100}%
  \Set
  \Test{0}{-1}%
  \Test{1}{-1}%
  \Test{100}{-1}%
  \bitsetReset{abc}%
  \Clear
  \Test{0}{0}%
  \Test{1}{1}%
  \Test{2}{2}%
  \Test{100}{100}%
  \Set
  \Test{0}{-1}%
  \Test{1}{-1}%
  \Test{100}{-1}%
  \bitsetSetBin{abc}{1}%
  \Clear
  \Test{0}{1}%
  \Test{1}{1}%
  \Test{2}{2}%
  \Test{100}{100}%
  \Set
  \Test{0}{0}%
  \Test{1}{-1}%
```

```
\Test{100}{-1}%
\bitsetSetBin{abc}{111000111000111000111}%
\Clear
\Test{0}{3}%
\Test{1}{3}%
\Test{2}{3}%
\Test{3}{3}%
\Test{4}{4}%
\Test{5}{5}%
\Test{6}{9}%
\Test{7}{9}%
\Test{8}{9}%
\Test{9}{9}%
\Test{10}{10}%
\Test{11}{11}%
\Test{12}{15}%
\Test{13}{15}%
\Test{14}{15}%
\Test{15}{15}%
\Test{16}{16}%
\Test{17}{17}%
\Test{18}{21}%
\Test{19}{21}%
\Test{20}{21}%
\Test{21}{21}%
\Test{22}{22}%
\Test{100}{100}%
\Set
\Test{0}{0}%
\Test{1}{1}%
\Test{2}{2}%
\Test{3}{6}%
\Test{4}{6}%
\Test{5}{6}%
\Test{6}{6}%
\Test{7}{7}%
\Test{8}{8}%
\Test{9}{12}%
\Test{10}{12}%
\Test{11}{12}%
\Test{12}{12}%
\Test{13}{13}%
\Test{14}{14}%
\Test{15}{18}%
\Test{16}{18}%
\Test{17}{18}%
\Test{18}{18}%
\Test{19}{19}%
\Test{20}{20}%
\Test{21}{-1}%
\Test{22}{-1}%
\Test{100}{-1}%
\bitsetSetBin{abc}{1111111}%
\Clear
\Test{6}{7}%
\Test{7}{7}%
\Test{8}{8}%
\Test{100}{100}%
```

```
  \Set
  \Test{6}{6}%
  \Test{7}{-1}%
  \Test{8}{-1}%
  \Test{100}{-1}%
  \bitsetSetBin{abc}{11111111}%
  \Clear
  \Test{7}{8}%
  \Test{8}{8}%
  \Test{9}{9}%
  \Test{100}{100}%
  \Set
  \Test{7}{7}%
  \Test{8}{-1}%
  \Test{9}{-1}%
  \Test{100}{-1}%
  \bitsetSetBin{abc}{111111111}%
  \Clear
  \Test{8}{9}%
  \Test{9}{9}%
  \Test{10}{10}%
  \Test{100}{100}%
  \Set
  \Test{8}{8}%
  \Test{9}{-1}%
  \Test{10}{-1}%
  \Test{100}{-1}%
  \bitsetSetBin{abc}{1111111111}%
  \Clear
  \Test{9}{10}%
  \Test{10}{10}%
  \Test{11}{11}%
  \Test{100}{100}%
  \Set
  \Test{9}{9}%
  \Test{10}{-1}%
  \Test{11}{-1}%
  \Test{100}{-1}%
\end{qstest}

\begin{qstest}{GetSetBitList}{GetSetBitList}
  \let\BS@abc\@undefined
  \expandafter\expandafter\expandafter\Expect
  \expandafter\expandafter\expandafter{%
    \bitsetGetSetBitList{abc}%
  }{}%
  \def\Test#1#2{%
    \bitsetSetBin{abc}{#1}%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetGetSetBitList{abc}%
    }{#2}%
  }%
  \Test{0}{}%
  \Test{1}{0}%
  \Test{10}{1}%
  \Test{11}{0,1}%
  \Test{10110100}{2,4,5,7}%
  \Test{101101001010011}{0,1,4,6,9,11,12,14}%
\end{qstest}

\begin{qstest}{GetDec}{GetDec}
  \TestGetterUndefined\bitsetGetDec
  \def\Test#1#2{%
    \RevSet{abc}{#1}%
⟨*noetex⟩
    \begingroup\expandafter\expandafter\expandafter\endgroup
⟨/noetex⟩
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetGetDec{abc}%
    }{#2}%
  }%
  \Test{0}{0}%
  \Test{1}{1}%
  \Test{10}{2}%
  \Test{11}{3}%
  \Test{100}{4}%
  \Test{101}{5}%
  \Test{110}{6}%
  \Test{111}{7}%
  \Test{1000}{8}%
  \Test{000111}{7}%
  \Test{1111111111111111%
        111111111111111}{2147483647}%
  \Test{0001111111111111111%
        111111111111111}{2147483647}%
  \Test{1000000000000000%
        0000000000000000}{2147483648}%
  \Test{1000000000000000%
        00000000000000000}{4294967296}%
  \Test{0001000000000000000%
        00000000000000000}{4294967296}%
  \Test{1100000000000000%
        00000000000000011}{6442450947}%
\end{qstest}

\begin{qstest}{Clear}{Clear}
  \def\Test#1#2#3{%
    \RevSet{abc}{#1}%
    \bitsetClear{abc}{#2}%
    \Expect*{\BS@abc}*{\Reverse#3!!}%
  }%
  \bitsetClear{abc}{2}%
  \RevCheck{abc}{0}%
  \TestErrorNegativeIndex{-1}{\bitsetClear{abc}{-1}}%
  \RevCheck{abc}{0}%
  \Test{0}{0}{0}%
  \Test{1}{0}{0}%
  \Test{111}{1}{101}%
  \Test{111}{30}{111}%
  \Test{0000111}{5}{0000111}% 111 would also be ok
  \Test{10000111}{5}{10000111}%
  \Test{1001001}{3}{1000001}%
\end{qstest}

```

```
\begin{qstest}{Set}{Set}
  \def\Test#1#2#3{%
    \RevSet{abc}{#1}%
    \bitsetSet{abc}{#2}%
    \Expect*{\BS@abc}*{\Reverse#3!!}%
  }%
  \bitsetSet{abc}{2}%
  \RevCheck{abc}{100}%
  \TestErrorNegativeIndex{-1}{\bitsetSet{abc}{-1}}%
  \RevCheck{abc}{100}%
  \Test{0}{0}{1}%
  \Test{1}{0}{1}%
  \Test{100}{1}{110}%
  \Test{111}{1}{111}%
  \Test{11}{1}{11}%
  \Test{11}{2}{111}%
  \Test{11}{3}{1011}%
  \Test{111}{10}{10000000111}%
  \Test{0000111}{5}{0100111}% 100111 would also be ok
  \Test{10000111}{5}{10100111}%
  \Test{1000001}{3}{1001001}%
  \Test{1001001}{3}{1001001}%
\end{qstest}

\begin{qstest}{Flip}{Flip}
  \def\Test#1#2#3{%
    \RevSet{abc}{#1}%
    \bitsetFlip{abc}{#2}%
    \Expect*{\BS@abc}*{\Reverse#3!!}%
  }%
  \bitsetFlip{abc}{2}%
  \RevCheck{abc}{100}%
  \TestErrorNegativeIndex{-1}{\bitsetFlip{abc}{-1}}%
  \RevCheck{abc}{100}%
  \Test{0}{0}{1}%
  \Test{1}{0}{0}%
  \Test{0}{2}{100}%
  \Test{100}{1}{110}%
  \Test{111}{1}{101}%
  \Test{11}{1}{1}%
  \Test{11}{2}{111}%
  \Test{11}{3}{1011}%
  \Test{111}{10}{10000000111}%
  \Test{0000111}{5}{0100111}% 100111 would also be ok
  \Test{10000111}{5}{10100111}%
  \Test{1000001}{3}{1001001}%
  \Test{1001001}{3}{1000001}%
  \Test{11111}{2}{11011}%
\end{qstest}

\begin{qstest}{SetValue}{SetValue}
  \def\Test#1#2{%
    \TestError{Invalid bit value (#2) not in range 0..1}{%
      \bitsetSetValue{abc}{#1}{#2}%
    }%
  }%
  \Test{0}{-1}%
  \Test{0}{2}%
  \Test{0}{10}%
  \def\Test#1#2#3{%
    \let\BS@abc\@undefined
    \bitsetSetValue{abc}{#1}{#2}%
    \bitsetSetBin{result}{#3}%
    \Expect*{\BS@abc}*{\BS@result}%
  }%
  \Test{0}{0}{0}%
  \Test{0}{1}{1}%
  \Test{1}{0}{0}%
  \Test{1}{1}{10}%
  \def\Test#1#2#3#4{%
    \bitsetSetBin{abc}{#1}%
    \bitsetSetBin{result}{#4}%
    \bitsetSetValue{abc}{#2}{#3}%
    \Expect*{\BS@abc}*{\BS@result}%
  }%
  \Test{0}{0}{0}{0}%
  \Test{0}{0}{0}{0}%
  \Test{0}{0}{1}{1}%
  \Test{0}{1}{0}{0}%
  \Test{0}{1}{1}{10}%
  \Test{1010}{2}{1}{1110}%
  \Test{1010}{4}{1}{11010}%
  \Test{1010}{6}{1}{1001010}%
  \Test{1010}{1}{0}{1000}%
  \Test{1010}{2}{0}{1010}%
  \Test{1010}{3}{0}{10}%
  \Test{1010}{4}{0}{1010}%
  \Test{1010}{6}{0}{1010}%
  \Test{1010}{2}{\csname iffalse\endcsname 0\else 1\fi}{1110}%
  \Test{1010}{1}{\csname iffalse\endcsname 1\else 0\fi}{1000}%
\end{qstest}

\begin{qstest}{IsDefined}{IsDefined}
  \let\BS@abc\@undefined
  \Expect*{\bitsetIsDefined{abc}{true}{false}}{false}%
  \bitsetReset{abc}%
  \Expect*{\bitsetIsDefined{abc}{true}{false}}{true}%
\end{qstest}

\begin{qstest}{IsEmpty}{IsEmpty}
  \let\BS@abc\@undefined
  \Expect*{\bitsetIsEmpty{abc}{true}{false}}{true}%
  \bitsetReset{abc}%
  \Expect*{\bitsetIsEmpty{abc}{true}{false}}{true}%
  \bitsetSet{abc}{1}%
  \Expect*{\bitsetIsEmpty{abc}{true}{false}}{false}%
\end{qstest}

\begin{qstest}{Equals}{Equals}
  \def\Test#1#2#3{%
    \Expect*{\bitsetEquals{#1}{#2}{true}{false}}{#3}%
  }%
  \let\BS@abc\@undefined
  \Test{abc}{abc}{true}%
  \Test{abc}{foo}{true}%
  \Test{foo}{abc}{true}%
  \bitsetReset{abc}%
  \Test{abc}{abc}{true}%
  \Test{abc}{foo}{false}%
  \Test{foo}{abc}{false}%
  \bitsetReset{foo}%
  \Test{abc}{foo}{true}%
  \Test{foo}{abc}{true}%
  \bitsetSet{abc}{4}%
  \Test{abc}{foo}{false}%
  \Test{foo}{abc}{false}%
  \bitsetFlip{foo}{4}%
  \Test{abc}{foo}{true}%
  \Test{foo}{abc}{true}%
\end{qstest}

\begin{qstest}{Intersects}{Intersects}
  \def\Test#1{%
    \Expect*{\bitsetIntersects{abc}{foo}{true}{false}}{#1}%
  }%
  \let\BS@abc\@undefined
  \let\BS@foo\@undefined
  \Test{false}%
  \Set{abc}{0}%
  \Test{false}%
  \Set{foo}{0}%
  \Test{false}%
  \let\BS@abc\@undefined
  \Test{false}%
  \Set{foo}{1}%
  \Test{false}%
  \Set{abc}{0}%
  \Test{false}%
  \Set{abc}{1}%
  \Test{true}%
  \let\BS@foo\@undefined
  \Test{false}%
  \Set{foo}{0}%
  \Test{false}%
  \def\Test#1#2#3{%
    \bitsetSetBin{abc}{#1}%
    \bitsetSetBin{foo}{#2}%
    \Expect*{\bitsetIntersects{abc}{foo}{true}{false}}{#3}%
  }%
  \Test{1010}{0101}{false}%
  \Test{0}{10}{false}%
  \Test{1}{11}{true}%
  \Test{11}{1}{true}%
  \Test{10}{1}{false}%
\end{qstest}

\begin{qstest}{And/AndNot/Or/Xor}{And/AndNot/Or/Xor}
  \def\@Test#1#2#3#4#5{%
    \begingroup
      #5%
      \begingroup
        \let\BS@foo\@undefined
        \csname bitset#1\endcsname{abc}{foo}%
        \CheckUndef{foo}%
        \Check{abc}{#2}%
      \endgroup
      \begingroup
        \bitsetReset{foo}%
        \csname bitset#1\endcsname{abc}{foo}%
        \Check{foo}{0}%
        \Check{abc}{#3}%
      \endgroup
      \begingroup
        \def\BS@foo{0101}%
        \csname bitset#1\endcsname{abc}{foo}%
        \Check{foo}{0101}%
        \Check{abc}{#4}%
      \endgroup
    \endgroup
  }%
  \def\Test#1{%
    \def\Op{#1}%
    \Test@
  }%
  \def\Test@#1#2#3#4#5#6#7#8#9{%
    \@Test\Op{#1}{#2}{#3}{%
      \let\BS@abc\@undefined
    }%
    \@Test\Op{#4}{#5}{#6}{%
      \bitsetReset{abc}%
    }%
    \@Test\Op{#7}{#8}{#9}{%
      \def\BS@abc{1001}%
    }%
  }%
  \Test{And}%
       {0}{0}{0}%
       {0}{0}{0}%
       {0}{0}{0001}%
  \Test{AndNot}%
       {0}{0}{0}%
       {0}{0}{0}%
       {1001}{1001}{1}%
  \Test{Or}%
       {0}{0}{0101}%
       {0}{0}{0101}%
       {1001}{1001}{1101}%
  \Test{Xor}%
       {0}{0}{0101}%
       {0}{0}{0101}%
       {1001}{1001}{11}%
  \def\Test#1#2#3{%
    \bitsetSetBin{abc}{#1}%
    \bitsetSetBin{foo}{#2}%
    \csname bitset\Op\endcsname{abc}{foo}%
    \RevCheck{foo}{#2}%
    \RevCheck{abc}{#3}%
  }%
  \def\Op{And}%
  \Test{1}{111}{1}%
  \Test{111}{1}{1}%
  \Test{10}{111}{10}%
```

```
  \Test{111}{10}{10}%
  \Test{111}{1000}{0}%
  \Test{1000}{111}{0}%
  \def\Op{AndNot}%
  \Test{1010}{11}{1000}%
  \Test{100}{100}{0}%
  \Test{111}{1111}{0}%
  \Test{100}{111}{0}%
  \def\Op{Or}%
  \Test{0}{0}{0}%
  \Test{1}{0}{1}%
  \Test{0}{1}{1}%
  \Test{1}{1}{1}%
  \Test{1000}{10}{1010}%
  \Test{10}{1000}{1010}%
  \def\Op{Xor}%
  \Test{0}{0}{0}%
  \Test{1}{0}{1}%
  \Test{0}{1}{1}%
  \Test{1}{1}{0}%
  \Test{1000}{10}{1010}%
  \Test{10}{1000}{1010}%
  \Test   {110011001100}%
       {111000111000111}%
       {111110100001011}%
  \Test{111000111000111}%
          {110011001100}%
       {111110100001011}%
\end{qstest}

\begin{qstest}{GetUndef}{GetUndef, GetBin, GetOct, GetHex}
  \def\TestUndef#1#2{%
    \let\BS@abc\@undefined
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \x{abc}{#1}%
    }{#2}%
  }%
  \let\x\bitsetGetBin
  \TestUndef{-1}{0}%
  \TestUndef{0}{0}%
  \TestUndef{1}{0}%
  \TestUndef{2}{00}%
  \TestUndef{8}{00000000}%
  \let\x\bitsetGetOct
  \TestUndef{-1}{0}%
  \TestUndef{0}{0}%
  \TestUndef{1}{0}%
  \TestUndef{2}{0}%
  \TestUndef{3}{0}%
  \TestUndef{4}{00}%
  \TestUndef{5}{00}%
  \TestUndef{6}{00}%
  \TestUndef{7}{000}%
  \TestUndef{8}{000}%
  \TestUndef{9}{000}%
  \TestUndef{10}{0000}%
  \let\x\bitsetGetHex
  \TestUndef{-1}{0}%
  \TestUndef{0}{0}%
  \TestUndef{1}{0}%
  \TestUndef{2}{0}%
  \TestUndef{3}{0}%
  \TestUndef{4}{0}%
  \TestUndef{5}{00}%
  \TestUndef{6}{00}%
  \TestUndef{7}{00}%
  \TestUndef{8}{00}%
  \TestUndef{9}{000}%
  \TestUndef{10}{000}%
  \TestUndef{12}{000}%
  \TestUndef{13}{0000}%
  \TestUndef{16}{0000}%
  \TestUndef{17}{00000}%
\end{qstest}

\begin{qstest}{SetBin}{SetBin}
  \def\Test#1#2{%
    \let\BS@abc\@undefined
    \bitsetSetBin{abc}{#1}%
    \expandafter\Expect\expandafter{\BS@abc}{#2}%
  }%
  \Test{}{0}%
  \Test{0}{0}%
  \Test{1}{1}%
  \Test{10}{01}%
  \Test{11}{11}%
  \Test{010}{01}%
  \Test{011}{11}%
  \Test{0010}{01}%
  \Test{1010}{0101}%
\end{qstest}

\begin{qstest}{SetOct}{SetOct}
  \def\Test#1#2{%
    \bitsetSetOct{abc}{#1}%
    \expandafter\Expect\expandafter{\BS@abc}{#2}%
  }%
  \Test{}{0}%
  \Test{0}{0}%
  \Test{000}{0}%
  \Test{1}{1}%
  \Test{001}{1}%
  \Test{010}{0001}%
  \Test{020}{00001}%
  \Test{42}{010001}%
  \Test{377}{11111111}%
  \Test{0377}{11111111}%
  \Test{76543210}{000100010110001101011111}%
  \Test{ 0 7 0 7 1 }{100111000111}%
\end{qstest}

\begin{qstest}{SetHex}{SetHex}
  \def\Test#1#2{%
    \bitsetSetHex{abc}{#1}%
    \expandafter\Expect\expandafter{\BS@abc}{#2}%
  }%
  \Test{}{0}%
  \Test{0}{0}%
  \Test{000}{0}%
  \Test{1}{1}%
  \Test{001}{1}%
  \Test{010}{00001}%
  \Test{020}{000001}%
  \Test{42}{0100001}%
  \Test{3F}{111111}%
  \Test{03F}{111111}%
  \Test{43210}{0000100001001100001}%
  \Test{98765}{10100110111000011001}%
  \Test{FEDCBA}{010111010011101101111111}%
  \Test{ 0 F 0 F 1 }{1000111100001111}%
\end{qstest}

\begin{qstest}{SetDec}{SetDec}
  \def\Test#1#2{%
    \bitsetSetDec{abc}{#1}%
    \expandafter\Expect\expandafter{\BS@abc}{#2}%
  }%
  \Test{}{0}%
  \Test{0}{0}%
  \Test{000}{0}%
  \Test{1}{1}%
  \Test{7}{111}%
  \Test{8}{0001}%
  \Test{001}{1}%
  \Test{010}{0101}%
  \Test{020}{00101}%
  \Test{53}{101011}%
  \Test{255}{11111111}%
  \Test{256}{000000001}%
  \Test{999999999}{111111111001001101011001110111}%
  \Test{1000000000}{000000000101001101011001110111}%
  \Test{4210987654}{01100001010010010111111101011111}%
  \Test{2147483647}{1111111111111111111111111111111}%
  \Test{2147483648}{00000000000000000000000000000001}%
\end{qstest}

\begin{qstest}{GetBin}{GetBin}
  \def\TestUndef#1#2{%
    \let\BS@abc\@undefined
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetGetBin{abc}{#1}%
    }{#2}%
  }%
  \TestUndef{-1}{0}%
  \TestUndef{0}{0}%
  \TestUndef{1}{0}%
  \TestUndef{2}{00}%
  \TestUndef{8}{00000000}%
  \def\Test#1#2{%
    \bitsetSetBin{abc}{#2}%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetGetBin{abc}{#1}%
    }{#2}%
  }%
  \Test{-1}{0}%
  \Test{0}{0}%
  \Test{1}{0}%
  \Test{1}{1}%
  \Test{2}{01}%
  \Test{2}{10}%
  \Test{3}{010}%
  \Test{2}{00}%
  \Test{2}{01}%
  \Test{8}{00101100}%
  \Test{2}{10101}%
  \Test{-100}{11011}%
\end{qstest}

\begin{qstest}{GetOct}{GetOct}
  \def\Test#1#2#3{%
    \edef\x{\zap@space#1 \@empty}%
    \edef\x{\noexpand\bitsetSetBin{abc}{\x}}%
    \x
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetGetOct{abc}{#2}%
    }{#3}%
  }%
  \Test{111 110 101 100 011 010 001 000}{0}{76543210}%
  \Test{000 111}{0}{7}%
  \Test{101 000}{-1}{50}%
  \Test{111}{-1}{7}%
  \Test{111}{0}{7}%
  \Test{111}{1}{7}%
  \Test{111}{3}{7}%
  \Test{111}{4}{07}%
  \Test{111}{6}{07}%
  \Test{111}{7}{007}%
  \Test{111 010}{6}{72}%
  \Test{111 010}{7}{072}%
  \Test{011 111}{0}{37}%
  \Test{011 111}{6}{37}%
  \Test{011 111}{7}{037}%
  \Test{001 111}{0}{17}%
  \Test{001 111}{6}{17}%
  \Test{001 111}{7}{017}%
\end{qstest}

\begin{qstest}{GetHex}{GetHex}
  \def\Test#1#2#3{%
    \bitsetSetBin{abc}{#1}%
    \expandafter\expandafter\expandafter\Expect
    \expandafter\expandafter\expandafter{%
      \bitsetGetHex{abc}{#2}%
    }{#3}%
  }%
  \Test{1111 1110 1101 1100 1011 1010 1001 1000}{0}{FEDCBA98}%
  \Test{0111 0110 0101 0100 0011 0010 0001 0000}{0}{76543210}%
  \Test{0000 1111}{0}{F}%
  \Test{0101 0000}{-1}{50}%
  \Test{1111}{-1}{F}%
  \Test{1111}{0}{F}%
  \Test{1111}{1}{F}%
  \Test{1111}{4}{F}%
  \Test{1111}{5}{0F}%
  \Test{1111}{8}{0F}%
  \Test{1111}{9}{00F}%
  \Test{1111 0010}{8}{F2}%
  \Test{1111 0010}{9}{0F2}%
  \Test{0111 1111}{0}{7F}%
  \Test{0111 1111}{8}{7F}%
  \Test{0111 1111}{9}{07F}%
  \Test{0011 1111}{0}{3F}%
  \Test{0011 1111}{8}{3F}%
  \Test{0011 1111}{9}{03F}%
  \Test{0001 1111}{0}{1F}%
  \Test{0001 1111}{8}{1F}%
  \Test{0001 1111}{9}{01F}%
\end{qstest}

\begin{qstest}{Range}{Range}
  \TestError{%
    Wrong index numbers in range [9..8]\MessageBreak% hash-ok
    for clear/set/flip on bit set ‘abc’.\MessageBreak
    The lower index exceeds the upper index.\MessageBreak
    Canceling the operation as error recovery%
  }{%
    \bitsetSetRange{abc}{9}{8}%
  }%
  \def\TestErrorNegInd#1#2#3#4#5#6{%
    \TestError{%
      Negative index in range [#2..#3]\MessageBreak % hash-ok
      for \string\bitset #1Range on bit set ‘abc’.\MessageBreak
      Using [#4..#5] as error recovery% hash-ok
    }{%
      \csname bitset#1Range\endcsname{abc}{#2}{#3}%
      \global\let\BS@global\BS@abc
    }%
    \Check{global}{#6}%
  }%
  \Set{abc}{111}%
  \TestErrorNegInd{Clear}{-1}{0}{0}{0}{111}%
  \TestErrorNegInd{Clear}{0}{-1}{0}{0}{111}%
  \TestErrorNegInd{Clear}{-2}{2}{0}{2}{001}%
  \bitsetReset{abc}%
  \TestErrorNegInd{Set}{-1}{0}{0}{0}{0}%
  \TestErrorNegInd{Set}{0}{-1}{0}{0}{0}%
  \TestErrorNegInd{Set}{-2}{2}{0}{2}{11}%
  \Set{abc}{101}%
  \TestErrorNegInd{Flip}{-1}{0}{0}{0}{101}%
  \TestErrorNegInd{Flip}{0}{-1}{0}{0}{101}%
  \TestErrorNegInd{Flip}{-2}{2}{0}{2}{011}%
  \def\Test#1#2#3#4{%
    \bitsetSetBin{abc}{#1}%
    \csname bitset\TestOp Range\endcsname{abc}{#2}{#3}%
    \Expect*{\bitsetGetBin{abc}{0}}{#4}%
  }%
  \def\TestOp{Clear}%
  \Test{0}{0}{1}{0}%
  \Test{1111}{1}{2}{1101}%
  \Test{1111}{1}{3}{1001}%
  \Test{1111111100000000}{12}{14}{1100111100000000}%
  \def\TestOp{Set}%
  \Test{0}{0}{1}{1}%
  \Test{1000}{1}{2}{1010}%
  \Test{0}{1}{2}{10}%
  \Test{1}{12}{15}{1110000000000001}%
  \Test{1111}{1}{3}{1111}%
  \Test{1000000000000000}{12}{14}{1011000000000000}%
  \def\TestOp{Flip}%
  \Test{0}{0}{1}{1}%
  \Test{1}{0}{1}{0}%
  \Test{10101010}{1}{5}{10110100}%
  \def\Test#1#2#3#4#5{%
    \bitsetSetBin{abc}{#1}%
    \bitsetSetValueRange{abc}{#2}{#3}{#4}%
    \Expect*{\bitsetGetBin{abc}{0}}{#5}%
  }%
  \Test{0}{0}{1}{0}{0}%
  \Test{0}{0}{1}{1}{1}%
  \Test{1010}{1}{3}{0}{1000}%
  \Test{1010}{1}{3}{1}{1110}%
\end{qstest}

\begin{qstest}{ShiftLeft/ShiftRight}{ShiftLeft/ShiftRight}
  \def\@Test#1#2{%
    \let\BS@abc\@undefined
    \csname bitsetShift#1\endcsname{abc}{#2}%
    \Expect*{\BS@abc}{0}%
  }%
  \def\Test#1{%
    \@Test{Left}{#1}%
    \@Test{Right}{#1}%
  }%
  \Test{-16}%
  \Test{-1}%
  \Test{0}%
  \Test{1}%
  \Test{16}%
  \def\Test#1#2#3{%
    \bitsetSetBin{abc}{#1}%
    \bitsetSetBin{result}{#3}%
    \csname bitsetShift\Op\endcsname{abc}{#2}%
    \Expect*{\bitsetGetBin{abc}{0}}*{\bitsetGetBin{result}{0}}%
  }%
  \def\Op{Left}%
  \Test{0}{0}{0}%
  \Test{0}{1}{0}%
  \Test{0}{-1}{0}%
  \Test{1}{0}{1}%
  \Test{1}{1}{10}%
  \Test{1}{-1}{0}%
  \Test{10}{1}{100}%
  \Test{10}{-1}{1}%
  \Test{1}{32}{100000000000000000000000000000000}%
  \Test{1}{-100}{0}%
  \def\Op{Right}%
  \Test{0}{0}{0}%
  \Test{0}{1}{0}%
  \Test{0}{-1}{0}%
  \Test{1}{0}{1}%
  \Test{1}{1}{0}%
  \Test{1}{-1}{10}%
  \Test{10}{1}{1}%
  \Test{10}{-1}{100}%
  \Test{1}{-32}{100000000000000000000000000000000}%
  \Test{1}{100}{0}%
  \Test{110110110110110}{10}{11011}%
  \Test{110110110110110}{100}{0}%
  \Test{1}{100000}{0}%
\end{qstest}

\begin{qstest}{Profile: Set}{Profile: Set}
  \bitsetSet{abc}{4095}%
  \global\let\BS@global\BS@abc
\end{qstest}

\begin{qstest}{Profile: Get}{Profile: Get}
  \edef\x{\bitsetGet{global}{4095}}%
\end{qstest}

\begin{document}
\end{document}
⟨/test2⟩
```

4 Installation

4.1 Download

Package. This package is available on CTAN[1]:

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 TEX Files" (CTAN:tds/tds.pdf). Directories with `texmf` in their name are usually organized this way.

4.2 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
```

[1] http://ctan.org/pkg/bitset

Script installation. Check the directory `TDS:scripts/oberdiek/` for scripts that need further installation steps. Package attachfile2 comes with the Perl script `pdfatfi.pl` that should be installed in such a way that it can be called as `pdfatfi`. Example (linux):

```
chmod +x scripts/oberdiek/pdfatfi.pl
cp scripts/oberdiek/pdfatfi.pl /usr/local/bin/
```

4.3 Package installation

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

```
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
test/bitset-test1.tex → doc/latex/oberdiek/test/bitset-test1.tex
test/bitset-test2.tex → doc/latex/oberdiek/test/bitset-test2.tex
test/bitset-test3.tex → doc/latex/oberdiek/test/bitset-test3.tex
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.

4.4 Refresh file name databases

If your TEX distribution (teTEX, mikTEX, . . .) relies on file name databases, you must refresh these. For example, teTEX users run `texhash` or `mktexlsr`.

4.5 Some details for the interested

Attached source. The PDF documentation on CTAN also includes the `.dtx` source file. It can be extracted by AcrobatReader 6 or higher. Another option is pdftk, e.g. unpack the file into the current directory:

```
pdftk bitset.pdf unpack_files output .
```

Unpacking with LATEX. The `.dtx` chooses its action depending on the format:

plain TEX: Run docstrip and extract the files.

LATEX: Generate the documentation.

If you insist on using LATEX for docstrip (really, docstrip does not need LATEX), 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 pdfLaTeX:

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

5 Catalogue

The following XML file can be used as source for the TeX Catalogue. The elements `caption` and `description` are imported from the original XML file from the Catalogue. The name of the XML file in the Catalogue is `bitset.xml`.

```
⟨*catalogue⟩
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE entry SYSTEM 'catalogue.dtd'>
<entry datestamp='$Date$' modifier='$Author$' id='bitset'>
  <name>bitset</name>
  <caption>Handle bit-vector datatype.</caption>
  <authorref id='auth:oberdiek'/>
  <copyright owner='Heiko Oberdiek' year='2007,2011'/>
  <license type='lppl1.3'/>
  <version number='1.2'/>
  <description>
    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.
    <p/>
    The package is part of the <xref refid='oberdiek'>oberdiek</xref> bundle.
  </description>
  <documentation details='Package documentation'
      href='ctan:/macros/latex/contrib/oberdiek/bitset.pdf'/>
  <ctan file='true' path='/macros/latex/contrib/oberdiek/bitset.dtx'/>
  <miktex location='oberdiek'/>
  <texlive location='oberdiek'/>
  <install path='/macros/latex/contrib/oberdiek/oberdiek.tds.zip'/>
</entry>
⟨/catalogue⟩
```

6 History

[2007/09/28 v1.0]

- First version.

[2011/01/30 v1.1]

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

[2016/05/16 v1.2]

- Documentation updates.

7 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.