bitfield

COLLABORATORS			
	TITLE : bitfield		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 9, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

bitfield

1.1 bitfield.doc

```
--overview--

bit_operations()

bitfield_combine()

bitfield_operations()

end()

field_operations()

field_range()

new()

range_errors()
```

1.2 bitfield.m/--overview--

```
 PURPOSE
To provide a multi-purpose bitfield.

 OVERVIEW
Bitfields  are a simple way of encoding a set of true/false states
for  a  number  of integer values. Rather than use a lookup table,
you can use a bitfield which holds 8 states per byte.

Basically, the bitfield is a collection of individual 'bits', each
of which is a simple boolean value, TRUE or FALSE.

You  can  set,  clear,  and test individual bit and ranges of bits
within  the bitfield. You can ask the bitfield either to ignore or
raise  exceptions  if you step outside of it. You can also combine
```

```
bitfields logically.
```

1.3 bitfield.m/bit_operations

```
                  NAME
bitfield.set() -- set an individual bit.
bitfield.clear() -- clear an individual bit.
bitfield.toggle() -- toggle an individual bit.
bitfield.test() -- test an individual bit.

 SYNOPSIS
state := set(bit)
state := clear(bit)
state := toggle(bit)
state := test(bit)

 FUNCTION
Will  test,  then perform an operation on an individual bit in the
bitfield:

set()    will set the bit to boolean TRUE.
clear()  will clear the bit to boolean FALSE.
toggle() will change a boolean FALSE bit to TRUE, and vice-versa.
test()   will perform no altering operation on the bit.

If  the bit specified is outwith the range stored by the bitfield,
a range error will occur.

 INPUTS
bit   - the bit to perform an operation on.

 RESULT
state - the  previous  state  of  the bit before the operation was
        performed on it, either TRUE or FALSE.

 SEE ALSO

              bitfield_operations()
              ,
              range_errors()
```

1.4 bitfield.m/bitfield_combine

```
                  NAME
bitfield.copy() -- copy from another bitfield.
bitfield.and() -- mask bitfield with another.
bitfield.or() -- overlay from another bitfield.
bitfield.xor() -- perform exclusive-or combine of bitfield.

 SYNOPSIS
bitfield.copy(bitfield2)
```

```
bitfield.and(bitfield2)
bitfield.or(bitfield2)
bitfield.xor(bitfield2)

 FUNCTION
Will  perform  a  logical operation on the bitfield using the data
held in another bitfield.

Currently,  both  bitfields  must be exactly the same size as each
other, or a range error will occur.

b1.copy(b2) will perform  b1 := b2
b1.and(b2)  will perform  b1 := b1 AND b2
b1.or(b2)   will perform  b1 := b1 OR b2
b1.xor(b2)  will perform  b1 := b1 XOR b2

If  the  bitfield  specified  falls  in  any way outwith the range
stored by the bitfield, a range error will occur.

 INPUTS
bitfield2 - pointer to another instance of the bitfield class.

 RESULT
Always returns TRUE, except if a range error occurs, in which case
it will return FALSE.

 SEE ALSO

              range_errors()
```

1.5 bitfield.m/bitfield_operations

```
                 NAME
bitfield.bf_set() -- set a range of bits.
bitfield.bf_clear() -- clear a range of bits.
bitfield.bf_invert() -- invert a range of bits.
bitfield.bf_test() -- test a range of bit.

 SYNOPSIS
andstate, orstate := bf_set(leftbit, rightbit)
andstate, orstate := bf_clear(leftbit, rightbit)
andstate, orstate := bf_invert(leftbit, rightbit)
andstate, orstate := bf_test(leftbit, rightbit)

 FUNCTION
Will  test all of, then perform an operation on a range of bits in
the bitfield:

bf_set()    will set all affected bits to boolean TRUE.
bf_clear()  will clear all affected bits to boolean FALSE.
bf_invert() will change boolean FALSE bits to TRUE, and vice-versa.
bf_test()   will perform no altering operation on the bits.

If  the  bitfield  specified  falls  in  any way outwith the range
```

```
stored by the bitfield, a range error will occur.

 INPUTS
leftbit, rightbit - the  bits you  want to affect, from leftbit to
                    rightbit  (inclusive).  left and right will be
                    swapped automatically if right < left.

 RESULT
andstate - an  AND-based  combination  of  the  states  of all the
           affected bits before any changes were made.

           If ALL of the bits were TRUE, the andstate is TRUE
           If any of the bits were FALSE, the andstate is FALSE.

orstate  - an  OR-based  combination  of  the  states  of  all the
           affected bits before any changes were made.

           If ANY of the bits were TRUE, the orstate is TRUE
           If all of the bits were FALSE, the orstate is FALSE.

 SEE ALSO

              bit_operations()
              ,
              range_errors()
```

1.6 bitfield.m/end

```
                  NAME
bitfield.end() -- Destructor.

 SYNOPSIS
end()

 FUNCTION
Frees resources used by an instance of the bitfield class.

 SEE ALSO

              new()
```

1.7 bitfield.m/field_operations

```
                  NAME
bitfield.setfield() -- set all bits.
bitfield.clearfield() -- clear all bits.
bitfield.invert() -- invert all bits.

 SYNOPSIS
setfield()
```

```
clearfield()
invert()

 FUNCTION
setfield() sets all bits in the bitfield to boolean TRUE.
clearfield() clears all bits in the bitfield to boolean FALSE.
invert() switches all bits to their opposite boolean value.

These  functions are more optimised than using the ranged bitfield
operations over the entire range.

 SEE ALSO

              bitfield_operations()
```

1.8 bitfield.m/field_range

```
                  NAME
bitfield.range() -- report range of representable integers.

 SYNOPSIS
min, max := range()

 FUNCTION
Returns  the  minimum  and  maximum  represented  integers in this
instance, as defined in the construction.

 RESULT
min - the minimum representable integer in this bitfield.
max - the maximum representable integer in this bitfield.

 SEE ALSO

              new()
```

1.9 bitfield.m/new

```
                  NAME
bitfield.new() -- Constructor.

 SYNOPSIS
new(min, max)
new(min, max, range)

 FUNCTION
Initialises  an  instance  of the bitfield class. Raises exception
"MEM"  if it cannot allocate enough memory for the required number
of bits. All bits are initially cleared.

 INPUT
```

```
min   - the  minimum integer value that will be represented in the
        field. This can be negative.

max   - the  maximum integer value that will be represented in the
        field.  This  can  also  be negative, and can also be less
        than min (the values will be swapped if they are).

range - whether range errors are fatal or not. See
              range_errors()
              .
        The default for this argument is FALSE.

 SEE ALSO

              end()
              , range(),
              range_errors()
```

1.10 bitfield.m/range_errors

```
                  NAME
bitfield.range_errors() -- define range error handling.

 SYNOPSIS
oldstate := range_errors(newstate)

 FUNCTION
Defines  the  handling  of  range errors with this instance of the
bitfield.

A  range  error  occurs  when  you  try  an  operation that cannot
logically  perform  its  function  with  the  parameters  you have
specified. For example, trying to toggle a bit that is outside the
range of the bitfield, or trying to combine two bitfields that are
not the same size.

When  a range error occurs, what happens next depends on the state
of  the range_errors flag for the bitfield instance, as set in the
construction, or at runtime with this function.

If  range_errors  =  FALSE,  NO  OPERATION  WILL  OCCUR,  and  the
operation will immediately return FALSE.

If range_errors = TRUE, NO OPERATION WILL OCCUR, and the operation
will throw the exception "rnge".

It is essential to realise that NO operation happens when there is
a  range  error. range_errors=FALSE does NOT mean that 'errors are
ignored'. It means that 'errors do not throw an exception'.

An  analogy  is  that  range_errors=TRUE  makes  errors  'loud' or
'fatal'.

 INPUTS
```

```
newstate - a TRUE or FALSE value to set the range_errors flag.

 RESULT
oldstate - the previous value of range_errors flag.

 SEE ALSO

              new()
              , range(),
              bitfield_operations()
              ,
              bit_operations()
              ,

              bitfield_combine()
```