# int64 : 64 bits integer vectors

Romain François - `romain@r-enthusiasts.com`

**int64** version 1.1.2

**Abstract**

The `int64` package adds 64 bit integer vectors to `R`. The package provides the `int64` and `uint64` classes for signed and unsigned integer vectors. This project has been sponsored by the Google Open Source Programs Office.

## 1 Background

Integers in `R` are represented internally as 32 bit `int`. Aplications now require larger ranges of values to represent large quantities. This package exposes C++ types `int64_t` and `uint64_t` to `R` for this purpose. The table 1 shows the limits of these types.

| C++ type | R type | min | max |
|---|---|---:|---:|
| `int` | `integer` | `-2147483647` | `2147483647` |
| `int64_t` | `int64` | `-9223372036854775807` | `9223372036854775807` |
| `uint64_t` | `uint64` | `0` | `18446744073709551614` |

Table 1: Numeric limits of integer types

## 2 Usage

This section shows a few examples on how to use the package.

```
> # create a new int64 vector
> x <- int64( 4 )
> # set a subset of values
> x[1:2] <- 1:2 # via integers
> x[3:4] <- c("123456789123456", "-9876543219876") # ... or characters
> x

[1] 1               2               123456789123456 -9876543219876

> # convert integer or character vectors into int64 vectors
> x <- as.int64( 1:6 )
> x

[1] 1 2 3 4 5 6

> y <- as.int64( c("-1234", "1234" ) )
> y

[1] -1234 1234

> # create a data frame with a column of int64
> df <- data.frame( a = 1:4 )
> df$y <- as.int64( 1:4 )
> df
```

```
  a y
1 1 1
2 2 2
3 3 3
4 4 4
```

# 3 The int64 and uint64 classes

## 3.1 Class representation

Both `int64` and `uint64` are represented as lists of pairs of integers.

```
> str( as.int64( 1:2 ) )
```

```
 int64 [1:2] 1 2 ...
```

Each int64 or uint64 number is represented as a couple of 32 bit integers. Internally, the C++ code goes back and forth between the native representation of these numbers as C++ data types (`int64_t` and `uint64_t`) and their representation as couples of 32 bit integers by splitting the 64 bits.

For example, the `int64_t` value (-123) is represented in memory as:

| |
|---|
| `1111111111111111111111111111111111111111111111111111111110000101` |

These 64 bits are split into the two following chunks:

| | |
|---|---|
| `11111111111111111111111111111111` | `11111111111111111111111110000101` |

The R representation of -123 is therefore composed by the two integers whose binary representation is above, i.e (-1,-123). This representation has been chosen against other alternatives to allow these key requirements:

- Data must be serializable
- int64 and uint64 vectors have to be usable of columns of data frames.
- The int64 and uint64 types must supposrt missing values (NA)

## 3.2 Creating new vectors

The functions `int64` and `uint64` can be used to create new vectors of signed or usigned 64 bit integers of the given length. These functions are similar to the usual `R` functions `numeric`, `integer`, etc ...

```
> int64(3)
```

```
[1] 0 0 0
```

```
> uint64(10)
```

```
 [1] 0 0 0 0 0 0 0 0 0 0
```

## 3.3 Converting integer or character vectors

The functions `as.int64` and `as.uint64` can be used to convert `integer` or `character` vectors into signed or unsigned 64 bit integers.

```
> as.int64( 1:4 )
```

```
[1] 1 2 3 4
```

```
> as.uint64( c("123456789", "987654321987654321" ) )
```

```
[1] 123456789          987654321987654321
```

Internally `integer` vectors are converted using a reguar cast, and `character` vectors are converted using the `C` function `atol`.

## 3.4 Subsetting

Extracting or setting subsets from a `int64` or `uint64` vector is similar to other vector classes in R.

```
> x <- as.int64( 1:4 )
> x[1:2]
```

```
[1] 1 2
```

```
> x[3:4] <- 5:6
> x
```

```
[1] 1 2 5 6
```

## 3.5 Arithmetic operations

The `Arith` group generic is implemented for classes `int64` and `uint64`.

```
> x <- as.int64( 1:4 )
> x + 1L
```

```
[1] 2 3 4 5
```

```
> x - 1:2
```

```
[1] 0 0 2 2
```

```
> x * x
```

```
[1] 1  4  9  16
```

```
> x / 2L
```

```
[1] 0 1 1 2
```

```
> x %% 2L
```

```
[1] 1 0 1 0
```

```
> x %/% 2L
```

```
[1] 0 1 1 2
```

## 3.6 Logical operations

The `Compare` group generic is implemented for classes `int64` and `uint64`.

```
> x <- as.int64( 1:5 )
> x < 3L
```

```
[1]  TRUE  TRUE FALSE FALSE FALSE
```

```
> x > 6L - x
```

```
[1] FALSE FALSE FALSE  TRUE  TRUE
```

```
> x != 3L
```

```
[1]  TRUE  TRUE FALSE  TRUE  TRUE
```

```
> x == 4L
```

```
[1] FALSE FALSE FALSE  TRUE FALSE
```

```
> x <= 3L
```

```
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

```
> x >= 5L
```

```
[1] FALSE FALSE FALSE FALSE  TRUE
```

### 3.7 Summary operations

The `Summary` group generic is implemented for classes `int64` and `uint64`.

```
> x <- as.int64( 1:5 )
> min( x )

[1] 1

> max( x )

[1] 5

> range( x )

[1] 1 5

> prod( x )

[1] 120

> sum( x )

[1] 15

> any( x )

[1] TRUE

> all( x )

[1] TRUE
```

## 4 Binary representation

The `binary` generic function shows the bit representation of `numeric`, `integer`, `int64` and `uint64`.

```
> binary( 1:4 ) # integer

[1] 00000000000000000000000000000001 00000000000000000000000000000010
[3] 00000000000000000000000000000011 00000000000000000000000000000100

> binary( c(1.2, 1.3) ) # numeric

[1] 0011111111110011001100110011001100110011001100110011001100110011
[2] 0011111111110100110011001100110011001100110011001100110011001101

> binary( as.int64( 1:4 ) ) # signed 64 bit integer (int64)

[1] 0000000000000000000000000000000000000000000000000000000000000001
[2] 0000000000000000000000000000000000000000000000000000000000000010
[3] 0000000000000000000000000000000000000000000000000000000000000011
[4] 0000000000000000000000000000000000000000000000000000000000000100

> binary( as.uint64( 1:4 ) ) # unsigned 64 bit integer (uint64)

[1] 0000000000000000000000000000000000000000000000000000000000000001
[2] 0000000000000000000000000000000000000000000000000000000000000010
[3] 0000000000000000000000000000000000000000000000000000000000000011
[4] 0000000000000000000000000000000000000000000000000000000000000100
```

## 5 Numeric limits and missing values

The `numeric_limits` function gives the limits for types `integer`, `int64`, `uint64`.

```
> numeric_limits( "integer" )
```

```
[1] -2147483647  2147483647
```

```
> numeric_limits( "int64" )
```

```
[1] -9223372036854775807 9223372036854775807
```

```
> numeric_limits( "uint64" )
```

```
[1] 0                    18446744073709551614
```

int64 and uint64 classes support missing values using the same mechanism as R uses for integer vectors.

For signed 64 bit integer vectors (int64), NA is represented by the value $-2^{63}$, hence the range of acceptable values is

$$[-2^{63}+1, 2^{63}-1]$$

For unsigned 64 bit integer vectors (uint64), NA is represented by the value $2^{64}-1$, hence the range of acceptable values is

$$[0, 2^{64}-2]$$

## 6 Reading 64 bit integers from files

The `int64` implements the necessary methods so that `read.csv` can read signed and unsigned 64 bit integers from files.

```
> tf <- tempfile()
> df <- data.frame( x = 1:10, y = 1:10, z = 1:10 )
> write.table( df, tf, sep = ",", row.names = FALSE )
> df <- read.csv( tf, colClasses = c("integer", "int64", "uint64" ) )
> df
```

```
    x  y  z
1   1  1  1
2   2  2  2
3   3  3  3
4   4  4  4
5   5  5  5
6   6  6  6
7   7  7  7
8   8  8  8
9   9  9  9
10 10 10 10
```

```
> sapply( df, class )
```

```
        x         y         z
"integer"   "int64"  "uint64"
```