Package ‘rmongodb’

September 3, 2011

Type Package

Title R-MongoDB driver

Version 1.0

Date 2011-06-27

Author Gerald Lindsly

Maintainer Gerald Lindsly <gerald.lindsly@gmail.com>

Description Provides an interface to MongoDB for R

License GPL (>=2)

LazyLoad yes

Archs i386, x64

R topics documented:

`rmongodb-package` *R-MongoDB driver*

Description

Provides an interface to MongoDB for R

Details

Package:	rmongodb
Type:	Package
Version:	1.0
Date:	2011-06-27
License:	GPL (>=2)
LazyLoad:	yes

Overview

Author(s)

Gerald Lindsly

Maintainer: Gerald Lindsly gerald.lindsly@gmail.com

References

http://www.mongodb.org

See Also

mongo

`as.character.mongo.oid` *Convert a mongo.oid object to a string*

Description

Convert a mongo.oid object to a string of 24 hex digits. This performs the inverse operation of `mongo.oid.from.string()`.

This function is an alias of `mongo.oid.to.string()` so that the class mechanism of R allows it to be called simply by `as.character(oid)`.

See `http://www.mongodb.org/display/DOCS/Object+IDs`

Usage

```
as.character.mongo.oid(x, ...)
```

Arguments

`x`	(mongo.oid) The OID to be converted.
`...`	Parameters passed from generic.

Value

(string) A string of 24 hex digits representing the bits of *oid*.

See Also

mongo.oid,
`mongo.oid.create`,
`as.character.mongo.oid`,
`mongo.oid.to.string`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.oid`,
mongo.bson.buffer,
mongo.bson.

Examples

```
oid <- mongo.oid.create()
print(as.character.mongo.oid(oid))
print(as.character(oid))  # print same thing as above line
```

`mongo` *The mongo (database connection) class*

Description

Objects of class "mongo" are used to connect to a MongoDB server and to perform database operations on that server.

mongo objects have "mongo" as their class and contain an externally managed pointer to the connection data. This pointer is stored in the "mongo" attribute of the object.

Note that the members of the mongo object only reflect
the initial parameters of `mongo.create()`. Only the external data actually changes if, for example, mongo.timeout is called after the initial call to `mongo.create`.

See Also

`mongo.create`,
`mongo.is.connected`,
`mongo.get.databases`,
`mongo.get.database.collections`,
`mongo.insert`,
`mongo.find.one`,
`mongo.find`,
`mongo.update`,
`mongo.remove`,
`mongo.drop`,
`mongo.drop.database`
mongo.gridfs.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Joe")
    mongo.bson.buffer.append(buf, "age", 22L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, "test.people", b)
}
```

`mongo.add.user` *Add a user and password*

Description

Add a user and password against to the given database on a MongoDB server for authentication purposes.

See `http://www.mongodb.org/display/DOCS/Security+and+Authentication`.

Usage

```
mongo.add.user(mongo, username, password, db="admin")
```

Arguments

`mongo`	(mongo) a mongo connection object.
`username`	(string) username to add.
`password`	(string) password corresponding to username.
`db`	(string) The database on the server to which to add the username and password.

See Also

`mongo.authenticate`,
mongo,
`mongo.create`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo))
    mongo.add.user(mongo, "Jeff", "H87b5dog")
```

`mongo.authenticate` *Authenticate a user and password*

Description

Authenticate a user and password against a given database on a MongoDB server.

See http://www.mongodb.org/display/DOCS/Security+and+Authentication.

Note that `mongo.create()` can authenticate a username and password before returning a connected mongo object.

Usage

```
mongo.authenticate(mongo, username, password, db="admin")
```

Arguments

`mongo`	(mongo) a mongo connection object.
`username`	(string) username to authenticate.
`password`	(string) password corresponding to username.
`db`	(string) The database on the server against which to validate the username and password.

See Also

`mongo.add.user`,
mongo,
`mongo.create`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo))
    mongo.authenticate(mongo, "Joe", "ZxYaBc217")
```

`mongo.binary.binary` *BSON binary data subtype constant for standard binary data*

Description

BSON binary data subtype constant for standard binary data.

Usage

```
mongo.binary.binary
```

Value

0L

See Also

`mongo.bson.buffer.append.raw`,
mongo.bson.

`mongo.binary.function` *BSON binary data subtype constant for function data*

Description

BSON binary data subtype constant for function data.

Usage

```
mongo.binary.function
```

Value

1L

See Also

`mongo.bson.buffer.append.raw`,
mongo.bson.

`mongo.binary.md5` *BSON binary data subtype constant for md5 data*

Description

BSON binary data subtype constant for md5 data.

Usage

```
mongo.binary.md5
```

Value

5L

See Also

`mongo.bson.buffer.append.raw`,
mongo.bson.

`mongo.binary.old` *BSON binary data subtype constant for old format data*

Description

BSON binary data subtype constant for old format data (deprecated).

Usage

```
mongo.binary.old
```

Value

2L

See Also

`mongo.bson.buffer.append.raw`,
mongo.bson.

`mongo.binary.user` *BSON binary data subtype constant for user data*

Description

BSON binary data subtype constant for user data.

Usage

```
mongo.binary.user
```

Value

128L

See Also

`mongo.bson.buffer.append.raw`,
mongo.bson.

mongo.binary.uuid *BSON binary data subtype constant for uuid data*

Description

BSON binary data subtype constant for uuid data.

Usage

```
mongo.binary.uuid
```

Value

4L

See Also

`mongo.bson.buffer.append.raw`,
mongo.bson.

mongo.bson *The mongo.bson class*

Description

Objects of class "mongo.bson" are used to store BSON documents. BSON is the form that MongoDB uses to store documents in its database. MongoDB network traffic also uses BSON in messages.

See `http://www.mongodb.org/display/DOCS/BSON`.

mongo.bson objects have "mongo.bson" as their class and contain an externally managed pointer to the actual document data. This pointer is stored in the "mongo.bson" attribute of the object.

Objects of class "mongo.bson.iterator" are used to iterate over a mongo.bson object to enumerate its keys and values.

Objects of class "mongo.bson.buffer" are used to build BSON documents.

See Also

`mongo.bson.from.list`,
`mongo.bson.to.list`,
mongo.bson.iterator,
mongo.bson.buffer,
`mongo.bson.from.buffer`,
`mongo.bson.empty`,
`mongo.find.one`,
`mongo.bson.destroy`.

Examples

```
b <- mongo.bson.from.list(list(name="Fred", age=29, city="Boston"))
iter <- mongo.bson.iterator.create(b)  # b is of class "mongo.bson"
while (mongo.bson.iterator.next(iter))
    print(mongo.bson.iterator.value(iter))
```

`mongo.bson.array` *BSON data type constant for an array*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (4L) to indicate that the value pointer to by an iterator is an array (containing child values).

Usage

```
mongo.bson.array
```

Value

4L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.iterator,
mongo.bson.

`mongo.bson.binary` *BSON data type constant for a binary data value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (5L) to indicate that the value pointer to by an iterator is binary data.

Usage

```
mongo.bson.binary
```

Value

5L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.iterator,
mongo.bson.

`mongo.bson.bool` *BSON data type constant for a bool value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (8L) to indicate that the value pointer to by an iterator is a bool.

Usage

```
mongo.bson.bool
```

Value

8L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.iterator,
mongo.bson.

`mongo.bson.buffer` *The mongo.bson.buffer class*

Description

Objects of class "mongo.bson.buffer" are used to build BSON documents (mongo.bson objects).

There are many functions for appending data into a mongo.bson.buffer object.
See `mongo.bson.buffer.append()` for a list of those functions.

After constructing your object in the buffer, `mongo.bson.from.buffer()` may be used to turn the buffer into a mongo.bson object.

mongo.bson.buffer objects have "mongo.bson.buffer" as their class and contain an externally managed pointer to the actual document data buffer. This pointer is stored in the "mongo.bson.buffer" attribute of the object.

See Also

mongo.bson,
`mongo.bson.buffer.size`,
`mongo.bson.from.buffer`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.start.object`,
`mongo.bson.buffer.start.array`,
`mongo.bson.buffer.finish.object`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "make", "Ford")
mongo.bson.buffer.append(buf, "model", "Mustang")
mongo.bson.buffer.append.int(buf, "year", 1968)
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.append` *Append a name/value pair into a mongo.bson.buffer*

Description

Append a name/value pair into a mongo.bson.buffer.

This function is a generic version of many 'append' functions. It will detect the type of the *value* parameter and perform the same action as the specific functions. These functions are:

- `mongo.bson.buffer.append.int()`
- `mongo.bson.buffer.append.string()`
- `mongo.bson.buffer.append.bool()`
- `mongo.bson.buffer.append.double()`
- `mongo.bson.buffer.append.complex()`
- `mongo.bson.buffer.append.null()`
- `mongo.bson.buffer.append.undefined()`
- `mongo.bson.buffer.append.symbol()`
- `mongo.bson.buffer.append.code()`
- `mongo.bson.buffer.append.code.w.scope()`
- `mongo.bson.buffer.append.raw()`
- `mongo.bson.buffer.append.time()`
- `mongo.bson.buffer.append.timestamp()`
- `mongo.bson.buffer.append.regex()`
- `mongo.bson.buffer.append.oid()`
- `mongo.bson.buffer.append.bson()`
- `mongo.bson.buffer.append.element()`
- `mongo.bson.buffer.append.list()`

`mongo.bson.buffer.append.long()` is missing from the above list since R has no 64-bit long integer type. If you wish a value to be stored in the BSON data as a long you must explicity call that function.

Usage

```
mongo.bson.buffer.append(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	The value of the field.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer.

Examples

```
buf <- mongo.bson.buffer.create()
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.append.bool` *Append a boolean field onto a mongo.bson.buffer*

Description

Append an logical (boolean) or vector of logical values onto a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.bool(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(logical vector) the booleans(s) to append to the buffer. If value has a names attribute, a subobject is appended and the subfields are given the indicated names. Othersize, if more than one element is present in value, the booleans are appended as a subarray. In the last case, a single as.boolean is appended as the value of the field.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.bool(buf, "wise", TRUE)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form { "wise" : true }

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.bool(buf, "bools", c(TRUE, FALSE, FALSE))
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "bools" : [true, false, false] }

buf <- mongo.bson.buffer.create()
flags <- c(FALSE, FALSE, TRUE)
names(flags) <- c("Tall", "Fat", "Pretty")
mongo.bson.buffer.append.bool(buf, "Looks", flags)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "Looks" : { "Tall" : false, "Fat" : false, "Pretty" : true } }
```

`mongo.bson.buffer.append.bson`

Append a mongo.bson object into a mongo.bson.buffer

Description

Append a mongo.bson object into a mongo.bson.buffer as a subobject.

Note that `mongo.bson.buffer.append()` will detect if its value parameter is a mongo.bson object and perform the same action as this function.

Usage

```
mongo.bson.buffer.append.bson(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the subobject field appended to the buffer.
`value`	(mongo.bson) a mongo.bson object.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.from.list`,
`mongo.bson.buffer.append`.

Examples

```
name <- mongo.bson.from.list(list(first="Joe", last="Smith"))
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.bson(buf, "name", name)
mongo.bson.buffer.append.string(buf, "city", "New York")
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "name" : { "first" : "Joe", "last" : "Smith" }, "city" : "New York" }
```

`mongo.bson.buffer.append.code`

Append a code field onto a mongo.bson.buffer

Description

Append a javascript code value onto a mongo.bson.buffer.

BSON has a special field type to indicate javascript code. This function appends such an indicator as the type of a field with its value.

Usage

```
mongo.bson.buffer.append.code(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	stringThe javascript code. Note that the value may simply be a string of javascript and not necessarily a mongo.code object.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.code,
`mongo.code.create`,
`mongo.bson.buffer.append`,
mongo.bson,
mongo.bson.buffer.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.code(buf, "SetXtoY", "x = y")
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "SetXtoY" : (CODE) "x = y" }

# The same result can be produced by the following code:
buf <- mongo.bson.buffer.create()
code <- mongo.code.create("x = y")
mongo.bson.buffer.append(buf, "SetXtoY", code)
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.append.code.w.scope` *Append a code field with a scope onto a mongo.bson.buffer*

Description

Append a javascript code value with a scope object onto a mongo.bson.buffer.

BSON has a special field type to indicate javascript code with a scope. This function appends such an indicator as the type of a field with its value.

Usage

```
mongo.bson.buffer.append.code.w.scope(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	mongo.code.w.scope The scoped javascript code.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.code.w.scope,
`mongo.code.w.scope.create`,
`mongo.bson.buffer.append`,
`mongo.bson.from.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
scope <- mongo.bson.from.list(list(scopevar="scopevalue"))
buf <- mongo.bson.buffer.create()
codeWscope <- mongo.code.w.scope.create("y = x", scope)
mongo.bson.buffer.append.code.w.scope(buf, "CodeWscope1",
     codeWscope)

# mongo.bson.buffer.append() will give the same result
# as it can detect the mongo.code.w.scope object
mongo.bson.buffer.append(buf, "CodeWscope2", codeWscope)

b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "CodeWscope1" : (CODEWSCOPE) "y = x"
#         (SCOPE) { "scopevar" : "scopevalue" },
#   "CodeWscope2" : (CODEWSCOPE) "y = x"
#         (SCOPE) { "scopevar" : "scopevalue" } }
```

---

mongo.bson.buffer.append.complex

Append a double field onto a mongo.bson.buffer

Description

Append a double or vector of doubles onto a mongo.bson.buffer.

Note that since BSON has no built-in complex type, R's complex values are appended as subobjects with two fields: "r" : the real part and "i" : the imaginary part.

Usage

```
mongo.bson.buffer.append.complex(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(complex vector) The values(s) to append to the buffer. If value has a names attribute, a subobject is appended and the subfields are given the indicated names. Othersize, if more than one element is present in value, the values are appended as a subarray. In the last case, a single complex is appended as the value of the field.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.complex(buf, "Alpha", 3.14159 + 2i)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "Alpha" : { "r" : 3.14159, "i" : 2 } }

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.complex(buf, "complexi", c(1.7 + 2.1i, 97.2))
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "complexi" : [ { "r": 1.7, i : 2.1}, { "r": 97.2, "i" : 0} ] }

buf <- mongo.bson.buffer.create()
values <- c(0.5 + 0.1i, 0.25)
names(values) <- c("Theta", "Epsilon")
mongo.bson.buffer.append.complex(buf, "Values", values)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "Values" : { "Theta"   : { "r" : 0.5, "i" : 0.1 },
#                "Epsilon" : { " r" : 0.25, "i" : 0 } } }
```

---

mongo.bson.buffer.append.double

Append a double field onto a mongo.bson.buffer

Description

Append a double or vector of doubles onto a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.double(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(double vector) The values(s) to append to the buffer. If value has a names attribute, a subobject is appended and the subfields are given the indicated names. Othersize, if more than one element is present in value, the values are appended as a subarray. In the last case, a single as.double is appended as the value of the field.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.double(buf, "YearSeconds",
     365.24219 * 24 * 60 * 60)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "YearSeconds" : 31556925.2 }

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.double(buf, "dbls",
    c(1.7, 87654321.123, 12345678.321))
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "dbls" : [1.7, 87654321.123, 12345678.321] }

buf <- mongo.bson.buffer.create()
fractions <- c(0.5, 0.25, 0.333333)
names(fractions) <- c("Half", "Quarter", "Third")
mongo.bson.buffer.append.double(buf, "Fractions", fractions)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "Fractions" : { "Half"    : 0.5,
#                   "Quarter" : 0.25,
#                   "Third"   : 0.333333 } }
```

`mongo.bson.buffer.append.element` *Append a mongo.bson.iterator's element into a mongo.bson.buffer*

Description

Append a mongo.bson.iterator's element into a mongo.bson.buffer.

`mongo.bson.buffer.append()` will detect if its value parameter is a mongo.bson.iterator object and perform the same action as this function.

Usage

```
mongo.bson.buffer.append.element(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the subobject field appended to the buffer. If NULL, the name appended will come from the element pointed to by the iterator.
`value`	A (mongo.bson.iterator) object.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.find`,
`mongo.bson.from.list`,
`mongo.bson.buffer.append`.

Examples

```
name <- mongo.bson.from.list(list(first="Joe", last="Smith"))
iter <- mongo.bson.find(name, "last")
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.element(buf, "last", iter)
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object (b) of the following form:
# { "last" : "Smith" }
```

`mongo.bson.buffer.append.int`

Append an integer field onto a mongo.bson.buffer

Description

Append an integer or vector of integers onto a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.int(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(integer vector) The integer(s) to append to the buffer. If value has a names attribute, a subobject is appended and the subfields are given the indicated names. Othersize, if more than one element is present in value it must be a vector of integers and the integers are appended as a subarray. In the last case, the single value must be coerible to an integer.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.int(buf, "age", 23L)
b <- mongo.bson.from.buffer(buf)

# the above produces a BSON object of the form { "age" : 21 }

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.int(buf, "ages", c(21L, 19L, 13L))
b <- mongo.bson.from.buffer(buf)

# the above produces a BSON object of the form { "ages" : [21, 19, 13] }

buf <- mongo.bson.buffer.create()
dim <- c(2L, 4L, 8L)
names(dim) <- c("width", "height", "length")
mongo.bson.buffer.append.int(buf, "board", dim)
b <- mongo.bson.from.buffer(buf)

# theabove produces a BSON object of the form:
# { "board" : { "width" : 2, "height" : 4, "length" : 8 } }
```

`mongo.bson.buffer.append.list` *Append a list onto a mongo.bson.buffer*

Description

Append a list onto a mongo.bson.buffer.

Note that the value parameter must be a true list, not an vector of a single atomic type.

Also note that this function is recursive and will append items that are lists themselves as subobjects.

Usage

```
mongo.bson.buffer.append.list(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(list) The list to append to the buffer as a subobject.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
l <- list(fruit = "apple", hasSeeds = TRUE)
mongo.bson.buffer.append.list(buf, "item", l)
b <- mongo.bson.from.buffer(buf)

# this produces a BSON object of the form:
# { "item" : { "fruit" : "apple", "hasSeeds" : true } }
```

`mongo.bson.buffer.append.long` *Append a long valued field onto a mongo.bson.buffer*

Description

Append a long value or vector of longs onto a mongo.bson.buffer.

Note that since R has no long (64-bit integer) type, doubles are used in R, but are converted to 64-bit values when stored in the buffer; some loss of precision may occur.

This is the only case in which `mongo.bson.buffer.append()` cannot make the proper guess about what type to encode into the buffer.
You must call `mongo.bson.buffer.append.long()` explicitly; otherwise, doubles are appended.

Usage

```
mongo.bson.buffer.append.long(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(double vector) The values(s) to append to the buffer. If value has a names attribute, a subobject is appended and the subfields are given the indicated names. Othersize, if more than one element is present in value, the values are appended as a subarray. In the last case, a single long is appended as the value of the field.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.long(buf, "YearSeconds",
    365.24219 * 24 * 60 * 60)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "YearSeconds" : 31556925 }

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.long(buf, "longs",
    c(1, 9087654321, 1234567809))
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "longs" : [1, 9087654321, 1234567809] }

buf <- mongo.bson.buffer.create()
distances <- c(473, 133871000, 188178313)
names(distances) <- c("Sol", "Proxima Centari", "Bernard's Star")
mongo.bson.buffer.append.long(buf, "Stars", distances)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "Stars" : { "Sol" : 474,
#               "Proxima Centari" : 133871000,
#               "Bernard's Star"  : 188178313 } }
```

`mongo.bson.buffer.append.null` *Append a double field onto a mongo.bson.buffer*

Description

Append a NULL value onto a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.null(buf, name)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.null(buf, "Nil")
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form { "Nil" : NULL }
```

`mongo.bson.buffer.append.oid` *Append a OID into a mongo.bson.buffer*

Description

Append a OID (Object ID) value into a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.oid(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(mongo.oid) An OID value.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
mongo.oid.create,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.oid(buf, "Now", mongo.oid.create())
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.append.raw` *Append a raw (binary) field onto a mongo.bson.buffer*

Description

Append raw (binary) data onto a mongo.bson.buffer.

BSON has a special field type to indicate binary data. This function appends such an indicator as the type of a field with its value.

Usage

```
mongo.bson.buffer.append.raw(buf, name, value, subtype=NULL)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(raw) the binary data.
`subtype`	(as.integer) The binary data subtype. If subtype == NULL, the "subtype" attribute of the raw is used. If this is not present, mongo.binary.binary is used. The following constants are defined: • `mongo.binary.binary` (0L) • `mongo.binary.function` (1L) • `mongo.binary.old` (2L) • `mongo.binary.uuid` (3L) • `mongo.binary.md5` (5L) • `mongo.binary.user` (128L)

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

`mongo.bson.buffer.append`,
mongo.bson,
mongo.bson.buffer.

Examples

```
buf <- mongo.bson.buffer.create()
bin <- raw(3)
for (i in 0:2)
    bin[i] <- as.raw(i * 3)
mongo.bson.buffer.append.raw(buf, "bin1", bin)

# Note that mongo.bson.buffer.append()
# will detect whether the value parameter
# is a raw object and append the appropriate value.
```

```
mongo.bson.buffer.append(buf, "bin2", bin)  # gives same result
```

---

mongo.bson.buffer.append.regex

Append a timestamp value into a mongo.bson.buffer

Description

Append a regular expression value into a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.regex(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(mongo.regex) A regular expression as created by `mongo.regex.create()`.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.regex.create,
`mongo.bson.buffer.append.regex`,
`mongo.bson.buffer.append`,
mongo.bson,
mongo.bson.buffer.

Examples

```
buf <- mongo.bson.buffer.create()
regex <- mongo.regex.create("acme.*corp", options="i")
mongo.bson.buffer.append.regex(buf, "MatchAcme", regex)
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.append.string`
Append a string field onto a mongo.bson.buffer

Description

Append an string or vector of strings onto a mongo.bson.buffer.

Usage

```
mongo.bson.buffer.append.string(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(string vector) The strings(s) to append to the buffer. If value has a names attribute, a subobject is appended and the subfields are given the indicated names. Othersize, if more than one element is present in value, the strings are appended as a subarray. In the last case, a single string is appended as the value of the field.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.string(buf, "name", "Joe")
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form { "name" : "Joe" }

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.string(buf, "names", c("Fred", "Jeff", "John"))
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "names" : ["Fred", "Jeff", "John"] }

buf <- mongo.bson.buffer.create()
staff <- c("Mark", "Jennifer", "Robert")
names(staff) <- c("Chairman", "President", "Secretary")
mongo.bson.buffer.append.string(buf, "board", staff)
```

```
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "board" : { "Chairman"  : "Mark",
#               "President" : "Jennifer",
#               "Secretary" : "Robert" } }
```

mongo.bson.buffer.append.symbol
Append a symbol field onto a mongo.bson.buffer

Description

Append a symbol value onto a mongo.bson.buffer.

BSON has a special field type to indicate a symbol. This function appends such an indicator as the type of a field with its value.

Usage

```
mongo.bson.buffer.append.symbol(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	(string) The value of the symbol. Note that the value may simply be a string representing the symbol's value and not necessarily a mongo.symbol object.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
mongo.symbol,
`mongo.symbol.create`,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.symbol(buf, "A", "Alpha")
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form { "A" : (SYMBOL) "Alpha" }

# The same result can be produced by the following code:
buf <- mongo.bson.buffer.create()
sym <- mongo.symbol.create("Alpha")
mongo.bson.buffer.append(buf, "A", sym)
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.append.time`
Append a time value into a mongo.bson.buffer

Description

Append a date/time value into a mongo.bson.buffer.

BSON has a special field type to indicate a date/time; these are 64-bit values.

However, R has a 'standard' object of class "POSIXct" used to represent date/time values, such as that returned by Sys.time(). Internally these are a 32-bit integer number of milliseconds since midnight January 1, 1970. On January 19, 2038, 32-bit versions of the the Unix time stamp will cease to work, as it will overflow the largest value that can be held in a signed 32-bit number. At such time, many applications, including R and this driver, will need to address that issue.

Usage

```
mongo.bson.buffer.append.time(buf, name, time)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`time`	(integer) A time value. This may also be an object of class "POSIXct", "POSIXlt" or "mongo.timestamp".

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.timestamp,
`mongo.timestamp.create`,
`mongo.bson.buffer.append`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.time(buf, "Now", Sys.time())
b <- mongo.bson.from.buffer(buf)
```

mongo.bson.buffer.append.timestamp

Append a timestamp value into a mongo.bson.buffer

Description

Append a timestamp value into a mongo.bson.buffer.

mongo.timestamp objects extend the "POSIXct" class to include an attrubute "increment".

See `mongo.bson.buffer.append.time()`.

Usage

```
mongo.bson.buffer.append.timestamp(buf, name, value)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.
`value`	A (mongo.timestamp) value as created by `mongo.timestamp.create()`.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.timestamp.create,
`mongo.bson.buffer.append.time`,
`mongo.bson.buffer.append`,
mongo.bson,
mongo.bson.buffer.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.timestamp(buf, "Now-27",
    mongo.timestamp.create(Sys.time(), 27))
b <- mongo.bson.from.buffer(buf)
```

mongo.bson.buffer.append.undefined

Append a undefined field onto a mongo.bson.buffer

Description

Append a undefined value onto a mongo.bson.buffer.

BSON has a special field type to indicate an undefined value. This function appends such an indicator as the value of a field.

Usage

```
mongo.bson.buffer.append.undefined(buf, name)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the field appended to the buffer.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
mongo.undefined,
`mongo.undefined.create`,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append.undefined(buf, "Undef")
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form { "Undef" : UNDEFINED }

# The same result can be produced by the following code:
buf <- mongo.bson.buffer.create()
undef <- mongo.undefined.create()
mongo.bson.buffer.append(buf, "Undef", undef)
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.create`
Create an new mongo.bson.buffer object

Description

Returns a fresh mongo.bson.buffer object ready to have data appended onto it.

mongo.bson.buffer objects are used to build mongo.bson objects.

Usage

```
mongo.bson.buffer.create()
```

Value

A fresh mongo.bson.buffer object

See Also

mongo.bson,
mongo.bson.buffer.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Donna")
b <- mongo.bson.from.buffer(buf)
```

`mongo.bson.buffer.finish.object` *Finish a subobject or array within a mongo.bson.buffer*

Description

BSON documents may themselves contain nested documents. Call this function to finish a subobject within a mongo.bson.buffer.

`mongo.bson.buffer.start.object()` and `mongo.bson.buffer.finish.object()` may be called in a stackwise (LIFO) order to further nest documents.

This function must also be called to finish arrays.

Usage

```
mongo.bson.buffer.finish.object(buf)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object on which to finish a subobject.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.start.object`,
`mongo.bson.buffer.start.array`,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.start.object(buf, "name")
mongo.bson.buffer.append(buf, "first", "Jeff")
mongo.bson.buffer.append(buf, "last", "Davis")
mongo.bson.buffer.finish.object(buf)
mongo.bson.buffer.append(buf, "city", "Toronto")
b <- mongo.bson.from.buffer(buf)
```

```
# the above produces a BSON object of the form:
# { "name" : { "first" : "Jeff", "last" : "Davis" }, "city" : "Toronto" }
```

---

mongo.bson.buffer.size

Get the size of a mongo.bson.buffer object

Description

Get the number of bytes which would be taken up by the BSON data when the buffer is converted to a mongo.bson object with `mongo.bson.from.buffer()`.

Usage

```
mongo.bson.buffer.size(buf)
```

Arguments

`buf`	(mongo.bson.buffer) the mongo.bson.buffer object to examine.

Value

(integer) the number of bytes which would be taken up by the BSON data with the buffer is converted to a mongo.bson object with `mongo.bson.from.buffer()`.

See Also

mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Fred")
mongo.bson.buffer.append(buf, "city", "Dayton")
# both should report 37
print(mongo.bson.buffer.size(buf))
y <- mongo.bson.from.buffer(buf)
print(mongo.bson.size(y))
```

mongo.bson.buffer.start.array

Start an array within a mongo.bson.buffer

Description

Call this function to start an array within a mongo.bson.buffer.
`mongo.bson.buffer.finish.object()` must be called when finished appending the elements of the array.

(`mongo.bson.buffer.start.object()`,`mongo.bson.buffer.start.array()`) and `mongo.bson.buffer.finsih.object()` may be called in a stackwise (LIFO) order to further nest arrays and documents.

The names of the elements appended should properly be given sequentially numbered strings.

Note that arrays will be automatically appended by the 'append' functions when appending vectors (containing more than one element) of atomic types.

Usage

```
mongo.bson.buffer.start.array(buf, name)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the array to be appended to the buffer.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.finish.object`,
`mongo.bson.buffer.start.array`,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.start.array(buf, "Fibonacci")
x <- 0
mongo.bson.buffer.append.int(buf, "0", x)
y <- 1
mongo.bson.buffer.append.int(buf, "1", y)
for (i in 2:8) {
    z <- x + y
    mongo.bson.buffer.append.int(buf, as.character(i), z)
    x <- y
    y <- z
}
```

```
mongo.bson.buffer.finish.object(buf)
b <- mongo.bson.from.buffer(buf)

# the above produces a BSON object of the form:
# { "Fibonacci" : [ 0, 1, 1, 2, 3, 5, 8, 13, 21 ] }
```

---

mongo.bson.buffer.start.object

Start a subobject within a mongo.bson.buffer

Description

BSON documents may themselves contain nested documents. Call this function to start a subobject within a mongo.bson.buffer.

`mongo.bson.buffer.finish.object()` must be called when finsihed appending subfields. (`mongo.bson.buffer.start.object()`, `mongo.bson.buffer.start.array()`) and `mongo.bson.buffer.finish.object()` may be called in a stackwise (LIFO) order to further nest documents and arrays.

Usage

```
mongo.bson.buffer.start.object(buf, name)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer object to which to append.
`name`	(string) The name (key) of the subobject to be appended to the buffer.

Value

TRUE if successful; otherwise, FALSE if an error occured appending the data.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.finish.object`,
`mongo.bson.buffer.start.array`,
`mongo.bson.buffer.append`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.start.object(buf, "name")
mongo.bson.buffer.append(buf, "first", "Jeff")
mongo.bson.buffer.append(buf, "last", "Davis")
mongo.bson.buffer.finish.object(buf)
mongo.bson.buffer.append(buf, "city", "Toronto")
b <- mongo.bson.from.buffer(buf)

# the above produces a BSON object of the form:
# { "name" : { "first" : "Jeff", "last" : "Davis" }, "city" : "Toronto" }
```

`mongo.bson.code` *BSON data type constant for a code value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (13L) to indicate that the value pointer to by an iterator is javascript code.

Usage

```
mongo.bson.code
```

Value

13L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.code.w.scope` *BSON data type constant for a code with scope value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (15L) to indicate that the value pointer to by an iterator is a javascript with a scope.

Usage

```
mongo.bson.code.w.scope
```

Value

15L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson

`mongo.bson.date` *BSON data type constant for a date value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (9L) to indicate that the value pointer to by an iterator is a date/time.

Usage

```
mongo.bson.date
```

Value

9L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.dbref` *BSON data type constant for a dbref value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (12L) to indicate that the value pointed to by an iterator is a dbref (database reference).

Note that this BSON data type is deprecated and rmongodb provides no support for it. Attempting to fetch the value of a dbref with `mongo.bson.to.list()` or `mongo.bson.iterator.value()` will throw an error. The field must be skipped by calling `mongo.bson.iterator.next()`.

Usage

```
mongo.bson.dbref
```

Value

12L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.destroy` *Destroy a mongo.bson object*

Description

Releases the resources associated with a mongo.bson object. It is not absolutely necessary to call this function since R's garbage collection will eventually get around to doing it for you.

Usage

```
mongo.bson.destroy(b)
```

Arguments

b	A (mongo.bson) object.

Value

NULL

See Also

mongo.bson,
`mongo.bson.from.list`,
`mongo.bson.from.buffer`.

Examples

```
b <- mongo.bson.from.list(list(name="Cheryl", age=29))
print(b)
mongo.bson.destroy(b)
```

`mongo.bson.double` *BSON data type constant for a double value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (1L) to indicate that the value pointer to by an iterator is a double.

Usage

```
mongo.bson.double
```

Value

1L

See Also

mongo.bson.iterator.type,
mongo.bson.iterator.next,
mongo.bson.

---

mongo.bson.empty *Create an empty mongo.bson object*

Description

Returns an empty mongo.bson object. mongo.bson objects have "mongo.bson" as their class and contain an externally managed pointer to the actual data. This pointer is stored in the "mongo.bson" attribute of the object.

Usage

```
mongo.bson.empty()
```

Value

An empty mongo.bson object

See Also

mongo.bson

Examples

```
# Use an empty mongo.bson for the query object which matches everything.
# This happens to be the default value for the query
# parameter to mongo.count,  but we explicity use mongo.bson.empty()
# here for an example.
mongo <- mongo.create()
if (mongo.is.connected(mongo))
    print(mongo.count(mongo, "test.people", query=mongo.bson.empty()))
```

---

mongo.bson.eoo *BSON data type constant for 'End Of Object'*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (0L) at the end of the object when there are no more fields through which to iterate.

Usage

```
mongo.bson.eoo
```

Value

0L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.find` *Find a field within a mongo.bson object by name*

Description

Find a field within a mongo.bson object by the name (key) of the field
and return a mongo.bson.iterator pointing to that field.

Usage

```
mongo.bson.find(b, name)
```

Arguments

`b`	(mongo.bson) The object in which to find the field.
`name`	(string) The name of the field to find.

Value

(mongo.bson.iterator) An iterator pointing to the field found if name was found among the names of the fields; otherwise, NULL.

See Also

mongo.bson.iterator,
`mongo.bson.iterator.value`,
mongo.bson.

Examples

```
b <- mongo.bson.from.list(list(name="John", age=32))
iter <- mongo.bson.find(b, "age")
print(mongo.bson.iterator.value(iter)) # print 32
```

`mongo.bson.from.buffer` *Convert a mongo.bson.buffer object to a mongo.bson object*

Description

Convert a mongo.bson.buffer object to a mongo.bson object.

Use this after appending data to a buffer to turn it into a mongo.bson object for network transport.

No futher data may be appended to the buffer after calling this function.

Usage

```
mongo.bson.from.buffer(buf)
```

Arguments

`buf`	(mongo.bson.buffer) The buffer to convert.

Value

A mongo.bson object as converted from the buffer parameter.

See Also

mongo.bson,
mongo.bson.buffer,
`mongo.bson.buffer.append`,
`mongo.bson.destroy`.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Fred")
mongo.bson.buffer.append(buf, "city", "Dayton")
b <- mongo.bson.from.buffer(buf)
print(b)
mongo.bson.destroy(b)
```

`mongo.bson.from.list` *Convert a list to a mongo.bson object*

Description

Convert a list to a mongo.bson object.

This function permits the simple and convenient creation of a mongo.bson object. This bypasses the creation of a mongo.bson.buffer, appending fields one by one, and then turning the buffer into a mongo.bson object with `mongo.bson.from.buffer()`.

Note that this function and `mongo.bson.to.list()` do not always perform inverse conversions since mongo.bson.to.list() will convert objects and subobjects to atomic vectors if possible.

Usage

```
mongo.bson.from.list(lst)
```

Arguments

`lst`	(list) The list to convert. This *must* be a list, *not* a vector of atomic types; otherwise, an error is thrown; use `as.list()` as necessary.

Value

(mongo.bson) A mongo.bson object serialized from *lst*.

See Also

`mongo.bson.to.list`,
mongo.bson,
`mongo.bson.destroy`.

Examples

```
lst <- list(name="John", age=32)
b <- mongo.bson.from.list(lst)
# the above produces a BSON object of the form:
# { "name" : "John", "age" : 32.0 }

# Convert a vector of an atomic type to a list and
# then to a mongo.bson object
v <- c(president="Jefferson", vice="Burr")
b <- mongo.bson.from.list(as.list(v))
# the above produces a BSON object of the form:
# { "president" : "Jefferson", "vice" : "Burr" }
```

`mongo.bson.int` *BSON data type constant for a integer value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (16L) to indicate that the value pointer to by an iterator is a integer (32-bit).

Usage

```
mongo.bson.int
```

Value

16L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

mongo.bson.iterator

The mongo.bson.iterator class

Description

Objects of class "mongo.bson.iterator" are used to iterate through BSON documents as stored in mongo.bson objects.

mongo.bson.iterator objects have "mongo.bson.iterator" as their class and contain an externally managed pointer to the actual document data. This pointer is stored in the "mongo.bson.iterator" attribute of the object.

See Also

`mongo.bson.iterator.create`,
`mongo.bson.find`,
`mongo.bson.iterator.next`,
`mongo.bson.iterator.key`,
`mongo.bson.iterator.value`,
mongo.bson.

Examples

```
b <- mongo.bson.from.list(list(name="Joy", age=35, city="Ontario"))
# b is of class "mongo.bson"
iter <- mongo.bson.iterator.create(b)
while (mongo.bson.iterator.next(iter))
    print(mongo.bson.iterator.value(iter))
```

mongo.bson.iterator.create

Create a mongo.bson.iterator object

Description

Create a mongo.bson.iterator object used to step through a given mongo.bson object one field at a time.

Usage

```
mongo.bson.iterator.create(b)
```

Arguments

`b`	(mongo.bson) The mongo.bson object through which to iterate. *b* may also be a mongo.bson.iterator and is expected to point to a subobject or array. The iterator returned may be used to step through the subobject or array.

Value

(mongo.bson.iterator) An iterator initialized to 'before' the start of the given mongo.bson object. `mongo.bson.iterator.next()` should be used on the iterator first to step to the first field.

See Also

mongo.bson.iterator,
`mongo.bson.find`,
`mongo.bson.iterator.next`,
`mongo.bson.iterator.key`,
`mongo.bson.iterator.type`,
`mongo.bson.iterator.value`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)

iter <- mongo.bson.iterator.create(b)
while (mongo.bson.iterator.next(iter))
    if (mongo.bson.iterator.key(iter) == "created") {
        print(mongo.bson.iterator.value(iter))
        break
    }

# The above is given for illustrative purposes, but may be performed
# much easier (and faster) by the following:
iter <- mongo.bson.find(b, "created")
print(mongo.bson.iterator.value(iter))
```

`mongo.bson.iterator.key`
Return the key (name) of the field pointed to by an iterator

Description

Return the key (name) of the field pointed to by a mongo.bson.iterator.

Usage

```
mongo.bson.iterator.key(iter)
```

Arguments

`iter`	A mongo.bson.iterator.

Value

(string) The key (name) of the field pointed to by *iter*

See Also

mongo.bson.iterator,
`mongo.bson.iterator.create`,
`mongo.bson.find`,
`mongo.bson.iterator.next`,
`mongo.bson.iterator.type`,
`mongo.bson.iterator.value`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)

# iterate through all values and print them with their keys (names)
iter <- mongo.bson.iterator.create(b)
while (mongo.bson.iterator.next(iter)) { # eoo at end stops loop
    print(mongo.bson.iterator.key(iter))
    print(mongo.bson.iterator.value(iter))
}
```

`mongo.bson.iterator.next`

Advance an iterator to the first or next field

Description

Advance a mongo.bson.iterator to the first or next field.

Usage

```
mongo.bson.iterator.next(iter)
```

Arguments

`iter`	A mongo.bson.iterator.

Value

(integer) The type of the next of the field pointed to by the iterator as indicated by the folllowing constants:

- mongo.bson.eoo – End of Object (0L)
- mongo.bson.double
- mongo.bson.string
- mongo.bson.object
- mongo.bson.array
- mongo.bson.binary
- mongo.bson.undefined
- mongo.bson.oid
- mongo.bson.bool
- mongo.bson.date
- mongo.bson.null
- mongo.bson.regex
- mongo.bson.dbref – deprecated (follow link for more info)
- mongo.bson.code
- mongo.bson.symbol
- mongo.bson.code.w.scope
- mongo.bson.int
- mongo.bson.timestamp
- mongo.bson.long

See Also

mongo.bson.iterator,
`mongo.bson.iterator.create`,
`mongo.bson.find`,
`mongo.bson.iterator.key`,
`mongo.bson.iterator.type`,
`mongo.bson.iterator.value`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)

iter <- mongo.bson.iterator.create(b)
# Advance to the "cars" field
while (mongo.bson.iterator.next(iter) != mongo.bson.null)
```

```
{
    # NOP
}
print(mongo.bson.iterator.value(iter))

# The above is given for illustrative purposes, but may be performed
# much easier by the following:
iter <- mongo.bson.find(b, "cars")
print(mongo.bson.iterator.value(iter))

# iterate through all values and print them with their keys (names)
iter <- mongo.bson.iterator.create(b)
while (mongo.bson.iterator.next(iter)) { # eoo at end stops loop
    print(mongo.bson.iterator.key(iter))
    print(mongo.bson.iterator.value(iter))
}
```

`mongo.bson.iterator.type`

Get the type of data pointed to by an iterator

Description

Return the type of the field currently pointed to by a mongo.bson.iterator.

Usage

```
mongo.bson.iterator.type(iter)
```

Arguments

`iter`	A mongo.bson.iterator.

Value

(integer) The type of the field pointed to by the iterator as indicated by the following constants:

- mongo.bson.eoo – End of Object (0L)
- mongo.bson.double
- mongo.bson.string
- mongo.bson.object
- mongo.bson.array
- mongo.bson.binary
- mongo.bson.undefined
- mongo.bson.oid
- mongo.bson.bool
- mongo.bson.date
- mongo.bson.null
- mongo.bson.regex

- mongo.bson.dbref – deprecated (follow link for more info)
- mongo.bson.code
- mongo.bson.symbol
- mongo.bson.code.w.scope
- mongo.bson.int
- mongo.bson.timestamp
- mongo.bson.long

See Also

mongo.bson.iterator,
`mongo.bson.iterator.create`,
`mongo.bson.find`,
`mongo.bson.iterator.next`,
`mongo.bson.iterator.key`,
`mongo.bson.iterator.value`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)

iter <- mongo.bson.iterator.create(b)
while (mongo.bson.iterator.next(iter))
    if (mongo.bson.iterator.type(iter) == mongo.bson.date) {
        print(mongo.bson.iterator.value(iter))
        break
    }

# The above is given for illustrative purposes, but may be performed
# much easier by the following:
iter <- mongo.bson.find(b, "created")
print(mongo.bson.iterator.value(iter))
```

`mongo.bson.iterator.value` *Return the value of the field pointed to by an iterator*

Description

Return the value of the field pointed to by a mongo.bson.iterator.

Usage

```
mongo.bson.iterator.value(iter)
```

Arguments

`iter`	A mongo.bson.iterator.

Value

The value of the field pointed to by *iter*.

This function returns an appropriate R object depending on the type of the field pointed to by the iterator. This mapping to values is as follows:

`mongo.bson.eoo`	0L
`mongo.bson.double`	A double
`mongo.bson.string`	A string
`mongo.bson.object`	If the object is recognized as a complex value
(of the form { "r" : double, "i" : double }),	
a complex value is returned; otherwise,	
a mongo.bson object is returned which may be further	
iterated through for the values of subfields.	
`mongo.bson.array`	If all fields of the array are of the same atomic type,
a vector of that type is returned.	
Arrays of complex values are also detected as above.	
otherwise, a mongo.bson object is returned which	
may be examined similarly as for mongo.bson.object.	
`mongo.bson.binary`	A raw object.
The subtype of the binary data is stored in the attribute "subtype".	
See `mongo.bson.buffer.append.raw()`.	
`mongo.bson.undefined`	A mongo.undefined object
`mongo.bson.oid`	A mongo.oid object
`mongo.bson.bool`	A logical
`mongo.bson.date`	A "POSIXct" class object
`mongo.bson.null`	NULL
`mongo.bson.regex`	A mongo.regex object
`mongo.bson.dbref`	Error! (deprecated – see link)
`mongo.bson.code`	A mongo.code object
`mongo.bson.symbol`	A mongo.symbol object
`mongo.bson.code.w.scope`	A mongo.code.w.scope object
`mongo.bson.int`	An integer
`mongo.bson.timestamp`	A mongo.timestamp object
`mongo.bson.long`	A double

See Also

mongo.bson.iterator,
`mongo.bson.iterator.create`,
`mongo.bson.find`,
`mongo.bson.iterator.next`,
`mongo.bson.iterator.key`,
`mongo.bson.iterator.type`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
```

```
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)

# iterate through all values and print them with their keys (names)
iter <- mongo.bson.iterator.create(b)
while (mongo.bson.iterator.next(iter)) { # eoo at end stops loop
    print(mongo.bson.iterator.key(iter))
    print(mongo.bson.iterator.value(iter))
}
```

`mongo.bson.long` *BSON data type constant for a long value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (18L) to indicate that the value pointer to by an iterator is a long integer (64 bits).

Usage

```
mongo.bson.long
```

Value

18L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.null` *BSON data type constant for a null value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (10L) to indicate that the value pointer to by an iterator is a null.

Usage

```
mongo.bson.null
```

Value

10L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.object` *BSON data type constant for a subobject value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (3L) to indicate that the value pointer to by an iterator is a subobject.

Usage

```
mongo.bson.object
```

Value

3L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.oid` *BSON data type constant for a oid value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (7L) to indicate that the value pointer to by an iterator is a oid (Object ID).

Usage

```
mongo.bson.oid
```

Value

7L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.print` *Display a mongo.bson object*

Description

Display formatted output of a mongo.bson object.

Output is tabbed (indented to show the nesting level of subobjects and arrays).

Usage

```
mongo.bson.print(x, ...)
```

Arguments

`x`	(mongo.bson) The mongo.bson object to display.
`...`	Parameters passed from generic.

Value

The parameter is returned unchanged.

See Also

mongo.bson

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Fred")
mongo.bson.buffer.append(buf, "city", "Dayton")
b <- mongo.bson.from.buffer(buf)

# all display the same thing
mongo.bson.print(b)
print.mongo.bson(b)
print(b)
```

`mongo.bson.regex` *BSON data type constant for a regex value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (11L) to indicate that the value pointer to by an iterator is a regular expression.

Usage

```
mongo.bson.regex
```

Value

11L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.size` *Get the size of a mongo.bson object*

Description

Get the number of bytes taken up by the BSON data attached to the mongo.bson object

Usage

```
mongo.bson.size(b)
```

Arguments

`b`	(mongo.bson) the mongo.bson object to examine.

Value

(integer) the number of bytes taken up by the BSON data attached to the mongo.bson object.

See Also

mongo.bson

Examples

```
# should report 5
print(mongo.bson.size(mongo.bson.empty()))

buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Fred")
mongo.bson.buffer.append(buf, "city", "Dayton")
y <- mongo.bson.from.buffer(buf)
# should report 37
print(mongo.bson.size(y))
```

`mongo.bson.string` *BSON data type constant for a string value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (2L) to indicate that the value pointer to by an iterator is a string.

Usage

```
mongo.bson.string
```

Value

2L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.symbol` *BSON data type constant for a symbol value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (14L) to indicate that the value pointer to by an iterator is a symbol.

Usage

```
mongo.bson.symbol
```

Value

14L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.timestamp` *BSON data type constant for a timestamp value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (17L) to indicate that the value pointer to by an iterator is a timestamp.

Usage

```
mongo.bson.timestamp
```

Value

17L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.to.list` *Convert a mongo.bson object to an R object.*

Description

Convert a mongo.bson object to an R object.

Note that this function and `mongo.bson.from.list()` do not always perform inverse conversions since `mongo.bson.to.list()` will convert objects and subobjects to atomic vectors if possible.

This function is somewhat schizophrenic depending on the types of the fields in the mongo.bson object. If all fields in an object (or subobject/array) can be converted to the same atomic R type (for example they are all strings or all integer, you'll actually get out a vector of the atomic type with the names attribute set.

For example, if you construct a mongo.bson object like such:

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "First", "Joe")
mongo.bson.buffer.append(buf, "Last", "Smith")
b <- mongo.bson.from.buffer(buf)
l <- mongo.bson.to.list(b)
```

You'll get a vector of strings out of it which may be indexed by number, like so:

```
print(l[1]) # display "Joe"
```

or by name, like so:

```
print(l[["Last"]]) # display "Smith"
```

If, however, the mongo.bson object is made up of disparate types like such:

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "Name", "Joe Smith")
mongo.bson.buffer.append(buf, "age", 21.5)
b <- mongo.bson.from.buffer(buf)
l <- mongo.bson.to.list(b)
```

You'll get a true list (with the names attribute set) which may be indexed by number also:

```
print(l[1]) # display "Joe Smith"
```

or by name, in the same fashion as above, like so

```
print(l[["Name"]]) # display "Joe Smith"
```

but also with the $ operator, like so:

```
print(l$age) # display 21.5
```

Note that `mongo.bson.to.list()` operates recursively on subobjects and arrays and you'll get lists whose members are lists or vectors themselves.

Perhaps the best way to see what you are going to get for your particular application is to test it.

Usage

```
mongo.bson.to.list(b)
```

Arguments

b	(mongo.bson) The mongo.bson object to convert.

Value

Best guess at an appropriate R object representing the mongo.bson object.

See Also

`mongo.bson.from.list`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Fred")
mongo.bson.buffer.append(buf, "city", "Dayton")
b <- mongo.bson.from.buffer(buf)

l <- mongo.bson.to.list(b)
print(l)
```

`mongo.bson.undefined` *BSON data type constant for a undefined value*

Description

`mongo.bson.iterator.type()` and `mongo.bson.iterator.next()` will return this constant (6L) to indicate that the value pointer to by an iterator is a undefined.

Usage

```
mongo.bson.undefined
```

Value

6L

See Also

`mongo.bson.iterator.type`,
`mongo.bson.iterator.next`,
mongo.bson.

`mongo.bson.value` *Return the value of a mongo.bson field*

Description

Search a mongo.bson object for a field by name and retrieve its value.

Usage

```
mongo.bson.value(b, name)
```

Arguments

`b`	A mongo.bson object.
`name`	(string) The name of a field within `b`

Value

NULL if name is not found;
otherwise, the value of the found field.

This function returns an appropriate R object depending on the type of the field found. This mapping to values is as follows:

`mongo.bson.double`	A double
`mongo.bson.string`	A string
`mongo.bson.object` (of the form { "r" : double, "i" : double }),	If the object is recognized as a complex value

	a complex value is returned; otherwise, a mongo.bson object is returned which may be further iterated through for the values of subfields.
`mongo.bson.array`	If all fields of the array are of the same atomic type, a vector of that type is returned. Arrays of complex values are also detected as above. otherwise, a mongo.bson object is returned which may be examined similarly as for mongo.bson.object.
`mongo.bson.binary`	A raw object. The subtype of the binary data is stored in the attribute "subtype". See `mongo.bson.buffer.append.raw()`.
`mongo.bson.undefined`	A mongo.undefined object
`mongo.bson.oid`	A mongo.oid object
`mongo.bson.bool`	A logical
`mongo.bson.date`	A "POSIXct" class object
`mongo.bson.null`	NULL
`mongo.bson.regex`	A mongo.regex object
`mongo.bson.dbref`	Error! (deprecated – see link)
`mongo.bson.code`	A mongo.code object
`mongo.bson.symbol`	A mongo.symbol object
`mongo.bson.code.w.scope`	A mongo.code.w.scope object
`mongo.bson.int`	An integer
`mongo.bson.timestamp`	A mongo.timestamp object
`mongo.bson.long`	A double

See Also

mongo.bson.iterator.value,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
# Append a string
mongo.bson.buffer.append(buf, "name", "Joe")
# Append a date/time
mongo.bson.buffer.append(buf, "created", Sys.time())
# Append a NULL
mongo.bson.buffer.append(buf, "cars", NULL)
b <- mongo.bson.from.buffer(buf)

# Display the date appended above
print(mongo.bson.value(b, "created"))
```

`mongo.code` *The mongo.code class*

Description

Objects of class "mongo.code" are used to represent javascript code values in BSON documents.

mongo.code objects' value is a string representing the value of the code.

mongo.code objects have "mongo.code" as their class so that
`mongo.bson.buffer.append()` may detect them and append the appropriate BSON code-typed value to a buffer.

These mongo.code values may also be present in a list and
will be handled properly by `mongo.bson.buffer.append.list()` and
`mongo.bson.from.list()`.

See Also

`mongo.code.create`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
code <- mongo.code.create("y = x")
mongo.bson.buffer.append(buf, "Code", code)
lst <- list(c1 = code, One = 1)
mongo.bson.buffer.append.list(buf, "listWcode", lst)
mongo.bson.buffer.append.code(buf, "Code2", "a = 1")
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "Code": (CODE) "y = x",
#   "listWcode" : { "c1"  : (CODE) "y = x",
#                   "One" : 1 },
#   "Code2" : (CODE) "a = 1" }
```

`mongo.code.create` *Create a mongo.code object*

Description

Create a mongo.code object for appending to a buffer with `mongo.bson.buffer.append()`
or for embedding in a list such that
`mongo.bson.buffer.append.list()` will properly insert a code value into the mongo.bson.buffer object.

Usage

```
mongo.code.create(code)
```

Arguments

`code`	(string) javascript code

Value

A mongo.code object

See Also

mongo.code,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
code <- mongo.code.create("y = x")
mongo.bson.buffer.append(buf, "Code", code)
lst <- list(c1 = code, One = 1)
mongo.bson.buffer.append.list(buf, "listWcode", lst)
mongo.bson.buffer.append.code(buf, "Code2", "a = 1")
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "Code": (CODE) "y = x",
#   "listWcode" : { "c1"  : (CODE) "y = x",
#                   "One" : 1 },
#   "Code2" : (CODE) "a = 1" }
```

`mongo.code.w.scope` *The mongo.code.w.scope class*

Description

Objects of class "mongo.code.w.scope" are used to represent javascript code values with scopes in BSON documents.

mongo.code.w.scope objects' value is a string representing the value of the code.

The scope is a mongo.bson object and is stored in the "scope" attribute of the mongo.code.w.scope object.

mongo.code.w.scope objects have "mongo.code.w.scope" as their class so that `mongo.bson.buffer.append()` may detect them and append the appropriate BSON code-typed value and scope to a buffer.

These mongo.code.w.scope values may also be present in a list and will be handled properly by `mongo.bson.buffer.append.list()` and `mongo.bson.from.list()`.

See Also

`mongo.code.w.scope.create`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "sv", "sx")
scope <- mongo.bson.from.buffer(buf)
codeWscope <- mongo.code.w.scope.create("y = x", scope)
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "CodeWscope", codeWscope)
lst <- list(c1 = codeWscope, One = 1)
mongo.bson.buffer.append.list(buf, "listWcodeWscope", lst)
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "CodeWscope" : (CODEWSCOPE) "y = x"
#                  (SCOPE) { "sv" : "sx"},
#   "listWcodeWscope" : { "c1" : (CODEWSCOPE) "y = x"
#                                (SCOPE) { "sv" : "sx"} } }
```

`mongo.code.w.scope.create` *Create a mongo.code.w.scope object*

Description

Create a mongo.code.w.scope object for appending to a buffer with `mongo.bson.buffer.append()` or for embedding in a list such that `mongo.bson.buffer.append.list()` will properly insert a code value into the mongo.bson.buffer object.

Usage

```
mongo.code.w.scope.create(code, scope)
```

Arguments

`code`	(string) javascript code
`scope`	(mongo.bson) the scope object

Value

A mongo.code.w.scope object

See Also

mongo.code.w.scope,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "scopevar", "scopevalue")
scope <- mongo.bson.from.buffer(buf)
codeWscope <- mongo.code.w.scope.create("y = x", scope)
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "CodeWscope", codeWscope)
b <- mongo.bson.from.buffer(buf)

# The above produces a BSON object of the form:
# { "CodeWscope" : (CODEWSCOPE) "y = x"
#                  (SCOPE) { "scopevar" : "scopevalue" } }
```

`mongo.command` *Issue a command to a database on MongoDB server*

Description

Issue a command to a MongoDB server and return the response from the server.

This function supports any of the MongoDB database commands by allowing you to specify the command object completely yourself.

See `http://www.mongodb.org/display/DOCS/List+of+Database+Commands`.

Usage

```
mongo.command(mongo, db, command)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`db`	(string) The name of the database upon which to perform the command.
`command`	(mongo.bson) An object describing the command. Alternately, `command` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

Value

NULL if the command failed. `mongo.get.err()` may be MONGO_COMMAND_FAILED.

(mongo.bson) The server's response if successful.

See Also

`mongo.get.err`,
`mongo.simple.command`,
`mongo.rename`,
`mongo.count`,
`mongo.drop.database`,
`mongo.drop`,
mongo,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {

    # alternate method of renaming a collection
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "renameCollection", "test.people")
    mongo.bson.buffer.append(buf, "to", "test.humans")
    command <- mongo.bson.from.buffer(buf)
    mongo.command(mongo, "admin", command)

    # use list notation to rename the collection back
    mongo.command(mongo, "admin",
        list(renameCollection="test.humans", to="test.people"))

    # Alternate method of counting people
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "count", "people")
    mongo.bson.buffer.append(buf, "query", mongo.bson.empty())
    command <- mongo.bson.from.buffer(buf)
    result = mongo.command(mongo, "test", command)
    if (!is.null(result)) {
        iter = mongo.bson.find(result, "n")
        print(mongo.bson.iterator.value(iter))
    }

}
```

`mongo.count` *Count records in a collection*

Description

Count the number of records in a collection that match a query See `http://www.mongodb.org/display/DOCS/Indexes`.

Usage

```
mongo.count(mongo, ns, query=mongo.bson.empty())
```

Arguments

`mongo`	(mongo) A mongo connection object.
`ns`	(string) The namespace of the collection in which to add count records.
`query`	mongo.bson The criteria with which to match records that are to be counted. The default of mongo.bson.empty() matches all records in the collection. Alternately, `query` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

Value

(double) The number of matching records.

See Also

mongo.find,
mongo.find.one,
mongo.insert,
mongo.update,
mongo.remove,
mongo,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    # Count the number of records in collection people of database test
    people.count <- mongo.count(mongo, "test.people")
    print("total people")
    print(people.count)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "age", 21L)
    query <- mongo.bson.from.buffer(buf)

    # Count the number of records in collection people of database test
    # where age == 21
    just.legal.count <- mongo.count(mongo, "test.people", query)
    print("people of age 21")
    print(just.legal.count)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.start.object(buf, "age")
    mongo.bson.buffer.append(buf, "$gte", 21L)
    mongo.bson.buffer.finish.object(buf)
    query <- mongo.bson.from.buffer(buf)

    # Count the number of records in collection people of database test
    # where age >= 21
    total.legal.count <- mongo.count(mongo, "test.people", query)
    print("people of age 21 or greater")
    print(total.legal.count)

    # shorthand using a list:
    ford.count <- mongo.count(mongo, "test.cars", list(make="Ford"))
}
```

`mongo.create` *Create an object of class "mongo"*

Description

Connect to a MongoDB server or replset and return an object of class "mongo" used for further communication over the connection.

All parameters are stored as attributes of the returned mongo object. Note that these attributes only reflect the initial parameters. Only the external data pointed to by the "mongo" attribute actually changes if, for example, mongo.timeout is called after the initial call to `mongo.create`.

Usage

```
mongo.create(host="127.0.0.1", name="",
             username="", password="", db="admin", timeout=0L)
```

Arguments

`host`	(string vector) A list of hosts/ports to which to connect. If a port is not given, 27017 is used.
`name`	(string) The name of the replset to which to connect. If name == "" (the default), the hosts are tried one by one until a connection is made. Otherwise, name must be the name of the replset and the given hosts are assumed to be seeds of the replset. Each of these is connected to and queried in turn until one reports that it is a master. This master is then queried for a list of hosts and these are in turn connected to and verified as belonging to the given replset name. When one of these reports that it is a master, that connection is used to form the actual connection as returned.
`username`	(string) The username to be used for authentication purposes. The default username of "" indicates that no user authentication is to be performed by the initial connect.
`password`	(string) The password corresponding to the given username.
`db`	(string) The name of the database upon which to authenticate the given username and password. If authentication fails, the connection is disconnected, but mongo.get.err() will indicate not indicate an error.
`timeout`	(as.integer) The number of milliseconds to wait before timing out of a network operation. The default (0L) indicates no timeout.

Value

If successful, a mongo object for use in subsequent database operations; otherwise, mongo.get.err() may be called on the returned mongo object to see why it failed.

See Also

mongo,
`mongo.is.connected`,
`mongo.disconnect`,
`mongo.reconnect`,
`mongo.get.err`,
`mongo.get.primary`,
`mongo.get.hosts`,
`mongo.get.socket`,
`mongo.set.timeout`,
`mongo.get.timeout`.

Examples

```
mongo <- mongo.create()
mongo <- mongo.create("192.168.0.3")
```

`mongo.cursor` *The mongo.cursor class*

Description

Objects of class "mongo.cursor" are returned from `mongo.find()` and used to iterate over the records matching the query.

`mongo.cursor.next(cursor)` is used to step to the first or next record.

`mongo.cursor.value(cursor)` returns a mongo.bson object representing the current record.

`mongo.cursor.destroy(cursor)` releases the resources attached to the cursor.

mongo.cursor objects have "mongo.cursor" as their class and contain an externally managed pointer to the actual cursor data. This pointer is stored in the "mongo.cursor" attribute of the object.

See Also

`mongo.find`,
`mongo.cursor.next`,
`mongo.cursor.value`,
`mongo.cursor.destroy`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "city", "St. Louis")
    query <- mongo.bson.from.buffer(buf)

    # Find the first 1000 records in collection people
    # of database test where city == "St. Louis"
    cursor <- mongo.find(mongo, "test.people", query, limit=1000L)
    # Step though the matching records and display them
    while (mongo.cursor.next(cursor))
        print(mongo.cursor.value(cursor))
    mongo.cursor.destroy(cursor)
}
```

`mongo.cursor.destroy` *Release resources attached to a cursor*

Description

`mongo.cursor.destroy(cursor)` is used to release resources attached to a cursor on both the client and server.

Note that `mongo.cursor.destroy(cursor)` may be called before all records of a result set are iterated through (for example, if a desired record is located in the result set).

Although the 'destroy' functions in this package are called automatically by garbage collection, this one in particular should be called as soon as feasible when finished with the cursor so that server resources are freed.

Usage

```
mongo.cursor.destroy(cursor)
```

Arguments

`cursor`	(mongo.cursor) A mongo.cursor object returned from `mongo.find()`.

Value

TRUE if successful; otherwise, FALSE (when an error occurs during sending the Kill Cursor operation to the server). in either case, the cursor should not be used for further operations.

See Also

`mongo.find`,
mongo.cursor,
`mongo.cursor.next`,
`mongo.cursor.value`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "city", "St. Louis")
    query <- mongo.bson.from.buffer(buf)

    # Find the first 1000 records in collection people
    # of database test where city == "St. Louis"
    cursor <- mongo.find(mongo, "test.people", query, limit=1000L)
    # Step though the matching records and display them
    while (mongo.cursor.next(cursor))
        print(mongo.cursor.destroy(cursor))
    mongo.cursor.destroy(cursor)
}
```

`mongo.cursor.next` *Advance a cursor to the next record*

Description

`mongo.cursor.next(cursor)` is used to step to the first or next record.

`mongo.cursor.value(cursor)` may then be used to examine it.

Usage

```
mongo.cursor.next(cursor)
```

Arguments

`cursor`	(mongo.cursor) A mongo.cursor object returned from `mongo.find()`.

Value

TRUE if there is a next record; otherwise, FALSE.

See Also

`mongo.find`,
mongo.cursor,
`mongo.cursor.value`,
`mongo.cursor.destroy`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "city", "St. Louis")
    query <- mongo.bson.from.buffer(buf)

    # Find the first 1000 records in collection people
    # of database test where city == "St. Louis"
    cursor <- mongo.find(mongo, "test.people", query, limit=1000L)
    # Step though the matching records and display them
    while (mongo.cursor.next(cursor))
        print(mongo.cursor.value(cursor))
    mongo.cursor.destroy(cursor)
}
```

`mongo.cursor.value` *Fetch the current value of a cursor*

Description

`mongo.cursor.value(cursor)` is used to fetch the current record belonging to a `mongo.find()` query.

Usage

```
mongo.cursor.value(cursor)
```

Arguments

`cursor`	(mongo.cursor) A mongo.cursor object returned from `mongo.find()`.

Value

(mongo.bson) The current record of the result set.

See Also

mongo.find,
mongo.cursor,
mongo.cursor.next,
mongo.cursor.value,
mongo.cursor.destroy,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "city", "St. Louis")
    query <- mongo.bson.from.buffer(buf)

    # Find the first 1000 records in collection people
    # of database test where city == "St. Louis"
    cursor <- mongo.find(mongo, "test.people", query, limit=1000L)
    # Step though the matching records and display them
    while (mongo.cursor.next(cursor))
        print(mongo.cursor.value(cursor))
    mongo.cursor.destroy(cursor)
}
```

`mongo.destroy` *Destroy a MongoDB connection*

Description

Destroy a mongo connection. The connection is disconnected first if it is still connected. No further communication is possible on the connection. Releases resources attached to the connection on both client and server.

Although the 'destroy' functions in this package are called automatically by garbage collection, this one in particular should be called as soon as feasible when finished with the connection so that server resources are freed.

Usage

```
mongo.destroy(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

NULL

See Also

mongo,
mongo.disconnect,
mongo.is.connected
mongo.reconnect.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    n_people <- mongo.count(mongo, "test.people")
    mongo.destroy(mongo)
    print(n_people)
}
```

---

mongo.disconnect *Disconnect from a MongoDB server*

Description

Disconnect from a MongoDB server. No further communication is possible on the connection. However, mongo.reconnect() may be called on the mongo object to restablish the connection.

Usage

```
mongo.disconnect(mongo)
```

Arguments

mongo	(mongo) a mongo connection object.

Value

The mongo object is returned.

See Also

mongo,
mongo.create,
mongo.reconnect,
mongo.is.connected.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    n_people <- mongo.count(mongo, "test.people")
    mongo.disconnect(mongo)
}
```

`mongo.drop` *Drop a collection from a MongoDB server*

Description

Drop a collection from a database on MongoDB server. This removes the entire collection. Obviously, care should be taken when using this command.

Usage

```
mongo.drop(mongo, ns)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`ns`	(string) The namespace of the collection to drop.

Value

(Logical) TRUE if successful; otherwise, FALSE

See Also

`mongo.drop.database`,
`mongo.command`,
`mongo.rename`,
`mongo.count`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.drop(mongo, "test.people"))

    mongo.destroy(mongo)
}
```

`mongo.drop.database` *Drop a database from a MongoDB server*

Description

Drop a database from MongoDB server. Removes the entire database and all collections in it. Obviously, care should be taken when using this command.

Usage

```
mongo.drop.database(mongo, db)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`db`	(string) The name of the database to drop.

Value

(Logical) TRUE if successful; otherwise, FALSE

See Also

`mongo.drop`,
`mongo.command`,
`mongo.rename`,
`mongo.count`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.drop.database(mongo, "test"))

    mongo.destroy(mongo)
}
```

`mongo.find` *Find records in a collection*

Description

Find records in a collection that match a given query.

See `http://www.mongodb.org/display/DOCS/Querying`.

Usage

```
mongo.find(mongo, ns, query=mongo.bson.empty(),
           sort=mongo.bson.empty(), fields=mongo.bson.empty(),
           limit=0L, skip=0L, options=0L)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`ns`	(string) namespace of the collection from which to find records.
`query`	(mongo.bson) The criteria with which to match the records to be found. The default of mongo.bson.empty() will cause the the very first record in the collection to be returned. Alternately, `query` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

`sort`	(mongo.bson) The desired fields by which to sort the returned records. The default of mongo.bson.empty() indicates that no special sorting is to be done; the records will come back in the order that indexes locate them. Alternately, `sort` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.
`fields`	(mongo.bson) The desired fields which are to be returned from the matching record. The default of mongo.bson.empty() will cause all fields of the matching record to be returned; however, specific fields may be specified to cut down on network traffic and memory overhead. Alternately, `fields` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.
`limit`	(as.integer) The maximum number of records to be returned. A limit of 0L will return all matching records not skipped.
`skip`	(as.integer) The number of matching records to skip before returning subsequent matching records.
`options`	(integer vector) Flags governing the requested operation as follows: • mongo.find.cursor.tailable • mongo.find.slave.ok • mongo.find.oplog.replay • mongo.find.no.cursor.timeout • mongo.find.await.data • mongo.find.exhaust • mongo.find.partial.results

Value

(mongo.cursor) An object of class "mongo.cursor" which is used to step through the matching records.

Note that an empty cursor will be returned if a database error occurred. `mongo.get.server.err()` and `mongo.get.server.err.string()` may be examined in that case.

See Also

`mongo.cursor`,
`mongo.cursor.next`,
`mongo.cursor.value`,
`mongo.find.one`,
`mongo.insert`,
`mongo.index.create`,
`mongo.update`,
`mongo.remove`,
mongo,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
```

```
        mongo.bson.buffer.append(buf, "age", 18L)
        query <- mongo.bson.from.buffer(buf)

        # Find the first 100 records
        #    in collection people of database test where age == 18
        cursor <- mongo.find(mongo, "test.people", query, limit=100L)
        # Step though the matching records and display them
        while (mongo.cursor.next(cursor))
            print(mongo.cursor.value(cursor))
        mongo.cursor.destroy(cursor)


        # shorthand: find all records where age=32, sorted by name,
        # and only return the name & address fields:
        cursor <- mongo.find(mongo, "test.people", list(age=32),
                             list(name=1L), list(name=1L, address=1L))
    }
```

`mongo.find.await.data` *mongo.find flag constant - await data*

Description

`mongo.find()` flag constant - await data.

Usage

```
mongo.find.await.data
```

Value

32L

`mongo.find.cursor.tailable` *mongo.find flag constant - cursor tailable*

Description

`mongo.find()` flag constant - cursor tailable.

Usage

```
mongo.find.cursor.tailable
```

Value

2L

`mongo.find.exhaust` *mongo.find flag constant - exhaust*

Description

`mongo.find()` flag constant - exhaust.

Usage

```
mongo.find.exhaust
```

Value

64L

`mongo.find.no.cursor.timeout` *mongo.find flag constant - no cursor timeout*

Description

`mongo.find()` flag constant - no cursor timeout.

Usage

```
mongo.find.no.cursor.timeout
```

Value

16L

`mongo.find.one` *Find one record in a collection*

Description

Find the first record in a collection that matches a given query.

This is a simplified version of mongo.find() which eliminates the need to step through returned records with a cursor.

See `http://www.mongodb.org/display/DOCS/Querying`.

Usage

```
mongo.find.one(mongo, ns, query=mongo.bson.empty(),
                          fields=mongo.bson.empty())
```

Arguments

`mongo`	(mongo) A mongo connection object.
`ns`	(string) The namespace of the collection from in which to find a record.
`query`	(mongo.bson) The criteria with which to match the record that is to be found. The default of mongo.bson.empty() will cause the the very first record in the collection to be returned. Alternately, `query` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.
`fields`	(mongo.bson) The desired fields which are to be returned frtom the matching record. The default of mongo.bson.empty() will cause all fields of the matching record to be returned; however, specific fields may be specified to cut down on network traffic and memory overhead. Alternately, `fields` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

Value

NULL if no record matching the criteria is found; otherwise,

(mongo.bson) The matching record/fields.

Note that NULL may also be returned if a database error occurred (when a badly formed query is used, for example). `mongo.get.server.err` and `mongo.get.server.err.string` may be examined in that case.

See Also

`mongo.find`,
`mongo.index.create`,
`mongo.insert`,
`mongo.update`,
`mongo.remove`,
mongo,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Jeff")
    query <- mongo.bson.from.buffer(buf)

    # find the first record where name is "Jeff"\
    #    in collection people of database test
    b <- mongo.find.one(mongo, "test.people", query)
    if (!is.null(b))
        print(b)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "_id", 1L)
    mongo.bson.buffer.append(buf, "age", 1L)
    fields <- mongo.bson.from.buffer(buf)
    # find the first record where name is "Jeff"
    #    in collection people of database test
    # return only the _id and age fields of the matched record
    b <- mongo.find.one(mongo, "test.people", query, fields)
    if (!is.null(b))
        print(b)

    # find the first record in collection cars of database test
    have.car <- !is.null(mongo.find.one(mongo, "test.cars"))

    # shorthand using a list:
    b <- mongo.find.one(mongo, "test.people", list(name="Jose"))
}
```

`mongo.find.oplog.replay` *mongo.find flag constant - oplog replay*

Description

`mongo.find()` flag constant - oplog replay.

Usage

```
mongo.find.oplog.replay
```

Value

8L

`mongo.find.partial.results` *mongo.find flag constant - partial results*

Description

`mongo.find()` flag constant - partial results.

Usage

```
mongo.find.partial.results
```

Value

128L

`mongo.find.slave.ok` *mongo.find flag constant - slave ok*

Description

`mongo.find()` flag constant - slave ok.

Usage

```
mongo.find.slave.ok
```

Value

4L

`mongo.get.database.collections` *Get a list of collections in a database*

Description

Get a list of collections in a database on a MongoDB server.

Usage

```
mongo.get.database.collections(mongo, db)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`db`	(string) Name of the database for which to get the list of collections.

Value

(string vector) List of collection namespaces in the given database.

Note this will not include the system collection *db*.system.indexes nor the indexes attached to the database. Use mongo.find(mongo, '*db*.system.indexes", limit=0L) for information on any indexes.

See Also

`mongo.get.databases`,
`mongo.drop.database`,
`mongo.drop`,
`mongo.command`,
`mongo.rename`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.get.database.collections(mongo, "test"))

    mongo.destroy(mongo)
}
```

---

mongo.get.databases

Get a list of databases from a MongoDB server

Description

Get a list of databases from a MongoDB server.

Usage

```
mongo.get.databases(mongo)
```

Arguments

`mongo`	(mongo) A mongo connection object.

Value

(string vector) List of databases. Note this will not include the system databases "admin" and "local".

See Also

`mongo.get.database.collections`,
`mongo.drop.database`,
`mongo.command`,
`mongo.rename`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.get.databases(mongo))

    mongo.destroy(mongo)
}
```

`mongo.get.err` *Retrieve an connection error code from a mongo object*

Description

Retrieve an connection error code from a mongo object indicating the failure code if mongo.create() failed.

Usage

```
mongo.get.err(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

(integer) error code as follows:

`0`	No Error

mongo.create() errors:

`1`	No socket - Could not create socket.
`2`	Fail - An error occurred attempting to connect to socket
`3`	Address fail - An error occured calling getaddrinfo().
`4`	Not Master - Warning: connected to a non-master node (read-only).
`5`	Bad set name - given name doesn't match the replica set.
`6`	No Primary - Cannot find primary in replica set - connection closed.

Other errors:

`7`	I/O error - An error occured reading or writing on the socket.
`8`	Read size error - The response is not the expected length.
`9`	Command failed - The command returned with 'ok' value of 0.
`10`	BSON invalid - Not valid for the specified operation.
`11`	BSON not finished - should not occur with R driver.

See Also

`mongo.create`,
mongo

Examples

```
mongo <- mongo.create()
if (!mongo.is.connected(mongo)) {
    print("Unable to connect.  Error code:")
    print(mongo.get.err(mongo))
}
```

`mongo.get.hosts` *Get a lists of hosts & ports as reported by a replica set master upon connection creation.*

Description

Get a lists of hosts & ports as reported by a replica set master upon connection creation.

Usage

```
mongo.get.hosts(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

NULL if a replica set was not connected to; otherwise, a list of host & port strings in the format "

See Also

`mongo.create`,
mongo

Examples

```
mongo <- mongo.create(c("127.0.0.1", "192.168.0.3"), name="Inventory")
if (mongo.is.connected(mongo))
    print(mongo.get.hosts(mongo))
```

`mongo.get.last.err` *Retrieve an server error code from a mongo connection object*

Description

Retrieve an server error record from a the MongoDB server. This describes the last error that occurs while accessing the give database. While this function retrieves an error record in the form of a mongo.bson record, it also sets the values returned by `mongo.get.server.err()` and `mongo.get.server.err.string()`. You may find it more convenient using those after calling `mongo.get.last.err()` rather than unpacking the returned mongo.bson object.

Usage

```
mongo.get.last.err(mongo, db)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`db`	(string) The name of the database for which to get the error status.

Value

NULL if no error was reported; otherwise,

(mongo.bson) This BSON object has the form { err : "*error message string*", code : *error code integer* }

See Also

`mongo.get.server.err`,
`mongo.get.server.err.string`,
`mongo.get.prev.err`
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {

    # try adding a duplicate record when index doesn't allow this

    db <- "test"
    ns <- "test.people"
    mongo.index.create(mongo, ns, "name", mongo.index.unique)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "John")
    mongo.bson.buffer.append(buf, "age", 22L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b);

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "John")
    mongo.bson.buffer.append(buf, "age", 27L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b);

    err <- mongo.get.last.err(mongo, db)
    print(mongo.get.server.err(mongo))
    print(mongo.get.server.err.string(mongo))
}
```

`mongo.get.prev.err` *Retrieve an server error code from a mongo connection object*

Description

Retrieve the previous server error record from a the MongoDB server. While this function retrieves an error record in the form of a mongo.bson record, it also sets the values returned by `mongo.get.server.err()` and `mongo.get.server.err.string()`. You may find it more convenient using those after calling `mongo.get.prev.err()` rather than unpacking the returned mongo.bson object.

Usage

```
mongo.get.prev.err(mongo, db)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`db`	(string) The name of the database for which to get the error status.

Value

NULL if no error was reported; otherwise,

(mongo.bson) This BSON object has the form { err : "*error message string*", code : *error code integer* }

See Also

`mongo.get.server.err`,
`mongo.get.server.err.string`,
`mongo.get.last.err`
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {

    # try adding a duplicate record when index doesn't allow this

    db <- "test"
    ns <- "test.people"
    mongo.index.create(mongo, ns, "name", mongo.index.unique)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "John")
    mongo.bson.buffer.append(buf, "age", 22L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b);

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "John")
    mongo.bson.buffer.append(buf, "age", 27L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b);

    # try insert again
    mongo.insert(mongo, ns, b);

    err <- mongo.get.prev.err(mongo, db)
    print(mongo.get.server.err(mongo))
    print(mongo.get.server.err.string(mongo))
}
```

mongo.get.primary *Get the host & port of the server to which a mongo object is connected.*

Description

Get the host & port of the server to which a mongo object is connected.

Usage

```
mongo.get.primary(mongo)
```

Arguments

mongo	(mongo) a mongo connection object.

Value

String host & port in the format "%s:%d".

See Also

mongo.create,
mongo.

Examples

```
mongo <- mongo.create(c("127.0.0.1", "192.168.0.3"))
if (mongo.is.connected(mongo)) {
    print(mongo.get.primary(mongo))
}
```

mongo.get.server.err *Retrieve an server error code from a mongo connection object*

Description

Retrieve an server error code from a mongo connection object.

mongo.find(), mongo.find.one(), mongo.index.create() set or clear this error code depending on whether they are successful or not.

mongo.get.last.err() and mongo.get.prev.err() both set or clear this error code according to what the server reports.

Usage

```
mongo.get.server.err(mongo)
```

Arguments

mongo	(mongo) a mongo connection object.

Value

(integer) Server error code

See Also

`mongo.get.server.err.string`,
`mongo.get.last.err`,
`mongo.get.prev.err`,
`mongo.find`,
`mongo.find.one`,
`mongo.index.create`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    # construct a query containing invalid operator
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.start.object(buf, "age")
    mongo.bson.buffer.append(buf, "$bad", 1L)
    mongo.bson.buffer.finish.object(buf)
    query <- mongo.bson.from.buffer(buf)

    result <- mongo.find.one(mongo, "test.people", query)
    if (is.null(result)) {
        print(mongo.get.server.err.string(mongo))
        print(mongo.get.server.err(mongo))
    }
}
```

`mongo.get.server.err.string` *Retrieve an server error code from a mongo connection object*

Description

Retrieve an server error string from a mongo connection object.

`mongo.find()`, `mongo.find.one()`, `mongo.index.create()` set or clear this error string depending on whether they are successful or not.

`mongo.get.last.err()` and `mongo.get.prev.err()` both set or clear this error string according to what the server reports.

Usage

```
mongo.get.server.err.string(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

(string) Server error string

See Also

`mongo.get.server.err`,
`mongo.get.last.err`,
`mongo.get.prev.err`,
`mongo.find`,
`mongo.find.one`,
`mongo.index.create`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    # construct a query containing invalid operator
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.start.object(buf, "age")
    mongo.bson.buffer.append(buf, "$bad", 1L)
    mongo.bson.buffer.finish.object(buf)
    query <- mongo.bson.from.buffer(buf)

    result <- mongo.find.one(mongo, "test.people", query)
    if (is.null(result)) {
        print(mongo.get.server.err(mongo))
        print(mongo.get.server.err.string(mongo))
    }
}
```

`mongo.get.socket` *Get the socket assigned to a mongo object by mongo.create().*

Description

Get the the low-level socket number assigned to the given mongo object by mongo.create().

Usage

```
mongo.get.socket(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

Integer socket number

See Also

`mongo.create`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo))
    print(mongo.get.socket(mongo))
```

`mongo.get.timeout` *Get the timeout value of a mongo connection*

Description

Get the timeout value for network operations on a mongo connection.

Usage

```
mongo.get.timeout(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

(integer) timeout value in milliseconds.

See Also

`mongo.set.timeout`,
`mongo.create`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    mongo.set.timeout(mongo, 2000L)
    timeout <- mongo.get.timeout(mongo)
    if (timeout != 2000L)
        error("expected timeout of 2000");
}
```

mongo.gridfile *The mongo.gridfile class*

Description

Objects of class "mongo.gridfile" are used to access gridfiles on a MongoDB server. They are created by mongo.gridfs.find().

mongo.gridfile objects have "mongo.gridfile" as their class and contain an externally managed pointer to the actual data used to manage operations on the gridfile. This pointer is stored in the "mongo.gridfile" attribute of the object. The object also has a "mongo.gridfs" attribute holding a pointer to the mongo.gridfs object used in creation to prevent garbage collection on the mongo.gridfs object while the mongo.gridfile is still active.

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile.get.filename,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.filename,
mongo.gridfile.get.length,
mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.md5,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,
mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    test.R <- file("test2.R")
    mongo.gridfile.pipe(gf, test.R)

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

mongo.gridfile.destroy

Destroy a mongo.gridfile object

Description

Releases the resources associated with a mongo.gridfile object.
These are created by `mongo.gridfs.find()`.

It is not absolutely necessary to call this function since R's garbage collection will eventually get around to doing it for you.

Usage

```
mongo.gridfile.destroy(gridfile)
```

Arguments

`gridfile`	A (mongo.gridfile) object.

Value

NULL

See Also

`mongo.gridfs.find`,
mongo.gridfile,
mongo.gridfs.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.upload.date(gf))
    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

mongo.gridfile.get.chunk

Get a chunk of a mongo.gridfile

Description

Get a chunk of a mongo.gridfile.

Usage

```
mongo.gridfile.get.chunk(gridfile, i)
```

Arguments

`gridfile`	A (mongo.gridfile) object.
`i`	(integer) The index of the chunk to fetch. This should be in the range 0 to `mongo.gridfile.get.chunk.count(gridfile) - 1`.

Value

(mongo.bson) the *i*th chunk of *gridfile* if successful; otherwise, NULL.

The value returned is the *i*th document in the 'chunks' collection of the GridFS. The 'data' field of this document contains the actual data belonging to the chunk.
See `http://www.mongodb.org/display/DOCS/GridFS+Specification`.

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.size`,
`mongo.gridfile.get.chunk.count`,
`mongo.gridfile.get.content.type`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunks`,
`mongo.gridfile.read`,
`mongo.gridfile.seek`,
`mongo.gridfile.pipe`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    chunk <- mongo.gridfile.get.chunk(gf, 0)
    iter <- mongo.bson.find(chunk, "data")

    f <- file("testChunk0.R", "wb")
    # write the binary (raw) data to a file
    writeBin(mongo.bson.iterator.value(iter), f)
    close(f)

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.get.chunk.count` *Get the chunk count of a mongo.gridfile*

Description

Get the chunk count of a mongo.gridfile. This is the number of chunks into which the gridfile is broken up on the server.

Usage

```
mongo.gridfile.get.chunk.count(gridfile)
```

Arguments

`gridfile`	A (mongo.gridfile) object.

Value

(integer) The chunk count of *gridfile*

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.size`,
`mongo.gridfile.get.content.type`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunk`,
`mongo.gridfile.get.chunks`,
`mongo.gridfile.read`,
`mongo.gridfile.seek`,
`mongo.gridfile.pipe`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.chunk.count(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.get.chunk.size` *Get the chunk.size of a mongo.gridfile*

Description

Get the chunk size of a mongo.gridfile. This is the size of the chunks into which file is broken up on the server.

Usage

```
mongo.gridfile.get.chunk.size(gridfile)
```

Arguments

`gridfile`	A (mongo.gridfile) object.

Value

(integer) The chunk size of *gridfile*.

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.count`,
`mongo.gridfile.get.content.type`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunk`,
`mongo.gridfile.get.chunks`,
`mongo.gridfile.read`,
`mongo.gridfile.seek`,
`mongo.gridfile.pipe`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.chunk.size(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

mongo.gridfile.get.chunks
Get a cursor for a range of chunks in a mongo.gridfile

Description

Get a mongo.cursor for a range of chunks in a mongo.gridfile.

Usage

```
mongo.gridfile.get.chunks(gridfile, start, count)
```

Arguments

`gridfile`	A (mongo.gridfile) object.
`start`	(integer) The index of the first chunk to fetch. This should be in the range 0 to `mongo.gridfile.get.chunk.count(gridfile) - 1`.
`count`	(integer) The number of chunks to fetch.

Value

(mongo.cursor) A cursor to be used to step through the requested chunks.

The values returned by `mongo.cursor.value()` will be consecutive documents in the 'chunks' collection of the GridFS. The 'data' field of these documents contains the actual data belonging to the chunk. See `http://www.mongodb.org/display/DOCS/GridFS+Specification`.

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.size`,
`mongo.gridfile.get.chunk.count`,
`mongo.gridfile.get.content.type`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunk`,
`mongo.gridfile.read`,
`mongo.gridfile.seek`,
`mongo.gridfile.pipe`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    gf <- mongo.gridfs.find(gridfs, "rmongodb.pdf")
    cursor <- mongo.gridfile.get.chunks(gf, 1, 2)

    f <- file("rmongodb.pdf.chunks12", "wb")
    while (mongo.cursor.next(cursor)) {
        chunk <- mongo.cursor.value(cursor)
        iter <- mongo.bson.find(chunk, "data")

        # write the binary (raw) data to the file
        writeBin(mongo.bson.iterator.value(iter), f)
    }
    close(f)
    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.get.content.type` *Get the content type of a mongo.gridfile*

Description

Get the MIME content type of a mongo.gridfile.

Usage

```
mongo.gridfile.get.content.type(gridfile)
```

Arguments

`gridfile`	A (mongo.gridfile) object.

Value

(string) The content.type (remote name) of *gridfile*. This may be an empty string if no content type is associated with the gridfile.

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.size`,
`mongo.gridfile.get.chunk.count`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunk`,
`mongo.gridfile.get.chunks`,
`mongo.gridfile.read`,

mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.content.type(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.get.descriptor

Get the descriptor of a mongo.gridfile

Description

Get the descriptor of a mongo.gridfile. This descriptor is the document describing the given gridfile as stored on the MongoDB server in the 'files' collection of the GridFS .

See http://www.mongodb.org/display/DOCS/GridFS+Specification.

Usage

```
mongo.gridfile.get.descriptor(gridfile)
```

Arguments

gridfile	A (mongo.gridfile) object.

Value

(mongo.bson) The descriptor of *gridfile*.

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.filename,
mongo.gridfile.get.length,
mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.md5,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,

mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.descriptor(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.get.filename

Get the filename of a mongo.gridfile

Description

Get the filename of a mongo.gridfile. This is the ’remote name’ that is used identify the file on the server.

Usage

```
mongo.gridfile.get.filename(gridfile)
```

Arguments

gridfile	A (mongo.gridfile) object.

Value

(string) The filename (remote name) of *gridfile*

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.length,
mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.md5,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,

mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    # find a GridFS file uploaded midnight July 4, 2008
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "uploadDate",
        strptime("07-04-2008", "%m-%d-%Y"))
    query <- mongo.bson.from.buffer(buf)

    gf <- mongo.gridfs.find(gridfs, query)
    if (!is.null(gf)) {
        print(mongo.gridfile.get.filename(gf))

        mongo.gridfile.destroy(gf)
    }
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.get.length

Get the length of a mongo.gridfile

Description

Get the length of a mongo.gridfile.

Usage

```
mongo.gridfile.get.length(gridfile)
```

Arguments

gridfile	A (mongo.gridfile) object.

Value

(double) The length of *gridfile*.

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.filename,

mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.md5,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,
mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.length(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

mongo.gridfile.get.md5

Get the MD5 hash of a mongo.gridfile

Description

Get the MD5 hash of a mongo.gridfile.

Usage

```
mongo.gridfile.get.md5(gridfile)
```

Arguments

gridfile	A (mongo.gridfile) object.

Value

(string) The MD5 hash (32 hex digits) of *gridfile*.

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.filename,
mongo.gridfile.get.length,

mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,
mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.md5(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.get.metadata
Get the metadata of a mongo.gridfile

Description

Get the metadata of a mongo.gridfile. Some applications may store metadata pertaining to a GridFS file in the "metadata" field of the descriptor. (See `mongo.gridfile.get.descriptor()`. This function retrieves that field as a mongo.bson object.

Usage

```
mongo.gridfile.get.metadata(gridfile)
```

Arguments

gridfile	A (mongo.gridfile) object.

Value

(mongo.bson) The metadata of *gridfile* if present; otherwise, NULL.

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.filename,

mongo.gridfile.get.length,
mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.md5,
mongo.gridfile.get.chunk,
mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.metadata(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.get.upload.date

Get the upload date of a mongo.gridfile

Description

Get the upload date of a mongo.gridfile.

Usage

```
mongo.gridfile.get.upload.date(gridfile)
```

Arguments

gridfile	A (mongo.gridfile) object.

Value

(POSIXct) The upload date/time of *gridfile*.

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.filename,
mongo.gridfile.get.length,

mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.md5,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,
mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek,
mongo.gridfile.pipe.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(mongo.gridfile.get.upload.date(gf))

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.pipe

Pipe a mongo.gridfile to an R connection

Description

Pipe a mongo.gridfile to an R connection. This outputs the entire GridFS file to a connection. If the connection is open, it must be in binary output mode; otherwise, the connection is opened in binary output mode and closed afterwards.

Usage

```
mongo.gridfile.pipe(gridfile, con)
```

Arguments

`gridfile`	A (mongo.gridfile) object.
`con`	(connection) An R connection object.

Value

NULL

See Also

mongo.gridfs,
mongo.gridfs.find,
mongo.gridfile,
mongo.gridfile.get.descriptor,
mongo.gridfile.get.filename,
mongo.gridfile.get.length,
mongo.gridfile.get.chunk.size,
mongo.gridfile.get.chunk.count,
mongo.gridfile.get.content.type,
mongo.gridfile.get.upload.date,
mongo.gridfile.get.md5,
mongo.gridfile.get.metadata,
mongo.gridfile.get.chunk,
mongo.gridfile.get.chunks,
mongo.gridfile.read,
mongo.gridfile.seek.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "rmongodb.pdf")
    if (!is.null(gf)) {
        f <- file("mongodb_copy.pdf")
        mongo.gridfile.pipe(gf, f)
    }

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfile.read *Read raw data from a mongo.gridfile*

Description

Read raw data from a mongo.gridfile. The data read may span multiple chunks.

A mongo.gridfile file maintains a current read position which is advanced by the size of each read. This position is initially at offset 0.

Since this function returns raw data, you may want to use R's `readBin()` to unpack it.

Usage

```
mongo.gridfile.read(gridfile, size)
```

Arguments

`gridfile`	A (mongo.gridfile) object.
`size`	(as.double) The number of bytes to read.

Value

(raw) The data read from emphgridfile. The length of this vector may be less than the requested size if there was not enough data remaining to be read. This length could also be 0 if an error occured during the operation. Check `mongo.get.err()` of the associated mongo connection object in this case.

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.size`,
`mongo.gridfile.get.chunk.count`,
`mongo.gridfile.get.content.type`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunk`,
`mongo.gridfile.get.chunks`,
`mongo.gridfile.seek`,
`mongo.gridfile.pipe`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "rmongodb.pdf")
    mongo.gridfile.seek(gf, 256*256*5)
    data <- mongo.gridfile.read(gf, 16384)

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.seek` *Seek to a position in a mongo.gridfile*

Description

Seek to a position in a mongo.gridfile.
This sets the position at which the next `mongo.gridfile.read()` will start.

Usage

```
mongo.gridfile.seek(gridfile, offset)
```

Arguments

`gridfile`	A (mongo.gridfile) object.
`offset`	(as.double) The position to which to seek.

Value

(double) The position set. This may be at the length of the GridFS file if *offset* was greater than that.

See Also

`mongo.gridfs`,
`mongo.gridfs.find`,
mongo.gridfile,
`mongo.gridfile.get.descriptor`,
`mongo.gridfile.get.filename`,
`mongo.gridfile.get.length`,
`mongo.gridfile.get.chunk.size`,
`mongo.gridfile.get.chunk.count`,
`mongo.gridfile.get.content.type`,
`mongo.gridfile.get.upload.date`,
`mongo.gridfile.get.md5`,
`mongo.gridfile.get.metadata`,
`mongo.gridfile.get.chunk`,
`mongo.gridfile.get.chunks`,
`mongo.gridfile.read`,
`mongo.gridfile.pipe`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "rmongodb.pdf")
    mongo.gridfile.seek(gf, 256*256*5)
    data <- mongo.gridfile.read(gf, 16384)

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.writer` *The mongo.gridfile.writer class*

Description

Objects of class "mongo.gridfile.writer" are used to buffer multiple writes to a single GridFS file.

Use `mongo.gridfile.writer.create` to create an object of this class,
`mongo.gridfile.writer.write` to write data to it, and
`mongo.gridfile.writer.finish` when done writing.

mongo.gridfile.writer objects have "mongo.gridfile.writer" as their class and contain an externally managed pointer to the actual data used to manage operations on the GridFS. This pointer is stored in the "mongo.gridfile" attribute of the object. The object also has a "mongo.gridfs" attribute holding a pointer to the mongo.gridfs object used in creation to prevent garbage collection on the mongo.gridfs object while the mongo.gridfile.writer is still active.

See Also

mongo.gridfs,
`mongo.gridfile.writer.create`,
`mongo.gridfile.writer.write`,
`mongo.gridfile.writer.finish`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gfw <- mongo.gridfile.writer.create(gridfs, "test.dat")

    # store 4 bytes
    mongo.gridfile.writer.write(gfw, charToRaw("test"))

    # store string & LF plus 0-byte terminator
    buf <- writeBin("Test\n", as.raw(1))
    mongo.gridfile.writer.write(gfw, buf)

    # store PI as a float
    buf <- writeBin(3.1415926, as.raw(1), size=4, endian="little")
    mongo.gridfile.writer.write(gfw, buf)

    mongo.gridfile.writer.finish(gfw)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.writer.create` *Create a mongo.gridfile.writer object*

Description

Create a mongo.gridfile.writer object used to buffer many writes to a single GridFS file. Once the mongo.gridfile.writer is created, use `mongo.gridfile.writer.write()` to write data to the buffered GridFS file and `mongo.gridfile.writer.finish()` when done.

Usage

```
mongo.gridfile.writer.create(gridfs, remotename, contenttype="")
```

Arguments

`gridfs`	A (mongo.gridfs) object.
`remotename`	(string) The name the file will be known as within the GridFS.
`contenttype`	(string) Optional MIME content type.

Value

(mongo.gridfile.writer) The object to be used for writing to the GridFS file.

See Also

mongo.gridfs,
`mongo.gridfs.create`,
`mongo.gridfile.writer.write`,
`mongo.gridfile.writer.finish`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gfw <- mongo.gridfile.writer.create(gridfs, "test.dat")

    # store 4 bytes
    mongo.gridfile.writer.write(gfw, charToRaw("test"))

    # store string & LF plus 0-byte terminator
    buf <- writeBin("Test\n", as.raw(1))
    mongo.gridfile.writer.write(gfw, buf)

    # store PI as a float
    buf <- writeBin(3.1415926, as.raw(1), size=4, endian="little")
    mongo.gridfile.writer.write(gfw, buf)

    mongo.gridfile.writer.finish(gfw)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.writer.finish` *Finish writing to a buffered GridFS file*

Description

Finish writing to a buffered GridFS file. This function flushes any partial buffer and finalizes the operation.

Usage

```
mongo.gridfile.writer.finish(gfw)
```

Arguments

`gfw`	A (mongo.gridfile.writer) object.

Value

TRUE, if successfil; false, if an error occurred.

See Also

mongo.gridfs,
`mongo.gridfile.writer.create`,
mongo.gridfile.writer,
`mongo.gridfile.writer.write`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gfw <- mongo.gridfile.writer.create(gridfs, "test.dat")

    # store 4 bytes
    mongo.gridfile.writer.write(gfw, charToRaw("test"))

    # store string & LF plus 0-byte terminator
    buf <- writeBin("Test\n", as.raw(1))
    mongo.gridfile.writer.write(gfw, buf)

    # store PI as a float
    buf <- writeBin(3.1415926, as.raw(1), size=4, endian="little")
    mongo.gridfile.writer.write(gfw, buf)

    mongo.gridfile.writer.finish(gfw)
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfile.writer.write` *Write raw data to a buffered GridFS file*

Description

Write raw data to a buffered GridFS file. The data is buffered and sent to the server in 256k chunks as it accumulates.

This function only handles the RAW type. Use `writeBin()` as necessary to pack your data appropriately for storage. See the examples and R's documentation on `writeBin()`.

Use `mongo.gridfs.store()` when you only need to write one data packet as a complete GridFS file.

Usage

```
mongo.gridfile.writer.write(gfw, raw)
```

Arguments

`gfw`	A (mongo.gridfile.writer) object.
`raw`	(raw) The data to write to the GridFS file.

Value

NULL

See Also

mongo.gridfs,
`mongo.gridfile.writer.create`,
mongo.gridfile.writer,
`mongo.gridfile.writer.finish`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gfw <- mongo.gridfile.writer.create(gridfs, "test.dat")

    # store 4 bytes
    mongo.gridfile.writer.write(gfw, charToRaw("test"))

    # store string & LF plus 0-byte terminator
    buf <- writeBin("Test\n", as.raw(1))
    mongo.gridfile.writer.write(gfw, buf)

    # store PI as a float
    buf <- writeBin(3.1415926, as.raw(1), size=4, endian="little")
    mongo.gridfile.writer.write(gfw, buf)

    mongo.gridfile.writer.finish(gfw)

    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfs` *The mongo.gridfs class*

Description

Objects of class "mongo.gridfs" are used to store and/or access a "Grid File System" (GridFS) on a MongoDB server. While primarily intended to store large documents that won't fit on the server as a single BSON object, GridFS may also be used to store large numbers of smaller files.

See http://www.mongodb.org/display/DOCS/GridFS and
http://www.mongodb.org/display/DOCS/When+to+use+GridFS.

mongo.gridfs objects have "mongo.gridfs" as their class and contain an externally managed pointer to the actual data used to manage operations on the GridFS.
This pointer is stored in the "mongo.gridfs" attribute of the object. The object also has a "mongo" attribute holding a pointer to the mongo connection object used in creation to prevent garbage collection on the mongo object while the mongo.gridfile is still active.

Objects of class "mongo.gridfile" are used to access gridfiles and read from them.

Objects of class "mongo.gridfile.writer" are used to write data to the GridFS.

See Also

`mongo.gridfs.destroy`,
`mongo.gridfs.store.file`,
`mongo.gridfs.remove.file`,
`mongo.gridfs.store`,
`mongo.gridfile.writer.create`,
`mongo.gridfs.find`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    # Copy a local file to the server as a GridFS file
    mongo.gridfs.store.file(gridfs, "../test.R", "test.R")

    # locate the file on the server
    gf <- mongo.gridfs.find(gridfs, "test.R")
    print(gf)
    # and pipe it to an R connection object
    test.R <- file("test2.R")
    mongo.gridfile.pipe(gf, test.R)

    mongo.gridfile.destroy(gf)
    mongo.gridfs.destroy(gridfs)
}
```

---

mongo.gridfs.create

Create a mongo.gridfs object

Description

Create a mongo.gridfs object used to access and store "grid files" on the MongoDB server.

Usage

```
mongo.gridfs.create(mongo, db, prefix="fs")
```

Arguments

`mongo`	A (mongo) connection object.
`db`	(string) The name of the database in which to access and/or store the gridfs-related collections.
`prefix`	(string) The prefix to use constructing the gridfs-related collection names. There are two collections used for this purpose: \"*db.prefix*.files\" and \"*db.prefix*.chunks\".

Value

(mongo.gridfs) An object to be used for subsequent operations on the grid file store.

See Also

mongo.gridfs,
`mongo.gridfs.destroy`,
`mongo.gridfs.store.file`,
`mongo.gridfs.remove.file`,
`mongo.gridfs.store`,
`mongo.gridfile.writer.create`,
`mongo.gridfs.find`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    # Copy a local file to the server as a gridfs file
    mongo.gridfs.store.file(gridfs, "../test.R", "test.R")
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfs.destroy` *Destroy a mongo.gridfs object*

Description

Releases the resources associated with a mongo.gridfs object.

It is not absolutely necessary to call this function since R's garbage collection will eventually get around to doing it for you.

Usage

```
mongo.gridfs.destroy(gridfs)
```

Arguments

`gridfs`	A (mongo.gridfs) object.

Value

NULL

See Also

mongo.gridfs,
`mongo.gridfs.create`,
`mongo.gridfs.store.file`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    # Copy a local file to the server as a gridfs file
    mongo.gridfs.store.file(gridfs, "../test.R", "test.R")
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfs.find` *Find a GridFS file*

Description

Find a GridFS file and return a mongo.gridfile object used for further operations on it

Usage

```
mongo.gridfs.find(gridfs, query)
```

Arguments

`gridfs`	A (mongo.gridfs) object.
`query`	(string) The name of the GridFS file to locate. This parameter may also be a mongo.bson query object and is used to search the GridFS "files" collection documents for matches. Alternately, `query` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

Value

NULL, if not found; otherwise, a mongo.gridfile object corresponding to the found GridFS file.

See Also

mongo.gridfile,
`mongo.gridfile.get.filename`,
mongo.gridfs.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")

    gf <- mongo.gridfs.find(gridfs, "test.dat")
    print(mongo.gridfile.get.length(gf))

    # find a GridFS file uploaded midnight July 4, 2008
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "uploadDate",
        strptime("07-04-2008", "%m-%d-%Y"))
    query <- mongo.bson.from.buffer(buf)
    gf <- mongo.gridfs.find(gridfs, query)

    if (!is.null(gf))
        print(mongo.gridfile.get.filename(gf))

    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfs.remove.file` *Remove a file from a GridFS on a MongoDB server*

Description

Remove a file from a GridFS on a MongoDB server.

Usage

```
mongo.gridfs.remove.file(gridfs, remotename)
```

Arguments

`gridfs`	A (mongo.gridfs) object.
`remotename`	(string) The name of the file to be removed (as known within the GridFS).

Value

NULL

See Also

mongo.gridfs,
`mongo.gridfs.store.file`
`mongo.gridfs.store`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    ## Not run: mongo.gridfs.remove.file(gridfs, "test.R")
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.gridfs.store` *Store raw data as a file in a GridFS*

Description

Store raw data as a file to a GridFS on a MongoDB server. This function stores the entire piece of data file on the server, breaking it up into 256K chunks as necessary.

This function only handles the RAW type. Use `writeBin()` as necessary to pack your data appropriately for storage. See the examples and R's documentation on `writeBin()`.

Use mongo.gridfile.writer when you need to buffer many writes to a GridFS file.

Usage

```
mongo.gridfs.store(gridfs, raw, remotename, contenttype="")
```

Arguments

`gridfs`	A (mongo.gridfs) object.
`raw`	(raw) The data to store on the server.
`remotename`	(string) The name the file will be known as within the GridFS.
`contenttype`	(string) Optional MIME content type.

Value

TRUE, if successful; FALSE, if an error occured during the operation.

See Also

mongo.gridfs,
`mongo.gridfs.create`,
`mongo.gridfs.remove.file`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    # store 4 bytes
    mongo.gridfs.store(gridfs, charToRaw("test"), "test4.dat")

    # store string & LF plus 0-byte terminator
    buf <- writeBin("Test\n", as.raw(1))
    mongo.gridfs.store(gridfs, buf, "test6.dat")
    # store PI as a float
    buf <- writeBin(3.1415926, as.raw(1), size=4, endian="little")
    mongo.gridfs.store(gridfs, buf, "PI.dat")

    mongo.gridfs.destroy(gridfs)
}
```

mongo.gridfs.store.file

Store a file into a GridFS on a MongoDB server

Description

Store a file into a GridFS on a MongoDB server. This function stores the entire given file on the server, breaking it up into 256K chunks as necessary.

Usage

```
mongo.gridfs.store.file(gridfs, filename, remotename="",
                         contenttype="")
```

Arguments

gridfs	A (mongo.gridfs) object.
filename	(string) The path/filename of the file to copy to the server.
remotename	(string) The name the file will be known as within the GridFS. If remotename=="" (the default), the remote file will be known by the given *filename*.
contenttype	(string) Optional MIME content type.

Value

TRUE, if successful; FALSE, if an error occured during the operation.

See Also

mongo.gridfs,
mongo.gridfs.create,
mongo.gridfs.remove.file.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    gridfs <- mongo.gridfs.create(mongo, "grid")
    # Copy a local file to the server as a gridfs file
    mongo.gridfs.store.file(gridfs, "../test.R", "test.R")
    mongo.gridfs.destroy(gridfs)
}
```

`mongo.index.background` *mongo.index.create flag constant - background*

Description

`mongo.index.create()` flag constant - background.

Usage

```
mongo.index.background
```

Value

8L

`mongo.index.create` *Add an index to a collection*

Description

Add an index to a collection.

See `http://www.mongodb.org/display/DOCS/Indexes`.

Usage

```
mongo.index.create(mongo, ns, key, options=0L)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`ns`	(string) The namespace of the collection to which to add an index.
`key`	An object enumerating the fields in order which are to participate in the index. This object may be a vector of strings listing the key fields or a mongo.bson object containing the key fields in the desired order. Alternately, `key` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.
`options`	(integer vector) Optional flags governing the operation: • `mongo.index.unique` • `mongo.index.drop.dups` • `mongo.index.background` • `mongo.index.sparse`

Value

NULL if successful; otherwise, a mongo.bson object describing the error.
`mongo.get.server.err()` or `mongo.get.server.err.string()` may alternately be called in this case instead of examining the returned object.

See Also

mongo.find,
mongo.find.one,
mongo.insert,
mongo.update,
mongo.remove,
mongo,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    # Add a city index to collection people in database test
    b <- mongo.index.create(mongo, "test.people", "city")
    if (!is.null(b)) {
        print(b)
        stop("Server error")
    }

    # Add an index to collection people in database test
    # which will speed up queries of age followed by name
    b <- mongo.index.create(mongo, "test.people", c("age", "name"))

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "age", 1L)
    mongo.bson.buffer.append(buf, "name", 1L)
    key <- mongo.bson.from.buffer(buf)

    # add an index using an alternate method of specifying the key fields
    b <- mongo.index.create(mongo, "test.people", key)

    # create an index using list of that enumerates the key fields
    b <- mongo.index.create(mongo, "test.cars", list(make=1L, model=1L))
}
```

---

mongo.index.drop.dups

mongo.index.create flag constant - drop duplicate keys

Description

`mongo.index.create()` flag constant - drop duplicate keys.

Usage

```
mongo.index.drop.dups
```

Value

4L

`mongo.index.sparse` *mongo.index.create flag constant - sparse*

Description

`mongo.index.create()` flag constant - sparse.

Usage

```
mongo.index.sparse
```

Value

16L

`mongo.index.unique` *mongo.index.create flag constant - unique keys*

Description

`mongo.index.create()` flag constant - unique keys (no duplicates).

Usage

```
mongo.index.unique
```

Value

1L

`mongo.insert` *Add record to a collection*

Description

Add record to a collection.

See `http://www.mongodb.org/display/DOCS/Inserting`.

Usage

```
mongo.insert(mongo, ns, b)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`ns`	(string) namespace of the collection to which to add the record.
`b`	(mongo.bson) The record to add. In addition, `b` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

Value

TRUE if the command was successfully sent to the server; otherwise, FALSE.

`mongo.get.last.err()` may be examined to verify that the insert was successful on the server if necessary.

See Also

`mongo.insert.batch`,
`mongo.update`,
`mongo.find`,
`mongo.find.one`,
`mongo.remove`,
mongo.bson,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    ns <- "test.people"

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Joe")
    mongo.bson.buffer.append(buf, "age", 22L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b)

    # do the same thing in shorthand:
    mongo.insert(mongo, ns, list(name="Joe", age=22L))
}
```

`mongo.insert.batch` *Add multiple records to a collection*

Description

Add multiple records to a collection. This function eliminates some network traffic and server overhead by sending all the records in a single message.

See `http://www.mongodb.org/display/DOCS/Inserting`.

Usage

```
mongo.insert.batch(mongo, ns, lst)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`ns`	(string) namespace of the collection to which to add the record.
`lst`	A list of (mongo.bson) records to add.

Value

TRUE if the command was successfully sent to the server; otherwise, FALSE.

`mongo.get.last.err()` may be examined to verify that the insert was successful on the server if necessary.

See Also

`mongo.insert`,
`mongo.update`,
`mongo.find`,
`mongo.find.one`,
`mongo.remove`,
mongo.bson,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    ns <- "test.people"

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Dave")
    mongo.bson.buffer.append(buf, "age", 27L)
    x <- mongo.bson.from.buffer(buf)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Fred")
    mongo.bson.buffer.append(buf, "age", 31L)
    y <- mongo.bson.from.buffer(buf)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Silvia")
    mongo.bson.buffer.append(buf, "city", 24L)
    z <- mongo.bson.from.buffer(buf)
    mongo.insert.batch(mongo, ns, list(x, y, z))
}
```

`mongo.is.connected` *Determine if a mongo object is connected to a MongoDB server*

Description

Returns TRUE if the parameter mongo object is connected to a MongoDB server; otherwise, FALSE.

Usage

```
mongo.is.connected(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

Logical TRUE if the mongo connection object is currently connected to a server; otherwise, FALSE.

See Also

`mongo.create`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.count(mongo, "test.people"))
}
```

`mongo.is.master` *Determine if a mongo connection object is connected to a master*

Description

Determine if a mongo connection object is connected to a master. Normally, this is only used with replsets to see if we are currently connected to the master of the replset. However, when connected to a singleton, this function reports TRUE also.

Usage

```
mongo.is.master(mongo)
```

Arguments

`mongo`	(mongo) a mongo connection object.

Value

(logical) TRUE if the server reports that it is a master; otherwise, FALSE.

See Also

`mongo.create`,
mongo.

Examples

```
mongo <- mongo.create(c("127.0.0.1", "192.168.0.3"), name="Accounts")
if (mongo.is.connected(mongo)) {
    print("isMaster")
    print(if (mongo.is.master(mongo)) "Yes" else "No")
}
```

`mongo.oid` *The mongo.oid class*

Description

Objects of class "mongo.oid" represent MongoDB Object IDs.

See `http://www.mongodb.org/display/DOCS/Object+IDs`

mongo.oid objects contain an externally managed pointer to the actual 12-byte object ID data. This pointer is stored in the "mongo.oid" attribute of the object.

mongo.oid objects have "mongo.oid" as their class so that `mongo.bson.buffer.append()` may detect them and append the appropriate BSON OID-typed value to a buffer.

mongo.oid values may also be present in a list and will be handled properly by `mongo.bson.buffer.append.list()` and `mongo.bson.from.list()`.

See Also

mongo.oid,
`mongo.oid.from.string`,
`as.character.mongo.oid`,
`mongo.oid.to.string`,
`mongo.oid.time`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.oid`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
oid <- mongo.oid.create()
mongo.bson.buffer.append(buf, "_id", oid)
b <- mongo.bson.from.buffer(buf)
```

`mongo.oid.create` *Create a mongo.oid object*

Description

Create a mongo.oid object for appending to a buffer with `mongo.bson.buffer.append.oid()` or `mongo.bson.buffer.append()`, or for embedding in a list such that `mongo.bson.buffer.append.list()` will properly insert a regular expression value value into a mongo.bson.buffer object.

See `http://www.mongodb.org/display/DOCS/Object+IDs`

Usage

```
mongo.oid.create()
```

Value

A mongo.oid object that is reasonably assured of being unique.

See Also

mongo.oid,
`mongo.oid.from.string`,
`as.character.mongo.oid`,
`mongo.oid.to.string`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.oid`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
oid <- mongo.oid.create()
mongo.bson.buffer.append(buf, "_id", oid)
b <- mongo.bson.from.buffer(buf)
```

`mongo.oid.from.string` *Create a mongo.oid object ftom a string*

Description

Create from a 24-character hex string a mongo.oid object representing a MongoDB Object ID.

See `http://www.mongodb.org/display/DOCS/Object+IDs`

Usage

```
mongo.oid.from.string(hexstr)
```

Arguments

`hexstr`	(string) 24 hex characters representing the OID. Note that although an error is thrown if the length is not 24, no error is thrown if the characters are not hex digits; you'll get zero bits for the invalid digits.

Value

A mongo.oid object constructed from hexstr.

See Also

mongo.oid,
`mongo.oid.create`,
`as.character.mongo.oid`,
`mongo.oid.to.string`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.oid`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
oid <- mongo.oid.from.string("ABCD1234EFAB5678CDEF9012")
mongo.bson.buffer.append(buf, "_id", oid)
b <- mongo.bson.from.buffer(buf)
```

`mongo.oid.print` *Display a mongo.oid object*

Description

Display formatted output of a mongo.oid object.

This version is an alias of `print.mongo.oid()` which allows `print()` to properly handle the mongo.oid class.

Usage

```
mongo.oid.print(x)
```

Arguments

`x`	mongo.oid The object to display.

Value

The parameter is returned unchanged.

See Also

`mongo.oid.print`,
`mongo.oid.to.string`,
mongo.bson.oid,
mongo.bson.

Examples

```
oid <- mongo.oid.create()

# all display the same thing
print.mongo.oid(oid)
mongo.oid.print(oid)
print(oid)
```

`mongo.oid.time` *Get an Object ID's time*

Description

Get the 32-bit UTC time portion of an OID (Object ID).

See `http://www.mongodb.org/display/DOCS/Object+IDs`

Usage

```
mongo.oid.time(oid)
```

Arguments

`oid`	(mongo.oid) The OID to be examined.

Value

(integer) ("POSIXct") The time portion of the given *oid*.

See Also

mongo.oid,
`mongo.oid.create`,
`as.character.mongo.oid`,
`mongo.oid.to.string`,
`mongo.oid.from.string`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.oid`,
mongo.bson.buffer,
mongo.bson.

Examples

```
oid <- mongo.oid.create()
print(mongo.oid.time(oid))
```

`mongo.oid.to.string` *Convert a mongo.oid object to a string*

Description

Convert a mongo.oid object to a string of 24 hex digits. This performs the inverse operation of `mongo.oid.from.string()`.

This function is an alias of `as.character.mongo.oid()` which you may perfer to use since the class mechanism of R allows that to be called simply by `as.character(oid)`.

See `http://www.mongodb.org/display/DOCS/Object+IDs`

Usage

```
mongo.oid.to.string(oid)
```

Arguments

`oid`	(mongo.oid) The OID to be converted.

Value

(string) A string of 24 hex digits representing the bits of *oid*.

See Also

mongo.oid,
`mongo.oid.create`,
`as.character.mongo.oid`,
`mongo.oid.from.string`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.oid`,
mongo.bson.buffer,
mongo.bson.

Examples

```
oid <- mongo.oid.create()
print(mongo.oid.to.string(oid))
print(as.character(oid))  # print same thing as above line
```

mongo.reconnect *Reconnect to a MongoDB server*

Description

Reconnect to a MongoDB server. Calls mongo.disconnect and then attempts to re-establish the connection.

Usage

```
mongo.reconnect(mongo)
```

Arguments

mongo	(mongo) a mongo connection object.

See Also

mongo.create,
mongo.disconnect,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo))
    mongo.reconnect(mongo)
```

mongo.regex *The mongo.regex class*

Description

Objects of class "mongo.regex" represent regular expressions and are strings with the options value stored in the "options" attribute.

See http://www.mongodb.org/display/DOCS/Advanced+Queries#AdvancedQueries-RegularE

mongo.regex objects have "mongo.regex" as their class so that
mongo.bson.buffer.append() may detect them and append the appropriate BSON regex-typed value to a buffer.

These mongo.regex values may also be present in a list and will be handled properly
by mongo.bson.buffer.append.list() and mongo.bson.from.list().

See Also

mongo.regex.create,
mongo.bson.buffer.append,
mongo.bson.buffer.append.list,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
regex <- mongo.regex.create("acme.*corp", options="i")
mongo.bson.buffer.append.regex(buf, "MatchAcme", regex)
b <- mongo.bson.from.buffer(buf)
```

`mongo.regex.create` *Create a mongo.regex object*

Description

Create a mongo.regex object for appending to a buffer with `mongo.bson.buffer.append.regex()` or `mongo.bson.buffer.append()`, or for embedding in a list such that `mongo.bson.buffer.append.list()` will properly insert a regular expression value value into a mongo.bson.buffer object.

See `http://www.mongodb.org/display/DOCS/Advanced+Queries#AdvancedQueries-RegularE`

Usage

```
mongo.regex.create(pattern, options="")
```

Arguments

`pattern`	(string) The regular expression.
`options`	(string) Options governing the parsing done with the pattern.

Value

A mongo.regex object

See Also

mongo.regex,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.regex`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
regex <- mongo.regex.create("acme.*corp", options="i")
mongo.bson.buffer.append.regex(buf, "MatchAcme", regex)
b <- mongo.bson.from.buffer(buf)
```

`mongo.remove` *Remove records from a collection*

Description

Remove all records from a collection that match a given criteria.

See `http://www.mongodb.org/display/DOCS/Removing`.

Usage

```
mongo.remove(mongo, ns, criteria=mongo.bson.empty())
```

Arguments

`mongo`	(mongo) a mongo connection object.
`ns`	(string) namespace of the collection from which to remove records.
`criteria`	(mongo.bson) The criteria with which to match records that are to be removed. The default of mongo.bson.empty() will cause *all* records in the given collection to be removed. Alternately, `criteria` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.

See Also

mongo,
mongo.bson,
`mongo.insert`,
`mongo.update`,
`mongo.find`,
`mongo.find.one`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Jeff")
    criteria <- mongo.bson.from.buffer(buf)

    # remove all records where name is "Jeff"
    # from collection people in database test
    mongo.remove(mongo, "test.people", criteria)

    # remove all records from collection cars in database test
    mongo.remove(mongo, "test.cars")

    # shorthand: remove all records where name is "Fred"
    mongo.remove(mongo, "test.people", list(name="Fred"))
}
```

`mongo.rename` *Rename a collection on a MongoDB server*

Description

Rename a collection on a MongoDB server.

Note that this may also be used to move a collection from one database to another.

Usage

```
mongo.rename(mongo, from.ns, to.ns)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`from.ns`	(string) The namespace of the collection to rename.
`to.ns`	(string) The new namespace of the collection.

Value

NULL if unsuccessful; otherwise,

(mongo.bson) Server response.

See Also

`mongo.drop.database`,
`mongo.drop`,
`mongo.command`,
`mongo.count`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.rename(mongo, "test.people", "test.humans"))

    mongo.destroy(mongo)
}
```

mongo.reset.err — *Retrieve an server error code from a mongo connection object*

Description

Send a "reset error" command to the server, it also resets the values returned by `mongo.get.server.err()` and `mongo.get.server.err.string()`.

Usage

```
mongo.reset.err(mongo, db)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`db`	(string) The name of the database on which to reset the error status.

Value

NULL

See Also

`mongo.get.server.err`,
`mongo.get.server.err.string`,
`mongo.get.last.err`,
`mongo.get.prev.err`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {

    # try adding a duplicate record when index doesn't allow this

    db <- "test"
    ns <- "test.people"
    mongo.index.create(mongo, ns, "name", mongo.index.unique)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "John")
    mongo.bson.buffer.append(buf, "age", 22L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b);

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "John")
    mongo.bson.buffer.append(buf, "age", 27L)
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, ns, b);

    err <- mongo.get.last.err(mongo, db)
```

```
        print(mongo.get.server.err(mongo))
        print(mongo.get.server.err.string(mongo))
        mongo.reset.err(mongo, db)
    }
```

`mongo.set.timeout` *Set the timeout value on a mongo connection*

Description

Set the timeout value for network operations on a mongo connection. Subsequent network operations will timeout if they take longer than the given number of milliseconds.

Usage

```
mongo.set.timeout(mongo, timeout)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`timeout`	(as.integer) number of milliseconds to which to set the timeout value.

See Also

`mongo.get.timeout`,
`mongo.create`,
mongo.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    mongo.set.timeout(mongo, 2000L)
    timeout <- mongo.get.timeout(mongo)
    if (timeout != 2000L)
        error("expected timeout of 2000");
}
```

`mongo.simple.command` *Issue a simple.command to a database on MongoDB server*

Description

Issue a simple command to a MongoDB server and return the response from the server.

This function supports many of the MongoDB database commands by allowing you to specify a simple command object which is entirely specified by the command name and an integer or string argument.

See `http://www.mongodb.org/display/DOCS/List+of+Database+Commands`.

Usage

```
mongo.simple.command(mongo, db, cmdstr, arg)
```

Arguments

`mongo`	(mongo) A mongo connection object.
`db`	(string) The name of the database upon which to perform the command.
`cmdstr`	(string) The name of the command.
`arg`	An argument to the command, may be a string or numeric (as.integer).

Value

NULL if the command failed. Use `mongo.get.last.err()` to determine the cause.

(mongo.bson) The server's response if successful.

See Also

`mongo.command`,
`mongo.rename`,
`mongo.count`,
`mongo.drop.database`,
`mongo.drop`,
mongo,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    print(mongo.simple.command(mongo, "admin", "buildInfo", 1))

    mongo.destroy(mongo)
}
```

`mongo.symbol` *The mongo.symbol class*

Description

Objects of class "mongo.symbol" are used to represent symbol values in BSON documents.

mongo.symbol objects' value is a string representing the value of the symbol.

mongo.symbol objects have "mongo.symbol" as their class so that `mongo.bson.buffer.append()` may detect them and append the appropriate BSON symbol-typed value to a buffer.

These mongo.symbol values may also be present in a list and will be handled properly by `mongo.bson.buffer.append.list()` and `mongo.bson.from.list()`.

See Also

mongo.symbol.create,
mongo.bson.buffer.append,
mongo.bson.buffer.append.list,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
sym <- mongo.symbol.create("Beta")
mongo.bson.buffer.append(buf, "B", sym)
l <- list(s1 = sym, Two = 2)
mongo.bson.buffer.append.list(buf, "listWsym", l)
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "B": (SYMBOL) "Beta",
#   "listWsym" : { "s1" : (SYMBOL) "Beta",
#                  "Two" : 2 } }
```

---

mongo.symbol.create

Create a mongo.symbol object

Description

Create a mongo.symbol object for appending to a buffer with `mongo.bson.buffer.append()` or for embedding in a list such that `mongo.bson.buffer.append.list()` will properly insert a symbol value into the mongo.bson.buffer object.

Usage

```
mongo.symbol.create(value)
```

Arguments

`value`	(string) The value of the symbol

Value

a mongo.symbol object

See Also

mongo.symbol,
mongo.bson.buffer.append,
mongo.bson.buffer.append.list,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
sym <- mongo.symbol.create("Alpha")
mongo.bson.buffer.append(buf, "A", sym)
lst <- list(s1 = sym, One = 1)
mongo.bson.buffer.append.list(buf, "listWsym", lst)
mongo.bson.buffer.append.symbol(buf, "D", "Delta")
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "A": (SYMBOL) "Alpha",
#   "listWsym" : { "a1"  : (SYMBOL) "Aplha",
#                  "One" : 1 },
#   "D" : (SYMBOL) "Delta" }
```

---

mongo.timestamp *The mongo.timestamp class*

Description

Objects of class "mongo.timestamp" are an extension of the POSIXct class. They have their increment value stored in the "increment" attribute of the object.

See http://www.mongodb.org/display/DOCS/Timestamp+Data+Type

mongo.timestamp objects have "mongo.timestamp", "POSIXct" & "POSIXt" as their class so that `mongo.bson.buffer.append()` may detect them and append the appropriate BSON code-typed value to a buffer.

These mongo.timestamp values may also be present in a list and will be handled properly by `mongo.bson.buffer.append.list()` and `mongo.bson.from.list()`.

See Also

`mongo.timestamp.create`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    # special Null timestamp -- automatically filled in
    # if one of first two fields in a record
    ts <- mongo.timestamp.create(0,0)
    mongo.bson.buffer.append(buf, "InsertTime", ts)
    mongo.bson.buffer.append(buf, "name", "Joe")
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, "test.people", b)

    # create using a POSIXlt
    ts <- mongo.timestamp.create(strptime("05-12-2012",
        "%m-%d-%Y"), increment=1)
}
```

`mongo.timestamp.create` *Create a mongo.timestamp object*

Description

Create a mongo.timestamp object for appending to a buffer with `mongo.bson.buffer.append.timestamp()` or `mongo.bson.buffer.append()`, or for embedding in a list such that `mongo.bson.buffer.append.list()` will properly insert a timestamp value into the mongo.bson.buffer object.

See `http://www.mongodb.org/display/DOCS/Timestamp+Data+Type`

Usage

```
mongo.timestamp.create(time, increment)
```

Arguments

`time`	(integer) date/time value (milliseconds since UTC epoch). This may also be a "POSIXct" or "POSIXlt" class object.
`increment`	increment ordinal

Value

A mongo.timestamp object

See Also

mongo.timestamp,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.time`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    buf <- mongo.bson.buffer.create()
    # special Null timestamp -- automatically filled in
    # if one of first two fields in a record
    ts <- mongo.timestamp.create(0,0)
    mongo.bson.buffer.append(buf, "InsertTime", ts)
    mongo.bson.buffer.append(buf, "name", "Joe")
    b <- mongo.bson.from.buffer(buf)
    mongo.insert(mongo, "test.people", b)

    # create using a POSIXlt
    ts <- mongo.timestamp.create(strptime("05-12-2012",
        "%m-%d-%Y"), increment=1)
}
```

---

mongo.undefined *The mongo.undefined class*

Description

Objects of class "mongo.undefined" are used to represent undefined values in BSON documents.

mongo.undefined objects are strings (a character vector) with a single value of "UNDEFINED"

mongo.undefined objects have "mongo.undefined" as their class so that `mongo.bson.buffer.append()` may detect them and append the appropriate BSON undefined value to a buffer.

These mongo.undefined values may also be present in a list and will be handled properly by `mongo.bson.buffer.append.list()` and `mongo.bson.from.list()`.

See Also

`mongo.undefined.create`,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
undef <- mongo.undefined.create()
mongo.bson.buffer.append(buf, "Undef", undef)
l <- list(u1 = undef, One = 1)
mongo.bson.buffer.append.list(buf, "listWundef", l)
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "Undef": UNDEFINED, "listWundef" : { "u1" : UNDEFINED, "One" : 1 } }
```

---

mongo.undefined.create *Create a mongo.undefined object*

Description

Create a mongo.undefined object for appending to a buffer with `mongo.bson.buffer.append()` or for embedding in a list such that `mongo.bson.buffer.append.list()` will properly insert an undefined value into the mongo.bson.buffer object.

Usage

```
mongo.undefined.create()
```

Value

a mongo.undefined object

See Also

mongo.undefined,
`mongo.bson.buffer.append`,
`mongo.bson.buffer.append.list`,
mongo.bson.buffer,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
undef <- mongo.undefined.create()
mongo.bson.buffer.append(buf, "Undef", undef)
l <- list(u1 = undef, One = 1)
mongo.bson.buffer.append.list(buf, "listWundef", l)
b <- mongo.bson.from.buffer(buf)

# the above will create a mongo.bson object of the following form:
# { "Undef": UNDEFINED, "listWundef" : { "u1" : UNDEFINED, "One" : 1 } }
```

`mongo.update` *Perform an update on a collection*

Description

Perform an update on a collection.

See `http://www.mongodb.org/display/DOCS/Updating`.

Usage

```
mongo.update(mongo, ns, criteria, objNew, flags=0L)
```

Arguments

`mongo`	(mongo) a mongo connection object.
`ns`	(string) namespace of the collection to which to update.
`criteria`	(mongo.bson) The criteria with which to match records that are to be updated. Alternately, `criteria` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.
`objNew`	(mongo.bson) The replacement object. Alternately, `objNew` may be a list which will be converted to a mongo.bson object by `mongo.bson.from.list()`.
`flags`	(integer vector) A list of optional flags governing the operation: • `mongo.update.upsert`: insert ObjNew into the database if no record matching criteria is found. • `mongo.update.multi`: update multiple records rather than just the first one matched by criteria. • `mongo.update.basic`: Perform a basic update.

See Also

mongo,
mongo.bson,
`mongo.insert`,
`mongo.find`,
`mongo.find.one`,
`mongo.remove`.

Examples

```
mongo <- mongo.create()
if (mongo.is.connected(mongo)) {
    ns <- "test.people"

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Joe")
    criteria <- mongo.bson.from.buffer(buf)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.start.object(buf, "$inc")
    mongo.bson.buffer.append(buf, "age", 1L)
    mongo.bson.buffer.finish.object(buf)
    objNew <- mongo.bson.from.buffer(buf)

    # increment the age field of the first record matching name "Joe"
    mongo.update(mongo, ns, criteria, objNew)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Jeff")
    criteria <- mongo.bson.from.buffer(buf)

    buf <- mongo.bson.buffer.create()
    mongo.bson.buffer.append(buf, "name", "Jeff")
    mongo.bson.buffer.append(buf, "age", 27L)
    objNew <- mongo.bson.from.buffer(buf)

    # update the entire record to { name: "Jeff", age: 27 }
    # where name equals "Jeff"
    # if such a record exists; otherwise, insert this as a new reord
    mongo.update(mongo, ns, criteria, objNew,
        mongo.update.upsert)

    # do a shorthand update:
    mongo.update(mongo, ns, list(name="John"), list(name="John", age=25))
}
```

`mongo.update.basic` *mongo.update() flag constant for performing a basic update*

Description

Flag to `mongo.update()` (4L): Perform a basic update.

Usage

```
mongo.update.basic
```

Value

4L

See Also

`mongo.update`,
`mongo.update.multi`
`mongo.update.upsert`

`mongo.update.multi` *mongo.update() flag constant for updating multiple records*

Description

Flag to `mongo.update()` (2L): Update multiple records rather than just the first one matched by criteria.

Usage

```
mongo.update.multi
```

Value

2L

See Also

`mongo.update`,
`mongo.update.upsert`,
`mongo.update.basic`.

`mongo.update.upsert` *mongo.update() flag constant for an upsert*

Description

Flag to `mongo.update()` (1L): insert ObjNew into the database if no record matching criteria is found.

Usage

```
mongo.update.upsert
```

Value

1L

See Also

`mongo.update`,
`mongo.update.multi`,
`mongo.update.basic`.

`print.mongo.bson` *Display a mongo.bson object*

Description

Display formatted output of a mongo.bson object.

Output is tabbed (indented to show the nesting level of subobjects and arrays).

This version is an alias of mongo.bson.print() so that print() will properly handle the mongo.bson class.

Usage

```
print.mongo.bson(x, ...)
```

Arguments

`x`	(mongo.bson The object to display.
`...`	Parameters passed from generic.

Value

The parameter is returned unchanged.

See Also

`mongo.bson.print`,
mongo.bson.

Examples

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "name", "Fred")
mongo.bson.buffer.append(buf, "city", "Dayton")
b <- mongo.bson.from.buffer(buf)

# all display the same thing
print.mongo.bson(b)
mongo.bson.print(b)
print(b)
```

`print.mongo.oid` *Display a mongo.oid object*

Description

Display formatted output of a mongo.oid object.

Output is tabbed (indented to show the nesting level of subobjects and arrays).

This version is an alias of `mongo.oid.print()` so that `print()` will properly handle the mongo.oid class.

Usage

```
print.mongo.oid(x, ...)
```

Arguments

`x`	mongo.oid The object to display.
`...`	Parameters passed from generic.

Value

The parameter is returned unchanged.

See Also

`mongo.oid.print`,
`mongo.oid.to.string`,
mongo.bson.oid,
mongo.bson.

Examples

```
oid <- mongo.oid.create()

# all display the same thing
print.mongo.oid(oid)
mongo.oid.print(oid)
print(oid)
```

Index