Package ‘RProtoBuf’

August 13, 2017

Version 0.4.10
Date 2017-08-13
Author Romain Francois, Dirk Eddelbuettel, Murray Stokely and Jeroen Ooms
Maintainer Dirk Eddelbuettel <edd@debian.org>
Title R Interface to the 'Protocol Buffers' 'API' (Version 2 or 3)
Description Protocol Buffers are a way of encoding structured data in an efficient yet extensible format. Google uses Protocol Buffers for almost all of its internal 'RPC' protocols and file formats. Additional documentation is available in two included vignettes one of which corresponds to our paper in the Journal of Statistical Software (2016, v71i02). Either version 2 or 3 of the 'Protocol Buffers' 'API' is supported.
Depends R (>= 3.0.0), methods
Imports utils, stats, tools, Rcpp, RCurl
LinkingTo Rcpp
Suggests RUnit, highlight
VignetteBuilder highlight
SystemRequirements ProtoBuf libraries and compiler version 2.2.0 or later; version 3.0.0 or later is supported as well. On Debian/Ubuntu these can be installed as libprotoc-dev and libprotobuf-dev, while on Fedora/CentOS protobuf-devel and protobuf-compiler are needed.
BugReports https://github.com/eddelbuettel/rprotobuf/issues
URL https://github.com/eddelbuettel/rprotobuf
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-08-13 20:55:40 UTC
### R topics documented:

- `RProtoBuf-package` .............................................. 3
- `add-methods` .................................................. 4
- `ArrayInputStream-class` ....................................... 4
- `ArrayInputStream-methods` ..................................... 6
- `ArrayOutputStream-class` ...................................... 6
- `ArrayOutputStream-methods` .................................. 7
- `as.list.Message` ............................................... 7
- `asMessage` .................................................... 9
- `BackUp-methods` ............................................... 10
- `ByteCount-methods` .......................................... 10
- `bytesize-methods` ............................................ 10
- `clear-methods` ................................................ 11
- `clone-methods` ................................................ 11
- `completion` ................................................... 12
- `ConnectionInputStream-class` ............................... 13
- `ConnectionInputStream-methods` ............................ 14
- `ConnectionOutputStream-class` .............................. 15
- `ConnectionOutputStream-methods` .......................... 15
- `containing_type-methods` ................................... 16
- `Descriptor-class` ............................................. 16
- `descriptor-methods` .......................................... 18
- `EnumDescriptor-class` ....................................... 18
- `EnumValueDescriptor-class` .................................. 20
- `enum_type-methods` .......................................... 21
- `enum_type_count-methods` ................................... 22
- `fetch-methods` ............................................... 22
- `field-methods` ............................................... 22
- `FieldDescriptor-class` ...................................... 23
- `field_count-methods` ....................................... 25
- `FileDescriptor-class` ...................................... 26
- `fileDescriptor-methods` ................................... 27
- `FileInputStream-class` ....................................... 27
- `FileInputStream-methods` ................................... 28
- `FileOutputStream-class` .................................... 29
- `FileOutputStream-methods` .................................. 30
- `GetErrno-methods` ........................................... 30
- `has-methods` .................................................. 30
- `invoke-methods` ............................................. 31
- `isInitialized-methods` ..................................... 31
- `is_extension-methods` ...................................... 32
- `label-methods` ............................................... 32
- `merge-methods` ................................................ 33
- `Message-class` ................................................. 34
- `MethodDescriptor-class` .................................... 36
- `name` .......................................................... 37
- `nested_type-methods` ....................................... 37
Description

Protocol Buffers are a way of encoding structured data in an efficient yet extensible format. Google uses Protocol Buffers for almost all of its internal RPC protocols and file formats.

This package provides R API to create, manipulate, parse and serialize protocol buffer messages from R

Author(s)

Romain Francois, Dirk Eddelbuettel, Murray Stokely and Jeroen Ooms.

References

https://github.com/eddelbuettel/rprotobuf

See Also

Message for some examples
Examples

```r
## Not run:
# an example proto file
system.file( "proto", "addressbook.proto", package = "RProtoBuf" )

# create a message of type AddressBook, defined in the example proto file
demo( "addressbook", package = "RProtoBuf" )

# using R binary connections and files to read and write messages
demo( "io", package = "RProtoBuf" )

# more documentation in the vignette
vignette( "RProtoBuf", package = "RProtoBuf" )

## End(Not run)
```

add-methods

```r
add elements of a repeated field of a message
```

Description

Add elements to a repeated field of a message.

Methods

signature(object = "Message") add elements to a repeated field of a message

Examples

```r
unitest.proto.file <- system.file("unitTests", "data", "unitest.proto",
package = "RProtoBuf" )
readProtoFiles(file = unitest.proto.file)

test <- new(probuf_unittest.TestAllTypes)
test$add("repeated_int32", 1)
test$add("repeated_int32", 2:10)
test$repeated_int32
```

ArrayInputStream-class

Class "ArrayInputStream"

Description

A ZeroCopyInputStream backed by an in-memory array of bytes
ArrayInputStream-class

Objects from the Class

Objects can be created by the ArrayInputStream function

Slots

pointer: External pointer to the google::protobuf::io::ArrayInputStream C++ object

Extends

Class "ZeroCopyInputStream", directly.

Methods

See ZeroCopyInputStream

Author(s)

Romain Francois <francisromain@free.fr>

References


See Also

ZeroCopyInputStream for methods

Examples

stream <- ArrayInputStream(as.raw(0:10))
stream$ReadRaw(5)

stringsstream <- ArrayInputStream(as.raw(c(0x74, 0x65, 0x73, 0x74, 0x69, 0x6e, 0x67)))
stringsstream$ReadString(7)

intstream <- ArrayInputStream(as.raw(c(0x9e, 0xa7, 0x05)))
intstream$ReadVarint32()
ArrayInputStream-methods

Creates an ArrayInputStream

Description

Constructor for ArrayInputStream objects

Methods

signature(payload = "raw", block_size = "missing") Creates a ArrayInputStream using the raw vector as the payload of the stream
signature(payload = "raw", block_size = "integer") Creates a ArrayInputStream ... same with block size.
signature(payload = "raw", block_size = "numeric") Creates a ArrayInputStream ... same with block size.

ArrayOutputStream-class

Class "ArrayOutputStream"

Description

A ZeroCopyOutputStream backed by an in-memory array of bytes

Objects from the Class

Objects can be created by the ArrayOutputStream function

Slots

pointer: External pointer to the google::protobuf::io::ArrayOutputStream C++ object

Extends

Class "ZeroCopyOutputStream", directly.

Methods

See ZeroCopyOutputStream

Author(s)

Romain Francois <francoisromain@free.fr>
References


See Also

ZeroCopyOutputStream for methods

---

ArrayOutputStream-methods

Creates an ArrayOutputStream

---

Description

Constructor for ArrayOutputStream objects

Methods

signature(size = "integer", block_size = "missing") Creates a ArrayOutputStream using of the given size
signature(size = "integer", block_size = "integer") Creates a ArrayOutputStream same with block size.
signature(size = "integer", block_size = "numeric") Creates a ArrayOutputStream same with block size.
signature(size = "numeric", block_size = "missing") Creates a ArrayOutputStream using of the given size
signature(size = "numeric", block_size = "integer") Creates a ArrayOutputStream same with block size.
signature(size = "numeric", block_size = "numeric") Creates a ArrayOutputStream same with block size.

---

as.list.Message

Grab the protocol buffer message as an R list

Description

Utility to grab the protocol buffer message as an R list, with one item per field.
Usage

```r
## S3 method for class 'Message'
as.list(x, ...)
## S3 method for class 'Descriptor'
as.list(x, ...)
## S3 method for class 'EnumDescriptor'
as.list(x, ...)
## S3 method for class 'FileDescriptor'
as.list(x, ...)
## S3 method for class 'ServiceDescriptor'
as.list(x, ...)
```

Arguments

- `x` A protocol buffer message, instance of `Message`, or a protocol message descriptor, instance of `Descriptor`
- `...` ignored

Value

For messages, a list of the content of the fields is returned.

For message type descriptors, a list containing nested type descriptors (`Descriptor` objects), enum type descriptors (`EnumDescriptor` objects), then field descriptors (`FieldDescriptor` objects) in that order.

For enum descriptors, a named list of the enumerated values.

For file descriptors, a named list of descriptors defined in the specified file descriptor.

For service descriptors, ...

Author(s)

Romain Francois <francoisromain@free.fr>

Examples

```r
Person <- P( "tutorial.Person" )
romain <- new( Person, email = "francoisromain@free.fr", id = 1 )
as.list( romain )
as.list( Person )
as.list( Person$PhoneNumber )
```
asMessage

asMessage (x, ...)

Description

c coerce an object to the Message class. This is a short-hand to the as method with the Class argument set to "Message"

Usage

asMessage(x, ...)

Arguments

x object to coerce to a protobuf message
...
Passed to as

Value

a Message object

Author(s)

Romain Francois <francoisromain@free.fr>

Examples

# coerce a message type descriptor to a message
asMessage( tutorial.Person )

# coerce a enum descriptor
asMessage( tutorial.Person.PhoneType )

# coerce a field descriptor
asMessage( tutorial.Person.email )

# coerce a file descriptor
asMessage( fileDescriptor( tutorial.Person ) )
**BackUp-methods**  
*Backs up a number of bytes from a stream*

**Description**

Backs up a number of bytes from a stream

**See Also**

*ZeroCopyInputStream* implements BackUp.

---

**ByteCount-methods**  
*The number of bytes read/written since the object was created*

**Description**

The number of bytes read/written since the object was created

**See Also**

*ZeroCopyInputStream* implements ByteCount.

---

**bytesize-methods**  
*The number of bytes taken by a message*

**Description**

The number of bytes taken by a *Message*

**Methods**

signature(object = "Message")  
The number of bytes the message would take when serialized

**Examples**

```r
message <- new( tutorial.Person, name = "dddd", email = "eeeeee", id = 1 )
bytesize( message )
```

clear-methods

Clear a field or all fields of the message and set them to their default values

Description

Clear one field or all fields of the message and set them to their default values

Methods

signature(object = "Message", field = "missing") Clear all fields of the message and set them to their default values

signature(object = "Message", field = "character") Clear the field identified by its name

signature(object = "Message", field = "integer") Clear the field identified by its tag number

signature(object = "Message", field = "numeric") Clear the field identified by its tag number

signature(object = "Message", field = "raw") Clear the field identified by its tag number

Examples

message <- new( tutorial.Person, name = "dddd", email = "eeeeee", id = 1 )
writelines( as.character( message ) )
clear( message )
  # clear works also as a pseudo method :
message$clear()
writelines( as.character( message ) )

  # clear single fields
message <- new( tutorial.Person, name = "dddd", email = "eeeeee", id = 1 )
message$clear( "name" )
writelines( as.character( message ) )

clone-methods

Clone protocol buffer messages

Description

Generic "clone" function and associated method for Message objects

Methods

signature(object = "Message") clone the message
Examples

## Not run:

```r
# example proto file supplied with this package
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )

# reading a proto file and creating the descriptor
Person <- P( "tutorial.Person", file = proto.file )
```

## End(Not run)

# creating a prototype message from the descriptor
sheep <- new( Person, email = "francoisromain@free.fr", id = 2 )

# cloning the sheep
newsheep <- clone( sheep )

# clone and update at once
newsheep <- clone( sheep, id = 3 )

# this can also be used as a pseudo method
sheep$clone()

sheep$clone( id = 3 )

---

**Completion support for protocol buffer messages and descriptors**

**Description**

These functions support completion of protocol buffer messages and descriptors.

**Usage**

```
## S3 method for class 'Message'
.DollarNames(x, pattern = "")

## S3 method for class 'Descriptor'
.DollarNames(x, pattern = "")

## S3 method for class 'EnumDescriptor'
.DollarNames(x, pattern = "")

## S3 method for class 'FieldDescriptor'
.DollarNames(x, pattern = "")

## S3 method for class 'FileDescriptor'
.DollarNames(x, pattern = "")

## S3 method for class 'ServiceDescriptor'
.DollarNames(x, pattern = "")

## S3 method for class 'MethodDescriptor'
.DollarNames(x, pattern = "")

## S3 method for class 'ZeroCopyInputStream'
```
ConnectionInputStream-class

Arguments

x message (Message) or descriptor (Descriptor)

pattern filter

Value

Character vector containing potential completions.

For Message objects, completions are the fields of the message and a set of pseudo methods ("has")

For EnumDescriptor objects, completions are the names of the possible constants

For Descriptor objects, completions are the names of the fields, enum types and nested message types defined in the associated message type.

For FileDescriptor objects, completions are the names of the top-level descriptors (message, enum or service) contained in the associated file, or pseudo methods.

Author(s)

Romain Francois <francoisromain@free.fr>

Examples

# creating a prototype message from the descriptor
p <- new( tutorial.Person )

.DollarNames( p )
.DollarNames( tutorial.Person )
# but this is usually used with the <TAB> expansion on the command line
# <TAB> means “press the TAB key”
# p$<TAB>
# Person$<TAB>

Description

A ZeroCopyInputStream reading from a binary R connection

Objects from the Class

Objects can be created by the ConnectionInputStream function
Slots

pointer: External pointer to the rprotobuf::ConnectionInputStream C++ object

Extends

Class "ZeroCopyInputStream", directly.

Methods

See ZeroCopyInputStream

Author(s)

Romain Francois <francoisromain@free.fr>

References

The internal C++ class ConnectionInputStream

See Also

ZeroCopyInputStream for methods

Description

Constructor for ConnectionInputStream objects

Methods

signature(object="connection") Creates a ConnectionInputStream reading from the given R binary connection.
ConnectionOutputStream-class

Description

A **ZeroCopyOutputStream** writing to a binary R connection

Objects from the Class

Objects can be created by the **ConnectionOutputStream** function

Slots

pointer: External pointer to the `rprotobuf::ConnectionOutputStream` C++ object

Extends

Class "**ZeroCopyOutputStream**", directly.

Methods

See **ZeroCopyOutputStream**

Author(s)

Romain François <francoisromain@free.fr>

References

The internal C++ class **ConnectionOutputStream**

See Also

**ZeroCopyOutputStream** for methods

ConnectionOutputStream-methods

*Creates an ConnectionOutputStream*

Description

Constructor for **ConnectionOutputStream** objects

Methods

`signature(object="connection")` Creates a **ConnectionOutputStream** writing to the given R binary connection.
containing_type-methods

*Gets the message type descriptor that contains a descriptor*

**Description**

Gets a **Descriptor** describing the message type that contains the descriptor.

**See Also**

The method is implemented for these classes: **Descriptor**, **EnumDescriptor**, **FieldDescriptor**

**Examples**

```r
# Containing type of a field is the message descriptor
tutorial.Person$id$containing_type()

# No containing type for the top-level message descriptor.
tutorial.Person$containing_type()
```

---

**Descriptor-class**

*Class "Descriptor"*

**Description**

full descriptive information about a protocol buffer message type. This is a thin wrapper around the C++ class **Descriptor**

**Objects from the Class**

Objects are usually created by calls to the **P** function.

**Slots**

pointer: external pointer holding a **Descriptor** object
type: full name of the corresponding message type

**Methods**

- **as.character** signature(x = "Descriptor"): returns the debug string of the descriptor. This is retrieved by a call to the **DebugString** method of the **Descriptor** object.
- **toString** signature(x = "Descriptor"): same as **as.character**
- **$** signature(x = "Descriptor"): retrieves a descriptor for a member of the message type. This can either be another "Descriptor" instance describing a nested type, or a **EnumDescriptor** object describing an enum type, or a **FieldDescriptor** object describing a field of the message
new signature(Class = "Descriptor"): creates a prototype message (Message) of this descriptor

show signature(object = "Descriptor"): simple information

containing_type signature(object = "Descriptor"): returns a descriptor of the message type that contains this message descriptor, or NULL if this is a top-level message type.

field_count signature(object = "Descriptor"): The number of fields of this message type.

nested_type_count signature(object = "Descriptor"): The number of nested types of this message type.

enum_type_count signature(object = "Descriptor"): The number of enum types of this message type.

field signature(object = "Descriptor"): extract a field descriptor from a descriptor. Exactly one argument of index, number or name has to be used. If index is used, the field descriptor is retrieved by position, using the field method of the google::protobuf::Descriptor C++ class. If number is used, the field descriptor is retrieved using the tag number, with the FindFieldByNumber C++ method. If name is used, the field descriptor is retrieved by name using the FindFieldByName

nested_type signature(object = "Descriptor"): extracts a message type descriptor that is nested in this descriptor. Exactly one argument of index of name has to be used. If index is used, the nested type will be retrieved using its position with the nested_type method of the google::protobuf::Descriptor C++ class. If name is used, the nested type will be retrieved using its name, with the FindNestedTypeByName C++ method

enum_type signature(object = "Descriptor"): extracts an enum type descriptor that is contained in this descriptor. Exactly one argument of index of name has to be used. If index is used, the enum type will be retrieved using its position with the enum_type method of the google::protobuf::Descriptor C++ class. If name is used, the enum type will be retrieved using its name, with the FindEnumTypeByName C++ method

[[ signature(x = "Descriptor"): extracts a field identified by its name or declared tag number

names signature(x = "Descriptor"): extracts names of this descriptor

length signature(x = "Descriptor"): extracts length of this descriptor

Author(s)

Romain Francois <francoisromain@free.fr>

References


See Also

the P function creates “Descriptor” messages.
Examples

```r
## Not run:
# example proto file supplied with this package
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )
# reading a proto file and creating the descriptor
Person <- P("tutorial.Person", file = proto.file )

## End(Not run)
```

# enum type
Person$PhoneType

# nested type
Person$PhoneNumber

# field
Person$email

# use this descriptor to create a message
new( Person )

---

**descriptor-methods**  *Get the descriptor of a message*

---

**Description**

Get the **Descriptor** associated with a **Message**

**Methods**

```r
descriptor(object = "Message") Get the descriptor of the message, as a **Descriptor** instance
```

---

**EnumDescriptor-class**  *Class "EnumDescriptor"*

---

**Description**

R representation of an enum descriptor. This is a thin wrapper around the EnumDescriptor c++ class.

**Objects from the Class**

Objects of this class are typically retrieved as members of **Descriptor** objects
**EnumDescriptor-class**

**Slots**
- **pointer**: external pointer to the EnumDescriptor instance
- **name**: simple name of the enum
- **full_name**: fully qualified name
- **type**: fully qualified name of the type that contains this enumeration

**Methods**

- **show** signature(object = "EnumDescriptor"): small information
- **as.character** signature(x = "EnumDescriptor"): returns the debug string of the enum descriptor. This is retrieved by a call to the DebugString method of the EnumDescriptor object.
- **toString** signature(x = "EnumDescriptor"): same as as.character
- **$** signature(x = "EnumDescriptor"): get the number associated with the name
- **has** signature(object = "EnumDescriptor"): indicate if the given name is a constant present in this enum.
- **containing_type** signature(object = "EnumDescriptor"): returns a Descriptor of the message type that contains this enum descriptor, or NULL if this is a top level enum descriptor.
- **length** signature(x = "EnumDescriptor"): number of constants in this enum.
- **value_count** signature(object = "EnumDescriptor"): number of constants in this enum.
- **value** signature(object = "EnumDescriptor"): extracts an EnumValueDescriptor. Exactly one argument of index, number or name has to be used. If index is used, the enum value descriptor is retrieved by position, using the value method of the C++ class. If number is used, the enum value descriptor is retrieved using the value of the constant, using the FindValueByNumber C++ method. If name is used, the enum value descriptor is retrieved using the name of the constant, using the FindValueByName C++ method.
- **[[** signature(x = "EnumDescriptor"): extracts field identified by its name or declared tag number
- **names** signature(x = "EnumDescriptor"): extracts names of this enum

**Author(s)**

Romain Francois <francoisromain@free.fr>

**References**

The EnumDescriptor C++ class

**See Also**

The **Descriptor** class
Examples

```r
## Not run:
# example proto file supplied with this package
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )

# reading a proto file and creating the descriptor
Person <- P("tutorial.Person", file = proto.file )

## End(Not run)

# enum type
Person$PhoneType

has(Person$PhoneType, "MOBILE")
has(Person$PhoneType, "HOME")
has(Person$PhoneType, "WORK")
has(Person$PhoneType, "FOOBAR")

length(Person$PhoneType)
```

---

**EnumValueDescriptor-class**

*Class "EnumValueDescriptor"*

---

Description

R representation of an enum value descriptor. This is a thin wrapper around the `EnumValueDescriptor` c++ class.

Objects from the Class

Objects of this class are typically retrieved with the `value` method of the `EnumDescriptor` class.

Slots

- **pointer**: external pointer to the `EnumValueDescriptor` instance
- **name**: simple name of the enum
- **full_name**: fully qualified name

Methods

- **show** signature(object = "EnumValueDescriptor"): small information
- **as.character** signature(x = "EnumValueDescriptor"): returns the debug string of the enum descriptor. This is retrieved by a call to the `DebugString` method of the `EnumDescriptor` object.
- **toString** signature(x = "EnumValueDescriptor"): same as `as.character`
$\text{signature}(x = "\text{EnumValueDescriptor"}"): \text{invoke pseudo methods}

\text{name signature(object = "\text{EnumValueDescriptor"}, full = "\text{logical"}"): return the name of this enum constant.}

\text{number signature(object = "\text{EnumValueDescriptor"}): return the numeric value of this enum constant.}

\text{enum_type signature(object = "\text{EnumDescriptor"}): retrieves the \text{EnumDescriptor} related to this value descriptor.}

\textbf{Author(s)}

Romain Francois <francoisromain@free.fr>

\textbf{References}

The \text{EnumValueDescriptor} C++ class. \url{http://code.google.com/apis/protocolbuffers/docs/reference/cpp/google.protobuf.descriptor.html#EnumValueDescriptor}

\textbf{Examples}

\begin{verbatim}
## Not run:
# example proto file supplied with this package
proto.file <- system.file("proto", "addressbook.proto", package = "RProtoBuf")
# reading a proto file and creating the descriptor
Person <- P("tutorial.Person", file = proto.file)

## End(Not run)

# enum type
Person$PhoneType

# enum value type
value(Person$PhoneType, 1)

name(value(Person$PhoneType, 1))
name(value(Person$PhoneType, 1), TRUE)

number(value(Person$PhoneType, number=1))

enum_type(value(Person$PhoneType, number=1))
\end{verbatim}

---

\textit{enum_type\text{-methods}} \quad \textit{Extract an enum type descriptor for a nested type}

\textbf{Description}

Extract a \text{EnumDescriptor} contained in a \text{Descriptor}

\textbf{See Also}

The method is implemented for the \text{Descriptor} class
enum_type_count

*The number of enum types*

Description

The number of enum types

See Also

The method is implemented for the Descriptor class

fetch

*Fetch content of a repeated field*

Description

Fetch content of a repeated field of a message

Methods

signature(object = "Message") Fetch content of a message repeated field

field

*Extract a field descriptor*

Description

Extract a FieldDescriptor from a Descriptor

See Also

The method is implemented for the Descriptor class
FieldDescriptor-class  Class "FieldDescriptor"

Description

R representation of message type field descriptor. This is a thin wrapper around the C++ class FieldDescriptor

Objects from the Class

Objects typically are retrieved from FieldDescriptor

Slots

pointer: external pointer to the FieldDescriptor C++ object
name: name of the field within the message type
full_name: Fully qualified name of the field
type: Fully qualified name of the type that contains this field

Methods

show signature(object = "FieldDescriptor"): small description
as.character signature(x = "FieldDescriptor"): returns the debug string of the field descriptor. This is retrieved by a call to the DebugString method of the FieldDescriptor object.
toString signature(x = "FieldDescriptor"): same as as.character
$ signature(x = "FieldDescriptor"): used to invoke pseudo methods
containing_type signature(object = "FieldDescriptor"): returns a Descriptor of the message type that contains this field descriptor.
is_extension signature(object = "FieldDescriptor"): indicates if this is an extension.
number signature(object = "FieldDescriptor"): gets the declared tag number of this field.
type signature(object = "FieldDescriptor"): type of this field.
cpp_type signature(object = "FieldDescriptor"): C++ type of this field.
label signature(object = "FieldDescriptor"): label of this field.
is_required signature(object = "FieldDescriptor"): is this field required.
is_optional signature(object = "FieldDescriptor"): is this field optional.
is_repeated signature(object = "FieldDescriptor"): is this field repeated.
has_default_value signature(object = "FieldDescriptor"): indicates if this field has a default value.
default_value signature(object = "FieldDescriptor"): the default value of this field.
message_type signature(object = "FieldDescriptor"): the Descriptor for the associated message type. Generates an error if this field is not a message type field.
enum_type signature(object = "FieldDescriptor"): the EnumDescriptor for the associated enum type. Generates an error if this field is not an enum type field.
Author(s)
Romain Francois <francoisromain@free.fr>

References
The FieldDescriptor C++ class

See Also
Descriptor

Examples
## Not run:
# example proto file supplied with this package
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )

# reading a proto file and creating the descriptor
Person <- P( "tutorial.Person", file = proto.file )

## End(Not run)

# field descriptor object
Person$id

# debug string
as.character( Person$email )

# or as a pseudo method
Person$email$as.character()

Person$email$is_required()
Person$email$is_optional()
Person$email$is_repeated()

Person$email$has_default_value()
Person$email$default_value()

Person$email$extension()

# Get the default values
has_default_value(Person$id)
has_default_value(Person$email)
has_default_value(Person$phone)
default_value(Person$id)
default_value(Person$email)
default_value(Person$phone)

# Get the types of field descriptors
type(Person$id)
type(Person$id, as.string=TRUE)
cpp_type(Person$email)
cpp_type(Person$email, TRUE)

# Get the label of a field descriptor
label(Person$id)
label(Person$email)
label(Person$phone)
label(Person$id, TRUE)
label(Person$email, TRUE)
label(Person$phone, TRUE)
LABEL_OPTIONAL
LABEL_REQUIRED
LABEL_REPEATED

# Test if a field is optional
is_optional(Person$id)
is_optional(Person$email)
is_optional(Person$phone)

# Test if a field is repeated
is_repeated(Person$id)
is_repeated(Person$email)
is_repeated(Person$phone)

# Test if a field is required
is_required(Person$id)
is_required(Person$email)
is_required(Person$phone)

# Return the class of a message field
message_type(Person$phone)

---

**field_count-methods**

*The number of fields*

**Description**

The number of fields

**See Also**

The method is implemented for the Descriptor class
FileDescriptor-class  

Class "FileDescriptor"

Description

Class "FileDescriptor"

Objects from the Class

Objects are usually created using the fileDescriptor method

Slots

- pointer: external pointer to a google::protobuf::FileDescriptor C++ object
- package: the package name defined in the file, e.g. 'tutorial'.
- filename: the filename of this FileDescriptor

Methods

- $ signature(x = "FileDescriptor"): used to invoke a pseudo method of the file descriptor or get a top level message, enum or service descriptor
- toString signature(x = "FileDescriptor"): gets the debug string
- as.character signature(x = "FileDescriptor"): gets the debug string
- show signature(x = "FileDescriptor"): prints small text
- name signature(object = "FileDescriptor"): name of the file

Author(s)

Romain Francois <francoisromain@free.fr>

References


See Also

Descriptor
Examples

```r
# example proto file supplied with this package
desc <- P("tutorial.Person")
person <- new(desc)

person$fileDescriptor()
nname(person$fileDescriptor())
# [1] "addressbook.proto"
as.character(person$fileDescriptor())
```

fileDescriptor-methods

gets the file descriptor of an object

Description

Gets the file descriptor of an object

Methods

- `signature(object = "Descriptor")` retrieves the file descriptor associated with this descriptor
- `signature(object = "Message")` retrieves the file descriptor associated with the descriptor of this message
- `signature(object = "EnumDescriptor")` retrieves the file descriptor associated with the enum descriptor
- `signature(object = "FieldDescriptor")` retrieves the file descriptor associated with the field descriptor
- `signature(object = "ServiceDescriptor")` retrieves the file descriptor associated with the service descriptor
- `signature(object = "MethodDescriptor")` retrieves the file descriptor associated with the method descriptor

FileInputStream-class

Class "FileInputStream"

Description

A `ZeroCopyInputStream` reading from a file

Objects from the Class

Objects can be created by the `FileInputStream` function
Slots
  pointer: External pointer to the google::protobuf::io::FileInputStream C++ object

Extends
  Class "ZeroCopyInputStream", directly.

Methods
  close signature(con="FileInputStream"): Flushes any buffers and closes the underlying file. Returns false if an error occurs during the process; use GetErrno to examine the error
  GetErrno signature(object="FileInputStream"): If an I/O error has occurred on this file descriptor, this is the errno from that error. Otherwise, this is zero. Once an error occurs, the stream is broken and all subsequent operations will fail.
  SetCloseOnDelete signature(object="FileInputStream"): set the close on delete behavior.
  See ZeroCopyInputStream for inherited methods

Author(s)
  Romain Francois <francoisromain@free.fr>

References

See Also
  ZeroCopyInputStream for methods

FileInputStream-methods
  Creates an FileInputStream

Description
  Constructor for FileInputStream objects

Methods
  signature(filename = "character", block_size = "logical", close.on.delete = "logical" )
  Creates a FileInputStream reading from the given file.
FileOutputStream-class

Class "FileOutputStream"

Description

A ZeroCopyOutputStream reading from a file

Objects from the Class

Objects can be created by the FileOutputStream function

Slots

pointer: External pointer to the google::protobuf::io::FileOutputStream C++ object

Extends

Class "ZeroCopyOutputStream", directly.

Methods

close signature(con="FileOutputStream"): Flushes any buffers and closes the underlying file. Returns false if an error occurs during the process; use GetErrno to examine the error

flush signature(con="FileOutputStream"): Flushes FileOutputStream’s buffers but does not close the underlying file

GetErrno signature(object="FileInputStream"): If an I/O error has occurred on this file descriptor, this is the errno from that error. Otherwise, this is zero. Once an error occurs, the stream is broken and all subsequent operations will fail.

SetCloseOnDelete signature(object="FileOutputStream"): set the close on delete behavior.
See ZeroCopyOutputStream for inherited methods

Author(s)

Romain Francois <francoisromain@free.fr>

References


See Also

ZeroCopyOutputStream for methods
FileStream-methods

*Creates an FileOutputStream*

**Description**

Constructor for FileOutputStream objects

**Methods**

signature(filename = "character", block_size = "logical", close.on.delete = "logical")

Creates a FileOutputStream writing to the given file.

GetErrno-methods

*Get the error number for an I/O error*

**Description**

If an I/O error has occurred on this file descriptor, this is the errno from that error

**Methods**

See classes FileInputStream and FileOutputStream for implementations.

Has-methods

*Indicates if an object has the given field set*

**Description**

This generic method, currently implemented for Message and EnumDescriptor indicates if the message or enum descriptor has the given field set.

For messages and non-repeated fields, a call to the HasField method of the corresponding Message is issued.

For messages and repeated fields, a call to the FieldSize method is issued, and the message is declared to have the field if the size is greater than 0.

NULL is returned if the descriptor for the message does not contain the given field at all.

For EnumDescriptors, a boolean value indicates if the given name is present in the enum definition.

**Methods**

**has** signature(object = "Message"): Indicates if the message has a given field.

**has** signature(object = "EnumDescriptor"): Indicates if the EnumDescriptor has a given named element.
Example

```java
unittest.proto.file <- system.file("unitTests", "data", "unittest.proto",
    package = "RProtoBuf")
readProtoFiles(file = unittest.proto.file)

test <- new(protobuf_unittest.TestAllTypes)
test$has("optional_int32")
# FALSE
test$add("repeated_int32", 1:10)
test$has("repeated_int32")
# TRUE
test$has("nonexistant")
# NULL
has(protobuf_unittest.TestAllTypes$NestedEnum, "FOO")
has(protobuf_unittest.TestAllTypes$NestedEnum, "BAR")
has(protobuf_unittest.TestAllTypes$NestedEnum, "XXX")
```

---

**invoke-methods**

* invoke a protobuf rpc method

**Description**

invoke a protobuf rpc method

**Methods**

signature(method = "MethodDescriptor", message = "Message") invoke a protobuf rpc method locally.
signature(method = "MethodDescriptor", message = "Message", protocol = "RpcHTTP")
    invoke a protobuf rpc method over http.

---

**isInitialized-methods**  *Indicates if a protocol buffer message is initialized*

**Description**

Indicates if a Message is initialized. A message is initialized if all its required fields are set.

**Methods**

signature(object = "Message") is the message initialized
Examples

message <- new( tutorial.Person, name = "" )
isInitialized( message ) # FALSE (id is not set)
message$initialized() # FALSE

message <- new( tutorial.Person, name = "", id = 2 )
isInitialized( message ) # TRUE
message$initialized() # TRUE

is_extension-methods Indicates if a field descriptor is an extension

Description

Indicates if a field descriptor is an extension

See Also

The method is implemented for the FieldDescriptor class

Examples

Person <- P( "tutorial.Person" )
is_extension(Person$id)

label-methods Gets the label of a field

Description

Gets the label of a field (optional, required, or repeated).

Arguments

object A FieldDescriptor object.
as.string If true, print a string representation of the type.

See Also

The method is implemented for the FieldDescriptor class
merge-methods

Examples

```r
## Not run:
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )
Person <- P( "tutorial.Person", file = proto.file )

## End(Not run)

label(Person$id)
label(Person$email)
label(Person$phone)
label(Person$id, TRUE)
label(Person$email, TRUE)
label(Person$phone, TRUE)
LABEL_OPTIONAL
LABEL_REQUIRED
LABEL_REPEATED
```

---

merge-methods

*Merge two messages of the same type*

Description

Merge two `Message` objects of the same type.

Methods

```r
signature(x = "Message", y = "Message") merge two messages of the same type
```

Errors

An error of class "IncompatibleType" is thrown if the two messages are not of the same message type.

Examples

```r
m1 <- new( tutorial.Person, email = "francoisromain@free.fr" )
m2 <- new( tutorial.Person, id = 5 )
m3 <- merge( m1, m2 )
writeLines( as.character( m1 ) )
writeLines( as.character( m2 ) )
writeLines( as.character( m3 ) )
```
**Message-class**  

Class "Message"

**Description**

R representation of protocol buffer messages. This is a thin wrapper around the Message c++ class that holds the actual message as an external pointer.

**Objects from the Class**

Objects are typically created by the `new` function invoked on a `Descriptor` object.

**Slots**

- **pointer**: external pointer to the c++ Message object
- **type**: fully qualified name of the message type

**Methods**

- **as.character** signature(x = "Message"): returns the debug string of the message. This is built from a call to the DebugString message of the Message object
- **toString** signature(x = "Message"): same as as.character
- **<-** signature(x = "Message"): set the value of a field of the message.
- **$** signature(x = "Message"): gets the value of a field. Primitive types are brought back to R as R objects of the closest matching R type. Messages are brought back as instances of the Message class.
- **[[** signature(x = "Message"): extracts a field identified by its name or declared tag number
- **[[<-** signature(x = "Message"): replace the value of a field identified by its name or declared tag number
- **serialize** signature(object = "Message"): serialize a message. If the "connection" argument is NULL, the payload of the message is returned as a raw vector, if the "connection" argument is a binary writable connection, the payload is written into the connection. If "connection" is a character vector, the message is sent to the file (in binary format).
- **show** signature(object = "Message"): displays a short text about the message
- **update** signature(object = "Message"): set several fields of the message at once
- **length** signature(x = "Message"): The number of fields actually contained in the message. A field counts in these two situations: the field is repeated and the field size is greater than 0, the field is not repeated and the message has the field.
- **setExtension** signature(object = "Message"): set an extension field of the Message.
- **getExtension** signature(object = "Message"): get the value of an extension field of the Message.
- **str** signature(object = "Message"): displays the structure of the message
identical signature(x = "Message", y = "Message"): Test if two messages are exactly identical

== signature(e1 = "Message", e2 = "Message"): Same as identical

!= signature(e1 = "Message", e2 = "Message"): Negation of identical

all.equal signature(e1 = "Message", e2 = "Message"): Test near equality
	names signature(x = "Message"): extracts the names of the message.

Author(s)

Romain Francois <francoisromain@free.fr>

References


See Also

p creates objects of class Descriptor that can be used to create messages.

Examples

```r
## Not run:
# example proto file supplied with this package
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )

# reading a proto file and creating the descriptor
Person <- P( "tutorial.Person", file = proto.file )

## End(Not run)

PhoneNumber <- P( "tutorial.Person.PhoneNumber" )

# creating a prototype message from the descriptor
p <- new( Person )
p$email # not set, returns default value
p$id  # not set, returns default value
as.character( p ) # empty
has( p, "email" ) # is the "email" field set
has( p, "phone" ) # is the "email" field set
length( p )     # number of fields actually set

# update several fields at once
romain <- update( new( Person ),
      email = "francoisromain@free.fr",
      id = 1,
      name = "Romain Francois",
      phone = new( PhoneNumber , number = "+33(0)...", type = "MOBILE" ) )
```

# supply parameters to the constructor
dirk <- new( Person, 
  email = "edd@debian.org", 
  id = 2, 
  name = "Dirk Eddelbuettel" )
# update the phone repeated field with a list of PhoneNumber messages
dirk$phone <- list(
  new( PhoneNumber, number = "+01...", type = "MOBILE" ),
  new( PhoneNumber, number = "+01...", type = "HOME" ) )

# with/within style
saptarshi <- within( new(Person), {
  id <- 3
  name <- "Saptarshi Guha"
  email <- "saptarshi.guha@gmail.com"
})

# make an addressbook
book <- new( tutorial.AddressBook, person = list( romain, dirk, saptarshi ) )

# serialize the message to a file
tf <- tempfile() 
serialize( book, tf )

# the payload of the message
serialize( book, NULL )

# read the file into a new message
m <- tutorial.AddressBook$read( tf )
writelines( as.character( m ) )
sapply( m$person, function(p) p$name )
Methods

- **as.character** signature(x = "MethodDescriptor"): debug string of the method
- **toString** signature(x = "MethodDescriptor"): debug string of the method
- **$** signature(x = "MethodDescriptor"): ...
- **$<-** signature(x = "MethodDescriptor"): ...
- **input_type** signature(object = "MethodDescriptor"): the **Descriptor** of the input type of the method
- **output_type** signature(object = "MethodDescriptor"): the **Descriptor** of the output type of the method

Author(s)

Romain Francois <francoisromain@free.fr>

<table>
<thead>
<tr>
<th>name</th>
<th>Name or full name of a descriptor</th>
</tr>
</thead>
</table>

Description

name or full name of a descriptor

Methods

- signature(object = "Descriptor") ...
- signature(object = "FieldDescriptor") ...
- signature(object = "EnumDescriptor") ...
- signature(object = "ServiceDescriptor") ...
- signature(object = "MethodDescriptor") ...

<table>
<thead>
<tr>
<th>nested_type-methods</th>
<th>Extract a message type descriptor for a nested type</th>
</tr>
</thead>
</table>

Description

Extract a **Descriptor** nested in another **Descriptor**

See Also

The method is implemented for the **Descriptor** class
### nested_type_count-methods

*The number of fields*

#### Description

The number of fields

#### See Also

The method is implemented for the **Descriptor** class

### Next-methods

*Obtains a chunk of data from the stream*

#### Description

Obtains a chunk of data from the stream

#### See Also

ZeroCopyInputStream implements Next.

### number-methods

*Gets the declared tag number of a field*

#### Description

Gets the declared tag number of a field

#### See Also

The method is implemented for **FieldDescriptor** and **EnumValueDescriptor** classes.

#### Examples

```r
## Not run:
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )
Person <- P("tutorial.Person", file = proto.file )

## End(Not run)

number(Person$id)
number(Person$email)
as.character(Person)
number(value(tutorial.Person$PhoneType, name="HOME"))
```
Protocol Buffer descriptor importer

Description

The `p` function searches for a protocol message descriptor in the descriptor pool.

Usage

`p(type, file)`

Arguments

- `type`: Fully qualified type name of the protocol buffer or extension
- `file`: optional proto file. If given, the definition contained in the file is first registered with the pool of message descriptors

Value

An object of class `Descriptor` for message types or `FieldDescriptor` for extensions. An error is generated otherwise.

Author(s)

Romain Francois <francoisromain@free.fr>

Examples

```r
## Not run:
proto.file <- system.file( "proto", "addressbook.proto", package = "RProtoBuf" )
Person <- p( "tutorial.Person", file = proto.file )

## End(Not run)

cat(as.character( Person ))
```

read-methods

Read a protocol buffer message from a connection

Description

Read a `Message` from a connection using its associated `Descriptor`
Methods

signature(descriptor = "Descriptor", input = "character") Read the message from a file
signature(descriptor = "Descriptor") Read from a binary connection.
signature(descriptor = "Descriptor", input = "raw") Read the message from a raw vector

Examples

# example file that contains a "tutorial.AddressBook" message
book <- system.file( "examples", "addressbook.pb", package = "RProtoBuf" )

# read the message
message <- read( tutorial.AddressBook, book )

# or using the pseudo method
message <- tutorial.AddressBook$read( book )

# write its debug string
writelines( as.character( message ) )

# grab the name of each person
sapply( message$person, function(p) p$name )

# read from a binary file connection
f <- file( book, open = "rb" )
message2 <- read( tutorial.AddressBook, f )
close( f )

# read from a message payload (raw vector)
payload <- readBin( book, raw(0), 5000 )
message3 <- tutorial.AddressBook$read( payload )

---

readASCII-methods read a message in ASCII format

Description

Method to read a Message in ASCII format

Methods

signature(descriptor = "Descriptor", input = "ANY") Read the message from a connection (file, etc ...)
signature(descriptor = "Descriptor", input = "character") Read the message directly from the character string
Examples

```r
# example file that contains a "tutorial.AddressBook" message
book <- system.file( "examples", "addressbook.pb", package = "RProtoBuf" )

# read the message
message <- read( tutorial.AddressBook, book )

# Output in text format to a temporary file
out.file <- tempfile()
writelines( as.character(message), file(out.file))

# Verify that we can read back in the message from a text file.
message2 <- readASCII( tutorial.AddressBook, file(out.file, "rb"))

# Verify that we can read back in the message from an unopened file.
message3 <- readASCII( tutorial.AddressBook, file(out.file))
```

---

**readProtoFiles**  
*protocol buffer descriptor importer*

**Description**

Imports proto files into the descriptor pool that is then used by the `P` function to resolve message type names.

**Usage**

```r
readProtoFiles(files, dir, package="RProtoBuf", pattern="\.proto$", lib.loc=NULL)
```

**Arguments**

- `files`  
  Proto files

- `dir`  
  Directory. If `files` is not specified, files with the "proto" extension in the `dir` directory are imported

- `package`  
  R package name. If `files` and `dir` are missing, "proto" files in the "proto" directory of the package tree are imported.

- `pattern`  
  A filename pattern to match proto files.

- `lib.loc`  
  Library location.

**Value**

`NULL`, invisibly.

**Author(s)**

Romain Francois <francoisromain@free.fr>
RpcHTTP-class

See Also

Examples

```r
## Not run:
# from a package
dump( proto_files )
readProtoFiles( package = "RProtoBuf" )

# from a directory
proto.dir <- system.file("proto", package = "RProtoBuf")
readProtoFiles( dir = proto.dir )

# set of files
proto.files <- list.files( proto.dir, full.names = TRUE )
readProtoFiles( proto.files )

## End(Not run)
```

---

Class "RpcHTTP"

Description

Support for protobuf rpc over HTTP

Objects from the Class

Objects can be created by calls of the form `new("RpcHTTP", host = "somehost", port = port.number, root = "")`

Slots

- `host`: Host name
- `port`: port number
- `root`: root directory of the protobuf http server

Author(s)

Romain Francois <francoisromain@free.fr>

See Also

`invoke` uses objects of this class to perform a method invocation over http.
Serialize R object to Protocol Buffer Message.

Description

Serializes R objects to a general purpose protobuf message using the same \texttt{rexp.proto} descriptor and mapping between R objects and protobuf messages as RHIPE.

Usage

\begin{verbatim}
serialize_pb(object, connection, ...)
\end{verbatim}

Arguments

- \texttt{object}: R object to serialize
- \texttt{connection}: passed on to \texttt{serialize}
- \texttt{...}: additional arguments passed on to \texttt{serialize}

Details

Clients need both the message and the \texttt{rexp.proto} descriptor to parse serialized R objects. The latter is included in the package installation proto directory: \texttt{system.file(package="RProtoBuf", "proto/rexp.proto")}.

The following storage types are natively supported by the descriptor: character, raw, double, complex, integer, list, and NULL. Objects with other storage types, such as functions, environments, S4 classes, etc, are serialized using base R \texttt{serialize} and stored in the proto native type. Missing values, attributes and numeric precision will be preserved.

Examples

\begin{verbatim}
msg <- tempfile();
serialize_pb(iris, msg);
obj <- unserialize_pb(msg);
identical(iris, obj);
\end{verbatim}

ServiceDescriptor-class

\texttt{Class "ServiceDescriptor"}

Description

R representation of Service Descriptors

Objects from the Class

TODO
SetCloseOnDelete-methods

Slots

pointer: External pointer to a google::protobuf::ServiceDescriptor C++ object
name: fully qualified name of the service

Methods

as.character signature(x = "ServiceDescriptor"): debug string of the service
toString signature(x = "ServiceDescriptor"): debug string of the service
show signature(x = "ServiceDescriptor"): ...
$ signature(x = "ServiceDescriptor"): invoke pseudo methods or retrieve method descriptors contained in this service descriptor.
[[ signature(x = "ServiceDescriptor"): extracts methods descriptors contained in this service descriptor
length signature(x = "ServiceDescriptor"): number of MethodDescriptor
method_count signature(x = "ServiceDescriptor"): number of MethodDescriptor
method signature(x = "ServiceDescriptor"): retrieves a MethodDescriptor

Author(s)

Romain Francois <francoisromain@free.fr>

---

set-methods

set a subset of values of a repeated field of a message

---

Description

set a subset of values of a repeated field of a message

Methods

signature(object = "Message"): set a subset of values of a repeated field of a message

---

SetCloseOnDelete-methods

set the close on delete behavior

---

Description

By default, the file descriptor is not closed when a stream is destroyed, use SetCloseOnDelete(stream, TRUE) to change that.

Methods

See classes FileInputStream and FileOutputStream for implementations.
**Description**

The number of objects currently in a given field of a protocol buffer message. For non repeated fields, the size is 1 if the message has the field, 0 otherwise. For repeated fields, the size is the number of objects in the array. For repeated fields, the size can also be assigned in order to shrink or grow the vector. Numeric types are given a default value of 0 when the new size is greater than the existing size. Character types are given a default value of "". Growing a repeated field in this way is not supported for message, group, and enum types.

**Methods**

signature(object = "Message") Number of objects in a message field

**Examples**

```r
unittest.proto.file <- system.file("unitTests", "data", "unittest.proto",
  package = "RProtoBuf")
readProtoFiles(file = unittest.proto.file)

test <- new(protobuf_unittest.TestAllTypes)
test$size("optional_int32")

test$add("repeated_int32", 1:10)
test$size("repeated_int32")
test$repeated_int32

size(test, "repeated_int32") <- 5
test$repeated_int32

size(test, "repeated_int32") <- 15
test$repeated_int32
```

**sizegets**

*Set the size of a field*

**Description**

Sets the size of a repeated field.

**Methods**

signature(object = "Message") sets the size of a message field
Skip-methods

Skips a number of bytes

Description

Skips a number of bytes

swap-methods

swap elements of a repeated field of a message

Description

swap elements of a repeated field of a message.

Methods

signature(object = "Message") swap elements of a repeated field of a message

References

See the SwapElements of the Reflection class, part of the protobuf library.  

type-methods

Gets the type or the C++ type of a field

Description

Gets the type or the C++ type of a field

Arguments

object A FieldDescriptor object.
as.string If true, print a string representation of the type.

See Also

The method is implemented for the FieldDescriptor class
with.Message

Examples

```r
## Not run:
proto.file <- system.file(  "proto",  "addressbook.proto", package = "RProtoBuf" )
Person <- P("tutorial.Person", file = proto.file )

## End(Not run)
type(Person$id)
type(Person$id, as.string=TRUE)
cpp_type(Person$email)
cpp_type(Person$email, TRUE)
```

Description

Convenience wrapper that allow getting and setting fields of protocol buffer messages from within
the object

Usage

```r
## S3 method for class 'Message'
with(data, expr, ...)
## S3 method for class 'Message'
within(data, expr, ...)
```

Arguments

- `data`: A protocol buffer message, instance of `Message`
- `expr`: R expression to evaluate
- `...`: ignored

Details

The expression is evaluated in an environment that allows to set and get fields of the message
The fields of the message are mapped to active bindings (see `makeActiveBinding`) so that they can
be accessed and modified from within the environment.

Value

- `with` returns the value of the expression
- `within` returns the data argument.

Author(s)

Romain Francois <francoisromain@free.fr>
Examples

```r
## Not run:
proto.file <- system.file("proto", "addressbook.proto", package = "RProtoBuf")
Person <- P("tutorial.Person", file = proto.file)

## End(Not run)

romain <- within(new(Person),{
  email <- "francoisromain@free.fr"
  id <- 10L
})
```

---

**ZeroCopyInputStream-class**

*Virtual Class "ZeroCopyInputStream"

---

**Description**

R wrapper for the ZeroCopyInputStream c++ class

**Objects from the Class**

This is a virtual class

**Slots**

pointer: external pointer to the google::protobuf::io::ZeroCopyInputStream object

**Methods**

- `$ signature(x="ZeroCopyInputStream")`: invokes a method
- `Next signature(object="ZeroCopyInputStream")`: Get a number of bytes from the stream as a raw vector.
- `Skip signature(object="ZeroCopyInputStream")`: skip a number of bytes
- `BackUp signature(object="ZeroCopyInputStream")`: Backs up a number of bytes, so that the next call to `Next` returns data again that was already returned by the last call to `Next`.
- `ByteCount signature(object="ZeroCopyInputStream")`: Returns the total number of bytes read since this object was created.
- `ReadRaw signature(object="ZeroCopyInputStream", size = "integer")`: read raw bytes from the stream
- `ReadRaw signature(object="ZeroCopyInputStream", size = "numeric")`: read raw bytes from the stream
- `ReadString signature(object="ZeroCopyInputStream", size = "integer")`: same as `ReadRaw` but formats the result as a string
**ZeroCopyOutputStream-class**

**Description**

R wrapper for the ZeroCopyOutputStream c++ class

**Objects from the Class**

This is a virtual class

**Slots**

`pointer`: external pointer to the google::protobuf::io::ZeroCopyOutputStream object

**Author(s)**

Romain Francois <francoisromain@free.fr>

**References**


**See Also**

TODO: add classes that extend
Methods

*-signature(x="ZeroCopyOutputStream"): invokes a method

**Next** signature(object="ZeroCopyOutputStream", payload = "raw"): push the raw vector into the stream. Returns the number of bytes actually written.

**BackUp** signature(object="ZeroCopyOutputStream"): Backs up a number of bytes, so that the end of the last buffer returned by Next is not actually written.

**ByteCount** signature(object="ZeroCopyOutputStream"): Returns the total number of bytes written since this object was created.

**WriteRaw** signature(object="ZeroCopyOutputStream", payload = "raw"): write the raw bytes to the stream

Author(s)

Romain Francois <francoisromain@free.fr>

References


See Also

TODO: add classes that extend
Index

! =, Message, Message-method
(Message-class), 34

* Topic classes
  ArrayInputStream-class, 4
  ArrayOutputStream-class, 6
  ConnectionInputStream-class, 13
  ConnectionOutputStream-class, 15
  Descriptor-class, 16
  EnumDescriptor-class, 18
  EnumValueDescriptor-class, 20
  FieldDescriptor-class, 23
  FileDescriptor-class, 26
  FileInputStream-class, 27
  FileOutputStream-class, 29
  Message-class, 34
  MethodDescriptor-class, 36
  RpcHTTP-class, 42
  ServiceDescriptor-class, 43
  with . Message, 47
  ZeroCopyInputStream-class, 48
  ZeroCopyOutputStream-class, 49

* Topic interface
  P, 39

* Topic methods
  add-methods, 4
  ArrayInputStream-methods, 6
  ArrayOutputStream-methods, 7
  BackUp-methods, 10
  ByteCount-methods, 10
  bytesize-methods, 10
  clear-methods, 11
  clone-methods, 11
  ConnectionInputStream-methods, 14
  ConnectionOutputStream-methods, 15
  containing_type-methods, 16
  descriptor-methods, 18
  enum_type-methods, 21
  enum_type_count-methods, 22
  fetch-methods, 22
  field-methods, 22
  field_count-methods, 25
  fileDescriptor-methods, 27
  FileInputStream-methods, 28
  FileOutputStream-methods, 30
  GetErrno-methods, 30
  has-methods, 30
  invoke-methods, 31
  is_extension-methods, 32
  isInitialized-methods, 31
  label-methods, 32
  merge-methods, 33
  name, 37
  nested_type-methods, 37
  nested_type_count-methods, 38
  Next-methods, 38
  number-methods, 38
  read-methods, 39
  readASCII-methods, 40
  set-methods, 44
  SetCloseOnDelete-methods, 44
  size-methods, 45
  sizegets, 45
  Skip-methods, 46
  swap-methods, 46
  type-methods, 46

* Topic package
  RProtoBuf-package, 3

* Topic programming
  as . list . Message, 7
  asMessage, 9
  completion, 12
  readProtoFiles, 41
  . DollarNames . Descriptor ( completion ), 12
  . DollarNames . EnumDescriptor ( completion ), 12
  . DollarNames . FieldDescriptor ( completion ), 12
  . DollarNames . FileDescriptor
all.equal, Message, Message-method (Message-class), 34
ArrayInputStream, 5, 6
ArrayInputStream
(ArrayInputStream-methods), 6
ArrayInputStream, raw, integer-method (ArrayInputStream-methods), 6
ArrayInputStream, raw, missing-method (ArrayInputStream-methods), 6
ArrayInputStream, raw, numeric-method (ArrayInputStream-methods), 6
ArrayInputStream-class, 4
ArrayInputStream-methods, 6
ArrayOutputStream, 6, 7
ArrayOutputStream
(ArrayOutputStream-methods), 7
ArrayOutputStream, integer, integer-method (ArrayOutputStream-methods), 7
ArrayOutputStream, integer, missing-method (ArrayOutputStream-methods), 7
ArrayOutputStream, integer, numeric-method (ArrayOutputStream-methods), 7
ArrayOutputStream, numeric, integer-method (ArrayOutputStream-methods), 7
ArrayOutputStream, numeric, missing-method (ArrayOutputStream-methods), 7
ArrayOutputStream, numeric, numeric-method (ArrayOutputStream-methods), 7
ArrayOutputStream-class, 6
ArrayOutputStream-methods, 7
as, 9
as.character, Descriptor-method (Descriptor-class), 16
as.character, EnumDescriptor-method (EnumDescriptor-class), 18
as.character, EnumValueDescriptor-method (EnumValueDescriptor-class), 20
as.character, FieldDescriptor-method (FieldDescriptor-class), 23
as.character, FileDescriptor-method (FileDescriptor-class), 26
as.character, Message-method (Message-class), 34
as.character, MethodDescriptor-method (MethodDescriptor-class), 36
as.character, ServiceDescriptor-method (ServiceDescriptor-class), 43
as.character, ZeroCopyInputStream-method (ZeroCopyInputStream-class), 48
as.character, ZeroCopyOutputStream-method (ZeroCopyOutputStream-class), 49
as<-, Descriptor-method (Descriptor-class), 16
as<-, Message-method (Message-class), 34
as<-, MethodDescriptor-method (MethodDescriptor-class), 36
add (add-methods), 4
add, Message-method (add-methods), 4
add-methods, 4
asNlistNenumdescriptor
(asNlistNmessage), 7
asNlistNFileDescriptor
(asNlistNmessage), 7
asNlistNMessage, 7
asNlistNServiceDescriptor
(asNlistNmessage), 7
asMessage, 9
ConnectionInputStream, 13, 14
ConnectionInputStream
(ConnectionInputStream-methods), 14
ConnectionInputStream,connection-method
(ConnectionInputStream-methods), 14
ConnectionInputStream-class, 13
ConnectionInputStream-methods, 14
ConnectionOutputStream, 15
ConnectionOutputStream
(ConnectionOutputStream-methods), 15
ConnectionOutputStream-class, 15
ConnectionOutputStream-methods, 15
containing_type
(containing_type-methods), 16
containing_type,Descriptor-method
(Descriptor-class), 16
containing_type,EnumDescriptor-method
(EnumDescriptor-class), 18
containing_type,FieldDescriptor-method
(FieldDescriptor-class), 23
containing_type-methods, 16
cppN_type-methods, 46
cppN_type,FieldDescriptor-method
(FieldDescriptor-class), 23
cppN_type-methods(type-methods), 46
CPPTYPE_BOOL (type-methods), 46
CPPTYPE_DOUBLE (type-methods), 46
CPPTYPE_ENUM (type-methods), 46
CPPTYPE_FLOAT (type-methods), 46
CPPTYPE_INT32 (type-methods), 46
CPPTYPE_INT64 (type-methods), 46
CPPTYPE_MESSAGE (type-methods), 46
CPPTYPE_STRING (type-methods), 46
CPPTYPE_UINT32 (type-methods), 46
CPPTYPE_UINT64 (type-methods), 46
defaultN_value (FieldDescriptor-class), 23
defaultN_value,FieldDescriptor-method
(FieldDescriptor-class), 23
defaultN_value-methods
(FieldDescriptor-class), 23
Backup (Backup-methods), 10
Backup,ZeroCopyInputStream-method
(ZeroCopyInputStream-class), 48
Backup,ZeroCopyOutputStream-method
(ZeroCopyOutputStream-class), 49
ByteCount-methods, 10
ByteCount,ZeroCopyInputStream-method
(ZeroCopyInputStream-class), 48
ByteCount,ZeroCopyOutputStream-method
(ZeroCopyOutputStream-class), 49
ByteCount-methods, 10
bytesize (bytesize-methods), 10
bytesize,Message-method
(bytesize-methods), 10
bytesize-methods, 10
can_serialize_pb (serialize_pb), 43
clear (clear-methods), 11
clear,Message,character-method
(clear-methods), 11
clear,Message,integer-method
(clear-methods), 11
clear,Message,missing-method
(clear-methods), 11
clear,Message,numeric-method
(clear-methods), 11
clear,Message,raw-method
(clear-methods), 11
clear-methods, 11
close (clone-methods), 11
close,Message-method (clone-methods), 11
close-methods, 11
close,FileInputStream-method
(FileInputStream-class), 27
close,FileOutputStream-method
(FileOutputStream-class), 29
completion, 12
nested_type-methods, 37  
nested_type_count  
  (nested_type_count-methods), 38  
nested_type_count, Descriptor-method  
  (Descriptor-class), 16  
nested_type_count-methods, 38  
new, Descriptor-method  
  (Descriptor-class), 16  
Next (Next-methods), 38  
Next, ZeroCopyInputStream, missing-method  
  (ZeroCopyInputStream-class), 48  
Next, ZeroCopyOutputStream, raw-method  
  (ZeroCopyOutputStream-class), 49  
Next-methods, 38  
number (number-methods), 38  
number, EnumValueDescriptor-method  
  (EnumValueDescriptor-class), 20  
number, FieldDescriptor-method  
  (FieldDescriptor-class), 23  
number-methods, 38  
output_type (MethodDescriptor-class), 36  
output_type, MethodDescriptor-method  
  (MethodDescriptor-class), 36  
output_type-methods  
  (MethodDescriptor-class), 36

P, 16, 17, 35, 39, 42

read (read-methods), 39  
read, Descriptor, ANY-method  
  (read-methods), 39  
read, Descriptor, character-method  
  (read-methods), 39  
read, Descriptor, raw-method  
  (read-methods), 39  
read-methods, 39  
readASCII (readASCII-methods), 40  
readASCII, Descriptor, ANY-method  
  (readASCII-methods), 40  
readASCII, Descriptor, character-method  
  (readASCII-methods), 40  
readASCII-methods, 40  
ReadLittleEndian32  
  (ZeroCopyInputStream-class), 48  
ReadLittleEndian32, ZeroCopyInputStream-method  
  (ZeroCopyInputStream-class), 48  
ReadLittleEndian32-methods  
  (ZeroCopyInputStream-class), 48  
ReadLittleEndian64  
  (ZeroCopyInputStream-class), 48  
ReadLittleEndian64, ZeroCopyInputStream-method  
  (ZeroCopyInputStream-class), 48  
ReadLittleEndian64-methods  
  (ZeroCopyInputStream-class), 48  
readProtoFiles, 41  
ReadRaw (ZeroCopyInputStream-class), 48  
ReadRaw, ZeroCopyInputStream, integer-method  
  (ZeroCopyInputStream-class), 48  
ReadRaw, ZeroCopyInputStream, numeric-method  
  (ZeroCopyInputStream-class), 48  
ReadRaw-methods  
  (ZeroCopyInputStream-class), 48  
ReadString (ZeroCopyInputStream-class), 48  
ReadString, ZeroCopyInputStream, integer-method  
  (ZeroCopyInputStream-class), 48  
ReadString, ZeroCopyInputStream, numeric-method  
  (ZeroCopyInputStream-class), 48  
ReadString-methods  
  (ZeroCopyInputStream-class), 48  
ReadVarint32  
  (ZeroCopyInputStream-class), 48  
ReadVarint32, ZeroCopyInputStream-method  
  (ZeroCopyInputStream-class), 48  
ReadVarint32-methods  
  (ZeroCopyInputStream-class), 48  
ReadVarint64  
  (ZeroCopyInputStream-class), 48  
ReadVarint64, ZeroCopyInputStream-method  
  (ZeroCopyInputStream-class), 48  
ReadVarint64-methods  
  (ZeroCopyInputStream-class), 48  
Rpchttp-class, 42  
RProtoBuf (RProtoBuf-package), 3  
RProtoBuf-package, 3  
serialize, 43  
serialize, Message-method  
  (Message-class), 34  
serialize_pb, 43  
ServiceDescriptor-class, 43  
set (set-methods), 44  
service-set, 44  
service-set-methods, 44  
set-methods, 44
WriteLittleEndian32, ZeroCopyOutputStream, integer-method (ZeroCopyOutputStream-class), 49
WriteLittleEndian32, ZeroCopyOutputStream, numeric-method (ZeroCopyOutputStream-class), 49
WriteLittleEndian32, ZeroCopyOutputStream, raw-method (ZeroCopyOutputStream-class), 49
WriteLittleEndian32-methods (ZeroCopyOutputStream-class), 49
WriteLittleEndian64, ZeroCopyOutputStream, integer-method (ZeroCopyOutputStream-class), 49
WriteLittleEndian64, ZeroCopyOutputStream, numeric-method (ZeroCopyOutputStream-class), 49
WriteLittleEndian64, ZeroCopyOutputStream, raw-method (ZeroCopyOutputStream-class), 49
WriteLittleEndian64-methods (ZeroCopyOutputStream-class), 49
WriteRaw (ZeroCopyOutputStream-class), 49
WriteRaw, ZeroCopyOutputStream, raw-method (ZeroCopyOutputStream-class), 49
WriteRaw-methods (ZeroCopyOutputStream-class), 49
WriteString (ZeroCopyOutputStream-class), 49
WriteString, ZeroCopyOutputStream, character-method (ZeroCopyOutputStream-class), 49
WriteString-methods (ZeroCopyOutputStream-class), 49
WriteVarint32 (ZeroCopyOutputStream-class), 49
WriteVarint32, ZeroCopyOutputStream, integer-method (ZeroCopyOutputStream-class), 49
WriteVarint32, ZeroCopyOutputStream, numeric-method (ZeroCopyOutputStream-class), 49
WriteVarint32, ZeroCopyOutputStream, raw-method (ZeroCopyOutputStream-class), 49
WriteVarint32-methods (ZeroCopyOutputStream-class), 49
WriteVarint64, ZeroCopyOutputStream, integer-method (ZeroCopyOutputStream-class), 49
WriteVarint64, ZeroCopyOutputStream, numeric-method (ZeroCopyOutputStream-class), 49
WriteVarint64, ZeroCopyOutputStream, raw-method (ZeroCopyOutputStream-class), 49
WriteVarint64-methods (ZeroCopyOutputStream-class), 49
ZeroCopyInputStream, 4, 5, 10, 13, 14, 27, 28, 38
ZeroCopyInputStream-class, 48
ZeroCopyOutputStream, 6, 7, 15, 29
ZeroCopyOutputStream-class, 49