Package ‘RBaseX’

December 2, 2022

Type Package
Title 'BaseX' Client
Version 1.1.2
Date 2022-12-02
Description 'BaseX' <https://basex.org> is a XML database engine and a compliant 'XQuery 3.1' processor with full support of 'W3C Update Facility'. This package is a full client-implementation of the client/server protocol for 'BaseX' and provides functionalities to create, manipulate and query on XML-data.

License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.2.2
Imports R6, RCurl, pingr, rex, htrr, stringr, dplyr, openssl, magrittr, tibble, data.table
Suggests testthat, glue
Author Ben Engbers [aut, cre]
Maintainer Ben Engbers <Ben.Engbers@Be-Logical.nl>
URL https://github.com/BenEngbers/RBaseX
SystemRequirements Needs a running BaseX server instance. The testuser with credentials ('Test'/'testBasex') should have admin rights.
Repository CRAN
NeedsCompilation no
Date/Publication 2022-12-02 13:40:02 UTC

R topics documented:

Add ................................................................. 2
Bind .................................................................. 3
Close .................................................................. 4
Command .......................................................... 5
Context ............................................................. 6
Description

Adds a new resource to the opened database.

Usage

Add(session, path, input)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>BasexClient instance-ID</td>
</tr>
<tr>
<td>path</td>
<td>Path</td>
</tr>
<tr>
<td>input</td>
<td>Additional input (optional)</td>
</tr>
</tbody>
</table>
Details

The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server. The utility-function `input_to_raw` can be used to convert an arbitrary character vector to a stream. This method returns `self` invisibly, thus making it possible to chain together multiple method calls.

Value

A list with two items

- info Additional info
- success Boolean, indicating if the command was completed successfully

Examples

```r
## Not run:
Add(Session, "test", "<xml>Add</xml>"

## End(Not run)
```

Description

Binds a value to a variable.

Usage

```r
Bind(query_obj, ...)
```

Arguments

- `query_obj` QueryClass instance-ID
- `...` Binding Information

Details

Binding information can be provided in the following ways:

- name, value Name and value for a variable.
- name, value, type Name, value and type for a variable.
- name, list(value) Name, list of values.
- name, list(value), list(type) Name, list of values, list of types.

For a list of possible types see [https://docs.basex.org/wiki/Java_Bindings#Data_Types](https://docs.basex.org/wiki/Java_Bindings#Data_Types)

This method returns `self` invisibly, thus making it possible to chain together multiple method calls.
Value

Boolean value which indicates if the operation was executed successfully.

Examples

```r
## Not run:
query_obj <- Query(Session,
    "declare variable $name external; for $i in 1 to 2 return element { $name } { $i }")
Bind(query_obj, "$name", "number")
print(Execute(query_obj))

query_obj <- Query(Session,
    "declare variable $name external; for $i in 3 to 4 return element { $name } { $i }")
Bind(query_obj, "$name", "number", "xs:string")
print(Execute(query_obj))

query_obj <- Query(Session,
    "declare variable $name external; for $t in collection('/quotesingle.Var TestDB/Books/quotesingle.Var')/book where $t/@author = $name return $t/@title/string()")
Bind(query_obj, "$name", list("Walmsley", "Wickham"))
print(Execute(query_obj))

query_obj <- Query(Session,
    "declare variable $name external; for $t in collection('TestDB/Books')/book where $t/@author = $name return $t/@title/string()")
Bind(query_obj, "$name", list("Walmsley", "Wickham"), list("xs:string", "xs:string"))
print(Execute(query_obj))

## End(Not run)
```

Description

Closes and unregisters the query with the specified ID.

Usage

Close(query_obj)

Arguments

query_obj QueryClass instance-ID
**Command**

**Details**

This method returns *self* invisibly, thus making it possible to chain together multiple method calls.

**Value**

This function returns a list with the following items:

- **info** Info
- **success** A boolean, indicating if the command was completed successfully

---

**Description**

Executes a database command or a query.

**Usage**

`Command(...)`

**Arguments**

... The command or query to be executed. When used to execute a command, a SessionID and a string which contains the command, are to be passed. When used to execute a query, the QueryClass instance-ID is passed.

**Details**

For a list of database commands see [https://docs.basex.org/wiki/Commands](https://docs.basex.org/wiki/Commands)

'BaseX' can be used in a Standard mode or Query mode.

In the standard mode of the Clients, a database command can be sent to the server using the Command() function of the Session. The query mode of the Clients allows you to bind external variables to a query and evaluate the query in an iterative manner.

**Value**

When used to execute commands in the Standard mode, this function returns a list with the following items:

- **result**
- **info** Additional info
- **success** A boolean, indicating if the command was completed successfully

When used to execute a query, it returns the result as a list.
## Not run:
Session <- NewBasexClient(user = <username>, password = "<password>")
print(Command(Session, "info")$info)

query_txt <- paste("for $i in 1 to 2", "return <xml>Text { $i }</xml>" , sep = " ")
query_obj <- Query(Session, query_txt)
print(Command(query_obj))

## End(Not run)

---

### Context

#### Description

Binds a value to the context. The type will be ignored if the string is empty. The function returns no value.

#### Usage

```
Context(query_obj, value, type)
```

#### Arguments

- `query_obj` QueryClass instance-ID
- `value` Value that should be bound to the context
- `type` The type will be ignored when the string is empty

#### Details

The type that is provided to the context, should be one of the standard-types. An alternative way is to parse the document information. This method returns `self` invisibly, thus making it possible to chain together multiple method calls.

#### Examples

```r
## Not run:
ctxt_query_txt <- "for $t in .//text() return string-length($t)"
ctxt_query <- Query(Session, ctxt_query_txt)
ctxt_txt <- paste0("<xml>",
  "<txt>Hi</txt>",
  "<txt>World</txt>",
  "</xml>"
)
Context(ctxt_query, ctxt_txt, type = "document-node()")
print(Execute(ctxt_query)) # returns "2" "5"
```
Create

ctxt_query_txt <- "for $t in parse-xml(.)//text() return string-length($t)"
Context(ctxt_query, ctxt_txt)
print(Execute(ctxt_query))

## End(Not run)

---

Create Create

Description

Creates a new database with the specified name and input (may be empty).

Usage

Create(session, name, input)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>BasexClient instance-ID</td>
</tr>
<tr>
<td>name</td>
<td>Database name</td>
</tr>
<tr>
<td>input</td>
<td>Additional input, may be empty</td>
</tr>
</tbody>
</table>

Details

The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server. ‘Check’ is a convenience command that combines OPEN and CREATE DB: If a database with the name input exists, and if there is no existing file or directory with the same name that has a newer timestamp, the database is opened. Otherwise, a new database is created; if the specified input points to an existing resource, it is stored as initial content. This method returns self invisibly, thus making it possible to chain together multiple method calls.

Value

A list with two items

- info Aditional info
- success A boolean, indicating if the command was completed successfull
Execute

Examples

```plaintext
## Not run:
Create(, "test", "<xml>Create test</xml>")
Execute(Session, "Check test")
Create(Session, "test2",
    "https://raw.githubusercontent.com/BaseXdb/basex/master/basex-api/src/test/resources/first.xml")
Create(Session, "test3", "/home/username/Test.xml")
## End(Not run)
```

Description

Executes a database command or a query.

Usage

```
Execute(...) 
```

Arguments

```
... The command or query to be executed. When used to execute a command, a 
SessionID and a string which contains the command, are to be passed. When 
used to execute a query, the QueryClass instance-ID is passed.
```

Details

The 'Execute' command is deprecated and has been renamed to 'Command'. 'Execute' is being 
kept as convenience.

Value

When used to execute commands in the Standard mode, this function returns a list with the following 
items:

- result
- info Aditional info
- success A boolean, indicating if the command was completed successfull

When used to execute a query, it return the result as a list.
Examples

```r
## Not run:
Session <- NewBaseClient(user = <username>, password = "<password>")
print(Execute(Session, "info")$info)

query_txt <- paste("for $i in 1 to 2", "return <xml>Text { $i </xml>", sep = " ")
query_obj <- Query(Session, query_txt)
print(Execute(query_obj))

## End(Not run)
```

<table>
<thead>
<tr>
<th>Full</th>
<th>Title Full</th>
</tr>
</thead>
</table>

Description

Executes a query and returns a list of vectors, each one representing a result as a string, prefixed by the 'XDM' (Xpath Data Model) Meta Data <https://www.xdm.org/>. Meta Data and results are separated by a '|'.

Usage

`Full(query_obj)`

Arguments

- `query_obj` QueryClass instance-ID

Examples

```r
## Not run:
query_txt <- "collection('/TestDB/Test.xml')"
query_obj <- Query(Session, query_txt)

print(Full(query_obj))

## Return
[[1]]
[1] "2f" 
"/TestDB/Test.xml"
[[2]]
[1] "3c" 
"Line_1 line="1">Content 1</Line_1"
[[3]]
[1] "2f" 
"/TestDB/Test.xml"
[[4]]
[1] "3c" 
"Line_2 line="2">Content 2</Line_2"

## End(Not run)
```
GetIntercept

Description
Current value for session\$Intercept

Usage
GetIntercept(session)

Arguments
- session: BasexClient instance-ID

Value
Current value

GetSuccess

Description
Current value from session\$Success

Usage
GetSuccess(session)

Arguments
- session: BasexClient instance-ID

Value
Current value
Description
Returns a string with query compilation and profiling info.

Usage
Info(query_obj)

Arguments
query_obj QueryClass instance-ID

Details
If the query object has not been executed yet, an empty string is returned.

Value
This function returns a list with the following items:
• Info Info
• success A boolean, indicating if the command was completed successful

Description
Convert input to a length-1 character vector.

Usage
input_to_raw(input)

Arguments
input Character vector length 1

Details
If input is a reference to a file, the number of bytes corresponding to the size is read. If it is an URL, the URL is read and converted to a 'Raw' vector. The function does not catch errors.

Value
'Raw' vector
Description

Indicates if there are any other results in the query-result.

Usage

More(query_obj)

Arguments

query_obj QueryClass instance-ID

Value

Boolean

Examples

## Not run:
Query_1 <- Query(Session, "collection('TestDB/Test.xml')")
iterResult <- c()

while (More(Query_1)) {
    iterResult <- c(iterResult, Next(Query_1))
}

print(iterResult)

[[1]]
[1] "0d" 
  "<Line_1 line="1">Content 1</Line_1>"

[[2]]
[1] "0d" 
  "<Line_2 line="2">Content 2</Line_2>"

## End(Not run)
NewBasexClient

Description
Create a BaseX-client

Usage
NewBasexClient(host = "localhost", port = 1984, user, password)

Arguments
host, port         Host name and port-number
user, password     User credentials

Details
This creates a BaseX-client. By default it listens to port 1984 on localhost. Username and password should be changed after the installation of `BaseX`.

Value
BasexClient-instance

Examples

```r
## Not run:
session <- NewBasexClient(user = <username>, password = "<password>"

## End(Not run)
```

Next

Description
Returns the next result when iterating over a query

Usage
Next(query_obj)
Arguments

query_obj QueryClass instance-ID

Examples

```r
## Not run:
Query_1 <- Query(Session, "collection('TestDB/Test.xml')")
iterResult <- c()

while (More(Query_1)) {
  iterResult <- c(iterResult, Next(Query_1))
}

print(iterResult)

[[1]]
1 "0d" "<Line_1 line="1">Content 1</Line_1>"

[[2]]
1 "0d" "<Line_2 line="2">Content 2</Line_2>"

## End(Not run)
```

Options

<table>
<thead>
<tr>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
</table>

Returns a string with all query serialization parameters, which can be assigned to the serializer option.

Usage

Options(query_obj)

Arguments

query_obj QueryClass instance-ID

Details

For a list of possible types see [https://docs.basex.org/wiki/Java_Bindings#Data_Types](https://docs.basex.org/wiki/Java_Bindings#Data_Types)
Value

This function returns a list with the following items:

- Options Options
- success A boolean, indicating if the command was completed successfully

Description

Adds or replaces a resource with the specified input.

Usage

put(session, path, input)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>BaseXClient instance-ID</td>
</tr>
<tr>
<td>path</td>
<td>Path where to store the data</td>
</tr>
<tr>
<td>input</td>
<td>Add or replacement</td>
</tr>
</tbody>
</table>

Details

The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server. This method returns `self` invisibly, thus making it possible to chain together multiple method calls.

Value

A list with two items

- info Additional info
- success A boolean, indicating if the command was completed successfully

Examples

```r
## Not run:
put(Session, "test", "<xml>Create test</xml>")
```

```r
## End(Not run)
```
Description

Store or replace a binary resource in the opened database.

Usage

putBinary(session, path, input)

Arguments

- **session**: BasexClient instance-ID
- **path**: Path where to store the data
- **input**: Additional input, may be empty

Details

Use the database-command `retrieve` to retrieve the resource. The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server. This method returns `self` invisibly, thus making it possible to chain together multiple method calls.

Value

A list with two items

- **info**: Additional info
- **success**: A boolean, indicating if the command was completed successfully

Examples

```r
## Not run:
Execute(Session, "DROP DB BinBase")
testBin <- Execute(Session, "Check BinBase")
bais <- raw()
for (b in 252:255) bais <- c(bais, c(b)) %>% as.raw()
test <- putBinary(Session, "test.bin", bais)
print(test$success)
baos <- Execute(Session, "BINARY GET test.bin")
print(bais)
print(baos$result)
## End(Not run)
```
Query

Description

Creates a new query instance and returns it’s id.

Usage

Query(session, query_string)

Arguments

- session: BasexClient instance-ID
- query_string: query string

Details

If paste0() is used to create a multi-line statement, the lines must be separated by a space or a newline \n-character.

Value

Query_ID

Examples

```r
## Not run:
query_txt <- "for $i in 1 to 2 return <xml>Text { $i }</xml>"
query_obj <- Query(Session, query_txt)
print(Execute(query_obj))
## End(Not run)
```

QueryClass

Description

The client can be used in ‘standard’ mode and in ‘query’ mode. Query mode is used to define queries, binding variables and for iterative evaluation.
Methods

Public methods:

- QueryClass$new()
- QueryClass$ExecuteQuery()
- QueryClass$Bind()
- QueryClass$Context()
- QueryClass$Full()
- QueryClass$More()
- QueryClass$Next()
- QueryClass$Info()
- QueryClass$Options()
- QueryClass$Updating()
- QueryClass$Close()
- QueryClass$clone()

Method new(): Initialize a new instance from QueryClass

Usage:
QueryClass$new(query, Parent)

Arguments:
query Query-string
Parent The 'Parent' for this QueryClass-instance

Details: QueryClass-instances can only be created by calling the 'Query'-method from the 'BasexClient'-class.

Method ExecuteQuery(): Executes a query.

Usage:
QueryClass$ExecuteQuery()

Method Bind(): Binds a value to a variable.

Usage:
QueryClass$Bind(...)

Arguments:
... Binding Information
query_obj QueryClass instance-ID

Details: When using the primitive functions, this function can be chained.

Method Context(): Binds a value to the context. The type will be ignored if the string is empty.

Usage:
QueryClass$Context(value, type)

Arguments:
value Value that should be bound to the context
type The type will be ignored when the string is empty
Details: When using the primitive functions, this function can be chained.

**Method Full()**: Executes a query and returns a vector with all resulting items as strings, prefixed by the 'XDM' (Xpath Data Model) Meta Data <https://www.xdm.org/>.

*Usage:*

QueryClass$Full()

**Method More()**: Indicates if there are any other results in the query-result.

*Usage:*

QueryClass$More()

**Method Next()**: Returns the next result when iterating over a query

*Usage:*

QueryClass$Next()

**Method Info()**: Returns a string with query compilation and profiling info.

*Usage:*

QueryClass$Info()

**Method Options()**: Returns a string with all query serialization parameters, which can e.g. be assigned to the serializer option.

*Usage:*

QueryClass$Options()

**Method Updating()**: Check if the query contains updating expressions.

*Usage:*

QueryClass$Updating()

**Method Close()**: Closes and unregisters the query with the specified ID

*Usage:*

QueryClass$Close()

Details: When using the primitive functions, this function can be chained.

**Method clone()**: The objects of this class are cloneable with this method.

*Usage:*

QueryClass$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.
Description

'BaseX' is a robust, high-performance XML database engine and a highly compliant XQuery 3.1 processor with full support of the W3C Update and Full Text extensions.

The client can be used in ‘standard’ mode and in ‘query’ mode. Standard Mode is used for connecting to a server and sending commands.

Details

'RBaseX' was developed using R6. For most of the public methods in the R6-classes, wrapper-functions are created. The differences in performance between R6-methods and wrapper-functions are minimal and slightly in advantage of the R6-version.

It is easy to use the R6-calls instead of the wrapper-functions. The only important difference is that in order to execute a query, you have to call ExecuteQuery() on a queryObject.

Methods

Public methods:

- `BasexClient$new()`
- `BasexClient$Command()`
- `BasexClient$Execute()`
- `BasexClient$Query()`
- `BasexClient$Create()`
- `BasexClient$Add()`
- `BasexClient$put()`
- `BasexClient$Replace()`
- `BasexClient$putBinary()`
- `BasexClient$Store()`
- `BasexClient$set_intercept()`
- `BasexClient$restore_intercept()`
- `BasexClient$get_intercept()`
- `BasexClient$get_socket()`
- `BasexClient$set_success()`
- `BasexClient$get_success()`
- `BasexClient$clone()`

Method `new()`: Initialize a new client-session

Usage:

`BasexClient$new(host, port = 1984L, username, password)`

Arguments:
Method Command(): Execute a command

Usage:
BasexClient$Command(command)

Arguments:
cmd Command

Details: For a list of database commands see https://docs.basex.org/wiki/Commands

Method Execute(): Execute a command

Usage:
BasexClient$Execute(command)

Arguments:
cmd Command

Details: For a list of database commands see https://docs.basex.org/wiki/Commands. This function is replaced by 'Command' and is obsolete.

Method Query(): Create a new query-object

Usage:
BasexClient$Query(query_string)

Arguments:
query_string Query-string

Details: A query-object has two fields. 'queryObject' is an ID for the new created 'QueryClass'-instance. 'success' holds the status from the last executed operation on the queryObject.

Returns: ID for the created query-object

Method Create(): Create a new database

Usage:
BasexClient$Create(name, input)

Arguments:
name Name
input Initial content, Optional

Details: Initial content can be offered as string, URL or file.

Method Add(): Add a new resource at the specified path

Usage:
BasexClient$Add(path, input)

Arguments:
path Path
input File, directory or XML-string

Method put(): Add or replace resource, addressed by path
Usage:
BaseXClient$put(path, input)

Arguments:
path  Path
input  File, directory or XML-string

Method Replace(): Replace resource, addressed by path. This function is deprecated and has been replaced by /put/.

Usage:
BaseXClient$Replace(path, input)

Arguments:
path  Path
input  File, directory or XML-string

Method putBinary(): Store binary content

Usage:
BaseXClient$putBinary(path, input)

Arguments:
path  Path
input  File, directory or XML-string

Details: Binary content can be retrieved by executing a retrieve-command

Method Store(): Store binary content

Usage:
BaseXClient$Store(path, input)

Arguments:
path  Path
input  File, directory or XML-string

Details: Binary content can be retrieved by executing a retrieve-command. This function is deprecated and has been replaced by /putBinary/.

Method set_intercept(): Toggles between using the 'success'-field, returned by the Execute-command or using regular error-handling (try-catch).

Usage:
BaseXClient$set_intercept(Intercept)

Arguments:
Intercept  Boolean

Method restore_intercept(): Restore the Intercept Toggles to the original value

Usage:
BaseXClient$restore_intercept()

Method get_intercept(): Get current Intercept
Usage:
BasexClient$get_intercept()

Method get_socket(): Get the socket-ID

Usage:
BasexClient$get_socket()

Returns: Socket-ID.

Method set_success(): Set the status success-from the last operation on the socket

Usage:
BasexClient$set_success(Success)

Arguments:
Success Boolean

Details: This function is intended to be used by instances from the QueryClass

Method get_success(): Get the status success-from the last operation on the socket

Usage:
BasexClient$get_success()

Returns: Boolean.

Method clone(): The objects of this class are cloneable with this method.

Usage:
BasexClient$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

Examples

```r
## Not run:
Session <- BasexClient$new("localhost", 1984L, username = "<username>", password = "<password>")
Session$Execute("Check test")
Session$Execute("delete /")
# Add resource
Session$Add("test.xml", "<root/>")

# Bindings ----
query_txt <- "declare variable $name external; for $i in 1 to 3 return element { $name } { $i }"
query_obj <- Session$Query(query_txt)
query_obj$QueryObject$Bind("$name", "number")
print(query_obj$QueryObject$ExecuteQuery())

## End(Not run)
```
Replace

Description

Replaces a resource with the specified input.

Usage

Replace(session, path, input)

Arguments

- `session`: BasexClient instance-ID
- `path`: Path where to store the data
- `input`: Replacement

Details

The 'Replace' command is deprecated and has been renamed to 'Put'. 'Replace' is being kept as convenience.

The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server. This method returns `self` invisibly, thus making it possible to chain together multiple method calls.

Value

A list with two items

- `info`: Additional info
- `success`: A boolean, indicating if the command was completed successfully

Examples

```r
## Not run:
Replace(Session, "test", "<xml>Create test</xml>")

## End(Not run)
```
Restore Intercept

Description

Restore Intercept to original new value

Usage

    RestoreIntercept(session)

Arguments

    session  BasexClient instance-ID

Details

    This method returns `self` invisibly, thus making it possible to chain together multiple method calls.

result2frame

Description

Converts the query-result to a frame. The query-result is either a list (sequence) or an array. If it is a list, 'cols' is needed to determine the number of columns.

Usage

    result2frame(...)  

Arguments

    ...  Query-result

Value

    Return result from query as dataframe
Description

Converts the query-result to a tibble. The query-result is either a list (sequence) or an array. If it is a list, ’cols’ is needed to determine the number of columns.

Usage

result2tibble(...)

Arguments

... Query-result

Value

Return result from query as tibble

---

SetIntercept SetIntercept

Description

Assign a new value to session$Intercept

Usage

SetIntercept(session, intercept)

Arguments

session BasexClient instance-ID
intercept New Intercept value

Details

This method returns self invisibly, thus making it possible to chain together multiple method calls.

Examples

```r
## Not run:
SetIntercept(TRUE)

## End(Not run)
```
**SetSuccess**

**Description**

Assign a new value to session$Success

**Usage**

SetSuccess(session, success)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>BasexClient instance-ID</td>
</tr>
<tr>
<td>success</td>
<td>Success-indicator for the last operation on the socket</td>
</tr>
</tbody>
</table>

**Examples**

```r
## Not run:
SetSuccess(TRUE)

## End(Not run)
```

---

**SocketClass**

**Description**

All methods that are used by BasexClient and QueryClass

**Methods**

**Public methods:**

- `SocketClass$new()`
- `SocketClass$finalize()`
- `SocketClass$handShake()`
- `SocketClass$write_Byte()`
- `SocketClass$clone()`

**Method** `new()`: Initialize a new socket

**Usage:**

`SocketClass$new(host, port = 1984L, username, password)`

**Arguments:**
host, port, username, password  Host-information and credentials

**Method finalize():** When releasing the session-object, close the socketConnection

*Usage:*
```
SocketClass$finalize()
```

**Method handShake():** Send input to the socket and return the response

*Usage:*
```
SocketClass$handShake(input)
```

*Arguments:*
- input  Input

*Details:* Input is a raw vector, built up by converting all input to raw and concatenating the results

**Method write_Byte():** Write 1 byte to the socket

*Usage:*
```
SocketClass$write_Byte(Byte)
```

*Arguments:*
- Byte  A vector length 1

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*
```
SocketClass$clone(deep = FALSE)
```

*Arguments:*
- deep  Whether to make a deep clone.

---

### Store

**Description**
Stores a binary resource in the opened database.

**Usage**
```
Store(session, path, input)
```

**Arguments**
- **session**  BasexClient instance-ID
- **path**  Path where to store the data
- **input**  Additional input, may be empty
Details

The 'Store' command is deprecated and has been renamed to 'putBinary'. 'Store' is being kept as convenience.

Use the database-command retrieve to retrieve the resource. The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server. This method returns self invisibly, thus making it possible to chain together multiple method calls.

Value

A list with two items

- info Additional info
- success A boolean, indicating if the command was completed successfully

Examples

```r
## Not run:
Execute(Session, "DROP DB BinBase")
testBin <- Execute(Session, "Check BinBase")
bais <- raw()
for (b in 252:255) bais <- c(bais, c(b)) %>% as.raw()
test <- Store(Session, "test.bin", bais)
print(test$success)
baos <- Execute(Session, "binary get test.bin")
print(bais)
print(baos$result)
## End(Not run)
```
Value

This function returns a list with the following items:

- result Result
- success A boolean, indicating if the command was completed successfully
Index

Add, 2
BasexClient (RBaseX), 20
Bind, 3
Close, 4
Command, 5
Context, 6
Create, 7
Execute, 8
Full, 9
GetIntercept, 10
GetSuccess, 10
Info, 11
input_to_raw, 11
More, 12
NewBasexClient, 13
Next, 13
Options, 14
put, 15
putBinary, 16
Query, 17
QueryClass, 17
RBaseX, 20
Replace, 24
RestoreIntercept, 25
result2frame, 25
result2tibble, 26
SetIntercept, 26
SetSuccess, 27
SocketClass, 27
Store, 28
Updating, 29