

# Package ‘RBaseX’

March 10, 2022

**Type** Package

**Title** 'BaseX' Client

**Version** 1.1.1

**Date** 2022-03-10

**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** 7.1.2

**Imports** R6, RCurl, pingr, rex, httr, 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.

**Repository** CRAN

**NeedsCompilation** no

**Date/Publication** 2022-03-10 15:10:12 UTC

## R topics documented:

Add	2
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
Query . . . . .	15
QueryClass . . . . .	16
RBaseX . . . . .	18
Replace . . . . .	21
RestoreIntercept . . . . .	22
result2frame . . . . .	22
result2tibble . . . . .	23
SetIntercept . . . . .	23
SetSuccess . . . . .	24
SocketClass . . . . .	24
Store . . . . .	26
Updating . . . . .	27
<b>Index</b>	<b>28</b>

Add

*Add***Description**

Adds a new resource to the opened database.

**Usage**

```
Add(session, path, input)
```

**Arguments**

session	BasexClient instance-ID
path	Path
input	Additional input (optional)

**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 successful

**Examples**

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

## End(Not run)
```

---

Bind

*Bind*

---

**Description**

Binds a value to a variable.

**Usage**

```
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 successful

**Examples**

```

## 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('TestDB/Books')/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)

```

---

Close

*Close*


---

**Description**

Closes and unregisters the query with the specified ID

**Usage**

```
Close(query_obj)
```

**Arguments**

```
query_obj      QueryClass instance-ID
```

**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 successful

---

Command	<i>Command</i>
---------	----------------

---

**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>

'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 successful

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

**Examples**

```
## 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

*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**

```
## 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"
```

```
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

session	BasexClient instance-ID
name	Database name
input	Additional input, may be empty

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

**Examples**

```
## 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)
```

---

 Execute
*Execute***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 obsolete 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 Additional 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**

```
## Not run:
Session <- NewBasexClient(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)
```

Full

*Title Full***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**

```
## 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	<i>GetIntercept</i>
--------------	---------------------

---

**Description**

Current value for session\$Intercept

**Usage**

GetIntercept(session)

**Arguments**

session	BasexClient instance-ID
---------	-------------------------

**Value**

Current value

---

GetSuccess	<i>GetSuccess</i>
------------	-------------------

---

**Description**

Current value from session\$Success

**Usage**

GetSuccess(session)

**Arguments**

session	BasexClient instance-ID
---------	-------------------------

**Value**

Current value

---

 Info
*Info***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 successfull

---

 input\_to\_raw
*input\_to\_raw***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

---

More

*More*

---

### 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	<i>Title</i>
----------------	--------------

---

**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**

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

---

Next	<i>Next</i>
------	-------------

---

**Description**

Returns the next result when iterating over a query

**Usage**

```
Next(query_obj)
```

**Arguments**

query\_obj      QueryClass instance-ID

**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)
```

---

Options

*Options*

---

**Description**

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 successful

---

Query

*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**

```
## 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

*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 'BaseClient'-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.

---

RBaseX

*RBaseX*


---

## 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$Replace()`
- `BasexClient$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:*

host, port, username, password Host-information and user-credentials

**Method** `Command()`: Execute a command

*Usage:*

BasexClient\$Command(command)

*Arguments:*

command 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:*

command 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 Replace():** Replace resource, addressed by path

*Usage:*

BasexClient\$Replace(path, input)

*Arguments:*

path Path  
input File, directory or XML-string

**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

**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:*

`deep` Whether to make a deep clone.

## Examples

```
## 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

*Replace*

---

## Description

Replaces a resource with the specified input.

## Usage

```
Replace(session, path, input)
```

## Arguments

<code>session</code>	BasexClient instance-ID
<code>path</code>	Path where to store the data
<code>input</code>	Replacement

## 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 successfull

**Examples**

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

## End(Not run)
```

---

RestoreIntercept	<i>RestoreIntercept</i>
------------------	-------------------------

---

**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	<i>result2frame</i>
--------------	---------------------

---

**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

---

result2tibble	<i>result2tibble</i>
---------------	----------------------

---

**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	<i>SetIntercept</i>
--------------	---------------------

---

**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**

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

## End(Not run)
```

---

SetSuccess

*SetSuccess*


---

**Description**

Assign a new value to session\$Success

**Usage**

```
SetSuccess(session, success)
```

**Arguments**

session	BasexClient instance-ID
success	Success-indicator for the last operation on the socket

**Examples**

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

## End(Not run)
```

---

SocketClass

*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

*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

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 successful

### Examples

```
## 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$success)
baos <- Execute(Session, "retrieve test.bin")
print(baos)
print(baos$result)

## End(Not run)
```

---

Updating

*Updating*

---

**Description**

Check if the query contains updating expressions.

**Usage**

Updating(query\_obj)

**Arguments**

query\_obj      Query instance-ID

**Details**

Returns *TRUE* if the query contains updating expressions; *FALSE* otherwise.

**Value**

This function returns a list with the following items:

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

# Index

Add, [2](#)

BasexClient (RBaseX), [18](#)

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](#)

Query, [15](#)

QueryClass, [16](#)

RBaseX, [18](#)

Replace, [21](#)

RestoreIntercept, [22](#)

result2frame, [22](#)

result2tibble, [23](#)

SetIntercept, [23](#)

SetSuccess, [24](#)

SocketClass, [24](#)

Store, [26](#)

Updating, [27](#)