Package ‘RBaseX’

May 1, 2020

Type Package
Title 'BaseX' Client
Version 0.3.0
Date 2020-05-01
Description 'BaseX' <http://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
LazyData true
RoxygenNote 7.1.0
Imports R6, RCurl, dplyr, openssl, stringr, magrittr, tibble, data.table
Suggests testthat
Author Ben Engbers [aut, cre]
Maintainer Ben Engbers <Ben.Engbers@Be-Logical.nl>
URL https://github.com/BenEngbers/RBaseX
SystemRequirements Needs a running BaseX-instance.
Repository CRAN
NeedsCompilation no
Date/Publication 2020-05-01 16:50:03 UTC

R topics documented:

Add ................................................................. 2
Bind .............................................................. 3
Close ......................................................... 4
Context ...................................................... 5
Create ....................................................... 6
Add

Description

Adds a new resource to the opened database.

Usage

Add(session, path, input)

Arguments

<table>
<thead>
<tr>
<th>session</th>
<th>BasexClient instance-ID</th>
</tr>
</thead>
<tbody>
<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 length-1 character vector which describes an element, a file-descriptor, an URL or a stream. 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.
**Bind**

**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**

`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 [http://docs.basex.org/wiki/Java_Bindings#Data_Types](http://docs.basex.org/wiki/Java_Bindings#Data_Types)

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

## Not run:
```r
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)
```

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.
Description

Binds a value to the context. The type will be ignored if the string is empty.

Usage

Context(query_obj, value, type)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_obj</td>
<td>QueryClass instance-ID</td>
</tr>
<tr>
<td>value</td>
<td>Value that should be bound to the context</td>
</tr>
<tr>
<td>type</td>
<td>The type will be ignored when the string is empty</td>
</tr>
</tbody>
</table>

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"

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

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

Initial content can be offered as string, URL or file. '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 successfully

Examples

```r
## 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

For a list of database commands see [http://docs.basex.org/wiki/Commands](http://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 Execute() 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
- success A boolean, indicating if the command was completed successfully

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

Examples

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

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)
```
Description
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/> 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 "0d" "TestDB/Test.xml <Line_1 line="1">Content 1</Line_1>"
## "0d" "TestDB/Test.xml <Line_2 line="2">Content 2</Line_2>"

## End(Not run)

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

### 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.
**Description**

Convert *input* to a length-1 character vector.

**Usage**

```r
input_to_raw(input, addZero = FALSE)
```

**Arguments**

- `input`: Character vector length 1
- `addZero`: If TRUE, add a zero-byte (0x00) to the raw-vector

**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**

```r
More(query_obj)
```

**Arguments**

- `query_obj`: QueryClass instance-ID

**Value**

Boolean
NewBasexClient

Examples

```r
## Not run:
query_iterate <- Query(Session, "collection('TestDB/Test.xml')")
while (More(query_iterate)) {
  iterResult <- c(iterResult, Next(query_iterate))
}
print(query_iterate)
## Return "0d" "<Line_1 line="1">Content 1</Line_1>"
  "0d" "<Line_2 line="2">Content 2</Line_2>"

## End(Not run)
```

<table>
<thead>
<tr>
<th>NewBasexClient</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Create a BaseX-client</td>
<td></td>
</tr>
</tbody>
</table>

Usage

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

Arguments

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

Details

This creates a BaseX-client that listens to port 1984 on localhost. Username and password should be changed after the installation of 'BaseX'.

Value

BaseClient-instance

Examples

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

## End(Not run)
```
Description

Returns the next result when iterating over a query

Usage

Next(query_obj)

Arguments

query_obj QueryClass instance-ID

Examples

## Not run:
query_iterate <- Query(Session, "collection('TestDB/Test.xml')")
while (More(query_iterate)) {
  iterResult <- c(iterResult, Next(query_iterate))
}

print(query_iterate)
## Return "0d" "<Line_1 line="1">Content 1</Line_1>"
  "0d" "<Line_2 line="2">Content 2</Line_2>"

## End(Not run)

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 http://docs.basex.org/wiki/Java_Bindings#Data_Types

---

## Query

### Description

Creates a new query instance and returns its id.

### Usage

```r
Query(session, query_string)
```

### Arguments

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

### 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$new()
• QueryClass$Bind()
• QueryClass$Context()
• QueryClass$Close()
• QueryClass$ExecuteQuery()
• QueryClass$Info()
• QueryClass$Options()
• QueryClass$Updating()
• QueryClass$More()
• QueryClass$Next()
• QueryClass$Full()
• QueryClass$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
sock Session-socket
Intercept Pointer to the Intercept-method from the Session-object

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

**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** 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** ExecuteQuery(): Executes a query.

*Usage:*

QueryClass$ExecuteQuery()

**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** 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** 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** 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$new()`
- `BasexClient$Execute()`
- `BasexClient$Query()`
- `BasexClient$Add()`
- `BasexClient$Create()`
- `BasexClient$Replace()`
- `BasexClient$Store()`
- `BasexClient$set_intercept()`
- `BasexClient$restore_intercept()`
- `BasexClient$get_intercept()`
- `BasexClient$get_socket()`
- `BasexClient$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 `Execute()`: Execute a command
Usage:
BaseXClient$Execute(command)

Arguments:
command Command

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

Method Query(): Create a new query-object

Usage:
BaseXClient$Query(query)

Arguments:
query 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 Add(): Add a new resource at the specified path

Usage:
BaseXClient$Add(path, input)

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

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 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
*Details: sgfdssfdsh*

**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

## Not run:
```r
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

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
## 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.

## 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

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
```r
result2tibble(...)```

Arguments

...  Query-result

Value
Return result from query as tibble

SetIntercept

Description
Assign a new value to session$Intercept

Usage
```r
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

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

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

Method new(): Initialize a new socket

Methods

### Public methods:
- SocketClass$new()
- SocketClass$finalize()
- SocketClass$bool_test_sock()
- SocketClass$void_send()
- SocketClass$str_receive()
- SocketClass$write_Byte()
- SocketClass$read_Byte()
- SocketClass$get_socket()
- SocketClass$clone()
SocketClass

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 bool_test_sock(): Return a boolean that indicates the result from the last action on the socket

Usage:
SocketClass$bool_test_sock()

Arguments:
socket  Socket-ID

Method void_send(): Send input to the socket

Usage:
SocketClass$void_send(input)

Arguments:
input  Input

Details: Input is either a string or data that is read from a stream

Method str_receive(): Read a string from a stream

Usage:
SocketClass$str_receive(bin = FALSE)

Arguments:
bin  Logical; TRUE when str_receive has to retrieve binary data

Method write_Byte(): Write 1 byte to the socket

Usage:
SocketClass$write_Byte(Byte)

Arguments:
Byte  A vector length 1

Method read_Byte(): Read 1 byte to the socket

Usage:
SocketClass$read_Byte()

Method get_socket(): Get socket-ID

Usage:
SocketClass$get_socket()
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.

---

**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. 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$success)
baos <- Execute(Session, "retrieve test.bin")
print(bais)
print(baos$result)
## End(Not run)
```
**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**

Boolean
Index

Add, 2
BaseClient (RBaseX), 16
Bind, 3
Close, 4
Context, 5
Create, 6
Execute, 7
Full, 8
GetIntercept, 8
GetSuccess, 9
Info, 9
input_to_raw, 10
More, 10
NewBasextClient, 11
Next, 12
Options, 12
Query, 13
QueryClass, 13
RBaseX, 16
Replace, 19
RestoreIntercept, 20
result2frame, 20
result2tibble, 21
SetIntercept, 21
SetSuccess, 22
SocketClass, 22
Store, 24
Updating, 25