Package ‘RBaseX’

March 11, 2020

Type Package
Title 'BaseX' Client
Version 0.2.4
Date 2020-03-11
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 7.0.2
Imports R6, RCurl, dplyr, openssl, stringr, magrittr, tibble
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 2020-03-11 09:40:02 UTC

R topics documented:

Add ................................................................. 2
Bind ............................................................... 3
Close .............................................................. 4
Context .......................................................... 4
Create ............................................................ 5
Execute ........................................................... 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 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.
**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.

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)

**Examples**

```r
## Not run:
query_txt <- "declare variable $name external; for $i in 1 to 3 return element { $name } { $i }
query_obj <- Query(Session, query_txt)
Bind(query_obj, "$name", "number")
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

Description

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

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.
Examples

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

ctx_query_txt <- "for $t in parse-xml(.)//text() return string-length($t)"
Context(ctx_query_txt, ctx_txt)
print(Execute(ctx_query)

## End(Not run)
```

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.

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

```r
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 Additional 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)
```

---

### Full

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

### 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 seaparated by a '|'.

### Usage

```r
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 "0d" "0d" "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)
```
### GetIntercept

<table>
<thead>
<tr>
<th>Description</th>
<th>Current value for session$Intercept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>GetIntercept(session)</td>
</tr>
<tr>
<td>Arguments</td>
<td>session</td>
</tr>
<tr>
<td>Value</td>
<td>Current value</td>
</tr>
</tbody>
</table>

### GetSuccess

<table>
<thead>
<tr>
<th>Description</th>
<th>Current value from session$Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>GetSuccess(session)</td>
</tr>
<tr>
<td>Arguments</td>
<td>session</td>
</tr>
<tr>
<td>Value</td>
<td>Current value</td>
</tr>
</tbody>
</table>
**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.

---

**input_to_raw**

**Description**

Convert input to a length-1 character vector.

**Usage**

`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
## More

### 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

### 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)

---

## NewBasexClient

### Description
Create a BaseX-client

### Usage

```r
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 that 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)
```

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_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>"
```
<table>
<thead>
<tr>
<th><strong>Options</strong></th>
<th><strong>Options</strong></th>
</tr>
</thead>
</table>

**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://docsibase.org/wiki/Java_BINDings#Data_Types

<table>
<thead>
<tr>
<th><strong>Query</strong></th>
<th><strong>Query</strong></th>
</tr>
</thead>
</table>

**Description**

Creates a new query instance and returns its id.

**Usage**

```
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:*

```java
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:*

```java
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:*

```java
QueryClass$Close()```

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

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

*Usage:*

```java
QueryClass$ExecuteQuery()```

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

*Usage:*

```java
QueryClass$Info()```

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

*Usage:*

```java
QueryClass$Options()```

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

*Usage:*

```java
QueryClass$Updating()```

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

*Usage:*

```java
QueryClass$More()```

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

*Usage:*

```java
```
QueryClass$Next()

**Method** Full(): Executes a query and returns a vector with all resulting items as strings, pre-fixed 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:*
depth 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$get_socket()`
• `BasexClient$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** `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](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:*
n 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: sgfdsffdsh

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:*

```r
BasexClient$get_success()
```

*Returns:* Boolean,

**Method** `clone()`:

The objects of this class are cloneable with this method.

*Usage:*

```r
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**

Replaces a resource with the specified input.

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>session</th>
<th>BasexClient instance-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path where to store the data</td>
</tr>
<tr>
<td>input</td>
<td>Replacement</td>
</tr>
</tbody>
</table>
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)
```

### RestoreIntercept

**Description**
Restore Intercept to original new value

**Usage**
```
RestoreIntercept(session)
```

**Arguments**
- `session` BasexClient instance-ID

---

### result2frame

**Description**
Converts the query-result to a frame. The query-result is a list. `cols` is needed to determine the number of columns.

**Usage**
```
result2frame(input, cols)
```

**Arguments**
- `input` Query-result
- `cols` Number of columns

**Value**
Return result from query as dataframe
**result2matrix**

**Description**

Converts the query-result to a matrix. The query-result is a list. 'cols' is needed to determine the number of columns.

**Usage**

`result2matrix(input, cols)`

**Arguments**

- **input**: Query-result
- **cols**: Number of columns

**Value**

Return result from query as matrix

---

**result2tibble**

**Description**

Converts the query-result to a tibble. The query-result is a list. 'cols' is needed to determine the number of columns.

**Usage**

`result2tibble(input, cols)`

**Arguments**

- **input**: Query-result
- **cols**: Number of columns

**Value**

Return result from query as tibble
SetIntercept

Description
Assign a new value to session$Intercept

Usage
SetIntercept(session, intercept)

Arguments
- session: BasexClient instance-ID
- intercept: New Intercept value

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
## Not run:
SetSuccess(TRUE)
## End(Not run)
Description

All methods that are used by BasexClient and QueryClass

Methods

**Public methods:**
- `SocketClass$new()`
- `SocketClass$finalize()`
- `SocketClass$bool_test_sock()`
- `SocketClass$void_send()`
- `SocketClass$str_receive()`
- `SocketClass$get_socket()`
- `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** `bool_test_sock()`: Return a boolean that indicates the result from the last action on the socket

*Usage:*
`SocketClass$bool_test_sock(socket)`

*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(input, output, bin = FALSE)

Arguments:
input, output Input- and output-stream
bin Boolean; TRUE when str_receive has to retrieve binary data

Details: This method is not intened to be called direct

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

<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>Additional input, may be empty</td>
</tr>
</tbody>
</table>

Details
Use the database-command retrieve to retrieve the resource.

Value
A list with two items
- info Additional info
- success A boolean, indicating if the command was completed successful
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)
```

### 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
BaseXClient (RBaseX), 15
Bind, 3
Close, 4
Context, 4
Create, 5
Execute, 6
Full, 7
GetIntercept, 8
GetSuccess, 8
Info, 9
input_to_raw, 9
More, 10
NewBaseXClient, 10
Next, 11
Options, 12
Query, 12
QueryClass, 13
RBaseX, 15
Replace, 18
RestoreIntercept, 19
result2frame, 19
result2matrix, 20
result2tibble, 20
SetIntercept, 21
SetSuccess, 21
SocketClass, 22
Store, 23
Updating, 24