Intercept Database

Release 1.2

Jul 21, 2023

Contents:

1 Getting Started with InterceptDB **1**

2 The InterceptDB Config file **3**

3 Connection commands **5**
- 3.1 dbCreateConnection configName . . . 5
- 3.2 dbCreateConnection [ip, port, user, pw, db] . . . 5
- 3.3 dbIsConnected Connection . . . 6
- 3.4 dbPing connection . . . 6
- 3.5 connection dbAddErrorHandler code . . . 6
- 3.6 connection dbLoadSchema schemaName . . . 6

4 Building Queries **9**
- 4.1 dbPrepareQuery query . . . 9
- 4.2 dbPrepareQuery [query, bindValues] . . . 9
- 4.3 dbPrepareQueryConfig configName . . . 9
- 4.4 dbPrepareQueryConfig [configName, bindValues] . . . 10
- 4.5 query dbBindValue value . . . 10
- 4.6 query dbBindValueArray [value, value...] . . . 10
- 4.7 dbGetBoundValues query . . . 11

5 Executing Queries **13**
- 5.1 connection dbExecute query . . . 13
- 5.2 connection dbExecuteAsync query . . . 13

6 Getting results **15**
- 6.1 Getting result data . . . 15
- 6.2 Handling Async results . . . 16

7 Miscellaneous commands **19**
- 7.1 dbVersion . . . 19

8 Future commands that aren't yet implemented **21**
- 8.1 dbResultError result . . . 21
- 8.2 dbResultErrorNum result . . . 21
- 8.3 dbResultIsError result . . . 21
- 8.4 connection dbConnectionEnableThrow bool . . . 21

8.5 query dbBindNamedValue [name, value] 22

9 Possible Errors **23**
9.1 Invalid number of bind values 23
9.2 Unsupported bind value type 23

CHAPTER 1

Getting Started with InterceptDB

1. Set up your *config file* which is included in the InterceptDB download.
2. Read the command docs.

#TODO make this better

CHAPTER 2

The InterceptDB Config file

```
accounts:
 maindb: #production db, don't break things here!
  ip: 127.0.0.1
  username: root
  password: lulz
  database: production
  port: 3306 #optional
 testdb: #testserver
  ip: 127.0.0.2
  username: root
  password: lulz
  database: production
  port: 3306 #optional
  opt_compress: false #set the MYSQL_OPT_COMPRESS option
  #opt_read_timeout: 5 #set the MYSQL_OPT_READ_TIMEOUT option
  #opt_write_timeout: 5 #set the MYSQL_OPT_WRITE_TIMEOUT option
  #opt_multi_statement: false #set the MARIADB_OPT_MULTI_STATEMENTS and MARIADB_OPT_
↪MULTI_RESULTS option. I think this is broken, you can try it if you want though.

statements:
 insertStuff: INSERT INTO table (a,b,c) VALUES (?,?,?)
 deleteStuff: DELETE FROM table WHERE a=?
 longQuery: >
  SELECT stuff
  FROM table
  WHERE
  isThisALongQuery=1 AND
  queriesCanBeMultiline=1 AND
  thatsWhyILikeYAML=5;
 queryWithOptions:
  query: SELECT NOW(), tinyIntValue FROM MyTable;
  parseDateType: array #I want this specific statement to return DateTime values in␣
↪array format
  parseTinyintAsBool: true #I want this specific statement to return my tinyInt as a␣
↪boolean in dbResultTo(Parsed)Array
```

(continues on next page)

(continued from previous page)

```
global:
 enableDynamicQueries: true #Allow queries to be created from SQF, if false only␣
↪statements from config are allowed
 parseDateType: string #This is a enum, one of the below values is allowed
  #string: default. Return Date/DateTime as "2018-12-24 13:45:11"
  #stringMS: Return Date/DateTime as "2018-12-24 13:45:11.123"
  #array: Return Date/DateTime as [year,month,day,hour,minute,second,millisecond]␣
↪(yes both have time too, date will be 0 hours) in dbResultTo(Parsed)Array
  #timestamp: Return Date/DateTime as a timestamp as a number (this can incur␣
↪precision loss)
  #timestampString: Return Date/DateTime as a unix timestamp in a string
  #timestampStringMS: Return Date/DateTime as a millisecond unix timestamp in a string
 parseTinyintAsBool: false #returns tinyint as bool in dbResultTo(Parsed)Array
 DBNullEqualEmptyString: false #whether dbNull == "" returns true
 logging:
  directory: dbLog #logging directory, relative to arma directory, will be created if␣
↪it doesn't exist
  querylog: false #log all queries with timestamp
  threadlog: false #log threading activity (high bandwidth log)
 workerCount: 1 #workerCount, be careful with this

schemas:
 test: schema.sql #Filename relative to config.yaml to be used in dbLoadSchema
```

Config has to be in `Arma 3/@InterceptDB/config.yaml` Other subfolders or renaming the @InterceptDB folder doesn't work. Folder name is also case sensitive on linux.

The config is loaded at preInit. If anything on the config loading fails, a error will be printed to the RPT.

Per-statement options take precendence over global options

CHAPTER 3

Connection commands

3.1 dbCreateConnection configName

Creates a connection based on details in the *config file* in `accounts.<configName>`

Note: Connection is not established until the first query.

configName `<STRING>` - The config name of the connection

Attention: configName is case-sensitive

Returns: <DBConnection>

3.2 dbCreateConnection [ip, port, user, pw, db]

Creates a connection.

Note: Connection is not established until the first query.

ip `<STRING>` - the IP Address or Domain of the database server

port `<NUMBER>` - the port of the database server (usually 3306)

user `<STRING>` - the user to log in with

pw `<STRING>` - the password (duh)

db `<STRING>` - the database to use (Equal to *use* <*db*> SQL command)

Returns: <DBConnection>

3.3 dbIsConnected Connection

Returns whether the connection is currently connected to the database server.
Also checks if a worker thread is connected.

connection `<DBCONNECTION>` - A connection

Returns: `<BOOL>`

3.4 dbPing connection

Executes a `SELECT 1;` on the database server and returns true if it get's 1 back. Returns false on error.
Suspends in scheduled, freezes in unscheduled.
(Should this return the actual error string somehow?, Should this call error handlers?)

connection `<DBCONNECTION>` - A connection

Returns: `<BOOL>`

3.5 connection dbAddErrorHandler code

Registers a global error handler on the connection, if any query on the connection causes an error, that function will be called with `_this = [errorString, errorCode, query].`
There can be multiple error handlers, they will be executed from first to last added.
If one of the error handlers returns `true` the error will be considered handled and the other handlers won't be called.
If error handlers are present, errors won't be printed to RPT.
Example _this:

```
["Lost connection to MySQL server at 'reading authorization packet', system
error:  10061",2013,"testQuery5"]
["You have an error in your SQL syntax; check the manual that corresponds to
your MariaDB server version for the right syntax to use near 'testQuery5' at
line 1",1064,"testQuery5"]
["Unknown column 'none' in 'field list'",1054,"SELECT none"]
```

#TODO add the query config name to _this too.
Error codes are explained on *config file*

connection `<DBCONNECTION>` - A connection

code `<CODE>` - Script code.

Returns: `<NOTHING>`

3.6 connection dbLoadSchema schemaName

Executes a SQL file. Path is defined in config.

connection `<DBCONNECTION>` - A connection

schemaName <STRING> - schema name from config.

Attention: schemaName is case-sensitive

Returns: <NOTHING>

CHAPTER 4

Building Queries

4.1 dbPrepareQuery query

Prepares a query.

query `<STRING>` - The SQL Query String

Returns: `<QUERY>`

4.2 dbPrepareQuery [query, bindValues]

Prepares a query and directly binds some values to it.

query `<STRING>` - The SQL Query String

bindValues `<ARRAY>` - List of values to bind to `?` in the query string. See *dbBindValueArray* for more information.

Returns: `<QUERY>`

Example:

```
dbPrepareQuery ["SELECT ? FROM ? WHERE ?=?", ["data", "table", "value", 5]]
-> SELECT data FROM table WHERE value=5
```

4.3 dbPrepareQueryConfig configName

Prepares a query based on details in the *config file* in `statements.<configName>`

configName `<STRING>` - The config name of the query

Attention: configName is case-sensitive

Returns: `<QUERY>`

4.4 dbPrepareQueryConfig [configName, bindValues]

Prepares a query based on details in the *config file* in `statements.<configName>`

configName `<STRING>` - The config name of the query

bindValues `<ARRAY>` - List of values to bind to `?` in the query string (See above)

Attention: configName is case-sensitive

Returns: `<QUERY>`

4.5 query dbBindValue value

query `<QUERY>`

value `<STRING>` OR `<NUMBER>` OR `<BOOL>` OR `<ARRAY>` - Value to bind to the next unbound `?` in the query

Returns: `<NOTHING>`

Note: ARRAY values are automatically converted to string. Meaning `[1,2,3]` will get bound as `"[1,2,3]"`

Warning: This command modifies the value in `query`. If you want to keep the old query intact you need to *dbCopyQuery* first.

4.6 query dbBindValueArray [value, value...]

Binds multiple values to the next `?` in the query, in same order as the `?` occur in the query.

query `<QUERY>`

value `<STRING>` OR `<NUMBER>` OR `<BOOL>` OR `<ARRAY>` - Value to bind to the next unbound `?` in the query

Returns: `<NOTHING>`

Note: ARRAY values are automatically converted to string. Meaning `[1,2,3]` will get bound as `"[1,2,3]"`

Warning: This command modifies the value in *query*. If you want to keep the old query intact you need to *dbCopyQuery* first.

Example: `_query = dbPrepareQuery "SELECT ? FROM ? WHERE ?=?" _query dbBindValueArray ["data", "table", "value", 5] -> SELECT data FROM table WHERE value=5`

4.7 dbGetBoundValues query

Returns array of all values currently bound to this query

returns `<ARRAY>`

4.7.1 dbCopyQuery query

query: `<QUERY>` - the query object returned by dbPrepareQuery

Tip: There is also the short version `+ query` which copies just like with Arrays and Numbers.

Returns: `<NOTHING>`

Copies a query with all currently bound values.
Example:
```
_query = dbPrepareQuery "SELECT ? FROM ? WHERE ?=?"
_query dbBindValueArray ["data", "table"]
_query -> SELECT data FROM table WHERE ?=?
_copyOfQuery = dbCopyQuery _query;
_copyOfQuery -> SELECT data FROM table WHERE ?=?
_copyOfQuery dbBindValueArray ["value", 5]
_copyOfQuery -> SELECT data FROM table WHERE value=5
_query -> SELECT data FROM table WHERE ?=?
```

CHAPTER 5

Executing Queries

5.1 connection dbExecute query

This function behaves differently in scheduled and unscheduled.
Scheduled: Suspends the script like a sleep/waitUntil would do, and continues once result is ready.
Unscheduled: Freezes the game until the result is ready. (You probably want to use dbExecuteAsync)

connection `<DBConnection>` - The connection to execute the query on

query `<QUERY>` - the query object returned by dbPrepareQuery

Returns: `<RESULT>`

5.2 connection dbExecuteAsync query

This function executes the query in a seperate thread and returns a handle to the task.
You can bind callbacks to it, or wait on the task to finish (see below)

connection `<DBConnection>` - The connection to execute the query on

query `<QUERY>` - the query object returned by dbPrepareQuery

Returns: `<ASYNC_RESULT>` (See results: Handling Async results)

CHAPTER 6

Getting results

6.1 Getting result data

6.1.1 dbResultAffectedRows result

Returns number of affected rows. woah.

result: `<RESULT>` - The result

Returns: `<NUMBER>`

6.1.2 dbResultLastInsertId result

Returns last insert id. woah.

result `<RESULT>` - The result

Returns: `<NUMBER>`

6.1.3 dbResultToArray result

Turns the result set into an array of rows.

Like this [row1,row2,row3];

Each row being an array made up of the values in that returned row.

row1 = [value1, value2, value3]

values can be of type NUMBER, STRING, BOOL, DBNULL (null values from the database will be returned as *dbNull*)

result `<RESULT>` - The result

Returns: `<ARRAY>`

6.1.4 dbResultToParsedArray result

Like dbResultToArray. But tries to parse all string values from the database.
Turns "true" into true
Turns "[1,2,3,4]" into [1,2,3,4]
Turns "123" into 123
If string starts with [it get's put through parseSimpleArray
If string starts with t/f/T/F/number it get's wrapped in [] and put through parseSimpleArray
Anything else is invalid

result <RESULT> - The result

Returns: <ARRAY>

6.2 Handling Async results

6.2.1 result dbBindCallback [code, (arguments)]

Code will be called with _this = [<RESULT>, arguments]

result <ASYNC_RESULT> - Value returned by dbExecuteAsync

code <CODE> - Script to execute once the results are ready

arguments <ANY> - Arguments passed to the code.

Returns: <NOTHING>

Example:

```
_result dbBindCallback [{
    params ["_result", "_args"];
    //_Args=1
    DB_RES = [dbResultToArray _result, _args];
    systemChat "got result!";
}, 1];
```

6.2.2 dbWaitForResult result

Does exactly what you think it does. But also freezes the game even in scheduled! (to be changed in future updates)
Essentially converts a ASYNC_RESULT into a normal RESULT

result <ASYNC_RESULT> - Value returned by dbExecuteAsync

Returns: <RESULT>

Example: _result = dbWaitForResult _asyncResult;

6.2.3 dbNull

Returns a dbNull value (just like objNull or other Arma null values)

Returns: `<DBNULL>`

dbNull type can be configured in the *config file* to compare equal to empty string

DBNullEqualEmptyString set to `true`

```
dbNull == "" //true
"" == dbNull //true
isNull dbNull //true
```

DBNullEqualEmptyString set to `false`

```
dbNull == "" //false
"" == dbNull //false
isNull dbNull //true
```

CHAPTER 7

Miscellaneous commands

7.1 dbVersion

Returns: `<STRING>`

Returns current version of InterceptDB extension.

CHAPTER 8

Future commands that aren't yet implemented

8.1 dbResultError result

Returns error as string if an error occurred while querying. Returns nil if there is no error. (Should it return empty string instead?)

result `<RESULT>` - The result

Returns: `<STRING>`

8.2 dbResultErrorNum result

Returns error code if there is one. Returns 0 if there is none.

result `<RESULT>` - The result

Returns: `<NUMBER>`

8.3 dbResultIsError result

Checks if a error occured in the query.

result `<RESULT>` - The result

Returns: `<BOOL>`

8.4 connection dbConnectionEnableThrow bool

Makes dbExecuteQuery and dbWaitForResult throw SQF Exceptions that can be caught using https://community.bistudio.com/wiki/catch

connection `<DBCONNECTION>` - A connection

bool `<BOOL>` - throwing enabled or disabled

Returns: `<NOTHING>`

8.5 query dbBindNamedValue [name, value]

This command modifies the value in *query*. If you want to keep the old query intact you need to *dbCopyQuery* first.

query `<QUERY>`

name `<STRING>` - Name of the value to bind

value `<STRING>` OR `<NUMBER>` OR `<BOOL>` - Value to bind to the next unbound **<*name*>** in the query

Returns: `<NOTHING>`

Example: `SELECT <value> FROM <table>;`
dbBindNamedValue [“value”, “onions”];
dbBindNamedValue [“table”, “shoppinglist”];
-> `SELECT onions FROM shoppinglist`

Maybe other syntax would be better? `$name`? `:name` ?

`:name` seems to be standard elsewhere
https://www.php.net/manual/de/pdostatement.bindparam.php
https://www.javaworld.com/article/2077706/named-parameters-for-preparedstatement.html
https://docs.oracle.com/cd/B10501_01/appdev.920/a96584/oci05bnd.htm
https://www.sqlite.org/c3ref/bind_blob.html

CHAPTER 9

Possible Errors

Here are the possible errors that InterceptDB can throw.
Errors are printed to the logfile and can be caught using the *dbAddErrorHandler* eventhandler.

Database server errors for Mysql can be found here:
https://dev.mysql.com/doc/refman/8.0/en/server-error-reference.html
InterceptDB specific errors are listed below

9.1 Invalid number of bind values

The number of provided bindValues doesn't match the number of required bindValues, see *dbBindValueArray*

errorID 2

errorText "Invalid number of bind values. Expected {number} got {number}"

9.2 Unsupported bind value type

A unsupported value was bound as a value, see *dbBindValueArray*

errorID 3

errorText "Unsupported bind value type. Got {typeName} on index {index} with value {str bindValue}"

InterceptDB is a Intercept based SQL Database Plugin for Arma 3