AceQL HTTP 2.1 - API User Guide

Using the API

The AceQL HTTP API allows you to execute SQL queries and updates on remote databases using pure HTTP calls, with a fluent and simple to use REST like API.

The Server operation is described in Server Installation and Configuration Guide.

This document describes all AceQL URLs to use in your HTTP calls. It also contains easy to copy and paste examples in cURL.

Java, C#, Swift and Python SDK

AceQL HTTP includes a C#, a Java, a Python and a Swift SDK that wrap the API and eliminate the tedious works of handling communications errors and parsing JSON results:

- C# SDK User Guide
- Java SDK User Guide
- Python SDK User Guide
- Swift SDK User Guide

Reading this User Guide is NOT required if you intend to use a SDK.

Authentication & session creation

Authentication and session creation is done with login call, which:

- Allows you to specify the username and password to gain access to a remote AceQL Server.
- Allows you to specify the database to use during the session.
- Sends back a session ID that will be used in all subsequent calls for authentication.

AceQL Server responses

Responses are returned in easy to read and parse JSON format.

All responses contain the status key name, whose value is either:

- "OK" if the call was successful.
- "FAIL" if an error occurred.

JSON error format

In case of error, the returned JSON always contains:

```
{
  "status":"FAIL",
  "error_type":{error type numeric value},
  "error_message":"{error message returned by the server}"
  "http_status":{http status code numeric value}
}
```

In some cases,the full Java Exception stack trace is included in the JSON:

```
{
  "status":"FAIL",
  "error_type":{error type numeric value},
  "error_message":"{error message returned by the server}",
  "http_status":{http status code numeric value}
  "stack_trace":{remote Java stack trace in string format}
}
```

Where:

Key Name	Value
error_type	A numeric value between 0 and 4 describing the type of error that occurred. See possible values below.
error_message	The detailed error message.
http_status	The HTTP Response Status Code. See possible values below.
stack_trace	The Java Exception stack trace, if any.

The error_type key

The error_type key allows you to get the type of error, and where the error occurred:

error_type Value	Description
0	The error occurred locally on the client side. See http_status for more info. Typical cases: no Internet connection, proxy authentication required.
1	The error is due to a JDBC Exception. It was raised by the remote JDBC Driver and is rerouted by AceQL as is. The JDBC error message is accessible via the error_message key. Typical case: an error in the SQL statement. Examples: wrong table or column name.
2	The error was raised by the AceQL Server. This means that the AceQL Server expected a value or parameter that was not sent by the client side. Typical cases: misspelling in URL parameter, missing required request parameters, JDBC Connection expiration, etc. The detailed error message is accessible via the error_message key. See below for the most common AceQL Server errors messages.
3	The AceQL Server forbade the execution of the SQL statement for a security reason. For security reasons, the error_message key gives access to voluntarily vague details.
4	The AceQL Server is on failure and raised an unexpected Java Exception. The stack track is included and accessible via the stack_trace key.

Most common AceQL server messages

AceQL Server Error Messages (error_type=2)
AceQL main servlet not found in path
An error occurred during Blob download
An error occurred during Blob upload
Blob directory defined in `DatabaseConfigurator.getBlobDirectory()` does not exist
Connection is invalidated (probably expired)
Database does not exist
Invalid blob_id. Cannot be used to create a file
Invalid blob_id. No Blob corresponding to blob_id
Invalid or exipred Connection
Invalid session_id
Invalid username or password
No action found in request
Unable to get a Connection
Unknown SQL action or not supported by software

HTTP Status Codes

The HTTP Status Code is 200 (OK) on successful completion calls.

When an error occurs:

- If error_type is 0, the HTTP Status Code is returned by the client side and may take all possible values in a malformed HTTP call.
- If error_type is > 0, the HTTP Status Code can take one the following values returned by server side:

HTTP Status Code	Description
400 (BAD REQUEST)	Missing element in URL path. Missing request parameters. All JDBC errors raised by the remote JDBC Driver.
401 (UNAUTHORIZED)	Invalid username or password in `login`. Invalid `session_id`. The AceQL Server forbade the execution of the SQL statement for security reasons.
404 (NOT_FOUND)	BLOB directory does not exist on server. BLOB file not found on server.
500 (INTERNAL_SERVER_ERROR)	The AceQL Server is on failure and raised an unexpected Java Exception.

API reference guide

login

Allows you to create a new session, authenticate on remote AceQL server, and connect to a remote SQL database.

URL Format
`server/aceql/database/{database}/username/{username}/login

Note that we will use two shortcuts through this User Guide in order to simplify the URL format:

- **server** is the shortcut for the scheme, server name, and port of the URL to call. Possible values for server are: http://localhost:9090, https://www.acme.com, etc.
- **aceql** is the shortcut for the AceQL Server servlet path. aceql is also the default configuration value. Any other value is possible, see Server Installation and Configuration Guide.

URL parameter	Description
database	The remote database name
username	The authentication user name

Request parameter	Requested	Description
password	Yes	The authentication password

Notes about URL and request
Supported request methods: GET and POST. All URL parameters must be URL encoded. All Request parameters must be URL encoded and formatted in UTF-8.

Notes about credentials (username, password)
These are *not* the username/password of the remote JDBC Driver, but are the authentication information checked by remote AceQL server with `DatabaseConfigurator.login(username, password)` method. See Server Installation and Configuration Guide.

Server response to login call

If everything is OK:

```
{
   "status":"OK",
   "session_id":"session ID alphanumeric string",
   "connection_id":"connection ID number"
}
```

In case of error:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```

login call – cURL example

All the following examples use a MySQL database named `kawansoft_example`.

Connection to the kawansoft_example database with (MyUsername, MySecret) credentials:

```
$ curl \
http://localhost:9090/aceql/database/kawansoft_example/username/\
MyUsername/login?password=MySecret
```

The call will return a JSON stream with a unique session ID to reuse with all other API calls:

```
{
  "status":"OK",
  "session_id":"hli7ppunldk07mg8ae4dvv70kc"
  "connection_id":"461003207"
}
```

The connection_id URL parameter

The `connection_id` identifies the remote `java.sql.Connection`. It can be used to distinguish different `java.sql.Connection` on the same database.

`connection_id` URL parameters may be passed to each API. If `connection_id` is not expicitely passed as URL parameter, the server side will used the default and first `java.sql.Connection` created at login.

Connection info queries

Allow to get info on remote database connection.

URL Format

`server/aceql/session/{session_id}/connection/{connection_id}/get_auto_commit`
`server/aceql/session/{session_id}/connection/{connection_id}/is_read_only`
`server/aceql/session/{session_id}/connection/{connection_id}/get_holdability`
`server/aceql/session/{session_id}/connection/{connection_id}/get_transaction_isolation_level`

URL parameter	Description
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.

Request parameter	Requested	Description
None		

Server response to connection info query calls

If everything is OK:

```
{
  "status":"OK",
  "result":"result of API call"
}
```

In case of error:

```
{
  "status":"FAIL",
  "error_type":{error type numeric value},
  "error_message":"{error message returned by the server}",
  "http_status":{http status code numeric value}
}
```

Connection info query calls – cURL example

Query if remote connection is in auto commit mode :

```
$ curl \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
get_auto_commit
```

The call will return a JSON stream with the result:

```
{
    "status":"OK",
    "result":"true"
}
```

Connection modifiers

Allow to modify the remote Connection.

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/commit`
`server/aceql/session/{session_id}/connection/{connection_id}/rollback`
`server/aceql/session/{session_id}/connection/{connection_id}/set_auto_commit/{boolean}`
`server/aceql/session/{session_id}/connection/{connection_id}/set_holdability/{holdability}`
`server/aceql/session/{session_id}/connection/{connection_id}/set_read_only/{boolean}`
`server/aceql/session/{session_id}/connection/{connection_id}/set_transaction_isolation_level/{level}`

URL parameter	Description / value
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.
boolean	false true
holdability	close_cursors_at_commit hold_cursors_over_commit
level	read_committed read_uncommitted repeatable_read serializable

Request parameter	Requested	Description
None		

Server response to connection modifier calls

If everything is OK:

```
{
    "status":"OK",
}
```

In case of error:

```
{
  "status":"FAIL",
  "error_type":{errortype numeric value},
  "error_message":"{error message returned by the server}",
  "http_status":{httpstatus code numeric value}
}
```

Connection modifier calls – cURLexample

Set the remote connection to auto commit mode:

```
$ curl \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\

set_auto_commit/true
```

The call will return a JSON stream with the result:

```
{
    "status":"OK"
}
```

execute_update

Allows to update remote database with a DDL (Data Definition Language), DML (Data Modification Language), or DCL (Data Control Language) statement.

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/execute_update`

URL parameter	**Description**
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.

Request parameter	**Requested**	**Description**
sql	Yes	The SQL statement.
prepared_statement	No	true or false. Defaults to false. Says if the statement is to be executed as a prepared statement on remote server.
param_type_{i}	No	For prepared statements only. Allows to define the parameter type of parameter of i index. See values below.
param_value_{i}	No	For prepared statements only. Allows to define the parameter value of parameter of i index.

Prepared Statement - Parameter type values
BIGINT, BINARY, BIT, BLOB, CHAR, CHARACTER, CLOB, DATE, DECIMAL, DOUBLE_PRECISION, FLOAT, INTEGER, LONGVARBINARY, LONGVARCHAR, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, URL, VARBINARY, VARCHAR.

Notes
Only statements that are not prepared statements are supported with GET method. It is allowed to define a prepared statement without any parameter. All Request parameters must be URL encoded and formatted in UTF-8. `DATE`, `TIME` and `TIMESTAMP` values must be passed in the form the number of milliseconds since January 1, 1970, 00:00:00 GMT.

Server response to execute_update call

If everything is OK:

```
{
    "status":"OK",
    "row_count":{number of rows affected by update statement}
}
```

In case of error:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```

execute_update call – cURL examples

Let's calla simple statement to insert a row:

```
$ curl --data-urlencode \
"sql=insert into customer values (1, 'Sir', 'Doe', 'John', '1Madison Ave', 'New York', 'NY
10010', NULL)" \       http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
execute_update
```

The call will return:

```
{
    "status":"OK",
    "row_count":1
}
```

Let's update the customer row using a prepared statement :

```
$ curl --data "prepared_statement=true" \
--data"param_type_1=VARCHAR&param_value_1=Jim" \
--data"param_type_2=INTEGER&param_value_2=1" \
--data-urlencode"sql=update customer set fname=? where customer_id=?" \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/execute_update
```

The call will return:

```
{
    "status":"OK",
    "row_count":1
}
```

execute_query

Allows to send a query to the database with a statement or prepared statement.

The query result is returned in the form of a JSON stream.

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/execute_query`

URL parameter	Description
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.

Request parameter	Requested	Description
sql	Yes	The SQL statement.
prepared_statement	No	true or false. Defaults to false. Says if the statement is to be executed as a prepared statement on remote server.
gzip_result	No	true or false. Defaults to false. Says if the query result is returned compressed with the GZIP file format.
column_types	No	true or false. Defaults to false. Says if the column types must be included in the JSON stream.
pretty_printing	No	true or false. Defaults to false. Says if the JSON stream will be pretty printed.
param_type_{i}	No	For prepared statements only. Allows to define the parameter type of parameter of i index. See values below.
param_value_{i}	No	For prepared statements only. Allows to define the parameter value of parameter of i index.

Prepared Statement - Parameter type values
BIGINT, BINARY, BIT, BLOB, CHAR, CHARACTER, CLOB, DATE, DECIMAL, DOUBLE_PRECISION, FLOAT, INTEGER, LONGVARBINARY, LONGVARCHAR, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, URL, VARBINARY, VARCHAR.

Notes
Only statements that are not prepared statements are supported with GET method. It is allowed to define a prepared statement without any parameter. All Request parameters must be URL encoded and formatted in UTF-8. `DATE`, `TIME` and `TIMESTAMP` values must be passed in the form the number of milliseconds since January 1, 1970, 00:00:00 GMT.

Server response to execute_query call

If everything is OK:

```
{
   "status":"OK",
   "query_types":[
        "{column 1 type}",
      …
        "{column m type}",
```

```
   ],
   "query_rows":[
      {
         "row_1":[
            {
               "column 1 name":"{column 1 value}"
            },

            {
             ...
            },

            {
               "column m name":"{column m value}"
            },
         ],

    "row_{i}":[
      ...
    ]

         "row_{n}":[
            {
               "column 1 name":"{column 1 value}"
            },

            {
             ...
            },

            {
               "column m name":"{column m value}"
            },
         ]
      }
   ],
   "row_count":n
}
```

Where:

row_{i}	**Designates the row number.**
row_count	Designates the number of rows returned by the query.
m	The number of column per row.
n	The total number of rows.

In case of error:

```
{
  "status":"FAIL",
  "error_type":{errortype numeric value},
  "error_message":"{error message returned by the server}",
  "http_status":{httpstatus code numeric value}
}
```

execute_query call – cURL examples

Let's call a select on the previously inserted row. We ask for a pretty printing and to include the column types:

```
$ curl  --data"pretty_printing=true" --data "column_types=true" \
--data-urlencode \
"sql=select customer_id,customer_title, fname from customer" \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
execute_query
```

The call will return:

```
{
    "status":"OK",
    "query_types":[
        "INTEGER",
        "CHAR",
        "VARCHAR"
    ],
    "query_rows":[
        {
            "row_1":[
                {
                    "customer_id":1
                },
                {
                    "customer_title":"Sir "
                },
                {
                    "fname":"Doe"
                }
            ]
        }
    ],
    "row_count":1
}
```

Same call, but as a more secure prepared statement with a where condition, a prepared statement parameter and without pretty printing:

```
$ curl --data "prepared_statement=true" --data"column_types=true" \
--data"param_type_1=INTEGER&param_value_1=1" --data-urlencode \
"sql=select customer_id,customer_title, fname from customer where customer_id = ?"\
 http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70k/execute_query
```

This call will return same data as previous one, but in condensed format:

```
{"status":"OK","query_types":["INTEGER","CHAR","VARCHAR"],"query_rows":[{"row_1":
[{"customer_id":1},{"customer_title":"Sir"},
{"fname":"Doe"}]}],"row_count":1}
```

blob_upload

Allows to upload a BLOB on remote server.

Effective database update will be done in a following `execute_update` prepared statement call.

This is a Multipart POST request in UTF-8 format.

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/blob_upload`

URL parameter	**Description**
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.

Request parameter	**Requested**	**Description**
blob_id	Yes	The reference Id of the uploaded BLOB. This reference will be used in the following `execute_update` call that will do the database update. The value must be valid file name (without path separators) on the remote host OS. Example: my_blob.jpg
file	yes	This a `type=file` request parameter. Parameter name must be `file`.

Notes
This call supports POST method only. All Request parameters must be URL encoded and formatted in UTF-8.

The equivalent HTML form format is:

```
<form action="server/aceql/session/{session_id}/blob_upload" method="post"
enctype="multipart/form-data" acceptcharset="UTF-8">

<br>Blob Id: <input type="text" name="blob_id">
<br>File   : <input type="file" name="file">
<br>
<br><input type="submit" value="Upload BLOB!">

</form>
```

Server response to blob_upload call

If everything is OK:

```
{
    "status":"OK"
}
```

In case of error:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```

update_query call for effective database update

The `blob_upload` call must be followed by an update_query prepared statement that will update the database with the BLOB content.

A request parameter of type BLOB will contain the `blob_id` value :

So we will have the sequence:

1) BLOB upload:

```
$ curl -F "blob_id=koala.jpg" -F "file=@/home/mike/koala.jpg" \
http://server/aceql/session/{session_id}/blob_upload
```

2) Database update:

```
$ curl --data "prepared_statement=true" \
--data "param_type_1=INTEGER&param_value_1=1" \
--data "param_type_2=VARCHAR&param_value_2=Koala" \
--data "param_type_3=BLOB&param_value_3=koala.jpg" \
--data-urlencode "sql=insert into product_image values(?, ?, ?)" \
http://server/aceql/session/{session_id}/execute_update
```

Note that the param_type value is always BLOB for all databases engines.

Database engines specific BLOB types syntax is not supported.

For example, you cannot code for MySQL:

```
--data "param_type_3=LONGBLOB&param_value_3=koala.jpg"
```

nor for Microsoft SQL Server:

```
--data"param_type_3=VarBinary(max)&param_value_3=koala.jpg"
```

blob_upload call – cURL example

We will use the MySQL table:

```
CREATE TABLE product_image
(
    product_id  integer     not null,
    name        varchar(64) not null,
    image       longblob    null,
        PRIMARY KEY(product_id)
);
```

Let's upload a BLOB on the server:

```
$ curl -F "blob_id=koala.jpg" -F "file=@/home/mike/koala.jpg" \
 http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/blob_upload
```

Call will return:

```
{
    "status":"OK"
}
```

And now ask for insert in product_image table:

```
$ curl --data "prepared_statement=true" \
 --data "param_type_1=INTEGER&param_value_1=1" \
 --data "param_type_2=VARCHAR&param_value_2=Koala" \
 --data "param_type_3=BLOB&param_value_3=koala.jpg" \
 --data-urlencode "sql=insert into product_image values(?, ?, ?)" \
 http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
execute_update
```

Call will return:

```
{
    "status":"OK",
    "row_count":1
}
```

get_blob_length

Allows to retrieve the size of a BLOB before it's download.

The call is optional and thus is not required before BLOB download.

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/get_blob_length`

URL parameter	Description
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.

Request parameter	Requested	Description
blob_id	Yes	The reference Id of the BLOB to download. The value is retrieved with a prior `execute_query` call, see below in: blob_download.

Notes
This call supports POST method only. All Request parameters must be URL encoded and formatted in UTF-8.

Server response to get_blob_length call

If everything is OK:

```
{
    "status":"OK",
    "length":"{BLOB length in bytes}"
}
```

In case of error:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```

blob_download

Allows to download a BLOB from the remote server:

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/blob_download`

URL parameter	Description
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` to use on server. Optional: if not passed, server will use the one created at login.

Request parameter	Requested	Description
blob_id	Yes	The reference Id of the BLOB to download. The value is retrieved with a prior `execute_query` call, see below.

This call supports POST method only. All Request parameters must be URL encoded and formatted in UTF-8.

Server response to blob_download call

If everything is OK, a stream containing the BLOB content is sent by the server.

In case of error, the stream will contain:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```

get_blob_length & blob_download call – cURL examples

We inserted previously a BLOB in the `product_image` table.

Let's query the `product_image` table:

```
$ curl \
--data-urlencode "sql=select * from product_image" \
--data "pretty_printing=true" \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
execute_query
```

Call will return:

```
{
    "status":"OK",
    "query_types":[
        "INTEGER",
        "VARCHAR",
        "LONGVARBINARY"
    ],
    "query_rows":[
        {
            "row_1":[
                {
                    "product_id":1
                },
                {
                    "name":"Koala"
                },
                {
                    "image":"25bc0d5cfdc04e78b5272d86fca1cd2f.blob"
                }
            ]
        }
    ],
    "row_count":1
}
```

The third column value contains the `blob_id` to use in with `get_blob_length` and `blob_dowbload`. The blob_id value is generated by the AceQLserver.

Let's get the BLOB length (this step is optional) :

```
$ curl --data "blob_id=4a77ce2e9d8a4f8ea581a37f1d3fedd1.blob" \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
get_blob_length
```

Call will return:

```
{
    "status":"OK",
    "length":"79747"
}
```

And let's now download our BLOB:

```
$ curl --data "blob_id=4a77ce2e9d8a4f8ea581a37f1d3fedd1.blob" \
http://localhost:9090/aceql/session/hli7ppunldk07mg8ae4dvv70kc/\
blob_download>/home/admin/blob.jpg
```

get_connection

Allows to open a new `java.sql.Connection` on the server without doing a new authentication with `login` .

URL Format
`server/aceql/session/{session_id}/get_connection`

URL parameter	Description
session_id	The session_id value returned by `login`.

Request parameter	Requested	Description
none		

Server response get_connection call

If everything is OK:

```
{
    "status":"OK",
    "length":"{BLOB length in bytes}"
    "connection_id":{"connection ID number"}
}
```

In case of error:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```

close

Allows to close a `connection` in use and release the corresponding remote JDBC `java.sql.Connection` into the pool.

URL Format
`server/aceql/session/{session_id}/connection/{connection_id}/close`

URL parameter	Description
session_id	The session_id value returned by `login`.
connection_id	The ID that refers the `java.sql.Connection` on the server.

Request parameter	Requested	Description
None		

logout

Allows to close the session and to release all the server JDBC `Connection`s into the pool.

URL Format
`server/aceql/session/{session_id}/logout`

URL parameter	Description
session_id	The session_id value returned by `login`.

Request parameter	Requested	Description
None		

It is important to always call `logout` at end of session in order to close all the JDBC `Connection` and release them in the pool. See Server Installation and Configuration Guide for more info.

Server response to logout call

If everything is OK:

```
{
    "status":"OK",
}
```

In case of error:

```
{
   "status":"FAIL",
   "error_type":{error type numeric value},
   "error_message":"{error message returned by the server}",
   "http_status":{http status code numeric value}
}
```