

Using Javascript User Functions

Date published: 2019-12-17

Date modified: 2022-05-05



Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Creating User Defined Functions.....	4
Developing JavaScript functions.....	5
Adding Java to the Functions language option.....	6
Using System Functions.....	7

Creating User Defined Functions

With SQL Stream Builder, you can create user functions to write powerful functions in JavaScript, that you can use to enhance the functionality of SQL.

About this task

User functions can be simple translation functions like Celsius to Fahrenheit, more complex business logic, or even looking up data from external sources. User functions are written in JavaScript. When you write them, you create a library of useful functions.

Procedure

1. Navigate to the Streaming SQL Console.
 - a) Go to your cluster in Cloudera Manager.
 - b) Select SQL Stream Builder from the list of services.
 - c) Click SQLStreamBuilder Console .

The **Streaming SQL Console** opens in a new window.

2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page. The Materialized View button will be available when you add a query to the SQL Editor.

3. Select Functions from the main menu.
4. Click New function.

User Defined Function

Properties

Name *

Description

Enter text ...

Output Type *

Input Types

Add Input Type

Function (JavaScript) Usage: 0

```
1 function myFunction(input){
2   return "Hello World 2";
3 }
4
5 myFunction($p0); // this line must exist
```

Cancel Create

5. Add a Name to the UDF.

For example, name the UDF to HELLO_WORLD.
6. Add a Description to the UDF.
7. Select the Output and Input data type.

For example, select STRING.

8. Paste the JavaScript code to the editor.

For example:

```
// check to see if the card is VISA
function HELLO_WORLD(card){
  var cardType = "Other";
  if (card.charAt(0) == 4){
    cardType = "Visa";
  }
  return cardType;
}
HELLO_WORLD($p0); // this line must exist
```

9. Click Create.**10.** Once created, you can use a User Function in your SQL statement:

For example:

```
-- simple usage
SELECT HELLO_WORLD(card) AS IS_VISA
FROM ev_sample_fraud;

-- in the predicate
SELECT amount, card
FROM ev_sample_fraud
WHERE HELLO_WORLD(card) = "Visa";
```

**Note:** Valid inputs can be a field in the source virtual table or any other valid input. Functions must be in upper case.**Note:** User Functions have access to the Java 8 API, this increases the overall usefulness and power. For example:

```
function GETPLANE(icao) {
  try {
    var c = new java.net.URL('http://yyyyyy.io' + icao).openConnection();
    c.requestMethod='GET';
    var reader = new java.io.BufferedReader(new java.io.InputStreamReader(c.inputStream));
    return reader.readLine();
  } catch(err) {
    return "Unknown: " + err;
  }
}
GETPLANE($p0);
```

Developing JavaScript functions

When developing JavaScript functions that are more complicated than just simple logic, it is recommended to use the `jjs` command-line utility to create and iterate while writing functions.

About this task

After the function performs the required task, migrate it to the console. Additionally these files/functions can be saved in a source code control system like git/Github.

Procedure

1. Create a file for your function.



Note: It is recommended to name the file with the same name as that of the function.

2. Create some sample input when calling the function.
3. Call `jjjs` on the command line to test the function.

```
$>cat TO_EPOCH.js
function TO_EPOCH(strDate) {
  var strFmt = "yyyy-MM-dd HH:ss:mm";
  var c = new java.text.SimpleDateFormat(strFmt).parse(strDate).getTime()
/1000;
  return c.toString();
}

print(TO_EPOCH("2019-02-02 22:23:13"));

then
$>jjs TO_EPOCH.js
1549167203
```

What to do next

After you have successfully developed the JavaScript code, copy and paste only the function to your code window when creating the JavaScript function in SQL Stream Builder.

Adding Java to the Functions language option

You can create User Defined Functions (UDF) using Java after manually adding the UDF function JAR file that contains the UDF class to the Flink connectors. After creating the function in the SQL window, you can select Java as a language for the UDFs.

Procedure

1. Create a Java UDF function JAR file based on the following example:

```
package udf;
import org.apache.flink.table.functions.ScalarFunction;

public class UdfTest extends ScalarFunction {
  public String eval(String input){
    return "Hello World " + input;
  }
}
```



Note: The function needs to be named as 'eval' in the JAR file.

2. Copy the JAR file to the flink connectors folder:

```
scp <jar_location>/<jar_filename> <your_hostname>:/usr/share/flink-connectors
```

3. Navigate to the Streaming SQL Console.
 - a) Go to your cluster in Cloudera Manager.
 - b) Select SQL Stream Builder from the list of services.
 - c) Click SQLStreamBuilder Console .

The **Streaming SQL Console** opens in a new window.

4. Run the following query to create the function:

```
CREATE FUNCTION myFunction AS 'udf.UdfTest' LANGUAGE java;
```

5. Test the function by running a SELECT query.

```
SELECT myFunction('test');
```

Using System Functions

The same set of system functions can be used for SQL Stream Builder as for Apache Flink.

As SSB runs on Flink, the following built-in system functions can also be used for data transformations in your SQL Jobs.

- [Comparison Functions](#)
- [Logical Functions](#)
- [Arithmetic Functions](#)
- [String Functions](#)
- [Temporal Functions](#)
- [Conditional Functions](#)
- [Type Conversion Functions](#)
- [Collection Functions](#)
- [JSON Functions](#)
- [Value Construction Functions](#)
- [Value Access Functions](#)
- [Grouping Functions](#)
- [Hash Functions](#)
- [Aggregate Functions](#)
- [Column Functions](#)

For more information about the list of supported Functions, see the [Apache Flink documentation](#).