**Pre-requisites for downloading Custom Components:**

1.  JDK 1.7 or higher

2.  Apache Maven 3.x

# Downloading custom components

You can download custom components using Download button

## Building Custom Components

To build custom components, run below command.

|  |
| --- |
| run:  mvn clean install –Pspark2 |

It will create a target folder parallel to **src**  folder where a jar file will be created with the name <module name **> -3.1.0-SNAPSHOT-jar-with-dependencies.jar**

For example, for SFDC channel, below jar will be created inside <module name **>/**target .

|  |
| --- |
| sax-spark-sfdc-channel-3.1.0-SNAPSHOT-jar-with-dependencies.jar |

The final jar file inside the target folder will be used for registering components with StreamAnalytix.

## How to register custom components and lookup functions in StreamAnalytix?

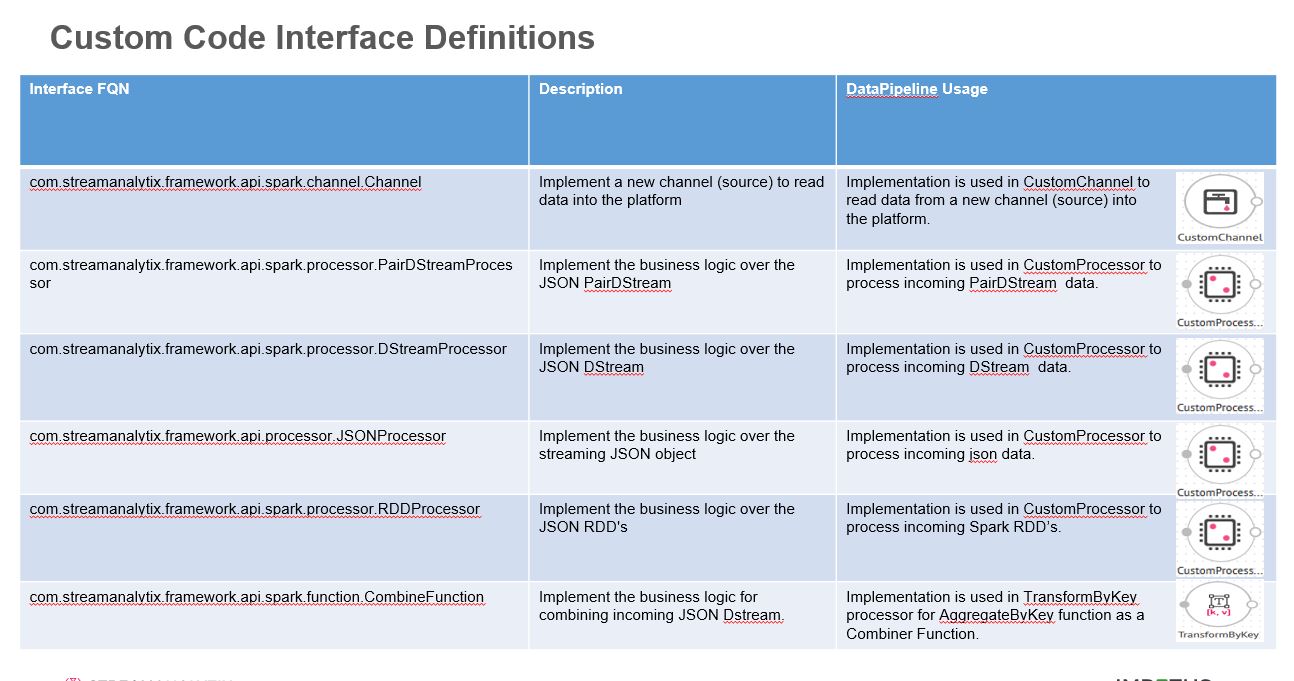
Go to **Register Entities**-🡪 **Register Components**

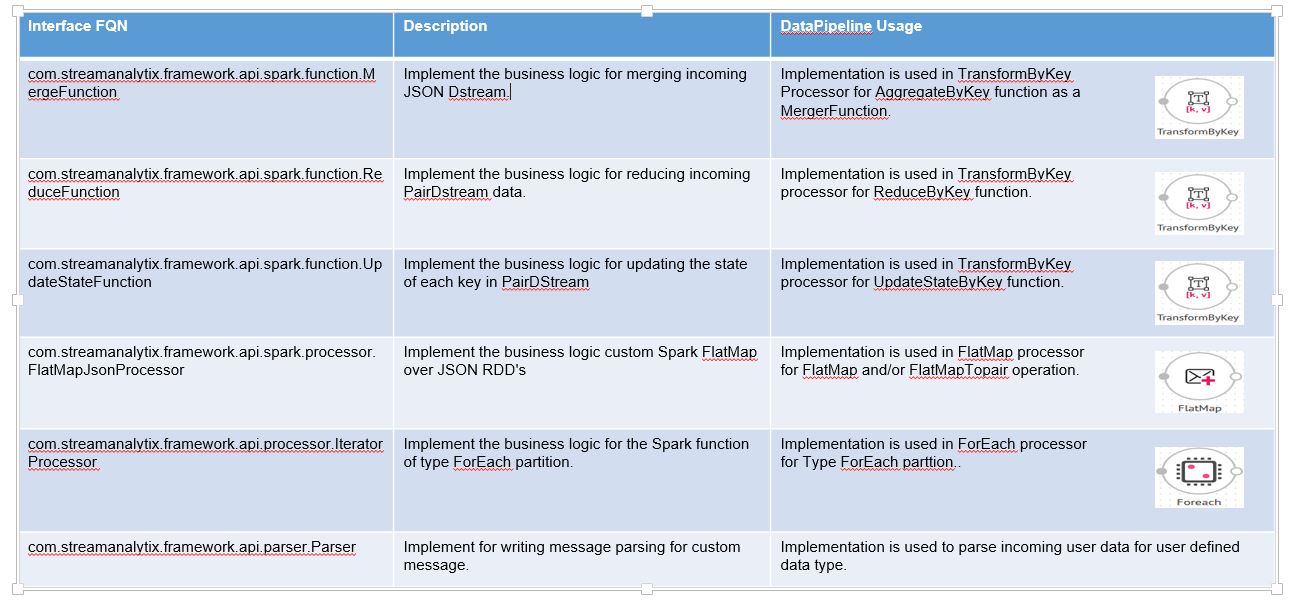
For registering **Custom Lookup** functions, you need to register Functions

Go to **Register Entities-🡪 Register Functions**

**Register Component**

**Custom Code Interface Definitions (Spark)**

****

****

**Example of a Custom Processor Implementation - BlueCoat Processor**

**BlueCoatProcessor** implements **JsonProcessor** – enables processing JSON data.

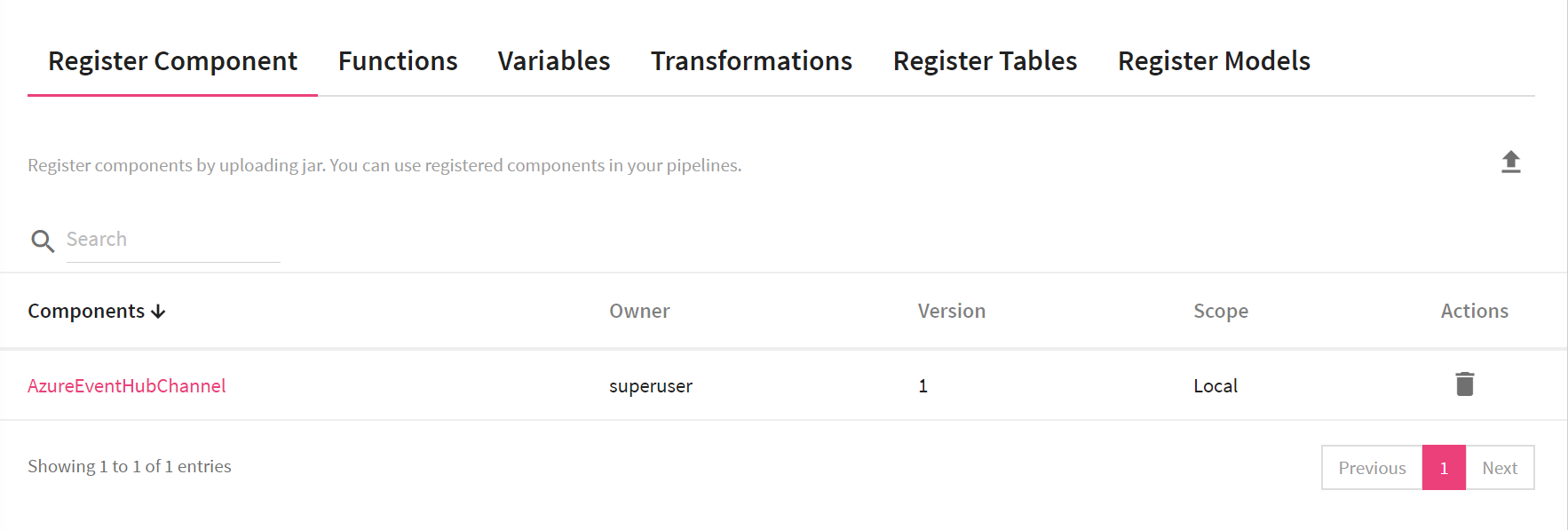
•  Implement life cycle methods

•     **Init** **–** initialization code (example establish connection with database or any other system)

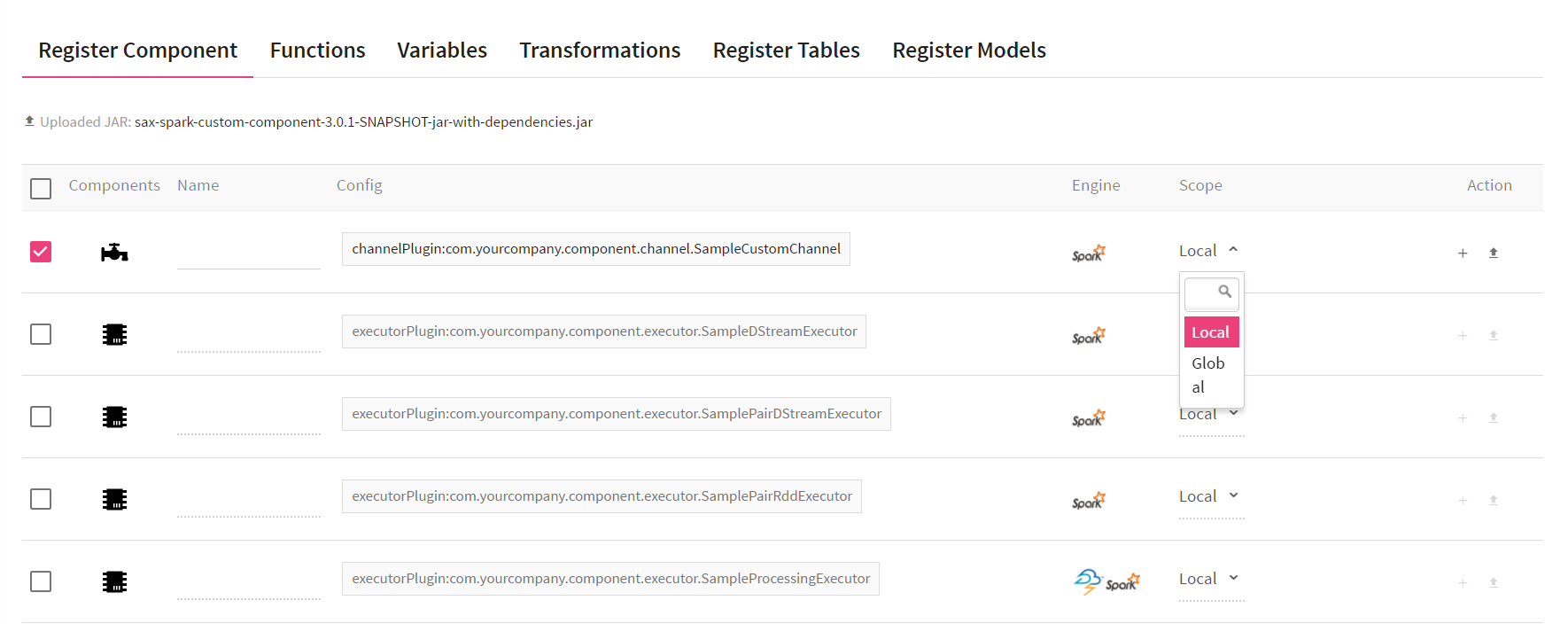
•     **Process** **–** Business logic goes here.

**Register Custom Code**

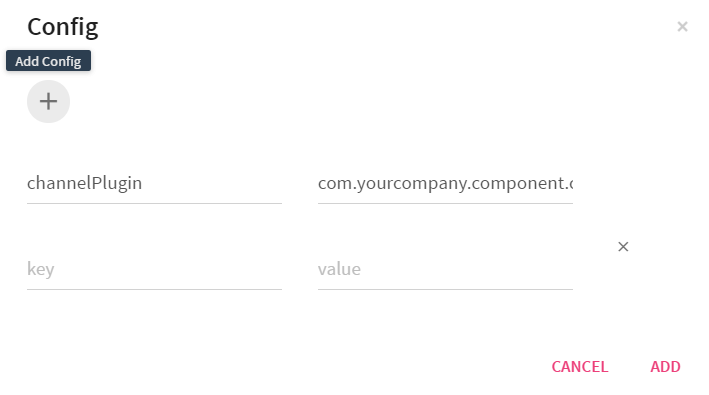
To register a custom component, go to **Register Entities** page.



Select component(s) that you want to register by providing a Component Name and a Scope (Local or Global).



Provide the class name of your custom code implemented in the uploaded jar file.



**How use a registered component in your pipeline**

To use a **Custom Parser**, while configuring a message, on the **Message Configuration** tab, select **Custom** for the **Message Parser Type.**

**Custom Channel** and **Custom Processor** are available on thepipeline canvas right panel under Channels and Processors.

You can register multiple versions of a registered component in your pipeline.

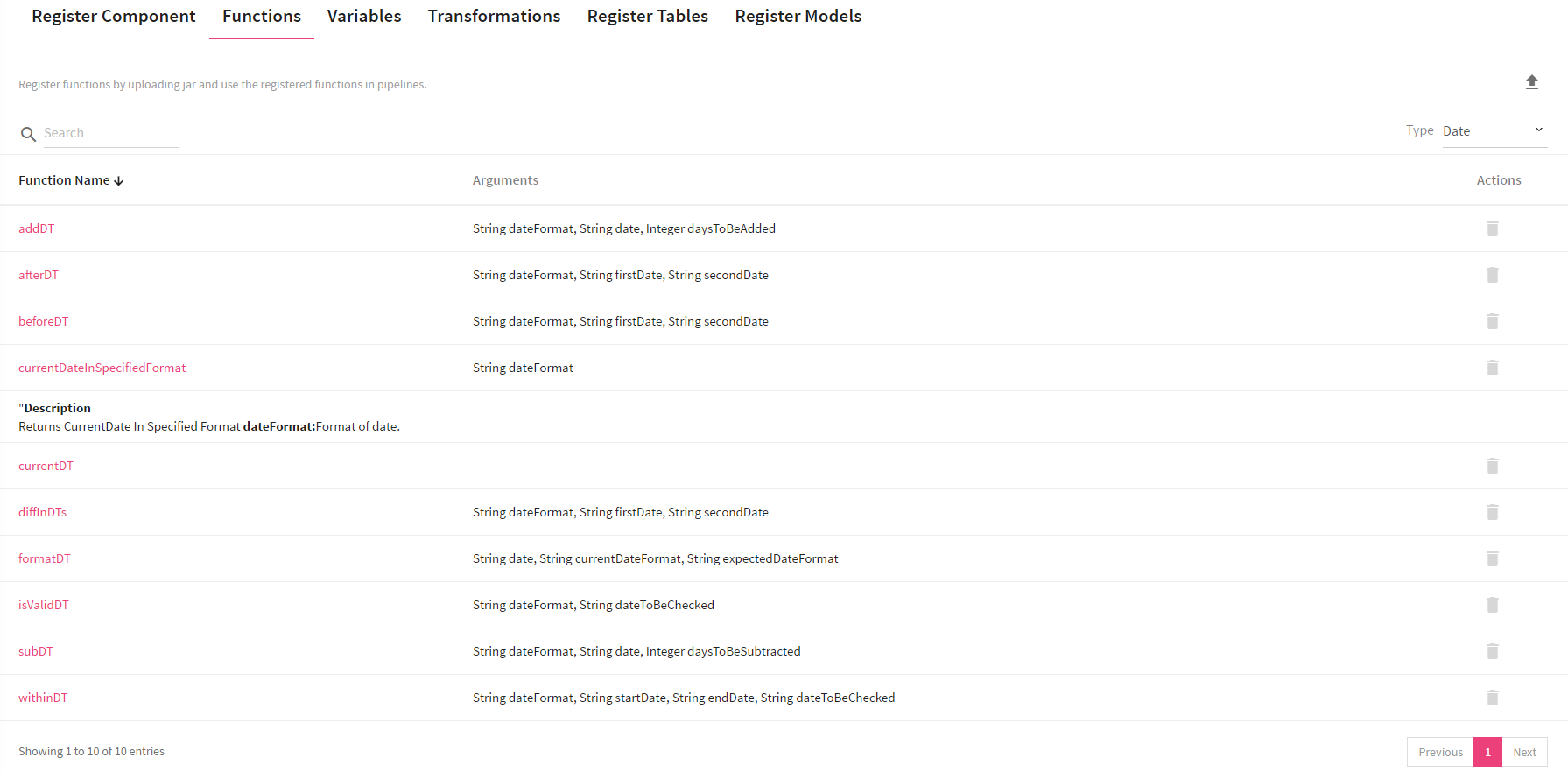
**NOTE:**

If you have used any registered component in the pipeline, make sure that all of the used **registered components** (that is registered with single jar) should be of the same version.

If you have registered component with any fully qualified name, then component with that fully qualified name cannot be registered with another jar in that workspace. Same FQN and same jar will create new version of that component(s).

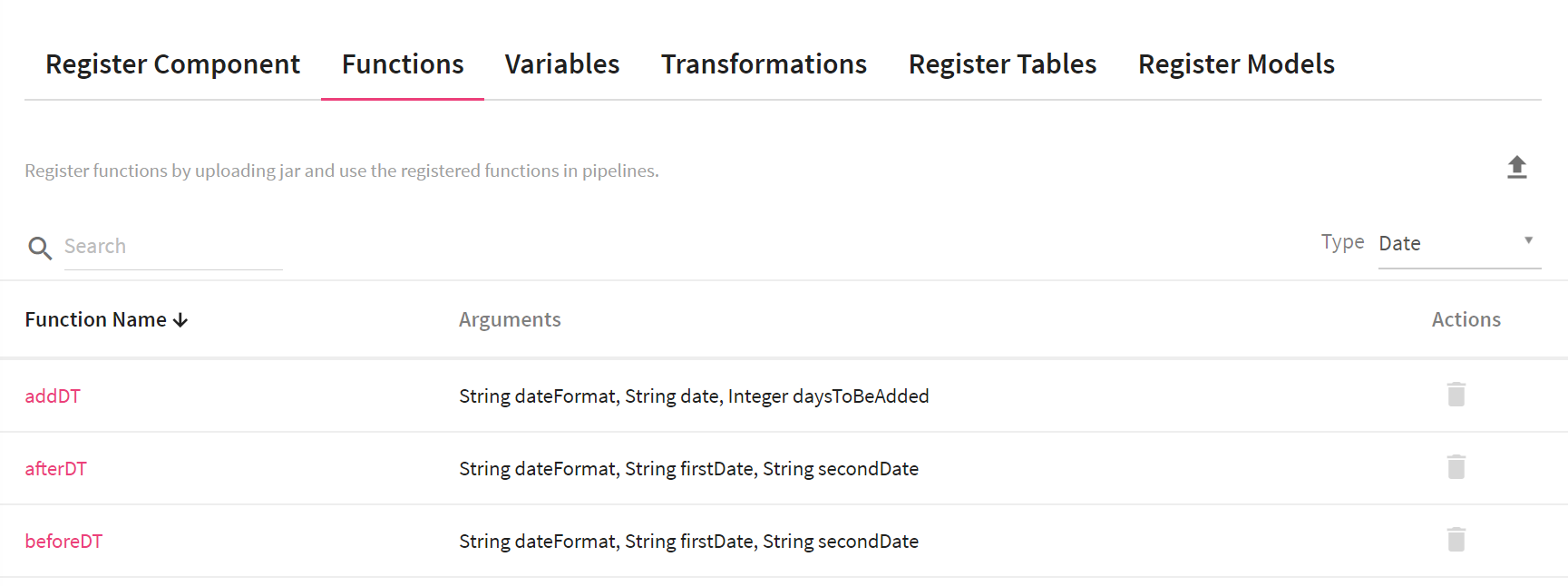
## Functions

Functions enables you to enrich an incoming message with additional data not originally provided by the source.



### User Defined Functions

StreamAnalytix provides a rich library of pre-defined function, and you can define your own functions as **user-defined functions**.

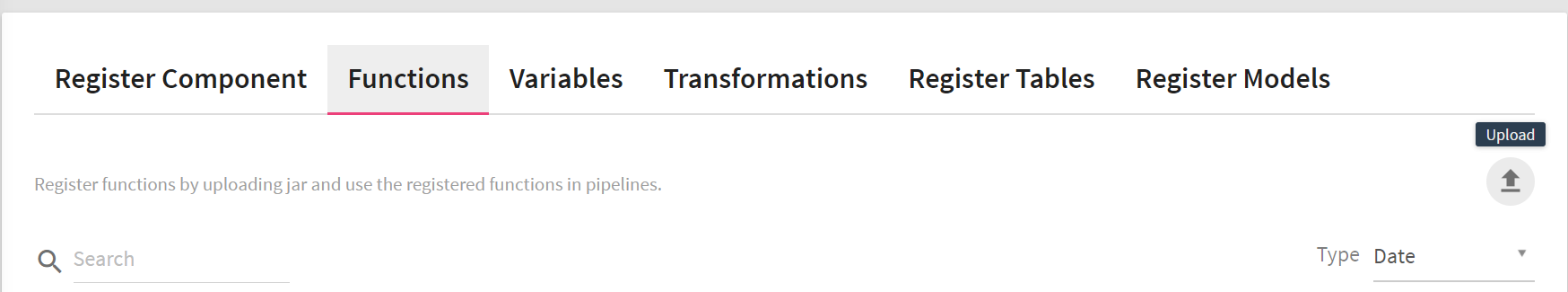


|  |  |
| --- | --- |
| **Field** | **Description** |
| **Function Name** | Name of the function |
| **Arguments** | Input arguments for the function |
| **Parameters** | Function configuration parameter |
| **Upload Jar** | Register functions by uploading a jar file containing custom code |

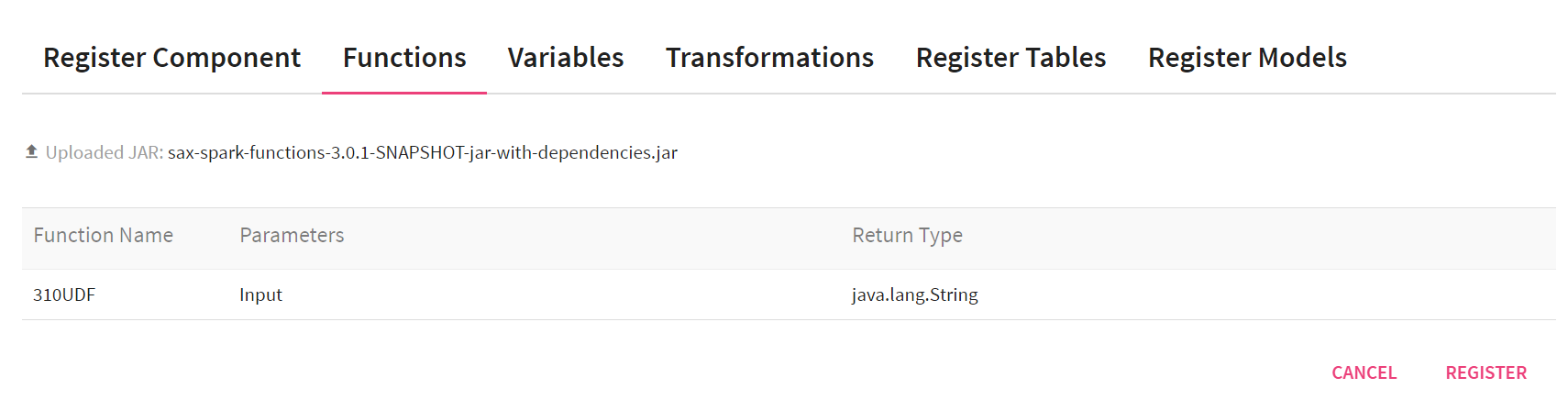
#### User Defined Functions for Spark Pipelines

Build the project through maven and upload the created jar under **Register Entities** -> **Functions**.

Upload lookup jar from the target folder as explained in section building the custom components



Click **Register** to complete Function registration.



#### Using User Defined Functions in a Pipeline

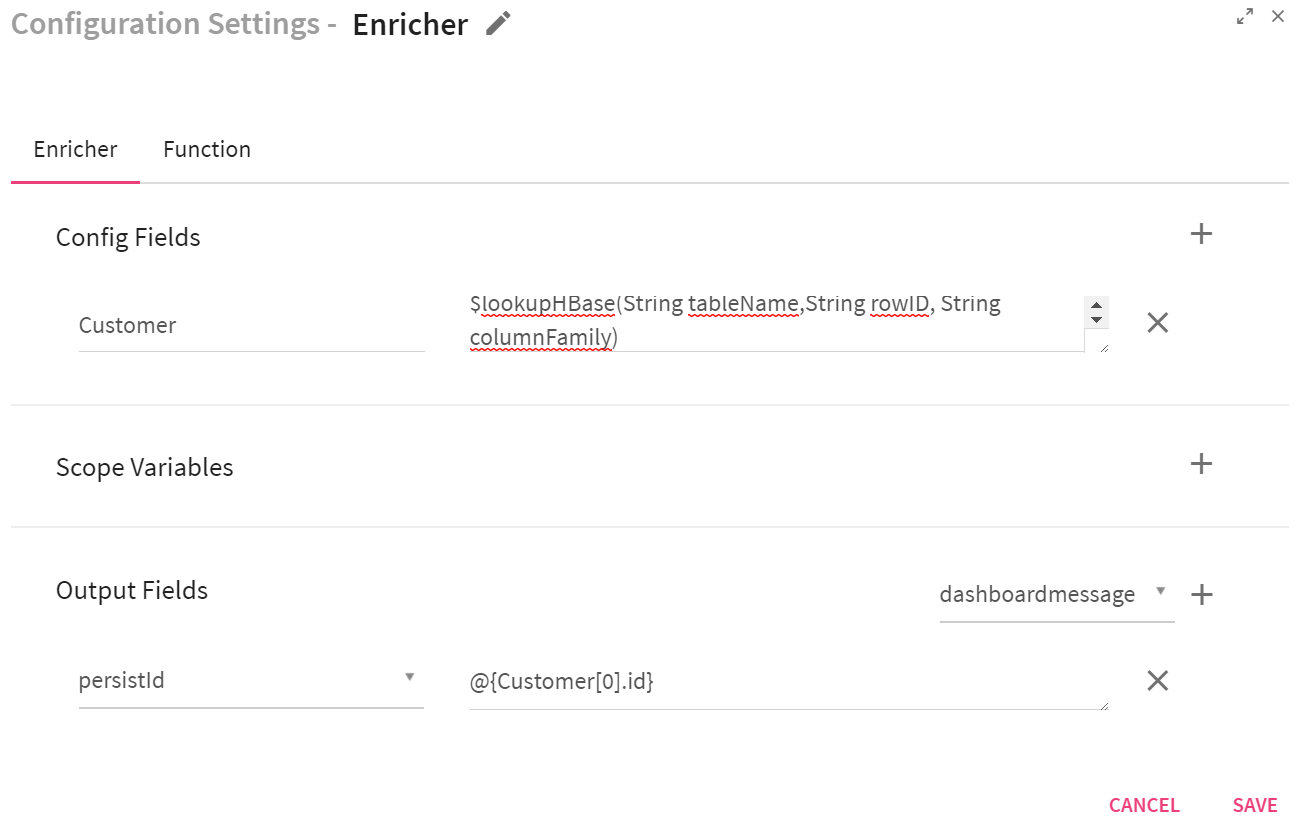
You can use the **Enricher** processor to use **User Defined Functions** in a data pipeline.

To use a function, type **$** in **Enricher's config field**. It will list all the available functions in a dropdown.

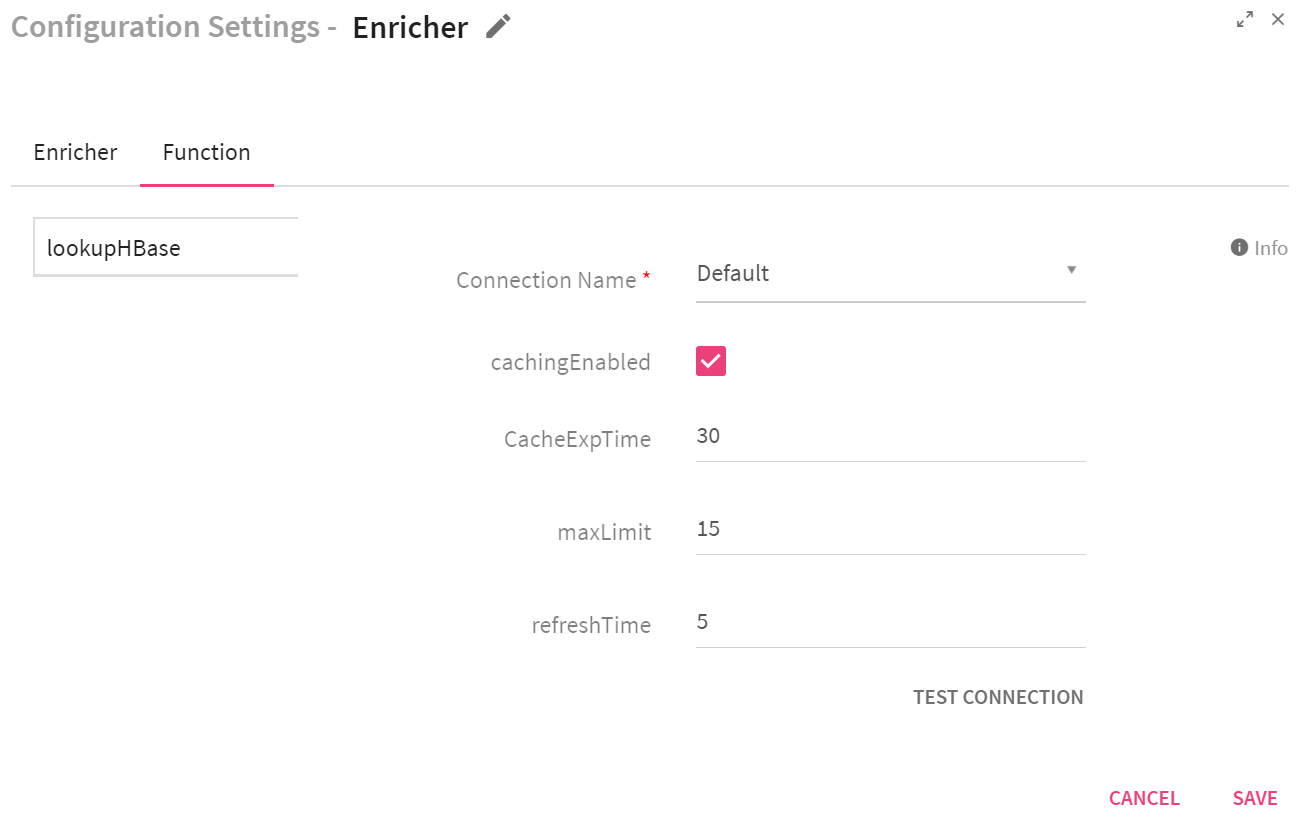
**Note:** It is recommended that all functions should be applied in Config Fields and then those config fields should be used in output fields in the **Enricher** processor.

As given in the below example, config field  Customer is calculated via a function and then it is reused in message field persistId .

**Note:** You need to wrap configuration field in **@{}** in the syntax for using Customer  config field in output field.



For function having parameters, another tab appears on the **Enricher** processor where you need to provide information for its configuration parameters specific to the function.



You can enable caching on individual functions where function results can be cached based on each key, which is created as **pipelineName + componentName + config/output** key name.

|  |  |
| --- | --- |
| **Field** | **Description** |
| **Connection Name** | Name of the connection |
| **cachingEnabled** | Select the checkbox to enable caching for selected function |
| **CacheExpTime** | Cache expiry time in minutes |
| **maxLimit** | Number of records to be cached |
| **refreshTime** | Cache refresh time in seconds |