

IBM Process Mining

Custom Process App Template Guide for a Logic File

Table of Contents

Introduction.....	2
Logic file.....	2
• Entry point (Function)	2
• Configurable variables	2
○ Getting the configuration variables from environment variables.....	2
○ Getting the Configuration Variables from the Context Object.....	3
• Output data format	3
• Support Python modules	4
Configure user input for configurable variables	4
Configure raw data source upload in the Custom Process App	6
Handling exceptions in the python code.....	7

Introduction

Welcome to this guide that helps you create the various elements needed for a successful Custom Process App. As you create your Custom Process App, the most important element is to provide a Python program (The Logic File). By doing data acquisition and transformation, this logic file produces a data set that will be the source of the data for your process mining project.

Logic file

The logic file contains the Python code for the data transformation pipeline. Its structure must meet 4 key requirements: Entry point (Function), Configurable variables, Output data format and Supported Python packages.

- **Entry point (Function)**

The entry point or the entry function is a dedicated function in the logic file that serves as the entry to the data transformation process.

```
def execute(context):
```

Note: If the logic file does not have a defined entry function, the data transformation process will fail before the process can even begin.

- **Configurable variables**

The logic file requires configurable variables to compute its transformation process. These configurable variables include variables such as token key, server or system URL, username and password, and other dynamic input fields. You have two ways to set configurable variables in the Custom Process App:

- Using Config Variable as Environment Variables.
- Using Config Variable as a Context Objects.

Config Variable is a configuration at the “Define user Inputs” step on the Custom Process App where user can create the necessary configurations that are needed for the logic file. When the configuration is in place, it can be used for the process creation.

- **Getting the configuration variables from environment variables**

Within the logic file, these configurable variables must be assessed and retrieved from an environment variable. In other words, each configurable variable needs to be set from an environment variable.

For example, if a username needs to be a configurable variable within the logic file, it must be declared as follows:

```
username = os.environ["username"]
```

Note: The configurable variable is case-sensitive, and you must ensure consistency across the board. For example, if “username” is set in this lowercase state, do not reference it as “UserName” or “userName” elsewhere within the logic file. Reference it only as “username”.

- Getting the Configuration Variables from the Context Object

In this example you can see that the execute function is defined as a ‘context’ parameter. The configuration variables are also available with a Python dictionary that you can retrieve using the context.config syntax.

```
def execute(context):  
    config = context["config"]  
    # example retrieving an input property named 'URL'  
    url = config["URL"]
```

- Output data format

The output of the data transformation process is returned by the same entry function in the logic file. The output format must be a python DataFrame object. A DataFrame is a way of storing and manipulating tabular data in Python. DataFrames are often likened to tables with columns and rows that you can find in any data warehouse. For example, Excel workbooks.

Pandas and Polars are the two supported python DataFrame modules. The output of the data transformation process in the logic file must either be Pandas or Polars DataFrame. See the following:

```
def output(event_list):  
    return pd.DataFrame(event_list)  
  
def execute(context):  
    extract_and_transform(context)  
    return output(event_list)
```

Note: the event_list here is just an example. How the parameter function is declared is up the creator of the logic file.

- Support Python modules

In addition to basic python modules such as **time**, you can find the supported modules for the data transformation logic file in the following list:

jsonschema==4.17.3 numpy==1.24.1 pandas==1.5.3 polars==0.16.8 PyGithub==1.57 python-dateutil==2.8.2 python-dotenv==0.21.1 pyzenhub==0.3.1 requests==2.28.2 types-requests==2.28.11.8 types-urllib3==1.26.25.4 typing_extensions==4.4.0 urllib3==1.26.14 ibm-cos-sdk==2.12.2	ibm-cos-sdk-core==2.12.2 ibm-cos-sdk-s3transfer==2.12.2 oracledb==1.2.2 gitdb==4.0.9 GitPython==3.1.31 confluent-kafka==2.0.2 pymongo==4.3.3 boto3==1.26.100 impyla==0.18.0 elasticsearch==8.6.2 sqlalchemy==2.0.7 google-cloud-bigquery==3.8.0 redis==4.5.4	salesforce-api==0.1.45 servicenow==2.1.0 servicenow-api==0.12.0 simple-salesforce==1.12.3 SOAPpy==0.12.22 splunk-sdk==1.7.3 hbase==1.0.0 neo4j==5.6.0 couchbase==4.1.3 pyexasol==0.25.2 pysolr==3.9.0 pymemcache==4.0.0 pysnc==1.0.7
--	--	--

Configure user input for configurable variables

Through the Custom Process App application (within Process Mining), the configurable variables that are defined within the logic file (as described above) must be configured in the **Define user Inputs** step using the **Add custom input** button (see Figure 1).

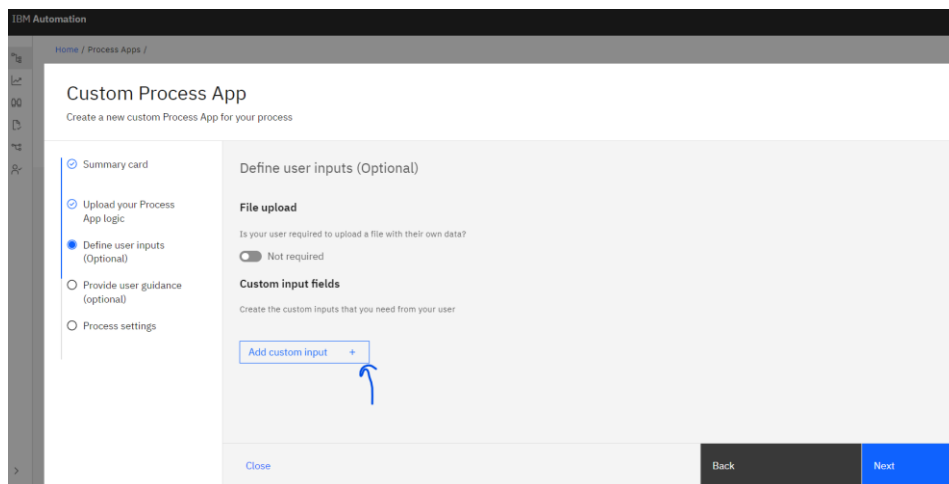


Figure 1: The **Define user Inputs** step using the **Add custom input** button.

For example, if a username is defined as a configurable variable in the logic file, it must be configured in the **Define user Inputs** step (see Figure 2).

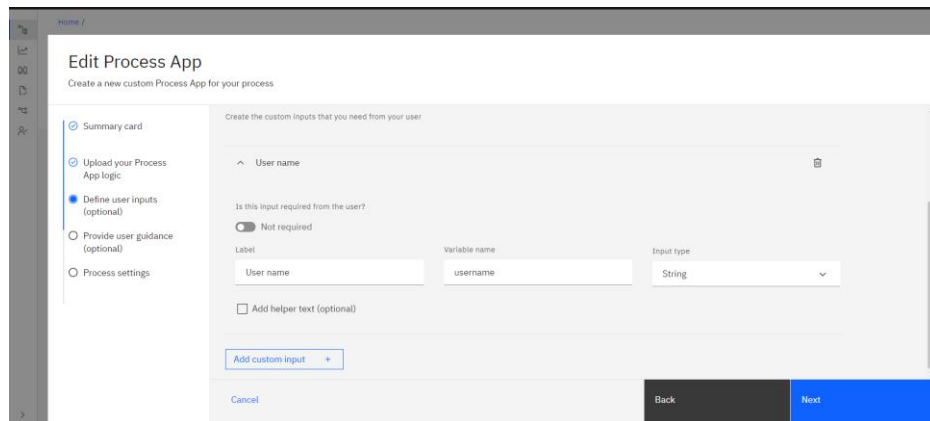


Figure 2: A sample configuration of username variable in the **Define user Inputs** step.

Clicking **Add custom input** displays an accordion that contains all the configurations that are needed for a variable. During the process of configuring the variable, the **Add custom input** button remains disabled until a currently configured variable is saved.

In the displayed accordion menu, do the following steps:

1. Enter the username in the **Label** input field.
2. In the **Variable name** input field, enter the exact variable name that is defined in the logic file. It will be mapped to the input field name property.
3. Select an input type from the dropdown list in the **input type** field. For example, you can select **String** if the username must be used as an example configurable variable.

Other available features of the configuration for a variable are:

- **Not required** toggle button, which indicate to a user whether the input field is required or not.
- **Add helper text** checkbox. If you check it, a text area is displayed to input helper text that will be displayed to the user as a tooltip for the specific configured input field variable.

After you complete configuring the variables, click **Next** to proceed, all the inputs are automatically saved.

You can click the **Delete** icon to remove the configured variable if it is no longer needed.

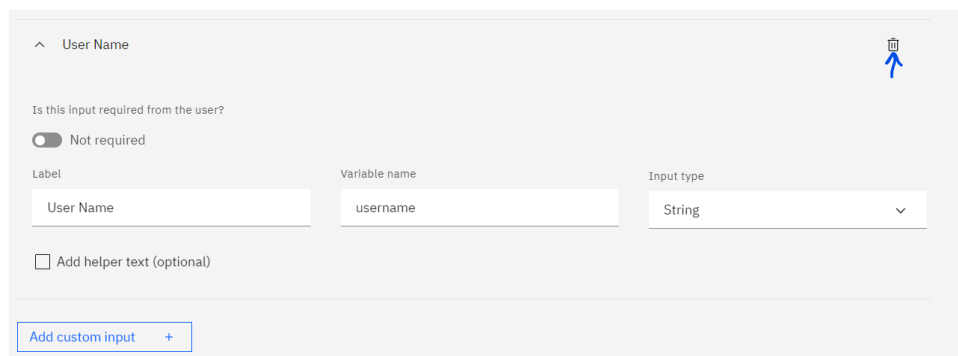


Figure 3: **Delete** and **Save** icon for a configured variable.

When all the required variables are configured and saved (by clicking the “Next” button to proceed in the Custom Process App configuration steps.), the relationship and connection between the logic file and the “Define user Inputs” step are established.

Configure raw data source upload in the Custom Process App

If the raw data source is in CSV file format, it can also be transformed to produce a data set that will be the source for your process mining project. Set the **Required** toggle to be true for **File upload** (see Figure 4). Then the raw data source can be uploaded when you use the created Custom Process App to create a process.

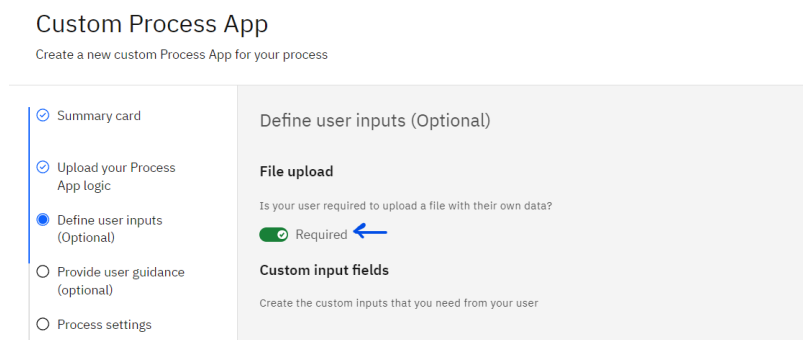


Figure 4: Toggle button at the **Define user Inputs** step to indicate that a raw data source upload is required.

Note: The raw data source must be packaged in a .zip file. The logic file can access the .zip file to perform the necessary data transformation in the same directory location. The logic file can also retrieve the name of the uploaded .zip file either from an environment variable or from the context JSON that is passed into the execute function of the logic file. The name of the key for the uploaded logic file name is “fileUploadName” and the value is whatever the uploaded .zip file name is.

For example, if the name of the uploaded .zip file is “Input_sample.zip”, the logic file can retrieve the name of the .zip file like this:

```
myFileUploadName = os.environ["fileUploadName"]
```

Or,

```
def execute(context):  
  
    # Example retrieving the name of an uploaded zip file  
    myFileUploadName = context["fileUploadName"]
```

The “Input_sample.zip” file is located in the same location as the logic file. Thus, the logic file can reference it inside the same location with the “myFileUploadName” that is returned in the preceding example.

```
|— <xyz>/  
  |— mySampleLogic.py  
  |— Input_sample.zip
```

Handling exceptions in the python code

You may want to send back information to the user of the process app that a credential is invalid or that the process app cannot access an external system. To do this, the python code can send a dedicated exception named `ProcessAppException`. The message that is carried by the exception becomes visible to the end user within the UI. For example:

```
from process_app import ProcessAppException

def execute(context):

    # Example raising a ProcessAppException
    raise ProcessAppException("cannot connect to the system")
```