IBM Process Mining

Custom Process App Template Guide for a Logic File

Table of Contents

Introduction

Welcome to this guide that will help you create the various elements that you need 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) that will do the data acquisition and transformation to produce a data set that will be the source of the data for your process mining project.

Logic File

The logic file is the file containing the Python code for the data transformation pipeline. Its structure/construct must meet 4 key requirements: Entry point (Function), Configurable Variables, Output Data Format and Support Python Packages.

- ### Configurable Variables

The configurable variables are the variables that require external configuration for the logic file these include variables such as token key, server/system URL, username and password, and all dynamics input fields which the logic file needs to compute its transformation process. There are 2 ways in which configurable variable can be configured in Custom Process App: Using Config Variable as Environment Variables and Using Config Variable as a Context Object.

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

- ### 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. It is the starting point and the function that will be invoked dynamically by the Python Process App service. This same entry function will be the function that returns the output of the transformed data in a DataFrame format. The name of the entry function must be called "execute". Here is how it must be defined in the logic file:

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

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

  - #### Getting the Configuration Variables as Environment Variables

Within the logic file, these configurable variables must be assessed/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 set/declared as follows:

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

Note: the configurable variable is case-sensitive, and consistency must be maintained in its usage across the board. For example, if "username" is set in this capital state, do not reference it as "UserName" or "userName" elsewhere within the logic file. Reference it as "username".

Getting the Configuration Variables from the Context Object

- You have noted that the execute function 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 into 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 could find in any data warehouse, e.g., Excel workbook.

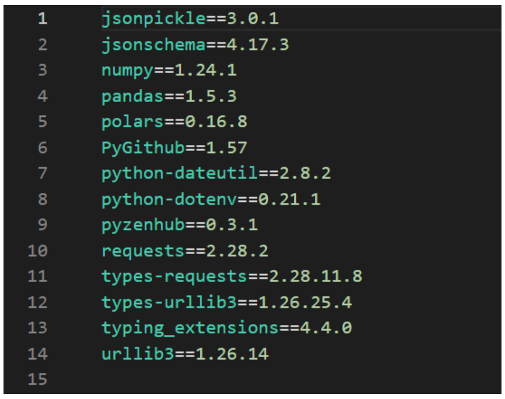
Pandas and Polars are the 2 supported python DataFrame modules. The output of the data transformation process in the logic file must either be Pandas or Polars DataFrame. Here is an example (Note: the event_list in the example below is just an example. How parameter function is declared is up the creator of the logic file):

```
def output(event_list):
    return pd.DataFrame(event_list)

def execute(context):
    extract_and_transform(context)
    return output(event_list)
```

Support Python Modules

In addition to basic python modules such time, below is a list of supported modules for the data transformation logic file.



```
jsonpickle==3.0.1
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

```

Configure User Input in the Custom Process App for Configurable Variables

At the Custom Process App end (within the 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 below).

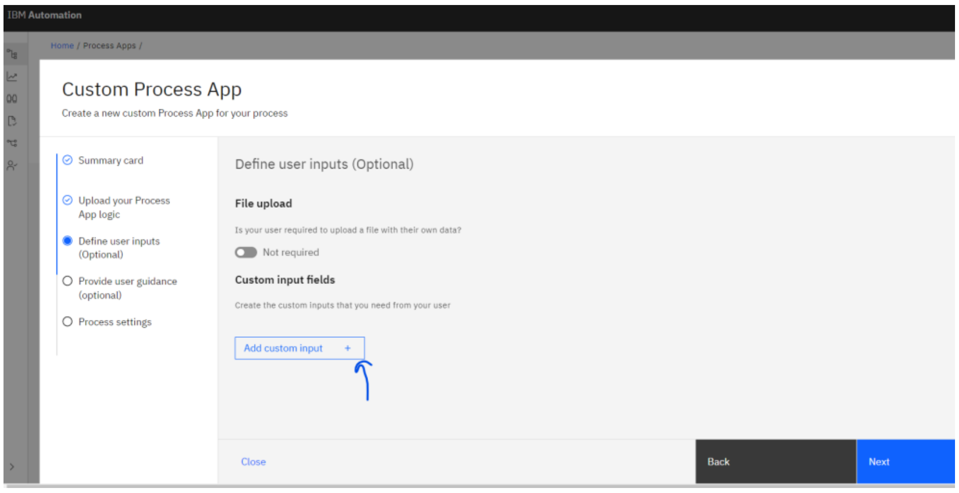
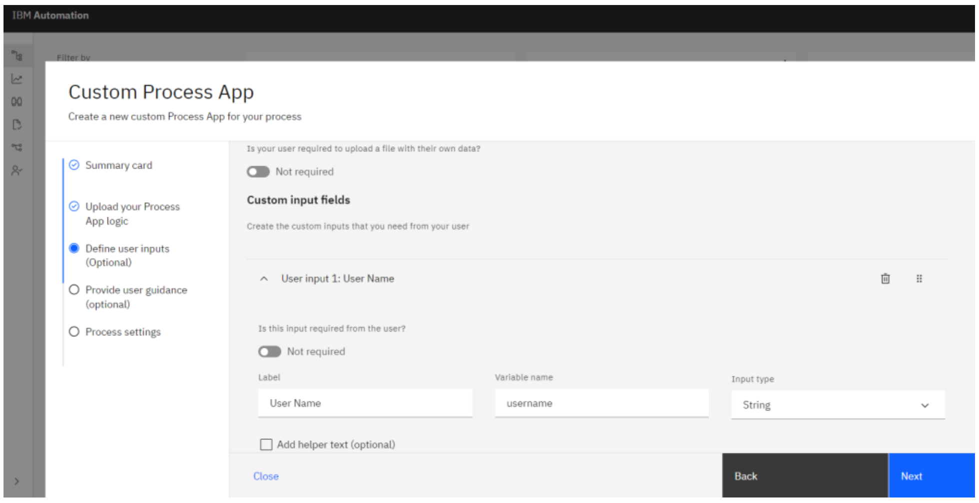

IBM Automation
Home / Process Apps /
Custom Process App
Create a new custom Process App for your process
Summary card
Upload your Process App logic
Define user inputs (Optional)
Provide user guidance (optional)
Process settings
Define user inputs (Optional)
File upload
Is your user required to upload a file with their own data?
Not required
Custom input fields
Create the custom inputs that you need from your user
Add custom input +
Close
Back
Next

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

If, for example, a username was defined as a configurable variable in the logic file, then it must be configured in the “Define user Inputs” step (see Figure 2 below).


IBM Automation
Custom Process App
Create a new custom Process App for your process
Summary card
Upload your Process App logic
Define user inputs (Optional)
Provide user guidance (optional)
Process settings
Is your user required to upload a file with their own data?
Not required
Custom input fields
Create the custom inputs that you need from your user
User input 1: User Name
Is this input required from the user?
Not required
Add helper text (optional)
Close
Back
Next

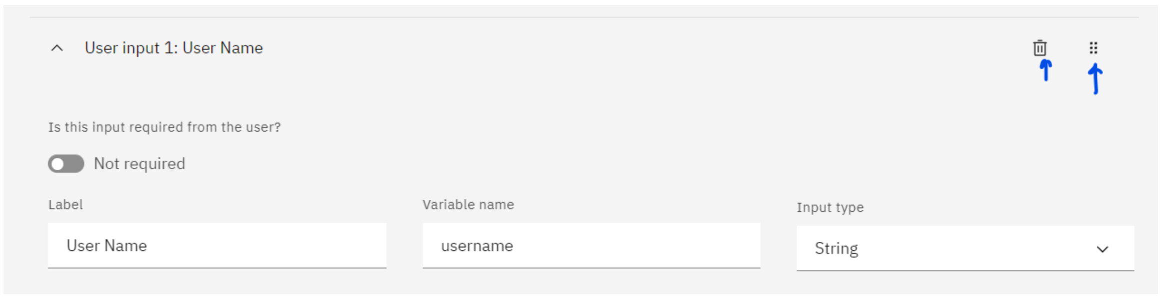
Label	Variable name	Input type
User Name	username	String

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

Click the “Add custom input” button will display an accordion that contains all the configurations needed to fill out for the variable. When the “Add custom input” button is clicked and 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, the “Label” input field represents name given to the input field. The “Variable name” input field represents the exact variable name that was defined in the logic file. It will be mapped to the input field name property. The “Input type” represents the input field type. The list of supported input field type is available as a dropdown. For example, the username that has to be used as an example configurable variable will be a string input type that can be selected from the dropdown of the “Input type”. Other available features of the configuration for a variable are: “Not required” toggle button (which will indicate to an end user where the input file is required or not); and “Add helper text” checkbox (which, if checked, will display a textarea to input helper texts that will be displayed to the end user as a tooltip for the specific configured input field variable).

For each configured variable that has been completed, the “Save” icon (shown in Figure 3 below) must be clicked to capture the configured variable before clicking the next button to go the next step. There is also a delete icon that can be used to remove the configured variable if it is no longer needed. Once the configure variable is saved the “Add custom input” button is enabled again, and more configurable variable can be added in a similar process.


User input 1: User Name

Is this input required from the user?

Not required

Label: User Name

Variable name: username

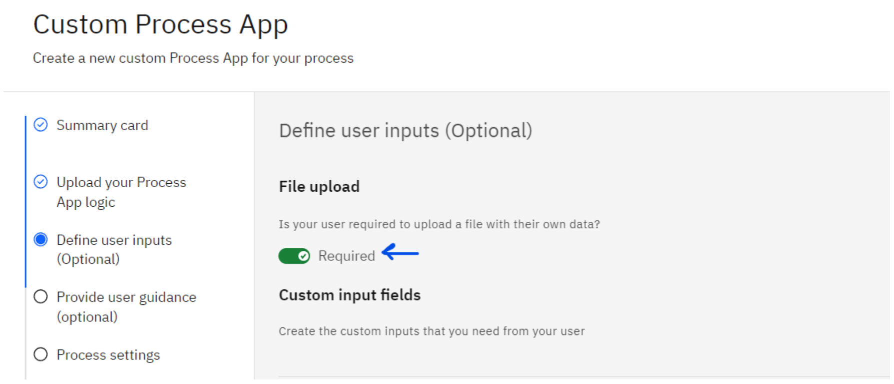
Input type: String

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

Once all the required variables have been filled/configured and saved, the relationship/connection between the logic file and the “Define user Inputs” step should be established, and the next button can be clicked to progress the next step in the Custom Process App configuration steps.

Configure Raw Data Source Upload in the Custom Process App

If the raw data source that needs to be transformed is available in CSV file(s) format, it can also be transformed to produce a data set that will be the source of the data for your process mining project. The “Define user input” step of the Custom Process App, can be used to configure the requirement for the raw data source by setting the toggle button under the “File upload” to true as shown in Figure 4 below.


Custom Process App

Create a new custom Process App for your process

Summary card
Upload your Process App logic
Define user inputs (Optional)
Provide user guidance (optional)
Process settings

Define user inputs (Optional)

File upload

Is your user required to upload a file with their own data?

Required

Custom input fields

Create the custom inputs that you need from your user

Figure4: *Toggle button at the “Define user Inputs” step to indicate that a raw data source upload is required.*

With the “Required” toggle button set to true, the raw data source can be uploaded when using the created Custom Process App to create a process. Note: If the Custom Process App has the “Required” File upload set to true, the raw data source must be packaged in a ZIP file when the Custom Process App is used to create a process mining. The logic file can access the ZIP file to perform the necessary data transformation in the same directory location of the logic file. The logic file can also retrieve the name of the uploaded ZIP file either from 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):
  config = context[“config”]
  # Example retrieving the name of an uploaded zip file
  myFileUploadName = config[“fileUploadName”]
```

The “Input_sample.zip” is located in the same location of the logic file, thus, the logic file can reference it inside the same location of the logic file with the “myFileUploadName” (gotten it the example above).

```
├── <xyz>/
    ├── mySampleLogic.py
    ├── Input_sample.zip
```