Custom Process App in IBM Process Mining

Custom Process App in IBM Process Mining

What is a Custom Process App

A Process App is a set of capabilities that cover a specific Process Mining scenario. A Process App contains what is needed to successfully analyze the use case. It covers the data acquisition and transformation phase to retrieve and prepare the data to be analyzed from the systems that are involved in the scenario. As an example, the "Procure to Pay" Process App proposes to transform data that is acquired from SAP ERP to get insights from the procurement process of a company implemented within SAP. A Process App also comes with a set of predefined analytics dashboard that covers the important elements that a business analyst would need to get insights on the specific uses case.

The notion of Custom Process App is the ability to create your own Process App by providing the elements that are needed to cover your particular use case. Building a Custom Process app requires a good understanding of the target use case because you need to understand the process that is being analyzed and what business insights you want to highlight. It also requires coding skills because the data acquisition and transformation need to be provided as a Python program.

When you build you own Custom Process App, the process app becomes available as a new tile in the process app list within the IBM Process Mining UI and can be used by all the business analysts that would want to use it to analyze a particular case.

This section creates a process app that connects to Salesforce and loads the Opportunities objects and their historical changes along the time.

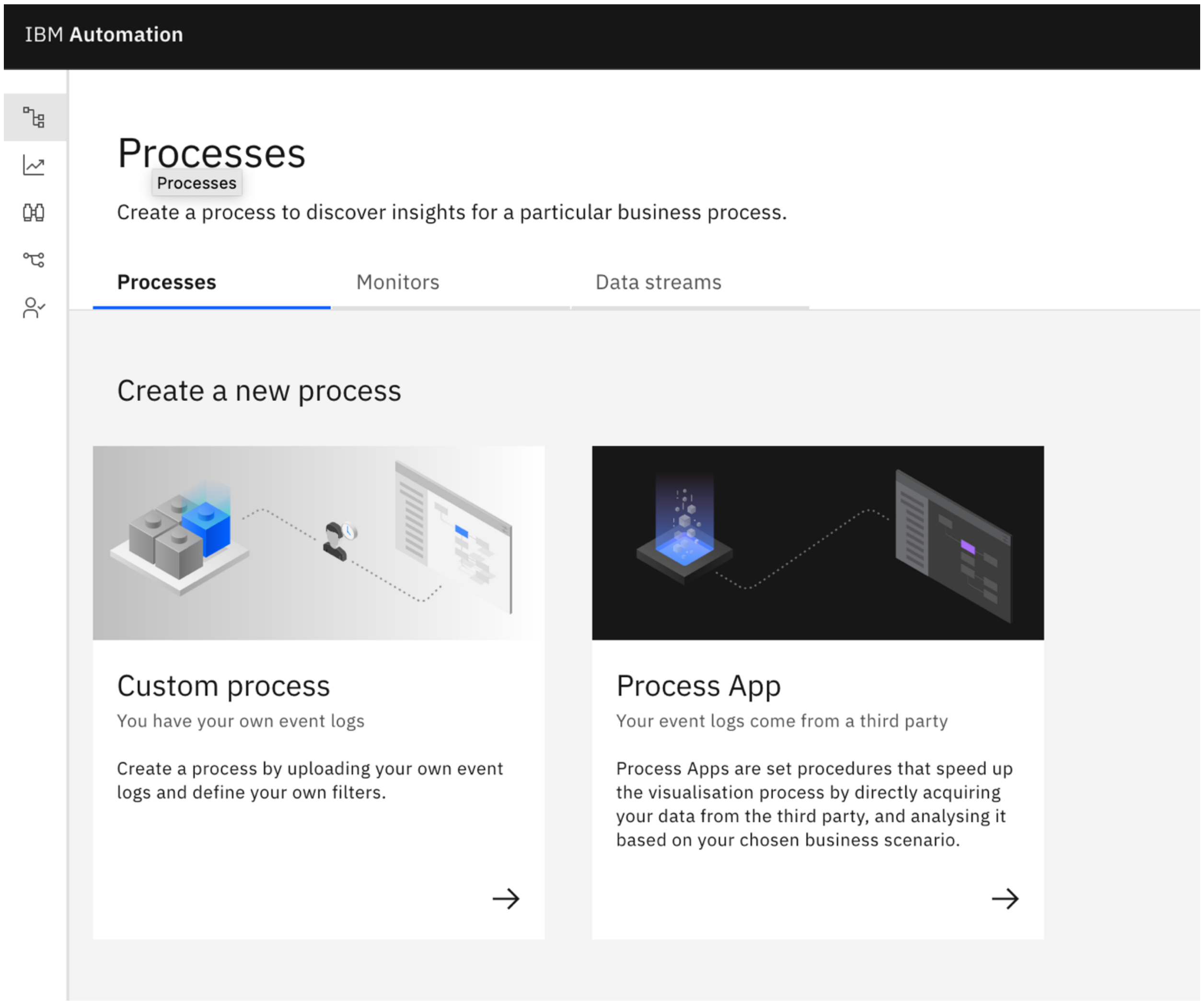
Creating a first custom process app

You create a Process App from the Process Mining UI by following the Process App creation wizard. In this wizard, you need to provide several main elements that constitute the process app:

- Some **metadata** on the process app such as its name, a description, and a document. The information is presented to the business analyst in the final Process App tile.
- The **Process App logic** that is the python program that performs the data acquisition and transformation.
- The **user input parameters** for the Process App. You can specify input parameters, for example, the credentials to connect to an external system to retrieve the data or other configuration parameters that you might need. The parameters will be provided to the python program so that the data extraction and the transformation into a process mining event log can leverage those configuration parameters.
- The final Project configuration as a Project backup file (.IDP file). The project backup file is an important part of the Custom Process App. In the project backup, you need to provide the Data Mapping between the data set that is produced by

the data acquisition in python. You can also provide some predefined dashboards, filters, or all elements that you can store in project backups of IBM Process Mining.

To create the process app, when logged in IBM Process Mining, open the **Process App** section:



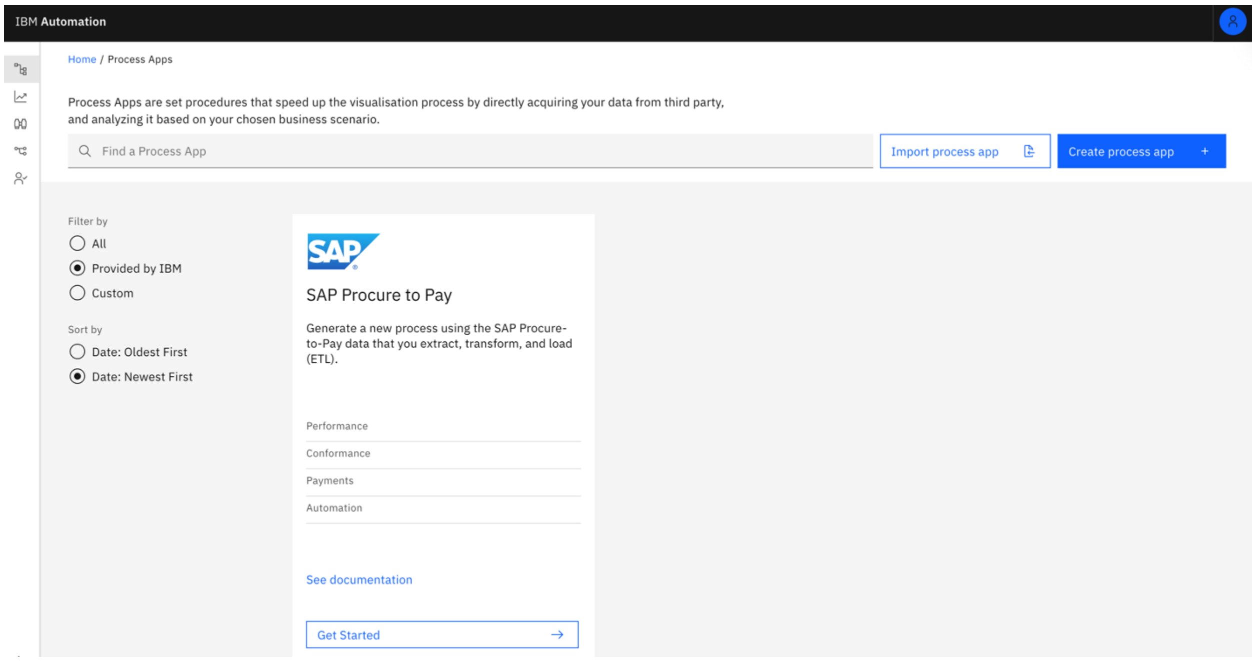
Create a process to discover insights for a particular business process.

Create a new process

Create a process by uploading your own event logs and define your own filters.

Process Apps are set procedures that speed up the visualisation process by directly acquiring your data from the third party, and analysing it based on your chosen business scenario.

A page is displayed with a list of all the available process apps. You can start creating a new process app by clicking the **Create process app** button.


IBM Automation

Home / Process Apps

Process Apps are set procedures that speed up the visualisation process by directly acquiring your data from third party, and analyzing it based on your chosen business scenario.

Filter by
- All
- [✓ Provided by IBM]
- Custom

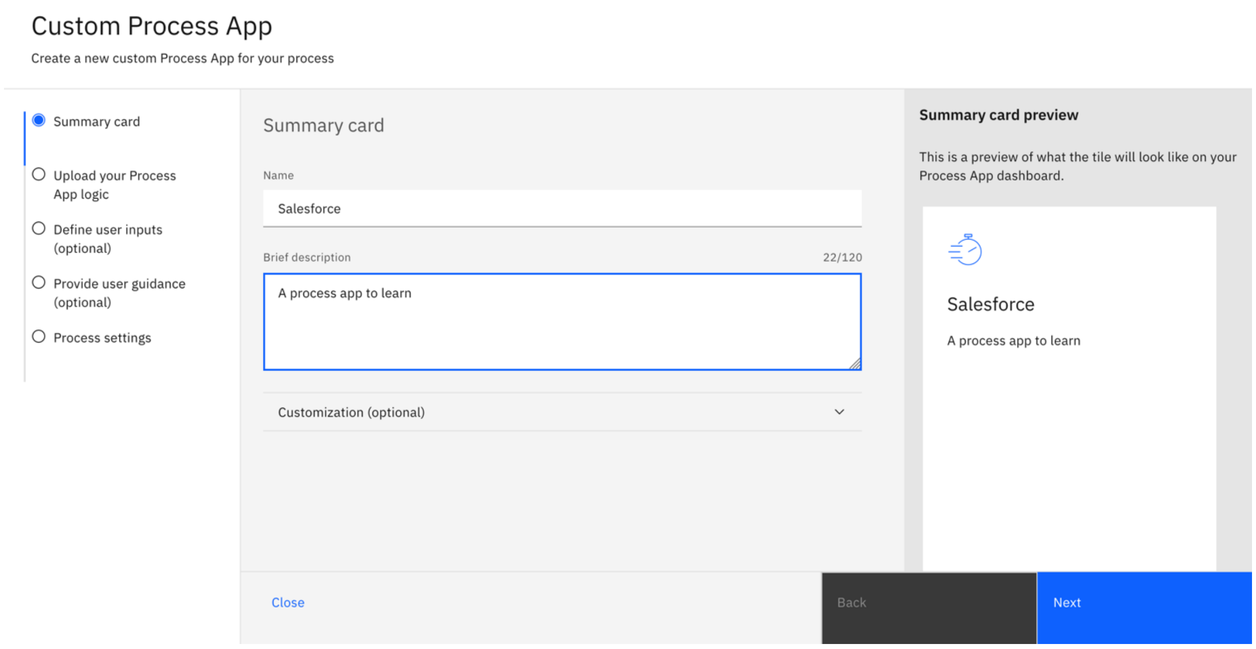
Sort by
- Date: Oldest First
- [✓ Date: Newest First]

SAP Procure to Pay

Generate a new process using the SAP Procure-to-Pay data that you extract, transform, and load (ETL).

Performance
Conformance
Payments
Automation

This button leads to the process app builder, where you can fill the process app information such as the name and a brief description. A preview of the process app card appears on the right side. Let’s call the process app “Salesforce”.


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

Summary card

Name: Salesforce

Brief description 22/120: A process app to learn

Summary card preview

This is a preview of what the tile will look like on your Process App dashboard.

Salesforce

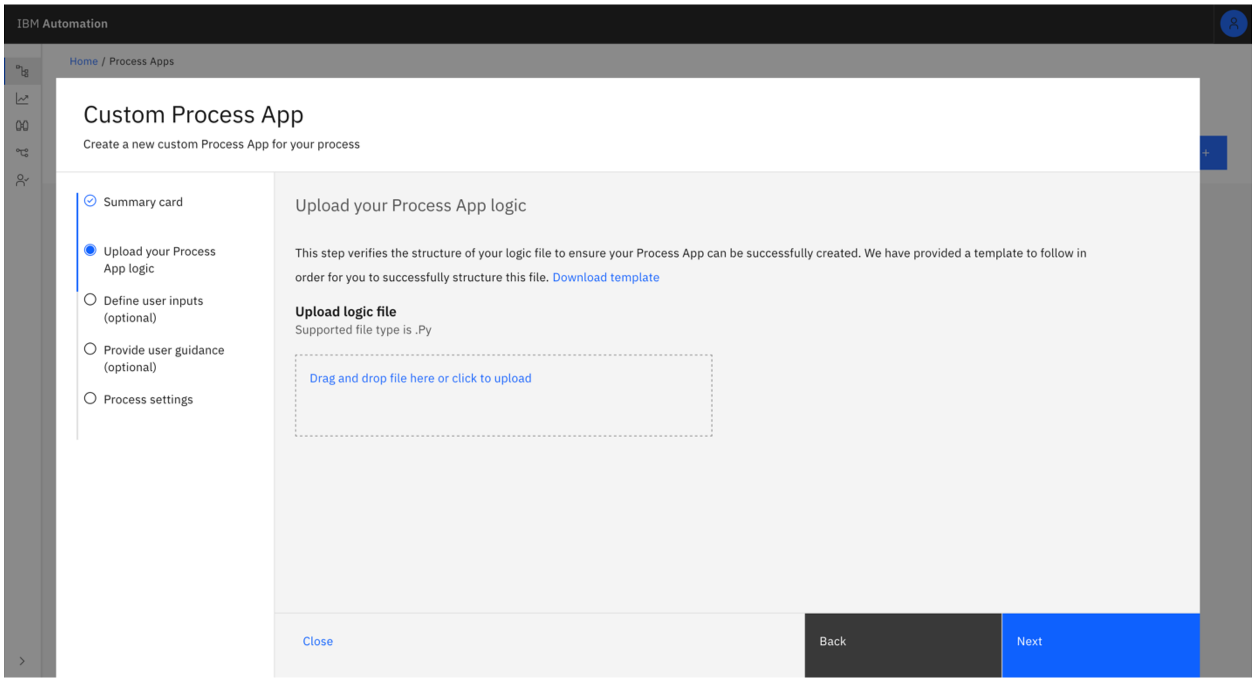
A process app to learn

You can continue to provide more information on the process app by opening the customization option for some additional configuration such as the icon, more bullet point information to place on the card or an additional URL to the process app documentation.
Click **Next** and move to the next step of the wizard that is the process app custom logic file.

The process app python code: the logic file

As you click **Next** in the wizard, you are asked to provide a python file that will do the actual data acquisition and transformation. In this case, we want the code to connect to Salesforce, collect data, and prepare it to be used by IBM Process mining.

Note that this step as you can see in the following picture provides a link to a code template that can help getting started with the code.


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
Close
Back
Next

Upload your Process App logic

This step verifies the structure of your logic file to ensure your Process App can be successfully created. We have provided a template to follow in order for you to successfully structure this file. Download template

Upload logic file

Supported file type is .Py

Drag and drop file here or click to upload

Within the template, you must write data acquisition and transformation within one single python file and the file must provide a method named **execute**.

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

This method has a single parameter that is named context and it is a python dictionary that contains all the contextual information that you need to write you process app.

The first goal is to connect to Salesforce account through the Salesforce REST API. To do so, you need to enable it by creating a "connected app". The "connected App" is the way to declare an application that wants to connect to Salesforce by using OAuth. For more information about how to create a connected app and retrieve the OAuth settings in your Salesforce organization, see the Salesforce documentation at: https://help.salesforce.com/articleView?id=connected_app_create.htm.

The connected app creates the ability for an application to connect to Salesforce by using OAuth. The oauth client id and secret are called the consumer key and consumer secret of the connected app. In addition to the OAuth parameters, to call the API to connect to salesforce, you also need a username and password that can log into the salesforce account.

These four parameters that include username, password, consumer secret and consumer key are going to become configuration parameters for the process app. Then the final user of the process app can provide the values to connect to the Salesforce account.

Then execute the code by retrieving the configuration parameters that you need (we will see after how to declare them within the process app builder).

Here is the code to get the process app configuration and connect to Salesforce.

```
import requests
import json
from requests.models import PreparedRequest
from process_app import ProcessAppException


def execute(config):

    # reading data from accelerator config
    consumerKey = config['consumerKey']
    secret = config['consumerSecret']
    username = config['username']
    password = config['password']

    if consumerKey=='' or secret =='' or username =='' or password =='':
      raise ProcessAppException('configuration not valid')

    # login to salesforce

    loginParams = {'client_id': consumerKey,'client_secret': secret,
                   'grant_type': 'password','username':username,
                   'password':password}
    req = PreparedRequest()
    req.prepare_url('https://login.salesforce.com/services/oauth2/token', loginParams)

    loginResponse = requests.post(req.url)
    if loginResponse.status_code != 200 :
      raise ProcessAppException('cannot log to salesforce account')
    data = loginResponse.json()
    accessToken = data.get('access_token')
    instanceUrl = data.get('instance_url')

    events = []

    importOpportunityCreation(accessToken, instanceUrl, events)
    importOpportunityChange(accessToken, instanceUrl, events)

    df = pd.DataFrame(events,
        columns= ['opportunityId',  'activity', 'startTime',  'resource',
                'opportunityAmount', 'opportunityStage'])
    return df
```

There are several important code sections here, first you see how the configuration parameters of the process app are available in a “config” dictionary that is present in the context dictionary.

```
    config = context[“config”]
```

In the code, we invoke the Salesforce REST ‘token’ API to get an authentication token using the python ‘request’ module.

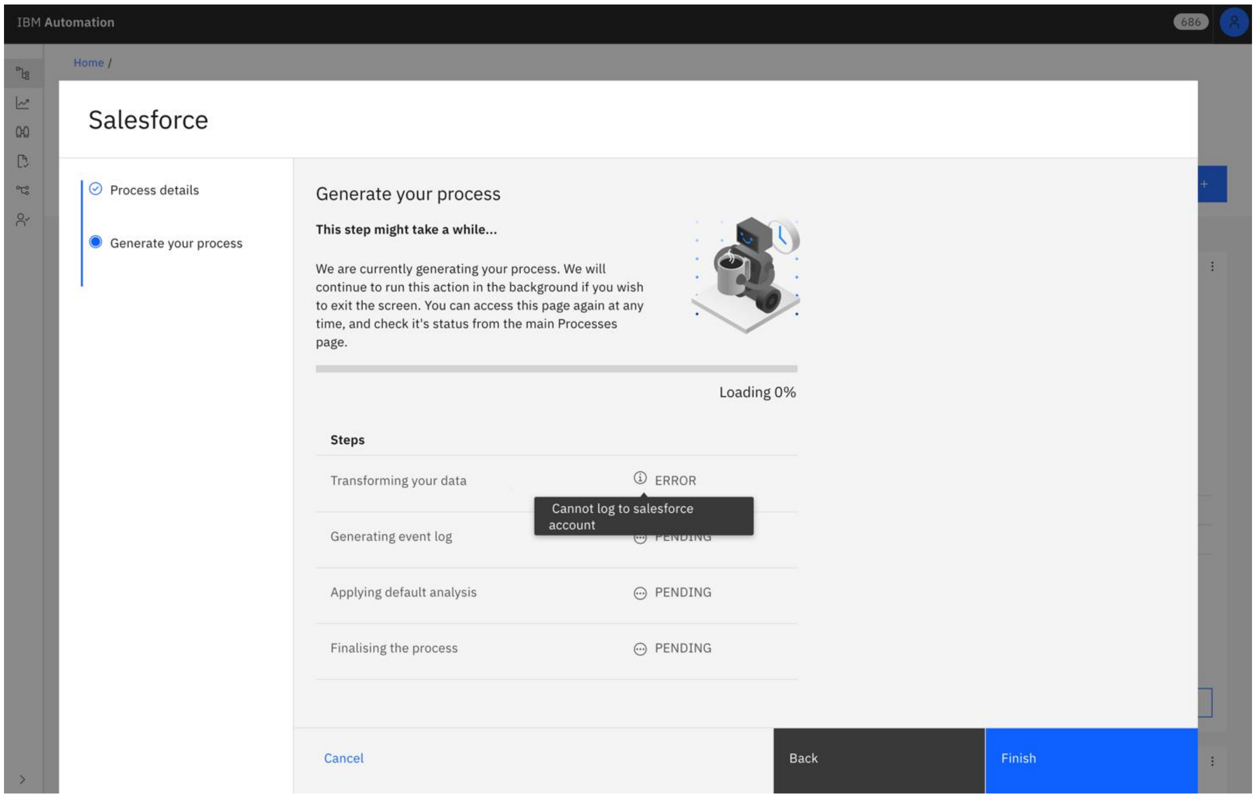
```
req.prepare_url('https://login.salesforce.com/services/oauth2/token', loginParams)

loginResponse = requests.post(req.url)
```

If you cannot log into Salesforce, we will throw a **ProcessAppException**.

```
  if loginResponse.status_code != 200 :
      raise ProcessAppException('cannot log to salesforce account')
```

It is important to note that this exception is a special exception and raising this exception does not correspond to a crash of the process app. On the contrary, the message that is contained in this exception is presented to the end users so that they can change the configuration parameter and retry to run the process app.

Here is how the error appears:



Generate your process

This step might take a while...

We are currently generating your process. We will continue to run this action in the background if you wish to exit the screen. You can access this page again at any time, and check it's status from the main Processes page.

Loading 0%

Steps	
Transforming your data	ERROR
Generating event log	PENDING
Applying default analysis	PENDING
Finalising the process	PENDING

Cannot log to salesforce account

Let’s continue the code itself. With an access token, we can query the opportunity data using salesforce API. In this example, we just retrieve the opportunity data and opportunity historical changes information, but salesforce manages much more data that could be interesting to load. We won’t go into this as the purpose is more about learning the process app concepts.

The last portion of the code looks like this:

```
events = []

importOpportunityCreation(accessToken, instanceUrl, events)
importOpportunityChange(accessToken, instanceUrl, events)

df = pd.DataFrame(events,
    columns= ['opportunityId',  'activity', 'startTime',  'resource',
            'opportunityAmount', 'opportunityStage'])
return df
```

The following are two methods to retrieve the Opportunity “creation” and “change” events. But what is important here is that the execute method ends by producing and returning a Pandas DataFrame that is created from the array of collected events. This returned DataFrame is the result of the data acquisition and transformation of the process app. Here the opportunity ID is the process ID, the activity column is the name of the activity in the process, the start time column is the start time of the activity, and the resource is the person performing the activity. We also collect the opportunity amount and stage.

We have seen all the elements that constitute the process app python code, but let's see how we retrieve the Salesforce data.

First, we need a method to execute SOQL query which is the Salesforce query language that is used to query the data.

```
def runSOQLQuery(query, accessToken, instanceUrl, nextRecordsUrl=None):
    req = PreparedRequest()
    if nextRecordsUrl != None:
      req.prepare_url(instanceUrl+nextRecordsUrl, {} )
    else:
      req.prepare_url(instanceUrl+'/services/data/v39.0/query/', {'q' : query})

    queryResponse = requests.get(req.url,
                  headers={'Authorization': 'Bearer ' + accessToken})
    if queryResponse.status_code != 200 :
      raise Exception('cannot perform SOQL query to salesforce account')
    return queryResponse.json()
```

You can see that an HTTP GET is done on the 'query' REST end point. One notable element here is the handling of the nextRecordsUrl. The response of this GET call can return an URL to retrieve the remaining elements when too many records are available so that the data can be retrieved in several chunks. This runSOQLQuery method will be called recursively with this nextRecordsUrl if needed.

The code to retrieve the opportunities in Salesforce is then a simple SOQL query within the Opportunity table. We query the opportunity id, amount, stage and the name of the person creating the opportunity.

```
def formatDate(date):
   return date[0: date.find('.')].replace('T', ' ')

def importOpportunityCreation(accessToken, instanceUrl, events, nextRecordsUrl=None):

   queryResults = runSOQLQuery(
     'Select Id, Amount, StageName, CreatedBy.Name from Opportunity ',
     accessToken, instanceUrl, nextRecordsUrl)

   records = queryResults.get('records')
   for record in records:
     resource = record.get('CreatedBy').get('Name')
     createDate = formatDate(record.get('CreatedDate'))
     amount = record.get('Amount')
```

```
      events.append([ record.get('Id'),
                'Create Opportunity', createDate,
                resource,
                '0' if 'null' == amount else  amount,
                record.get('StageName') ])

    if queryResults.get('nextRecordsUrl') != None :
      importOpportunityCreation(accessToken, instanceUrl, events ,
                             queryResults.get('nextRecordsUrl'))
```

You can see the events created for the event log of process mining are appended in the events array.

To retrieve the events that correspond to changes in the opportunities, we use a query in the OpportunityFieldHistory table:

```
def importOpportunityChange(accessToken, instanceUrl, events, nextRecordsUrl=None):

    queryResults = runSOQLQuery('Select OpportunityId, CreatedBy.Name,  CreatedDate,'
      + ' Field, NewValue, OldValue from OpportunityFieldHistory',
       accessToken, instanceUrl, nextRecordsUrl)

    records = queryResults.get('records')
    for record in records:
      resource = record.get('CreatedBy').get('Name')
      opportunityId = record.get('OpportunityId')
      createDate = formatDate(record.get('CreatedDate'))
      field = record.get('Field')

      if 'StageName' == field:
        activity='Set Opportunity to ' + record.get('NewValue')
      elif 'Owner' == field:
        activity='Change opportunity Owner'
      elif 'AccountId' == field:
        activity = 'Change opportunity Account'
      elif 'Amount' == field:
        activity= 'Change opportunity Amount'

      events.append([  opportunityId, activity, createDate, resource, 0, ‘’ ])

    if queryResults.get('nextRecordsUrl') != None :
      importOpportunityChange(accessToken, instanceUrl, events,
          queryResults.get('nextRecordsUrl'))
```

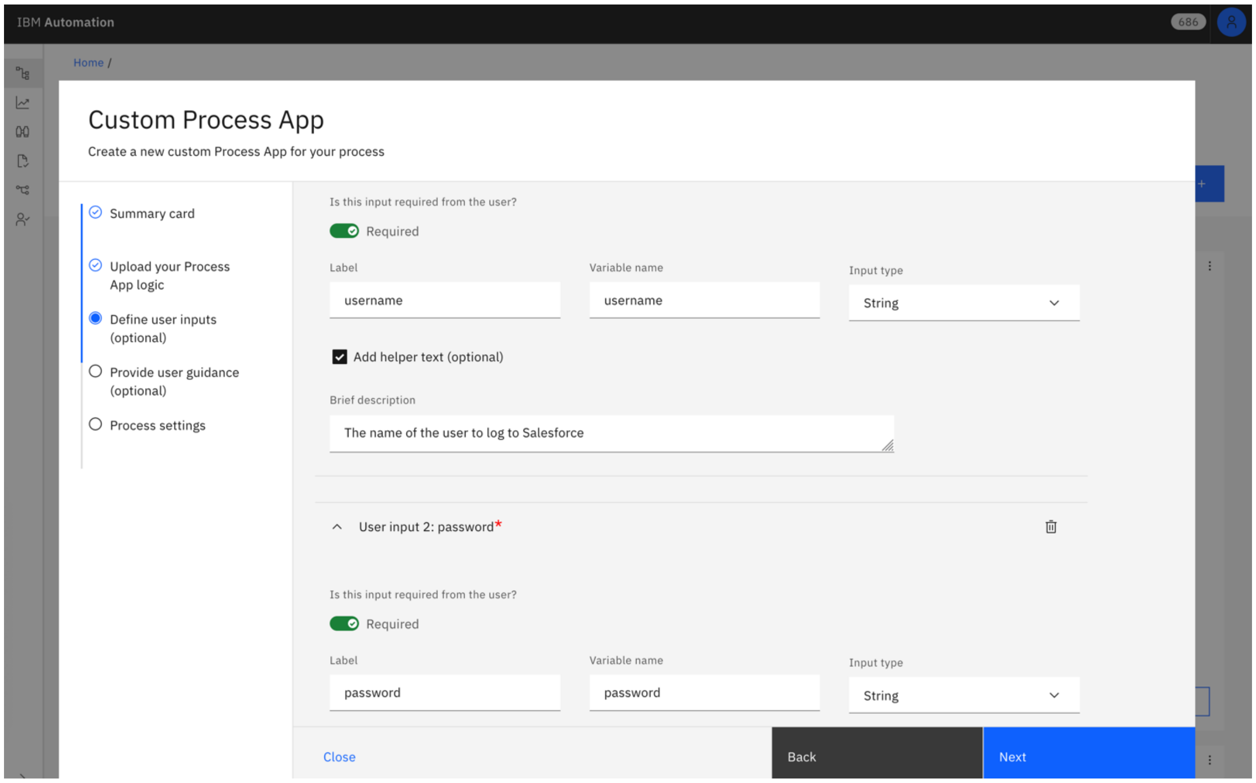
We now have a full python code that can generate an event log from Salesforce with different types of events corresponding to the creation and modification of opportunities. We have seen how to retrieve configuration parameters and how to build and return an event log.

Declaring the process app configuration parameters

Let's go back to the process app builder. In the code, we needed four variables for our process app. The properties need to be declared when we create the process app. This declaration is done in the **Define user inputs** page in the process app builder.

In this section you can declare every individual property by providing a label, a type and a variable name for each property. Note the variable name is the name that we use in the python code.

We set all properties as "Required" because we want the end user to provide those properties and they are not optional.


IBM Automation
686
Home /
Summary card
Upload your Process App logic
Define user inputs (optional)
Provide user guidance (optional)
Process settings
Close
Back
Next

Custom Process App

Create a new custom Process App for your process

Is this input required from the user?

Required

Label: username

Variable name: username

Input type: String

☑ Add helper text (optional)

Brief description: The name of the user to log to Salesforce

User input 2: password*

Is this input required from the user?

Required

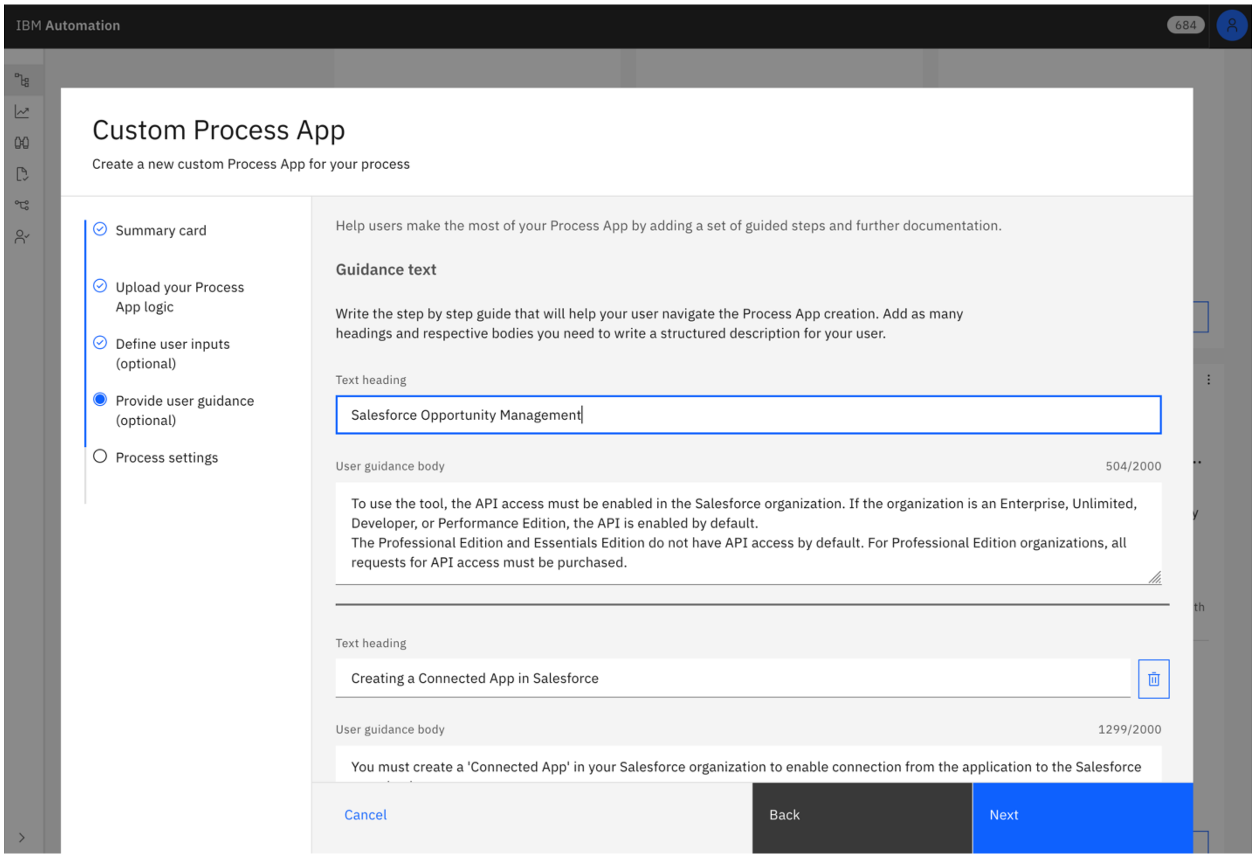
Label: password

Variable name: password

Input type: String

Providing documentation for the process app

The next step of the process app builder is about providing documentation for our process app. In this page you can provide several paragraphs that appear when the user is creating a process with this process app, and an additional URL to link to some additional documentation.


IBM Automation
684
Summary card
Upload your Process App logic
Define user inputs (optional)
Provide user guidance (optional)
Process settings
Cancel
Back
Next

Custom Process App

Create a new custom Process App for your process

Help users make the most of your Process App by adding a set of guided steps and further documentation.

Guidance text

Write the step by step guide that will help your user navigate the Process App creation. Add as many headings and respective bodies you need to write a structured description for your user.

Text heading: Salesforce Opportunity Management

User guidance body (504/2000): To use the tool, the API access must be enabled in the Salesforce organization. If the organization is an Enterprise, Unlimited, Developer, or Performance Edition, the API is enabled by default.
The Professional Edition and Essentials Edition do not have API access by default. For Professional Edition organizations, all requests for API access must be purchased.

Text heading: Creating a Connected App in Salesforce

User guidance body (1299/2000): You must create a 'Connected App' in your Salesforce organization to enable connection from the application to the Salesforce

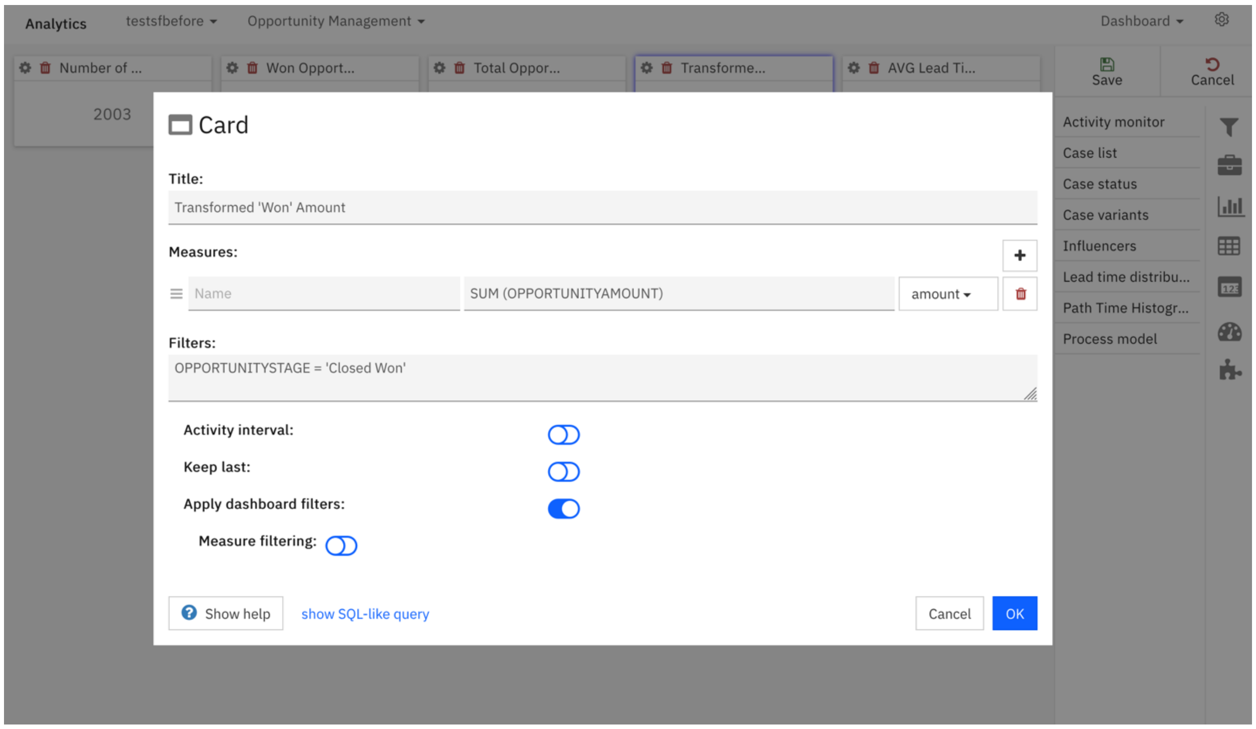
Association of a project backup

We are nearly reaching the end of the process app builder. The last step is to import a project backup. But as we do not have a project yet, we have no backup. One way to prepare a project backup is to create a Process Mining project with some data, even fake data. In our case, we can do this by creating a CSV file with the same columns produced by our process app : 'opportunityId', 'activity', 'startTime', 'resource', 'opportunityAmount', 'opportunityStage'.

With this CSV file, we can create a process mining project, and then create a number of dashboards (see the section Analytics Dashboard Setup) / custom KPI and custom filters that we want to see in the process created by our process app.

A simple example is to create a dashboard widget showing the total opportunity amount.

Here is a simple dashboard card editing showing the creation of a card to display the amount of opportunity that are won.



Title: Transformed 'Won' Amount

Measures: Name | SUM (OPPORTUNITYAMOUNT) | amount

Filters: OPPORTUNITYSTAGE = 'Closed Won'

Activity interval: off

Keep last: off

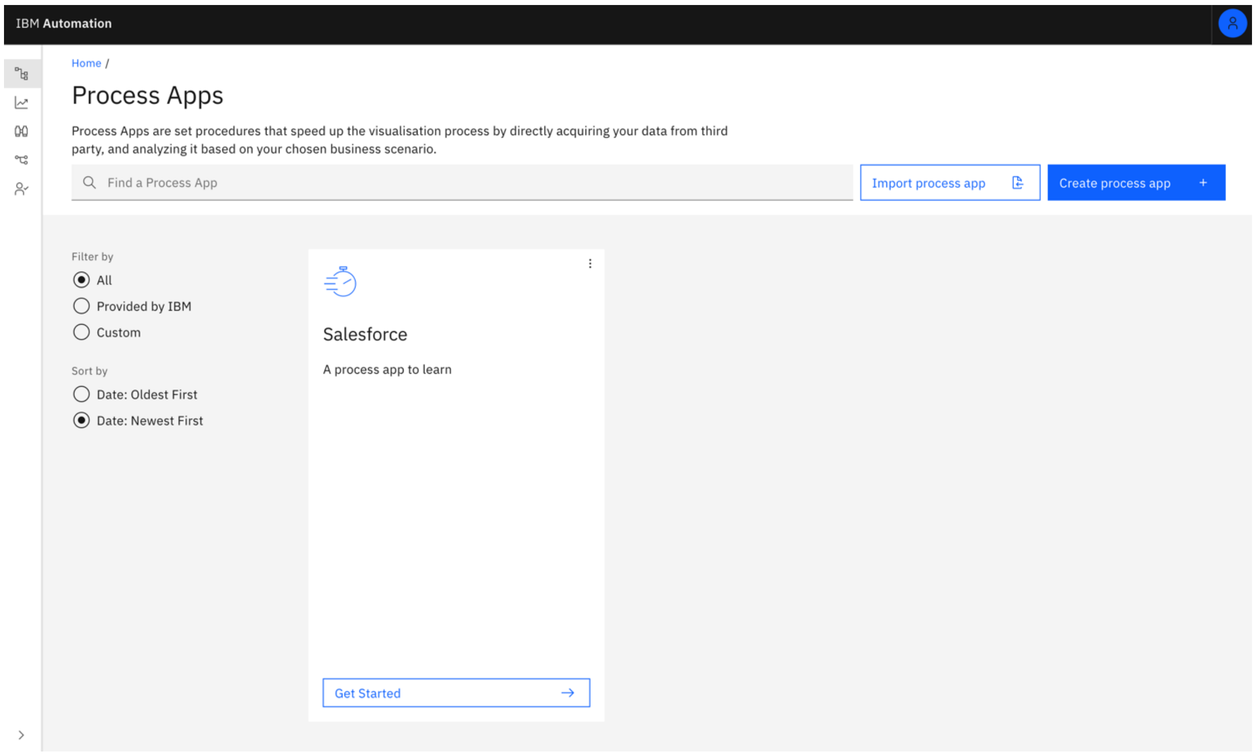
Apply dashboard filters: on

Measure filtering: off

Then we can create a process backup from this project.

Final testing of the process app

When the process app is finally created, it is ready to be used in the list of available process app. It can then be consumed by all business analysts that want to get insights on the salesforce usage.


IBM Automation

Home /

Process Apps

Process Apps are set procedures that speed up the visualisation process by directly acquiring your data from third party, and analyzing it based on your chosen business scenario.

Find a Process App | Import process app | Create process app +

Filter by
- [✓ All]
- Provided by IBM
- Custom

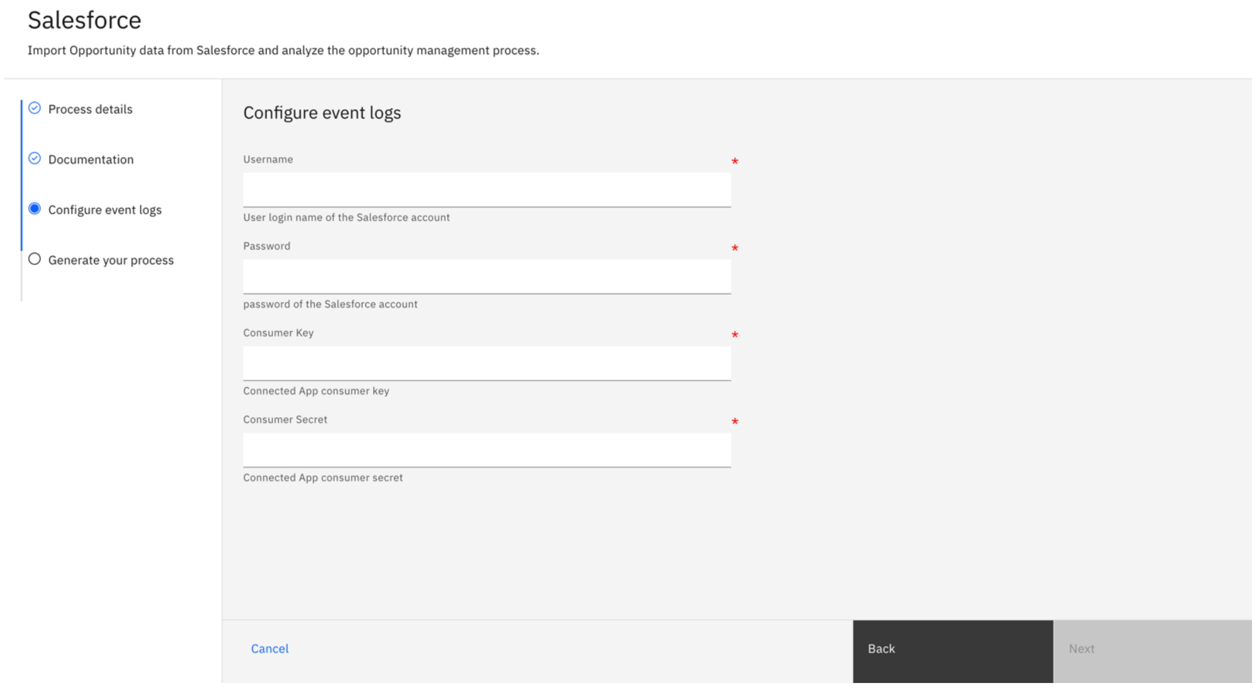
Sort by
- Date: Oldest First
- [✓ Date: Newest First]

Salesforce

A process app to learn

Get Started →

Use the **Get Started** button to run the process app and you are then guided to provide the configuration parameters for your process app:


Salesforce

Import Opportunity data from Salesforce and analyze the opportunity management process.

- Process details
- Documentation
- Configure event logs
- Generate your process

Configure event logs

Username *

User login name of the Salesforce account

Password *

password of the Salesforce account

Consumer Key *

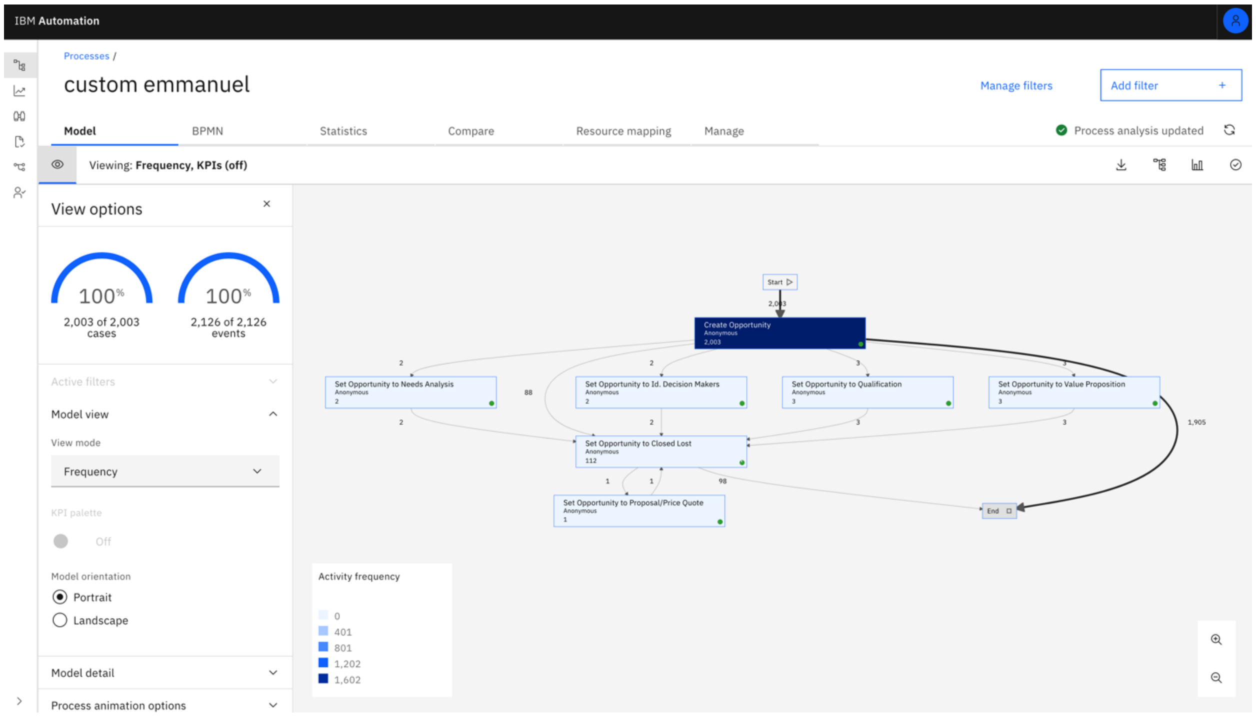
Connected App consumer key

Consumer Secret *

Connected App consumer secret

Cancel | Back | Next

Here is an example of process diagram that is created with this process app.



IBM Automation

Processes /

custom emmanuel

Manage filters

Add filter +

Model | BPMN | Statistics | Compare | Resource mapping | Manage

Process analysis updated

Viewing: **Frequency, KPIs (off)**

View options

100% 2,003 of 2,003 cases

100% 2,126 of 2,126 events

Active filters

Model view

View mode: Frequency

KPI palette: Off

Model orientation: [✓ Portrait] Landscape

Model detail

Process animation options

Start
2,003
Create Opportunity
Anonymous
2,003
2
Set Opportunity to Needs Analysis
Anonymous
2
88
2
Set Opportunity to Id. Decision Makers
Anonymous
2
3
Set Opportunity to Qualification
Anonymous
3
3
Set Opportunity to Value Proposition
Anonymous
3
2
2
3
3
1,905
Set Opportunity to Closed Lost
Anonymous
112
1
1
98
Set Opportunity to Proposal/Price Quote
Anonymous
1
End

Activity frequency

0
401
801
1,202
1,602