Web services best practices

Project name: Web services Best Practices

Package version: 1.1.18

THE BEST RUN SAP

Version	Date	Description
0	February 3, 2021	Document created
1.1	March 08, 2021	Change template Add information about SDK dependencies
1.2	May 18, 2021	Add automations: - GET response buffer and save it as file - Download file
1.3	August 12, 2021	Changing the version of the package

TABLE OF CONTENTS

INTRODUCTION

This document describes the SAP Build Process Automation sample **Web services Best Practices** and provides the following information:

- Description (functional and technical)
- Version used to generate this sample

It also contains information on prerequisites, such as the steps to follow before launching the agent.

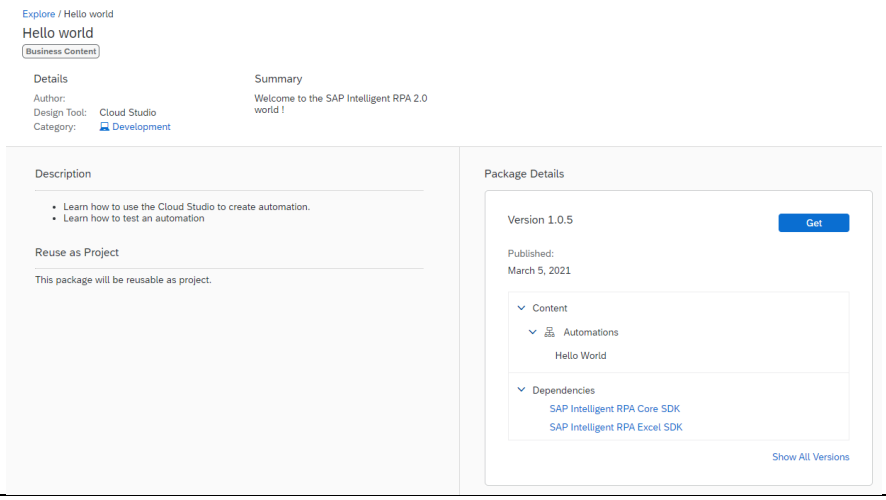
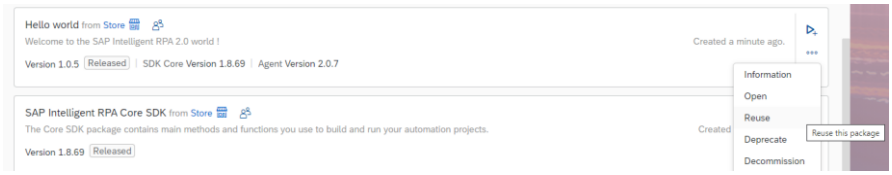
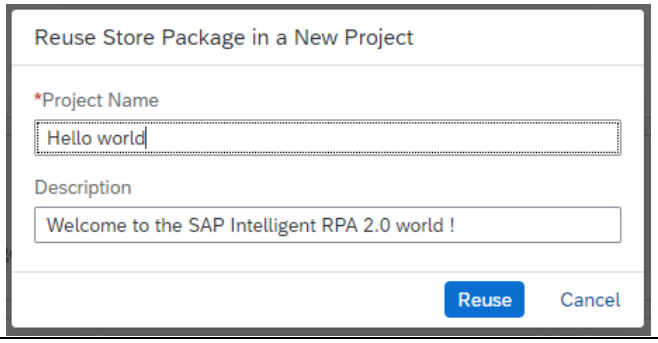
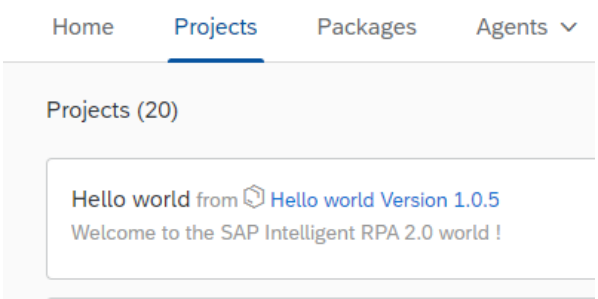
IMPORTANT RECOMMANDATION

General

To use this sample, you need to have a basic knowledge and understanding of SAP Build Process Automation tool. At the very least you need to know how to build an automation, add and modify activities and generate a package.

Reuse the sample as a new project

Note: screenshot might display a different name than the one of this sample.

From the Cloud Factory, open the Store tab and select the sample you want to retrieve. Click on the **Get** button.	
Once the package is retrieved, open the Packages tab of the Cloud Factory. Click on the Options button of the package you just retrieved and select the option **Reuse**.	
Set a name for the project to be created.	
Open the project that has just been created.	
If needed, update the content of this project, and generate a new package from it.	

You need to execute this procedure to be able to open the project and see all its content (the captured applications, the declared items, the automations, etc.).

DESCRIPTION

This package contains captures, datatype and automations that are described below. See chapter Version for more details about the version of the Desktop Agent and the SDK dependencies.

Settings

This section describes the settings of the project such as environment variables or dependent packages that are used in the automation.

Environment variables

Name	Description	Type
BasicAuthPassword	Password used to authenticate to a basic-auth protected endpoint.	Password
BasicAuthUsername	Username used to authenticate to a basic-auth protected endpoint	String
openWeather_appId	This token allows you to use the openweather APIs	Password

Dependent packages

N/A

Captures

This section describes the captures which were made to pilot the application in this sample. It will also describe the different methods which were used to capture the pages and declare the items.

N/A

Datatypes

This section describes the datatype used in this sample. It describes the structure of the datatype and where it is used in the automations.

Web service output

Name of attribute	Type	Description
Code	Number	The result code of the web service call (ex: 200, 201, 404, etc.).
Message	String	The result message of the web service call.
Data	Any	The response body of the web service call.
Headers	Any	The headers of the response of the web service call.

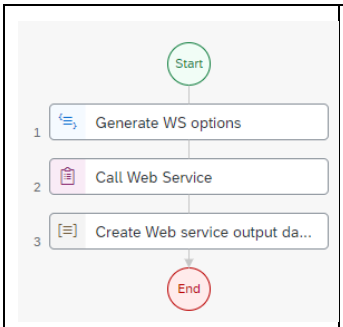
Web service input openweather

Name of attribute	Type	Description
appId	Password	The token you need to provide to be able to use the openweather API.
q	String	The query to perform when calling the openweather web service.

Automations

GET response and headers using URL

Input: The location to send to the service (ex: London)
Output: Web service output



The *Call Web Service* activity requires an object as input parameter. This object is created in the *Custom Script* activity.

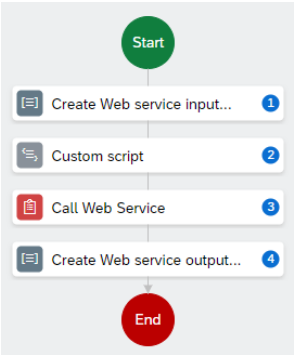
The Custom Script activity contains the following code:

```
return {
    'method': 'GET',
    'url': 'https://api.openweathermap.org/data/2.5/weather?q='
+ city + '&appid=' + token.toString(),
    'responseType':'json', // used to get the body of the response as a JSON o
bject
    'resolveBodyOnly':false // used to get all the response, including headers
};
```

In this case, all the parameters of the web service call are inserted in the query string.

Using *resolveBodyOnly:false* allows to get the whole response, including the headers which might contains token, cookies, etc.

GET response only using URL
Input: the location to send to the service (ex: London)
Output: Web service output



In this automation, the input is parsed to be added in the query string. The *Custom Script* activity contains the following code :

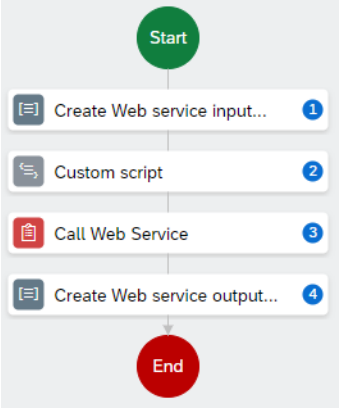
```
let url = 'https://api.openweathermap.org/data/2.5/weather';
let qs = [];

for(let i in input){
    // this check is mandatory as the object might have functions
    // we want to keep the attributes only
    if (input.hasOwnProperty(i)){
        qs.push(i + '=' + input[i].toString());
    }
}

return {
    'method': 'GET',
    'url': url + '?' + qs.join('&'),
    'responseType':'json', // parse the body of the result as a JSON object
    'resolveBodyOnly':true // get only the body of the response
};
```

In this automation the *resolveBodyOnly* attribute now has the *true* value to get only the body of the response. In that case it is not possible to read the headers of the response.

GET response with body in URL
Input: the location to send to the service (ex: London)
Output: Web service output



In this automation, the input is automatically added in the query string.

The *Custom Script* activity contains the following code :

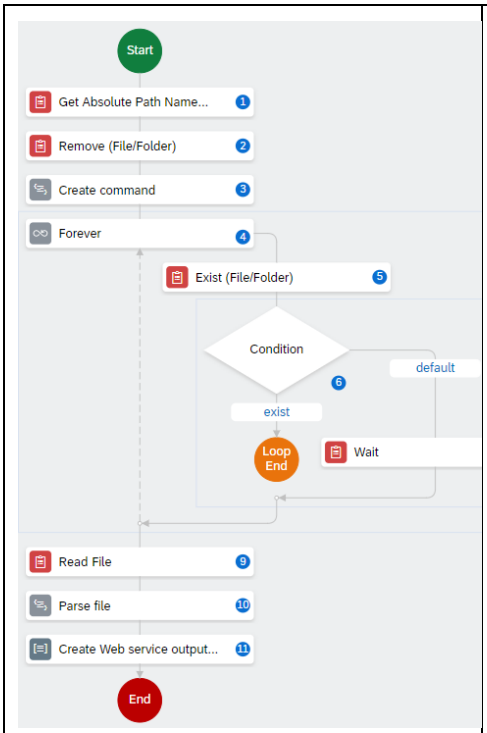
```
// input is an instance of a datatype
// it must NOT be directly assigned to searchParams
// as it contains methods which must not be sent as data
let data = {
    q: input.q,
    appid: input.appId
};

return {
  method: 'GET',
  url:'https://api.openweathermap.org/data/2.5/weather',
  resolveBodyOnly: true,
  responseType: 'json',
  searchParams: data
};
```

GET response with cURL

Input: the location to send to the service, and the path of the folder where the output of the curl command will be generated

Output: Web service output



In some cases, it might be useful to use cURL to execute a web service call. This action might take a while but there is no way to wait for the end of the web service call.

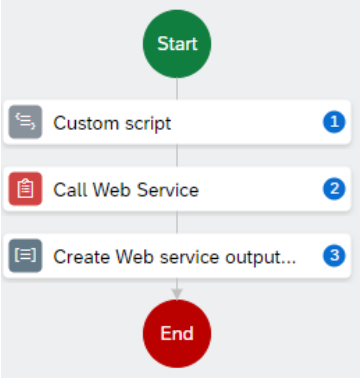
In that case, we ask cURL to redirect the output to a given file and we check if the file exists or not every 2 seconds.

Once the file is generated, we can parse it to obtain an object (which will be readable, etc.)

Note: the execution of the cURL command is done in the Custom Script activity, using the following code:

```
/* params is an array where you can insert all the parameters
of the command line tool cURL */
let params = ['--location'];
params.push('--request GET');
params.push('"https://api.openweathermap.org/data/2.5/weather?q='
+ city + '&appid=' + token.toString() + '"');
params.push('-o ' + outputFile);
/* as irpa_core.core.shellexec does not return the result
of the web service call
 * we need to redirect the write of the result from the standard o
utput to a file.
 * This file can then be read and parsed to get an object.
 */
irpa_core.core.shellexec('curl', params.join(' '), wkDir);
```

POST data with JSON format
Output: Web service output

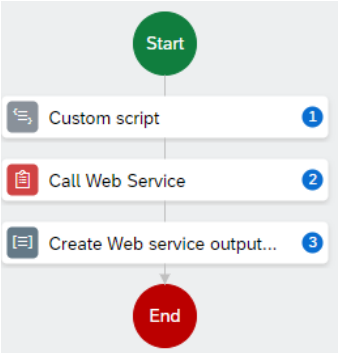


This example presents how to POST data using the JSON format. The content of the *Custom Script* activity is:

```
// input MUST have the JSON notation
let input = {
    "id":42,
    "label":"The id is the Answer"
};

return {
    method: 'POST',
    url: 'https://postman-echo.com/post',
    responseType: 'json', // parse the body of the result to get a JSON obj
ect
    resolveBodyOnly: true, // get only the body of the response
    json: input
};
```

POST data with form
Output: Web service output

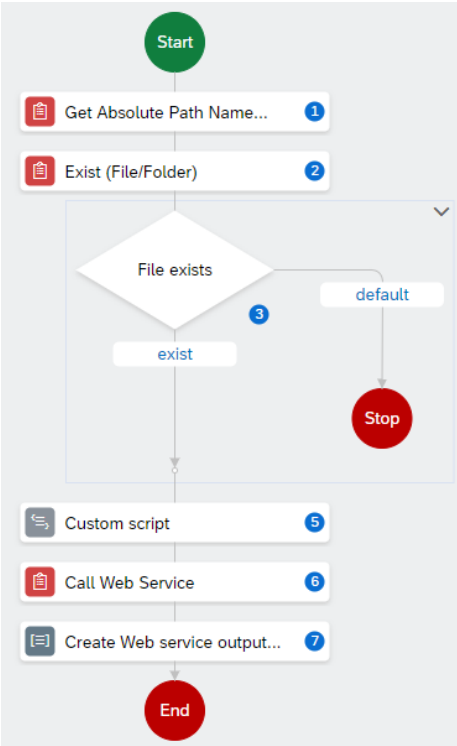


In this example, we send data using POST method. *input* is a JS object.

```
// input is a simple JS object
let input = {
    id:42,
    label:'This is the Answer'
};

return {
    method: 'POST',Ok
    url: 'https://postman-echo.com/post',
    responseType: 'json', // parse the body of the result to get a JSON obje
ct
    resolveBodyOnly: true, // get only the body of the response
    form: input // using the form to send data
};
```

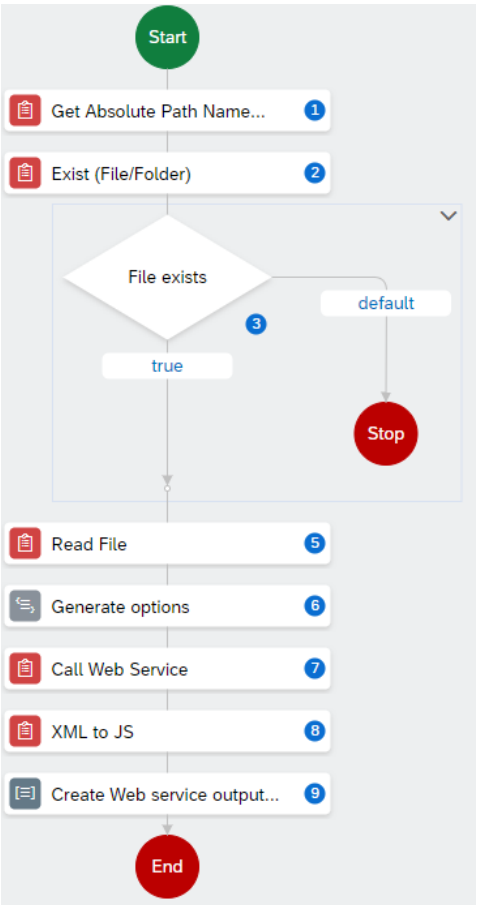
POST data and upload file
Input: Path of the file to be uploaded
Output: Web service output



In this example, we send a file to the endpoint using the POST method:

```
return {
    method: 'POST',
    url: 'https://postman-echo.com/post',
    responseType: 'json', // parse the result to a JSON obj
ect
    resolveBodyOnly: true, // get the body of response only
    headers:{}, // mandatory when using metadata attribute
    metadata:[
        // object of metadata to send file. Type might chan
ge
        // ex: text/plain, application/pdf, etc.
        {
            name:'file',
            file: filePath,
            type:'text/plain'
        }
    ]
};
```

SOAP and convert response to JS
Input: the location of the path with the SOAP request
Output: Web service output



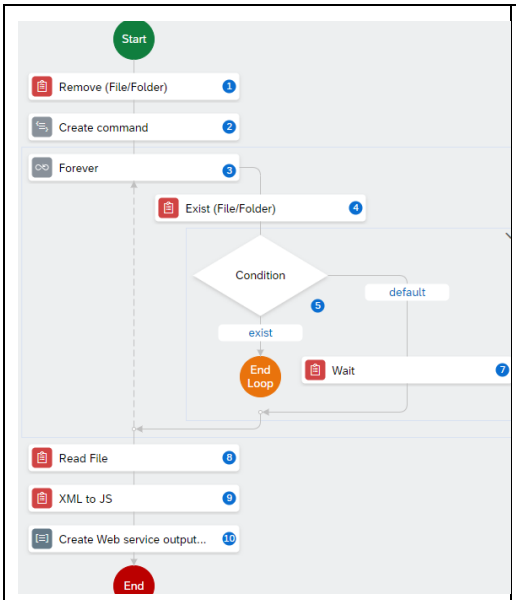
It is easier to call a web service using the SOAP protocol by reading an XML file with the data, instead of writing it directly in the options of the web service activity.

First the agent reads the content of the file, which will be inserted in the body of the request. Once we get the result of the web service call, we need to parse it to get a JS object.

The content of the options is:

```
return {
  method: 'POST',
  url: 'http://www.learnwebservices.com/services/hello',
  headers: {
    // Content-Type is the type of the file (typically xml)
    'Content-Type': irpa_core.enums.request.content.xmlText
  },
  resolveBodyOnly: true, // get only the body of the response
  body: content // data to be sent to web service
};
```

SOAP with cURL
Input: The folder containing the curl command to be executed, and the name of the file containing the command
Output: Web service output



In some cases, it might be useful to use cURL to execute a web service call. This action might take a while but there is no way to wait for the end of the web service call.

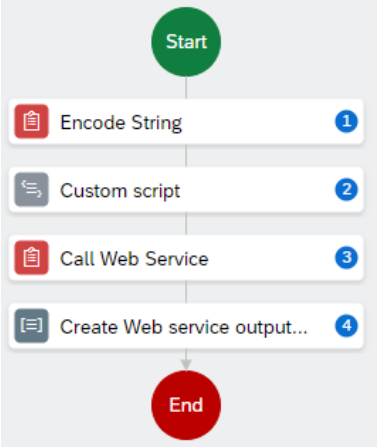
In that case, we ask cURL to redirect the output to a given file and we check if the file exists or not every 2 seconds.

Once the file is generated, we can parse it to obtain an object (which will be readable, etc.)

Note: the execution of the cURL command is done in the Custom Script activity, using the following code:

```
/* params is an array where you can insert all the parameters o
f the command line tool cURL */
let params = ['--location'];
params.push('--request POST');
params.push('"http://www.learnwebservices.com/services/hello
"');
params.push('--header "Content-Type: text/xml"');
params.push('--
data @' + inputFile); //specify the body of the request which i
s stored in a file
params.push('-o ' + outputFile);
/* as irpa_core.core.shellexec does not return the result of
the web service call
 * we need to redirect the write of the result from the standar
d output to a file.
 * This file can then be read and parsed to get an object.
 */
irpa_core.core.shellexec('curl', params.join(' '), wkDir);
```

Authenticate with Basic authentication
Output: Web service output



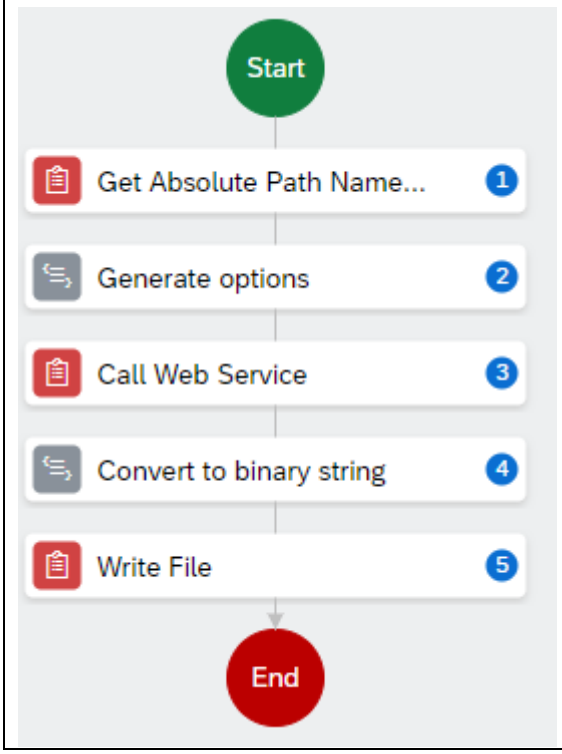
To use the Basic authentication, we first encode the following string:

username:password

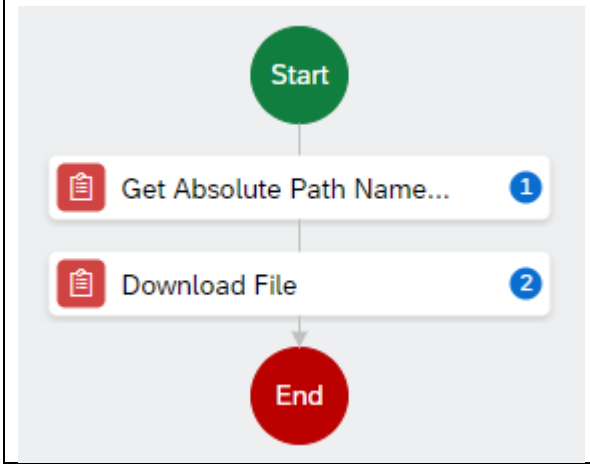
The content of the Custom Script activity is :

```
return {
    method:'GET',
    url:'https://postman-echo.com/basic-auth',
    responseType:'json', // parse the response to get a JSON object
    resolveBodyOnly:true, // get the body of the response only
    headers:{
        // token is the base64 encoded string 'user:password'
        Authorization: 'Basic ' + token
    }
};
```

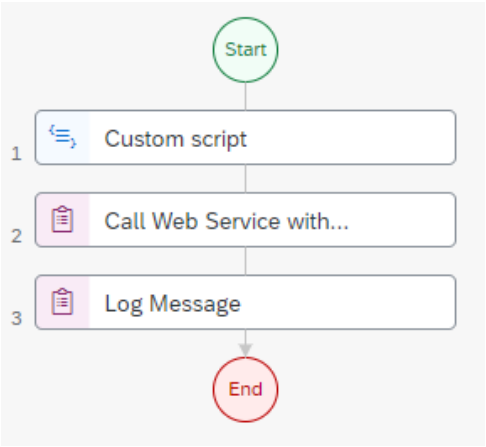
GET response buffer and save it as file
Input: Location where the pdf file will be saved
Output: Web service output

	The agent executes a query to download some raw data. Then it convert it to data which can be written into a file.

Download file
Input:
Output: Web service output

	The agent downloads a file located at a URL passed as parameter and save it to a location specified as parameter.

Call Web Service with Destination
Input: SalesOrder (string)
Output: N/A

	The agent uses a Destination to connect to the service. In the Custom Script, we only provide the path of the query and also the parameters (if needed). We only specify the type of the output.

VERSION

The product versions used to generate this sample are detailed below. This sample is provided “as is”, with no warranty that it will work correctly with other versions. If some versions of your software are different (such as the tool version or the target application version), you may need to recapture the application and/or update the workflow activities.

SAP Build Process Automation

This sample targets the Desktop Agent **2.0.8** or higher.

The following SDK dependencies were used to generate this sample: 1.27.60

See documentation for more details about the compatibility between SDK version and Desktop Agent.

Target application

N/A

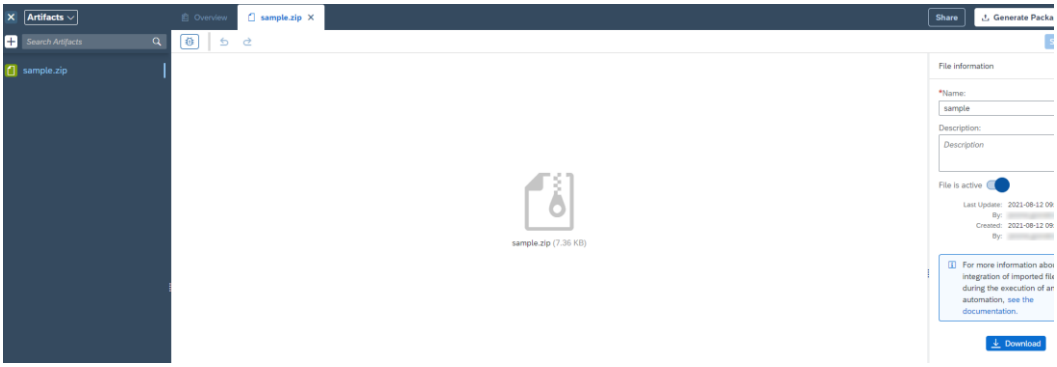
PREREQUISITES

Global setup

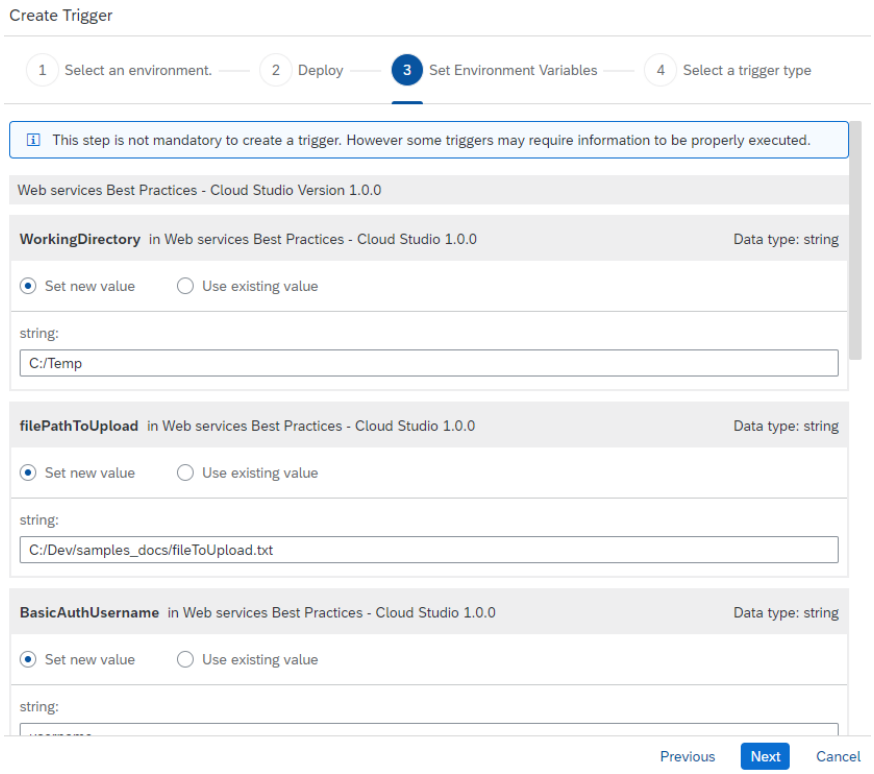
SAP Build Process Automation must be installed in accordance with the installation guide available here. An SAP Build Process Automation Factory must be available with a suitable environment (containing an agent). All information can be found in the “Getting Started” section accessible via the above link.

Specific steps to follow before launching the agent

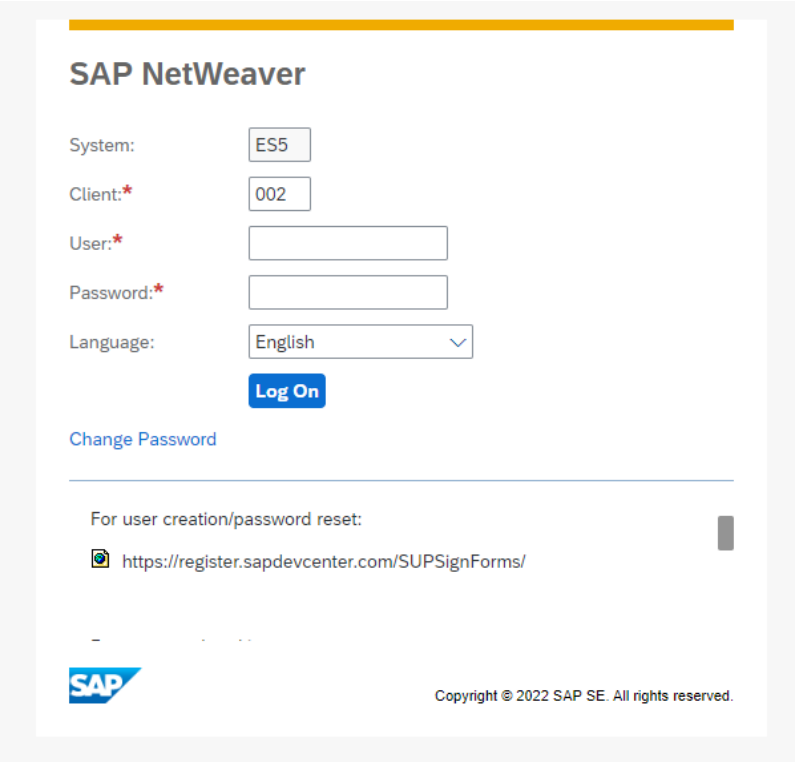
- Open the package and go to the **Files** section
- Download the **samples.zip** archive and extract its content to a given location



- When you deploy the package of this sample, set a value for each environment variable:
  - Set the value **postman** for the variable *BasicAuthUsername*
  - Set the value **password** for the variable *BasicAuthPassword*
  - *openweather_appId* must be the token of your openweather API account (check here)



- Go to https://sapes5.sapdevcenter.com/sap/bc/gui/sap/its/webgui and create a user



- Then you need to create a Destination in BTP Cockpit. Destination should have the following information:

URL	https://sapes5.sapdevcenter.com/
Authentication	BasicAuthentication
User	Your user
Password	Your password

EXPECTED OUTPUT
Each automation returns a **Web service output** datatype instance with all the data sent back by the endpoint selected in the options of the **Web Service Call** activity.

www.sap.com/contactsap

THE BEST RUN SAP®