# Table of Contents

**About the ABBYY FlexiCapture Connector for Pega** .......................................................... 4  
**System requirements** ........................................................................................................... 4  

**ABBYY FlexiCapture as input** .............................................................................................. 4  
  - Architecture .......................................................................................................................... 4  
  - Installation ............................................................................................................................ 5  
  - Configuring export of data from ABBYY FlexiCapture to Pega ........................................... 5  
    - Adding the assemblies to a Document Definition ........................................................... 5  
    - Creating an export profile ............................................................................................... 6  
    - Using a script to specify export settings ....................................................................... 6  
    - Using an XML file to specify export settings ................................................................ 7  
    - Importing the ABBYY FlexiCapture application into Pega ............................................. 7  
    - Creating your own Pega application ............................................................................. 8  
      - Creating a data type ....................................................................................................... 8  
      - Creating a case type ...................................................................................................... 9  
      - Creating a Service REST method ............................................................................... 10  
      - Creating an activity ...................................................................................................... 12  
      - Creating a DELETE method for files ......................................................................... 13  
      - Creating a GET method for cases ............................................................................. 14  

**ABBYY FlexiCapture as a service** ....................................................................................... 16  
  - Architecture ....................................................................................................................... 16  
  - Installation .......................................................................................................................... 17  
  - Configuring ABBYY FlexiCapture ..................................................................................... 21  
    - Configuring ABBYY FlexiCapture as an unattended service ....................................... 23  
    - Creating an export profile ............................................................................................ 25  
  - Configuring Pega ................................................................................................................ 26  
    - Creating a Data Transform record ............................................................................... 26  
    - Creating a new activity and adding it to your workflow ............................................... 28  
    - Creating activity steps .................................................................................................. 31  
  - FlexiCaptureConnector API specification ......................................................................... 36  
    - FlexiCaptureWebServiceClient class .......................................................................... 36  
    - FlexiCaptureProject class ............................................................................................ 39  
    - Credentials class .......................................................................................................... 40  
    - BatchCreationJob class ............................................................................................... 41  
    - RegistrationProperty class .......................................................................................... 42  
    - File class ....................................................................................................................... 42  
    - Result class .................................................................................................................... 42  
    - Stage enum .................................................................................................................... 43  

ABBYY® FlexiCapture® 12 Connector for Pega Administrator’s guide
# Table of Contents

**ABBYY FlexiCapture as a service with verification** ................................................................. 43

- Architecture .......................................................................................................................... 43
- Installation ............................................................................................................................... 44
  - Creating a set of functions for processing and verifying a single document ...................... 50
  - Creating a set of functions for processing and verifying multiple documents ..................... 56
  - Creating general functions ................................................................................................. 65
  - Creating a case with a workflow ......................................................................................... 65
    - Configuring workflow stages .......................................................................................... 66
    - Creating a single-document processing case .................................................................... 69
      - Configuring the user interface ....................................................................................... 74
    - Creating a multi-document processing case .................................................................... 81
      - Configuring the user interface ....................................................................................... 92
  - Creating HTML fragments .................................................................................................. 92
  - Creating a set of activities in Pega .................................................................................... 97
    - Creating a Pega activity .................................................................................................... 97
    - Creating a set of activities for single-document processing case ....................................... 98
      - SendForProcessing ....................................................................................................... 99
      - FetchingCapturedData ................................................................................................. 105
      - CreateError ............................................................................................................... 107
      - VerificationCompleted ............................................................................................... 108
      - Creating a set of activities for multi-document processing case .................................... 109
      - SendForProcessing ....................................................................................................... 110
      - FetchingCapturedData ................................................................................................. 117
      - UpdateVerificationUrl ................................................................................................. 123
      - CreateError ............................................................................................................... 126
      - VerificationCompleted ............................................................................................... 127
  - Creating a set of data transforms in Pega .......................................................................... 127
    - Creating a data transform for a single-document processing case .................................... 128
    - Creating a set of transforms for a multi-document processing case .................................. 129
  - Creating a flow action ........................................................................................................ 133
    - Creating a verification section ........................................................................................ 135
    - Creating a file collection section .................................................................................... 139
  - Creating a custom data type .............................................................................................. 140
    - Data types for a single-document processing case .......................................................... 142
    - Data types for a multi-document processing case ............................................................ 143
  - Configuring your ABBYY FlexiCapture project ................................................................. 144

**Installation** .......................................................................................................................... 44
About the ABBYY FlexiCapture Connector for Pega

ABBYY FlexiCapture enables users to perform high-quality full-text recognition, classify documents, detect fields and capture their data, and convert files into various formats (e.g. searchable PDF).

ABBYY FlexiCapture Connector for Pega is intended for loading document files and captured data into the Pega BPM system.

The ABBYY FlexiCapture Connector for Pega can typically be used in the following two scenarios:

- **ABBYY FlexiCapture as input**
- **ABBYY FlexiCapture as a service**
- **ABBYY FlexiCapture as a service with verification**

System requirements

The ABBYY FlexiCapture Connector for Pega supports:

- **ABBYY FlexiCapture** 12 Distributed 12.0.2.3042 or ABBYY FlexiCapture Cloud;
- **Pega** 8.1.0 or higher.

Export will be performed on the ABBYY FlexiCapture Processing Stations.

To use the **ABBYY FlexiCapture as input** scenario, the ABBYY FlexiCapture Connector for Pega requires Microsoft .NET Framework 4.5 or higher installed on the Pega server and on all of the computers where export will be performed.

To use the **ABBYY FlexiCapture as a service with verification** scenario, the connection between Pega and ABBYY FlexiCapture should be established using the same protocol on both sides (http or https).

**ABBYY FlexiCapture as input**

Architecture

For this scenario, the ABBYY FlexiCapture Connector for Pega includes the following components:

- **DmsConnector.dll**, a library of classes used to convert data from an abstract DMS into ABBYY FlexiCapture data.

- **AfcDmsConnector.dll**, a library accessed by the ABBYY FlexiCapture export script; this library accepts ABBYY FlexiCapture objects as input, converts them, and passes them on to the ABBYY FlexiCapture Connector for Pega.

- **PegaConnector.dll**, a connector for Pega; this library provides a simplified programming interface to interact with the Pega server.

- **Newtonsoft.Json.dll**, a support library; this library is used by PegaConnector.dll and provides functionality to work with JSON strings.
Installation

ABBYY FlexiCapture Connector for Pega is distributed as a ZIP archive that includes all the required libraries and sample export scripts.

Configuring export of data from ABBYY FlexiCapture to Pega

In ABBYY FlexiCapture, documents are processed within projects. ABBYY FlexiCapture will identify the type of each document and apply the appropriate Document Definition that has been prepared specifically for that type. Each Document Definition includes one or more export profiles to be used for exporting document images and data. If you are not familiar with the aforementioned concepts, please refer to the ABBYY FlexiCapture documentation, where you will find a detailed description of the capture process.

Before setting up the connector, please make sure that you have the following software installed:

- ABBYY FlexiCapture 12 Project Setup Station (required if you are using ABBYY FlexiCapture Distributed Edition)
- Administrator Station (if you are using ABBYY FlexiCapture Standalone Edition).

You will also need access rights to Pega.

In the ABBYY FlexiCapture Connector for Pega, you must set up data export independently for each Document Definition.

To set up export of data from ABBYY FlexiCapture to Pega, complete the following steps:

1. Add the assemblies of the ABBYY FlexiCapture Connector for Pega to your ABBYY FlexiCapture Document Definition.
2. Create an export profile in your Document Definition.
3. Import the ABBYYFC application into Pega.

You can also create your own application in Pega or add required functionality to the existing application. For details, see Creating your own Pega application.

Adding the assemblies to a Document Definition

Before you can configure document export, you must add the assemblies to your ABBYY FlexiCapture project. To add the assemblies, complete the following steps:

1. Open your project on the ABBYY FlexiCapture Project Setup Station or the ABBYY FlexiCapture Administrator Station.
2. Click Project → Document Definitions... to open the list of Document Definitions available for the project, select the Document Definition for which you want to specify new export settings, and click Edit....
3. Click Document Definition → Document Definition Properties... to open the properties of the Document Definition and then click the .Net References tab.
4. Click the Add... button on the .Net References tab. In the Add Assembly dialog box, select Attached file from the Type drop-down list, and in the Reference field, specify the path to the DLL file that you wish to add. Repeat this step for the AfcDmsConnector.dll, DmsConnector.dll, PegaConnector.dll, and Newtonsoft.Json.dll files from the ZIP archive with the connector files.

Creating an export profile

ABBYY FlexiCapture exports data based on the export profiles in the Document Definitions.

To create an export profile for Pega:

1. Open your ABBYY FlexiCapture project on the Project Setup Station or on the ABBYY FlexiCapture Administrator Station.

   ▶ Note: Make sure that all the necessary assemblies are added to the Document Definition.

2. Click Project → Document Definitions... to open the list of Document Definitions available for the project, select the Document Definition for which you want to specify new export settings, and click Edit....

3. Click Document Definition → Export Settings....

4. Click the Add... button to start the export profile creation wizard.

5. In the Select the Type of Destination step, select Custom export (script) from the Type drop-down list and then select Errors are irrelevant in the Document condition field. Leave the other settings unchanged or modify them as described in ABBYY FlexiCapture Help.

6. In the Script Export step, click the Edit Script... button and paste into the script editor one of the two sample scripts provided in the ZIP archive with the connector files, either the named "Pega - ExportScript - Configure by script.cs" or the one named "Pega - ExportScript - Configure by XML.cs." For detailed instructions on using each of these scripts, see Using a script to specify export settings and Using an XML file to specify export settings below. You can always edit this script later, but for the moment, save the changes and close the script editor. Detailed instructions for setting up your export script will be provided below.

7. In the Select the Destination Name step, specify a name for your export profile and click Finish.

8. The newly created export profile will appear in the list of available export profiles in the export settings dialog box. To make ABBYY FlexiCapture use this profile on Export stage, select the box next to the profile in the Enabled column.

9. Click OK, then save and close the Document Definition editor. Next, click the Publish button to publish your Document Definition.

   ▶ Note: If documents are being processed within the project at the time when you add your new export profile, update them to the latest version by pressing Alt+Shift+U or by selecting Update to Latest Version from their shortcut menu.

Using a script to specify export settings

One advantage of using a script is that all the export settings will be automatically used by all the ABBYY FlexiCapture stations from which data are exported to Pega. When using an XML file, you can only specify static settings, whereas scripts allow you to specify different file naming rules, file locations, and field mappings depending on the data contained in the exported document.
When working with ABBYY FlexiCapture Cloud, you also need to use a script to specify export settings, since ABBYY FlexiCapture Cloud does not provide the ability to store an XML file with configuration on processing stations.

However, to modify the export settings, you will have to make changes to the script and, consequently, publish a new version of the Document Definition.

Please use the script named "Pega - ExportScript - Configure by script.cs" from the ZIP archive that contains the connector files.

Using an XML file to specify export settings

One advantage of using an XML file is that you don’t have to edit the script code or the Document Definition. All modifications can be made by editing the XML file, which is much simpler than editing the script code. However, the XML file should be located on every ABBYY FlexiCapture processing station. And you will also have to synchronize it manually on all of the ABBYY FlexiCapture stations from which data is to be exported to Pega.

To specify export settings in an XML file:

1. As your export script, use the file named "Pega - ExportScript - Configure by XML.cs" from the ZIP archive containing the ABBYY FlexiCapture Connector for Pega files.

2. Create an XML file named "<Project name>_<Document Definition name>.xml." This naming scheme will allow you to have multiple XML files for different Document Definitions.

3. In the registry of the computer with a processing station, locate HKEY_LOCAL_MACHINE\SOFTWARE\ABBYY\FlexiCapture\12.0\Connectors\Pega and in its "XmlFolder" string value field, specify the path to the folder where the settings file is to be stored.

4. In the registry of the computer with a processing station, locate the HKEY_LOCAL_MACHINE\SOFTWARE\ABBYY\FlexiCapture\12.0\Connectors\ and set its "Pega" string value field to true.

Note: For an example of an XML configuration file with detailed comments, please see the file named "default.config.xml" that is provided with the ABBYY FlexiCapture Connector for Pega.

Importing the ABBYY FlexiCapture application into Pega

The connector's installation package includes a Pega application named ABBYYFC that will accept files and data. To import into Pega the ABBYY FlexiCapture application provided with ABBYY FlexiCapture Connector for Pega, complete the following steps:

1. Open Dev Studio and click Configure → Application → Distribution → Import to launch the Import Wizard.

2. In the Import Wizard, click Browse... and browse to the ZIP archive containing the ABBYYFC application.

3. Select the Enable advanced mode to provide more granular control over the import process option and click Next.

4. To allow the wizard to apply the schema changes automatically, select the Automatic option and click Next.
5. Wait until the application schema is imported into the database and click Next.

6. The Import Wizard will consecutively display the objects previously imported into Pega, a list of new objects that are about to be imported, and the components that will be updated as a result of the import operation. Review the changes and click Next.

7. Wait until the application is installed and click Done.

8. Click Records → Organization → Operator ID and select the user named Admin.ABBYYFC. Click the Security tab, disable the Disable Operator option, and click Save.

9. Repeat step 8 for the user named User.ABBYYFC.

Use the following credentials to access the imported ABBYYFC application:

- Username: Admin.ABBYYFC. Password: rules (to log in is as an administrator)
- Username: User.ABBYYFC. Password: rules (to log in as a user)

Note: If you need to make the application accessible to another user, click Records → Organization → Operator ID and select that user. In the Application Access section, add an Access Group for the ABBYYFC application (e.g. ABBYYFC:Administrators).

Creating your own Pega application

You can create your own Pega application to interact with ABBYY FlexiCapture Connector for Pega. It should contain:

- a case for storing data;
- a web service and an activity for loading files and data into cases.

To create a new Pega application:

1. Log in to Pega as an administrator.

2. Open Dev Studio and click the New Application button on the upper dashboard of the home page.

3. Choose Custom as the type of your application an click the Use this application type button.

4. Type in a name for your application. Clicking Advanced configuration button opens the advanced settings page, where you can change your application ID, organization settings, and class settings. When you are done, click Save to save your changes and then click the Create application button.

5. Click Done if you want to return to the PegaSamples application or click Log Off if you want to start working with the new application.

Creating a data type

If your case will handle table values (e.g. invoice line items), you need to create a custom data type.

To create a new data type:
1. Open Dev Studio, click the **Data types** tab on the left, click the Explorer menu at the top, and select **Add data type**.

2. In the **Add Data Type** dialog box, type **Line Items** in the **Label** field and click **Submit**.

3. On the **Data model** tab, add the fields that will make up the line items. When you are done, click **Save** to save your changes.

### Creating a case type

Your new case will store document data and document originals exported from ABBYY FlexiCapture.

To create a new case type:

1. Once you have created your Pega application, click the **Cases** tab on the left and then click **Add a case type**.

2. In the **Add case type** dialog box, type in a name for your case type and click **Submit**.

3. On the **Data model** tab, add the fields to be handled by your case.

**Note:** If you are planning to work with table data (e.g. invoice line items), you must also add a field of type **Field group (list)**. In the **Options** column, specify the **data type** you created earlier for this type of data.

4. Click **Save** to save the newly created case.

5. If you need to change the data displayed in your case in different views, click the **Views** tab. On this tab:
   - **Create** is the view that you see when you create your case
   - **Review** is the view that you see when you review your case
   - **Edit** is the view that you see when you edit your case
   - **Case attachments** is the view that you see when you display or add attached files
   - **Case details** provides information about your case

6. Click **Submit**.

7. Specify the full name of the case in the **Templates** field of the ABBYY FlexiCapture Connector for Pega configuration file.

**Note:** The full name of the case is made up of the full name of the application and the name of the case type and has the format "Organization-Name of application-Class-Name of case type." To find out the full name of the application, click the **App** tab on the left.

### Adding the ability to delete the case

To be able to delete the case, you need to add a **Delete** button to its UI. Complete the following steps:

1. Click the **App** tab on the left. Then click the name of the case type to expand it and navigate to **User Interface → Section → pyCaseInformation**.
2. On the Design tab, click Add new and add a new Button control.

3. Click the gear icon next to the newly added Button control to open its properties. On the General tab, replace the default text in the Button caption field with Delete.

4. While still in the Cell Properties dialog box, click the Actions tab and then click Create an action set. Under Action set 1, click Add an event and select Click. Then click Add an action → All actions and select Run activity from the Other category. In the Activity field, type DeleteCase as the name for the new activity. Then open the new activity and click Create and open.

5. On the Edit Activity form, click the Parameters tab and add a new CaseHandle parameter of type String. Then click the Steps tab, add the Obj-Delete-By-Handle method, and select the boxes next to Lock, ReleaseOnCommit, and Immediate. In the value field of the InstanceHandle parameter, type Param.CaseHandle. Click Save to save your settings.

6. Once the activity is created, type the .pzInsKey in the CaseHandle field. Next, click Add a condition and add a Cancel action. In the Transition field, select Go home (which returns the user to the list of cases) and click Submit.

7. In the case editor, click Save to save your changes.

The newly created Delete button will appear in the case UI. Clicking this button will delete the case and close its page. To update the list of cases, click Refresh.

Creating a Service REST method

First, you need to create a service package:

1. Click the Records tab on the left, expand Integration Resources, right-click Service Package, and select the Create command from the shortcut menu.

2. On the Create Service Package form, type customapi in the Service Package Name field. Provide a description for the package in the Short description field and click Create and open.

3. On the Edit Service Package form, specify an access group in the Service access group field (the format of the group is <Application_name>:<Access group>).

Note: To view the list of available access groups, click the Records tab on the left and navigate to Security → Access Group.

4. Click Save.

Next, you need to add a new role to the access group to enable users to access the Web API:


2. In the Available roles field, add a PegaRULES:PegaAPI role.

Now you need to create a new service:

1. Click the Records tab on the left, expand Integration-Services, right-click Service REST, and select the Create command from the shortcut menu.
2. On the form that opens, specify the following:

- Service name: `files`
- Service package name: `customapi`
- URI Template: `cases/{id}/files/{name}`
- Service version: v2

3. Click **Create and open**.

4. Click the **Service** tab and type `Pega-API-CaseManagement` in the **Pega class** field. In the **URL path parameters** section, type `Param.CaseId` in the **Map to Key** column for the **Id** parameter. Then type `Param.FileName` for the **name** parameter.

5. Click the **Methods** tab. Expand the POST method and on the **Request** tab, specify the following for the **Message data** fields:

- Description: `Request Data`
- Map to: `Clipboard`
- Map to key: `Param.Request`

6. Click the **Response** tab and create three response conditions:

<table>
<thead>
<tr>
<th>Process what</th>
<th>Parameter values</th>
<th>Header fields</th>
<th>Message data</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxIsInternalError</td>
<td>• Condition: <strong>When</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• When name: <code>pxIsInternalError</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Content type: <code>application/json</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Status code: <code>Param.StatusCode</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pxIsErrorResponse</td>
<td>• Condition: <strong>When</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• When name: <code>pxIsErrorResponse</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Content type: <code>application/json</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Status code: <code>Param.StatusCode</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Success response</td>
<td>• Condition: <strong>Default</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Name</strong>: <code>location</code>; <strong>Map from</strong>: <code>Clipboard</code>; <strong>Map from key</strong>: <code>Param.Location</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Description</strong>: <code>Response</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Map from</strong>: <code>Clipboard</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11
<table>
<thead>
<tr>
<th>Process what</th>
<th>Parameter values</th>
<th>Header fields</th>
<th>Message data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Content type:</td>
<td>Name: etag;</td>
<td>Map from key:</td>
</tr>
<tr>
<td></td>
<td>application/json</td>
<td>Map from:</td>
<td>Param.eTag</td>
</tr>
<tr>
<td></td>
<td>Status code:</td>
<td>Clipboard;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Param.StatusCode</td>
<td></td>
</tr>
</tbody>
</table>

**Important!** Make sure that the URL of the service method is as follows: https://Server_Name/prweb/PRRestService/customapi/v2/cases/{id}/files/{name}. Otherwise, the connector will return an error when the user attempts to export a file to Pega.

**Creating an activity**

To create a new activity:

1. In the POST request of the service, type a name for your new activity (e.g. UploadFileToCase) in the **Activity name** field.

2. Click **Create and open** to configure the new activity.

3. On the **Security** tab, add the following classes that will be accessed by the method and the activity: Pega-API-CaseManagement and Work-.

4. On the **Pages & Classes** tab, declare the classes that will be used by the activity:
   - AttachmentPage of type Data-WorkAttach-File
   - pyWorkPage of type Work-

**Note:** In Pega, a page corresponds to a class object.

5. On the **Parameters** tab, add the parameters that will be used by the activity.
   - Parameters:
     - FileStream of type String – This parameter will contain the contents of the file in Base64.
     - Local variables:
       - IsConverted of type Boolean – The logical variable that will contain the result of executing the method that converts the JSON object in the POST request into global variables of myStepPage.

6. On the **Steps** tab, create new steps for methods that will load objects and link the case to its file.
   - To load the JSON object from the POST request into the activity parameters:
     - In the **Method** field, type this method name: Property-Set
     - In the **Step page** field, type .createCaseRequest
     - Convert the result of the @((Pega-RULES:Page).pxConvertStringToPage(tools, myStepPage, Param.Request,"json")) method to the Local.IsConverted variable. The value
is true if the JSON object from the POST request is successfully converted to the myStepPage object. Otherwise, the value is false.

- Once the myStepPage object has been populated, populate the activity variables: Param.FileStream = .FileStream. To add new properties to myStepPage, click the target icon on the right and then click Create and open on the Create Property form.

- To get the object containing the case by its ID:
  - In the Method field, type pxGetCase: Call pxGetCase to call the activity.
  - In the Step page field, type the name of the object that will store the pyWorkPage result. In the WorkObjectId parameter, type Param.CaseId (i.e. the case ID).

- To create a new empty object named AttachmentPage that was declared on the Pages & Classes tab:
  - In the Method field, type Page-New.
  - In the Step page field, type AttachmentPage.
  - Leave the method parameters empty.

- For the parameters of the AttachmentPage object:
  - The .pxAttachName parameter must be populated with the value from Param.FileName, the .pxAttachStream parameter must be populated with the value from Param.FileStream, the .pxAttachKey parameter must be populated with the value from @getCurrentTimeStampUnique(), and the .pxRefObjectKey parameter must be populated with the value from pyWorkPage.pzInsKey.
  - In the Method field, type Property-Set.
  - In the Step page field type AttachmentPage.

- Use the Odj-Save method to save the AttachmentPage object.

- Use the Link-Objects method to link the pyWorkPage object (i.e. the case that we got by using pxGetCase: Call pxGetCase) to the AttachmentPage object. In the LinkToPage parameter, type AttachmentPage; in the LinkClass parameter, type Link-Attachment; and in the LinkMemo parameter, type Param.FileName.

- Use the Commit method to commit the changes to the database.

- If necessary, write to Param.Response the pyWorkPage – @(Pega-RULES:Page).pxConvertPageToString(tools, pyWorkPage, "json") object received in the response from the service.

## Creating a DELETE method for files

In Pega, you can create a Service REST containing both POST and DELETE methods. To create a DELETE method:
1. Open the previously created POST method by clicking Records → Integration-Services → Service REST → cases/{id}/files/{name} and then open the Methods tab.

2. Expand the DELETE method and open the Response tab.

3. Create three conditions identical to the conditions you created on the Response conditions tab of the POST method.

Next, you need to create an activity named DeleteFiles:

1. In the Activity name field, type DeleteFiles, then click the target icon on the right, and then click Create and open on the Create Property form.

2. On the Pages & Classes tab, create a new page named pyWorkPage of the Work- class.

3. On the Parameters tab, create three variables:
   - currentLinkedRefTo of type String
   - IsConverted of type Boolean
   - currentFileId of type String

4. On the Steps tab, complete the following steps:
   a. Open and load your case into the pyWorkPage page using the Obj-Open-By-Handle method. Use Param.CaseId parameter as the case identifier (this parameter can be taken from the /cases/{id}/files/{name} request).
   b. Delete all files from the case attachment. To do this, type Property-Set in the Method field, then open Loop, select For each element in value list, and type pyWorkPage.pyAttachments in the Value List Property field. Add the following two new parameters to the method:
      Local.currentLinkedRefTo – pyWorkPage.pyAttachments(<CURRENT>).pxLinkedRefTo;
      Local.currentFileId – pyWorkPage.pyAttachments(<CURRENT>).pzInsKey;
   c. Create a Call DeleteAttachment activity call and in the Target field and specify pyWorkPage. Set the WorkPageName parameter to Local.currentLinkedRefTo and the LinkHandle parameter to Local.currentFileId.
   d. In the When field, specify the following condition: if Param.FileName = pyWorkPage.pyAttachments(<CURRENT>).pyFileName, delete the files; otherwise, skip this step.

Creating a GET method for cases

To create a GET method for Cases:

1. Create a new Service REST (click Records → Integration Resources, right-click Service Package, and then click the Create command on the shortcut menu).

2. On the form that opens, specify Cases as the name of the service, customapi as the name of the package, v2 as the version of the package, and cases as the name of the URI template and click Create and open.
**Note:** After you have created the service, make sure that the URL of the service method is as follows: `https://Server_Name/prweb/PRRestService/customapi/v2/cases`.

3. On the **Methods** tab, expand the GET method.

4. Click the **Response** tab and create two response conditions:

<table>
<thead>
<tr>
<th>Process what</th>
<th>Parameter values</th>
<th>Header fields</th>
<th>Message data</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxlsInternalError</td>
<td>• Condition: <em>When</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• When name: <code>pxlsInternalError</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Content type: <code>application/json</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Status code: <code>Param.StatusCode</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Success response</td>
<td>• Condition: <em>Default</em></td>
<td>• Name: <code>etag</code>; Map from: <code>Clipboard</code>; Map from key: <code>Param.eTag</code></td>
<td>• Description: <code>Response</code></td>
</tr>
<tr>
<td></td>
<td>• Content type: <code>application/json</code></td>
<td>• Name: <code>cache-control</code>; Map from: <code>Constant</code>; Map from key: &quot;no-cache&quot;</td>
<td>• Map from: <code>Clipboard</code></td>
</tr>
<tr>
<td></td>
<td>• Status code: <code>Param.StatusCode</code></td>
<td></td>
<td>• Map from key: <code>Param.Response</code></td>
</tr>
</tbody>
</table>

After you have configured the method, you need to create a new activity named **GetCases**:

1. On the **Parameters** tab, create three variables:
   - result of type `String`
   - currentCaseID of type `String`
   - tempResult of type `String`

2. On the **Steps** tab, complete the following steps:
   a. Get a list of all the cases available to the current user. To do this, call `Call pzGetCasesByFilter` and select the **Pass current parameter page** option. `pzGetCasesByFilter` will write all cases to `myStepPage`.
   b. Create a step with a **Property-Set** method that will return cases from `myStepPage` in a loop. To do this, add a loop, and for this loop select the **For each element in value list** and specify the `myStepPage.cases` property. In the method parameters, set the Param.currentCaseID variable to `myStepCase.cases(<CURRENT>).ID`.
   c. Create a step that will get the details of the case. To do this, type `Call GetCaseDetails` on the **Method** tab and then type `myStepPage.cases(<CURRENT>)` in **Step Page**. Set the `caseID` parameter to `Param.currentCaseID`.

**Note:** Instructions for creating a GetCaseDetails activity are provided [below](#).
d. Convert the result obtained in the previous step into a string and write it to the Param.tempResult variable.

e. To generate a valid JSON string, add a comma to the Param.result parameter (i.e. \textit{Param.result + ","}) in the \textit{When} field, add the following two conditions:

\begin{itemize}
  \item If @IsParamBlank(tools,"result") is \texttt{true}, Skip Step; if the parameter is \texttt{false}, Continue Whens.
  \item If @IsParamBlank(tools,"tempResult") is \texttt{true}, Skip Step; if the parameter is \texttt{false}, Continue Whens.
\end{itemize}

f. Check if \texttt{tempResult} is empty. If it is not empty, add the value from \texttt{tempResult} to the main result (i.e. \texttt{Param.result + Param.tempResult}).

g. Create the final response \texttt{"["+Param.result+"]"} and write it to the Param.Response variable.

To create a \texttt{GetCaseDetails} activity:

1. On the \texttt{Parameters} tab, create three variables:

\begin{itemize}
  \item caseID of type \texttt{String}
  \item fileName of type \texttt{String}
  \item listFileName of type \texttt{String}
\end{itemize}

2. Open the \texttt{Pages & Classes} tab and add a page named \texttt{Case} of the \texttt{Work-} class.

3. On the \texttt{Steps} tab, complete the following steps:

\begin{itemize}
  \item In the \texttt{Method} field, type \texttt{Obj-Open-By-Handle}. In the \texttt{Step page} field, add \texttt{Case}. In the \texttt{InstanceHandle} parameter, type \texttt{Param.caseID}.
  \item Create a new \texttt{.attachments} property and populate it with the \texttt{Case.pyAttachments} list. To create an attachment property, click the target icon on the right, and then click \textbf{Create and open} on the \textbf{Create Property} form. Next, select \texttt{Page List} as the property type and type \texttt{Link-Attachment} in the \texttt{Page definition} field.
  \item Populate the page with content values. To the \texttt{Method} field, write \texttt{Page-Copy}; to the \texttt{CopyFrom} field, write \texttt{Case}; and to the \texttt{CopyInfo} field, write \texttt{.content}.
\end{itemize}

**ABYY FlexiCapture as a service**

**Architecture**

For this scenario, the ABBYY FlexiCapture Connector for Pega includes a JAR file which allows you to access the classes used for working with the ABBYY FlexiCapture 12 Web API.

This connector module runs on the Pega side, where it sends files to ABBYY FlexiCapture for processing and gets back processing results.
Installation

To install the ABBYY FlexiCapture Connector for Pega, you need to import the connector to the server and create a Java function in Pega.

To import the connector to the server:

1. Open Dev Studio and click Configure → Application → Distribution → Import.

2. Load the FlexiCaptureConnector.jar file on to the server.

3. Open the Pega Self-Service Portal and click Restart Server.

To create a Java function that will call methods from connector classes:

1. Create a new library by clicking Records → Technical → Library → Create and name it, say, FlexiCaptureLibrary. Then include the following packages:

   - java.util.*
   - java.nio.file.*
   - com.abbyy.connectors.*
   - com.abbyy.connectors.models.*
2. Click **Save** followed by **Generate Library**.

3. Create a new function by clicking **Records → Technical → Function → Create**.

4. In the **Label** field, provide a name for the new function (e.g. **CaptureData**).

5. In the **Parameters** section, specify the following two input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>String</td>
</tr>
<tr>
<td>Base64Content</td>
<td>String</td>
</tr>
</tbody>
</table>

6. In the **Library** field, enter the name of the library you created in step 1 and click **Create and open**.

7. In the window that opens, on the **Parameters** tab, specify **String** as the data type in **Java data type** field:
Copy the code from the CodeExample.java file, which you can find in %Installation Path%\Connector for FlexiCapture as a Service\Samples\Pega Function (for your convenience, the code is reproduced below). While still in the same window, click the Java tab and paste the code you have just copied. This sample code sends files to ABBYY FlexiCapture and gets back the processing results in the form of JSON strings with the names of the captured fields and their values. The code fragments highlighted in yellow should be replaced with valid connection data required to connect to the ABBYY FlexiCapture Application Server. Then upload your project to the ABBYY FlexiCapture Application Server and configure it.

```java
// Creating FlexiCapture Web Services API client. Please provide FlexiCapture Application Server address here.
try (FlexiCaptureWebServiceClient client = new FlexiCaptureWebServiceClient("https://FlexiCaptureApplicationServer")) {

    // Please uncomment the line below and provide a company name to interact with FlexiCapture Cloud or tenant.
    // client.setTenant("Company");

    // Please provide valid FlexiCapture user credentials.
    client.setCredentials(new Credentials("Username", "Password"));

    // Getting the FlexiCapture project. Please provide a project name here.
    try (FlexiCaptureProject project = client.getProject("InvoiceDemoProject")) {
```
// Preparing a file to upload to FlexiCapture. The 'FileName' and 'Base64Content'
// are Pega function parameters.
List<File> files = new ArrayList<File>();
files.add(new File(FileName, Base64.getDecoder().decode(Base64Content.replace("\n", "\n").replace("\r", "\r"))));

// Preparing a job for a FlexiCapture batch processing.
BatchCreationJob job = new BatchCreationJob(files);
job.setName("Pega_" + System.currentTimeMillis());  // Setting FlexiCapture batch name.
job.setType("Default");  // Setting FlexiCapture batch type name.

// FlexiCapture batch creation and run.
int batchId = project.create(job);

// Waiting for the batch to be processed and getting the processing results.
Result result = project.getResult(batchId);

// Check if the batch stage is Processed.
if (result.getStage() == com.abbyy.connectors.models.Stage.Processed) {

    // Returning captured fields as a JSON string.
    return result.getFields();

} else {

    // Throwing an exception other stages.
    // Instead of the throwing, processing of these stages can be configured here.
    throw new FlexiCaptureConnectorException(String.format("Unsupported
FlexiCapture stage type 's'.", result.getStage()));

}
9. Click the **Imports & Exceptions** tab and type *Exception* in the **Exceptions thrown** section.

10. Go back to the **Java** tab, select the **Function ready to be compiled?** option, save your changes, and click **Generate function**. If the function is generated successfully, you will see this message:

```
Configuring ABBYY FlexiCapture

In ABBYY FlexiCapture, documents are processed within projects. ABBYY FlexiCapture will identify the type of each document and apply the appropriate Document Definition that has been prepared specifically for that type. Each Document Definition includes one or more export profiles to be used for exporting document images and data. If you are not familiar with the aforementioned concepts, please refer to the [ABBYY FlexiCapture documentation](#), where you will find a detailed description of the capture process. To set up the export of data from ABBYY FlexiCapture, you will need to create a new script export profile in your ABBYY FlexiCapture project.
```
Before Configuring ABBYY FlexiCapture, please make sure that you have the following software installed:

- ABBYY FlexiCapture 12 Project Setup Station (required if you are using ABBYY FlexiCapture Distributed Edition or ABBYY FlexiCapture Cloud).

If you already have an ABBYY FlexiCapture project for which you need to set up data export to Pega, open that project. Otherwise, you will need to create a project. Launch the Project Setup Station and in the **Open Project** dialog box click:

- **Create New...** to create a new project.
- **Browse...** to open an existing project stored on your local drive.
- **Browse from Server...** to open a project that you earlier uploaded to the ABBYY FlexiCapture Application Server.

Next, you need to upload the project on to the ABBYY FlexiCapture Application Server:

1. Click **File → Upload Project to Server...**

2. In the **Upload Project to Application Server** dialog box specify the address of the server, the protocol to be used, and your credentials.

3. If you are using ABBYY FlexiCapture Cloud, complete the **URL** field, specifying one of the following three FlexiCapture Cloud instances (you should specify the instance that your company is registered in):
   - https://europe.flexicapture.com
   - https://australia.flexicapture.com
   - https://usa.flexicapture.com

4. In the **Tenant name** field specify the name of your company, select the **Use server authentication** option, and enter your credentials.

5. Click **Test Connection** to check that you have entered the correct data.
6. Click **OK** to upload project to the Application Server.

![Upload Project to Application Server](image)

**Note:** When uploading a project to the cloud or when opening a project from the cloud, the following error may occur: "Application server version is not compatible." If you see this error, the version of your Project Setup Station does not match the version of the cloud-based Application Server. To obtain a compatible version of ABBYY FlexiCapture, please contact the ABBYY sales office servicing your region (see https://www.abbyy.com/contacts).

7. When you see the "Operation completed" message, click **Close**. In the next dialog box, click **Yes** to open the uploaded project from the server.

![Operation completed](image)

Next, you need to **configure ABBYY FlexiCapture as an unattended service** and **create an export profile**.

### Configuring ABBYY FlexiCapture as an unattended service

To configure ABBYY FlexiCapture as an unattended service:

1. On the Project Setup Station, click **Project → Project Properties...** and then click the **Workflow** tab.

2. From the **Schema** drop-down list, select **Unattended**.
3. Make sure that the **Export root path** on the **General** tab is left empty.
4. Click OK.

Creating an export profile

To configure an export profile:

1. On the ABBYY FlexiCapture Project Setup Station, click **Project → Document Definitions...** to open the list of Document Definitions available for the project. Select the Document Definition for which you want to specify new export settings and click **Edit...**.

2. Click **Document Definition → Export Settings...**.

3. Click the **Add...** button to start the export profile creation wizard.

4. In the **Select the Type of Destination** step, select **Export to data files** from the **Type** drop-down list, then select the **Errors are irrelevant** item from the **Document condition** drop-down list and click **Next**.

5. In the **Export Data to Files** step, make sure the **Export path** field is empty and specify a file name template. Click **Next**.

7. Provide a name for your newly created export profile and click Finish.

8. In the Export Settings dialog box, select the Enabled option for the newly created export profile, click Apply, and then click OK.

9. Save and close the Document Definition editor. Next, click the Publish button to publish your Document Definition.

Configuring Pega

To be able to send files to ABBYY FlexiCapture and get back the processing results, you need to complete the following steps:

Create a Data Transform record

Create a new activity and add it to your workflow

Create steps for your new activity

Creating a Data Transform record

The Java function described in the Installation section returns a JSON string. It is recommended that you transform this string so that the result is displayed as case fields. To do this, you need to create a record of type Data Transform.

To create a Data Transform record:

1. Open Dev Studio and create a new case type.

2. On the Data model tab, add the fields to be handled by your case and click Save.

   ![Data model table]

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Type</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachments</td>
<td>Attachments</td>
<td>Attachment</td>
<td>Category: Attachments</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Invoice Date</td>
<td>InvoiceDate</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Invoice Number</td>
<td>InvoiceNumber</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Line items</td>
<td>Lineitems</td>
<td>Field group (list)</td>
<td>Data type: Line item</td>
</tr>
<tr>
<td>Total</td>
<td>Total</td>
<td>Text (single line)</td>
<td></td>
</tr>
</tbody>
</table>

3. Click Records → Data Model → Data Transform and then click Create to open the Create Data Transform window.

4. In the Label field, specify a name for your new record.
5. In the **Additional configuration options** section, select **JSON**.

6. In the **Apply to** field, specify the full name of the case type you created earlier.

7. Click **Create and open**. In the window that opens, map the JSON properties (displayed in the **JSON** column) to the case fields (displayed in the **Clipboard** column):
8. Click **Save**.

**Creating a new activity and adding it to your workflow**

Next, you need to create a new activity and add it to your workflow.

1. In **Dev Studio**, open the case type that you created earlier.

2. On the **Workflow** tab, click **Add life cycle**.

3. Create a first stage with a **Collect Information** step. This step will open the dialog box where files can be added.

4. Create a **Process** stage, add any step to this stage (e.g. **Change stage**), and click **Save**.

5. Next, click **Records → Process → Flow** to open the list of flows and double-click the **Process_Flow** flow for the case type that you created earlier.

6. Replace the **Change stage** step with a **Utility** step.
7. Double-click the **Utility** item to open its properties. Change the name of the item to **Capture Data**. In the **Rule** field, type **CaptureData** to create a new activity with this name and click the ☰ icon next to this field.

8. In the window that opens, click **Create and open**.

9. On the **Pages & Classes** tab, create the following three objects:

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
<tr>
<td>Attachment</td>
<td>Data-WorkAttach-File</td>
</tr>
<tr>
<td>AttachmentInfo</td>
<td>Data-WorkAttach-File</td>
</tr>
</tbody>
</table>
10. On the **Parameters** tab, create the following three parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseId</td>
<td>String</td>
</tr>
<tr>
<td>jsonData</td>
<td>String</td>
</tr>
<tr>
<td>executionMode</td>
<td>String</td>
</tr>
</tbody>
</table>

11. Create the following three local variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedRefTo</td>
<td>String</td>
</tr>
<tr>
<td>Id</td>
<td>String</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Local variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LinkedRefTo</td>
<td>String</td>
</tr>
<tr>
<td>2</td>
<td>Id</td>
<td>String</td>
</tr>
<tr>
<td>3</td>
<td>Index</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Creating activity steps

Next, you need to create steps for the CaptureData activity.

1. On the Steps tab, create a new step and for this step: in the Method field, select Obj-Open-By-Handle; in the Step page field, select Case; and in the InstanceHandle field, type Param.Caseld.

<table>
<thead>
<tr>
<th>Label</th>
<th>Method</th>
<th>Step page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, Loop When</td>
<td>Obj-Open-By-Handle</td>
<td>Case</td>
<td>Open record by handle</td>
</tr>
</tbody>
</table>

Method Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceHandle</td>
<td>Param.Caseld</td>
</tr>
<tr>
<td>Lock</td>
<td></td>
</tr>
<tr>
<td>ReleaseOnCommit</td>
<td></td>
</tr>
<tr>
<td>LockInfoPage</td>
<td></td>
</tr>
<tr>
<td>CheckSecondaryStorage</td>
<td></td>
</tr>
</tbody>
</table>

2. Create a second step and for this step: in the Method field, select Call RemoveFromPageList; in the PageListProperty field, type LineItems; in the Index field, type 1 (i.e. the index of an empty string that Pega creates by default); and in the FinishingActivity field, type "" (i.e. an empty string).

| Loop When | Call RemoveFromPageList | Call activity RemoveFromPageList |

Method Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageListProperty</td>
<td>LineItems</td>
</tr>
<tr>
<td>BaseReference</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>1</td>
</tr>
<tr>
<td>FinishingActivity</td>
<td></td>
</tr>
<tr>
<td>PageName</td>
<td></td>
</tr>
</tbody>
</table>

3. Create a third step and for this step: in the Method field, select Property-Set; in the PropertiesName field, select Local.Index; and in the PropertiesValue field, type 0.

| Loop When | Property-Set | Set property values |

Method Parameters

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.Index</td>
<td>0</td>
</tr>
</tbody>
</table>
4. Create a fourth step that will get file identifiers and information about files. For this step, select `Property-Set` in the `Method` field and create the following parameters: `Local.LinkedRefTo` and `Local.Id`.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.LinkedRefTo</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pxLinkedRefTo</td>
</tr>
<tr>
<td>Local.Id</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pzInsKey</td>
</tr>
</tbody>
</table>

4.  

5. Add the following loop to step 4: *For each element in value list – Case.pyAttachments.*

6. Add a `When` condition to step 4 and set this condition to `Local.Index==0`. This will cause the activity to process only the first attachment.

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>if false</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.Index==0</td>
<td>Continue Whens</td>
<td>Skip Step</td>
</tr>
</tbody>
</table>

7. Create six substeps related to step 4 (to create a substep, create a new step and drag into step 4).
o For the first substep: in the Method field, select **Obj-Open-By-Handle**; in the Step page field, select **Attachment**; and in the InstanceHandle field, type **Local.LinkedRefTo**.

![Method Parameters table](image)

1. □ Loop When ▼ Obj-Open-By-Handle Attachment Open record by handle

Method Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Star" /> InstanceHandle</td>
<td>Local.LinkedRefTo</td>
</tr>
<tr>
<td>Lock</td>
<td>□</td>
</tr>
<tr>
<td>ReleaseOnCommit</td>
<td>□</td>
</tr>
<tr>
<td>LockInfoPage</td>
<td>□</td>
</tr>
<tr>
<td>CheckSecondaryStorage</td>
<td>□</td>
</tr>
</tbody>
</table>

o For the second substep: in the Method field, select **Obj-Open-By-Handle**; in the Step page field, select **AttachmentInfo**; and in the InstanceHandle field, type **Local.Id**.

![Method Parameters table](image)

2. □ Loop When ▼ Obj-Open-By-Handle AttachmentInfo Open record by handle

Method Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Star" /> InstanceHandle</td>
<td>Local.Id</td>
</tr>
<tr>
<td>Lock</td>
<td>□</td>
</tr>
<tr>
<td>ReleaseOnCommit</td>
<td>□</td>
</tr>
<tr>
<td>LockInfoPage</td>
<td>□</td>
</tr>
<tr>
<td>CheckSecondaryStorage</td>
<td>□</td>
</tr>
</tbody>
</table>

o For the third substep, select **Property-Set** in the Method field and create the following parameter for this method: **ParamjsonData – @FlexiCaptureLibrary.CaptureData(AttachmentInfo.pyMemo, Attachment.pyAttachStream)** (after the “@” character, specify the name of the library described in the Installation section; after the dot, specify the name of the function). Additionally, create a parameter named **Param.executionMode – DESERIALIZE**.

![Method Parameters table](image)

3. □ Loop When ▼ Property-Set Open record by handle

Method Parameters

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Star" /> ParamjsonData</td>
<td>@FlexiCaptureLibrary.CaptureData(AttachmentInfo.pyMemo, Attachment.pyAttachStream)</td>
</tr>
<tr>
<td><img src="image" alt="Star" /> Param.executionMode</td>
<td>DESERIALIZE</td>
</tr>
</tbody>
</table>

Next, click the **Jump** button and add a **StepStatusFail** parameter for error handling.
When | if true | true param | if false | false param | On exception, jump to later step label
--- | --- | --- | --- | --- | ---
StepStatusFail | Jump to Later Step | ERR | Jump to Later Step | OK | ERR

Click **Submit**.

- For the fourth substep, select *Page-Set-Messages* in the **Method** field and specify the following parameter for this method: *Message – @getWorstMessageName(tools)*. Mark this substep as **ERR**. Next, you need to prevent the activity from performing any subsequent steps if an error occurs. To do this, click the **Jump** button and add the following parameters:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Jump to Later Step</td>
<td>INC</td>
<td>Jump to Later Step</td>
<td>INC</td>
</tr>
</tbody>
</table>

- For the fifth substep select *Apply-DataTransform* in the **Method** field and specify the following parameter for this method: *DataTransform – JsonToInvoiceFields*. Mark this substep as **OK** and select the **PassParameterPage** option.
For the sixth substep, select Property-Set in the Method field, create a Local.Index parameter for this method, and set this parameter to Local.Index+ 1. Mark this substep as INC.

At this point, your list of steps and substeps should look as follows:

8. Click Save.

9. Open the properties of the Utility item, select CaptureData in the Rule field, and set the CaseId parameter to .pzInsKey.
FlexiCaptureConnector API specification

This section describes the following interfaces provided by the FlexiCaptureConnector library:

- Class FlexiCaptureWebServiceClient
- Class FlexiCaptureProject
- Class Credentials
- Class BatchCreationJob
- Class RegistrationProperty
- Class File
- Class Result
- Enum Stage

FlexiCaptureWebServiceClient class

The FlexiCaptureWebServiceClient class is in the com.abbyy.connectors namespace. It inherits from the AutoCloseable class and is used to connect to the ABBYY FlexiCapture 12 web service.

The methods of this class are described below.
Constructor Details

public FlexiCaptureWebServiceClient(String url)

Creates a new client object to connect to the ABBYY FlexiCapture 12 web service

Parameters:

- url – the address of the ABBYY FlexiCapture Application Server in the following format:
  http://<ApplicationServer>.

Method Details

public List<String> getProjectNames()

Returns a list of the names of the projects on the ABBYY FlexiCapture 12 server.

Returns:

- A list of project names

Throws:

- FlexiCaptureConnectorException – if an error occurs when getting a list of project names

public FlexiCaptureProject getProject(String name)

Opens a project based on the name parameter (i.e. the name of the project on the ABBYY
FlexiCapture 12 server) and creates and returns a new FlexiCaptureProject object.

Parameters:

- name – the name of the project on the ABBYY FlexiCapture 12 server

Returns:

- FlexiCaptureProject object

Throws:

- FlexiCaptureConnectorException – if an error occurs when opening the project

See also:

FlexiCaptureProject

public String getTenant()

Returns the name of the tenant.

Returns:
• A string containing the name of the tenant on the ABBYY FlexiCapture 12 server. This string is empty by default

public void setTenant(String tenant)

Sets the name of the tenant.

Parameters:

• tenant – the name of the tenant on the ABBYY FlexiCapture 12 server. This parameter is used when working with the tenant and is not specified when working with the root.

Throws:

• IllegalArgumentException – if the input parameter is empty or 0

public int getTimeout()

Returns the time to wait for a server response.

Returns:

• The time (in seconds) to wait for a response from the ABBYY FlexiCapture 12 server. The default value is 300 seconds

public void setTimeout(int timeout)

Sets the time (in seconds) to wait for a response from the server. The default timeout value is 300 seconds.

Parameters:

• timeout – the time to wait for a response from the ABBYY FlexiCapture 12 server

Throws:

• IllegalArgumentException – if the input parameter is less than or equals 0

public Credentials getCredentials()

Returns the user's name and password.

Returns:

• An object containing the user's name and password

See also:

Credentials
public void setCredentials(Credentials credentials)

Sets the user's name and password.

**Parameters:**

- credentials – an object containing the user's name and password

**Throws:**

- IllegalArgumentException – if the input parameters are empty or 0

*See also:*

[Credentials](#)

public void close()

Closes the connection to the ABBYY FlexiCapture 12 Web API.

**FlexiCaptureProject class**

The FlexiCaptureProject class is in the `com.abbyy.connectors` namespace. It inherits from the AutoCloseable class and provides methods for working with an ABBYY FlexiCapture 12 project.

The methods of this class are described below.

**Method Details**

public List<String> getBatchTypeNames()

Gets a list of the names of all the batch types in a project.

**Returns:**

- A list of batch type names

**Throws:**

- FlexiCaptureConnectorException – if an error occurs when getting a list of batch type names

public int create(BatchCreationJob job)

Creates a new batch of files on the ABBYY FlexiCapture 12 server and starts the processing of the batch.

**Parameters:**

- job – a job that creates a new batch on the ABBYY FlexiCapture 12 server

**Returns:**

- The ID of the batch
Throws:

- FlexiCaptureConnectorException – if an error occurs when creating a batch
- NullPointerException – if the input parameter is 0
- IllegalArgumentException – if the input parameter contains invalid property values

See also:

BatchCreationJob

public Result getResult(int batchId)

Gets batch processing results from the ABBYY FlexiCapture 12 server.

Parameters:

- batchId – the ID of the batch

Returns:

- An object containing batch processing results

Throws:

- FlexiCaptureConnectorException – if an error occurs when getting batch processing results
- IllegalArgumentException – if the input parameter is less than or equals 0

See also:

Result

public void close()

Closes the project.

Credentials class

The Credentials class is in the com.abbyy.connectors.models namespace. It stores the login and password of the ABBYY FlexiCapture 12 user.

The methods of this class are described below.

Constructor Details

public Credentials(String user, String password)

Creates a credentials object.

Parameters:
• user – the user’s name in ABBYY FlexiCapture 12
• password – the user’s password for accessing ABBYY FlexiCapture 12

**BatchCreationJob class**

The *BatchCreationJob* class is in the `com.abbyy.connectors.models` namespace. It stores a job that creates a new batch in an ABBYY FlexiCapture 12 project.

The methods of this class are described below.

**Constructor Details**

```java
public BatchCreationJob(List<File> files)
```

Creates a new job that will create a batch in ABBYY FlexiCapture 12.

**Parameters:**

• files – the list of files to be processed in ABBYY FlexiCapture 12

**See also:**

File

**Method Details**

```java
public void setName(String name)
```

Specifies a name for the batch.

**Parameters:**

• name – the name of the batch. The default name is *Batch_ + System.currentTimeMillis()*

```java
public void setType(String type)
```

Specifies a type for the batch.

**Parameters:**

• type – the type of the batch. The default type is *Default*

```java
public void setRegistrationProperties(List<RegistrationProperty> properties)
```

Specifies a list of registration parameters for the batch.

**Parameters:**

• properties – the registration parameters of the batch

**See also:**
RegistrationProperty class

The RegistrationProperty class is in the com.abbyy.connectors.models namespace. It is used for working with batch registration parameters.

The methods of this class are described below.

Constructor Details

public RegistrationProperty(String name, String value)

Creates a registration parameter object in a batch.

Parameters:

- name – the name of the registration parameter
- value – the value of the registration parameter

File class

The File class is in the com.abbyy.connectors.models namespace. It contains information about the target file.

The methods of this class are described below.

Constructor Details

public File(String name, byte[] bytes)

Creates a file object.

Parameters:

- name – the name of the file
- bytes – the contents of the file as a byte array

Result class

The Result class is in the com.abbyy.connectors.models namespace. It contains processing results.

The methods of this class are described below.

Method Details

public Stage getStage()

Returns the ID of the stage where the batch processing stopped.

Returns:

- The stage where the batch processing stopped: Verification (500), Processed (900)

See also:
Stage

public List<File> getFiles()

Returns a list of output files.

Returns:

- A list of output files. The list will be empty if the document does not go through the "Export" stage and stops at the "Verification (500)" stage. The list will also be empty if there is no "Export to data files" export profile configured in the Document Definition.

public String getFields()

Returns recognition results as JSON string.

Returns:

- A JSON string containing recognition results. The string contains the names of the captured fields and their values. It can be processed on the Pega side, where the field values can be extracted and written to their respective fields in the case. The string will be empty if there is no "Export to XML" export profile configured in the Document Definition used by the ABBYY FlexiCapture project. The string will also be empty if the document does not go through the "Export" stage and stops at the "Verification (500)" stage.

Stage enum

The Stage enum is in the com.abbyy.connectors.models namespace. It contains the stages where the processing of a document stopped.

The constants of this class are described below.

Enum constants

- Verification (500) – the document stopped at the "Verification" stage
- Processed (900) – the document has been successfully processed

ABBYY FlexiCapture as a service with verification

Architecture

For this scenario, ABBYY FlexiCapture Connector for Pega includes a JAR file which allows you to access the classes used for working with the ABBYY FlexiCapture 12 Web API.

This connector module runs on the Pega side, where it sends files to ABBYY FlexiCapture for processing and gets back processing results.

The abbyy.flexicapture.webapi.client.jar methods that implement the connector's functionality are described in the Javadoc file, which is included in the distribution package.
The distribution package includes the Pega application, which could be imported to your Pega server as an example of ready-to-use ABBYY FlexiCapture integration. For more information about importing the application, see Importing the ABBYY FlexiCapture application into Pega.

**Installation**

To install the ABBYY FlexiCapture Connector for Pega, you need to:

- [Import a connector to the Pega server](#)
- [Create a library](#)
- [Create required functions](#)

**Importing a connector to the Pega server**

To import the connector to the server:

1. Open [Dev Studio](#) and click `Configure → Application → Distribution → Import`.

2. Load the `abbyy.flexicapture.webapi.client.jar` file on to the server.

3. Open the [Pega Self-Service Portal](#) and click `Restart Server`.

---

![Dev Studio Configuration](image)

**Portal Operations Enabled**

- [Restart Server](#)
- [Clear Cache](#)
- [Backup DB](#)
- [Restore DB](#)
- [Update](#)

- **Clear Extract Marker & Restart App Server**
- **Check Box to Clear Database Cache**
  (Restarts all app nodes)

- **Take a Database Backup (1 Per System) –**
  
  Enter a description for this backup

- **Restore the Database Backup – BackupID: x**
  
  Description:

- **In-place Update of the Pega Platform**
  
  Select Build:
  
  - 8.1.0-Alpha
  - Daily Build: 8.1.0-HEAD6130

---

44
Creating a library

To create a library:

1. Click **Records → Technical → Library → Create**.

2. In the **Label** field of the **Create Library** tab, specify a name for the library (e.g. **ABBYY**) and fill in the **Identifier** field. Make sure that the library is being created within the context of your application and then click **Create and open**.

3. On the **Packages** tab, add the following packages:
   - `java.util.*`
   - `java.nio.file.*`
   - `java.net.*`
   - `java.lang.*`
   - `java.util.stream.*`
   - `org.apache.commons.lang3.*`
   - `abbyy.flexicapture.webapi.client.*`
   - `abbyy.flexicapture.webapi.client.api.*`
   - `abbyy.flexicapture.webapi.client.models.*`
4. On the **Static Variables** tab, create the following string variables:

   - `baseUri`
   - `tenantName`
   - `userName`
   - `Password`
   - `multiFileProjectName`
   - `singleFileProjectName`
5. Click **Save** and then click **Generate Library**.

**Creating a function**

A function contains Java code that is executed when the function is called from other Pega objects. The input and output parameters of a function are specified on the **Parameters** tab, while exceptions are specified on the **Imports & Exceptions** tab.

To create a function:

1. Click **Records → Technical → Function → Create** and fill in the **Label** and **Identifier** fields.

2. In the **Library** field, specify the name of the library you created earlier.

3. In the **Parameters** section, specify the following two input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>String</td>
</tr>
<tr>
<td>Base64Content</td>
<td>String</td>
</tr>
</tbody>
</table>

4. Make sure that the function is being created within the context of your application and click **Create** and open.
5. In the window that opens, on the **Parameters** tab, specify **int, string or boolean** as the data type in the **Java data type** field:

6. Click the **Imports & Exceptions** tab and type **Exception** in the **Exceptions thrown** section.
7. Go back to the **Java** tab, insert the code for the function in the **Java source** field, select the **Function ready to be compiled?** option, save your changes, and click **Generate function**.

**Note:** The code for the function can be copied from the java file, which you can find in `%Installation Path%Connector for FlexiCapture as a Service with verification\Samples\Code examples\Pega Functions`.

To implement a single-document processing scenario, you need to:

1. **Create a set of functions to be used for the processing of a single document.**

2. **Create general functions.**
3. Create a single-document processing case.
4. Configure the user interface.
5. Create a set of activities for your single-document processing case.
6. Create a set of data transforms for your single-document processing case.
7. Create a flow action.
8. Create data types for your single-document processing case.

To implement a multi-document processing scenario, you need to:

1. Create a set of functions to be used for the processing of multiple documents.
2. Create general functions.
3. Create a multi-document processing case.
4. Create HTML fragments.
5. Create a set of activities for your multi-document processing case.
6. Create a set of data transforms for your multi-document processing case.
7. Create a flow action.
8. Create data types for your multi-document processing case.

Creating a set of functions for processing and verifying a single document

To implement a single-document processing scenario, create the following functions:

SendForProcessing
WaitFirstVerificationOrProcessed
FetchingCapturedData
SingleDocumentBatchIsProcessed

SendForProcessing

The `SendForProcessing` function sends a file to ABBYY FlexiCapture for recognition and returns a batch identifier.

Input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>String</td>
<td>Name of the file</td>
</tr>
<tr>
<td>base64Content</td>
<td>String</td>
<td>File contents in Base64 format</td>
</tr>
</tbody>
</table>
**Output parameter type:** int (batch identifier).

**Exceptions thrown:** Exception.

**Java source:**

```java
// Creating FlexiCapture Web Services API client.
// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(singleFileProjectName)) {

        // Creating FlexiCapture batch and document.
        Batch batch = new Batch();
        Document doc = new Document();

        // Adding batch into FlexiCapture project.
        int batchId = projApi.getBatches().add(batch);

        // Getting the FlexiCapture batch api. Please provide a batch identifier here.
        BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

        // Creating instance of file.
        File file = new File(fileName, Base64.getDecoder().decode(base64Content.replace("\n", "").replace("\r", "")));

        // Adding document into FlexiCapture batch.
        // File is document content.
        int docId = batchApi.getDocuments().add(doc, file);

        // Running the FlexiCapture batch for processing.
        batchApi.start();

        //Returning batch identifier.
    }
}
```
return batchId;
}
} catch(Exception ex) {
    // Chaining error messages.
    Throwable cause = ex.getCause();
    String message = ex.getMessage();
    while (cause != null) {
        message += " " + cause.getMessage();
        cause = cause.getCause();
    }
    // Throw chained message.
    throw new Exception(message);
}

#### WaitFirstVerificationOrProcessed

The `WaitFirstVerificationOrProcessed` function returns a verification URL. If the document has not reached the verification stage, an empty string is returned.

**Input parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
</tbody>
</table>

**Output parameter type:** `String` (verification URL).

**Exceptions thrown:** Exception.

**Java source:**

```java
// Creating FlexiCapture Web Services API client.

// Please provide FlexiCapture Application Server address, tenant name, user name and password here.

try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(singleFileProjectName)) {
```
// Getting the FlexiCapture batch api. Please provide a batch identifier here.
BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

// Waiting for the batch to be stopped on the verification stage and getting the FlexiCapture document.
Document doc = batchApi.getDocuments().waitFirstVerificationOrProcessed();

// Check if the batch stage is Verification.
if (doc.getStageType() == ProcessingStageType.Verification) {

    // Returning first verification url string for FlexiCapture batch.
    return batchApi.getVerificationUrls(WebPageMode.Mini).get(0).toString();
}

//Returning empty string if batch was processed without verification stage.
return "";

} catch(Exception ex) {

    // Chaining error messages.
    Throwable cause = ex.getCause();
    String message = ex.getMessage();

    while (cause != null) {
        message += " " + cause.getMessage();
        cause = cause.getCause();
    }

    // Throw chained message.
    throw new Exception(message);
}

FetchingCapturedData

The FetchingCapturedData function returns a list of output fields in JSON format.
### Input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
</tbody>
</table>

**Output parameter type:** `String` (document fields in JSON format).

**Exceptions thrown:** `Exception`.

**Java source:**

// Creating FlexiCapture Web Services API client.

// Please provide FlexiCapture Application Server address, tenant name, user name and password here.

try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

// Getting the FlexiCapture project api. Please provide a project name here.

try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(singleFileProjectName)) {

// Getting the FlexiCapture batch api. Please provide a batch identifier here.

BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

// Waiting for the batch to be processed and getting the FlexiCapture document.

Document doc = batchApi.getDocuments().waitFirstVerificationOrProcessed();

// Getting the FlexiCapture document api. Please provide a document identifier here.

DocumentApi docApi = batchApi.getDocuments().getDocumentApi(doc.getId());

// Returning captured fields as a JSON string.

return docApi.getExportedFields();

} catch(Exception ex) {

// Chaining error messages.

Throwable cause = ex.getCause();

String message = ex.getMessage();

while (cause != null) {
    message += " " + cause.getMessage();
    cause = cause.getCause();
}

// Throw chained message.
throw new Exception(message);

**SingleDocumentBatchIsProcessed**

The `SingleDocumentBatchIsProcessed` function returns a value that specifies whether a batch has been processed or not.

**Input parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
</tbody>
</table>

**Output parameter type:** `boolean` *(true if the document is at the Processed stage and false otherwise).*

**Exceptions thrown:** `Exception`.

**Java source:**

```java
// Creating FlexiCapture Web Services API client.
// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    try (ProjectApi projApi = tenantClient.getProjects().getProjectApi(singleFileProjectName)) {

        BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

        // Waiting for the batch to be stopped on the verification or processed stage and getting the FlexiCapture document.
        Document doc = batchApi.getDocuments().waitFirstVerificationOrProcessed();
```
// Check if the batch stage is Processed.
return doc.getStageType() == ProcessingStageType.Processed;
}
catch(Exception ex){

// Chaining error messages.
Throwable cause = ex.getCause();
String message = ex.getMessage();
while (cause != null) {
    message += " " + cause.getMessage();
    cause = cause.getCause();
}

// Throw chained message.
throw new Exception(message);
}

Creating a set of functions for processing and verifying multiple documents
To implement a multi-document processing scenario, create the following functions:

CreateNewBatch
AddDocumentToBatch
RunBatchProcessing
WaitFirstVerificationOrProcessedForMultiFileInvoice
FetchingCapturedData
MultiDocumentBatchIsProcessed

CreateNewBatch

The CreateNewBatch function creates a new batch and returns its identifier.
There are no input parameters for this function.
Output parameter type: int (batch identifier).
Exceptions thrown: Exception.
Java source:

```java
// Creating FlexiCapture Web Services API client.
// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(multiFileProjectName)) {

        // Creating FlexiCapture batch and document.
        Batch batch = new Batch();
        Document doc = new Document();

        // Adding batch into FlexiCapture project.
        return projApi.getBatches().add(batch);
    }
} catch(Exception ex){

    // Chaining error messages.
    Throwable cause = ex.getCause();
    String message = ex.getMessage();

    while (cause != null) {
        message += " " + cause.getMessage();
        cause = cause.getCause();
    }

    // Throw chained message.
    throw new Exception(message);
}
```

**AddDocumentToBatch**

The *AddDocumentToBatch* function adds a document to a batch.
Input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
<tr>
<td>fileName</td>
<td>String</td>
<td>Name of the file</td>
</tr>
<tr>
<td>base64Content</td>
<td>String</td>
<td>File contents in Base64 format</td>
</tr>
</tbody>
</table>

There are no output parameters for this function.

Exceptions thrown: Exception.

Java source:

```java
// Creating FlexiCapture Web Services API client.
// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(multiFileProjectName)) {

        // Creating FlexiCapture document.
        Document doc = new Document();

        // Getting the FlexiCapture batch api. Please provide a batch identifier here.
        BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

        // Creating instance of file.
        File file = new File(fileName, Base64.getDecoder().decode(base64Content.replace("\n", "").replace("\r", "")));

        // Adding document into FlexiCapture batch.
        // File is document content.
        batchApi.getDocuments().add(doc, file);
    }
}
```
// Chaining error messages.
Throwable cause = ex.getCause();
String message = ex.getMessage();

while (cause != null) {
    message += " " + cause.getMessage();
    cause = cause.getCause();
}

// Throw chained message.
throw new Exception(message);

RunBatchProcessing

The RunBatchProcessing function initializes batch processing.

Input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
</tbody>
</table>

There are no output parameters for this function.

Exceptions thrown: Exception.

Java source:

// Creating FlexiCapture Web Services API client.
// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(multiFileProjectName)) {

        // Getting the FlexiCapture batch api. Please provide a batch identifier here.
        BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);
// Running the FlexiCapture batch for processing.
batchApi.start();

try{
    // Chaining error messages.
    Throwable cause = ex.getCause();
    String message = ex.getMessage();

    while (cause != null) {
        message += " " + cause.getMessage();
        cause = cause.getCause();
    }

    // Throw chained message.
    throw new Exception(message);
}

WaitFirstVerificationOrProcessedForMultiFileInvoice

The `WaitFirstVerificationOrProcessedForMultiFileInvoice` function returns a verification URL. If the document has not reached the verification stage, an empty string is returned.

**Input parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
</tbody>
</table>

**Output parameter type:** `String` (verification URL).

**Exceptions thrown:** Exception.

**Java source:**

```
// Creating FlexiCapture Web Services API client.

// Please provide FlexiCapture Application Server address, tenant name, user name and password here.

try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {
```
// Getting the FlexiCapture project api. Please provide a project name here.
try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(multiFileProjectName)) {

// Getting the FlexiCapture batch api. Please provide a batch identifier here.
BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

// Waiting for the batch to be stopped on the verification stage and getting the
// FlexiCapture document.
Document doc = batchApi.getDocuments().waitFirstVerificationOrProcessed();

// Check if the batch stage is Verification.
if (doc.getStageType() == ProcessingStageType.Verification) {

    // Returning first verification url string for FlexiCapture batch.
    return batchApi.getVerificationUrls(WebPageMode.Mini).get(0).toString();
}

//Returning empty string if batch was processed without verification stage.
return "";
} catch(Exception ex) {

    // Chaining error messages.
    Throwable cause = ex.getCause();
    String message = ex.getMessage();

    while (cause != null) {
        message += " " + cause.getMessage();
        cause = cause.getCause();
    }

    // Throw chained message.
    throw new Exception(message);
}
**FetchingCapturedData**

The *FetchingCapturedData* function returns a list of output fields in JSON format.

**Input parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchId</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
<tr>
<td>templateName</td>
<td>String</td>
<td>Document Definition name</td>
</tr>
</tbody>
</table>

**Output parameter type:** *String* (document fields in JSON format).

**Exceptions thrown:** Exception.

**Java source:**

```java
// Creating FlexiCapture Web Services API client.

// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try (ProjectApi projApi = tenantClient.getProjects().getProjectApi(multiFileProjectName)) {

        // Getting the FlexiCapture batch api. Please provide a batch identifier here.
        BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

        // Getting all documents from FlexiCapture batch.
        List<Document> documents = batchApi.getDocuments().getAll();

        for (Document document:

            documents) {

                // Cheking document template name.
                if (document.getTemplateName().equals(templateName)) {

                    // Getting the FlexiCapture document api. Please provide a document identifier here.
                    DocumentApi docApi = batchApi.getDocuments().getDocumentApi(document.getId());
```
// Returning captured fields as a JSON string.
return docApi.getExportedFields();

// Returning empty JSON string if document template name not found.
return "{}";

) catch (Exception ex) {

  // Chaining error messages.
  Throwable cause = ex.getCause();
  String message = ex.getMessage();

  while (cause != null) {
    message += " " + cause.getMessage();
    cause = cause.getCause();
  }

  // Throw chained message.
  throw new Exception(message);
}

MultiDocumentBatchIsProcessed

The `MultiDocumentBatchIsProcessed` function returns a value that specifies whether the batch has been processed or not.

Input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchSize</td>
<td>int</td>
<td>Batch identifier</td>
</tr>
</tbody>
</table>

Output parameter type: `boolean` (*true* if the document is at the *Processed* stage and *false* otherwise).

Exceptions thrown: Exception.

Java source:
// Creating FlexiCapture Web Services API client.
// Please provide FlexiCapture Application Server address, tenant name, user name and password here.
try (TenantClient tenantClient = new TenantClient(new URI(baseUri), tenantName, userName, password)) {

    // Getting the FlexiCapture project api. Please provide a project name here.
    try(ProjectApi projApi = tenantClient.getProjects().getProjectApi(multiFileProjectName)) {

        // Getting the FlexiCapture batch api. Please provide a batch identifier here.
        BatchApi batchApi = projApi.getBatches().getBatchApi(batchId);

        //Waiting for the batch to be stopped on the verification or processed stage and getting the FlexiCapture document.
        Document doc = batchApi.getDocuments().waitFirstVerificationOrProcessed();

        // Check if the batch stage is Processed.
        return doc.getStageType() == ProcessingStageType.Processed;
    }
} catch(Exception ex){

    // Chaining error messages.
    Throwable cause = ex.getCause();
    String message = ex.getMessage();

    while (cause != null) {
        message += " " + cause.getMessage();
        cause = cause.getCause();
    }

    // Throw chained message.
    throw new Exception(message);
}
Creating general functions

To implement your scenarios, you will also need to create the following function:

unescapeHTML

The \textit{unescapeHTML} function converts HTML codes into Unicode characters. This function is needed for correctly displaying error messages.

\textbf{Input parameters:}

<table>
<thead>
<tr>
<th>Name</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>htmlString</td>
<td>String</td>
<td>String of HTML codes</td>
</tr>
</tbody>
</table>

\textbf{Output parameter type:} \textit{String} (string of Unicode characters).

The \textbf{Exceptions thrown} field should be left blank.

\textbf{Java source:}

\begin{verbatim}
//Unescapes a string containing entity escapes to a string containing the actual Unicode characters
//corresponding to the escapes. Supports HTML 4.0 entities.
return StringEscapeUtils.unescapeHtml4(htmlString);
\end{verbatim}

Creating a case with a workflow

To create a new case:

1. Open \textit{Dev Studio} and click \textit{Case Types} → \textit{Add a case type}.
2. Specify a name for your case and click \textit{Submit}.
3. On the \textit{Data model} tab, add new case fields by clicking \textit{Add field}.

Specify the following required fields:

- \textbf{Attachments} — \textit{Attachment} type (files) with the option \textit{Category: Attachments};
- \textbf{BatchId} — \textit{Integer} type (batch identifier);
- \textbf{VerificationUrl} — \textit{Text} type (verification URL).
4. On the **Workflow** tab, click **Add life cycle** and create the following stages with the **Collect information** steps:

**Note:** You can give any names to the stages and steps.

- **Invoice received** is the first stage. At this stage, documents can be added to the case.
  - **Send for processing** is a stage where invoice processing is initiated on the ABBYY FlexiCapture side and the program starts waiting for a processing status. At this stage, Pega
    - extracts an image from the case attachment;
    - sends the image to ABBYY FlexiCapture for processing;
    - records the ABBYY FlexiCapture batch identifier in the case fields;
    - waits for the batch processing to finish;
    - updates the case field containing the verification URL if the batch has stopped at the verification stage;
    - proceeds to the ABBYY FlexiCapture data fetching stage if the batch has been processed completely.
- **Verification** – At this stage, a verification page is embedded in the case interface using the iframe element.
- **Fetching captured data** – At this stage, export results are downloaded from ABBYY FlexiCapture and the case fields are filled in.
- **Next stage** – You can use this stage to include more steps.

5. Click **Save**.

**Configuring workflow stages**

When setting up scenarios for working with single or multiple documents, you will need to configure workflow stages.

To configure workflow stages, do the following:

1. Open your workflow and click **Configure process → Open process**. Alternatively, click **Records → Process → Flow** in Dev Studio.
2. On the **Diagram** tab, delete the existing step and add a **Utility step**, which will call the necessary activity during the scenario. You will also need to add a **Decision step** that will be used to check the conditions at some of the stages.

### The Utility step

To set up an activity call:

1. Add a new **Utility** step to the process, specify a name for it, and connect it to the **Start** and **End** steps.

2. Open the activity window and choose an **Activity** rule type.

3. In the **Rule** field, specify the name of the activity to be called. If the activity does not exist in the context of the case, it can be created by clicking the configuration button on the left. If the activity already exists, clicking the configuration button will open it. For more details about creating activities, see [Creating Pega activities](#).
The Decision step

To set up a branch:

1. Add a *Decision* step to the process and specify a name for it.

2. Double-click the step and select *Boolean Expression* in the *Type* field.

3. Click the configuration button to the left of the *Expression* field. In the *Expression builder* window, specify the required expression. To see a full list of available functions and properties, click *Browse*. Click *Test* to test your expression.
4. To apply the changes you have made, click **Submit**. The *Decision* step will now have two branches, *true* and *false*.

**Creating a single-document processing case**

For more details about creating cases, see *Creating a case with a workflow*.

For a single-document processing case that includes verification, the **Workflow** tab should look as follows:

The **Data Model** tab should have the following fields (in addition to the required *Attachments*, *BatchId*, and *VerificationUrl* fields):

<table>
<thead>
<tr>
<th>Name</th>
<th>Id</th>
<th>Type</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>City</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>Country</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>Currency</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>InvoiceDate</td>
<td>InvoiceDate</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>InvoiceNumber</td>
<td>InvoiceNumber</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Line Items</td>
<td>LineItems</td>
<td>Field group (list)</td>
<td>Data type: Line Item</td>
</tr>
<tr>
<td>Total</td>
<td>Total</td>
<td>Text</td>
<td></td>
</tr>
</tbody>
</table>
Configuring a stage that will submit a document for processing

For more detailed instructions, see the Configuring workflow stages and Utility step sections.

1. Open the Send for processing stage in configuration mode, replace the Edit details step with a Utility step, and specify the following name for the step: Send for processing.

2. Open the Utility step and call the SendForProcessing activity. For more details about creating this activity, see SendForProcessing.

**Note:** You can begin creating an activity by clicking the configuration button to the right of the Rule field. Alternatively, you can leave the pyUtilityStub value specified in the Rule field and come back to this step once you have created an activity set.

3. In the Caseld field, specify the following: .pzInsKey.
Configuring a stage that will fetch the processing results

For more detailed instructions, see the Configuring workflow stages and Utility step sections.
1. Open the *Fetching captured data* stage in configuration mode and delete the *Edit details* step.

2. Add a *Decision* step and name it *Batch stage is processed*.

3. Open this step and select *Boolean Expression* in the *Type* field. Specify or configure a `@SingleDocumentBatchIsProcessed(BatchId)` condition in the *Expression* field. For more details about creating this function, see SingleDocumentBatchIsProcessed.

   **Decision properties**

   - **Decision:** *Batch stage is processed*
     - Automatically evaluate a business decision to determine how this case progresses.

   **Details**

   - **Type**: *Boolean Expression*

   **Expression**

   `@SingleDocumentBatchIsProcessed(BatchId)`

   **Result in**

   4. For the *true* branch in the *Utility* step, add a step named *Fetching captured data*. To do this, open the *Utility* step and call the *FetchingCapturedData* activity. Then, in the *CaseId* field, specify the following value: `.pzInsKey`. 
5. For the false branch of the Utility step, add a step named Create Error. To do this, open the Utility step and call the CreateError activity. Then, in the CaseId parameter field, specify the following value: .pzInsKey. In the errorMessage field, enter the following text: "Please complete the verification task before proceeding."
Configuring the user interface

In order to skip the Create view when creating a new case and go straight to case configuration, do the following:

1. Open the case in Case types and click Settings.

2. On the General tab, select Skip ‘Create’ view when users create a new case.
Adding files to a case

To add files to a configured case:

1. Click **Add attachment… → File from device**.
2. Select the desired files and click **Attach** in the dialog box that opens.
3. Click **Submit** to run the workflow.

To create an interface for adding files, do the following:

1. Open the **InvoiceReceived_Flow** flow in **Dev Studio** by clicking **Record → Process → Flow** or **App → <case name> → Process → Flow**.
2. Open the properties for the connector of the two stages. In the **Flow action** field, add a flow action for adding files. For more details, see **Creating a flow action**.
Verification page

The verification page is displayed in the case header when the document arrives at the verification stage. To continue processing the document after the verification is finished, click Complete Task, wait for a message that says "Task completed," and then click Submit.
To create a verification page interface, do the following:

1. Open the Verification_Flow flow in Dev Studio by clicking (Record $\rightarrow$ Process $\rightarrow$ Flow or App $\rightarrow$ <case name> $\rightarrow$ Process $\rightarrow$ Flow).

2. Open the properties for the connector of the two stages. In the Flow action field, add a flow action containing a verification page. For more details, see Creating a flow action.
Skipping the verification stage

If no verification stage is set up on the ABBYY FlexiCapture side or if a batch skips verification for other reasons and goes straight to the Processed stage, then you don't need to verify this batch in this case either.

To skip the verification stage in a workflow, do the following:

1. Select the verification stage in the workflow, click the Validation tab and create a new VerificationUrlEmpty.

2. Create a length of .VerificationUrl is Equal To 0 rule.
The **pyCaseInformation section**

For the data received from ABBYY FlexiCapture to be displayed in a Pega case, you need to create and configure a section.

To create a user interface section, do the following:

1. On the **App** tab, click **Create → User Interface → Section**.
2. In the **Label** and **Identifier** fields, enter the following value: *pyCaselInformation*. Make sure that the section is being created within the context of the application and that the full name of the case type is specified in the **Apply to** field. Next, click **Create and open**.
3. Open the section that you have created, click the HTML tab, and clear the **Auto-generated HTML** option.

4. In the **HTML Source** field, paste the code from the `pyCaseInformation.html` file, which you can find in `%Installation Path%\Connector for FlexiCapture as a Service with verification\Samples\Code examples\Configure UI\Single document`.

**Creating a multi-document processing case**

For more details about creating cases, see [Creating a case with a workflow](#).

A configured workflow for processing multiple documents should look as follows:

![Diagram of multi-document processing workflow](#)

For a multi-document processing case that includes verification, the **Workflow tab** should look as follows:

![Workflow diagram](#)
The **Data Model** tab should have the following fields (in addition to the required *Attachments*, *BatchId*, and *VerificationUrl* fields):

<table>
<thead>
<tr>
<th>Name</th>
<th>Id</th>
<th>Type</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Address</td>
<td>Field group (list)</td>
<td>Data type: Address</td>
</tr>
<tr>
<td>BirthDate</td>
<td>BirthDate</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>City</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>ContractDate</td>
<td>ContractDate</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>ContractNumber</td>
<td>ContractNumber</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>DeliveryAddress</td>
<td>DeliveryAddress</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>EmployerName</td>
<td>EmployerName</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>FirstName</td>
<td>FirstName</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>HomePhone</td>
<td>HomePhone</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td>Initial</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>InvoiceCompany</td>
<td>InvoiceCompany</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>InvoiceDate</td>
<td>InvoiceDate</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>InvoiceNumber</td>
<td>InvoiceNumber</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td><strong>InvoiceTable</strong></td>
<td>InvoiceTable</td>
<td>Field group (list)</td>
<td>Data type: InvoiceTable</td>
</tr>
<tr>
<td>LastName</td>
<td>LastName</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>LetterDate</td>
<td>LetterDate</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>LetterFrom</td>
<td>LetterFrom</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>LetterTo</td>
<td>LetterTo</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>PriceCompany</td>
<td>PriceCompany</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td><strong>PriceTable</strong></td>
<td>PriceTable</td>
<td>Field group (list)</td>
<td>Data type: PriceTable</td>
</tr>
<tr>
<td>SSN</td>
<td>SSN</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Total</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>WorkPhone</td>
<td>WorkPhone</td>
<td>Text</td>
<td></td>
</tr>
</tbody>
</table>
### Data model

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Field group (list)</td>
<td>Data type: Address</td>
</tr>
<tr>
<td>Attachments</td>
<td>Attachment</td>
<td>Category: Attachments</td>
</tr>
<tr>
<td>BatchId</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>ContractDate</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>ContractNumber</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>DeliveryAddress</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>EmployerName</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>FirstName</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>HomePhone</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>InvoiceCompany</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>InvoiceDate</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>InvoiceNumber</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>InvoiceTable</td>
<td>Field group (list)</td>
<td>Data type: InvoiceTable</td>
</tr>
<tr>
<td>LastName</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>LetterDate</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>LetterFrom</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>LetterTo</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>PriceCompany</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>PriceTable</td>
<td>Field group (list)</td>
<td>Data type: PriceTable</td>
</tr>
<tr>
<td>SSN</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>VerificationUrl</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>WorkPhone</td>
<td>Text (single line)</td>
<td></td>
</tr>
</tbody>
</table>

### Configuring a stage that will submit a document for processing

For more detailed instructions, see the [Configuring workflow stages](#) and [Utility step](#) sections.
1. Open the Send for processing stage in configuration mode, replace the Edit details step with a Utility step, and specify the following name for this step: Send for processing.

2. Open the Utility step and call the SendForProcessing activity. For more details about creating this activity, see SendForProcessing.

**Note:** You can begin creating an activity by clicking the configuration button to the right of the Rule field. Alternatively, you can leave the pyUtilityStub value specified in the Rule field and come back to this step once you have created an activity set.

3. In the CaseId field, specify the following: .pzInsKey.
Configuring a verification stage

For more detailed instructions, see the Configuring workflow stages and Utility step sections.

1. Open the Verification stage in configuration mode, add a Utility step, and set its name to be Update Verification Url.

2. Rename the Edit details step to Verification.

3. Open the Update Verification Url step and call the UpdateVerificationUrl activity. For more details about creating an activity for a multi-document case, see UpdateVerificationUrl.

4. Connect the Verification and Update Verification Url steps using a Verification flow action connector. For more information about creating a flow action, see Creating a flow action.

5. In the CaseId field, specify the following: .pzInsKey.

Configuring a stage that will fetch the processing results

For more detailed instructions, see the Configuring workflow stages, Utility step, and Decision step sections.

1. Open the Fetching captured data stage in configuration mode and delete the Edit details step.

2. Add a Decision step and name it Batch stage is processed.

3. Open the step and select Boolean Expression in the Type field.

4. In the Expression field, enter or configure a @MultiDocumentBatchIsProcessed(.BatchId) condition. For more details about creating this function, see MultiDocumentBatchIsProcessed.
5. Click **Submit**.

6. Create a **Utility** step named *Fetching Invoice data* for the **true** branch and call the *FetchingCapturedData* activity. For more information about creating this activity, see *FetchingCapturedData*.

7. Enter `.pzInsKey` in the **casId** field and *Invoice_eng* in the **templateName** field.
8. For the true branch, create a Utility step named *Fetching Banking data* and call the *FetchingCapturedData* activity.

9. Enter `.pzInsKey` in the **caseId** field and *Banking_eng* in the **templateName** field.
10. For the true branch, create a Utility step named Fetching Contract data and call the FetchingCapturedData activity.

11. Enter .pzInsKey in the caseId field and Contract_eng in the templateName field.
12. For the true branch, create a Utility step named Fetching Letter data and call the FetchingCapturedData activity.

13. Enter .pzInsKey in the caseId field and Letter_eng in the templateName field.
14. For the true branch, create a Utility step named **Fetching Price data** and call the **FetchingCapturedData** activity.

15. Enter `.pzInsKey` in the **caseId** field and **Price_eng** in the **templateName** field.
16. For the false branch, create a Utility step named Create Error and call the CreateError activity. For more details about creating this activity, see CreateError.

17. Enter .pzInsKey in the caseId field. In the errorMessage field, enter the following text: “Please complete the verification task before proceeding.”
Configuring the user interface

The user interface for a multi-document processing case is configured exactly the same way as the user interface for a single-document case. The only difference is in the code that you need to paste into the HTML Source field. This code should be copied from the pyCaseInformation.html file, which you can find in %Installation Path%\Connector for FlexiCapture as a Service with verification\Samples\Code examples\Configure UI\Multi-document.

Creating HTML fragments

HTML fragments draw parts of the user interface used for a multi-document processing case.

To create a new fragment, do the following:

1. In Dev Studio, click Records → Technical → HTML Fragment → Create.

2. In the Label and Identifier fields, enter the following value: InvoiceDefaultFragment.
3. Click **Create and open**.

4. In the **HTML Source** field, enter the code of your fragment. The code to be used for each of the fragments is provided below.

**Note:** The code of all the fragments can also be found in the connector installation folder at %Installation Path%\Connector for FlexiCapture as a Service with verification\Samples\Code examples\Configure UI\Multi-document\HTML fragments.

**InvoiceDefaultFragment**

```html
<head>
  <meta charset="utf-8" />
  <style>
    *
    box-sizing: border-box;
  }
  .value {
    color: black;
  }
  .name{
    color: grey;
  }
  .invoicecolumn {
    float: left;
    width: 50%;
    padding: 10px;
    height: 130px;
  }
  .row:after {
```
<h1>
    Invoice
</h1>

<table style="width:100%">
    <tr>
        <th>Reference</th>
        <th>Designation</th>
        <th>Quantity</th>
        <th>Order Number</th>
        <th>Unit Price</th>
        <th>Total</th>
    </tr>
    <tr>
        <td></td>
        <td></td>
        <td></td>
        <td></td>
        <td></td>
        <td></td>
    </tr>
</table>

<p> <span class="name"> Total </span> <br> <span class="value"> ---- </span> </p>
BankingDefaultFragment

<h1>
Banking
</h1>

<div class="row">
<div class="bankingcolumn">
<p><span class="name"> Full Name </span> <br> <span class="value"> ---- </span></p>
<p><span class="name"> City </span> <br> <span class="value"> ---- </span></p>
<p><span class="name"> Birth Date </span> <br> <span class="value"> ---- </span></p>
<p><span class="name"> SSN </span> <br> <span class="value"> ---- </span></p>
</div>
<div class="bankingcolumn">
<p><span class="name"> Home Phone </span> <br> <span class="value"> ---- </span></p>
<p><span class="name"> Work Phone </span> <br> <span class="value"> ---- </span></p>
<p><span class="name"> Employer Name </span> <br> <span class="value"> ---- </span></p>
</div>
</div>

<h4>Address</h4>
<table style="width:100%">
<tr>
<th>Address type</th>
<th>Country</th>
<th>ZIP</th>
<th>State</th>
<th>Street</th>
<th>City</th>
</tr>
</table>
<br>

**ContractDefaultFragment**

<h1>
   Contract <br> <hr>
</h1>

<p> <span class="name"> Contract Number </span> <br> <span class="value"> ---- </span> </p>

<p> <span class="name"> Contract Date </span> <br> <span class="value"> ---- </span> </p>

<br>

**LetterDefaultFragment**

<h1>
   Letter <br> <hr>
</h1>

<p> <span class="name"> Letter Date </span> <br> <span class="value"> ---- </span> </p>

<p> <span class="name"> Letter From </span> <br> <span class="value"> ---- </span> </p>

<p> <span class="name"> Letter To </span> <br> <span class="value"> ---- </span> </p>

<br>

**PriceDefaultFragment**

<h1>
   Price <br> <hr>
</h1>

<p> <span class="name"> Company </span> <br> <span class="value"> ---- </span> </p>

<h4>Price Table</h4>

<table style="width:100%">
  <tr>
    <th>Designation</th>
    <th>Unit Price</th>
  </tr>
</table>
Creating a set of activities in Pega

Creating a Pega activity

An activity consists of a set of consecutive steps, each of which carries out a specific method or instruction.

To create a new activity, do the following:

1. In Dev Studio, click Records → Technical → Activity → Create.

2. Fill in the Label and Identifier fields and make sure that the activity is being created in the context of the case (for more details, see Creating a case with a workflow). Next, click Create and open.

3. On the Steps tab that, add the required activity steps.

4. On the Parameters tab, specify the required variables. On the Pages & Classes tab, specify a page to be used for the activity steps.

Setting up activity steps

To configure your activity, you need to create loops and substeps.

To create a loop, click Loop for a step and select a repetition type (in our example, For each element in value list) and a list of values. Click Submit.
To create a substep, create a new step and drag it into the step for which the substep is being created.

**Note:** When setting up the activity, you will also need to set up the following:

- The **When** condition – This condition is checked before a step is carried out. Depending on the result, the step is either carried out or skipped.

- The **Jump** condition – This condition is checked after a step. Depending on the result, the activity can either advance to the next step or finish.

- **Label** specifies a step to which you can go from another step.

**Creating a set of activities for single-document processing case**

- SendForProcessing
- FetchingCapturedData
- CreateError
- VerificationCompleted
**SendForProcessing**

For more information about creating activities, see [Creating a Pega activity](#).

### Pages & Classes

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
<tr>
<td>Attachment</td>
<td>Data-WorkAttach-File</td>
</tr>
<tr>
<td>AttachmentInfo</td>
<td>Data-WorkAttach-File</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseId</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Local variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedRefTo</td>
<td>String</td>
</tr>
<tr>
<td>Id</td>
<td>String</td>
</tr>
<tr>
<td>Index</td>
<td>String</td>
</tr>
</tbody>
</table>

### Steps

**Step 1:**

**Method:** Obj-Open-By-Handle.

**Step page:** Case.

**InstanceHandle:** Param.CaseId.

**Method Parameters**:

- **InstanceHandle**: Param.CaseId
- **Lock**: 
- **ReleaseOnCommit**: 
- **LockInfoPage**: 
- **CheckSecondaryStorage**: 

**Step 2:**
Method: Property-Set.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.Index</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 3:

Method: Property-Set.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.LinkedRefTo</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pxLinkedRefTo</td>
</tr>
<tr>
<td>Local.Id</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pzInsKey</td>
</tr>
</tbody>
</table>

Loop:

Repeat Value List Property

For each element in value list Case.pyAttachments

When:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.Index==0</td>
<td>Continue Whens</td>
<td>Skip Step</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Substep 1:

**Method:** Obj-Open-By-Handle.

**Step page:** Attachment.

**InstanceHandle:** Local.LinkedRefTo

Substep 2:

**Method:** Obj-Open-By-Handle.

**Step page:** AttachmentInfo.

**InstanceHandle:** Local.Id.
Substep 3:

**Method:** Call-Function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABBYY.SendForProcessing(AttachmentInfo.pyMemo, Attachment.pyAttachStream)</td>
<td>.BatchId</td>
</tr>
</tbody>
</table>

For more information, see [SendForProcessing](#).

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**On exception, jump to later step label:** ERR.

Substep 4:

**Method:** Call-Function.
<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABBYY.WaitFirstVerificationOrProcessed(BatchId)</td>
<td>.VerificationUrl</td>
</tr>
</tbody>
</table>

For more information, see [WaitFirstVerificationOrProcessed](#).

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td>Jump to Later Step</td>
<td>INC</td>
</tr>
</tbody>
</table>

**On exception, jump to later step label: ERR**

Substep 5:

**Label:** ERR.

**Method:** Page-Set-Messages.

**Message:** @unescapeHTML(@getWorstMessageName(tools)).

For more information, see [unescapeHTML](#).

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Exit Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Substep 6:

**Label:** INC.

**Method:** Property-Set.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.Index</td>
<td>Local.Index+ 1</td>
</tr>
</tbody>
</table>
FetchingCapturedData
For more information about creating activities, see Creating a Pega activity.

Pages & Classes

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseId</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>jsonData</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>executionMode</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Steps

Step 1:

**Method:** Obj-Open-By-Handle.

**Step page:** Case.

**InstanceHandle:** Param.CaseId.

Method Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceHandle</td>
<td>Param.CaseId</td>
</tr>
<tr>
<td>Lock</td>
<td></td>
</tr>
<tr>
<td>ReleaseOnCommit</td>
<td></td>
</tr>
<tr>
<td>LockInfoPage</td>
<td></td>
</tr>
<tr>
<td>CheckSecondaryStorage</td>
<td></td>
</tr>
</tbody>
</table>

Step 2:

**Method:** Property-Set.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param.jsonData</td>
<td>@ABBYY.FetchingCapturedData(.BatchId)</td>
</tr>
<tr>
<td>Param.executionMode</td>
<td>DESERIALIZE</td>
</tr>
</tbody>
</table>

For more information, see FetchingCapturedData.
Jump:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td>Jump to Later Step</td>
<td>OK</td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.

Step 3:

Label: ERR.


Message: @unescapeHTML(@getWorstMessageName(tools)).

Jump:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Exit Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information, see unescapeHTML.
Step 4:

**Label**: OK.

**Method**: Apply-DataTransform.

**DataTransform**: JsonToInvoiceWithVerificationFields.

**PassParameterPage**: true.

For more information about creating data transforms, see [Creating a data transform for a single-document processing case](#).

**CreateError**

For more information about creating an activity, see [Creating a Pega activity](#).

**Pages & Classes**

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caseld</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>errorMessage</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Steps**

Step 1:
**Method:** Obj-Open-By-Handle.

**Step page:** Case.

**InstanceHandle:** Param.caseId.

**Step 2:**

**Method:** Page-Set-Messages.

**Message:** Param.errorMessage.

**Verification Completed**

**Steps**

**Step 1:**

**Method:** Property-Set

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>.VerificationUrl</td>
<td>&quot;The verification task is completed. Click Submit to proceed.&quot;</td>
</tr>
</tbody>
</table>

**Step 2:**

**Method:** Obj-Save
WriteNow: True

WithError: False

OnlyIfNew: False

Creating a set of activities for multi-document processing case

SendForProcessing

FetchingCapturedData

UpdateVerificationUrl

CreateError

VerificationCompleted
SendForProcessing

For more information about creating activities, see Creating a Pega activity.

Pages & Classes

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
<tr>
<td>Attachment</td>
<td>Data-WorkAttach-File</td>
</tr>
<tr>
<td>AttachmentInfo</td>
<td>Data-WorkAttach-File</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caseld</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Local variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedRefTo</td>
<td>String</td>
</tr>
<tr>
<td>Id</td>
<td>String</td>
</tr>
<tr>
<td>BatchId</td>
<td>String</td>
</tr>
</tbody>
</table>

Steps

Step 1:

**Method**: Obj-Open-By-Handle.

**Step page**: Case.

**InstanceHandle**: Param.Caseld.

Method Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceHandle</td>
<td>Param.Caseld</td>
</tr>
<tr>
<td>Lock</td>
<td></td>
</tr>
<tr>
<td>ReleaseOnCommit</td>
<td></td>
</tr>
<tr>
<td>LockInfoPage</td>
<td></td>
</tr>
</tbody>
</table>

Step 2:
**Method:** Page-Set-Messages

**Message:** "Please add files to case before proceeding."

**When:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttachmentsIsNull Null</td>
<td>Continue Whens</td>
<td>Skip Step</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** In order to create a condition, specify `AttachmentsIsNull` in the **When** field and click the configuration button. If such a condition does not already exist, window will open where you will be able to create a new condition. Make sure that the full name of the case is specified in the **Apply to** field and click **Create and open**. On the **Conditions** tab, create the following condition: `length of .pyAttachments is Equal To 0`. To add the condition, click **Configure advanced conditions here…** and select the `length of [a pagelist property] Is [comparison operator] [value]` for the condition type.

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Exit Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Method Parameters**

- **Page**
- **Category**
- **Message**: "Please add files to case before proceeding."
Step 3:

**Method:** Call-Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABBYY.CreateNewBatch()</td>
<td>Local.BatchId</td>
</tr>
</tbody>
</table>

For more information, see [CreateNewBatch](#).

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**On exception, jump to later step label:** ERR.

Step 4:

**Method:** Property-Set
<table>
<thead>
<tr>
<th>Properties Name</th>
<th>Properties Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.LinkedRefTo</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pxLinkedRefTo</td>
</tr>
<tr>
<td>Local.Id</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pzInsKey</td>
</tr>
</tbody>
</table>

**Loop:**

**Repeat**

For each element in value list

**Value List Property**

Case.pyAttachments

---

**Method Parameters**

**Properties Name**

<table>
<thead>
<tr>
<th>Local.LinkedRefTo</th>
<th>Case.pyAttachments(&lt;CURRENT&gt;).pxLinkedRefTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local.Id</td>
<td>Case.pyAttachments(&lt;CURRENT&gt;).pzInsKey</td>
</tr>
</tbody>
</table>

**Substep 1:**

**Method**: Obj-Open-By-Handle.

**Step page**: Attachment.

**InstanceHandle**: Local.LinkedRefTo.

**Method Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceHandle</td>
<td>Local.LinkedRefTo</td>
</tr>
<tr>
<td>Lock</td>
<td></td>
</tr>
<tr>
<td>ReleaseOnCommit</td>
<td></td>
</tr>
<tr>
<td>LockInfoPage</td>
<td></td>
</tr>
<tr>
<td>CheckSecondaryStorage</td>
<td></td>
</tr>
</tbody>
</table>

**Substep 2:**

**Method**: Obj-Open-By-Handle.
Step page: AttachmentInfo.

InstanceHandle: Local.Id.

Substep 3:

Method: Call-Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABBYY.AddDocumentToBatch(Local.BatchId, AttachmentInfo.pyMemo, Attachment.pyAttachStream)</td>
<td></td>
</tr>
</tbody>
</table>

For more information, see AddDocumentToBatch.

Jump:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.

Method Parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enable conditions after this action

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.
Step 5:

**Method:** Call-Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABBYY.RunBatchProcessing(Local.BatchId)</td>
<td></td>
</tr>
</tbody>
</table>

For more information, see [RunBatchProcessing](#).

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**On exception, jump to later step label:** ERR.

Step 6:

**Method:** Call-Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABBYY.WaitFirstVerificationOrProcessedForMultiFileInvoice(Local.BatchId)</td>
<td>.VerificationUrl</td>
</tr>
</tbody>
</table>

For more information, see [WaitFirstVerificationOrProcessedForMultiFileInvoice](#).

**Jump:**
<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td>Jump to Later Step</td>
<td>OK</td>
</tr>
</tbody>
</table>

**On exception, jump to later step label:** ERR.

**Step 7:**

**Label:** ERR.

**Method:** Page-Set-Messages.

**Message:** @unescapeHTML(@getWorstMessageName(tools)).

For more information, see [unescapeHTML](#).

**Step 8:**

**Label:** OK.

**Method:** Property-Set.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BatchId</td>
<td>Local.BatchId</td>
</tr>
</tbody>
</table>
FetchingCapturedData

For more information about creating activities, see Creating a Pega activity.

Pages & Classes

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseId</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>templateName</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>jsonData</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>executionMode</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Steps

Step 1:

**Method**: Obj-Open-By-Handle.

**Step page**: Case.

**InstanceHandle**: Param.caseId.

Step 2:

**Method**: Property-Set
<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param.executionMode</td>
<td>DESERIALIZE</td>
</tr>
<tr>
<td>Param.jsonData</td>
<td>@ABYY.FetchingCapturedData(BatchId, Param.templateName)</td>
</tr>
</tbody>
</table>

For more information, see FetchingCapturedData.

Jump:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td>Continue</td>
<td>Whens</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Param.templateName==&quot;Invoice_eng&quot;</td>
<td>Jump to Later Step</td>
<td>INVOICE</td>
<td>Continue</td>
<td>Whens</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Param.templateName==&quot;Banking_eng&quot;</td>
<td>Jump to Later Step</td>
<td>BANKING</td>
<td>Continue</td>
<td>Whens</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Param.templateName==&quot;Contract_eng&quot;</td>
<td>Jump to Later Step</td>
<td>CONTRACT</td>
<td>Continue</td>
<td>Whens</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Param.templateName==&quot;Letter_eng&quot;</td>
<td>Jump to Later Step</td>
<td>LETTER</td>
<td>Continue</td>
<td>Whens</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Param.templateName==&quot;Price_eng&quot;</td>
<td>Jump to Later Step</td>
<td>PRICE</td>
<td>Exit Activity</td>
<td></td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.
Step 3:

**Label:** INVOICE.

**Method:** Apply-DataTransform

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTransform</td>
<td>jsonToInvoice</td>
</tr>
<tr>
<td>PassParameterPage</td>
<td>true</td>
</tr>
</tbody>
</table>

For more information, see `jsonToInvoice`.

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
<td>Exit Activity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.
Step 4:

**Label**: BANKING.

**Method**: Apply-DataTransform

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTransform</td>
<td>jsonToBanking</td>
</tr>
<tr>
<td>PassParameterPage</td>
<td>true</td>
</tr>
</tbody>
</table>

For more information, see [jsonToInvoice](#).

**Jump**:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Exit Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.

Step 5:

**Label**: CONTRACT.
**Method:** Apply-DataTransform

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTransform</td>
<td>jsonToContract</td>
</tr>
<tr>
<td>PassParameterPage</td>
<td>true</td>
</tr>
</tbody>
</table>

For more information, see [jsonToContract](#).

**Jump:**

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
<td>Exit Activity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**On exception, jump to later step label:** ERR.

---

### Step 6:

**Label:** LETTER.

**Method:** Apply-DataTransform

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTransform</td>
<td>jsonToLetter</td>
</tr>
<tr>
<td>PassParameterPage</td>
<td>True</td>
</tr>
</tbody>
</table>

For more information, see [jsonToLetter](#).

**Jump:**
Step 7:

**Label**: PRICE.

**Method**: Apply-DataTransform

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTransform</td>
<td>jsonToPrice</td>
</tr>
<tr>
<td>PassParameterPage</td>
<td>True</td>
</tr>
</tbody>
</table>

For more information, see [jsonToPrice](#).

**Jump**:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
<td>Exit Activity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**On exception, jump to later step label**: ERR.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTransform</td>
<td>jsonToPrice</td>
</tr>
<tr>
<td>PassParameterPage</td>
<td></td>
</tr>
</tbody>
</table>
Step 8:

**Label**: ERR.

**Method**: Page-Set-Messages.

**Message**: \[@unescapeHTML(@getWorstMessageName(tools))\].

For more information, see [unescapeHTML](#).

**UpdateVerificationUrl**

For more information about creating activities, see [Creating a Pega activity](#).

### Pages & Classes

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caseld</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Steps

Step 1:

**Method**: Obj-Open-By-Handle.

**Step page**: Case.
Step 2:

**Method**: Property-Set.

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>.VerificationUrl</td>
<td>@ABBYY.WaitFirstVerificationOrProcessedForMultiFileInvoice(.BatchId)</td>
</tr>
</tbody>
</table>

**Jump**:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepStatusFail</td>
<td>Jump to Later Step</td>
<td>ERR</td>
<td>Jump to Later Step</td>
<td>OK</td>
</tr>
</tbody>
</table>

On exception, jump to later step label: ERR.

Step 3:
Label: ERR.


Message: @unescapeHTML(@getWorstMessageName(tools)).

Jump:

<table>
<thead>
<tr>
<th>When</th>
<th>if true</th>
<th>true param</th>
<th>if false</th>
<th>false param</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Exit Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information, see [unescapeHTML](#).

Step 4:

Label: OK.

Method: Obj-Save.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteNow</td>
<td>True</td>
</tr>
<tr>
<td>WithErrors</td>
<td>False</td>
</tr>
<tr>
<td>OnlyIfNew</td>
<td>False</td>
</tr>
</tbody>
</table>
CreateError
For more information about creating activities, see Creating a Pega activity.

Pages & Classes

<table>
<thead>
<tr>
<th>Page name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Work-</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caseld</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>errorMessage</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Steps

Step 1:

**Method**: Obj-Open-By-Handle.

**Step page**: Case.

**InstanceHandle**: Param.caseld.

Step 2:

**Method**: Page-Set-Messages.

**Message**: Param.errorMessage.
VerificationCompleted
Steps

Step 1:

**Method:** Property-Set

<table>
<thead>
<tr>
<th>PropertiesName</th>
<th>PropertiesValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>.VerificationUrl</td>
<td>&quot;The verification task is completed. Click Submit to proceed.&quot;</td>
</tr>
</tbody>
</table>

Step 2:

**Method:** Obj-Save

**WriteNow:** True

**WithError:** False

**OnlIfNew:** False

Creating a set of data transforms in Pega

A **data transform** copies data from the data structure to the properties of a case.

To create a new data transform:

1. In **Dev Studio**, click **Records → Data Model → Data Transform → Create**.

2. In the dialog box that opens, fill in the **Label** and **Identifier** fields.
3. In the **Additional configuration options** section, select **JSON**.

4. In the **Apply to** field, replace "Rule-Obj-Activity" with the full name of the case.

5. Click **Create and open**. In the dialog box that opens, click **Definition** and connect the case fields to the data that will be received from ABBYY FlexiCapture when document processing has finished.

**Creating a data transform for a single-document processing case**

`jsonToInvoiceWithVerificationFields`

For more information about creating data transforms, see [Creating a set of data transforms in Pega](#).
### Creating a set of transforms for a multi-document processing case

**jsonToInvoice**

For more information about creating data transforms, see [Creating a set of data transforms in Pega](#).
<table>
<thead>
<tr>
<th>Action</th>
<th>Clipboard</th>
<th>Relation</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>.InvoiceNumber</td>
<td>equal to</td>
<td>InvoiceNumber</td>
</tr>
<tr>
<td>Set</td>
<td>.InvoiceDate</td>
<td>equal to</td>
<td>InvoiceDate</td>
</tr>
<tr>
<td>Set</td>
<td>.Total</td>
<td>equal to</td>
<td>Total</td>
</tr>
<tr>
<td>Set</td>
<td>.DeliveryAddress</td>
<td>equal to</td>
<td>DeliveryAddress</td>
</tr>
<tr>
<td>Set</td>
<td>.InvoiceCompany</td>
<td>equal to</td>
<td>InvoiceCompany</td>
</tr>
<tr>
<td>Auto-map</td>
<td>.InvoiceTable</td>
<td>To</td>
<td>InvoiceTable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Clipboard</th>
<th>Relation</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Set</td>
<td>.InvoiceNumber</td>
<td>equal to</td>
<td>InvoiceNumber</td>
</tr>
<tr>
<td>2 Set</td>
<td>.InvoiceDate</td>
<td>equal to</td>
<td>InvoiceDate</td>
</tr>
<tr>
<td>3 Set</td>
<td>.Total</td>
<td>equal to</td>
<td>TotalAmount</td>
</tr>
<tr>
<td>4 Set</td>
<td>.DeliveryAddress</td>
<td>equal to</td>
<td>DeliveryAddress</td>
</tr>
<tr>
<td>5 Set</td>
<td>.InvoiceCompany</td>
<td>equal to</td>
<td>Company</td>
</tr>
<tr>
<td>6 Auto-map</td>
<td>.InvoiceTable</td>
<td>To</td>
<td>InvoiceTable</td>
</tr>
</tbody>
</table>
jsonToBanking

<table>
<thead>
<tr>
<th>Action</th>
<th>Clipboard</th>
<th>Relation</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>.SSN</td>
<td>equal to</td>
<td>SSN</td>
</tr>
<tr>
<td>Set</td>
<td>.LastName</td>
<td>equal to</td>
<td>LastName</td>
</tr>
<tr>
<td>Set</td>
<td>.FirstName</td>
<td>equal to</td>
<td>FirstName</td>
</tr>
<tr>
<td>Set</td>
<td>.Initial</td>
<td>equal to</td>
<td>Initial</td>
</tr>
<tr>
<td>Set</td>
<td>.City</td>
<td>equal to</td>
<td>City</td>
</tr>
<tr>
<td>Set</td>
<td>.BirthDate</td>
<td>equal to</td>
<td>BirthDate</td>
</tr>
<tr>
<td>Set</td>
<td>.HomePhone</td>
<td>equal to</td>
<td>HomePhone</td>
</tr>
<tr>
<td>Set</td>
<td>.WorkPhone</td>
<td>equal to</td>
<td>WorkPhone</td>
</tr>
<tr>
<td>Set</td>
<td>.EmployerName</td>
<td>equal to</td>
<td>EmployerName</td>
</tr>
<tr>
<td>Auto-map</td>
<td>.Address</td>
<td>To</td>
<td>Address</td>
</tr>
</tbody>
</table>

Auto-map all data

Top element structure

Object

Action

Set 

Set 

Set 

Set 

Set 

Set 

Set 

Set 

Set 

Auto-map

Add element
### jsonToContract

<table>
<thead>
<tr>
<th>Action</th>
<th>Clipboard</th>
<th>Relation</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>.ContractNumber</td>
<td>equal to</td>
<td>ContractNumber</td>
</tr>
<tr>
<td>Set</td>
<td>.ContractDate</td>
<td>equal to</td>
<td>ContractDate</td>
</tr>
</tbody>
</table>

### jsonToLetter

<table>
<thead>
<tr>
<th>Action</th>
<th>Clipboard</th>
<th>Relation</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>.LetterDate</td>
<td>equal to</td>
<td>LetterDate</td>
</tr>
<tr>
<td>Set</td>
<td>.LetterFrom</td>
<td>equal to</td>
<td>LetterFrom</td>
</tr>
<tr>
<td>Set</td>
<td>.LetterTo</td>
<td>equal to</td>
<td>LetterTo</td>
</tr>
</tbody>
</table>
Creating a flow action

A flow action allows the user to enter and modify case data. In the example below, a flow action will be used to display the verification page and controls for adding files to the case.

To create a flow action, do the following:

2. Fill in the Label and Identifier fields. Next, select the application containing the case and replace "Rule-Obj-Activity" in the Apply to field with the full name of the case.
3. Click **Create and open**. On the **Layout** tab, specify the section name in the **Section** field. Next, click the configuration button to the right of the field.

4. In the dialog box that opens, fill in the **Label** and **Identifier** fields. Next, select the application containing the case and check that the full name of the case type is specified in the **Apply to** field.

5. Click **Create and open**. To view a list of all sections, open **Dev Studio** and click **Records** → **User Interface** → **Section**.
Creating a verification section

To create a verification section, do the following:

1. In Verification Flow Action, create a section named Verification and open it on the Design tab.

2. Add two dynamic layouts. To the first dynamic layout, add an Embedded section named VerificationSection. To the second, add a button named Submit.

VerificationSection

When adding an Embedded section to a dynamic layout:

1. Select Use current page context in the Page context list. Select By name in the Section field and specify the following: VerificationSection.

2. Click the configuration button to create a new section.

3. Click Create and open and then click the HTML tab.

4. Clear the Auto-generated HTML option and enter the code provided below.

Note: This code can also be copied from the VerificationSection.html file, which you can find in %Installation Path%\Connector for FlexiCapture as a Service with verification\Samples\Code examples\Flow actions.

```html
<body>
  <% if (tools.findPage("pyWorkPage").getProperty(".VerificationUrl").getStringValue().contains("https://")) {
    <iframe id="verifyFrame" style="width: 880px; height: 750px;" src="<pega:reference name=".VerificationUrl" mode="javascript" />" onload="changeWidth()"></iframe>
  } else { %>
    <h1><pega:reference name=".VerificationUrl" /></h1>
  <% } %>
</body>
<script>
```
function changeWidth() {

    console.log("Check if the IE11 browser..." acompania);
    if (!!window.MSInputMethodContext && !!document.documentMode) {
        console.log("IE11 browser.");
        return;
    }

    console.log("Other browser. Change width for iframe...");
    document.getElementById("verifyFrame").style = "width: 1000px; height: 750px;";
    console.log("iframe width was changed...");
}

/*
 Add new listener for iframe.
 */
window.addEventListener("message", receiveMessage, false);

function receiveMessage(event) {

    console.log("Check if verification task was closed....");
    if (event.data.includes("FC_Verification_TaskClosed")) {

        console.log("Create new option for VerificationCompleted activity running...");
        var options = {
            name: "VerificationCompleted"
        };

        console.log("Run VerificationCompleted activity...");
        pega.api.ui.actions.runActivity(options);
    }
    console.log("Message was processed.");
}
Creating a custom Submit button

Before creating a button in the dynamic layout, you need to align it to the right of the screen. To do this, open the properties of the dynamic layout, click the Presentation tab, and select Right in the Float field. Click Submit to apply the changes.

To add the button, do the following:

1. Add a basic Button element to the dynamic layout and open its properties.
2. Click the General tab and specify Submit for the button caption text.
3. On the **Presentation** tab, select **Strong** in the **Control format** list.

4. On the **Action** tab, add a **Click** event and connect it to the **Finish Assignment** action.
5. Click **Submit**. To hide the standard set of buttons for the flow (i.e. **Cancel**, **Save**, and **Submit**), open **Verification** by clicking **Records → Process → Flow Action**, click the **Action** tab, and select **Hide the default action section buttons**.

<table>
<thead>
<tr>
<th>Action buttons</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅ Hide the default action section buttons</td>
</tr>
</tbody>
</table>

**Important!** If the **Submit** button is not displayed for your case, select **Standard** in the **Control format** list in **step 3**.

### Creating a file collection section

To create a file collection section, do the following:

1. In **Dev Studio**, click **Records → Process → Flow Action → Collect** and then click the configuration button.

### Layout Validation Action Help setup Security HTML Pages & Classes Specific

**Section configuration**

- **Page context**
  - Use current page context

- **Applies to**
  - InsCo-FlexiCap-Work-InvoiceWithVerification

- **Section**
  - Collect

2. On the **Design** tab, delete the default dynamic layout, add **Layout → AJAX Container**, and specify the following settings in the dialog box that opens:

- **Default view** – **Section**
- **Page context** – **Use other context**
- **Section** – **pyCaseAttachments**
- **Context** – **Work**-
AJAX Container Settings

- Default view: Section
- Page context: Use other context
- Section: pyCaseAttachments
- Context: Work

Accessibility
- Aria Role: Presentation

3. Click OK. The Design tab should now look as follows:

Creating a custom data type

To create a custom data type, do the following:

1. In Dev Studio, open the Data types context menu and select Add data type.
2. Fill in the **Label** and **Description** fields.

**Add Data Type**

- **New Data Type**
- **Existing Data Type**

- **Label**
  - InvoiceTable

- **Description**
  - InvoiceTable

**Advanced**

- **Parent class**
  - InsCo-FlexiCap-Data

- **Identifier**: InsCo-FlexiCap-Data-InvoiceTable

**Choose app layer**

- **FlexiCapture for Invoices**
- **UI Kit**
- **Pega Platform**

**Add to ruleset**

- **FlexiCap**

3. Click **Submit**.

4. Click the **Data model** tab and add the required fields.
## Data types for a single-document processing case

**Line Item**

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article Number</td>
<td>ArticleNumber</td>
<td>Text</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency</td>
<td>Text</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
<td>Text</td>
</tr>
<tr>
<td>Order Number</td>
<td>OrderNumber</td>
<td>Text</td>
</tr>
<tr>
<td>Quantity</td>
<td>Quantity</td>
<td>Text</td>
</tr>
<tr>
<td>Total Price Netto</td>
<td>TotalPriceNetto</td>
<td>Text</td>
</tr>
<tr>
<td>Unit Price</td>
<td>UnitPrice</td>
<td>Text</td>
</tr>
</tbody>
</table>

### Data model

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Scope (applies to)</th>
<th>Type</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article Number</td>
<td>ArticleNumber</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>Currency</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Order Number</td>
<td>OrderNumber</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td>Quantity</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Total Price Netto</td>
<td>TotalPriceNetto</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
<tr>
<td>Unit Price</td>
<td>UnitPrice</td>
<td>Line Item</td>
<td>Text (single line)</td>
<td></td>
</tr>
</tbody>
</table>
Data types for a multi-document processing case

Address

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addresstype</td>
<td>Addresstype</td>
<td>Text</td>
</tr>
<tr>
<td>City</td>
<td>City</td>
<td>Text</td>
</tr>
<tr>
<td>Country</td>
<td>Country</td>
<td>Text</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
<td>Text</td>
</tr>
<tr>
<td>Street</td>
<td>Street</td>
<td>Text</td>
</tr>
<tr>
<td>Zip</td>
<td>Zip</td>
<td>Text</td>
</tr>
</tbody>
</table>

InvoiceTable

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designation</td>
<td>Designation</td>
<td>Text</td>
</tr>
<tr>
<td>Quantity</td>
<td>Quantity</td>
<td>Text</td>
</tr>
<tr>
<td>Reference</td>
<td>Reference</td>
<td>Text</td>
</tr>
<tr>
<td>Total</td>
<td>Total</td>
<td>Text</td>
</tr>
<tr>
<td>UnitPrice</td>
<td>UnitPrice</td>
<td>Text</td>
</tr>
</tbody>
</table>
Configuring your ABBYY FlexiCapture project

Before you start configuring your project, please make sure that you have the following software installed:

- ABBYY FlexiCapture 12 Project Setup Station (required if you are using ABBYY FlexiCapture Distributed Edition or ABBYY FlexiCapture Cloud).

If you already have an ABBYY FlexiCapture project for which you need to set up data export to XML, open that project. Otherwise, you will need to create a project. Launch the Project Setup Station and in the Open Project dialog box click:

- **Create New...** to create a new project.
- **Browse...** to open an existing project stored on your local drive.
- **Browse from Server...** to open a project that you earlier uploaded to the ABBYY FlexiCapture Application Server.
Next, you need to configure ABBYY FlexiCapture as a service with verification:

1. Open the Project Setup Station.
2. Click **Project → Project Properties...** and click the **Workflow** tab. From the **Schema** drop-down list, select **Advanced**.
3. Select the **Verification** option and click **Edit...**
4. In the dialog box that opens, on the **General** tab select the **Enable web stations** option.
5. Click **OK**.

6. Make sure that the **Export root path** on the **General** tab is left empty.
7. Click **OK**.

Next, you need to create an export profile for each document definition.

1. In the ABBYY FlexiCapture Project Setup Station click **Project → Document Definitions...** to open the list of Document Definitions available for the project. Select the Document Definition for which you want to specify new export settings, and click **Edit....**
2. Click **Document Definition → Export Settings**....

3. Click the **Add...** button to start the export profile creation wizard.

4. In the **Select the Type of Destination** step, select **Export to data files** from the **Type** drop-down list, select the **Critical** option below, select the **Errors are irrelevant** item from the **Document condition** drop-down list, and click **Next**.
5. In the **Export Data to Files** step, make sure the **Export path** field is empty and specify a file name template. Click **Next**.

![Export Destination Wizard](image)

6. In the **Data File Format** step, select **XML Document** in the **File type** field, and click **Next**.

![Export Destination Wizard](image)

7. Provide a name for your newly created export profile and click **Finish**.
8. In the **Export Settings** dialog box, check the **Enabled** box for the newly created export profile.

9. Click **Apply**, then click **OK**.
10. Save and publish your Document Definition.