SharePoint Integration without component Using Graph API

Integration Approaches

- **Using REST API**
    - Explanation of SharePoint REST API.
    - Authentication methods (OAuth 2.0, App-only authentication).
    - Examples of REST API calls (GET, POST, PUT).

This document I will focus only how to push file into sharePoint. Before move to development we will test this using postman.

➢ Prerequisites for SharePoint Integration in Pega

Before integrating SharePoint with Pega using a custom approach, ensure you have the following details ready:

1. Required Credentials

- **Client ID** – The unique identifier for your registered Azure AD application.
- **Client Secret** – The secret key generated for your application (used for authentication).
- **Tenant ID -** unique identifier assigned to an Azure Active Directory (Azure AD) instance.
- **Tenant Name**
- **Site Name**

2. Authentication Details

- **Authorization Endpoint:** Used to request an access token.
- **Token Endpoint:** Used to exchange credentials for an access token.
    - https://login.microsoftonline.com/{Tenant-Id} /oauth2/v2.0/token

3. Authentication Method

erandamweerasinghe@gmail.com

- **OAuth 2.0** – Standard authentication mechanism used for secure API communication.

4. Grant Type

- **Client Credentials** – The application authenticates itself without requiring user interaction.

5. SharePoint API Permissions

Ensure that the Azure AD application is granted appropriate **SharePoint API permissions** to access documents, metadata, or perform required operations.

Once these prerequisites are met, you can proceed with configuring the REST Connector in Pega to communicate with SharePoint using OAuth 2.0 authentication.

Step 1: Get SharePoint Site ID

Using this Link fork or download Microsoft postman API collection → **Postman API Collection**

Use this collection: **Microsoft Graph → Delegated →SharePoint**

Other Resources → **Get a site Resources**

Before making API calls to interact with SharePoint, you need to retrieve the **Site ID**. This ID is required for operations like uploading, downloading, and managing files in SharePoint.

API Request to Get Site ID

You can use the **Microsoft Graph API** to fetch the Site ID.

Request URL:

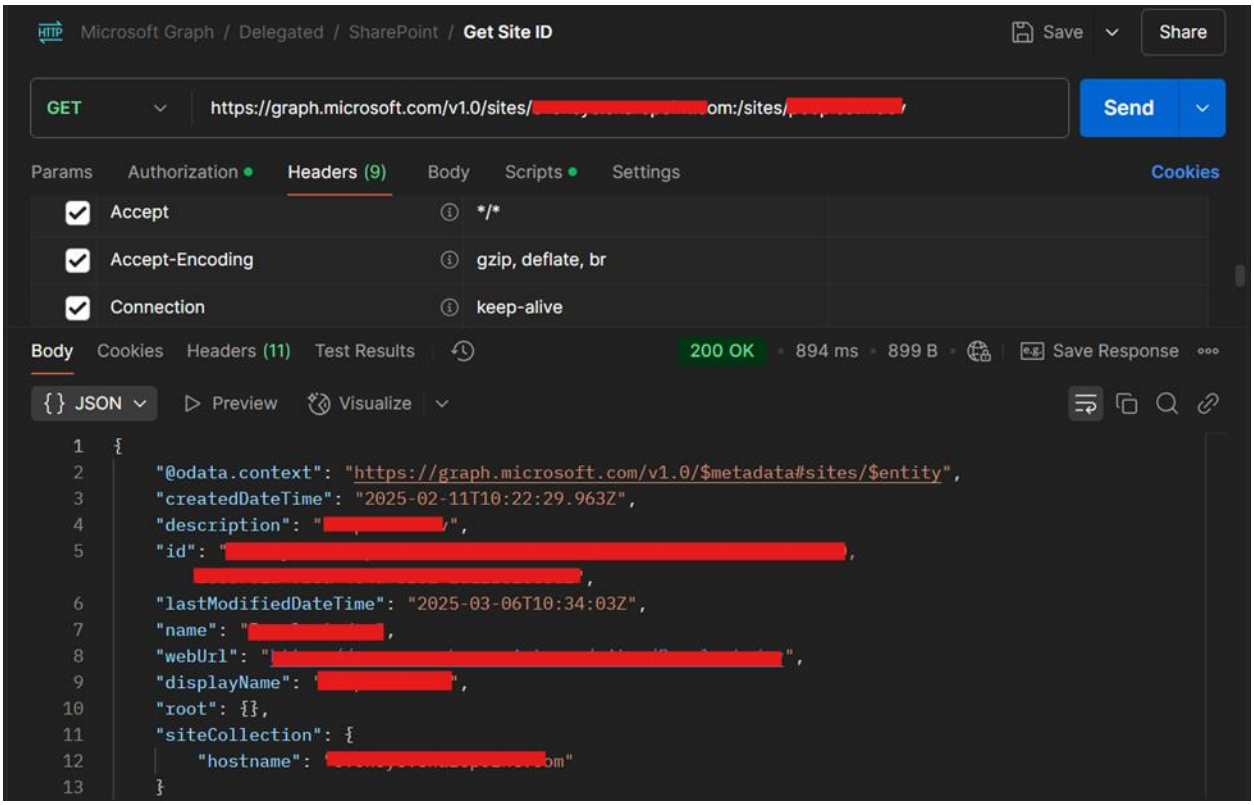
GET **https://graph.microsoft.com/v1.0/sites/{tenant-name}.sharepoint.com:/sites/{site-name}**

Headers:

Authorization: Bearer {access_token}

Content-Type: application/json

erandamweerasinghe@gmail.com



You will get **Id** property from response , save it.

Step 2: Uploading a File to SharePoint Using Microsoft Graph API

Once you have the **Site ID**, you can upload a file to a document library in SharePoint using the **Microsoft Graph API**.

Microsoft API Resources Link → Upload or replace the contents of a driveItem

API Request to Upload a File

Request URL:

PUT https://graph.microsoft.com/v1.0/sites/{site-id}/drive/root:/FolderName/{file-name}:/content

- Replace {site-id} with the actual Site ID retrieved in Step 1.

erandamweerasinghe@gmail.com

- Replace FolderName with the target folder name in SharePoint.
- Replace {file-name} with the name of the file being uploaded.

1.Why is the PUT method used instead of POST?

- The **PUT** method is typically used when **creating or updating** a resource at a specific location.
- In this case, **PUT** ensures that the file is either uploaded or replaced if it already exists in SharePoint.
- If the file exists, **PUT** will overwrite it, whereas **POST** would generally be used to create a new resource without necessarily specifying its exact location.

2. Why use " : " after root and file-name?

- The : notation in the URL is used to indicate a **path-based addressing scheme** in Microsoft Graph API.
- **/root:/FolderName/{file-name}:/content** is a way to specify the **path** where the file should be stored inside SharePoint.
- The : acts as a delimiter between the path and the file operation (/content in this case).

3. What is drive and root in this URL?

- **drive**: Represents a document library in SharePoint. Every site in SharePoint has at least one document library (Drive).
- **root**: Represents the top-level directory of that document library.

- Example:
    - /drive/root → Refers to the **main document library (Documents)**
    - /drive/root:/FolderName/ → Refers to a folder inside the document library

4. Full Explanation of URL

https://graph.microsoft.com/v1.0/sites/{site-id}/drive/root:/FolderName/{file-name}:/content

- sites/{site-id} → Identifies the SharePoint site where the document library exists.
- drive/root → Points to the default **Documents** library.
- :/FolderName/{file-name}:/content → Specifies the file path inside SharePoint and uploads the file's **binary content**.

erandamweerasinghe@gmail.com

Headers:

Authorization: Bearer {access_token}

Content-Type: application/octet-stream

Body:

- The raw binary content of the file must be sent in the request body.

Sample success Response


```
{
  "id": "file-id",
  "name": "example.pdf",
  "size": 12345,
  "webUrl": "https://tenant-name.sharepoint.com/sites/MySite/Shared Documents/example.pdf",
  "createdDateTime": "2025-03-06T12:00:00Z",
  "lastModifiedDateTime": "2025-03-06T12:05:00Z"
}
```


Important Notes:

- If the file is **less than 4 MB**, this method works directly.
- If the file is **larger than 4 MB**, you need to use a **resumable upload session** (chunked upload).

Step 3: Creating an Authentication Profile in Pega

Before creating the **REST Connector**, we need to set up an **Authentication Profile** in Pega to handle OAuth 2.0 authentication. This will allow Pega to securely request an access token from Azure AD before making API calls to SharePoint.

erandamweerasinghe@gmail.com

Step 3.1: Navigate to Authentication Profiles

1. **Go to Dev Studio**.
2. Click **Records → Security → Authentication Profile**.
3. Click **Create** and provide a name (e.g., SharePointOAuthProfile).

Step 3.2: Configure Authentication Profile

1. General Settings

- **Authentication Type:** OAuth 2.0
- **Grant Type:** Client Credentials

2. Configure OAuth 2.0 Details

- **Client ID:** (from Azure AD App registration)
- **Client Secret:** (from Azure AD App registration)
- **Token Endpoint:**

  https://login.microsoftonline.com/{tenant-id}/oauth2/v2.0/token
- **Scope:**

  https://graph.microsoft.com/.default

  Reference → Permission Scopes

3. Headers Configuration

- **Add the following headers:**

  Content-Type: application/octet-stream

4. Token Request Body Parameters

- Add the following key-value pairs in the request body:

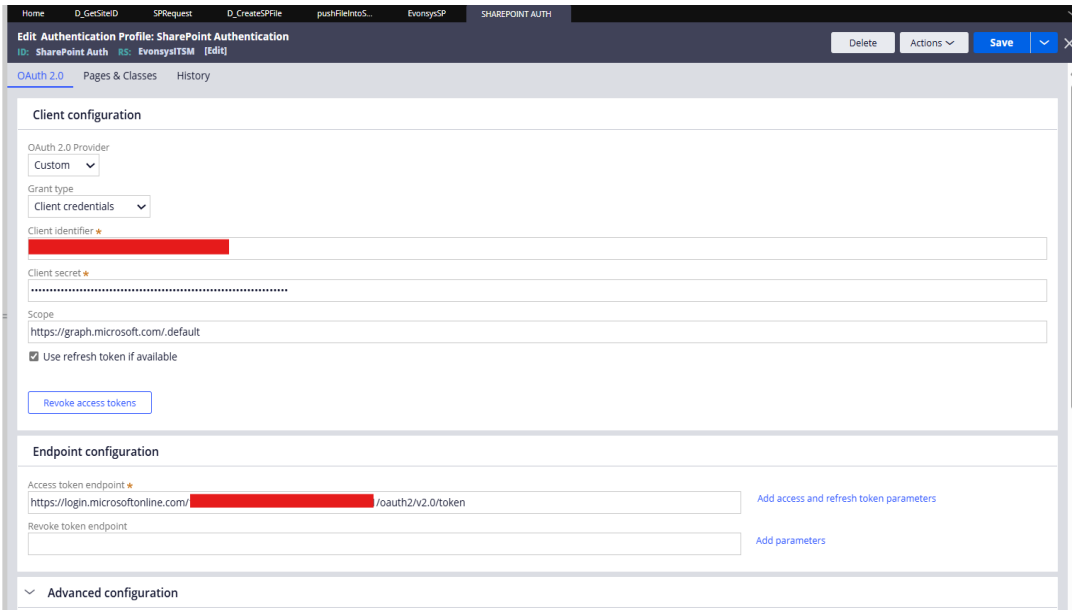
grant_type=client_credentials

client_id={your-client-id}

client_secret={your-client-secret}

erandamweerasinghe@gmail.com

scope=https://graph.microsoft.com/.default



Step 4: Creating a Connect REST Rule in Pega for SharePoint Integration

Now that we have created the **Authentication Profile**, we can proceed with setting up the **Connect REST** rule to interact with SharePoint's API. This will allow us to upload files and perform other operations using the Microsoft Graph API.

Step 4.1: Navigate to Create a Connect REST Rule

1. **Go to Dev Studio.**
2. Click **Create → Integration → Connectors → Connect REST**.
3. Provide the following details:
4. Click **Create and open**.

Step 4.2: Configure the Connector

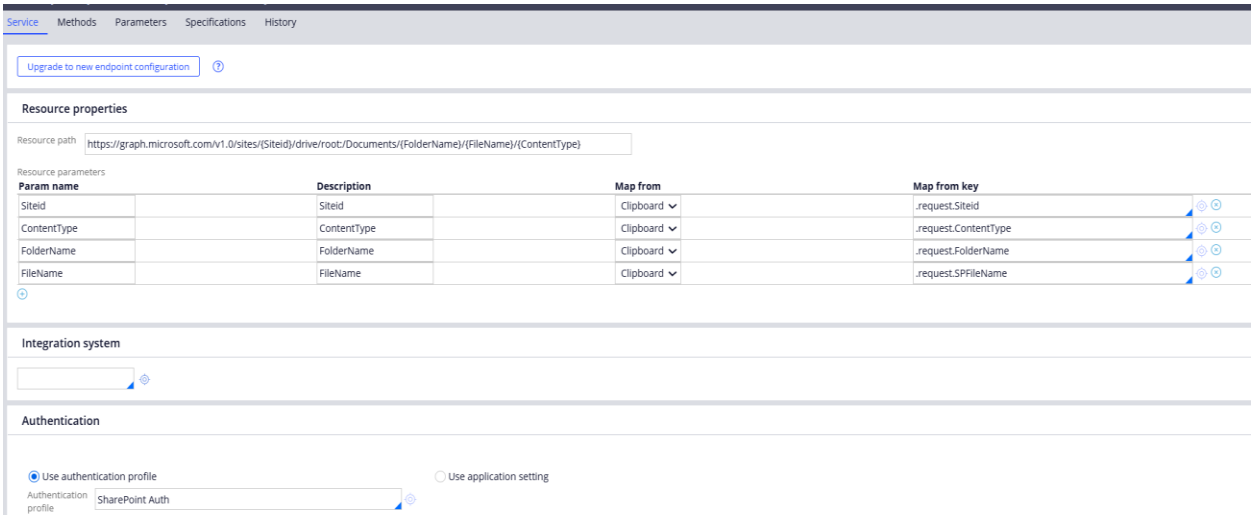
1. Define the Service Endpoint

erandamweerasinghe@gmail.com

- **Resource URL:**

https://graph.microsoft.com/v1.0/sites/{site-id}/drive/root:/FolderName/{file-name}:/content

  - Replace {site-id} with the actual **Site ID** retrieved earlier.
  - Replace **FolderName** with the target SharePoint folder.
  - Replace {file-name} with the file name being uploaded.
- **Authentication Profile:** Select the Authentication Profile you created earlier.
- **HTTP Method:** PUT



Note: Since we are not using the **method to get the resource path from a Data Page**, we will **manually define** the resource path in the Connect REST rule.

Step 4.3: Configure Request & Headers

1. Set Request Headers

- Click on the **Headers tab** and add the following headers:

Authorization: Bearer {access_token}

Content-Type: application/octet-stream

2. Define Request Body

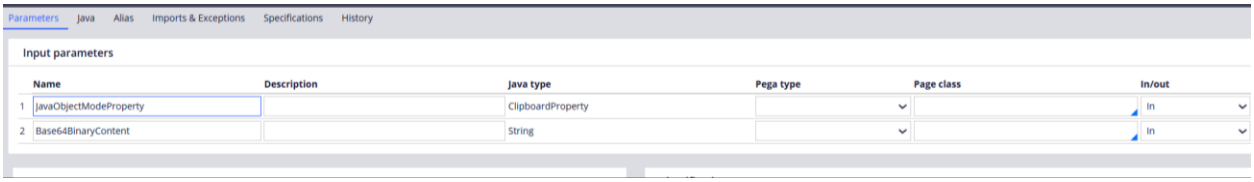
- **Body Type:** Select Binary since we are uploading a file.
- **Data Mapping:**

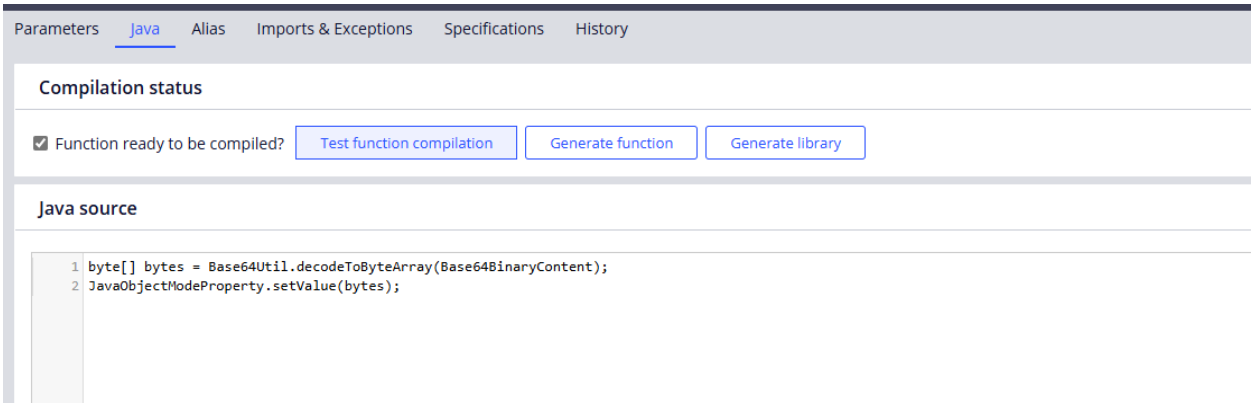
erandamweerasinghe@gmail.com

- **Property Name:** Provide the property where the file content is stored (e.g., .FileContent).

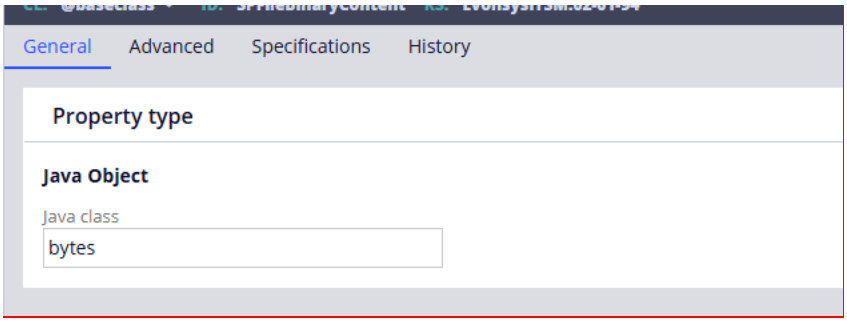
Converting Base64 File to Binary Format and Mapping it to the Request Body

- Since the file is in Base64 format and we need to send it as binary format, we can use a simple function in Pega to convert the Base64 content to binary before mapping it to the request body. Here's how you can do this





File data will hold java object property type



erandamweerasinghe@gmail.com

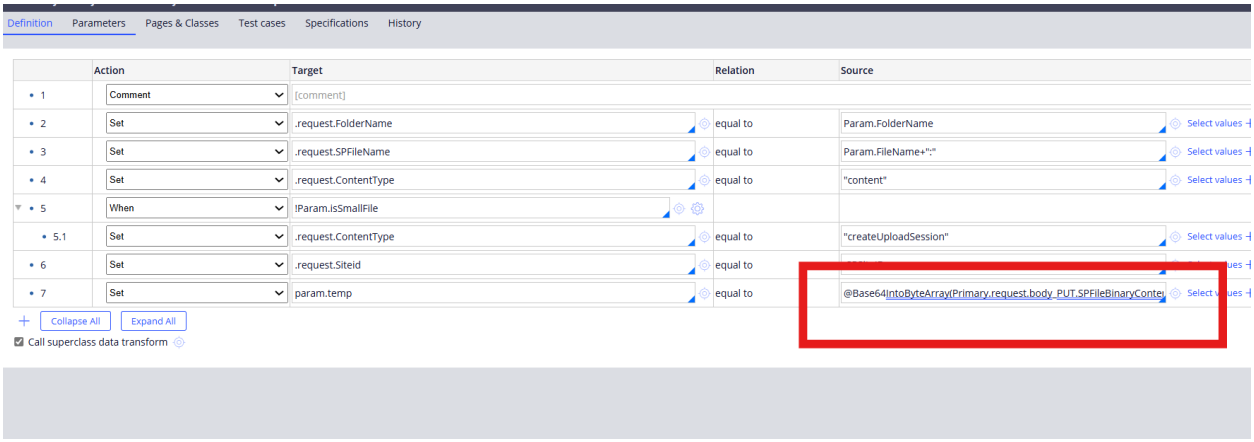
Base64 Conversion and Handling Files

When dealing with files in **Base64 encoding**, you'll convert the Base64 string to its binary representation for actual file transmission. A **Java Object** property type is needed to store the binary content after conversion because Base64, while representing binary data, is a **text string** and doesn't support binary content directly.

The **Java Object** property can store the **byte array** after conversion from Base64, ensuring that the file data can be sent correctly as binary in the API request body.

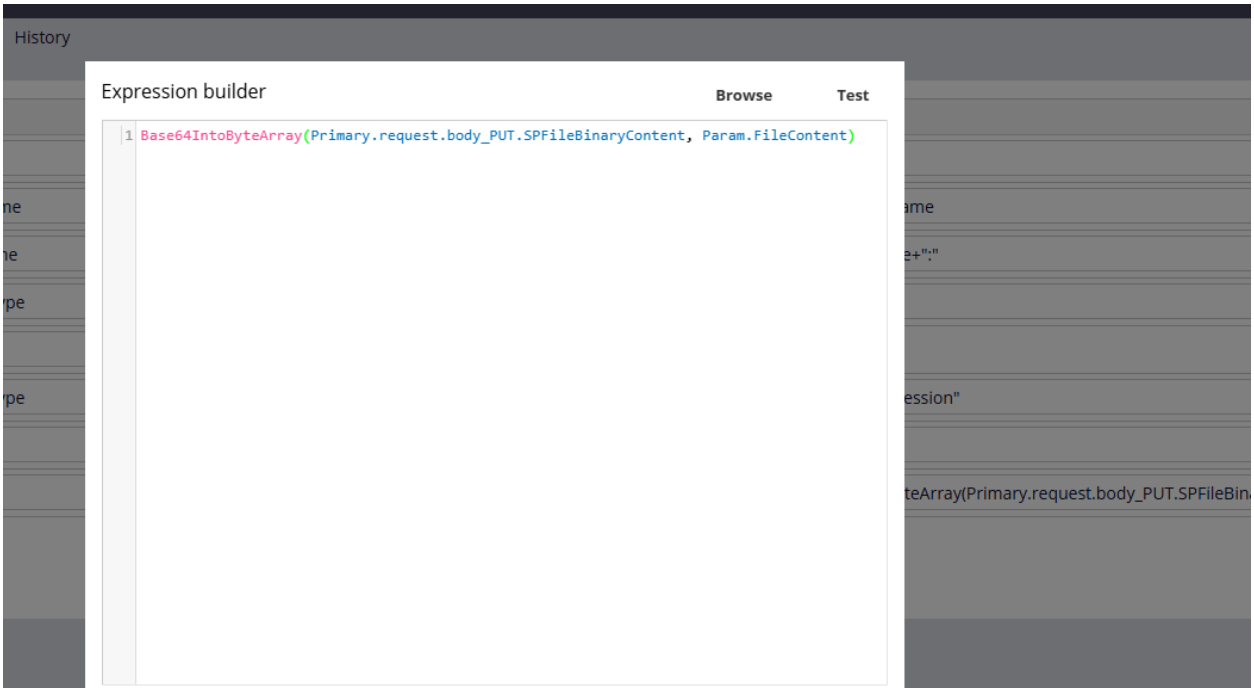
Text Property Limitations

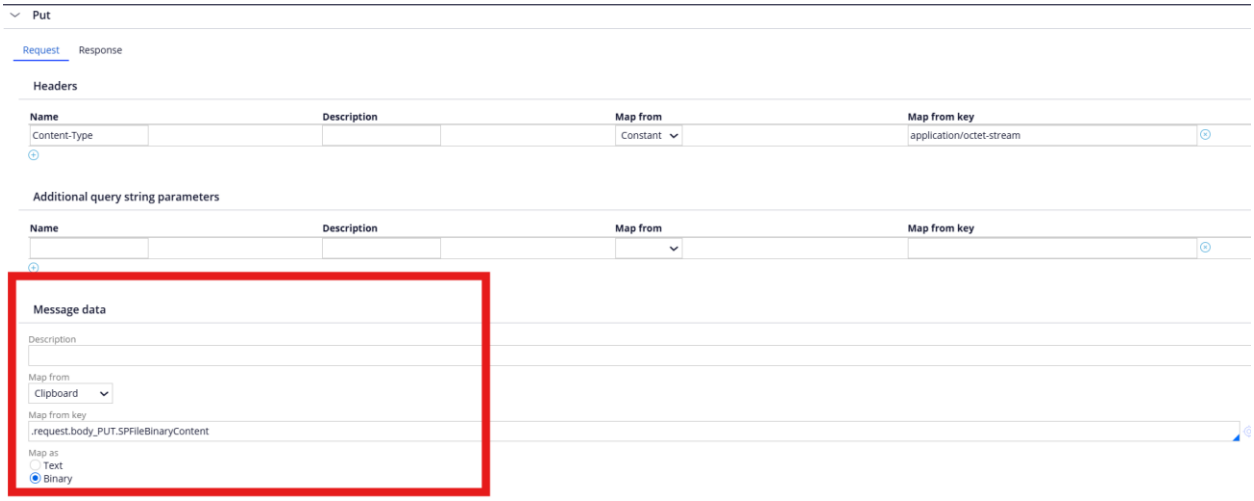
- **Text Property Type** in Pega is limited by the maximum size for text fields and may not be the best choice for large data sets like files or binary streams.
- Text properties are also not suitable for handling **complex data types** like files, as Pega treats them as strings and not as raw binary objects.



The Connector request Data transform map all the request body.

erandamweerasinghe@gmail.com





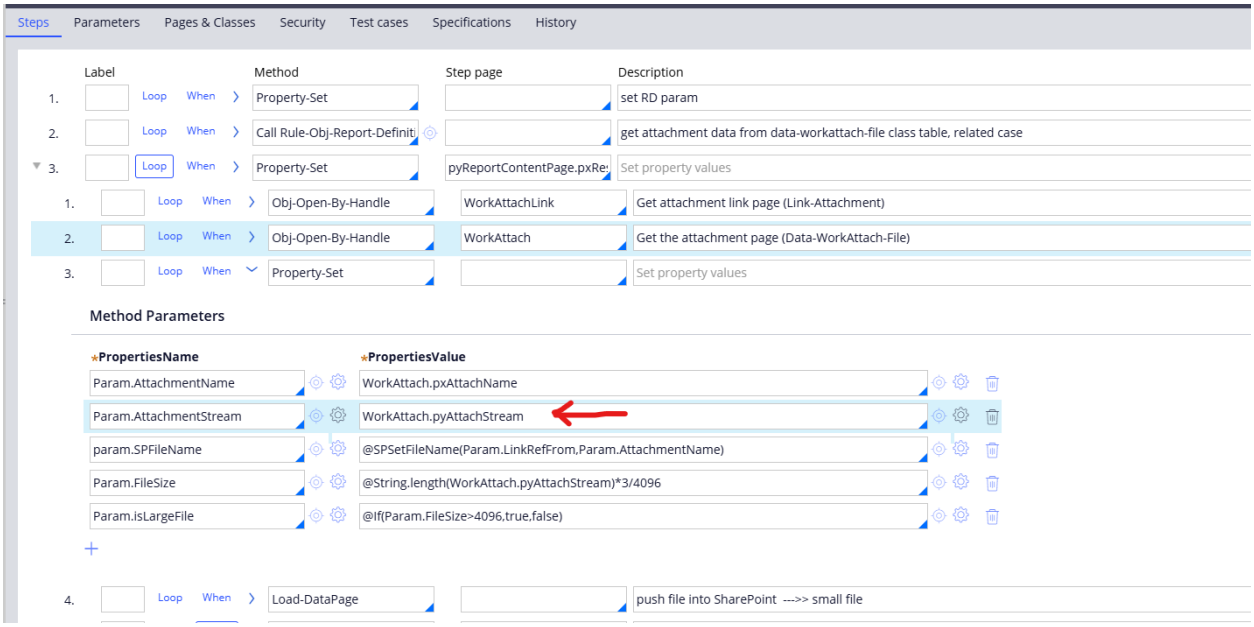
The connector rule map as binary

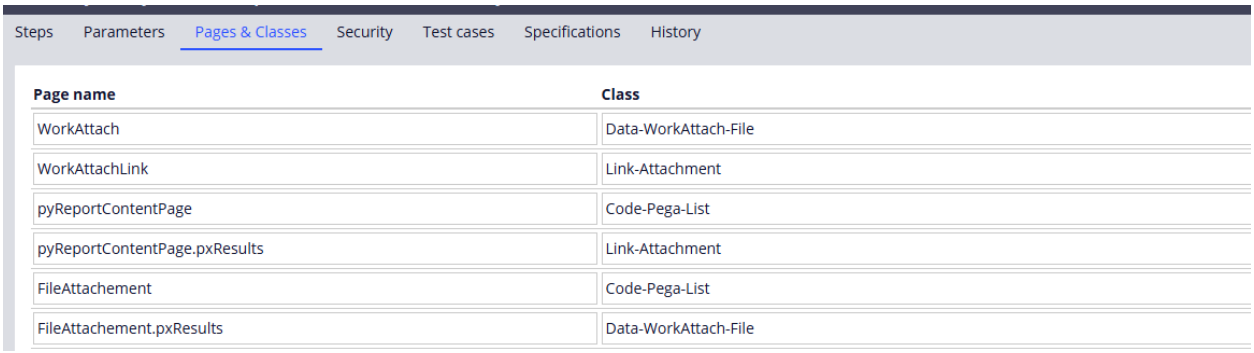
To get the file stream (pyAttachStream) from the **Data-WorkAttach-File** class (which holds the actual file contents) in Pega, you can follow the steps outlined below:

Steps to Get the File Stream from Data-WorkAttach-File Class

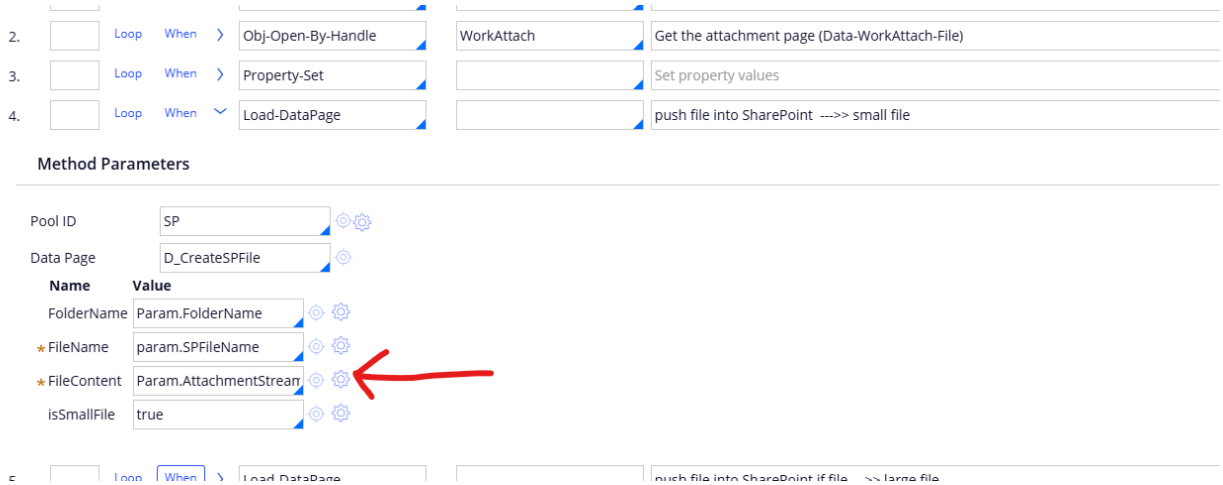
erandamweerasinghe@gmail.com

1. **Identify the Work Object and the Attachment**
    - In Pega, the attachment is linked to a work object. First, ensure that you have the work object associated with the file you want to retrieve. Typically, the file is attached using the **Link-Attachment** class, and the actual file data is stored in the **Data-WorkAttach-File** class.
2. **Use a Property-Set to Retrieve the Attachment Data**
    - You will need to retrieve the pyAttachStream from the Data-WorkAttach-File class. You can achieve this by using a **Data Page** or **Activity** to load the attachment from the table.
3. **Create a Data Page (Optional, but recommended for reusable access)**
    - If you don't already have a Data Page that fetches attachment data, you can create a new one to retrieve the pyAttachStream property from the Data-WorkAttach-File table.





erandamweerasinghe@gmail.com



Passing file content as base64 format

→ Now you can continue coding with your own way …

Next Steps

- **Create an Activity or Data page to Call the Connector.**
- **Handle Response and Exception in Pega.**

Reference:

- Use Postman with the Microsoft Graph API
- Microsoft Graph Permissions Explorer
- Upload or replace the contents of a driveItem
- **Get a site Resources**

…… Happy Learning ……

erandamweerasinghe@gmail.com

erandamweerasinghe@gmail.com