# TechnicalDocumentAutomation\_1\_0\_0\_5 Project Report

This document outlines the technical aspects of the TechnicalDocumentAutomation\_1\_0\_0\_5 project.

# Overview

This system provides an automated solution for generating technical documentation for Power Platform solutions packaged as ZIP files.  
  
The process is initiated when a user uploads a solution ZIP file to a designated SharePoint folder. The system automatically detects the new file and creates a dedicated subfolder within a specified documentation library to store the generated output for that specific solution.  
  
The core of the process involves extracting the solution's metadata from the customizations.xml file contained within the ZIP. This metadata is parsed to identify all individual cloud flows and desktop flows included in the solution. For each identified flow, the system retrieves its definition and utilizes an AI model to generate a comprehensive text description, covering aspects such as the flow's purpose, trigger type, inputs, outputs, external dependencies, core logic, error handling, and connection references.  
  
In parallel, the AI model generates Graphviz DOT code representing the visual structure of each flow. This code is then processed by an external service to produce an SVG flowchart diagram, which is saved alongside the text documentation in the solution's dedicated SharePoint folder. The link to the generated diagram is included in the flow's text description.  
  
After processing all individual flows, the system aggregates the generated text descriptions and uses the AI model again to produce an overall overview of the entire solution and provide potential improvement recommendations. Finally, all the collected documentation components—the overall overview, recommendations, and the detailed information (text description and flowchart link) for each individual flow—are combined and used to populate a predefined Word document template. The resulting formatted document is saved in the solution's documentation folder, and the user who uploaded the original ZIP file receives an email notification confirming the completion of the documentation process and providing a direct link to the output folder.

# Individual Cloud flows

## Generate Formatted Word Document

This flow is designed to automate the generation of a formatted Microsoft Word document based on structured data provided as a JSON input.  
It serves the purpose of taking technical documentation details, populating a predefined Word template with this information, and saving the resulting document to a specified location within a SharePoint Online document library.  
Its role within a larger system is likely to streamline the creation of standardized documentation, such as Technical Design Documents (TDDs), by integrating with systems that can provide the necessary data in a structured format. The flow is manually triggered, indicating it is likely initiated on demand, perhaps from a user interface or another automated process.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Generate%20Formatted%20Word%20Document.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• SharePoint Online  
 • Word Online (Business)

### External flow Dependencies

• SharePoint Online  
 • Word Online (Business)

### Input Variables

• Document xml as json  
 • Destination FolderPath  
 • Document File name

### Output Variables

### Core Logic

The flow is initiated by a manual trigger, which expects three inputs: the document content in JSON format, the destination folder path in SharePoint, and the desired file name for the output document.  
Upon triggering, the flow first composes the input JSON content and then parses it using a 'Parse JSON' action. This action validates the input against a defined schema, extracting key fields such as 'Solution Name', 'Solution Overview', 'Recommendation', 'generatedDate', and a repeating section array for 'Desktop flows'. It also attempts to extract a repeating section for 'Cloud flows', although this is not explicitly defined in the provided schema.  
The destination folder path and the document file name inputs are also composed. The file name is then further processed in another compose action to append the '.docx' file extension.  
The core document generation is performed by the 'Populate a Microsoft Word template' action. This action connects to a specified SharePoint site and document library, using a predefined Word template identified by its unique ID. It maps the extracted data from the parsed JSON and the composed repeating section arrays to corresponding content controls within the Word template.  
After the template is populated, the 'Create file' action takes the generated document content from the previous step and saves it as a new file in the specified SharePoint folder path, using the composed file name with the '.docx' extension.  
Finally, the flow concludes by responding to the trigger with a success status (HTTP 200) and an empty response body, indicating the process has completed.

### Error Handling

### Connection References

• shared\_sharepointonline  
 • shared\_wordonlinebusiness

### Business Exception

### System Exception

## Generate Content with AI Model

This cloud flow is designed to interact with an external AI model API to generate text content based on user-provided prompts.  
  
It is triggered manually, likely from a Power App or another flow, accepting specific inputs for the prompt, flow name, and request type.  
  
The flow retrieves a necessary API key from Azure Key Vault, formats the user's prompt into a structure suitable for the AI model's API, makes an HTTP call to the AI endpoint, processes the response, cleans the generated text, and returns the final output to the caller. Its primary role is to serve as an intermediary between a user interface or process and a generative AI service.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Generate%20Content%20with%20AI%20Model.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• Google Generative Language API (generativelanguage.googleapis.com)  
 • Azure Key Vault  
 • Custom Connector or Service for Text Replacement (fa\_5Fzip-20folder-20operations)

### External flow Dependencies

• Google Generative Language API (generativelanguage.googleapis.com)  
 • Azure Key Vault  
 • Custom Connector or Service for Text Replacement (fa\_5Fzip-20folder-20operations)

### Input Variables

• Prompt  
 • Flow Name  
 • Request Type  
 • $authentication  
 • $connections  
 • AI API Endpoint URL (fa\_AIAPIEndpoint)

### Output Variables

• ai\_output

### Core Logic

The flow begins with a manual trigger that accepts three string inputs: 'Prompt', 'Flow Name', and 'Request Type'.  
  
Following the trigger, the flow retrieves a secret named 'GeminiAPIKey' from a configured Azure Key Vault connection. This key is essential for authenticating with the AI model API.  
  
The user's 'Prompt' input is then composed into a simple variable. This is followed by composing the full request body required by the AI model API. This body includes the user's prompt formatted with a 'user' role and specific generation configuration parameters like temperature, topK, topP, maxOutputTokens, and responseMimeType.  
  
An HTTP POST request is then made to the AI model's API endpoint. The URL for this request is constructed by combining a predefined endpoint URL parameter with the API key retrieved from Key Vault. The request body is the output of the previous compose step. This HTTP action includes a retry policy for robustness.  
  
Upon receiving a response from the AI API, the flow parses the JSON body to extract the generated content. It specifically targets the text content within the first candidate's first part.  
  
The extracted text is then passed to a 'Replace words in text' action, which uses a custom connector. This action cleans the AI output by removing specific strings like '' and '', and replaces escaped plus signs ('[BACKSLASH][BACKSLASH][PLUS]') with a single backslash ('[BACKSLASH]').  
  
The cleaned text is composed into a final output variable.  
  
Finally, the flow responds to the original trigger (likely a Power App or calling flow) by returning the cleaned and formatted AI-generated text in an output variable named 'ai\_output'.

### Error Handling

• HTTP action includes a fixed retry policy with 3 retries at 30-second intervals.

### Connection References

• fa\_5Fzip-20folder-20operations  
 • fa\_AzureKeyVault\_DEVSANDVAULT

### Business Exception

### System Exception

## Retrieve Cloud Flow Details

This flow is designed to extract and format the JSON definition of a specific cloud flow contained within a solution ZIP file.  
It serves as a utility to programmatically access the underlying structure and configuration of a Power Automate cloud flow.  
  
The flow is triggered manually, indicating it's likely initiated on demand, perhaps as part of a larger process for documentation, analysis, or migration of solution components.  
Its primary role is to retrieve the flow's definition and prepare it for further processing, including replacing internal identifiers with human-readable names.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Retrieve%20Cloud%20Flow%20Details.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• SharePoint Online  
 • Custom Connector: ZIP Folder Operations

### External flow Dependencies

• SharePoint Online  
 • Custom Connector: ZIP Folder Operations

### Input Variables

• Solution Zip File ID  
 • FlowID List  
 • Flow Name  
 • Flow json File

### Output Variables

• output  
 • Compose\_-\_Cloud\_Flow\_Information\_json

### Core Logic

The flow begins with a manual trigger that accepts four string inputs: the ID of the solution ZIP file in SharePoint, a list of Flow IDs and their corresponding names (likely in JSON format), the name of the target flow, and the file path to the flow's JSON definition within the ZIP archive.  
  
These inputs are immediately captured into separate Compose actions for easier reference throughout the flow.  
  
Using the provided Solution Zip File ID and the configured SharePoint site URL (from a flow parameter), the flow retrieves the content of the solution ZIP file from SharePoint via the 'Get file content' action.  
  
Subsequently, the 'Get file content from a ZIP archive' action from the custom ZIP operations connector is invoked. It takes the retrieved ZIP file content and the specified Flow json File path to extract the raw content of the target flow's definition file from within the archive.  
  
The extracted content, which is base64 encoded, is then decoded into a standard string format using the 'Compose\_-\_Read\_json\_File' action.  
  
A 'Compose\_-\_Formatted\_Desktop\_Flow\_Information' action structures the extracted flow JSON string and the Flow Name input into a specific text format, likely for subsequent parsing or display.  
  
The 'Replace words in text' action, also from the custom connector, processes the formatted text. It uses the 'FlowID List' input (parsed as JSON) to find and replace internal flow identifiers within the extracted JSON content with their corresponding names, making the definition more readable.  
  
Finally, the result of the replacement is stored in the 'Compose\_-\_Cloud\_Flow\_Information\_json' action, and this content is returned as the 'output' of the flow via the 'Respond to a Power App or flow' action.

### Error Handling

### Connection References

• fa\_sharedsharepointonline\_be196  
 • fa\_sharedfa5fzip20folder20operations5f71779ace6cb872af\_9b6b7

### Business Exception

### System Exception

## Generate Flowchart from Graphviz Code

This cloud flow is designed to automate the process of converting Graphviz DOT code into an SVG flowchart image.  
  
It is triggered manually, allowing a user or another system to initiate the process by providing the necessary inputs: the Graphviz code itself, the desired name for the output file, and the destination folder path within a specified SharePoint document library.  
  
The flow interacts with external services, including a custom connector for text manipulation and a Graphviz API endpoint (like Quickchart.io) to perform the conversion. The resulting SVG image is then saved to SharePoint, and a link to the saved file is returned to the caller. This flow serves as a utility for generating visual diagrams from code, integrating diagram generation into workflows or applications.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Generate%20Flowchart%20from%20Graphviz%20Code.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• SharePoint Online  
 • Custom Connector (fa\_5Fzip-20folder-20operations)  
 • Graphviz API (e.g., Quickchart.io)

### External flow Dependencies

• SharePoint Online  
 • Custom Connector (fa\_5Fzip-20folder-20operations)  
 • Graphviz API (e.g., Quickchart.io)

### Input Variables

• Graphviz Code  
 • Flow name  
 • Destination FolderPath

### Output Variables

• link\_to\_svg\_file

### Core Logic

The flow begins with a Manual trigger, accepting three inputs: the Graphviz DOT code, the desired name for the output SVG file, and the SharePoint folder path where the file should be saved.  
  
These inputs are stored in separate Compose actions for clarity and ease of use in subsequent steps. The Graphviz DOT code input is then passed to a custom connector action ('Replace words in text - Prepare Graphviz Code for API') which performs text replacements to prepare the code for the external API, handling characters like [BACKSLASH] and hyphens.  
  
The processed Graphviz code is sent via an HTTP POST request to the configured Graphviz API endpoint URL. The API is requested to return the output in SVG format. The response from the API, which contains the SVG data typically base64 encoded, is then processed.  
  
A Compose action ('Compose - svg xml text') decodes the base64 content of the API response into plain text SVG XML. This text is then passed to another custom connector action ('Replace words in text - Rescale xml') which modifies the SVG XML, specifically targeting the 'scale(1 1)' attribute and replacing it with 'scale(0.6 0.6)' to rescale the diagram.  
  
The modified SVG XML text is used in a SharePoint 'Create file' action ('Create file - Create SVG Diagram File in SharePoint') to create the .svg file in the specified destination folder path with the provided flow name. This action is configured with chunked transfer mode for potentially large files.  
  
Following the 'Create file' action, there is branching logic based on its outcome. If the file creation succeeds, a SharePoint 'Get file properties' action ('Get file properties - for SVG File') retrieves the properties of the newly created file using its ItemId. If the 'Create file' action fails (e.g., because a file with the same name already exists), a SharePoint 'Get file metadata using path' action ('Get file metadata using path - if file already exists') is executed to find the existing file by its path. If this succeeds, another 'Get file properties' action ('Get file properties - For Existing SVG File') retrieves the properties of the existing file using its ItemId.  
  
A final Compose action ('Compose - SVG File Link Output') is configured to run after either of the 'Get file properties' actions (or if they were skipped). It attempts to retrieve the '{Link}' property from the body of either the 'Get file properties - for SVG File' output or the 'Get file properties - For Existing SVG File' output. Since only one of the branches leading to this compose can succeed or be relevant, it effectively gets the link from the file that was either created or found to exist.  
  
Finally, a 'Respond to a Power App or flow' action returns a JSON response containing the 'link\_to\_svg\_file' property, populated with the SVG file link obtained from the preceding Compose action.

### Error Handling

• Handles the scenario where the file already exists during the 'Create file' action by attempting to retrieve the existing file's properties using its path and ItemId.  
 • 'Compose - SVG File Link Output' is configured to run even if preceding 'Get file properties' actions are skipped, suggesting it attempts to consolidate the link regardless of which path (create or get existing) was taken.

### Connection References

• fa\_5Fzip-20folder-20operations  
 • shared\_sharepointonline

### Business Exception

### System Exception

## Generate Entire Flow Overview and Recommendations

This cloud flow is designed to process a provided text input by sending it to an external AI model via a child flow.  
It is triggered manually, allowing a user to initiate the process and provide the text directly.  
The primary function is to leverage AI capabilities to generate an overview and recommendations based on the input text, likely related to code analysis or documentation, and then return the AI-generated output.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Generate%20Entire%20Flow%20Overview%20and%20Recommendations.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• Child Flow: Generate Content with AI Model  
 • AI Model (accessed via child flow)

### External flow Dependencies

• Child Flow: Generate Content with AI Model  
 • AI Model (accessed via child flow)

### Input Variables

• Entire Flow Text

### Output Variables

• entire\_flow\_json

### Core Logic

The flow begins with a manual trigger that accepts a text input labeled 'Entire Flow Text'.  
This input text is immediately stored in a Compose action named 'Compose - Entire Flow Text Input' for easy reference.  
The core processing is encapsulated within a Scope action named 'Scope - Generate Entire Flow Overview and Recommendations'.  
Inside the scope, a Compose action named 'Compose - AI Prompt - Entire Flow Overview and Recommendations' constructs the specific prompt text required by the AI model, incorporating the initial text input.  
This prompt is then passed as input to a child flow action named 'Run a Child Flow - AI Request for Entire Flow Overview and Recs', which calls the 'Generate Content with AI Model' flow.  
The output from the child flow, specifically the 'ai\_output' property, is captured by another Compose action named 'Compose - AI Output - Entire Flow Overview and Recommendations'.  
Finally, the flow concludes with a 'Respond to a Power App or flow' action, returning the AI-generated output captured in the previous Compose action as a JSON property named 'entire\_flow\_json'.

### Error Handling

### Connection References

### Business Exception

### System Exception

## Process Individual Desktop Flow

This flow is designed to process documentation for a single desktop flow. It takes inputs related to the desktop flow's data, IDs, name, and a destination folder path.  
It orchestrates calls to child flows to retrieve the desktop flow's code write-up, generate a text description and a flowchart (in DOT format) using an AI model, and then generate an SVG image from the DOT code.  
Finally, it compiles the generated text description, parsed AI output, and the flowchart link into a structured JSON object for documentation purposes and returns this information.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Process%20Individual%20Desktop%20Flow.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• SharePoint Online  
 • AI Model  
 • Graphviz

### External flow Dependencies

• SharePoint Online  
 • AI Model  
 • Graphviz

### Input Variables

• Flow Input  
 • Flow List Ids  
 • Flow Name  
 • Destination FolderPath

### Output Variables

• Flow Text  
 • Flow json

### Core Logic

The flow is triggered manually, accepting inputs for the desktop flow data, IDs, name, and destination folder path.  
It first calls a child flow, 'Retrieve Desktop Flow Details', passing the input data to obtain a code write-up for the specified desktop flow.  
Concurrently or subsequently, it prepares an AI prompt using the retrieved desktop flow write-up to generate a detailed text description of the flow's purpose, logic, etc.  
It calls another child flow, 'Generate Content with AI Model' (specifically for 'Unit Flow Text Generation'), with this prompt to get the text description.  
In parallel, it attempts to check if an SVG flowchart file for the desktop flow already exists in the specified SharePoint destination folder.  
If the SVG file exists, it retrieves its properties, including the link.  
If the SVG file does not exist (or the check fails), it prepares a separate AI prompt using the desktop flow write-up to generate Graphviz DOT code for a flowchart, following specific styling and simplification rules.  
It calls the 'Generate Content with AI Model' child flow again (for 'Unit Flowchart Generation') with the DOT code prompt.  
It extracts the generated DOT code from the AI output.  
It then calls a third child flow, 'Generate Flowchart from Graphviz Code', passing the DOT code, flow name, and destination folder path to create the SVG file on SharePoint and obtain its link.  
A compose action determines the final SVG link, prioritizing the existing file's link if found, otherwise using the newly generated link. It includes a warning if no link could be obtained.  
The flow parses the JSON output received from the AI text generation step, extracting structured information like overview, variables, dependencies, error handling, etc.  
A scope is used to convert the various lists obtained from the parsed JSON (Input Variables, Output Variables, External Infrastructure, External Dependencies, Error Handling, Connection References, Business Exceptions, System Exceptions) into bulleted string formats using 'Select' and 'Join' actions.  
Finally, a compose action assembles a comprehensive JSON object containing all the extracted and generated documentation details, including the desktop flow name, overview (with the SVG link), bulleted lists for various components, core logic, and exceptions.  
The flow concludes by responding to the caller (Power App or flow) with the original desktop flow code write-up and the newly composed documentation JSON.

### Error Handling

• Retry policies configured for child flow calls (3 retries for 'Retrieve Desktop Flow Details', 'AI Text Generation', 'Generate Flowchart SVG'; 2 retries for 'AI Flowchart DOT Code Generation')  
 • Conditional execution paths allowing the flow to proceed even if the SVG file existence check fails (using 'runAfter' on 'Failed' status)  
 • Explicit check for an empty SVG link and inclusion of a warning message in the output JSON  
 • The final response action is configured to run regardless of the success or failure of the main documentation generation scope, ensuring a response is always attempted.

### Connection References

• shared\_sharepointonline

### Business Exception

### System Exception

## Extract Solution XML to JSON

This cloud flow is designed to process a Dynamics 365 solution zip file retrieved from SharePoint.  
It is triggered manually, accepting the ID of the zip file as input.  
Its primary purpose is to extract the 'customizations.xml' file from within the zip, convert its XML content into a JSON object, and return the resulting JSON along with a status indicating success or failure.  
This flow plays a role in automating the analysis or processing of solution components by providing a structured JSON representation of the customizations.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Extract%20Solution%20XML%20to%20JSON.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• SharePoint Online  
 • fa\_5Fzip-20folder-20operations (Zip Folder Operations API)

### External flow Dependencies

• SharePoint Online  
 • fa\_5Fzip-20folder-20operations (Zip Folder Operations API)

### Input Variables

• Zip File ID  
 • TDD Solution Site (fa\_TDDSolutionSite)

### Output Variables

• status  
 • json\_output

### Core Logic

The flow starts with a manual trigger that accepts a 'Zip File ID' as input.  
A string variable named 'status' is initialized to store the processing result.  
The input 'Zip File ID' is composed for use in subsequent actions.  
The flow then retrieves the content of the specified zip file from the SharePoint site defined by the 'TDD Solution Site' parameter.  
Using a dedicated connector, it lists the files contained within the retrieved zip file content.  
A filter array action is used to find entries in the zip file list where the file name is 'customizations.xml', performing a case-insensitive comparison.  
A condition checks if the number of items found by the filter (i.e., 'customizations.xml' files) is greater than 1.  
If the condition is TRUE (more than one 'customizations.xml' found), the flow executes the 'if' branch, setting the 'status' variable to a specific Business Exception message.  
If the condition is FALSE (0 or 1 'customizations.xml' found), the flow executes the 'else' branch. In this branch, it reads the content of the first item found by the filter (assuming one was found), decodes its base64 content into a string, converts this XML string into a JSON object, and sets the 'status' variable to 'success'.  
Finally, the flow responds to the manual trigger, returning the final value of the 'status' variable and the resulting JSON object (which will be null if the Business Exception branch was taken).

### Error Handling

• Checks if the number of 'customizations.xml' files found is greater than 1. If true, sets a specific Business Exception status message.

### Connection References

• fa\_sharedsharepointonline\_be196  
 • fa\_sharedfa5fzip20folder20operations5f71779ace6cb872af\_9b6b7

### Business Exception

• Business Exception - There is no customization.xml file in the Zip

### System Exception

## Process Individual Cloud Flow

This cloud flow is designed to process a single cloud flow definition and generate comprehensive documentation for it.  
It is triggered manually via a button in a Power App or another flow.  
Its primary role is to orchestrate the retrieval of flow details, generate a text description and a flowchart using AI models, handle the creation or retrieval of the flowchart SVG file, and format the generated information into a structured JSON output.  
This flow is a key component in an automated documentation generation system.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Process%20Individual%20Cloud%20Flow.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• shared\_sharepointonline

### External flow Dependencies

• shared\_sharepointonline

### Input Variables

• Solution Zip File ID  
 • Flow List Ids  
 • Flow Name  
 • Destination FolderPath  
 • Flow json File

### Output Variables

• Flow Text  
 • Flow json

### Core Logic

The flow starts with a manual trigger that accepts five string inputs: 'Solution Zip File ID', 'Flow List Ids', 'Flow Name', 'Destination FolderPath', and 'Flow json File'.  
These inputs are immediately composed into separate variables for easier use.  
The flow then calls a child flow named 'Retrieve Cloud Flow Details', passing the composed inputs. This child flow is expected to return the detailed definition or write-up of the specified cloud flow.  
The output from the 'Retrieve Cloud Flow Details' child flow is composed into a variable named 'Compose - Cloud Flow Defination'.  
An AI prompt is constructed ('Compose - AI Prompt - Generate Flow Text Description') using the retrieved flow definition and the flow name. This prompt instructs an AI model to generate a structured JSON description of the flow, covering its purpose, trigger, inputs, outputs, logic, error handling, etc.  
Another child flow, 'Generate Content with AI Model', is called with this text generation prompt.  
Following the text generation, the flow enters a scope ('Scope - Check if SVG file has been created') dedicated to handling the flowchart generation and file management.  
Inside this scope, it first attempts to retrieve metadata for an SVG file corresponding to the flow name in the specified destination folder path using the SharePoint connector.  
If the file metadata retrieval fails (indicating the file does not exist), an AI prompt ('Compose - AI Prompt - Generate Flowchart DOT Code') is composed using the flow definition, requesting DOT Graphviz code for a flowchart.  
The 'Generate Content with AI Model' child flow is called again with this DOT code generation prompt.  
The generated DOT code is extracted from the AI output.  
A third child flow, 'Generate Flowchart from Graphviz Code', is called to convert the DOT code into an SVG file and save it to the destination folder, returning a link to the file.  
If the initial file metadata retrieval succeeds (indicating the file already exists), the flow retrieves the properties of the existing file using the SharePoint connector to get its link.  
After the SVG scope completes (either by generating a new file or finding an existing one), the flow composes the SVG link ('Compose - SVG Link') by combining the potential links from the two paths. Since only one path succeeds, one link will be valid and the other empty.  
A check is performed ('Compose - Check if Link is empty') to see if the final SVG link is empty. If it is, a warning message is composed; otherwise, the actual link is used.  
The JSON output from the AI text generation child flow is parsed ('Parse JSON - Individual Flow JSON') based on a predefined schema.  
A subsequent scope ('Scope - Convert lists to bullet points') processes various arrays from the parsed JSON (Input Variables, Output Variables, External Infrastructure Used, External flow Dependencies, Error Handling, Connection References, BusinessException, SystemException) using 'Select' actions to format each item with a bullet point.  
Finally, the flow composes the final output JSON object ('Compose - Flow json') by combining the parsed data and the bulleted lists. Note that the 'External Infrastructure List' and 'External flows Used' fields in the output JSON are both populated using the bulleted list derived from the 'External Infrastructure Used' array from the AI output.  
The flow concludes by responding to the caller ('Respond to a Power App or flow'), returning the raw flow definition text and the composed documentation JSON.

### Error Handling

• Retry policy on 'Run a Child Flow - Retrieve Cloud flow info' (fixed, 3 retries, 30s interval)  
 • Retry policy on 'Run a Child Flow - AI Text Generation for Flow' (fixed, 3 retries, 20s interval)  
 • Retry policy on 'Run a Child Flow - AI Flowchart DOT Code Generation' (fixed, 2 retries, 30s interval)  
 • Retry policy on 'Run a Child Flow - Generate Flowchart SVG' (fixed, 3 retries, 20s interval)  
 • Conditional execution path in 'Scope - Check if SVG file has been created' based on success/failure of checking for existing SVG file  
 • Warning message composed if SVG diagram link is empty ('Compose - Check if Link is empty')  
 • Final response action ('Respond to a Power App or flow') is configured to run after the main documentation scope regardless of its status (Succeeded, Skipped, Failed, TimedOut)

### Connection References

• shared\_sharepointonline

### Business Exception

### System Exception

## Retrieve Desktop Flow Details

This cloud flow, named 'Retrieve Desktop Flow Details', is designed to process and extract detailed information about a specific Power Automate Desktop flow.  
It serves as a utility flow, likely called from another process (such as a parent cloud flow or application) that provides the raw JSON output representing the desktop flow's structure and metadata.  
Its primary role is to parse this complex input, format key details like inputs, outputs, dependencies, connection references, and the flow definition itself, and then present this information in a human-readable format.  
The flow also includes logic to replace Desktop flow IDs found within the definition or dependencies with corresponding flow names, enhancing readability and understanding of the desktop flow's structure and relationships.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Retrieve%20Desktop%20Flow%20Details.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• shared\_fa-5fzip-20folder-20operations-5f71779ace6cb872af (Zip Folder Operations connector)

### External flow Dependencies

• shared\_fa-5fzip-20folder-20operations-5f71779ace6cb872af (Zip Folder Operations connector)

### Input Variables

• Desktop flow info  
 • FlowID List  
 • Flow Name

### Output Variables

• output

### Core Logic

The flow is triggered manually, accepting three string inputs: 'Desktop flow info' (expected to be a JSON string containing desktop flow details), 'FlowID List' (expected to be a JSON string mapping desktop flow IDs to names for replacement), and 'Flow Name' (though this input is not used in the subsequent logic).  
  
The core processing happens within the 'Scope - Extract and Format Desktop Flow Info'. First, the 'Desktop flow info' input is composed and then parsed as JSON ('Parse JSON - Desktop Flow Info'). This action extracts various fields from the input JSON, including '@[BACKSLASH]@[BACKSLASH]WorkflowId', '@[BACKSLASH]@[BACKSLASH]Name', 'JsonFileName', 'Type', 'RunAs', 'UIFlowType', 'Inputs', 'Outputs', 'Dependencies', 'ConnectionReferences', and 'Definition'.  
  
Following the parsing, the flow proceeds to format specific sections. 'Scope - Format Inputs' parses the 'Inputs' string as JSON and extracts the 'required' list from the schema. 'Scope - Format Dependencies' parses the 'Dependencies' string as JSON and extracts the 'childFlows' list. 'Scope - Format Outputs' parses the 'Outputs' string as JSON and checks if a 'schema' exists. If it does, it uses the 'Zip Folder Operations' connector ('Extract keys from a JSON object - Extract Output Keys') to extract keys from the output schema properties and composes an output list.  
  
'Scope - Format Definition' focuses on cleaning up the desktop flow definition text. It composes the raw 'Definition' string and defines a list of replacement rules ('Compose - Definition Replacement JSON'). These rules use regex and literal matches to remove or replace specific patterns (like internal annotations, import/disable statements, appmasking, and renaming 'External.RunFlow FlowId' to 'Run Desktopflow' and 'function' to 'Subflow'). The 'Zip Folder Operations' connector ('Replace words in text - Format Flow Definition Text') applies these replacements to the definition text, and the result is composed as 'Formatted Definition Text'.  
  
After formatting the individual sections, the flow composes an initial formatted string ('Compose - Formatted Desktop Flow Information') combining the flow name, inputs, outputs, dependencies, connection references, and the formatted definition text. Finally, it uses the 'Zip Folder Operations' connector ('Replace words in text - Replace Workflow IDs with Flow Names') to replace any Desktop flow IDs present in the formatted string with their corresponding names provided in the 'Flow ID List' input JSON. The result is composed as 'Extracted Desktop Flow Information'.  
  
The flow concludes by responding to the trigger ('Respond to a Power App or flow') with the final 'Extracted Desktop Flow Information' string.

### Error Handling

### Connection References

• fa\_5Fzip-20folder-20operations

### Business Exception

### System Exception

## Process Flows in Solution Zip

This Power Automate flow is designed to process a solution zip file to extract information about the Cloud flows and Desktop flows contained within it. It takes the ID of the zip file, a destination folder path, and a desired document file name as inputs.  
  
The flow orchestrates several child flows to achieve its purpose. It first extracts and converts the customization XML from the solution zip into a JSON format. It then identifies and processes each Cloud flow and Desktop flow found in the XML, aggregating information about them.  
  
Finally, it uses the collected information to generate a structured document summarizing the flows, including an overall overview, details for each flow, and recommendations. The trigger type for this flow is Manual.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Process%20Flows%20in%20Solution%20Zip.svg

### Trigger Overview

Manual Type

### External Infrastructure Used

• Power Automate (for calling child flows)  
 • File Storage (implicitly used by child flows for document generation/storage)

### External flow Dependencies

• Power Automate (for calling child flows)  
 • File Storage (implicitly used by child flows for document generation/storage)

### Input Variables

• Zip File ID  
 • Destination FolderPath  
 • Document File name

### Output Variables

• Entire Desktop flow Info  
 • Entire Cloud flow Info  
 • Desktop flows Repeating sections  
 • Cloud flows Repeating sections  
 • Business Exceptions  
 • Connection References  
 • Compose - Document XML Content as JSON

### Core Logic

The flow begins with a Manual trigger, accepting inputs for the zip file ID, destination folder path, and document file name. These inputs are immediately composed into separate outputs for easier reference.  
  
The first major step is calling the child flow 'Extract Solution XML to JSON', passing the Zip File ID input. The output from this child flow, expected to be the customization XML converted to JSON, is then parsed using the 'Parse JSON - Customization XML JSON' action.  
  
Several variables are initialized: 'Entire Desktop flow Info' (string), 'Entire Cloud flow Info' (string), 'Desktop flows Repeating sections' (array), 'Cloud flows Repeating sections' (array), 'Business Exceptions' (array), and 'Connection References' (array). Note that the 'Business Exceptions' and 'Connection References' variables are initialized but not populated or used in the subsequent steps shown in this definition.  
  
A 'Scope - Get Flow IDs' is executed to process the parsed customization XML. It uses a 'Select' action to build a list of Workflow IDs and their corresponding Names from the 'Workflows' section of the XML. This list is then composed into a string output, likely intended for use in child flows for replacing IDs with names.  
  
Two parallel scopes, 'Scope - Process Desktop flows' and 'Scope - Process Cloud flows', are then executed. The 'Process Desktop flows' scope filters the workflows from the parsed XML where the 'Category' is '6' (representing Desktop flows). It then iterates through each filtered Desktop flow using a 'For each' loop. Inside the loop, it composes the individual flow's XML as a string and its name. It calls the child flow 'Process Individual Desktop Flow', passing the flow XML string, the composed Flow ID list, the flow name, and the destination folder path. The text output ('flow\_text') from this child flow is appended to the 'Entire Desktop flow Info' string variable, and the JSON output ('flow\_json') is parsed and appended to the 'Desktop flows Repeating sections' array variable.  
  
Similarly, the 'Process Cloud flows' scope filters workflows where the 'Category' is '5' (representing Cloud flows). It iterates through each Cloud flow. Inside its loop, it composes the individual flow's XML string, its 'JsonFileName', and its name. It calls the child flow 'Process Individual Cloud flow', passing the zip ID, Flow ID list, flow name, destination path, and the JSON file name. The text output ('flow\_text') from this child flow is appended to the 'Entire Cloud flow Info' string variable, and the JSON output ('flow\_json') is parsed and appended to the 'Cloud flows Repeating sections' array variable.  
  
After both the Desktop and Cloud flow processing scopes complete, the flow calls the child flow 'Generate Entire Flow Overview and Recommendations'. It passes the concatenated content of the 'Entire Desktop flow Info' and 'Entire Cloud flow Info' string variables as input, separated by '[NEWLINE][BACKSLASH][BACKSLASH]n[NEWLINE][BACKSLASH][BACKSLASH]n'. The JSON output from this child flow is composed and then parsed to extract 'Overview' and 'Improvement Recommendation' properties.  
  
A 'Compose - Document XML Content as JSON' action builds a final JSON object containing the 'Solution Name' (from input), the extracted 'Solution Overview', the 'repeatingSection Desktop flows' array, the 'repeatingSection Cloud flows' array, the extracted 'Recommendation', and the 'generatedDate' (current UTC date formatted). This JSON object represents the content structure for the final document.  
  
The flow then calls the child flow 'Generate Formatted Word Document', passing the string representation of the composed document content JSON, the destination folder path input, and the document file name input. This child flow is responsible for creating the actual document.  
  
Finally, the flow responds to the trigger (likely a Power App or another flow) with a status code 200 and an empty body.

### Error Handling

• Retry Policy: Child flow calls ('Run a Child Flow - Process Entire Flow Text', 'Run a Child Flow - Process Individual Desktop flow', 'Run a Child Flow - Process Individual Cloud flow', 'Run a Child Flow - Generate Formatted Word Document') are configured with a fixed retry policy of 3 attempts with a 30-second interval.  
 • No explicit error handling actions (e.g., Try/Catch scopes) or checks for failed action status are present in this flow definition.

### Connection References

• A variable named 'Connection References' is initialized as an empty array but is not populated or used by any actions within this flow definition. The flow parses connection reference data from the input XML, and the child flows it calls likely utilize connection references.

### Business Exception

• A variable named 'Business Exceptions' is initialized as an empty array but is not populated or used by any actions within this flow definition. No explicit business exception handling logic is present.

### System Exception

• Retry policies are implemented on child flow calls to handle transient system errors.

## Automate Solution Doc Generation on File Upload

This automated cloud flow is designed to streamline the process of generating documentation for Power Platform solutions. It is triggered whenever a new file (specifically intended for solution ZIP files) is uploaded to a designated folder within a SharePoint document library.  
  
The primary purpose of this flow is to automate the initial steps required before documentation can be generated. It identifies the uploaded solution file, creates a dedicated folder for its documentation, and then initiates a separate child flow responsible for the actual documentation generation process. Finally, it sends an email notification to the user who uploaded the file, informing them of the outcome (success or failure) and providing a link to the documentation folder.   
 Link to flow diagram - https://autonomateio.sharepoint.com/sites/FemiSandbox/Shared%20Documents/Solutions%20generated%20document%20folder/TechnicalDocumentAutomation\_1\_0\_0\_5/Automate%20Solution%20Doc%20Generation%20on%20File%20Upload.svg

### Trigger Overview

Automated: Triggered when a file is created (properties only) in a specific SharePoint folder. It monitors the folder specified by the 'Solution Upload Folder Path' parameter within the document library specified by the 'TDD Document Library' parameter on the SharePoint site specified by the 'TDD Solution Site URL' parameter. The trigger checks for new files every 1 minute.

### External Infrastructure Used

• SharePoint Online  
 • Office 365 Outlook  
 • Process Flows in Solution Zip (Child Flow)

### External flow Dependencies

• SharePoint Online  
 • Office 365 Outlook  
 • Process Flows in Solution Zip (Child Flow)

### Input Variables

• TDD Solution Site URL (fa\_TDDSolutionSite)  
 • TDD Document Library (fa\_TDDSolutionDocumentfolder)  
 • Solution Upload Folder Path (fa\_SolutionFolderpath)  
 • Documentation Output Folder Name (fa\_Generateddocumentfolderpath)

### Output Variables

• Compose\_-\_Uploaded\_Solution\_File\_Name  
 • Compose\_-\_Documentation\_Folder\_Path  
 • Compose\_-\_Uploaded\_Solution\_File\_ID  
 • Create\_new\_folder\_-\_Documentation\_Folder  
 • Compose\_-\_Document\_Destination\_Folder\_path  
 • Compose\_-\_Successful\_Documentation\_Mail\_Subject  
 • Compose\_-\_Successful\_Documentation\_Mail\_Body  
 • Compose\_-\_Failed\_Documentation\_Mail\_Subject  
 • Compose\_-\_Failed\_Documentation\_Mail\_Body

### Core Logic

This is an automated flow that starts when a new file is created in a configured SharePoint folder.  
  
Upon trigger, the flow first captures key details about the newly uploaded file using Compose actions: the file name ('Compose\_-\_Uploaded\_Solution\_File\_Name'), the file identifier ('Compose\_-\_Uploaded\_Solution\_File\_ID'), and constructs the intended path for the documentation folder by combining the configured output folder name and the uploaded file name ('Compose\_-\_Documentation\_Folder\_Path').  
  
Next, it uses the 'Create new folder - Documentation Folder' action to create a new folder in the specified SharePoint document library using the path composed in the previous step. This folder will house the generated documentation for the uploaded solution.  
  
After successfully creating the folder, it composes the full destination folder path ('Compose\_-\_Document\_Destination\_Folder\_path') using the output from the folder creation action. This full path is needed by the child flow.  
  
The flow then calls a child flow named 'Process Flows in Solution Zip'. It passes the uploaded file ID, the composed documentation destination folder path, and the uploaded file name to the child flow. This child flow is responsible for extracting the solution, processing its components (like flows), and generating the actual documentation files within the newly created folder. The child flow call is configured with a retry policy of 3 retries with a 30-second interval.  
  
Following the execution of the child flow, the parent flow prepares an email notification. It uses Compose actions to create the subject and body for both a successful outcome ('Compose\_-\_Successful\_Documentation\_Mail\_Subject', 'Compose\_-\_Successful\_Documentation\_Mail\_Body') and a failed outcome ('Compose\_-\_Failed\_Documentation\_Mail\_Subject', 'Compose\_-\_Failed\_Documentation\_Mail\_Body'). These compose actions are configured to run conditionally based on whether the child flow succeeded or failed/timed out.  
  
Finally, the 'Send an email (V2)' action sends an email to the user who uploaded the file. The subject and body of the email dynamically include the content from the success or failure compose actions, effectively reporting the result of the documentation process and providing a link to the documentation folder.

### Error Handling

• Child flow execution includes a retry policy (3 retries with 30-second interval).  
 • Email notification is sent regardless of child flow success or failure.  
 • Email subject and body content are conditional based on child flow outcome (success vs. failed/timed out).

### Connection References

• shared\_sharepointonline  
 • shared\_office365

### Business Exception

### System Exception

# Individual desktops flows

# Recommendation

Several potential improvements could enhance the robustness, efficiency, and usability of this documentation generation system.  
  
Firstly, the reliance on external APIs for AI content generation and Graphviz diagram rendering introduces external dependencies. Implementing fallback mechanisms or exploring alternative, potentially self-hosted, solutions could improve reliability and reduce dependency on external service availability and performance.  
  
Secondly, the current process documents all flows within a solution upon upload. For larger solutions or during iterative development, it might be more efficient to allow users to select specific flows for documentation or to implement logic that only updates documentation for flows that have changed since the last generation. This would reduce processing time and resource consumption.  
  
Thirdly, the AI prompts contain specific formatting instructions and negative constraints (e.g., 'Do not use backslashes'). While necessary for the current parsing logic, this suggests potential fragility. Enhancing the parsing of AI output, perhaps by using more structured output formats from the AI model or implementing more resilient text processing, could make the system less susceptible to unexpected AI responses.  
  
Fourthly, error handling could be more detailed. While the system sends a failure email, providing more specific information about which step failed (e.g., AI generation for a specific flow, Graphviz conversion) and the nature of the error would significantly aid troubleshooting.  
  
Fifthly, offering alternative output formats beyond a single Word document, such as Markdown, HTML, or direct integration with documentation platforms, could cater to different user needs and workflows.  
  
Finally, the handling of existing SVG files appears to check for existence but might benefit from clearer logic regarding overwriting, versioning, or skipping generation if a file already exists, especially in scenarios where the process might be re-run.

Generated: 25-Apr-2025