Journal of Computer and Communications, 2023, 11, 23-50
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

A360 Bot Framework: Empowering Smart Robotic Process Automation Solutions

Sai Madhur Potturu

Robotics Center of Excellence (CoE) Zoetis Inc., Parsippany, USA
Email: saimadhurpotturu@gmail.com

How to cite this paper: Potturu, S.M. (2023) A360 Bot Framework: Empowering Smart Robotic Process Automation Solutions. *Journal of Computer and Communications*, **11**, 23-50.
https://doi.org/10.4236/jcc.2023.1110003

Received: September 6, 2023
Accepted: October 20, 2023
Published: October 23, 2023

Abstract

This research paper explores the significance of the "A360 Bot Framework" in Automation 360 (A360) platform. A360 is Automation Anywhere's cloud-based automation platform designed to make business processes more efficient. It's known for its user-friendly interface, which allows both technical and non-technical users to use it effectively. Automation 360 is versatile, offering a range of tools to automate tasks, manage complex workflows, and integrate various applications. It empowers users to create customized solutions for their specific needs. Being cloud-based it ensures scalability, security, and real-time updates, making it a top choice in the fast-paced digital world. As demand for A360 rises, having a structured way to develop bots becomes crucial. The paper introduces the "A360 Bot Framework" as a guiding approach for bot developments. This framework ensures consistency and scalability, especially when working with both professional developers and non-technical users. It outlines key elements like setting up work folders, managing logs, dealing with errors, and ensuring secure bot execution. Ultimately, the "A360 Bot Framework" is presented as a foundational structure that enhances consistency, resiliency, and development efficiency. By following predefined practices and templates, bot developers can mitigate risks and streamline debugging processes. This framework accelerates the bot development lifecycle, allowing developers to focus on specific functionalities and value-added features. The research paper aims to provide insights into the benefits of adopting the A360 Bot Framework and its potential to revolutionize A360 bot development practices, leading to more efficient and effective automation solutions.

Keywords

Robotic Process Automation (RPA), Automation Anywhere, Automation 360, A360 Bot Framework, Reusability, Citizen Development, Scalability

1. Introduction

Automation Anywhere is a top player in the Robotic Process Automation (RPA) and Intelligent automation space. A360 is Automation Anywhere's cloud-based platform which makes building automation solutions easy and reliable. It offers seamless integrations with various tools and technologies to facilitate digital transformation [1] [2] [3].

A360 is a No-Code/Low-Code platform. With its intuitive visual interface, creating automation solutions becomes effortless through drag-and-drop actions. This interface harmoniously integrates with an extensive range of systems, tools, and technologies. Additionally, A360 provides a bot store, a collection of pre-designed actions and components catering to various functions across different systems. These can be seamlessly integrated into RPA bots without requiring manual coding [4].

The user-friendly interface, along with comprehensive documentation and training resources provided by Automation Anywhere, helps both developers and citizen developers [5] to quickly grasp the concepts and start building automation solutions.

A360 has high demand in the industry due to its ability to automate tasks, boost efficiency, cut costs, ensure accuracy, integrate with existing systems, and facilitate rapid scalability while driving digital transformation and enhancing competitive advantage. As the Adaption of the A360 continues to grow, the need for systematic and structured approaches to A360 bot development becomes paramount. The central idea is to create a standardized framework that guides developers in building bots. This framework ensures consistency, scalability, and success, especially when collaborating with citizen developers and establishing a thriving Center of Excellence (COE) for RPA [6].

In this research paper, we establish and examine the fundamental elements of the "A360 Bot Framework" and showcase its practical application. We emphasize key components including the creation of working folders, log management, downloading bot-dependent files, reading configuration files, error handling, standardized communication, and establishing a secure working environment for bot executions.

The "A360 Bot Framework" provides a foundational structure that ensures consistency in bot design, and development. By adhering to predefined templates and practices, bot developers can mitigate risks, enhance resiliency, and streamline the process of debugging errors.

Furthermore, the A360 Bot Framework approach significantly accelerates the development lifecycle by providing developers with a well-defined starting point. Developers can focus on crafting functionalities specific to their use cases, without the need to reinvent common practices for each bot. This streamlined approach not only saves time and resources but also empowers developers to innovate and create value-added features.

In conclusion, the implementation of A360 Bot Framework presents a paradigm shift in A360 bot development, emphasizing the significance of consistency, resilience, design patterns, and development acceleration. This paper aims to offer insights into the benefits of adopting A360 Bot Framework and their potential to revolutionize A360 bot development practices, ultimately contributing to the realization of more efficient and effective automation solutions [7].

2. Solution

The solution elaborates on various tasks/components, including close/kill applications, creating a context ID, setting up a working folder, creating a log file and defining log format, downloading files required for the bot's functionality, reading configuration files, utilizing the Orchestrator task to create a template for developing primary use cases, and leveraging Send Notifications for efficient communication. We provide detailed explanations of each aspect and elucidate the procedure for saving this code as reusable templates. All the components and templates are developed using the Task Bot editor, which provides hundreds of commands and drag-and-drop actions to create automated processes.

2.1. Close/Kill Applications

Closing application sessions before deploying an RPA bot is crucial for optimal performance and smooth operations. By closing unnecessary applications, resource efficiency is enhanced, providing the bot with sufficient CPU, memory, and disk resources to execute tasks without hindrance. This practice fosters a predictable environment, shielding the bot from unexpected behavior caused by conflicting applications. Additionally, it maintains consistency by eliminating external factors that might introduce variability into the automation process. This controlled setup also simplifies identifying and fixing problems, making the process of debugging, and troubleshooting easier and leading to a seamless RPA deployment and operation.

The "Close Application Sessions" can be created as a Task Bot in the "Utility" folder (**Figure 1**). A Task Bot is an interface/workspace where a developer can access pre-defined and custom packages to develop a software robot [8].

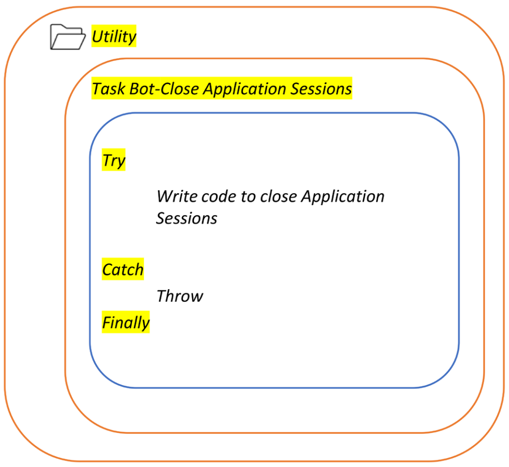
Utility
Task Bot-Close Application Sessions
Try
Write code to close Application Sessions
Catch
Throw
Finally

Figure 1. Close application sessions task syntax.

2.2. Generate Context ID

A context ID is a unique code generated by the bot for each bot execution. This is a specific value assigned to individual processes, transactions, or events in an automation process. This identifier serves to track and distinguish different instances of these processes or events. It is commonly used in process execution logs to provide a means of identifying and organizing log entries related to various activities within a system.

A context ID offers valuable benefits in bot execution logs. It ensures traceability by enabling the clear tracking of processes and events in a system, aiding in understanding their sequence. This ID facilitates efficient troubleshooting and debugging, allowing swift identification of issues and their locations. Moreover, it enables correlation of logs from different bot components, granting a comprehensive view of activities. In terms of compliance, it supports auditing efforts by creating an accountable record of actions. Additionally, these IDs aid performance analysis, assisting in identifying bottlenecks and areas for improvement. They enhance bot reliability in distributed environments, offering a consistent way to follow activities across different components. In essence, unique/context IDs enhance the clarity, organization, and manageability of the bot execution logs, playing a vital role in maintaining bot functionality and diagnosing problems effectively.

The "Generate Context ID" can be created as a task bot in the "Framework" folder (**Figure 2**). This task generates a unique, random string that can be used throughout the bot's execution.

2.3. Create Working Folder

The A360 bots are deployed on a machine for execution [9]. Attended bots are deployed on a local user's machine, and unattended bots are deployed on an unattended bot runner machine [10]. In both cases, it is important to create a folder with a time stamp for storing bot execution information.

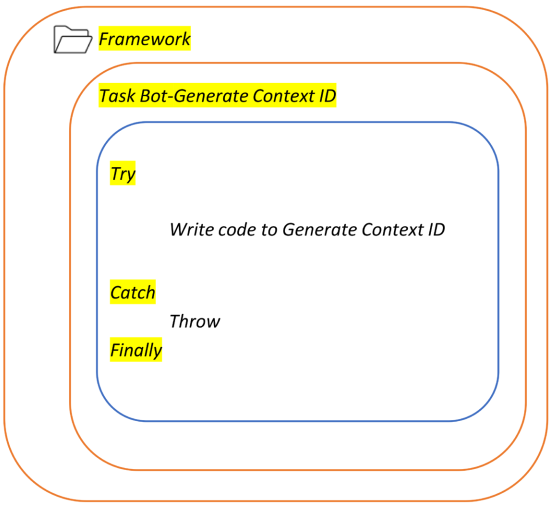
Framework
Task Bot-Generate Context ID
Try
Write code to Generate Context ID
Catch
Throw
Finally

Figure 2. Generate context ID task syntax.

This component creates a working folder with a time stamp for each bot execution in the user's directory, which is frequently backed up. Generally, users' Documents folders are backed up, or SharePoint/network drives can be used as root directories to create working folders for saving all information related to that specific bot execution. Creating working folders in backed-up directories reduces the risk of data loss in case of a user machine crash. The bot stores all process-related data, including configuration files, bot-dependent files, imports, exports from third-party applications, consolidated reports, and more in these folders.

These working folders can be referenced in exception emails sent to the support team. This helps the support team identify the working folder for a specific bot execution and troubleshoot errors. Additionally, it maintains transparency in bot activities for audit purposes. It is advisable to create repositories for each bot in a SharePoint document library to store bot-dependent files like configuration files, models, templates, and other files. This approach streamlines file management and facilitates quicker changes to files when needed. These repositories can also serve as storage for bot outputs (Figure 3).

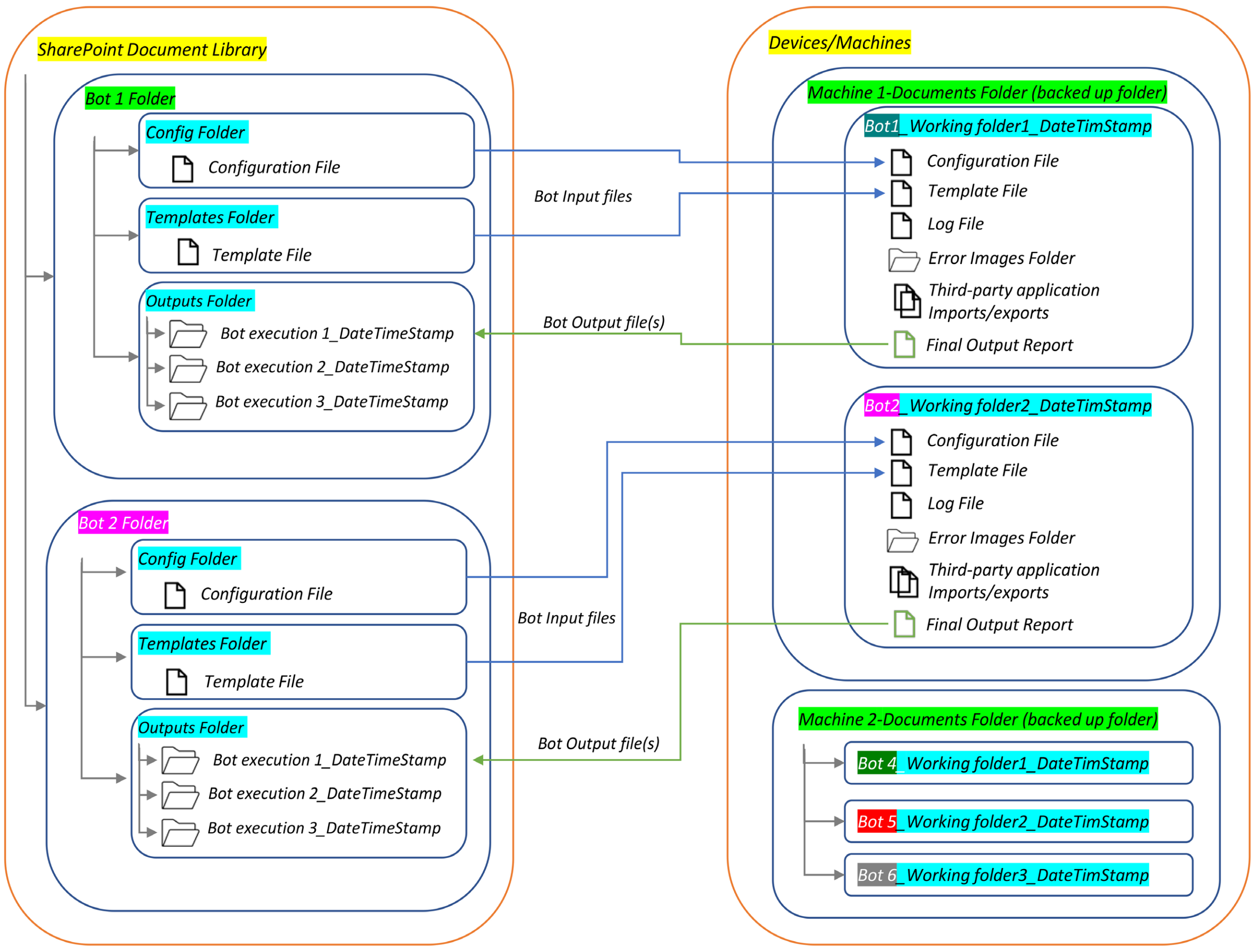
SharePoint Document Library
Bot 1 Folder
Config Folder
Configuration File
Templates Folder
Template File
Outputs Folder
Bot execution 1_DateTimeStamp
Bot execution 2_DateTimeStamp
Bot execution 3_DateTimeStamp
Bot 2 Folder
Config Folder
Configuration File
Templates Folder
Template File
Outputs Folder
Bot execution 1_DateTimeStamp
Bot execution 2_DateTimeStamp
Bot execution 3_DateTimeStamp
Bot Input files
Bot Output file(s)
Bot Input files
Bot Output file(s)
Devices/Machines
Machine 1-Documents Folder (backed up folder)
Bot1_Working folder1_DateTimStamp
Configuration File
Template File
Log File
Error Images Folder
Third-party application Imports/exports
Final Output Report
Bot2_Working folder2_DateTimStamp
Configuration File
Template File
Log File
Error Images Folder
Third-party application Imports/exports
Final Output Report
Machine 2-Documents Folder (backed up folder)
Bot 4_Working folder1_DateTimStamp
Bot 5_Working folder2_DateTimStamp
Bot 6_Working folder3_DateTimStamp

Figure 3. Data flow between Sharepoint Reporistory and working folder.

A global value/variable [11] can be created to save the Bot Dependencies repository path. This global value/variable can be accessed by any bot within the control room (**Figure 4**). Each environment, Development, QA/UAT, and Production will have this global variable. However, the variable's value corresponds to the respective bot dependencies repository of each environment.

The "Create Working Folder" can be created as a task bot in the "Framework" folder. This task establishes mappings to process-specific directories within the bot dependencies repository by using the "Bot dependencies repository" global value.

This task creates a working folder on the user's machine within the Documents directory, using the current date and time (**Figure 5**).

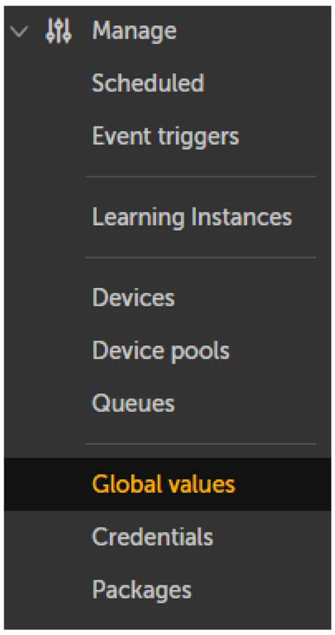

Figure 4. Global values in control room.

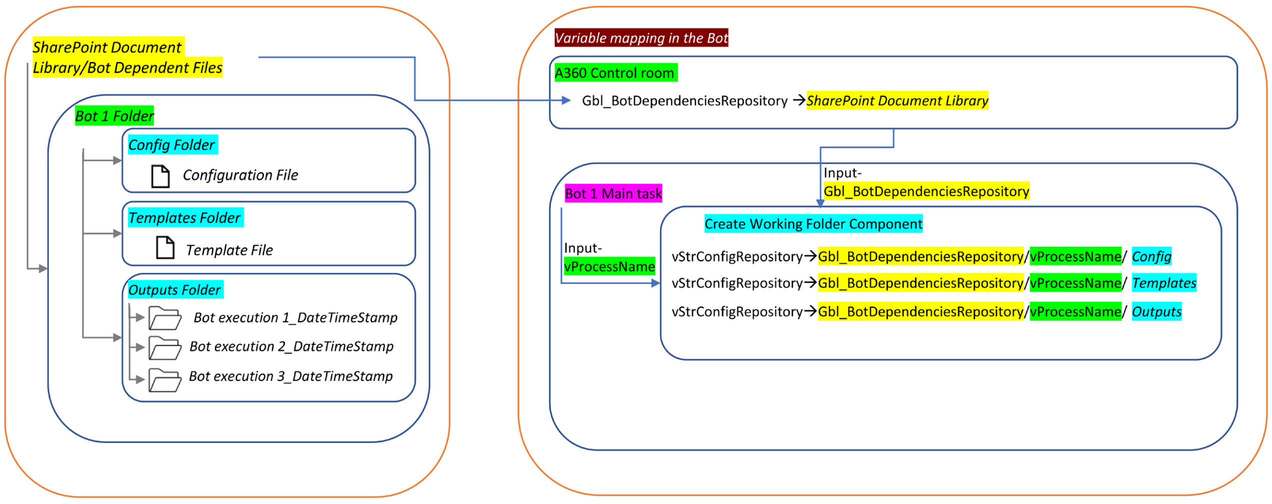
SharePoint Document Library/Bot Dependent Files
Bot 1 Folder
Config Folder
Configuration File
Templates Folder
Template File
Outputs Folder
Bot execution 1_DateTimeStamp
Bot execution 2_DateTimeStamp
Bot execution 3_DateTimeStamp
Variable mapping in the Bot
A360 Control room
Gbl_BotDependenciesRepository →SharePoint Document Library
Input-
Gbl_BotDependenciesRepository
Bot 1 Main task
Input-
vProcessName
Create Working Folder Component
vStrConfigRepository→Gbl_BotDependenciesRepository/vProcessName/ Config
vStrConfigRepository→Gbl_BotDependenciesRepository/vProcessName/ Templates
vStrConfigRepository→Gbl_BotDependenciesRepository/vProcessName/ Outputs

Figure 5. Mapping process specific directories.

This task creates a folder within the Output directory of the process-specific folder in the bot's dependencies repository, appending a time stamp (**Figure 6**). This folder is utilized for storing the final bot report generated during that execution.

2.4. Create Log File, Log Format and Error Images Folder

Process execution log files are of paramount importance for multiple reasons. They serve as invaluable tools for debugging and troubleshooting, offering a comprehensive record of a process's actions during execution to identify errors and irregularities. Furthermore, these log files enable real-time monitoring and auditing, ensuring processes function as expected and maintaining a historical record of executions for compliance purposes. Performance analysis benefits from log files by tracking execution times and resource usage, aiding in process optimization. These logs also serve as documentation, facilitating communication among developers, and contributing to continuous improvement efforts by identifying recurring issues. Overall, process execution log files are essential components in maintaining reliability, performance, and security across automation solutions.

The "Create Log File and Error Folder" can be created as a task bot in the "Framework" folder (**Figure 7**).

This task creates a text log file in the bot's working folder, incorporating the process name and timestamp in the file name.

For example: //Working folder path/Process Name/execution log_mm.dd.yyyy_hh.mm.ss.txt.

This task creates a log format including the Context ID, Machine Name, and Process Name. This log format is utilized in the log file alongside the log message [12].

This workflow creates Error images folder in the bot's working folder. Screenshots of errors or exceptions captured during the bot's execution are stored in this designated folder.

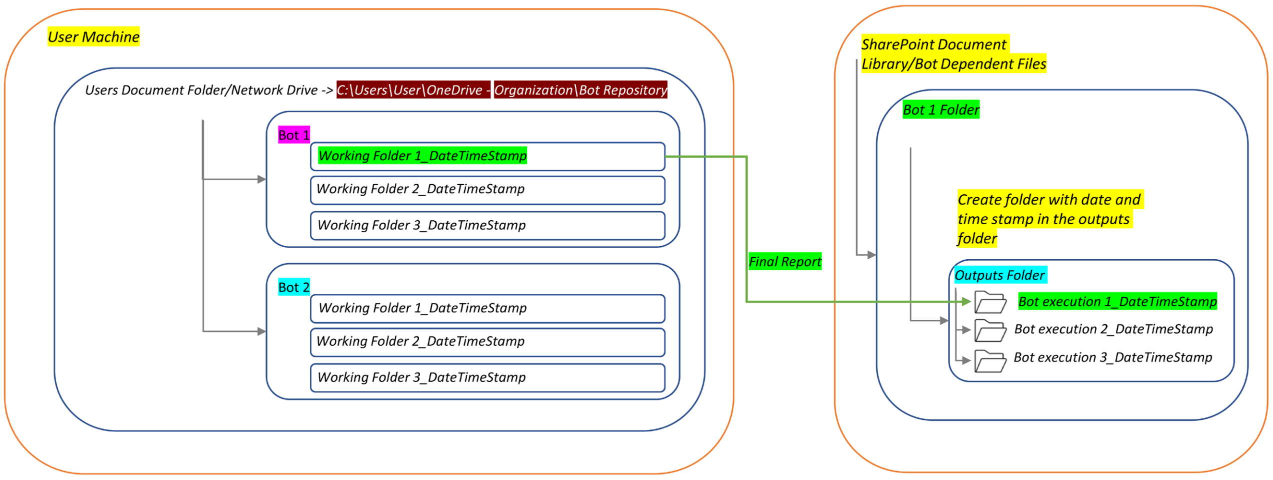
User Machine
Users Document Folder/Network Drive -> C:\Users\User\OneDrive - Organization\Bot Repository
Bot 1
Working Folder 1_DateTimeStamp
Working Folder 2_DateTimeStamp
Working Folder 3_DateTimeStamp
Bot 2
Working Folder 1_DateTimeStamp
Working Folder 2_DateTimeStamp
Working Folder 3_DateTimeStamp
Final Report
SharePoint Document Library/Bot Dependent Files
Bot 1 Folder
Create folder with date and time stamp in the outputs folder
Outputs Folder
Bot execution 1_DateTimeStamp
Bot execution 2_DateTimeStamp
Bot execution 3_DateTimeStamp

Figure 6. Date and time stamp folder in process specific output directory.

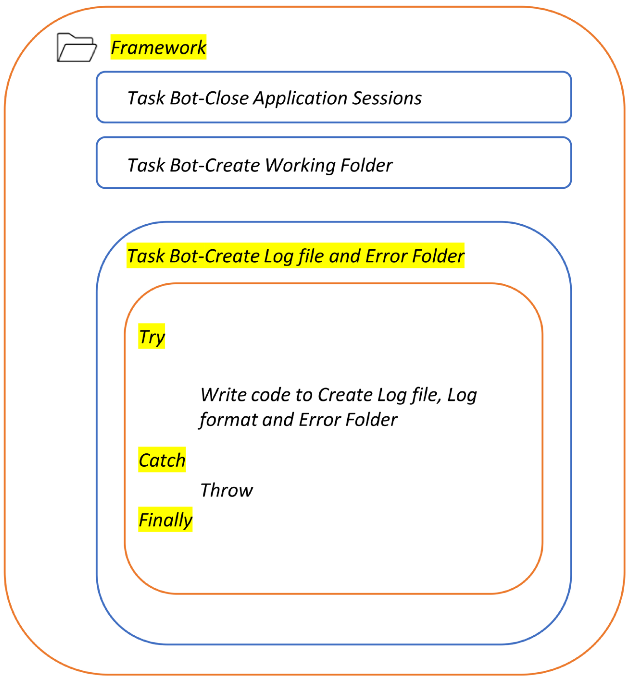
Framework
Task Bot-Close Application Sessions
Task Bot-Create Working Folder
Task Bot-Create Log file and Error Folder
Try
Write code to Create Log file, Log format and Error Folder
Catch
Throw
Finally

Figure 7. Create log file and error folder task syntax.

2.5. Copy Bot Dependent Files

The "Copy Bot Dependent Files" can be created as a task bot in the "Framework" folder (**Figure 8**).

This task copies the bot-dependent files from SharePoint repository to working folder. It receives folder paths created in the "Create Working Folder" task as Inputs.

2.6. Read Configuration File

The "Read Configuration File" can be created as a task bot in the "Framework" folder (**Figure 9**).

This task read the values from the configuration file into an output Dictionary. This dictionary can then be passed as input to various subtasks within the bot, eliminating the need to create redundant variables for each task.

2.7. Framework Integration

The "Framework Integration" can be created as a task bot in the "Framework" folder.

This task integrates [13] the "Generate Context ID", "Create Working Folder", "Create Log file and Error Folder", "Copy Bot Dependent Files" and "Read Configuration File" tasks from the Framework Folder (**Figure 10**).

This Framework Integration task, along with all the tasks integrated with it, are reusable components and do not require a frequent change. These tasks serve as common steps for setting up a working environment for each bot/automation process and are stored in the "Public" folder [14] [15] (**Figure 11**).

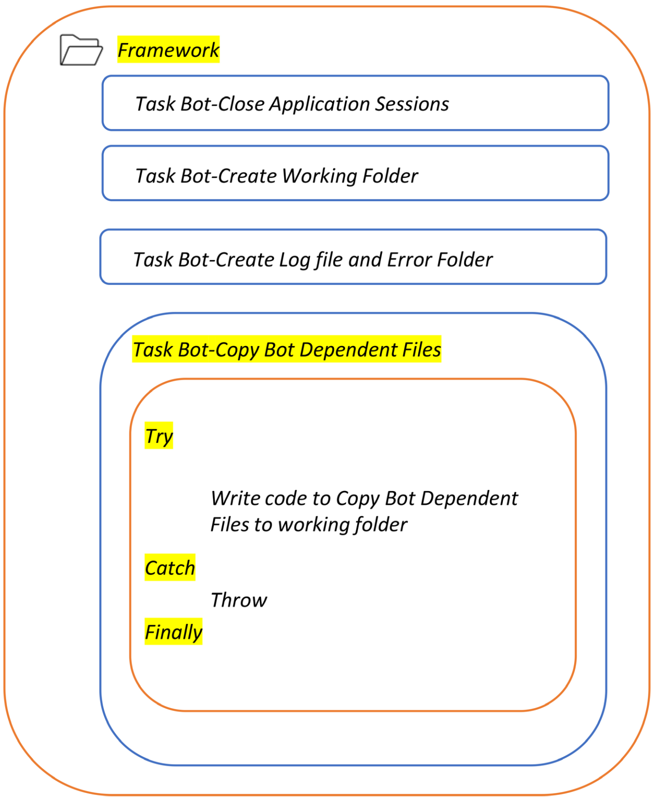
Framework
Task Bot-Close Application Sessions
Task Bot-Create Working Folder
Task Bot-Create Log file and Error Folder
Task Bot-Copy Bot Dependent Files
Try
Write code to Copy Bot Dependent Files to working folder
Catch
Throw
Finally

Figure 8. Copy bot dependent files task syntax.

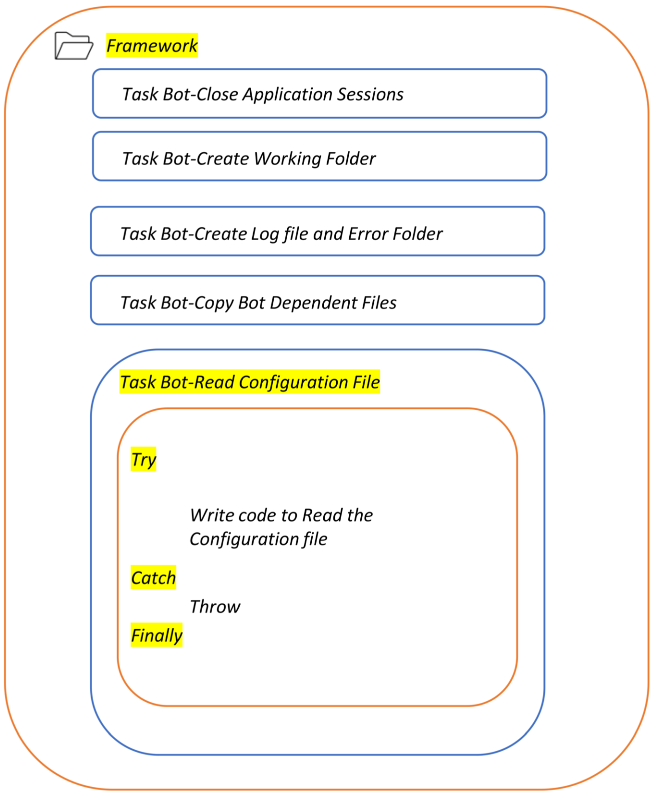
Framework
Task Bot-Close Application Sessions
Task Bot-Create Working Folder
Task Bot-Create Log file and Error Folder
Task Bot-Copy Bot Dependent Files
Task Bot-Read Configuration File
Try
Write code to Read the Configuration file
Catch
Throw
Finally

Figure 9. Read configuration file task syntax.

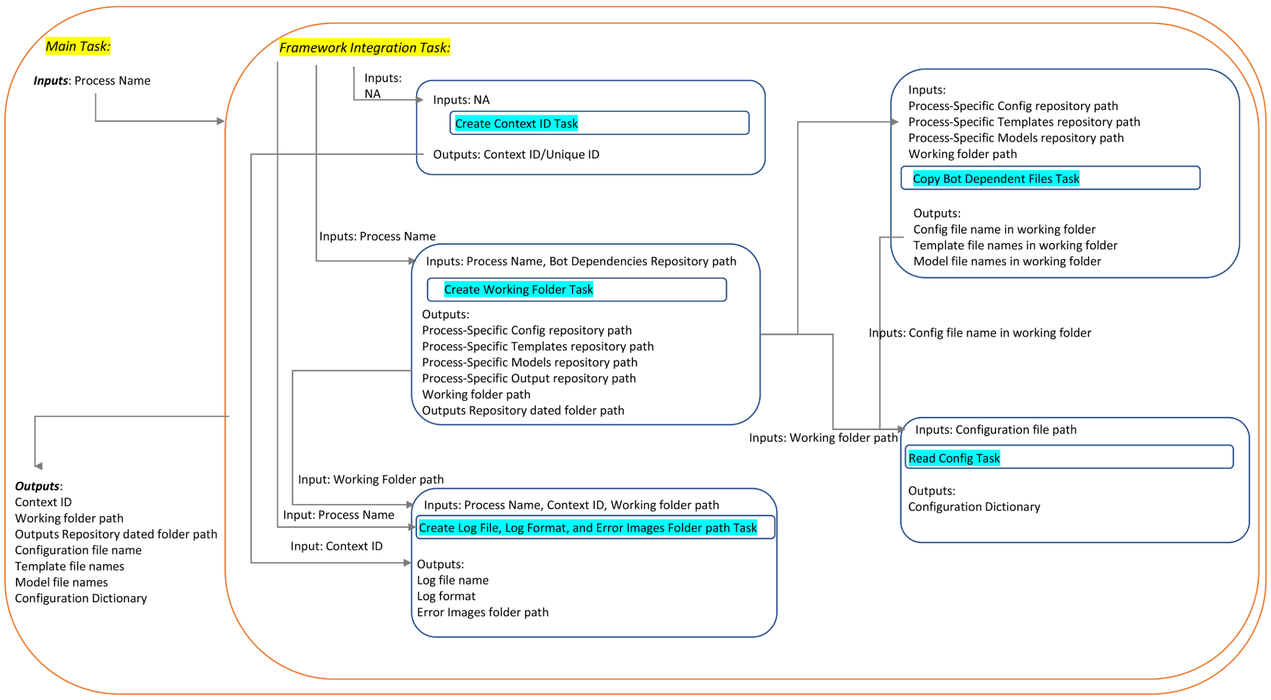
Main Task:
Inputs: Process Name
Outputs:
Context ID
Working folder path
Outputs Repository dated folder path
Configuration file name
Template file names
Model file names
Configuration Dictionary
Framework Integration Task:
Inputs: NA
Inputs: NA
Create Context ID Task
Outputs: Context ID/Unique ID
Inputs: Process Name
Inputs: Process Name, Bot Dependencies Repository path
Create Working Folder Task
Outputs:
Process-Specific Config repository path
Process-Specific Templates repository path
Process-Specific Models repository path
Process-Specific Output repository path
Working folder path
Outputs Repository dated folder path
Inputs:
Process-Specific Config repository path
Process-Specific Templates repository path
Process-Specific Models repository path
Working folder path
Copy Bot Dependent Files Task
Outputs:
Config file name in working folder
Template file names in working folder
Model file names in working folder
Inputs: Config file name in working folder
Inputs: Working folder path
Inputs: Configuration file path
Read Config Task
Outputs:
Configuration Dictionary
Input: Working Folder path
Input: Process Name
Input: Context ID
Inputs: Process Name, Context ID, Working folder path
Create Log File, Log Format, and Error Images Folder path Task
Outputs:
Log file name
Log format
Error Images folder path

Figure 10. Framework integration task design.

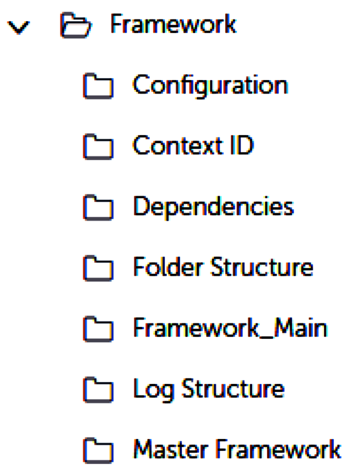

Figure 11. Framework folder.

2.8. Orchestrator Task Template

The "Orchestrator Task Template" can be created as a task bot in the "Bot Framework" folder within the public folder.

The Orchestrator task is a blank task with defined error handling. This task is equipped with all the necessary inputs to initiate the development of the actual business case. The input and output parameters required for bot development have already been created.

This task is saved as a template in the "Public" folder [16] or on GitHub [17]. It can be copied into a relevant bot subfolder, where the actual business case can be developed and integrated into the bot's Main task (Figure 12).

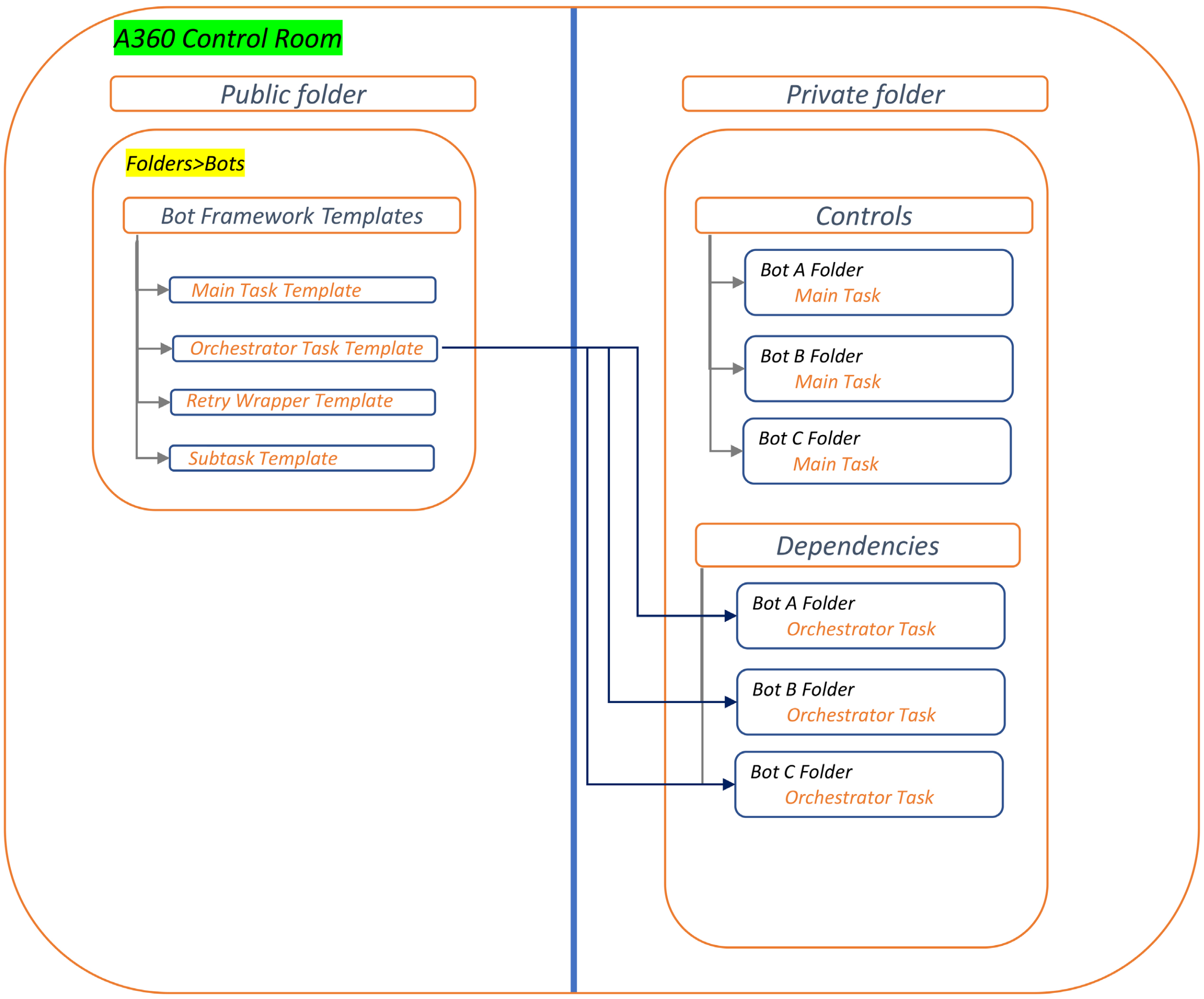

Figure 12. Copy orchestrator task template from public folder to bot specific priate folder.

A developer can create subtasks to perform various functions, integrating and orchestrating these subtasks within the orchestrator task as needed.

2.9. Subtask and Retry Framework

In Automation Anywhere (A360), a subtask is a smaller unit of work within an automation process or bot. It represents a specific action that contributes to achieving a task or objective, allowing for better organization and reusability. The subtasks are self-contained modules that can be easily reused across multiple bots or scenarios. By breaking down complex processes into manageable components, subtasks streamline bot development and improve automation efficiency.

There are different approaches for developing subtasks and retrying templates in A360. These methods can aid developers in creating high-quality and consistent code, while also assisting support teams in easily maintaining the code. Furthermore, these methods improve the efficiency and resiliency of the automation solution. Additionally, the retry framework can be leveraged to retry subtasks a defined number of times or until the desired output is generated in the event of system or unknown exceptions. This approach can enhance the bot's efficiency and reduce the manual support effort required to retrigger the bot [8].

The "Subtask" and "Retry" templates are saved in the "Bot Framework" folder within the "Public" folder. These templates are then copied to the bot-specific private folder for developing the subtasks/functionalities (**Figure 13**).

2.10. Main Task Template

The "Main Task Template" can be created as a task bot in the "Bot Framework" folder within the public folder.

The Main Task serves as the entry point for the automation process. It dictates the sequence in which subtasks are executed, handles data flow between different steps, manages error handling and exception scenarios, and interacts with external systems and applications as required.

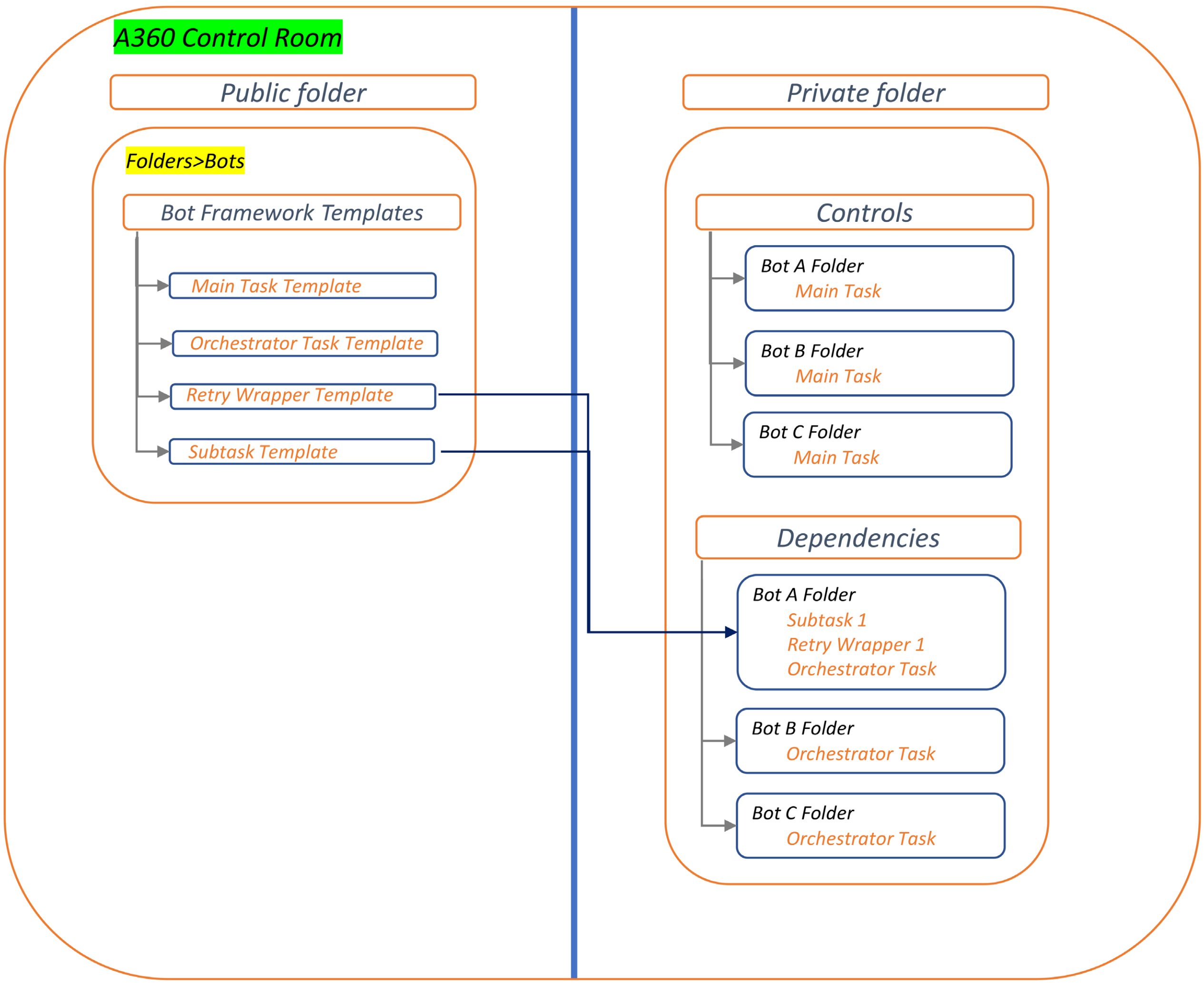

Figure 13. Copy retry and subtask templates from public folder to bot specific priate folder.

The Main task is saved as a template in the "Public" folder or on GitHub. It can be copied into a relevant bot Main Task folder, update the placeholders, and orchestrate the process flow as required (Figure 14).

The following components are integrated in the Main Task Template (Figure 15):

Process Name: Provide a place holder for the Process Name, this is crucial for identifying the bot-specific folders within the bot dependencies repositories and for sending out notifications.

Clean/Kill Application Sessions: Closing or killing application sessions before bot execution ensures a smooth and reliable environment for bot execution.

Framework Interaction Task: All framework components including "Generate Context ID", "Create working folder", "Create log file and error folder", "Copy bot dependent files", and "Read Configuration file" are integrated within the Framework Interaction task. This task sets up a safe and secure working environment for bot execution.

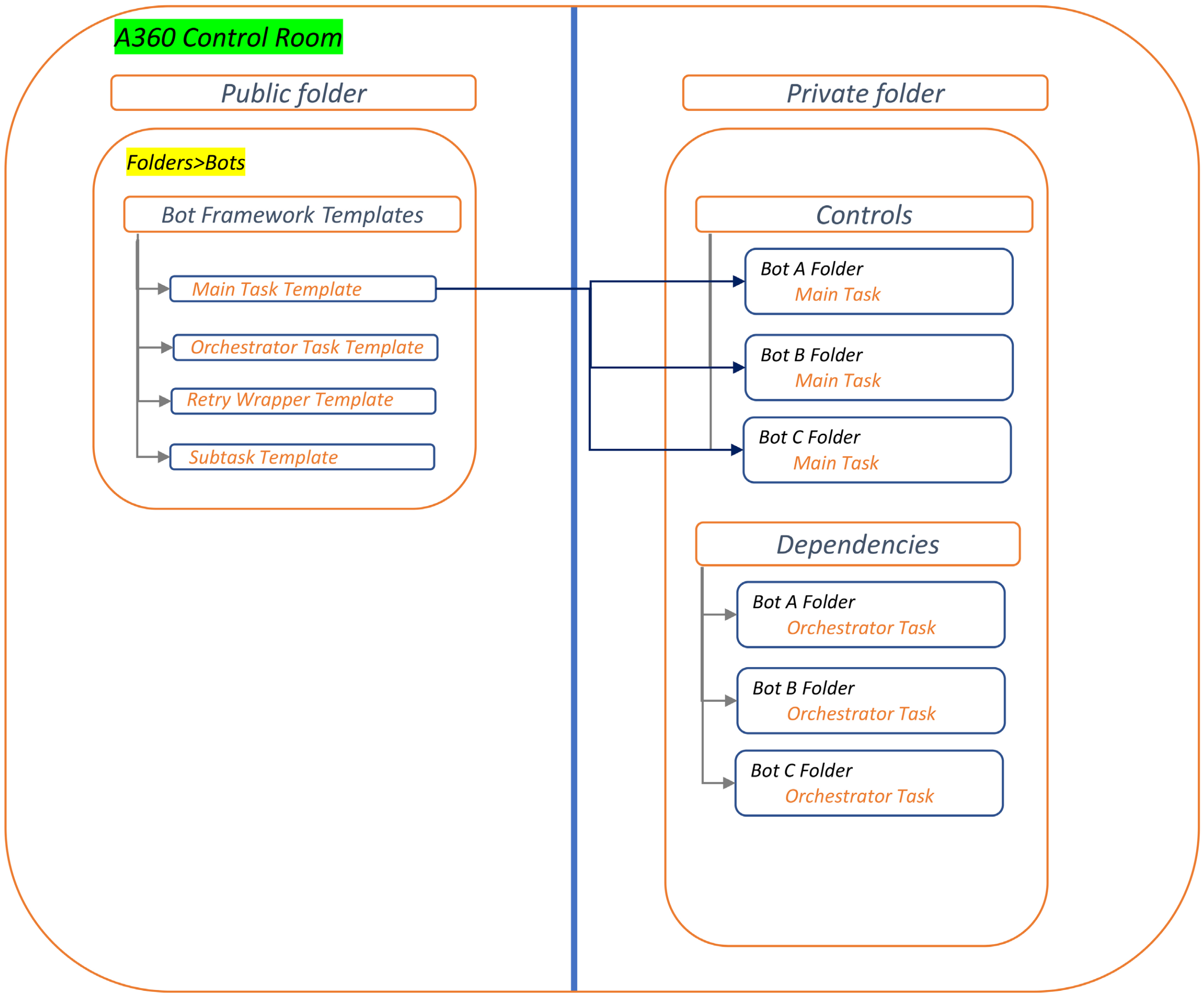

Figure 14. Copy main task template from public folder to bot specific priate folder.

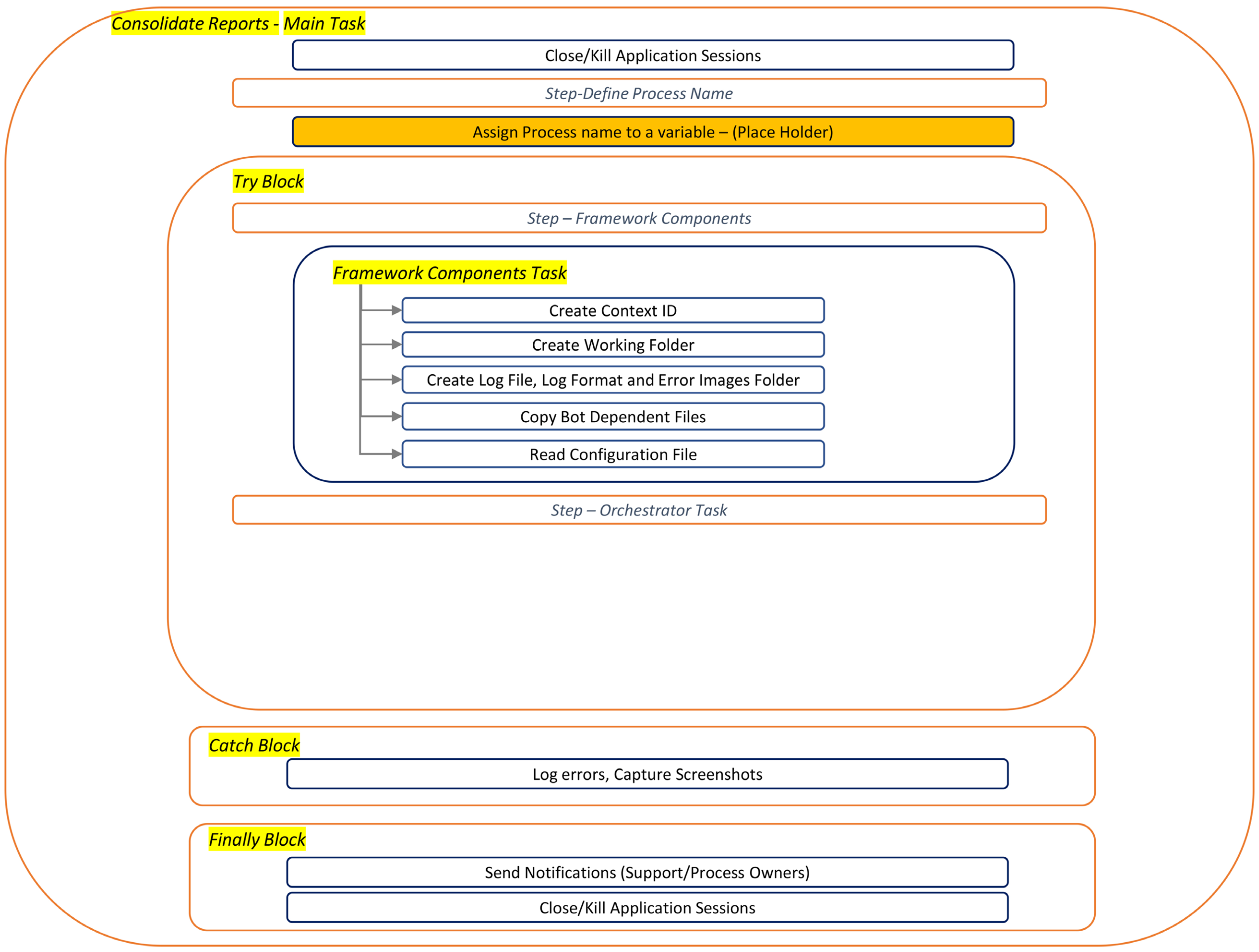
Consolidate Reports - Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – (Place Holder)
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 15. Main task template design.

Send Notifications Task: The Send Notifications task can be used to send notifications to either the support team or the process owners/stakeholders based on the execution status of the orchestrator task.

Clean/Kill Application Sessions: Closing application sessions after bot execution helps manage resources efficiently, ensures better performance for upcoming tasks, and maintains a clean, predictable environment. This practice lays the foundation for smooth and reliable automation workflows in subsequent bot runs.

3. Application Scenarios: Demonstrating the A360 Bot Framework in Action

This section presents two comprehensive case studies that showcase the practical implementation of the "A360 Bot Framework". These case studies illustrate how the framework serves as a foundational starting point to develop distinct automation use cases. By leveraging the framework's standardized components, streamlined practices, and reliable error management, developers can efficiently create automation solutions that address specific business challenges. Each case study highlights how the framework's structure and approach accelerate development while maintaining consistency and robustness.

3.1. Case Study 1: Data Entry

In this scenario, we'll explore how the "A360 Bot Framework" acts as a foundational starting point for developing a bot that automates the process of filling out a web form using data from an Excel spreadsheet. This use case is common in scenarios where manual data entry into web forms is time-consuming and error prone.

1) Prerequisites

Assuming the bot's name is "Data Entry," create a folder titled "Data Entry" in the Bot dependencies repository (SharePoint/network drive). Create directories such as Config, Templates, Models, and Outputs, and store bot-dependent files such as Configuration file, Template file, and Model file in the corresponding folders within the "Data Entry" folder (**Figure 16**).

2) Setting up the Main Task

Create a folder for the bot to save its primary/main task. Copy the "Main Task" template from the public folder to the bot folder [18]. Rename the Main Task to "Data Entry" (**Figure 17**).

Enter the bot's name "Data Entry" into the Process name placeholder within the main task (**Figure 18**). This action establishes the context for the bot's workflow, which involves downloading files from the designated bot name folder in the repository to the working folder and executing the process.

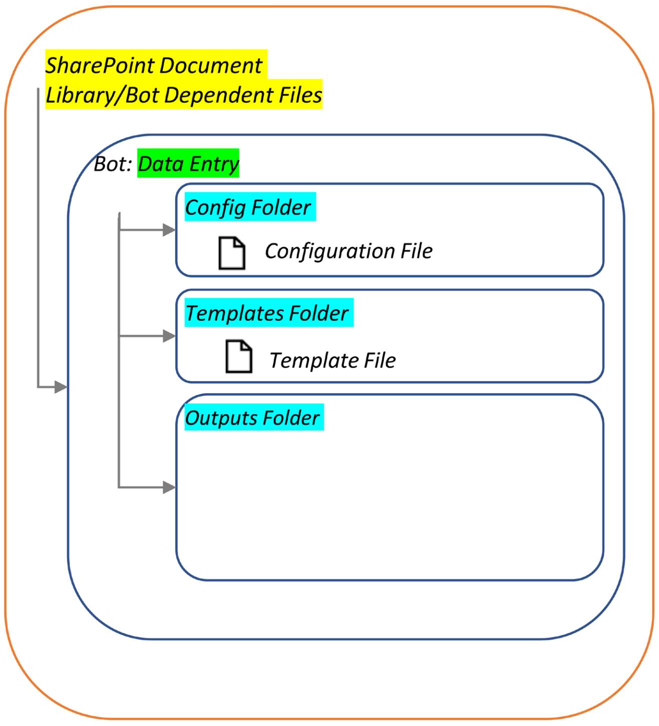

Figure 16. "Data entry" folder in bot dependencies repository.

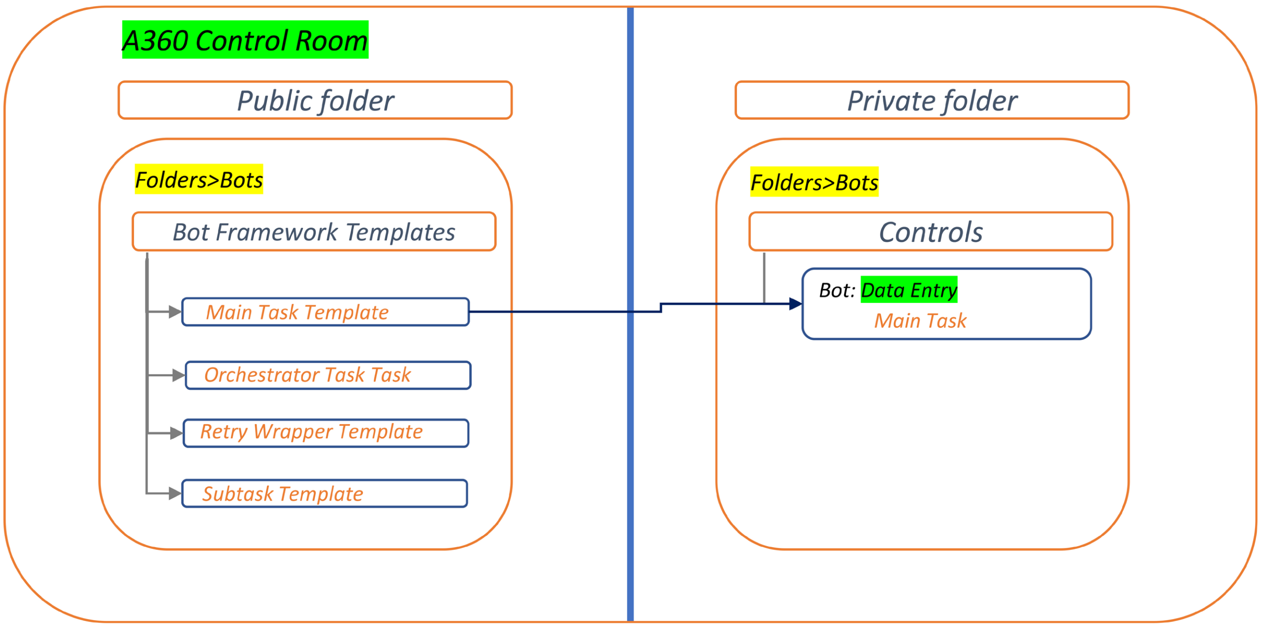

Figure 17. Copy main task template.

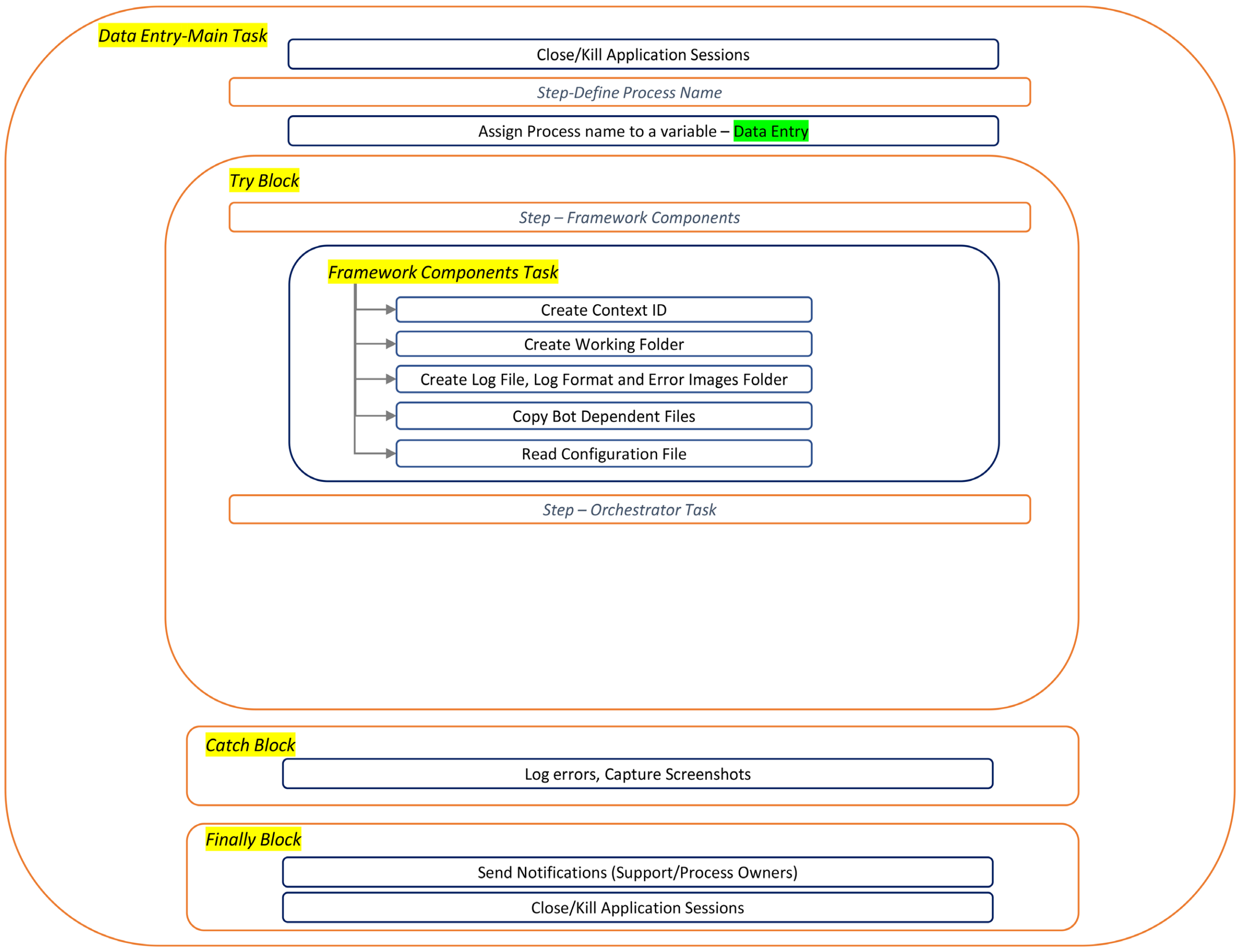
Data Entry-Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Data Entry
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 18. Assign "data entry" to process name place holder.

3) Framework Integration Task

The framework integration task is already integrated into the main task and operates in alignment with the context of the bot/process name. The Framework Integration task generates a context ID, creates a working folder in the bot's named folder in the user's documents directory, generates log files and log formats, establishes an error images folder, copies dependencies from the bot dependencies repository to the working folder, and reads the configuration file (Figure 19). The process name serves as the key input parameter for this task.

4) Orchestrator Task Setup

Create a folder for the bot to store its orchestrator task. Copy the "Orchestrator Task" template from the public folder to the bot folder (Figure 20). Invoke

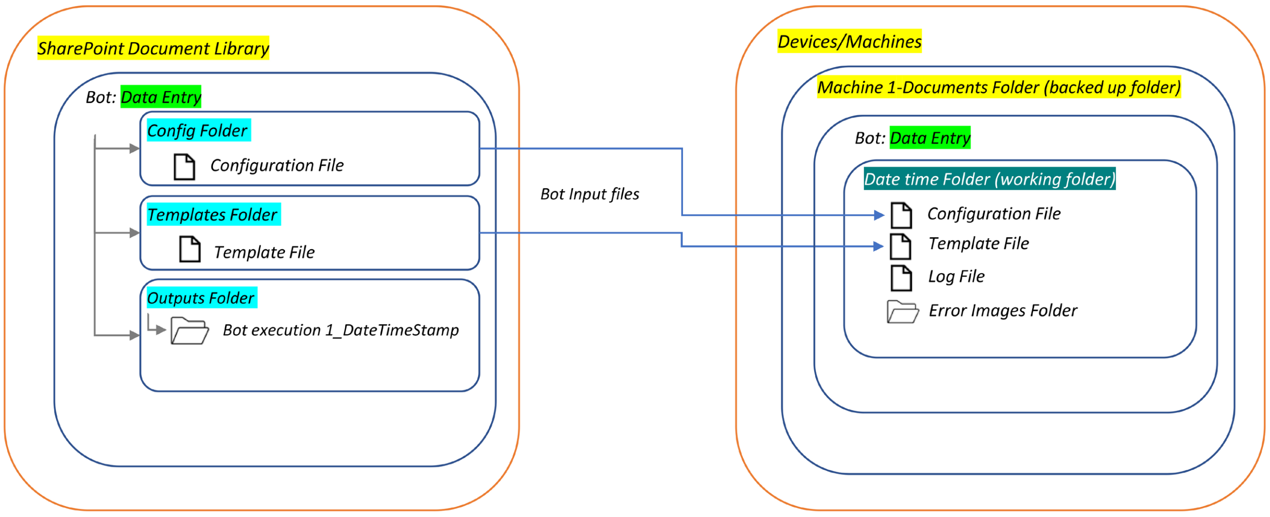

Figure 19. "Data entry" framework inregration task functionality.

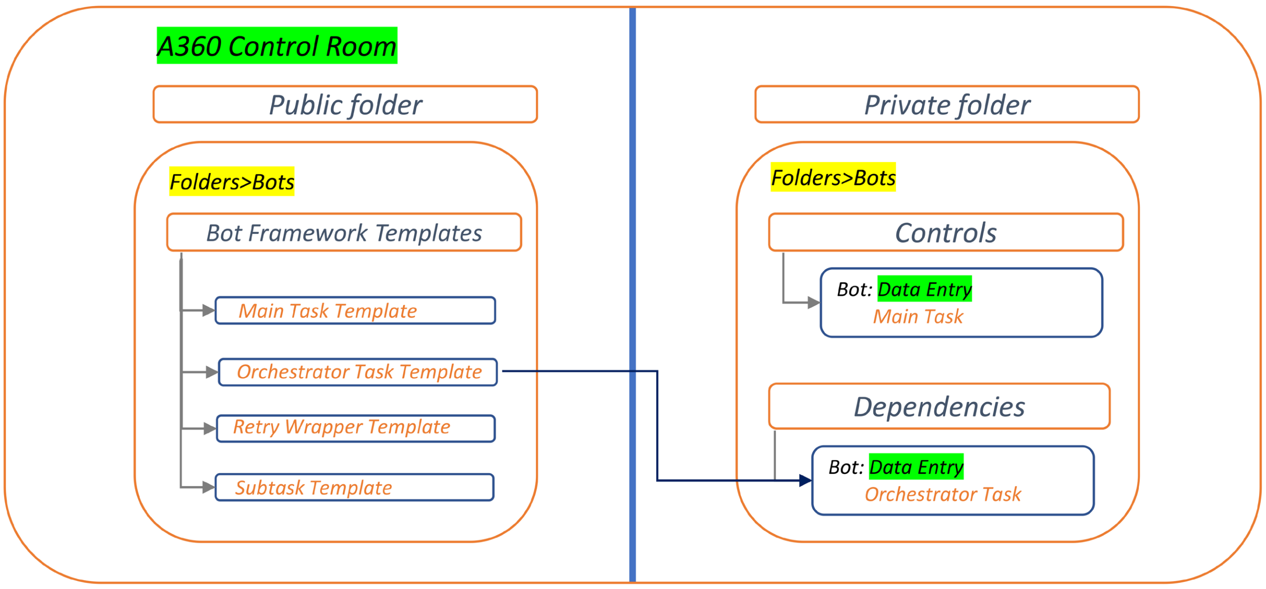

Figure 20. Copy orchestrator task template.

the orchestrator task from the bot's main task and establish mappings for the variables "Context ID", "Working folder path", "Outputs dated folder path", "Config file name", "Template file names", "Model file names" and "Configuration Dictionary" (Figure 21).

5) Subtask Task Setup

Create the subtasks "Read Excel file" and "Complete Webform". The "Complete Webform" subtask is invoked within the "Read Excel file" task, where the "Read Excel file" task extracts data from the Excel file and passes it to the "Complete Webform" task to complete the web form. Both subtasks are integrated within the orchestrator task (Figure 22). The pre-defined variables in the orchestrator, such as the configuration dictionary and working folder, are mapped to these subtasks.

In conclusion, throughout the bot development process, a developer only needs to invest time in creating the subtasks "Read Excel file" and "Complete Webform". The remaining templates are utilized as they are and integrated to construct a comprehensive end-to-end automation solution (Figure 23). This approach substantially reduces development time, enhances efficiency, minimizes errors, and elevates overall automation quality, consistency, and reliability.

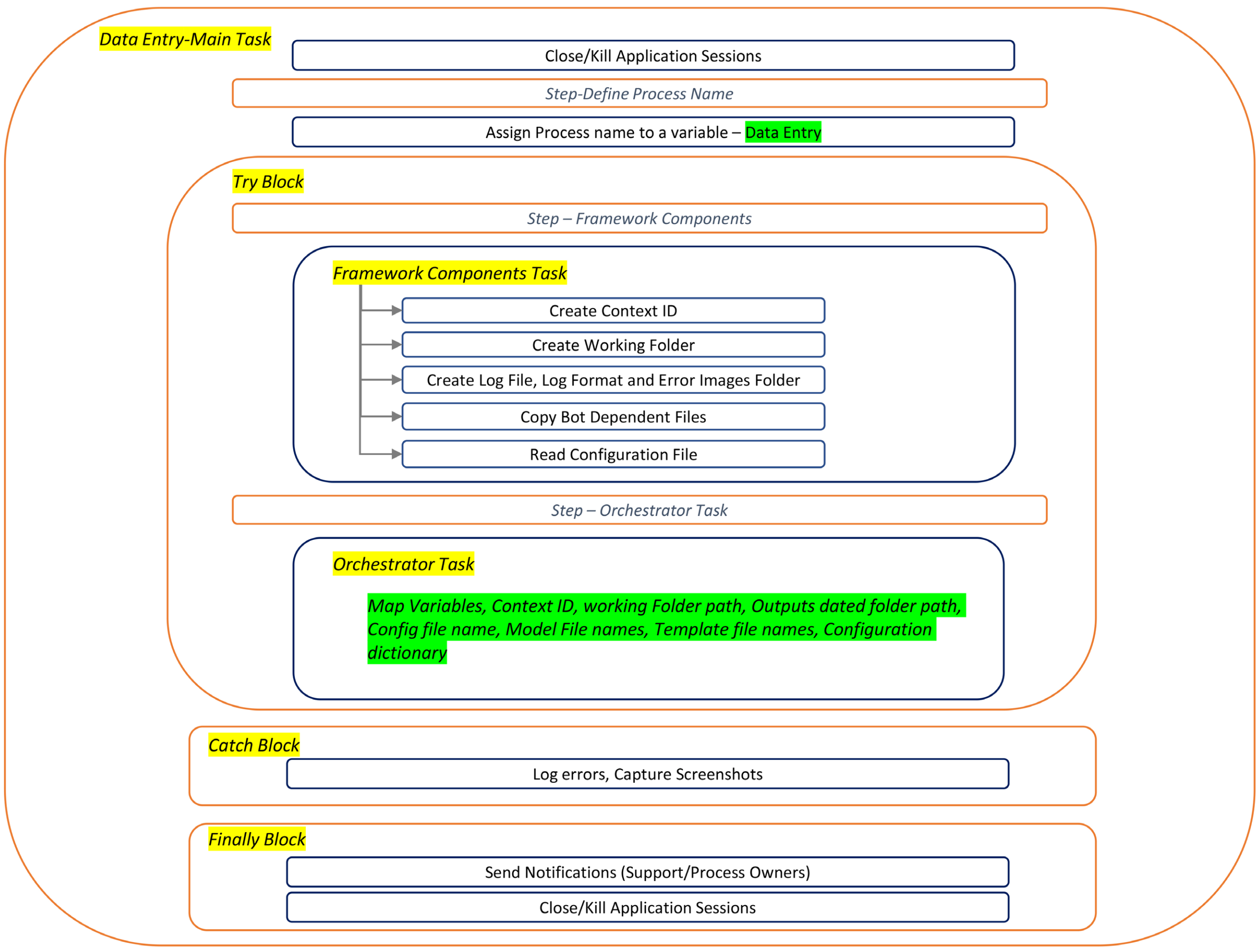
Data Entry-Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Data Entry
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Orchestrator Task
Map Variables, Context ID, working Folder path, Outputs dated folder path, Config file name, Model File names, Template file names, Configuration dictionary
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 21. Map variables.

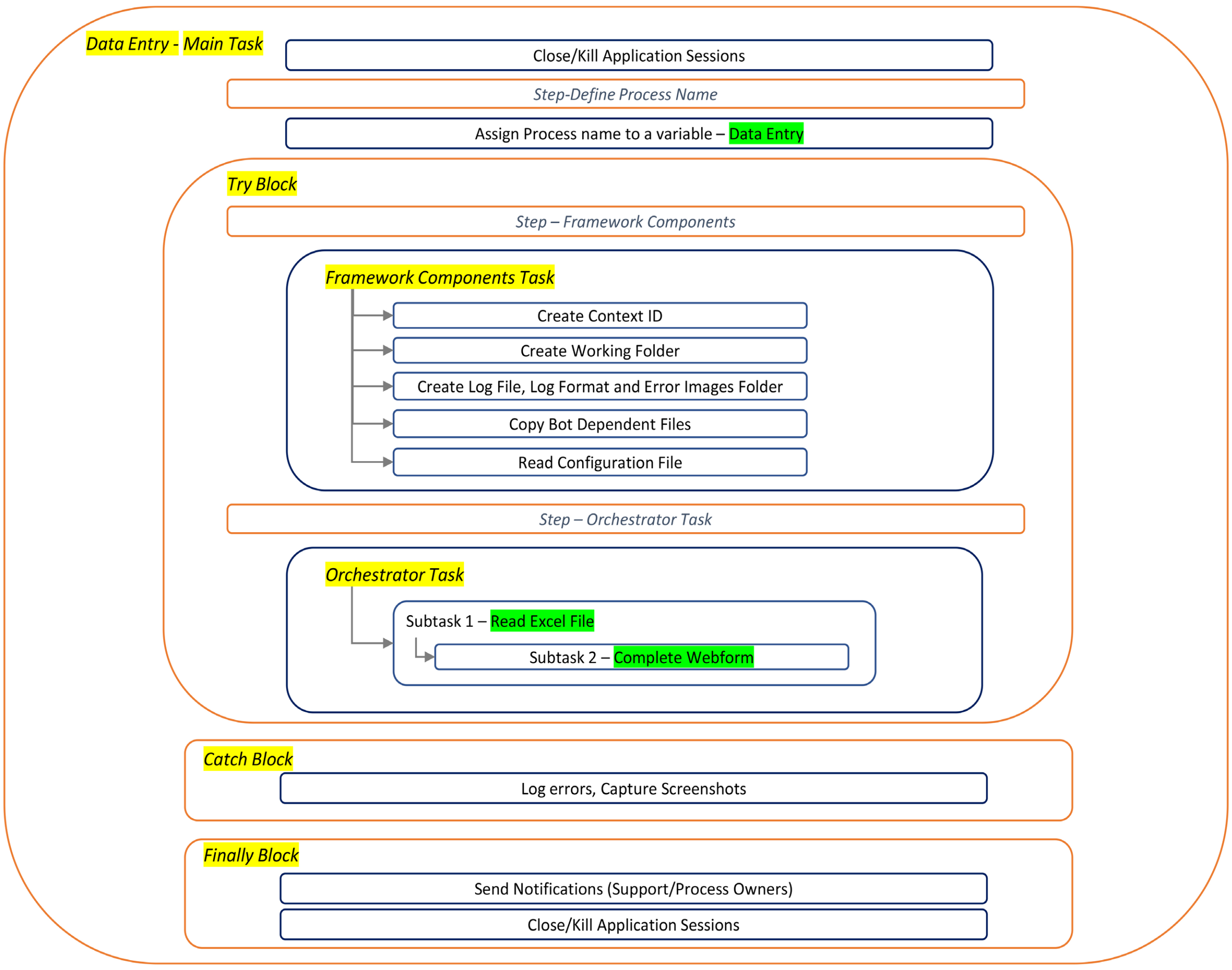
Data Entry - Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Data Entry
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Orchestrator Task
Subtask 1 – Read Excel File
Subtask 2 – Complete Webform
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 22. Create subtasks.

3.2. Case Study 2: Consolidate Reports

This case study demonstrates how the "A360 Bot Framework" serves as a starting point for developing a bot that downloads data from both a web application and a Windows application, followed by consolidating the acquired data into comprehensive reports.

1) Prerequisites

Assuming the bot's name is "Consolidate Reports", create a folder titled "Consolidate Reports" in the Bot dependencies repository (SharePoint/network drive). Create directories such as Config, Templates, Models, and Outputs, and store bot-dependent files such as Configuration file, Template file, and Model file in the corresponding folders within the "Consolidate Reports" folder (**Figure 24**).

2) Setting up the Main Task

Create a folder for the bot to save its primary/main task. Copy the "Main Task" template from the public folder to the bot folder (**Figure 25**). Rename the Main Task to "Consolidate Reports".

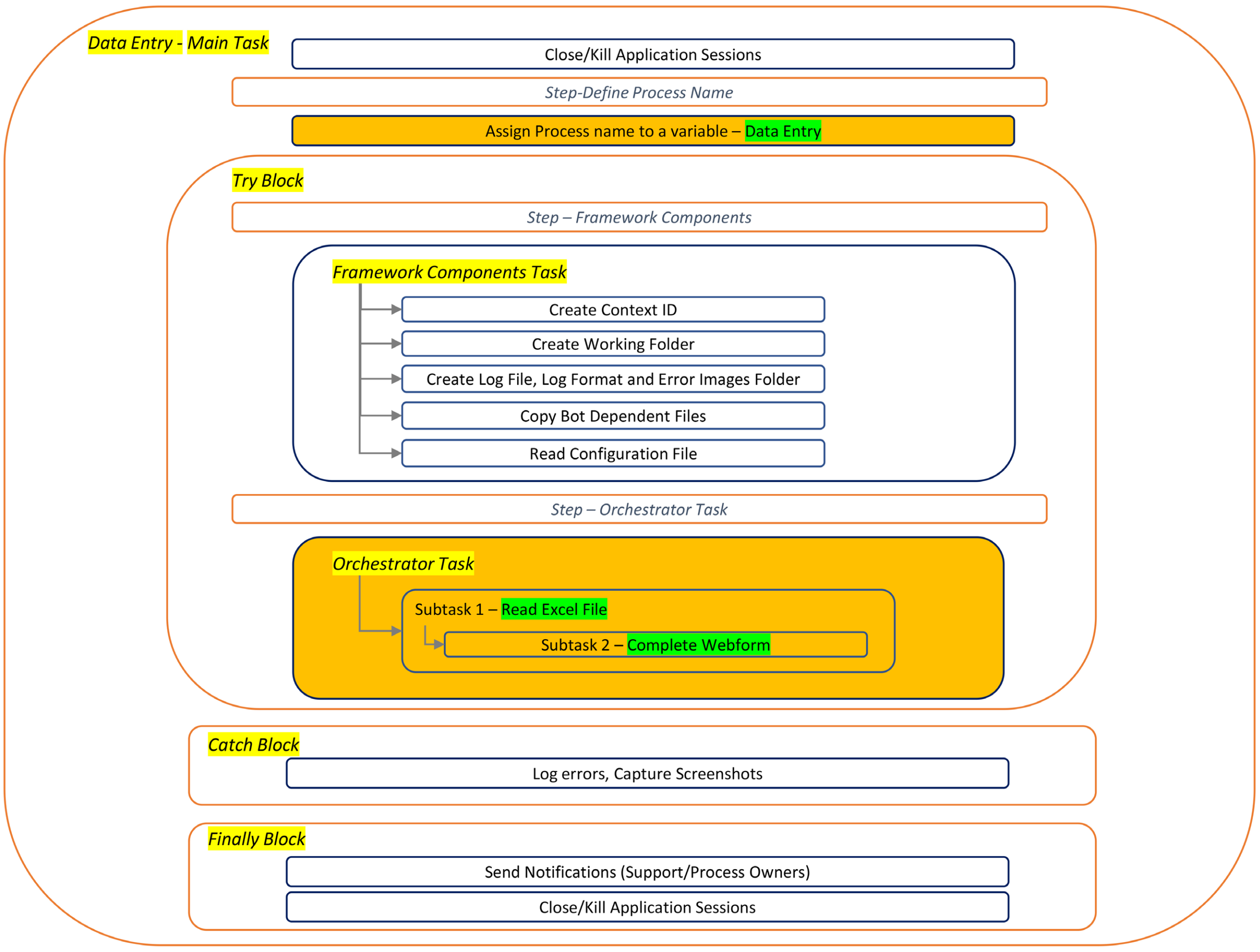
Data Entry - Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Data Entry
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Orchestrator Task
Subtask 1 – Read Excel File
Subtask 2 – Complete Webform
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 23. Data entry end to end Bot design.

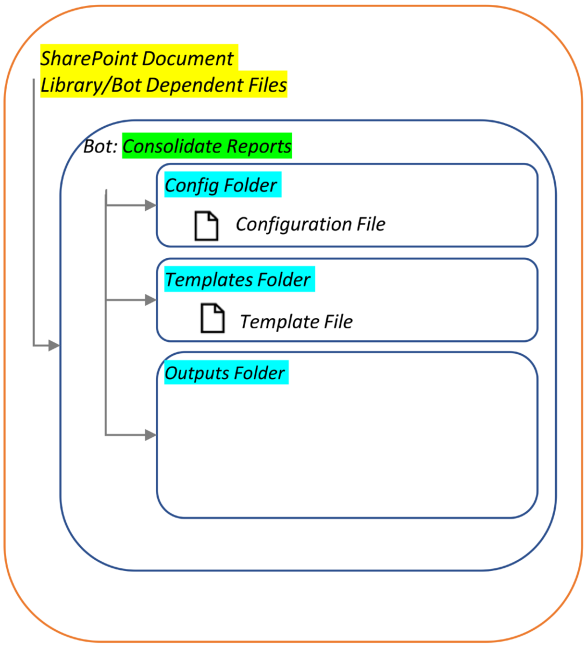

Figure 24. "Consolidate reports" folder in Bot dependencies repository.

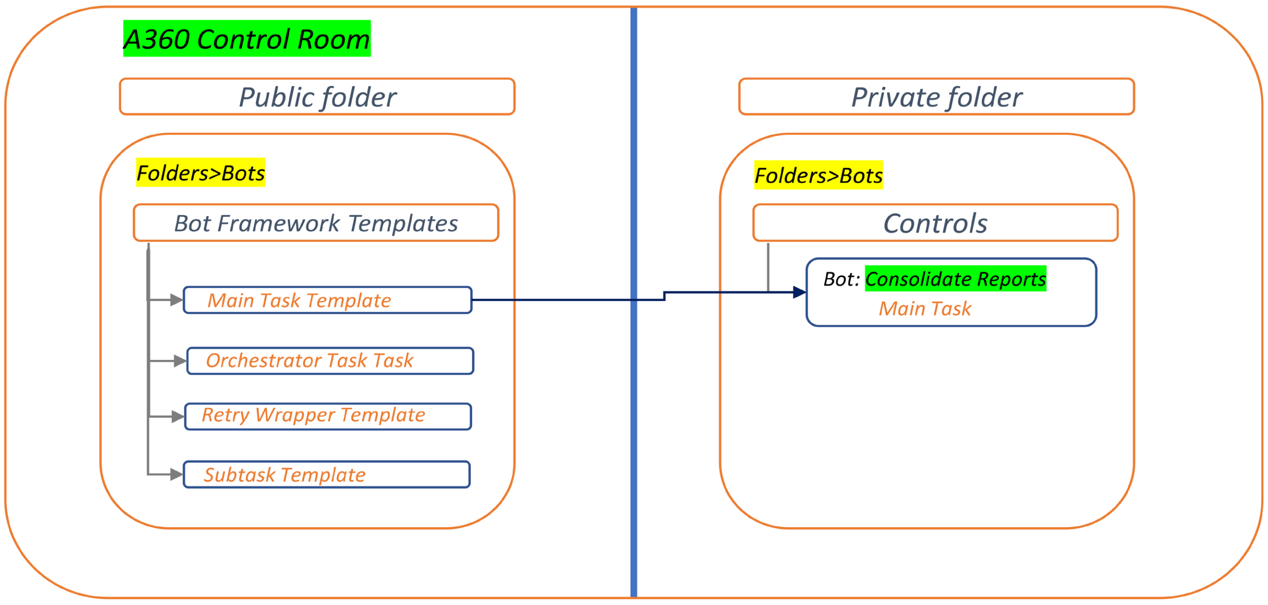

Figure 25. Copy main task template.

Enter the bot's name "Consolidate Reports" into the Process name placeholder within the main task (**Figure 26**). This action establishes the context for the bot's workflow, which involves downloading files from the designated bot name folder in the repository to the working folder and executing the process.

3) Framework Integration Task

The framework integration task is already integrated into the main task and operates in alignment with the context of the bot/process name. The Framework Integration task generates a context ID, creates a working folder in the bot's named folder in the user's documents directory, generates log files and log formats, establishes an error images folder, copies dependencies from the bot dependencies repository to the working folder, and reads the configuration file (**Figure 27**). The process name serves as the key input parameter for this task.

4) Orchestrator Task Setup

Create a folder for the bot to store its orchestrator task. Copy the "Orchestrator Task" template from the public folder to the bot folder (**Figure 28**). Invoke the orchestrator task from the bot's main task and establish mappings for the variables: "Context ID", "Working folder path", "Outputs dated folder path", "Config file name", "Template file names", "Model file names" and "Configuration Dictionary".

5) Subtask Task Setup

Create the subtasks: "Download Web file", "Download Windows File" and "Run Excel Macro". The "Download Web file" task downloads files from a web application, the "Download Windows File" task extracts data from a Windows application, and the "Run Excel Macro" task consolidates the data into a final report. These three subtasks are integrated within the orchestrator task (**Figure 29**). The predefined variables within the orchestrator, including the configuration dictionary and working folder, are mapped to these subtasks.

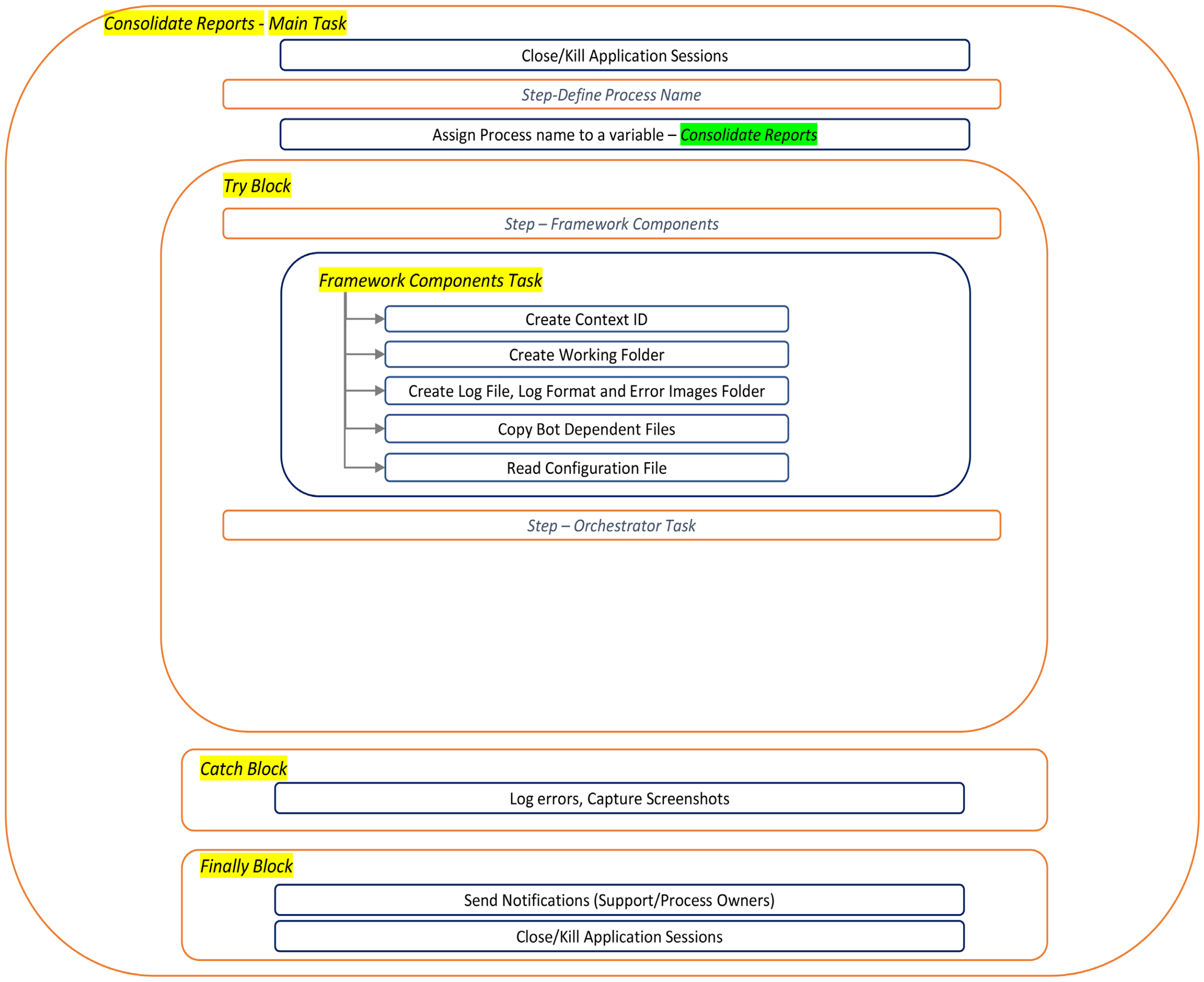
Consolidate Reports - Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Consolidate Reports
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 26. Assign "consolidate reports" to process name place holder.

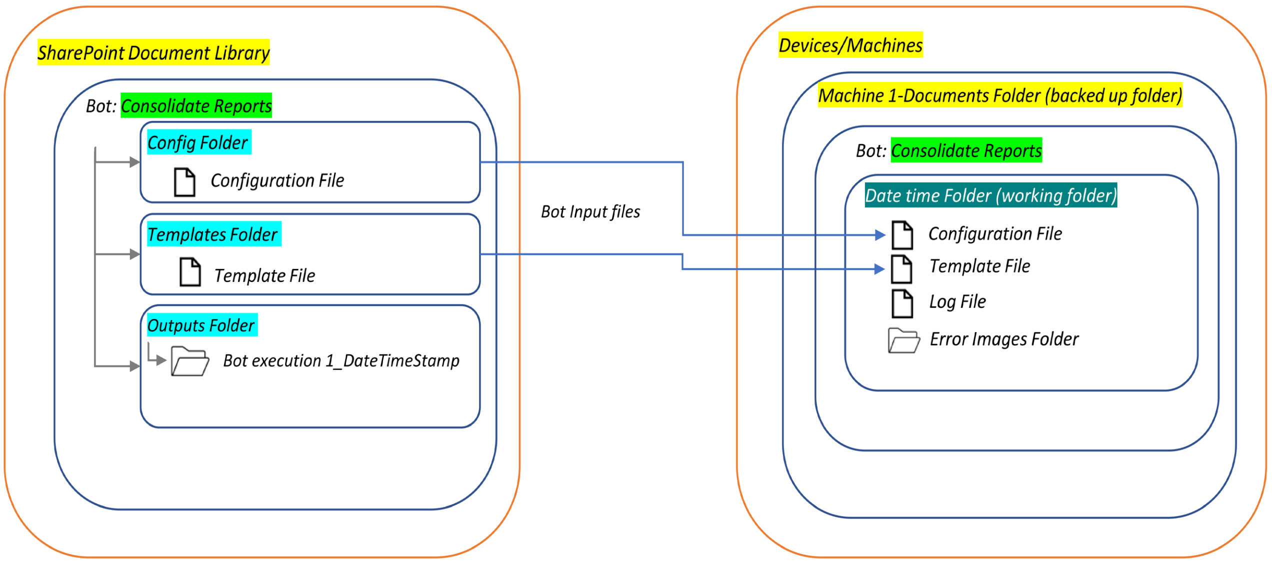

Figure 27. "Consolidate reports" framework inregration task functionality.

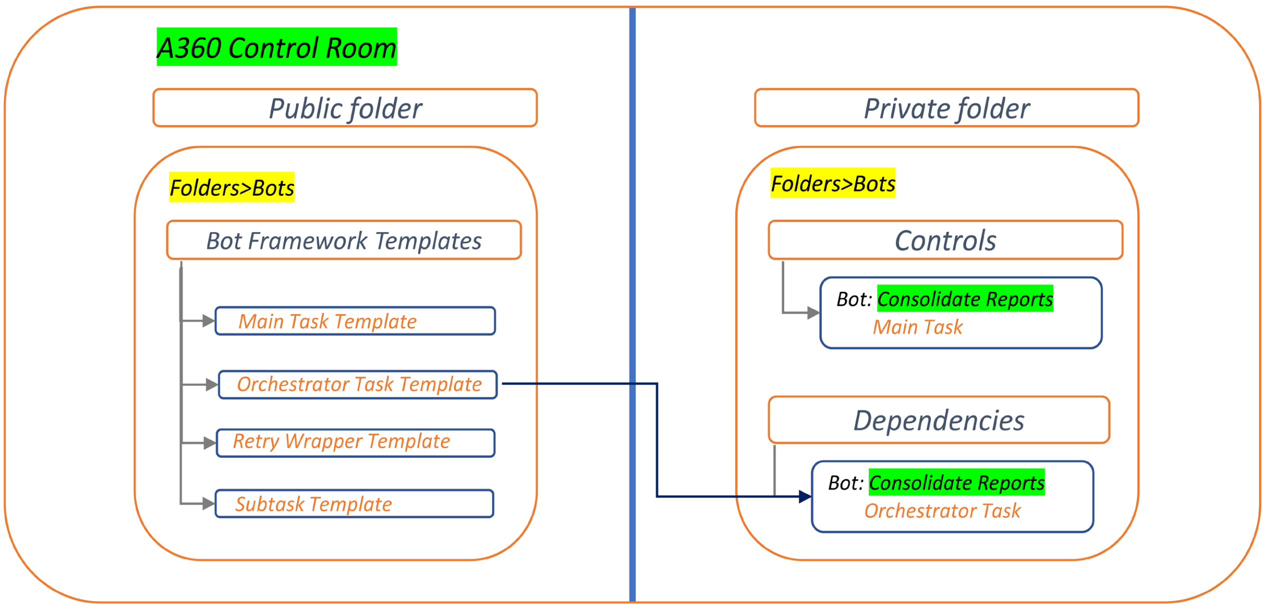

Figure 28. Copy orchestrator task template.

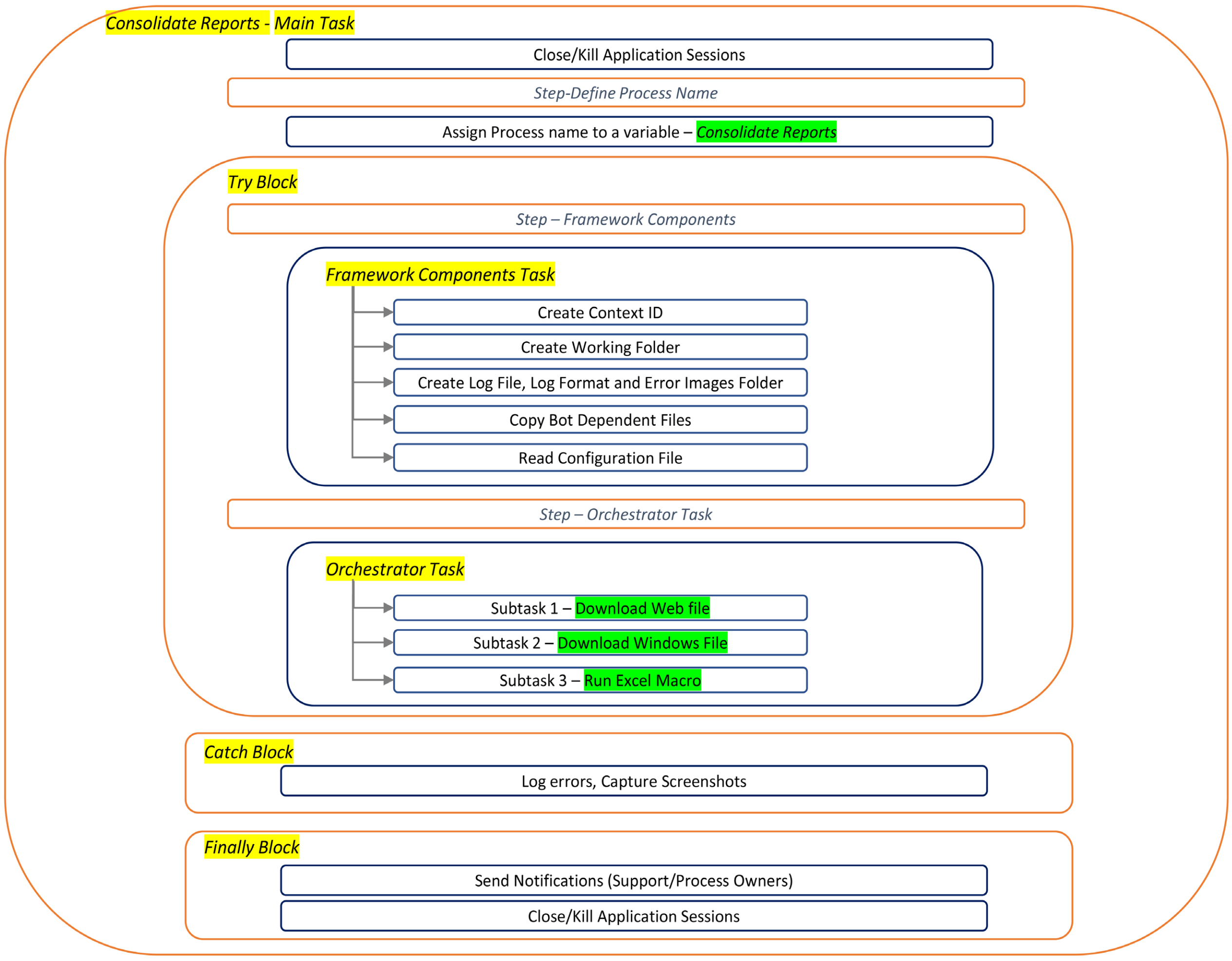
Consolidate Reports - Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Consolidate Reports
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Orchestrator Task
Subtask 1 – Download Web file
Subtask 2 – Download Windows File
Subtask 3 – Run Excel Macro
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 29. Create subtasks.

In conclusion, throughout the bot development process, a developer only needs to invest time in creating the subtasks "Download Web file", "Download Windows File", and "Run Excel Macro". The remaining templates are utilized as they are and integrated to construct a comprehensive end-to-end automation solution (**Figure 30**). This approach substantially reduces development time, enhances efficiency, minimizes errors, and elevates overall automation quality, consistency, and reliability.

4. Benefits of the Solution

The "A360 Bot Framework" offers several significant benefits to the field of Robotic Process Automation (RPA) and automation solutions developed using A360 [4] [19] [20].

4.1. Consistency and Standardization

The framework establishes a standardized approach to bot development. By adhering to predefined templates and practices, developers ensure a consistent structure and design across various automation solutions. This consistency reduces the risk of errors and enhances the overall quality of the developed bots.

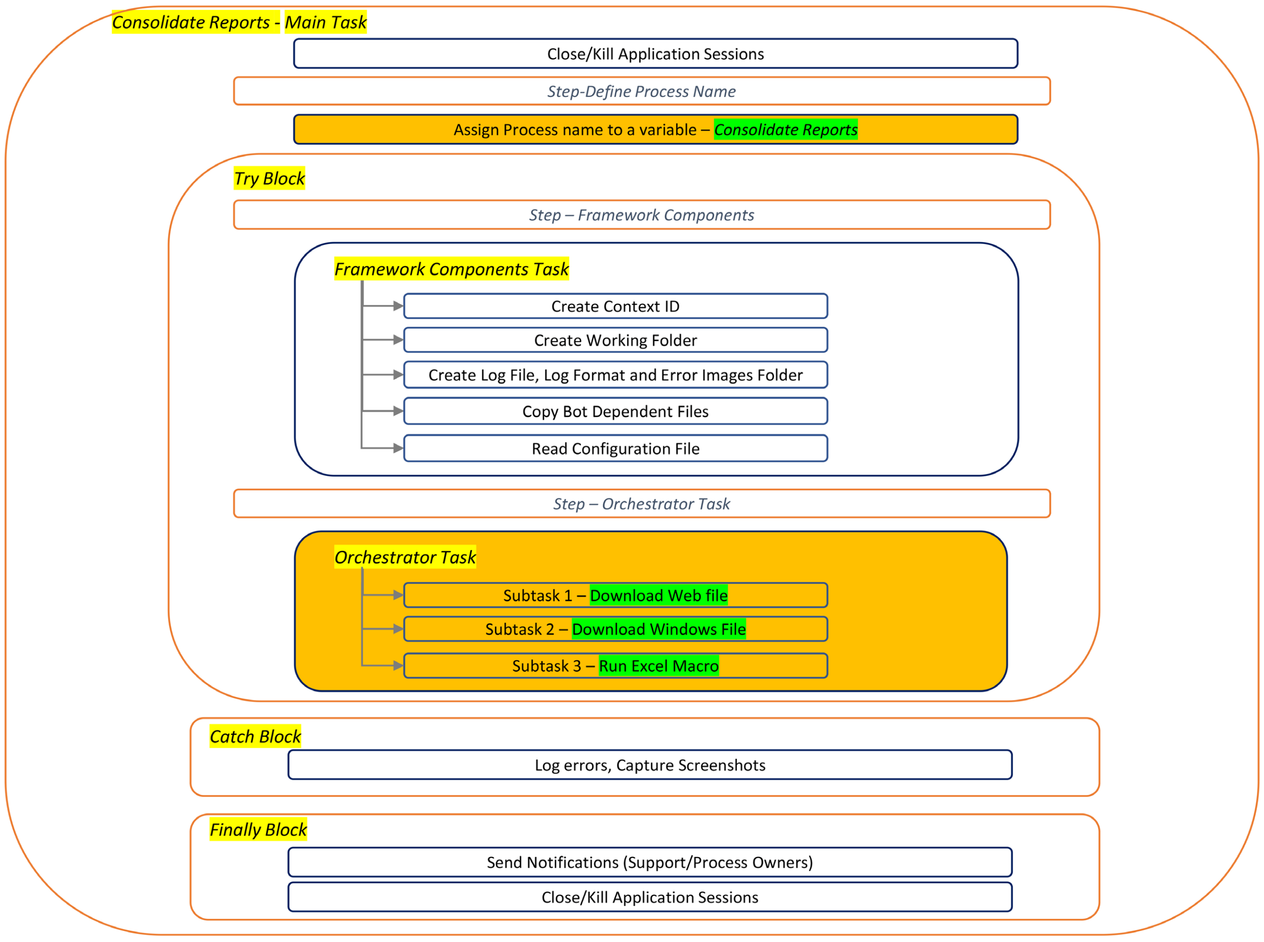
Consolidate Reports - Main Task
Close/Kill Application Sessions
Step-Define Process Name
Assign Process name to a variable – Consolidate Reports
Try Block
Step – Framework Components
Framework Components Task
Create Context ID
Create Working Folder
Create Log File, Log Format and Error Images Folder
Copy Bot Dependent Files
Read Configuration File
Step – Orchestrator Task
Orchestrator Task
Subtask 1 – Download Web file
Subtask 2 – Download Windows File
Subtask 3 – Run Excel Macro
Catch Block
Log errors, Capture Screenshots
Finally Block
Send Notifications (Support/Process Owners)
Close/Kill Application Sessions

Figure 30. Consolidate reports end to end Bot design.

4.2. Efficient Development

The framework accelerates the development lifecycle by providing developers with a well-defined starting point. Developers can focus on crafting functionalities specific to their use cases rather than reinventing common practices for each bot. This streamlined approach saves time, reduces redundancy, and promotes efficiency.

4.3. Reduced Risk and Resilience

With built-in error handling mechanisms and logging practices, the framework enhances the resiliency of developed bots. Error capture, reporting, and resolution processes are standardized, making it easier to identify and address issues during bot execution. This results in more robust and reliable automation solutions.

4.4. Simplified Collaboration

The framework facilitates collaboration between professional developers and citizen developers within an organization. It offers a structured methodology that is easy to understand and follow, enabling smoother communication and knowledge sharing across teams.

4.5. Scalability and Reusability

The framework's standardized components can be reused across different automation projects. This reusability promotes scalability by providing a consistent foundation for building a variety of automation solutions, regardless of their complexity.

4.6. Enhanced Debugging

The standardized logging mechanisms and error handling procedures make debugging and troubleshooting more efficient. Developers can easily track the execution flow, identify errors, and access relevant logs, leading to quicker problem resolution.

4.7. Lower Learning Curve

The user-friendly interface of the A360 platform, combined with the A360 Bot Framework, reduces the learning curve for both professional developers and citizen developers. This enables quicker onboarding and faster adoption of RPA within the organization.

4.8. Innovation

By abstracting common tasks and design patterns, the framework frees developers from repetitive tasks, allowing them to concentrate on creating value-added features and addressing unique automation challenges. This empowers developers to focus on innovation and creativity.

4.9. Effective Bot Management

The framework's standardized communication and logging practices enable effective bot management and monitoring. Auditing and compliance efforts are simplified due to the comprehensive logs and records maintained throughout bot executions.

4.10. Business Agility

The framework's efficiency and consistency contribute to quicker deployment and adaptation of automation solutions. This agility is crucial in rapidly evolving business environments, allowing organizations to respond promptly to changing needs.

5. Conclusions

In conclusion, the research paper has delved into the transformative potential of the "A360 Bot Framework" within the realm of Robotic Process Automation (RPA) and automation solution development. The framework, conceptualized within the context of this research, serves as a robust foundation that empowers both professional developers and citizen developers to create automation solutions that are efficient, consistent, and resilient.

The paper has highlighted the importance of A360, a cloud-based platform that simplifies the creation of automation solutions through its user-friendly interface, seamless integrations, and a repository of pre-designed actions. The "A360 Bot Framework" emerges as a response to the growing demand for systematic and structured approaches to bot development, providing a standardized blueprint that ensures uniformity, scalability, and success.

Through comprehensive case studies, the paper has showcased the versatility of the framework in addressing diverse automation challenges. From automating web form filling using Excel data to downloading and consolidating data from various sources, the "A360 Bot Framework" acts as a guiding compass, accelerating development while maintaining consistency and reliability.

The solution's benefits are far-reaching. It fosters consistency and standardization, accelerates development cycles, reduces risk through robust error management, and simplifies collaboration between developers and teams. Additionally, the framework's streamlined approach allows developers to focus on innovation, creating value-added features and solutions. The benefits extend across multiple business functions, enhancing efficiency and aligning with the broader goals of digital transformation.

In the ever-evolving landscape of automation, the "A360 Bot Framework" stands as a testament to the power of innovation and collaboration. As technology continues to reshape industries, this framework offers a steadfast pathway towards efficient, consistent, and transformative automation solutions. It is a testament to the evolution of RPA, where the fusion of human ingenuity and intelligent automation brings forth a new era of possibilities.

Conflicts of Interest

The author, Sai Madhur Potturu, is employed at Zoetis Inc., specifically in the Robotics Center of Excellence (CoE) department. Zoetis Inc. is a company that provides animal healthcare products and services. The development and implementation of the digital solution presented in this manuscript align with the author's role and responsibilities within the organization. The author declares no financial or personal relationships that may have influenced the content or findings presented in this manuscript.

Data Availability Statement

The data used to support the findings of this study are available from the corresponding author upon reasonable request. The data include the PowerApps application design, RPA solution implementation details, and relevant datasets used for testing and evaluation. Access to the data will be provided to researchers or individuals to replicate the study findings or conduct further analyses related to the presented digital solution.

References

[1] Automation Anywhere (2023) Automation 360. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/security-architecture/cloud-automation-anywhere-enterprise-overview.html

[2] Mahey, H. (2020) Robotic Process Automation with Automation Anywhere: Techniques to Fuel Business Productivity and Intelligent Automation Using RPA. Packt Publishing Ltd., Birmingham.

[3] Sorin, A. (2017) Robotic Automation Process—The Next Major Revolution in Terms of Back Office Operations Improvement. *Proceedings of the International Conference on Business Excellence*, **11**, 676-686. https://doi.org/10.1515/picbe-2017-0072

[4] Mullakara, N. and Asokan, A. K. (2020) Robotic Process Automation Projects: Build Real-World RPA Solutions Using UiPath and Automation Anywhere. Packt Publishing Ltd., Birmingham.

[5] Automation Anywhere (2023) What Is Citizen Development and How Can You Benefit from It? https://www.automationanywhere.com/company/blog/rpa-thought-leadership/what-citizen-development-and-how-can-you-benefit-it

[6] Automation Anywhere (2020) Tips for Scaling Automation #1: Build a Bot Shell. https://www.youtube.com/watch?v=L6ichPI2EAQ

[7] Eight Best Practices for RPA Developers. https://www.cai.io/resources/thought-leadership/eight-best-practices-for-rpa-developers

[8] Potturu, S. (2023) Maximizing the Efficiency of Automation Solutions with Automation 360: Approaches for Developing Subtasks and Retry Framework. *Intelligent Control and Automation*, **14**, 19-35. https://doi.org/10.4236/ica.2023.142002

[9] Automation Anywhere (2023) Bot Runner Overview. https://docs.automationanywhere.com/bundle/enterprise-v11.3/page/enterprise/top

ics/aae-architecture-implementation/bot-runner-overview.html

[10] Automation Anywhere (2023) Attended and Unattended Automation. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/attended-automation/attend-automation-overview.html

[11] Automation Anywhere (2023) Global Values. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/using-variables/cloud-global-variables.html

[12] Automation Anywhere (2023) Using Log to File Action. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/cloud-inserting-log-to-file-command.html

[13] Automation Anywhere (2023) Task Bot Package. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/cloud-taskbot-command.html

[14] Automation Anywhere (2023) Check in a Bot. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/working-with-automation-tasks/cloud-bot-check-in.html

[15] Automation Anywhere (2023) Check out a Single Bot. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/working-with-automation-tasks/cloud-bot-check-out.html

[16] Automation Anywhere (2023) Enabling Version Control in Automation Anywhere Control Room. https://docs.automationanywhere.com/bundle/enterprise-v11.3/page/enterprise/topics/aae-client/bot-creator/using-special-features/enabling-version-control-in-automation-anywhere.html

[17] Automation Anywhere (2023) Integrating Control Room with Git Repositories. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/control-room/git-integration/cloud-cr-git-integration.html

[18] Automation Anywhere (2023) Copy a Bot. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/working-with-automation-tasks/cloud-copy-a-bot.html

[19] Beachnet (2023) The Benefits of Automation for Different Industries. https://www.beachnet.com/industries-automation-benefits/

[20] KOFAX (2020) 7 Biggest Benefits of RPA (Robotic Process Automation). https://www.kofax.com/learn/blog/benefits-of-rpa