Intelligent Control and Automation, 2023, 14, 19-35
https://www.scirp.org/journal/ica
ISSN Online: 2153-0661
ISSN Print: 2153-0653

# Maximizing the Efficiency of Automation Solutions with Automation 360: Approaches for Developing Subtasks and Retry Framework

**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) 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

**Received:** April 15, 2023
**Accepted:** May 28, 2023
**Published:** May 31, 2023

## Abstract

In this paper, I present a solution that explores the use of A360 subtasks as a comparable concept to functions in programming. By leveraging subtasks as reusable and maintainable functions, users can efficiently develop customized high-quality automation solutions. Additionally, the paper introduces the retry framework, which allows for the automatic retrying of subtasks in the event of system or unknown exceptions. This framework enhances efficiency and reduces the manual effort required to retrigger bots. The A360 Subtask and Retry Framework templates provide valuable assistance to both professional and citizen developers, improving code quality, maintainability, and the overall efficiency and resiliency of automation solutions.

## Keywords

Automation 360, Robotics Process Automation (RPA), Subtasks, Retry Framework, Efficiency, Resiliency, Exception Handling, Reusability

## 1. Introduction

Automation Anywhere is a top player in the RPA and Intelligent automation space [1] [2] [3]. 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.

A360 is a No-Code/Low-Code platform [4]. It allows users to develop automation solutions without any prior programming experience, making it an accessible platform for all skill levels [4]. While A360 does not have a native concept of functions [5] [6], this article explores the potential of subtasks as a comparable concept to functions in programming.

A Task Bot is an interface where a developer can access pre-defined and custom packages to develop a software robot. A Subtask is created by using a task bot in Automation Anywhere.

In Automation Anywhere, a subtask is a smaller unit of work within an automation process or bot [7]. 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 [8].

This paper provides an in-depth explanation of 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, this paper will cover the retry framework, which 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.

## 2. Solution

The solution provided explains different aspects of task classification and methods of writing subtasks in A360. The first part discusses the classification of tasks into main tasks and subtasks, where main tasks integrate multiple subtasks within a process or bot [9]. The second part focuses on three different methods for writing subtasks: resilient subtasks, exception-bubbling subtasks, and retry frameworks for resilient and exception-bubbling subtasks. Each method is described with steps, scenarios, advantages, and suggested use. Resilient subtasks handle exceptions within the subtask while exception-bubbling subtasks throw exceptions to the caller/main task. Retry frameworks allow the retrying of subtasks until the desired output is achieved, improving bot efficiency and reducing manual support efforts.

### 2.1. Classification of Tasks

In A360 the tasks are classified primarily into a Main task and a Subtask.

- A Main task is a task that integrates all the Subtasks and is considered a Process/Bot [9].
- A Subtask is typically a function that performs a specific piece of work within the process.

For example, consider the scenario of an 'Inventory Reporting' process where a bot needs to download a report from SAP and sends it as an email attachment to the manager every day at 9 AM.

This process can include two subtasks and one main task.

Subtasks:

• Export Report: Download reports from SAP.

• Send Email: Send an email with an attachment to the manager.

Main Task:

• Inventory Report: Integrates the above Subtasks. This Main task will be scheduled to run at 9 AM every day in the control room.

However, there is no difference in the interface or availability of commands/packages in the Main Task or Subtask. It is just a classification depending on their usage.

## 2.2. Different Methods of Writing Subtasks

The following section presents different approaches for writing a subtask and integrating different types of subtasks within the retry framework.

### 2.2.1. Resilient Subtasks

Subtasks that can capture and act on the exceptions. (Figure 1)

**1) Steps**

• Begin by writing the code within a Try block [10] [11].

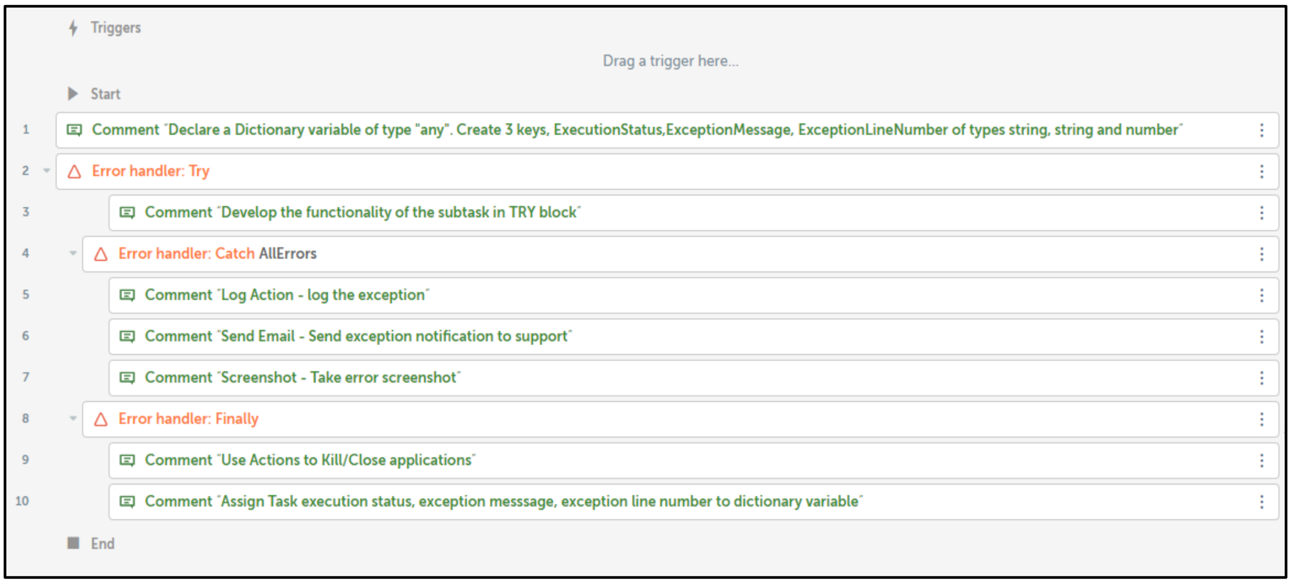
• If an exception occurs within the Try block, all subsequent code is skipped, and the program moves to the Catch block [11] [12].

• The Catch block catches and handles any errors (e.g., logging the error, capturing a screenshot, sending an email) [12].

• The "Finally" block can be utilized to perform any necessary actions after the task, regardless of whether an exception has occurred [13].

**2) Scenarios**

The behavior of resilient subtasks is described in cases where errors occur and in cases where errors do not occur within the subtask.



```
Triggers
                    Drag a trigger here...
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus,ExceptionMessage, ExceptionLineNumber of types string, string and number"
Error handler: Try
    Comment "Develop the functionality of the subtask in TRY block"
  Error handler: Catch AllErrors
    Comment "Log Action - log the exception"
    Comment "Send Email - Send exception notification to support"
    Comment "Screenshot - Take error screenshot"
  Error handler: Finally
    Comment "Use Actions to Kill/Close applications"
    Comment "Assign Task execution status, exception messsage, exception line number to dictionary variable"
End
```

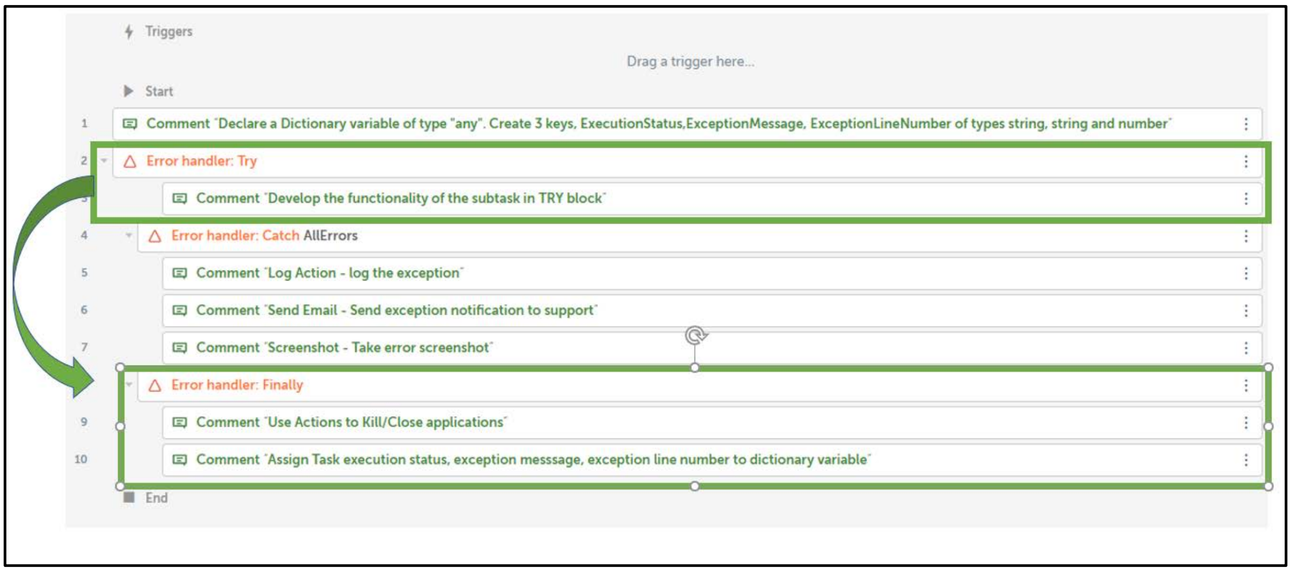
Figure 1. A360 Resilient task syntax.

**a) No Error:**

The code in the try block is executed then the code in the "Finally" block is executed [11] [13]. The catch block code will not be executed. (**Figure 2**)

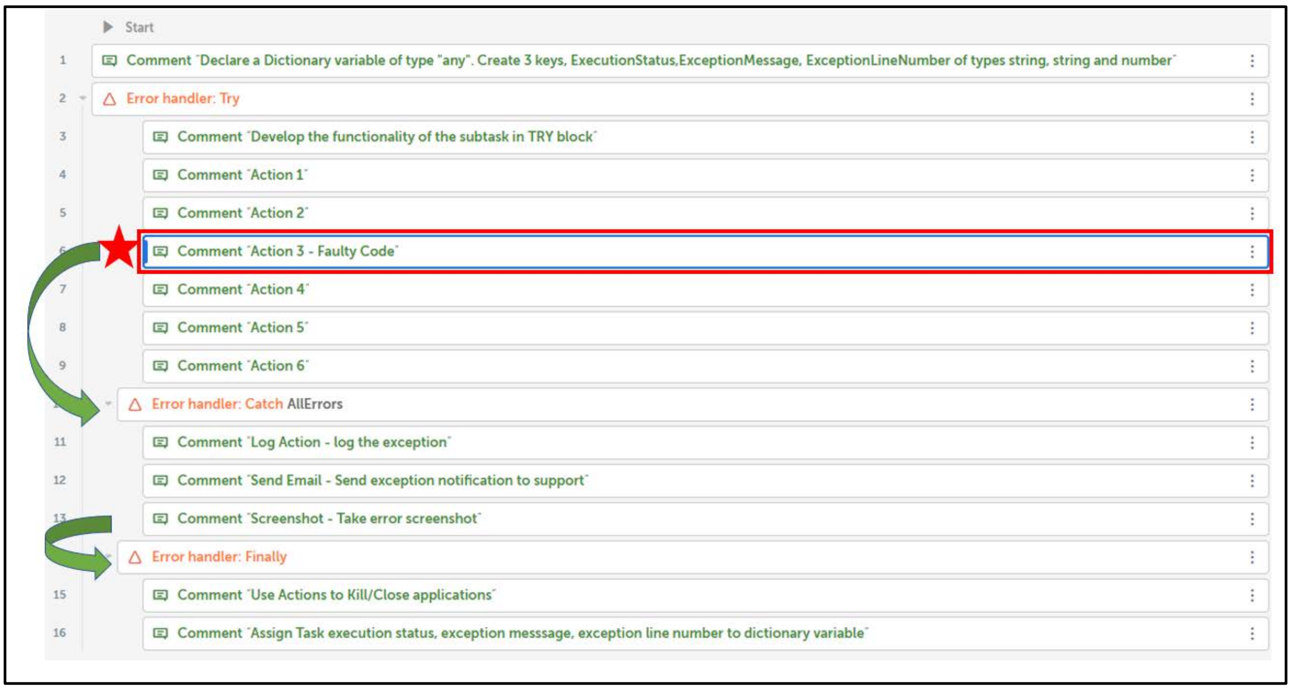
**b) Error:**

When an error occurs in the try block [11], the code after the faulty code is skipped, and the control jumps to the Catch block [12]. The Catch block captures the error and handles the exception. The "Finally" block code will be executed [13]. (**Figure 3**)



```
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus,ExceptionMessage, ExceptionLineNumber of types string, string and number"
Error handler: Try
    Comment "Develop the functionality of the subtask in TRY block"
Error handler: Catch AllErrors
    Comment "Log Action - log the exception"
    Comment "Send Email - Send exception notification to support"
    Comment "Screenshot - Take error screenshot"
Error handler: Finally
    Comment "Use Actions to Kill/Close applications"
    Comment "Assign Task execution status, exception messsage, exception line number to dictionary variable"
```

Figure 2. A360 Resilient task with no error syntax.



```
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus,ExceptionMessage, ExceptionLineNumber of types string, string and number"
Error handler: Try
    Comment "Develop the functionality of the subtask in TRY block"
    Comment "Action 1"
    Comment "Action 2"
    Comment "Action 3 - Faulty Code"
    Comment "Action 4"
    Comment "Action 5"
    Comment "Action 6"
Error handler: Catch AllErrors
    Comment "Log Action - log the exception"
    Comment "Send Email - Send exception notification to support"
    Comment "Screenshot - Take error screenshot"
Error handler: Finally
    Comment "Use Actions to Kill/Close applications"
    Comment "Assign Task execution status, exception messsage, exception line number to dictionary variable"
```

Figure 3. A360 Resilient task with no error syntax.

**3) Advantages**

This approach makes the task independent and enables it to handle exceptions. The method can provide the caller task with exception information for further validation and actions, allowing the caller to determine whether to continue or exit the process [13].

**4) Suggested Use**

Use this approach when subtasks are independent of each other within the process/Main task. Each subtask receives specific inputs from the caller/main task.

### 2.2.2. Exception Bubbling Subtasks

Subtask that captures and throws the exception to caller/main task. (Figure 4)

**1) Steps**

• Create a task with a Try-Catch-Finally block structure [10] [11] [12] [13].

• Write the code within the Try block.

• When an exception occurs within the Try block, the code below the faulty code is skipped and the exception is caught by the Catch block.

• The catch block throws [14] the exception to the caller/main task for further handling.

• Use the "Finally" block to perform actions at the end of the task, regardless of whether an exception was encountered or not.

**2) Scenarios**

The behavior of Exception Bubbling subtasks is described in cases where errors occur and in cases where errors do not occur within the subtask.

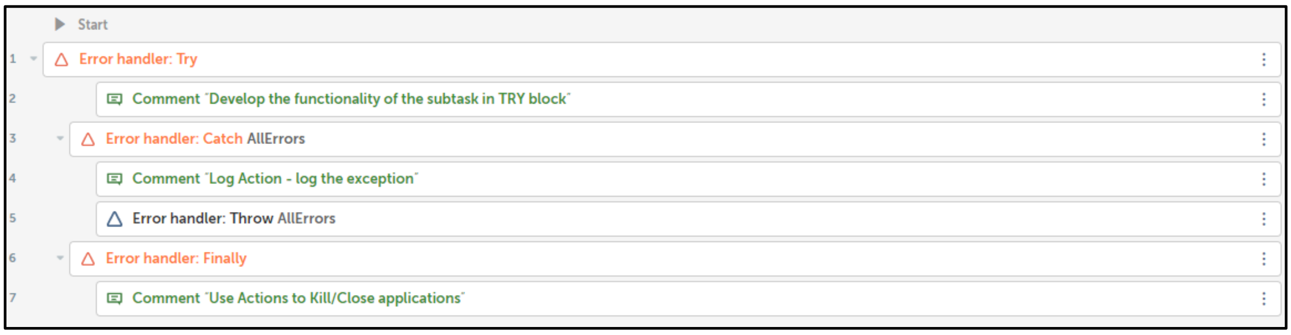
**a) No Error:**

The code in the Try block is executed then the code in the "Finally" block is executed. The Catch block code will not be executed. (Figure 5)

**b) Error:**

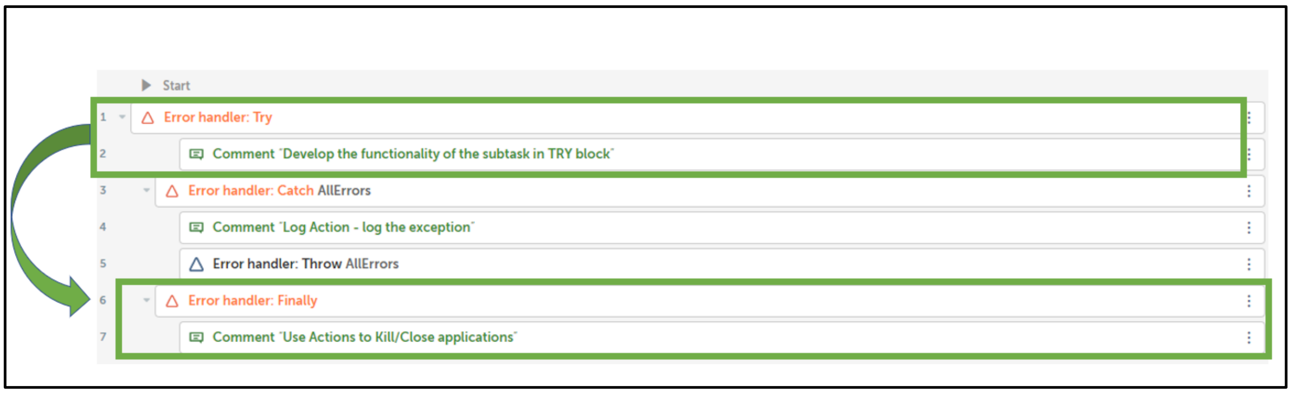
The code in the Try block is executed. If an exception occurs at a faulty/error line, the control jumps to the catch block, skipping the code after the faulty code. The Catch block captures the error and throws the exception to the caller/main task to handle. The "Finally" block code will be executed. (Figure 6)

**3) Advantages**



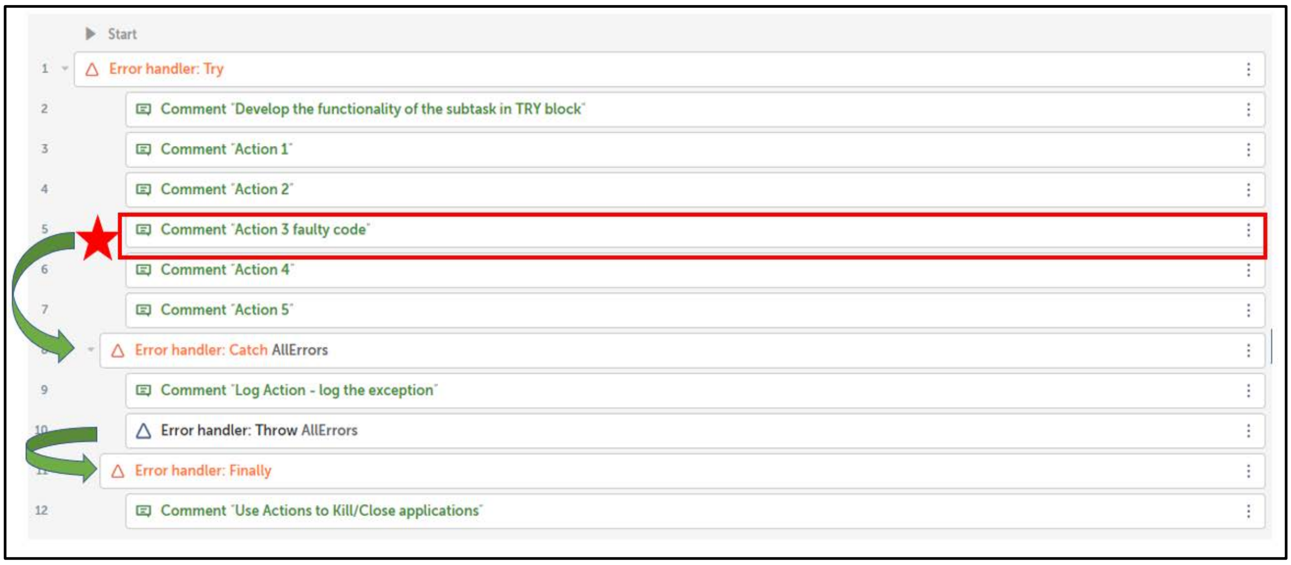
```
Start
Error handler: Try
    Comment "Develop the functionality of the subtask in TRY block"
Error handler: Catch AllErrors
    Comment "Log Action - log the exception"
    Error handler: Throw AllErrors
Error handler: Finally
    Comment "Use Actions to Kill/Close applications"
```

Figure 4. A360 Exception bubbling syntax.



Start
1 Error handler: Try
2 Comment "Develop the functionality of the subtask in TRY block"
3 Error handler: Catch AllErrors
4 Comment "Log Action - log the exception"
5 Error handler: Throw AllErrors
6 Error handler: Finally
7 Comment "Use Actions to Kill/Close applications"

**Figure 5.** A360 Exception bubbling no error syntax.



Start
1 Error handler: Try
2 Comment "Develop the functionality of the subtask in TRY block"
3 Comment "Action 1"
4 Comment "Action 2"
5 Comment "Action 3 faulty code"
6 Comment "Action 4"
7 Comment "Action 5"
8 Error handler: Catch AllErrors
9 Comment "Log Action - log the exception"
10 Error handler: Throw AllErrors
11 Error handler: Finally
12 Comment "Use Actions to Kill/Close applications"

**Figure 6.** A360 Exception bubbling error syntax.

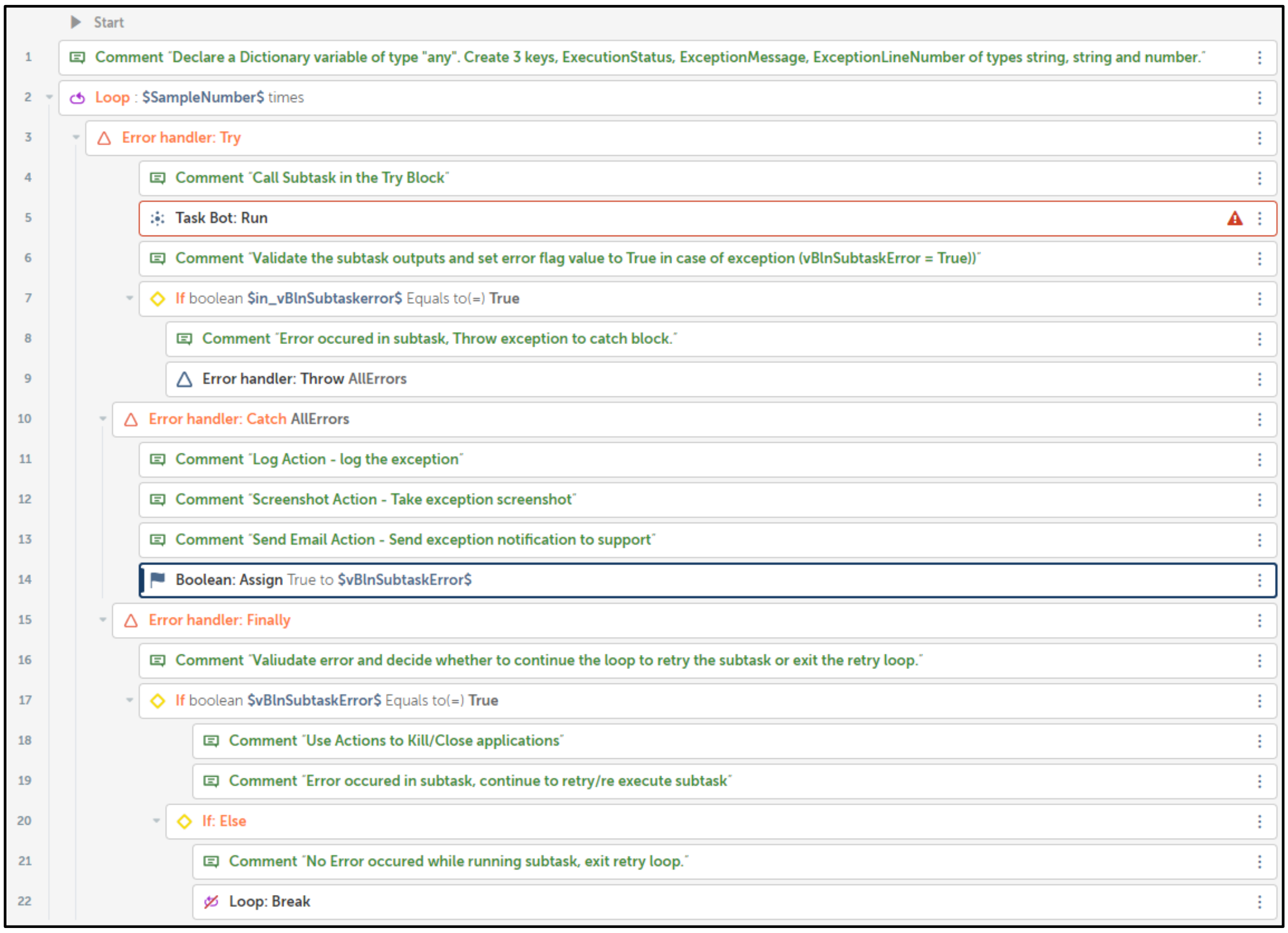
This approach helps to avoid redundant lines of exception-handling code in every subtask.

**4) Suggested Use**

This approach is suitable for cases where subtasks are interdependent, meaning that they rely on each other's outputs. If subtask 1 fails, there is no need to execute subtask 2 in the caller/Main task.

### 2.2.3. Retry Framework for Subtask—Resilient Task

The resilient task [15] can handle the exceptions and return task outputs to the caller. When the retry task calls a resilient subtask, the subtask returns task status and error message values as outputs. The retry task validates the subtask outputs. If the subtask fails, the validation block throws an exception to the catch block to capture error information and rerun the subtask a specified number of times until it successfully completes the task. If the subtask runs successfully, the retry framework task will end its execution and return control to the main task. (**Figure 7**)



```
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus, ExceptionMessage, ExceptionLineNumber of types string, string and number."
Loop : $SampleNumber$ times
  Error handler: Try
      Comment "Call Subtask in the Try Block"
      Task Bot: Run
      Comment "Validate the subtask outputs and set error flag value to True in case of exception (vBlnSubtaskError = True))"
      If boolean $in_vBlnSubtaskerror$ Equals to(=) True
        Comment "Error occured in subtask, Throw exception to catch block."
        Error handler: Throw AllErrors
    Error handler: Catch AllErrors
      Comment "Log Action - log the exception"
      Comment "Screenshot Action - Take exception screenshot"
      Comment "Send Email Action - Send exception notification to support"
      Boolean: Assign True to $vBlnSubtaskError$
    Error handler: Finally
      Comment "Valiudate error and decide whether to continue the loop to retry the subtask or exit the retry loop."
      If boolean $vBlnSubtaskError$ Equals to(=) True
          Comment "Use Actions to Kill/Close applications"
          Comment "Error occured in subtask, continue to retry/re execute subtask"
        If: Else
          Comment "No Error occured while running subtask, exit retry loop."
          Loop: Break
```

Figure 7. A360 Retry Framework syntax—Resilient task.

**1) Steps**

• Create a "For loop" that executes for n times (where n is a configurable value passed from the caller task).

• Create a Try-Catch-Finally block [10] [11] [12] [13].

• Call the subtask within the Try block.

• The retry framework task validates the subtask outputs.

• If an error occurs in the subtask, the validation block in the retry framework task throws an exception to the catch block. The catch block captures the error, notifies the relevant support group, closes/kills applications, and clears the exception.

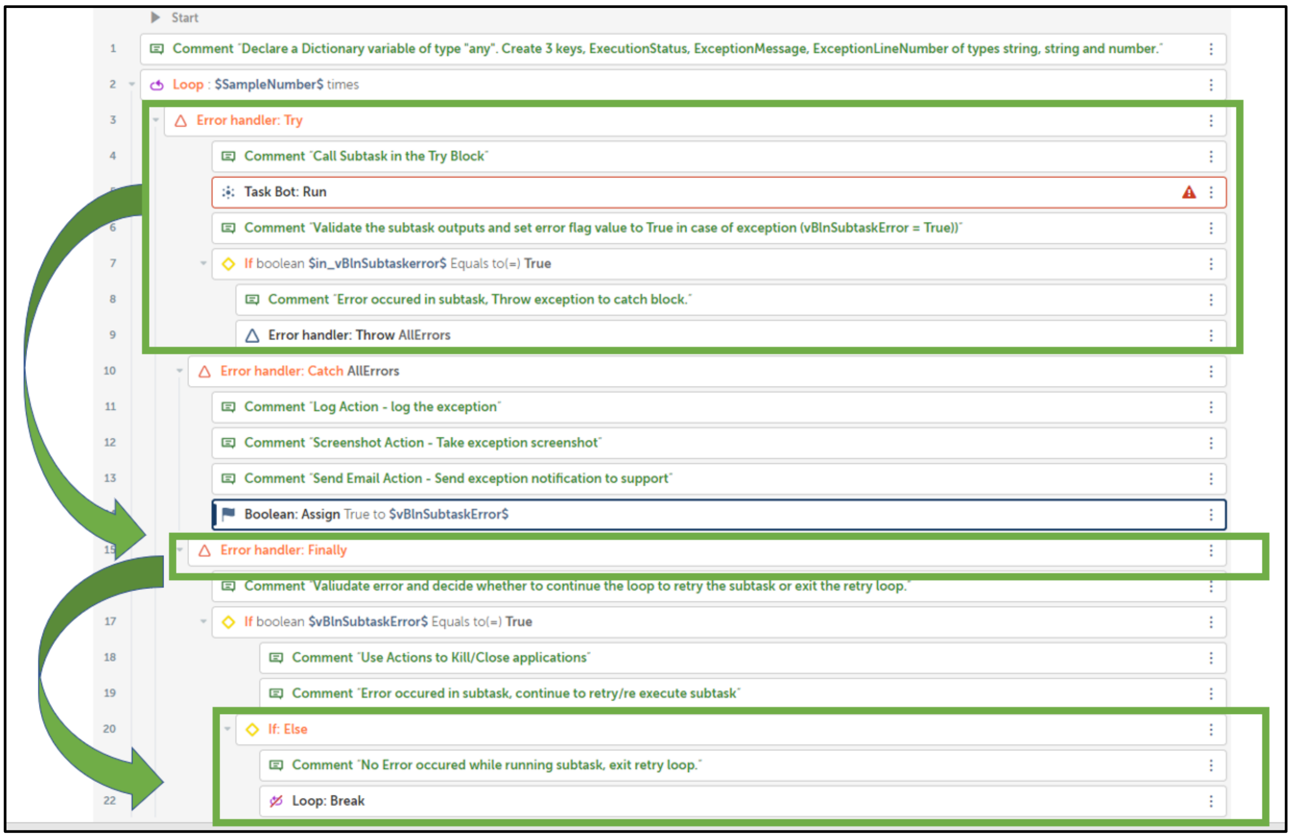
• The subtask will be retried until it either reaches the maximum retry limit or successfully completes the task.

**2) Scenarios**

The behavior of Retry framework is described in cases where errors occur and in cases where errors do not occur in the resilient subtask.

**a) No Error:**

The "Finally" block will validate the output from the subtask and exit the retry framework when the subtask runs successfully. (Figure 8)



```
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus, ExceptionMessage, ExceptionLineNumber of types string, string and number."
Loop : $SampleNumber$ times
  Error handler: Try
      Comment "Call Subtask in the Try Block"
      Task Bot: Run
      Comment "Validate the subtask outputs and set error flag value to True in case of exception (vBlnSubtaskError = True))"
      If boolean $in_vBlnSubtaskerror$ Equals to(=) True
        Comment "Error occured in subtask, Throw exception to catch block."
        Error handler: Throw AllErrors
    Error handler: Catch AllErrors
      Comment "Log Action - log the exception"
      Comment "Screenshot Action - Take exception screenshot"
      Comment "Send Email Action - Send exception notification to support"
      Boolean: Assign True to $vBlnSubtaskError$
    Error handler: Finally
      Comment "Valiudate error and decide whether to continue the loop to retry the subtask or exit the retry loop."
      If boolean $vBlnSubtaskError$ Equals to(=) True
          Comment "Use Actions to Kill/Close applications"
          Comment "Error occured in subtask, continue to retry/re execute subtask"
        If: Else
          Comment "No Error occured while running subtask, exit retry loop."
          Loop: Break
```

**Figure 8.** A360 Retry Framework no error syntax—Resilient task.

**b) Error:**

Catch block captures the error, notifies the relevant support group, closes/kills applications, and clears the exception. The "Finally" block validates the subtask outputs and reruns the subtask in the event of a system exception. (Figure 9)

**3) Advantages**

This method allows retry framework task to validate subtask outputs. If the subtask fails, the retry framework runs the subtask multiple times until the desired output is achieved. This method improves bot efficiency and reduces manual support effort to retrigger the process.

**4) Suggested Use**

This method is suitable for situations where the behavior of applications (web, windows, etc.) is unpredictable, and sometimes killing/closing the applications and relaunching them can fix any temporary issues/bugs and releases the system resources. For example, loading web browsers or SAP windows, etc. [15].

#### 2.2.4. Retry Framework for Subtask—Exception Bubbling Task

The Exception bubbling task [16] bubbles/throws the exceptions to the caller task. When the retry framework calls an exception bubbling subtask, the subtask either returns an exception or an output. If the subtask returns an exception, the subtask is marked as failed and the control jumps to the catch block in the retry framework task. The catch block will capture the error information and the subtask is retriggered for a specified number of times until it successfully completes the task. If the subtask runs successfully, the retry framework task will end its execution and return control to the main task. (**Figure 10**)

**1) Steps**

• Create a "For loop" that executes for n times (where N is a configurable value passed from the caller task).

• Create a Try-Catch-Finally block [10] [11] [12] [13].

• Call the subtask within the Try block.

• If the subtask throws an exception, the control jumps to the catch block in the retry framework task. The catch block captures the error, notifies the relevant support group, closes/kills applications, and clears the exception.

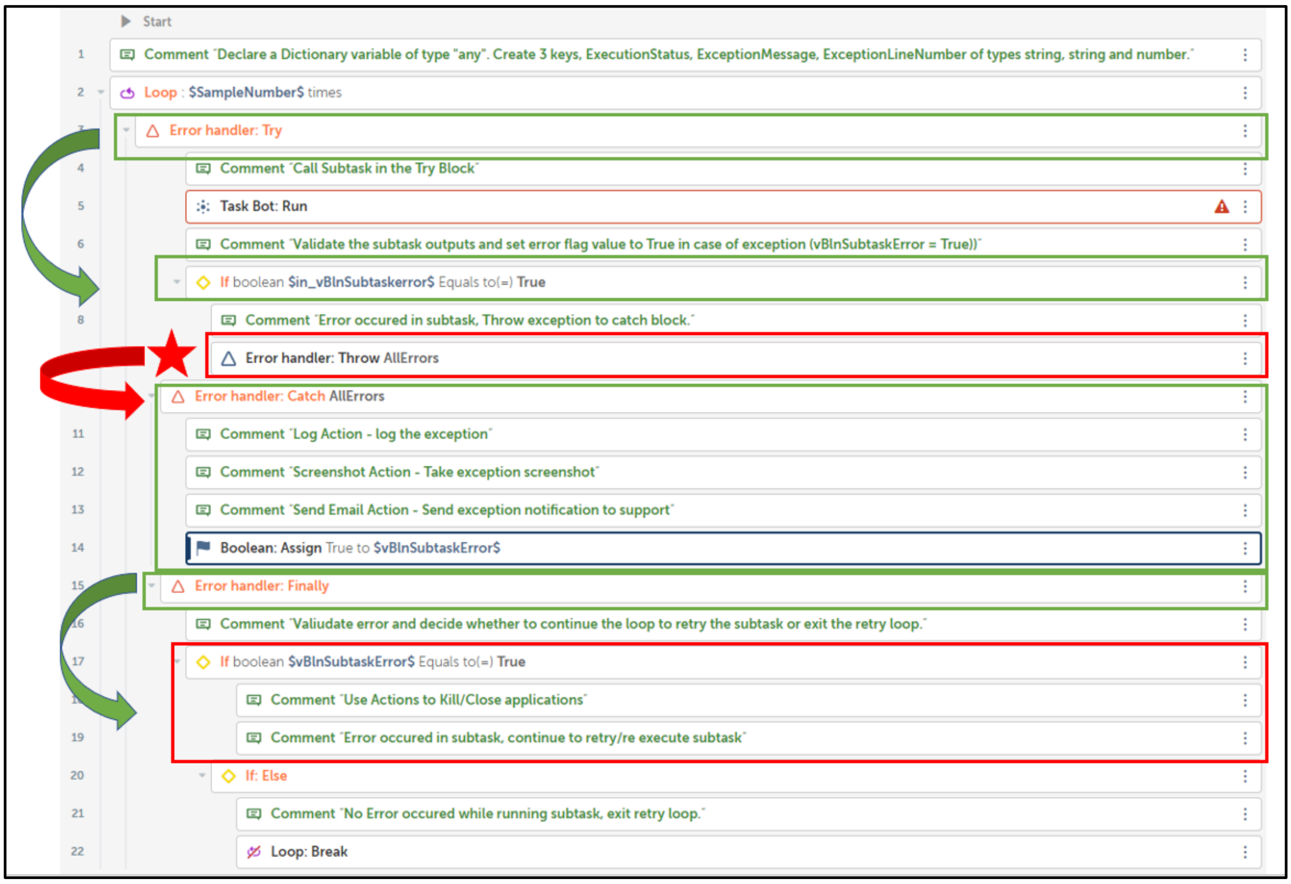
• The subtask will be retried until it either reaches the maximum retry limit or successfully completes the task.

**2) Scenarios**

The behavior of Retry framework is described in cases where errors occur and in cases where errors do not occur in the exception bubbling subtask.

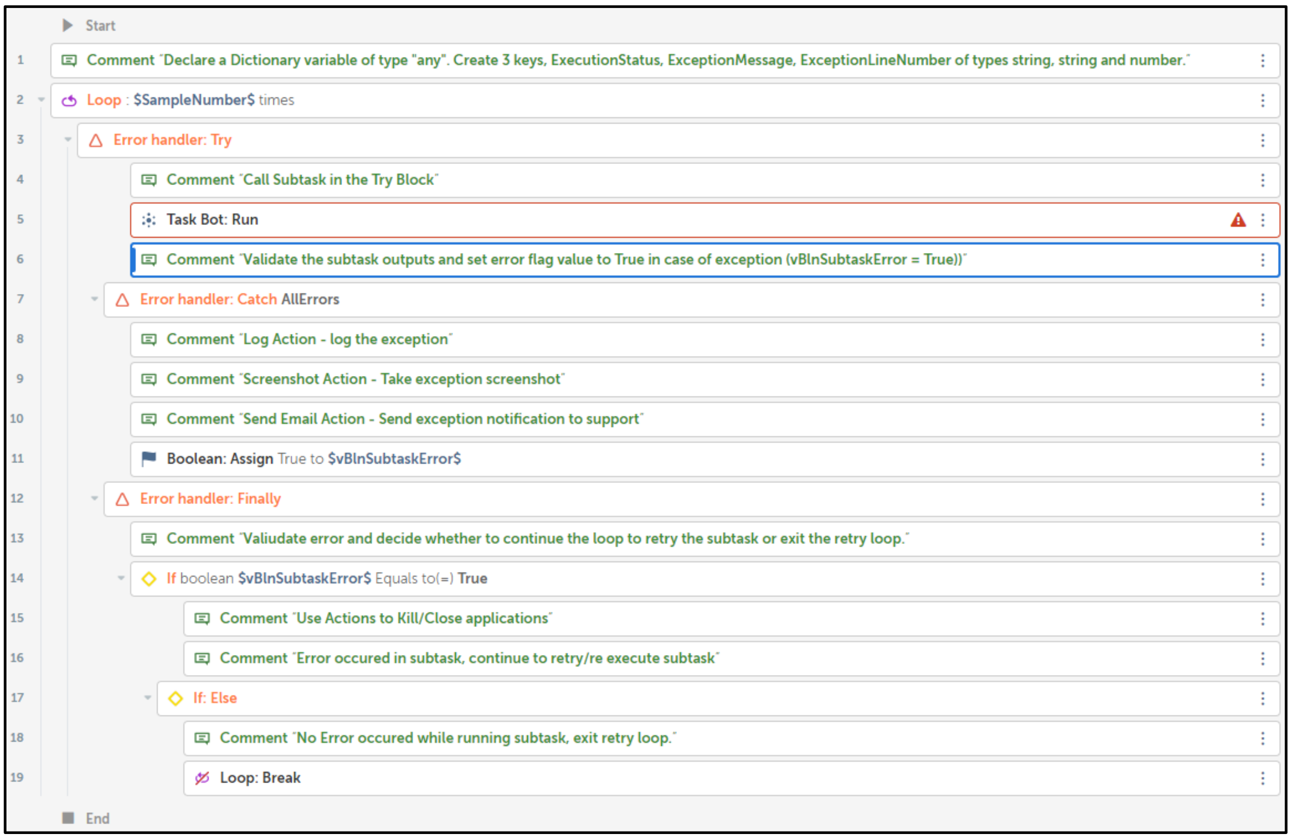
**a) No Error:**

The "Finally" block will validate the output from the subtask and exit the retry framework when the subtask runs successfully. (**Figure 11**)



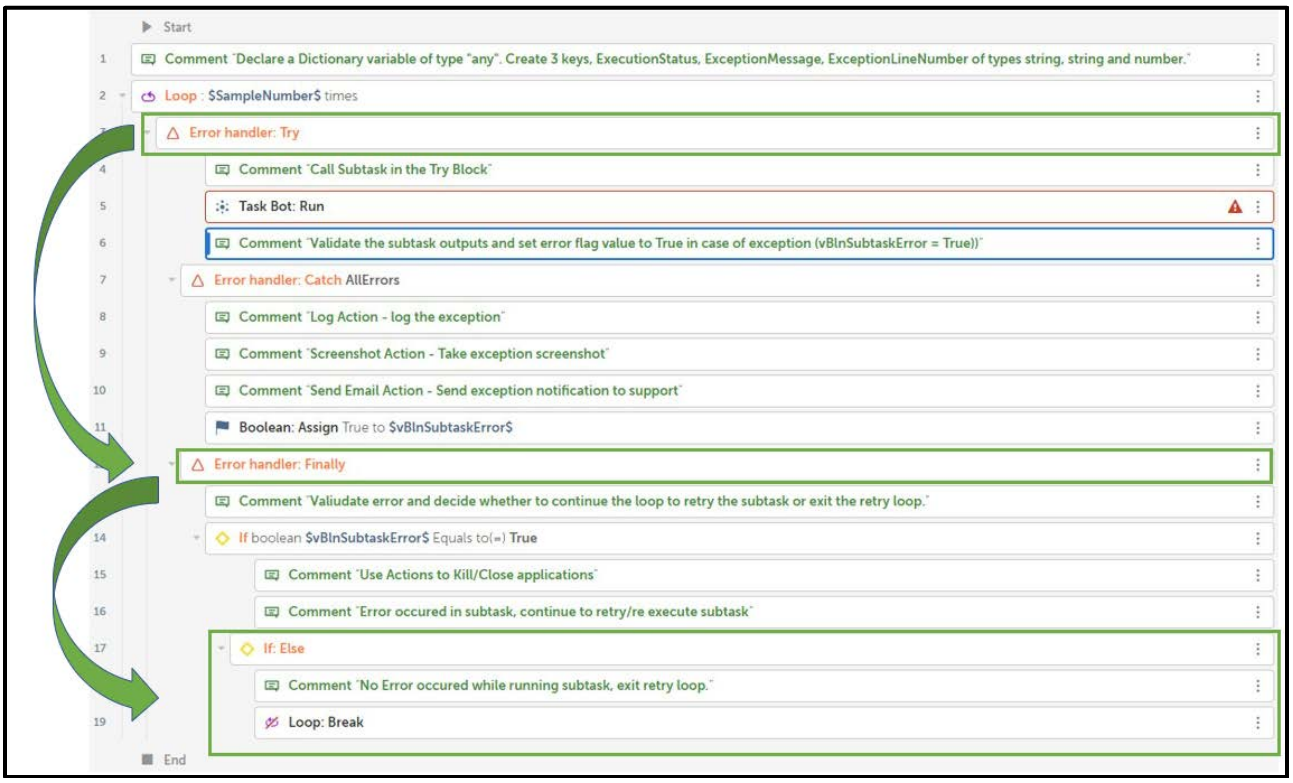
```
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus, ExceptionMessage, ExceptionLineNumber of types string, string and number."
Loop : $SampleNumber$ times
  Error handler: Try
      Comment "Call Subtask in the Try Block"
      Task Bot: Run
      Comment "Validate the subtask outputs and set error flag value to True in case of exception (vBlnSubtaskError = True)"
      If boolean $in_vBlnSubtaskerror$ Equals to(=) True
        Comment "Error occured in subtask, Throw exception to catch block."
        Error handler: Throw AllErrors
    Error handler: Catch AllErrors
      Comment "Log Action - log the exception"
      Comment "Screenshot Action - Take exception screenshot"
      Comment "Send Email Action - Send exception notification to support"
      Boolean: Assign True to $vBlnSubtaskError$
    Error handler: Finally
      Comment "Valiudate error and decide whether to continue the loop to retry the subtask or exit the retry loop."
      If boolean $vBlnSubtaskError$ Equals to(=) True
          Comment "Use Actions to Kill/Close applications"
          Comment "Error occured in subtask, continue to retry/re execute subtask"
        If: Else
          Comment "No Error occured while running subtask, exit retry loop."
          Loop: Break
```

**Figure 9.** A360 Retry Framework error syntax—Resilient task.



```
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus, ExceptionMessage, ExceptionLineNumber of types string, string and number."
Loop : $SampleNumber$ times
    Error handler: Try
        Comment "Call Subtask in the Try Block"
        Task Bot: Run
        Comment "Validate the subtask outputs and set error flag value to True in case of exception (vBlnSubtaskError = True))"
    Error handler: Catch AllErrors
        Comment "Log Action - log the exception"
        Comment "Screenshot Action - Take exception screenshot"
        Comment "Send Email Action - Send exception notification to support"
        Boolean: Assign True to $vBlnSubtaskError$
    Error handler: Finally
        Comment "Valiudate error and decide whether to continue the loop to retry the subtask or exit the retry loop."
        If boolean $vBlnSubtaskError$ Equals to(=) True
            Comment "Use Actions to Kill/Close applications"
            Comment "Error occured in subtask, continue to retry/re execute subtask"
        If: Else
            Comment "No Error occured while running subtask, exit retry loop."
            Loop: Break
End
```

Figure 10. A360 Retry Framework syntax—Exception bubbling task.



```
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus, ExceptionMessage, ExceptionLineNumber of types string, string and number."
Loop : $SampleNumber$ times
    Error handler: Try
        Comment "Call Subtask in the Try Block"
        Task Bot: Run
        Comment "Validate the subtask outputs and set error flag value to True in case of exception (vBlnSubtaskError = True))"
    Error handler: Catch AllErrors
        Comment "Log Action - log the exception"
        Comment "Screenshot Action - Take exception screenshot"
        Comment "Send Email Action - Send exception notification to support"
        Boolean: Assign True to $vBlnSubtaskError$
    Error handler: Finally
        Comment "Valiudate error and decide whether to continue the loop to retry the subtask or exit the retry loop."
        If boolean $vBlnSubtaskError$ Equals to(=) True
            Comment "Use Actions to Kill/Close applications"
            Comment "Error occured in subtask, continue to retry/re execute subtask"
        If: Else
            Comment "No Error occured while running subtask, exit retry loop."
            Loop: Break
End
```

Figure 11. A360 Retry Framework no error syntax—Exception bubbling task.

**b) Error:**

Catch block captures the error, notifies the relevant support group, closes/kills applications, and clears the exception. The “Finally” block validates the subtask outputs and reruns the subtask in the event of a system exception. (Figure 12)

**3) Advantages**

This method allows the subtask to run multiple times until the desired output is achieved; it improves bot efficiency and reduces manual support effort to re-trigger the process.

**4) Suggested Use**

This method is suitable for situations where the behavior of applications (web, windows, etc.) is unpredictable, and sometimes killing/closing the applications and relaunching them can fix any temporary issues/bugs and releases the system resources. For example, loading web browsers or SAP windows, etc.

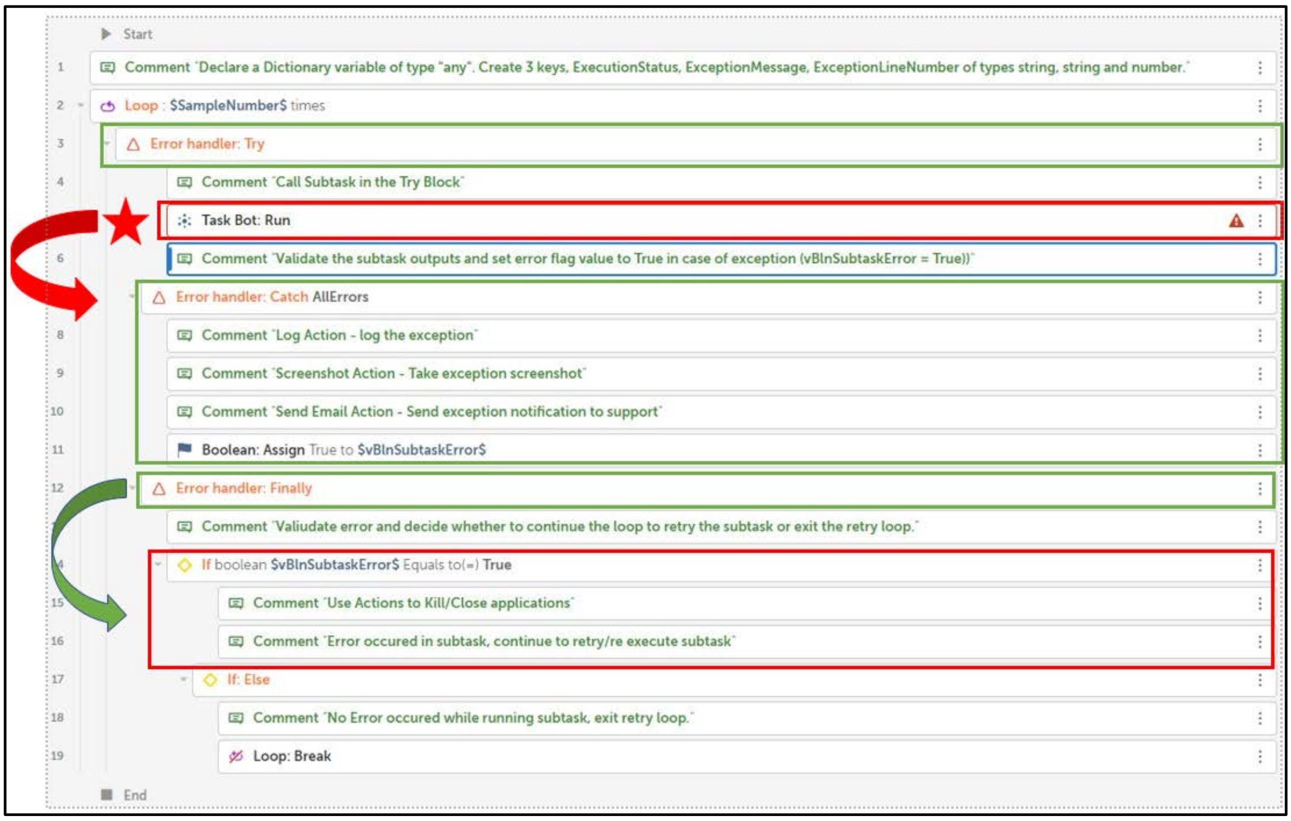
## 3. Case Studies

The effectiveness and feasibility of the proposed solution are verified with the following case studies.

### 3.1. Exception Bubbling Subtasks

Scenario: Automating data extraction and processing from a website.

Main Task: Extract Data from the website.



```
Start
Comment "Declare a Dictionary variable of type "any". Create 3 keys, ExecutionStatus, ExceptionMessage, ExceptionLineNumber of types string, string and number."
Loop : $SampleNumber$ times
    Error handler: Try
        Comment "Call Subtask in the Try Block"
        Task Bot: Run
        Comment "Validate the subtask outputs and set error flag value to True in case of exception (vBlnSubtaskError = True))"
    Error handler: Catch AllErrors
        Comment "Log Action - log the exception"
        Comment "Screenshot Action - Take exception screenshot"
        Comment "Send Email Action - Send exception notification to support"
        Boolean: Assign True to $vBlnSubtaskError$
    Error handler: Finally
        Comment "Valiudate error and decide whether to continue the loop to retry the subtask or exit the retry loop."
        If boolean $vBlnSubtaskError$ Equals to(=) True
            Comment "Use Actions to Kill/Close applications"
            Comment "Error occured in subtask, continue to retry/re execute subtask"
        If: Else
            Comment "No Error occured while running subtask, exit retry loop."
            Loop: Break
End
```

Figure 12. A360 Retry Framework error syntax—Exception bubbling task.

Subtasks:

• Open website: Launches a website URL and login to the website.

• Extract Data: Extract relevant data from the website.

• Process Data: Process the extracted data.

In this scenario, the successful execution of each subtask depends on the performance of the previous subtask integrated into the main task. If the "Extract Data" subtask fails, the "Process Data" subtask cannot run, and if the "Open Website" subtask fails, the "Extract Data" subtask cannot run either. Consequently, it is unnecessary to proceed with the next task if the previous task has failed. To handle this, each subtask will be implemented as an exception bubbling task, where the code is encapsulated within a Try-Catch-Finally block. If an exception occurs, it will be thrown to the main task, allowing it to catch the exception, notify the relevant support group, and terminate the execution of the process.

### Advantages

• The Exception Bubbling Subtasks approach avoids the need of developing and executing redundant validation code in the subtask and main task. It greatly improves the processing time of the automation.

• Exception information can be logged, captured, or used for further validation and actions in the main task.

• It improves the overall efficiency and resiliency of the automation solution.

## 3.2. Resilient Subtasks

Scenario: Automating data extraction and processing from a website for multiple transactions.

Main Task: Process Data for all transactions.

Subtasks:

• Open website: Launches a website URL and login to the website.

• Extract Data: Extract relevant data from the website.

• Process Data: Process the extracted data.

In this scenario, every transaction operates independently and follows the same set of processing steps. If the "Open website" task encounters a failure, it only affects the current transaction. This exception does not require stopping the processing of other transactions. To handle this, each subtask will be implemented as a resilient task, encapsulating the code within a Try-Catch-Finally block. If an exception occurs, it will be caught and handled within the subtask, allowing the main task to validate the exception from the subtask, mark the current transaction as failed, skip it, and proceed with processing the next transaction.

### Advantages

• The implementation of Resilient Subtasks enables the main task to validate the outputs of each subtask, thereby reducing the impact of exceptions on other transactions. This approach ensures that all transactions are processed effectively.

- Exception information can be logged, captured, or used for further validation and actions in the subtask task.
- It improves the overall efficiency and resiliency of the automation solution.

### 3.3. Retry Framework

Scenario: Automating data extraction and processing from a website.

Main Task: Process Data from a website.

Subtasks:

1) Retry Framework for Open Website: Open Website subtask is integrated/called in the retry framework task.

a) Open website: Launches a website URL and login to the website.

2) Retry Framework for Extract Data: Extract Data subtask is integrated/called in the retry framework task.

a) Extract Data: Extract relevant data from the website.

3) Retry Framework for Process Data: Process Data subtask is integrated/called in the retry framework task.

a) Process Data: Process the extracted data.

In this scenario, each subtask is coupled with a retry framework task that attempts to execute the subtask a specified number of times until it succeeds. If the "Open Website" task fails to launch a website due to browser or internet connectivity issues, the retry framework acts by closing or terminating the browser session. It then triggers the subtask again, enabling it to relaunch the website and perform the necessary login. Similarly, if the "Extract Data" or "Process Data" subtasks encounter difficulties in identifying an element within an application, the retry framework task clears the application sessions. This allows the subtasks to retry the extraction or processing of the data.

#### Advantages

- By incorporating a Retry Framework into the automation process, the overall resilience is enhanced, ensuring robust and error-proof execution. This framework offers the capability to resolve temporary system bugs encountered during the execution of high runtime processes, thus improving the reliability of the automation solution.
- It improves the overall efficiency and resiliency of the automation solution.

## 4. Extensibility of the Solution

The extensibility of A360 subtasks allows for the development of diverse application functionalities that can be reused across multiple automation processes. A360 subtasks allow developers to quickly develop automation solutions by leveraging reusable assets and simplifying maintenance efforts. Below are some of the application functions that can be built by using A360 subtasks [2] [15] [16].

### 4.1. Web Applications

A360 subtasks can be extended to perform various web operations, such as website login, data entry, and data extraction.

### 4.2. Windows Applications

A360 subtasks can be extended to handle a wide range of Windows operations, including launching applications, managing files and folders, validating data, performing data entry, and extracting information.

### 4.3. Database Systems

A360 subtasks can be extended to perform diverse database operations, such as data validation, data extraction, and data synchronization.

### 4.4. Legacy Systems

A360 subtasks can be extended to execute various operations on legacy systems, such as simulating mouse clicks, emulating keystrokes, and performing other necessary functions.

### 4.5. API Integrations

A360 subtasks can be extended to execute various API operations across different systems, including actions like GET, PUT, POST, DELETE, and other required functionalities for seamless integration.

## 5. Contribution and Innovation

The research work presented in the article introduces innovative methods and frameworks to enhance the efficiency of automation solutions developed using Automation 360. The main contributions and innovations of this research work can be summarized as follows.

### 5.1. Introduction of A360 Subtasks as Comparable Concepts to Functions in Programming

The article explores the potential of using subtasks in Automation 360 as reusable and maintainable tasks, like the concept of functions in programming. This approach allows users, regardless of their programming experience, to efficiently develop customized automation solutions. By leveraging subtasks, users can create high-quality automation solutions that are easier to maintain and improve overall code quality [4] [10] [11] [14].

### 5.2. Introduction of the Retry Framework

The article presents a retry framework that automatically retries subtasks in the event of system or unknown exceptions. This framework enhances efficiency by reducing the manual efforts required to retrigger bots and improving the resiliency of automation solutions. The retry framework allows for the repeated execution of subtasks until the desired output is achieved, thereby increasing the chances of successful completion, and reducing the need for manual intervention.

### 5.3. Assistance to Professional and Citizen Developers

The A360 Subtask and Retry Framework templates are designed to assist both professional and citizen developers. These templates provide valuable resources for developers of all skill levels to create reliable, reusable, and efficient automation solutions. By utilizing these templates, developers can streamline the development process, ensure consistent code quality, and simplify maintenance and support efforts.

Overall, the research work contributes to the field of automation solutions by introducing innovative methods and frameworks that improve the efficiency, resiliency, and maintainability of automation solutions developed using Automation 360.

## 6. Advantages of the Solution

The proposed solution in the article offers several advantages in terms of performance, efficiency, and other key aspects. These advantages include [15] [16].

### 6.1. Reusability and Maintainability

By using A360 subtasks as reusable and maintainable functions, developers can create automation solutions more efficiently. Subtasks can be developed once and reused in multiple processes or bots, reducing duplication of effort, and enhancing code maintainability. This approach improves development productivity and allows for consistent code implementation across different automation projects.

### 6.2. Code Quality and Consistency

The use of subtasks as comparable concepts to functions in programming promotes high-quality and consistent code development. Developers can define specific tasks within subtasks, ensuring modular and well-structured code. This approach improves code readability, reduces errors, and enhances overall code quality.

### 6.3. Efficiency and Resiliency

The retry framework introduced in the paper improves the efficiency and resiliency of automation solutions. In the event of system or unknown exceptions, the retry framework automatically retries subtasks until the desired output is achieved. This framework ensures robust and error-proof execution of the automation solution by resolving temporary systems bugs encountered during the execution of high runtime processes. This automation reduces the need for manual intervention and minimizes the effort required to retrigger bots, resulting in improved efficiency and resource utilization.

### 6.4. Error Handling and Exception Management

The solution provides effective error handling and exception management mechanisms. The resilient subtasks handle exceptions within the subtask and exception bubbling subtasks throw exceptions to the caller/main task. Both solutions allow for better error tracking, logging, and notification. By capturing and handling exceptions appropriately, the solution enhances the reliability and stability of automation processes.

## 7. Conclusion

The A360 Subtask and Retry Framework templates provide a valuable resource for developers of all skill levels to create reliable, reusable, and efficient automation solutions. These templates not only ensure high-quality, consistent code but also facilitate maintenance and support efforts. The Retry framework is a powerful tool that can enhance the efficiency and resiliency of bots by retrying subtasks in the event of a system or unknown exception. This framework makes automation solutions error-proof and reduces the need for manual intervention for retriggering the automation process. Overall, utilizing these templates can greatly improve the effectiveness of automation solutions and streamline development processes.

## Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

## References

[1] Automation Anywhere (n.d.) Automation 360. Automationanywhere.com. 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] Anagnoste, S. (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] Automation Anywhere (n.d.) No-Code Is an Approach to Designing and Using Applications That Doesn't Require Any Coding or Knowledge of Programming Languages. Automationanywhere.com. https://www.automationanywhere.com/rpa/no-code-automation

[5] Make Use of (n.d.) What Is a Function in Programming? Makeuseof.com. https://www.makeuseof.com/what-is-a-function-programming/

[6] Make Use of (n.d.) Why Programming Languages Can't Exist without Functions.

Makeuseof.com.
https://www.makeuseof.com/tag/programming-languages-need-functions/?newsletter_popup=1

[7] Automation Anywhere (n.d.) Building Reusable Bots. Automationanywhere.com. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/developer-recommendations/cloud-build-reusable-bots.html

[8] Automation Anywhere (n.d.) Managing Reusable Assets—Start Phase. Automationanywhere.com.
https://community.automationanywhere.com/developers-blog-85009/managing-reusable-assets-start-phase-85218

[9] Automation Anywhere (n.d.) Using the Run Action. Automationanywhere.com. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/using-the-run-action.html

[10] Automation Anywhere (n.d.) Error Handler Package. Automationanywhere.com. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/cloud-error-handling-command.html

[11] Automation Anywhere (n.d.) Try Action in Error Handler. Automationanywhere.com. https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/error-handler-try.html

[12] Automation Anywhere (n.d.) Catch Action in Error Handler. Automationanywhere.com.
https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/error-handler-catch.html

[13] Automation Anywhere (n.d.) Finally Action in Error Handler. Automationanywhere.com.
https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/error-handler-finally.html

[14] Automation Anywhere (n.d.) Throw Action in Error Handler. Automationanywhere.com.
https://docs.automationanywhere.com/bundle/enterprise-v2019/page/enterprise-cloud/topics/aae-client/bot-creator/commands/error-handler-throw.html

[15] Worksoft (n.d.) Solving Bot Fragility with Change Resilient RPA. Worksoft.com. https://www.worksoft.com/corporate-blog/solving-bot-fragility-with-change-resilient-rpa

[16] DeepSource (n.d.) What Happens in the Absence of Exception Handling? Deepsource.com.
https://deepsource.com/glossary/exception-handling#:~:text=If%20an%20exception%20is%20thrown,unless%20a%20function%20handles%20it