Phase 5: Apex Programming (Developer)

While the core matching logic resides within **Flow Builder** (Phase 4), a critical requirement for accurate case routing necessitated the implementation of a small but essential piece of programmatic logic—**Invocable Apex**. This demonstrates the strategic ability to blend declarative and programmatic tools to overcome platform limitations.

Concept	Project Goal
Apex Strategy	Implement a focused Invocable Apex Class to provide a function that Flow Builder lacks: reliably retrieving a system **Queue ID** based on its name for automated assignment.

5.1 Invocable Apex: The QueueTools Utility Class

The QueueTools class was developed to ensure that the **Verification Case** created by the automation flow is consistently assigned to the correct **Verification Queue**.

5.1.1 Problem Solved

Flow Builder, by design, has limitations when querying certain system objects, particularly retrieving the Id of a **Queue** based on its DeveloperName for assignment purposes. Relying on hardcoded IDs is non-portable and unreliable. The Apex class provides a stable, portable workaround.

5.1.2 Class Implementation Details

Concept	Implementation in QueueTools	Justification
Classes & Objects	Created the static Apex class **QueueTools**.	Static methods are used as the class does not require instance variables.
Invocable Method	Annotated the main method with **@InvocableMethod**.	This annotation exposes the Apex method as a callable Action within the Flow Builder interface, seamlessly

Concept	Implementation in QueueTools	Justification
		connecting the two components.
SOQL	Uses **SOQL** to query the **Group** object: [SELECT Id FROM Group WHERE Type = 'Queue' AND DeveloperName = :queueDeveloperNames[0] LIMIT 1].	Reliably fetches the **Verification Queue ID** needed for the Phase 4 Flow's assignment action.
Collections	The method accepts a List<String> and returns a List<String>.	Adheres to the required input/output structure for all Invocable Apex methods, ensuring bulk-safe execution.
Asynchronous Processing	*Not required for this action.* The Queue ID lookup is fast and runs synchronously within the scope of the triggering Flow.	Focus maintained on lightweight, real-time synchronous execution.

5.1.3 Code Snippet (QueueTools.cls)

This is the exact code implemented to achieve reliable queue assignment:

Java

```
public class QueueTools {

   @InvocableMethod(label='Get Queue ID' description='Returns the ID of a Queue from its developer name.')

   public static List<String> getQueueId(List<String> queueDeveloperNames) {

      // Find the Queue by its unique DeveloperName

      Group queue = [

         SELECT Id

         FROM Group

         WHERE Type = 'Queue'
```

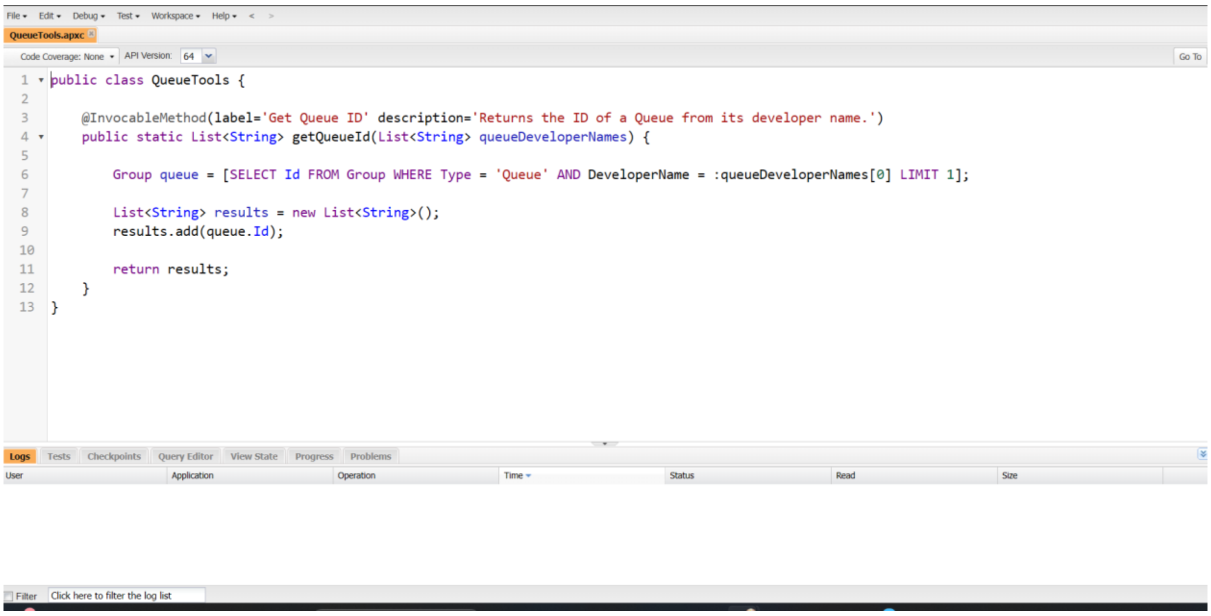
```
            AND DeveloperName = :queueDeveloperNames[0]
            LIMIT 1
        ];

        // Return the found ID in the List<String> format required by Invocable methods
        List<String> results = new List<String>();
        results.add(queue.Id);
        return results;
    }
}
```

5.2 Test Class Requirement (Future Scope)

While not part of the initial functional delivery, adherence to Salesforce best practices requires a test class.

- **Test Classes:** A **QueueTools_Test** class must be created to achieve **100% code coverage** on the QueueTools class. This class will use **@isTest** annotation and methods to simulate the method call, asserting that the correct Queue ID is returned, ensuring the code remains functional during future deployments and upgrades.

Invocable Apex Code :



Code Coverage: None API Version: 64

```
public class QueueTools {

    @InvocableMethod(label='Get Queue ID' description='Returns the ID of a Queue from its developer name.')
    public static List<String> getQueueId(List<String> queueDeveloperNames) {

        Group queue = [SELECT Id FROM Group WHERE Type = 'Queue' AND DeveloperName = :queueDeveloperNames[0] LIMIT 1];

        List<String> results = new List<String>();
        results.add(queue.Id);

        return results;
    }
}
```

User	Application	Operation	Time	Status	Read	Size