Strategies for creating your own conversational test assistant with Robot Framework and other tools

RoboCon 2020 - Workshop

Presented by Joshua Gorospe

Before we begin...

Just want to again, thank the following people and groups.

- Pekka Klarck
- Ed Manlove
- Antti Karjalainen
- James Bach
- Jonathon Bach
- Michael Bolton
- Cem Kaner
- Kristian Karl
- Mikko Korpela
- Bryan Oakley
- Shiva Prasad Adirala
- Henry Hinnefeld
- Louis Grenard
- The Docker development team
- The Graphwalker development team
- The entire Robot Framework community and its contributors.

Introduction

Some information about me...

- I work at Fuzz Productions, https://fuzzproductions.com/
- I am a senior test engineer and test lead working on an internal product team called "Koala".
- Previously I was working at https://www.intersection.com/
    - Worked in the Intersection Connected Communities team.
- I enjoy experimenting with test tools.
- I like considering myself as simply a tester, I am proud of it.

What value would a Robot Framework conversational test assistant provide?

Take Robot Framework's capabilities...

Combined with capabilities similar to Amazon Echo's Alexa...

?

Let’s consider a few useful possibilities and ideas…

- Large collections of your existing Robot Framework scripts could become more accessible to your entire team.
    - The conversational test assistant and your collections of scripts could constantly be in an always-available, always-ready-to-use state.

- The ease of use of a conversational test assistant might be able to lower technical barriers for non-technical people.
    - For example, testers might be out-of-office and team members may need to run a few quick automated checks.
    - Non-testers (technical or non-technical) on teams may find the assistant less intimidating than setting up and running Robot Framework themselves.

- Just uttering a word or simply typing short text commands to the conversational test assistant could trigger any Robot Framework RPA task or automated check.
    - Combine vocal or text commands with various capabilities of the Robot Framework tool ecosystem and libraries…
    - Combine vocal or text commands with locally or remotely running Robot Framework processes (in Docker Containers)…

There are probably several more interesting possibilities, but let’s move on…

That all sounds great in theory, but how feasible is this in 2020?...

As it turns out, it is very feasible now. Especially with...

Leon AI

Open-Source Personal Assistant.

Leon's author Louis Grenard was kind enough to provide:

- A re-usable core architecture
- A generic packages/modules structure, that has a similar purpose as Amazon's Alexa skills
- An offline mode that allows you to communicate with leon-ai (text or voice) without any third party services
- An open source tool that anyone can contribute to

Before we move on...

- **FYI to Windows users.** All of the examples in this workshop were created to run in a Linux and MacOS environment.
- Here is a possible option that may help you, **but I have not tested it myself**...
    - VirtualBox (I recommend you run Ubuntu on it)
    - Windows Subsystem For Linux (Here are some guides...)
        - https://docs.microsoft.com/en-us/windows/wsl/install-win10
        - https://www.windowscentral.com/install-windows-subsystem-linux-windows-10
        - https://itsfoss.com/install-bash-on-windows/
    - After you set up your working Linux virtual environment, please install the following...
        - Git
        - Docker and docker-compose
        - Python 3
        - Pipenv
        - NodeJS

Before we move on...

- **There are known challenges and issues with leon-ai's offline speech recognition. Typing text commands through the leon-ai UI works 100% perfectly fine.**
- While using leon-ai's offline speech-to-text engine (DeepSpeech), you may notice that sometimes it makes mistakes interpreting what you are saying or it just hangs and does nothing.
  - It uses Mozilla DeepSpeech, a TensorFlow implementation of Baidu's DeepSpeech architecture.
- Here are some hypothetical assumptions I've made based on some quick research.
  - Tensorflow hardware requirements favor NVidia GPU technologies more. If you use a non-NVidia GPU (no CUDA cores etc.) it will be underutilized. For example, only using onboard graphics on a laptop may affect performance.
    - https://github.com/mozilla/DeepSpeech/releases
    - https://github.com/mozilla/DeepSpeech/issues/2128
    - https://discourse.mozilla.org/t/doesn-t-use-gpu-while-training-but-during-recognition-it-uses-one/27965/20
    - https://towardsdatascience.com/tensorflow-speech-recognition-challenge-solution-outline-9c42dbd219c9
  - DeepSpeech Speech-to-Text Engine itself has been reported having accuracy issues.
    - https://discourse.mozilla.org/t/improving-the-accuracy-of-the-speech-recognition-with-newer-iterations-of-deepspeech-or-other-competing-techniques-in-deep-speech-recognition/34006
    - https://discourse.mozilla.org/t/terrible-accuracy/46823
    - https://github.com/tensorflow/models/issues/7567
  - If you have an unreliable or poor quality microphone it may affect performance.
    - https://github.com/mozilla/DeepSpeech/issues/1156#issuecomment-434872303

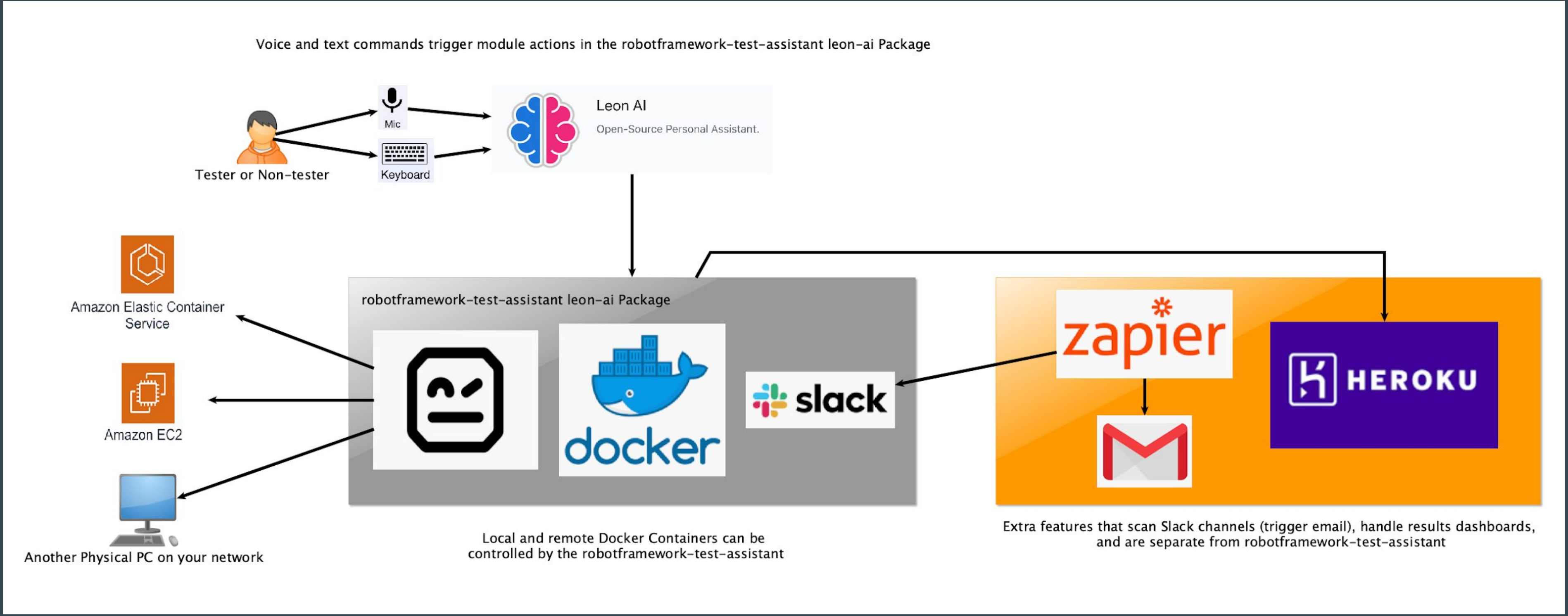
Workshop Outline

I will be running a lot of commands and showing the code. This workshop will be mostly hands-on, and is organized into the following parts...

- Part One - Quick walkthrough of the robotframework-test-assistant and leon-ai
- Part Two - Hands-on Walkthrough of the Basic Stand-alone Commands (12 commands)
- Short Break: For answering questions etc.
- Part Three - Hands-on Walkthrough of the Docker Commands from an existing GitHub project (12 commands)
- Short Break: For answering questions etc.
- Part Four - Hands-on Walkthrough of the Custom Automation Runners and Advanced Stand-alone Commands (32 commands)

Part One - Quick walkthrough of the robotframework-test-assistant and leon-ai

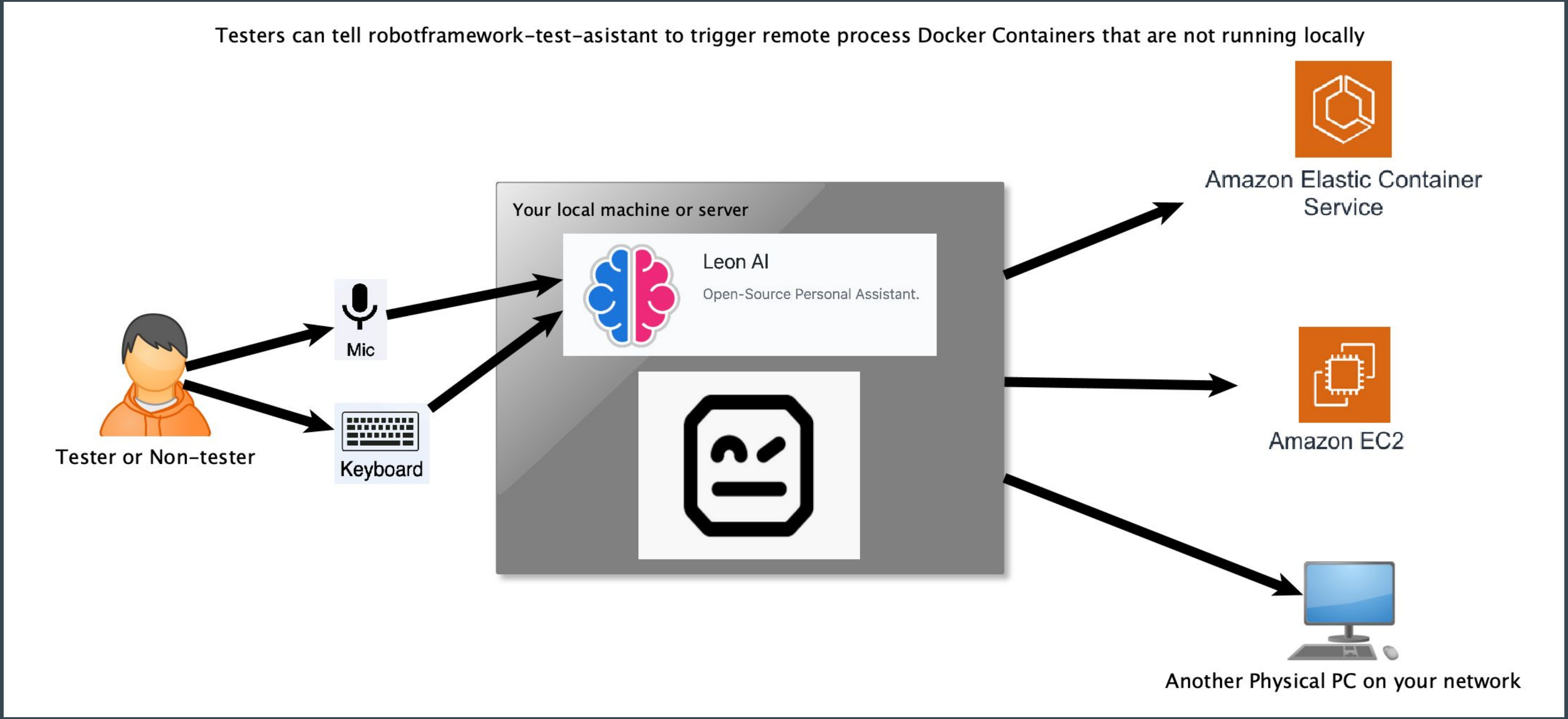
The conversational test assistant we will be exploring today is called robotframework-test-assistant. The following diagrams show the main components and special capabilities of this assistant, and a high-level explanation of how leon-ai is interacting with Robot Framework and other tools.



Voice and text commands trigger module actions in the robotframework-test-assistant leon-ai Package
Tester or Non-tester
Mic
Keyboard
Leon AI
Open-Source Personal Assistant.
Amazon Elastic Container Service
Amazon EC2
Another Physical PC on your network
robotframework-test-assistant leon-ai Package
docker
slack
zapier
HEROKU
Local and remote Docker Containers can be controlled by the robotframework-test-assistant
Extra features that scan Slack channels (trigger email), handle results dashboards, and are separate from robotframework-test-assistant

Part One - Quick walkthrough of the robotframework-test-assistant and leon-ai

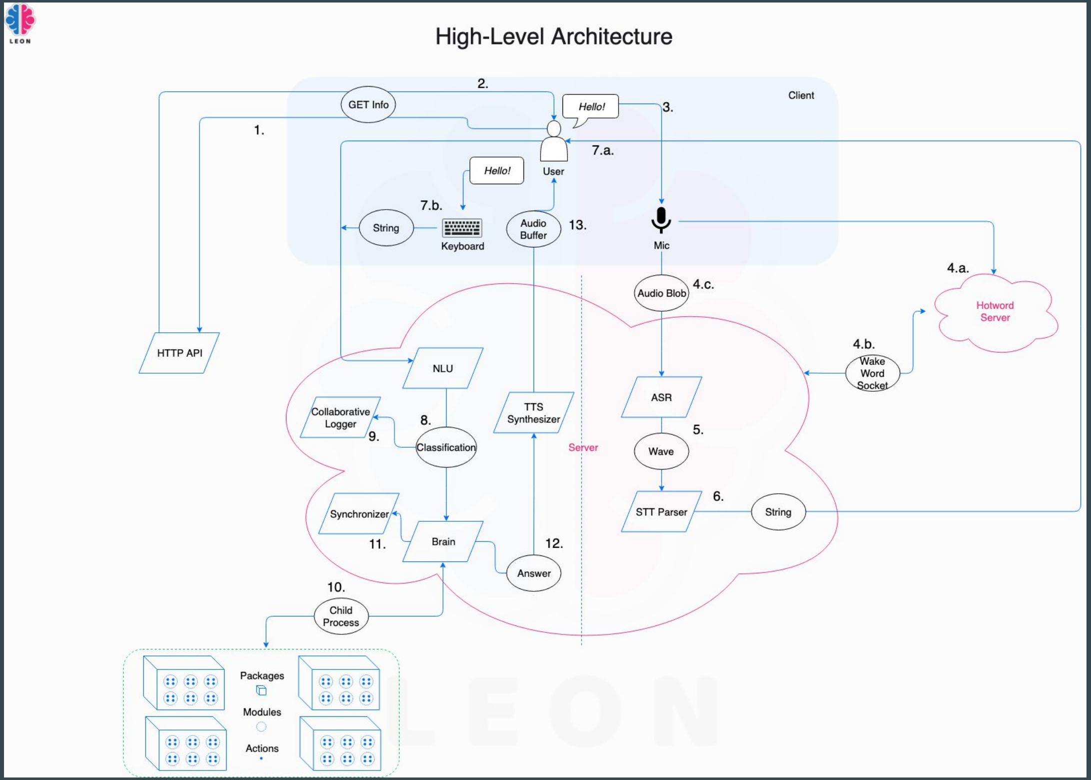
This diagram shows in more detail how the conversational test assistant can trigger remote processes through a webhook running in a Docker Container on Amazon EC2, ECS, or another physical machine on the same network as the assistant. This is the webhook tool that is being triggered from the examples shown in this workshop → https://github.com/adnanh/webhook


Testers can tell robotframework-test-asistant to trigger remote process Docker Containers that are not running locally
Tester or Non-tester
Mic
Keyboard
Your local machine or server
Leon AI
Open-Source Personal Assistant.
Amazon Elastic Container Service
Amazon EC2
Another Physical PC on your network

Part One - Quick walkthrough of the robotframework-test-assistant and leon-ai

This High-Level Architecture Schema diagram of leon-ai comes from https://docs.getleon.ai/architecture.html#high-level-architecture-schema, and it shows...

- How a user interacts with the Hotword Detection Server + Automatic Speech Recognition (ASR), when he says a command through a microphone.
- How a user interacts with the Natural Language Understanding (NLU) + Classification, when he types text commands through a keyboard.
- How the text-to-speech (TTS) and speech-to-text (STT) events flow through the leon-ai Server.
- How the “Brain” of the leon-ai Server decides which Package + Module + Actions to use based on the commands given by a user.



Part One - Quick walkthrough of the robotframework-test-assistant and leon-ai

This Scenario comes from https://docs.getleon.ai/architecture.html#scenario, and it demonstrates what happens when a user types or says “Hello” to leon-ai…

- Client (web app, etc.) makes an HTTP request to GET some information about Leon.
- HTTP API responds information to client.
- User talks with their microphone.
  - a. If hotword server is launched, Leon listens (offline) if user is calling him by saying “Leon”.
  - b. If Leon understands user is calling him, Leon emits a message to the main server via a WebSocket. Now Leon is listening (offline) to user.
  - c. User said “Hello!” to Leon, client transforms the audio input to an audio blob.
- ASR transforms audio blob to a wave file.
- STT parser transforms wave file to string “Hello”. ***(This workshop uses this for STT →*** ***https://docs.getleon.ai/offline.html#deepspeech-stt)***
  - a. User receives string and string is forwarded to NLU.
  - b. Or user type “Hello!” with their keyboard (and ignores steps 1. to 7.a.). “Hello!” string is forwarded to NLU.
- NLU classifies string and pick up classification.
- If collaborative logger is enabled, classification is sent to collaborative logger.
- Brain creates a child process and executes the chosen module.
- If synchronizer is enabled and module has this option, it synchronizes content.
- Brain creates an answer and forwards it to TTS synthesizer. ***(This workshop uses this for TTS →*** ***https://docs.getleon.ai/offline.html#flite-tts)***
- TTS synthesizer transforms text answer (and send it to user as text) to audio buffer which is played by client.

Part Two - Hands-on Walkthrough of the Basic Stand-alone Commands

I will be demonstrating the following Basic Stand-alone Commands...

- "help" <-- This on-demand (no time delay) command will immediately display a GitHub link to this documentation.
- "clean" <-- This on-demand (no time delay) command will immediately clean up all results and logs.
- "check one" <-- This on-demand (no time delay) command will immediately run a single automated check called Check One, and quickly display the results file.
- "check two" <-- This on-demand (no time delay) command will immediately run a single automated check called Check Two, and quickly display the results file.
- "check three" <-- This on-demand (no time delay) command will immediately run a single automated check called Check Three, and quickly display the results file.
- "check four" <-- This on-demand (no time delay) command will immediately run a single automated check called Check Four, and quickly display the results file.
- "group one" <-- This on-demand (no time delay) command will immediately run parallel automated checks called Group One, and quickly display the results file.
- "group two" <-- This on-demand (no time delay) command will immediately run parallel automated checks called Group Two, and quickly display the results file.
- "group three" <-- This on-demand (no time delay) command will immediately run parallel automated checks called Group Three, and quickly display the results file.
- "group four" <-- This on-demand (no time delay) command will immediately run parallel automated checks called Group Four, and quickly display the results file.
- "slack notify" <-- This on-demand (no time delay) command will immediately send out all console logs as a Slack notification to a specific channel.
- "desktop browsers" <-- This on-demand (no time delay) command will immediately run parallel automated checks on multiple browsers using PaBot, and quickly display the results file.

I can also assign new commands to existing actions...

- For example, typing this into the UI → "group of checks for release 123"
- This command will also trigger ""group one""

Questions about Part One or Part Two?

Part Three - Hands-on Walkthrough of the Docker Commands from an existing GitHub project

I will be demonstrating the following Docker Commands from an existing GitHub project...

- "docker clean" <-- This command NEEDS TO RUN BEFORE OTHERS in this group. This will stop and clean up all Docker Containers running on your local machine.
- "docker build" <-- This command NEEDS TO RUN BEFORE OTHERS in this group, and NEEDS TO RUN AFTER the "docker clean" command is finished running. This will set up all of the necessary Docker Containers for this group of commands.
- "response check" <-- This command can RUN AFTER the "docker build" command is finished running. It will run API response checks using Robot Framework in a Docker Container.
- "random order response checks" <-- This command can RUN AFTER the "docker build" command is finished running. It will run API response checks in a random order using Robot Framework in a Docker Container.
- "run graphwalker checks" <-- This command can RUN AFTER the "docker build" command is finished running. It will run API response checks using a Model-based Testing tool called GraphWalker combined with Robot Framework in a Docker Container.
- "display graph walker path" <-- This command can RUN AFTER the "docker build" command is finished running. It will display the combined results of the "run graphwalker checks" command.
- "run graph walker again" <-- This command can RUN AFTER the "docker build" command is finished running. It will re-run the GraphWalker path that was generated when the "run graphwalker checks" command last used.
- "remote docker one" <-- This command can RUN AFTER the "docker build" command is finished running. This will setup a Docker Container running Robot Framework that can be used locally or remotely (for example on AWS ECS, EC2, or another computer on the same network), and it can be triggered by a webhook.
- "remote docker two" <-- This command can RUN AFTER the "docker build" command is finished running. This will setup a Docker Container running Robot Framework that can be used locally or remotely (for example on AWS ECS, EC2, or another computer on the same network), and it can be triggered by a webhook.
- "trigger remote one" <-- This command can RUN AFTER the "remote docker one" command is finished running. This command will trigger the webhook for the Docker Container that was setup after running the "remote docker one" command.
- "trigger remote two" <-- This command can RUN AFTER the "remote docker one" command is finished running. This command will trigger the webhook for the Docker Container that was setup after running the "remote docker two" command.
- "trigger parallel docker" <-- This command can RUN AFTER BOTH the "remote docker one" and "remote docker two" commands are finished running. This command will trigger the both webhooks for the Docker Containers that were setup after running the "remote docker one" and "remote docker two" commands. This will result in both remote containers to run in parallel.

Questions about Part Three?

Part Four - Hands-on Walkthrough of the Custom Automation Runners and Advanced Stand-alone Commands

I will be demonstrating the following Custom Automation Runners and Advanced Stand-alone Commands...

Custom Runner One...

- "set up runner one" <-- This command NEEDS TO RUN BEFORE OTHERS in this group. This will set up a custom serial automation runner that allows the user to define which checks will run using a CustomSerialAutomationRunnerFile.csv file.
- "set check one" <-- This command can RUN AFTER the "set up runner one" command is finished running. It will set the custom serial automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Serial-Runner.robot file. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "set check two" <-- This command can RUN AFTER the "set up runner one" command is finished running. It will set the custom serial automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Serial-Runner.robot file. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "set check three" <-- This command can RUN AFTER the "set up runner one" command is finished running. It will set the custom serial automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Serial-Runner.robot file. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "set check four" <-- This command can RUN AFTER the "set up runner one" command is finished running. It will set the custom serial automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Serial-Runner.robot file. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "custom runner one" <-- This on-demand (no time delay) command NEEDS TO RUN AFTER the "set up runner one" and the "set xxxx xxxx etc." command(s) finished running. It will immediately run the series of checks from this group of related commands in an order defined by you.
- "display runner one" <-- This command NEEDS TO RUN AFTER the "custom runner one" command is finished running. It will display the combined results of the "custom runner one" command.

Part Four - Hands-on Walkthrough of the Custom Automation Runners and Advanced Stand-alone Commands

I will be demonstrating the following Custom Automation Runners and Advanced Stand-alone Commands...

Custom Runner Two...

- "set up runner two" <-- This command NEEDS TO RUN BEFORE OTHERS in this group. This will set up a custom parallel automation runner that allows the user to define which checks will run using a CustomParallelAutomationRunnerFile.csv file.
- "set group one" <-- This command can RUN AFTER the "set up runner two" command is finished running. It will set the custom parallel automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Parallel-Runner1.robot and Leon-Robot-Framework-Customizable-Parallel-Runner2.robot files. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "set group two" <-- This command can RUN AFTER the "set up runner two" command is finished running. It will set the custom parallel automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Parallel-Runner1.robot and Leon-Robot-Framework-Customizable-Parallel-Runner2.robot files. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "set group three" <-- This command can RUN AFTER the "set up runner two" command is finished running. It will set the custom parallel automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Parallel-Runner1.robot and Leon-Robot-Framework-Customizable-Parallel-Runner2.robot files. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "set group four" <-- This command can RUN AFTER the "set up runner two" command is finished running. It will set the custom parallel automation runner to run this specifically tagged check in the Leon-Robot-Framework-Customizable-Parallel-Runner1.robot and Leon-Robot-Framework-Customizable-Parallel-Runner2.robot files. This check works the same way as the similar command found in the >> Stand-alone Commands <<.
- "custom runner two" <-- This on-demand (no time delay) command NEEDS TO RUN AFTER the "set up runner two" and the "set xxxx xxxx etc." command(s) finished running. It will immediately run the series of checks from this group of related commands in an order defined by you.
- "display runner two" <-- This command NEEDS TO RUN AFTER the "custom runner one" command is finished running. It will display the combined results of the "custom runner two" command.

Part Four - Hands-on Walkthrough of the Custom Automation Runners and Advanced Stand-alone Commands

I will be demonstrating the following Custom Automation Runners and Advanced Stand-alone Commands...

RPA Custom Tasks And Suites Runner...

- "set up custom task runner" <-- This command NEEDS TO RUN BEFORE OTHERS in this group. This will set up an RPA custom task automation runner that allows the user to define which RPA tasks will run using a CustomizedTasksAndRobotFrameworkSuitesOrderSequence.csv file.
- "set docker clean" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set docker build" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set remote docker one" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set remote docker two" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set trigger remote one" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set trigger remote two" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set trigger parallel docker" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the [Docker Container Examples].
- "set bug risk predictor" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This will run an RPA task that will trigger a Docker Container that predicts bug risk in a GitHub and sends out alerts.
- "set slack" <-- This command can RUN AFTER the "set up custom task" command is finished running. It will set the RPA custom task automation runner to run this specifically tagged RPA task in the Leon-Robot-Framework-Customizable-RPA-Task-Runner.robot file. This RPA task works the same way as the similar command found in the >> Stand-alone Commands <<.
- "custom task runner" <-- This on-demand (no time delay) command NEEDS TO RUN AFTER the "set up custom task" and the "set xxxx xxxx etc." command(s) finished running. It will immediately run the parallel and serial tasks from this group of related commands in an order defined by you.
- "display rpa results" <-- This command NEEDS TO RUN AFTER the "custom task runner" command is finished running. It will display the combined results of the "custom task runner" command.

Part Four - Hands-on Walkthrough of the Custom Automation Runners and Advanced Stand-alone Commands

I will be demonstrating the following Custom Automation Runners and Advanced Stand-alone Commands...

Advanced Stand-alone Commands...

- "time delayed runner one" <-- This time delayed command will wait for a specific amount of time defined in the robotframework-test-assistant.py leon-ai module. After waiting it will run a single command or many commands chained together in a specific order.
- "time delayed runner two" <-- This time delayed command will wait for a specific amount of time defined in the robotframework-test-assistant.py leon-ai module. After waiting it will run a single command or many commands chained together in a specific order.
- "time delayed runner three" <-- This time delayed command will wait for a specific amount of time defined in the robotframework-test-assistant.py leon-ai module. After waiting it will run a single command or many commands chained together in a specific order.
- "bug risk" <-- This time delayed command will wait for a short duration, run a set of Robot Framework RPA commands that uses a Docker Container that checks the git commit bug risk of a specific repo using a tool called gitrisky, and sends notifications to specific Slack channels using slacktee if there are commits with a bug risk score of 0.1 or higher.
- "generic on demand runner" <-- This on-demand (no time delay) command will immediately run many commands chained together in a specific order.
- "results dashboard" <-- This on-demand (no time delay) command requires a Heroku account integrated with a GitHub repo you control. If you don't set this up to handle your GitUb repo on your own Heroku account, then leon-ai will probably immediately reply with the last combined results log that I already deployed to Heroku.

Questions about Part Four?

The End...

:-) ...for now.

References

- “Leon Docs”, https://docs.getleon.ai/
- “Robot Framework RPA”, https://robotframework.org/rpa/