# Detailed Documentation of the Provided Python Script

This document provides a structured, block-by-block explanation of the provided Python script, which automates the extraction of NIC (Network Interface Card) link status from server iLO/management interfaces using Selenium, processes the data, applies color formatting, and writes results into an Excel file using **openpyxl**.

---

## 1. Import Statements

This block imports all required libraries for browser automation, Excel manipulation, web scraping, handling warnings, timing, and filesystem operations.

```
from openpyxl import Workbook, load_workbook
from openpyxl.cell.text import InlineFont
from openpyxl.cell.rich_text import TextBlock, CellRichText
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.action_chains import ActionChains
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.alert import Alert
from bs4 import BeautifulSoup
import warnings
import requests
import time
import os
import glob
```

### Purpose:

- `openpyxl` → modify Excel files and embed rich text with colors.
- `selenium` → automate web browser interactions.
- `BeautifulSoup` → used for parsing HTML when needed.
- `pandas`, `os`, `glob`, `time`, etc. → utility functions.

---

## 2. Browser Setup (Selenium Edge WebDriver)

Creates an Edge browser instance configured to stay open and ignore SSL certificate errors.

```
opt = webdriver.EdgeOptions()
opt.add_experimental_option('detach', True)
opt.add_argument("--ignore-certificate-errors")
opt.set_capability("acceptInsecureCerts", True)
driver = webdriver.Edge(options=opt)
driver.minimize_window()
print("Browser opened")
```

**Purpose:**

- Allows Selenium to open HTTPS sites with invalid certificates (common in internal servers).
- Keeps the browser open after the script ends for debugging.
- Minimizes the window to avoid distractions.

---

## 3. Load or Create Excel Workbook

Checks if the Excel file exists; if not, creates a new workbook.

```
try:
    wb = load_workbook("DR_sheet -Daily.xlsx")
    ws = wb.active
except FileNotFoundError:
    wb = Workbook()
    ws = wb.active
```

**Purpose:**

- Ensures script can write NIC status to an existing or newly created sheet.

---

## 4. NIC Text Coloring Function

Defines how NIC port text is styled for Excel.

```
def get_nic_ports(link_status, port):
    text_color = "FF000000"  # default: black
    if str(link_status) == "Up":
        text_color = "FF00FF00"  # green
    elif str(link_status) == "Down":
        text_color = "FFFF0000"  # red
    return TextBlock(InlineFont(color=text_color), port+" ")
```

**Purpose:**

- Returns colored text blocks used in Excel rich text formatting.

- Green for **Up**, Red for **Down**, Black for anything else.

---

## 5. Insert Row for Standard NICs (Single Slot Servers)

Stores NIC info into Excel column **H**.

```
def insert_row(ip_address, status, ws, row, two_slot_exist=False, slot_names
= []):
```

### Logic Summary:

- If the server has two NIC slots → complex formatting.
- If only one NIC → simpler formatting.
- Writes result into column `H`.

### Purpose:

- Converts extracted NIC status into formatted Excel rich text.
- Handles both **single-slot** and **dual-slot** servers.

---

## 6. Insert Row for Integrated NICs (Column I)

Handles integrated NIC ports (up to 4 ports).

```
def insert_row_integrated(ip_address, status, ws, row, slot_names=[]):
```

### Logic Summary:

- Creates a rich text cell containing only non-empty NIC values.
- Skips empty ports.
- Writes into column **I**.

### Purpose:

- Ensures integrated slot NIC link statuses are formatted cleanly.

---

## 7. Main Function: get_info()

This is the heart of the script. It logs into an iLO/iDRAC-like interface and extracts NIC port statuses.

```
def get_info(driver, ip_address, row):
```

### Key Steps:

1. Identify if server contains **two NIC slots** using a hardcoded list.
2. Open browser: `driver.get(f'https://{ip_address}')`.
3. Login using credentials.
4. Navigate through nested frames.
5. Expand **Hardware** → **Network Devices**.
6. Extract NIC status for:
7. Single-slot NICs
8. Dual-slot NICs
9. Extract **Integrated NIC** status (up to 4 ports).
10. Pass data to Excel insertion functions.

### Error Handling:

If any exception occurs: - Saves current workbook state. - Closes browser.

---

## 8. Retrieve IP List and Run the Script

Loads IPs from Excel column **F**, starting from row 3.

```
ip = [cell.value for cell in ws['F'][2:]]
for index, address in enumerate(ip):
    get_info(driver, address, int(ip.index(address))+3)
```

### Purpose:

- Iterates through each server IP.
- Calls `get_info()` to extract NIC statuses.
- Row index is calculated dynamically.

---

## 9. Save and Close

Final save and browser shutdown.

```
wb.save("output.xlsx")
driver.close()
```

### Purpose:

- Ensures processed data is safely written to Excel.
- Closes browser once all IPs are processed.

---

## Summary of Script Functionality

This script automates the monitoring of server NIC link statuses. It: - Opens an internal management webpage. - Logs in automatically. - Navigates the hardware menu. - Reads NIC link statuses. - Colors them depending on Up/Down. - Handles integrated and detachable NIC slots. - Writes the final results to Excel.

---

If you want, I can also add: - Flowchart diagram - Sequence diagram - PDF export of this documentation - A simplified or optimized version of the script - Refactoring into modular classes or OOP structures

Just let me know!