Last Resort

Release v2024.11-3

Zdeněk Lach

Jan 14, 2025

CONTENTS:

CHAPTER

ONE

LAST_RESORT MODULE

last_resort.**cleanup**()
Perform cleanup actions.

This function performs necessary cleanup actions such as deleting swap files and locking the run report.

Returns None

last_resort.**main**()
Main entry point for the Last Resort application.

This function initializes the application by performing a series of setup and initialization tasks. It handles the following steps:

1. Parses command-line arguments using the argument_parser module. - The arguments include options for silent mode, debug mode, verbose mode, test mode, and displaying help information.
2. Initializes the Tkinter root window. - Sets up the main window for the graphical user interface.
3. If the –info argument is provided, prints the help message and exits. - Provides users with detailed usage instructions and exits the application.
4. Prints the introductory message. - Displays a welcome message and information about the application.
5. Generates a unique identifier (RUID) for the session. - Creates a unique run identifier to track the session.
6. Identifies all folders around the launch directory. - Determines the paths for various project directories such as mask_name, revision, dataprep, and final_mask.
7. Initializes the main window of the user interface. - Sets up the main window components and layout.
8. Initializes the folder structure. - Calls the init_folder_structure function to set up the necessary directories.
9. Searches for .jb and .po files in the revision directory. - Looks for job files (.jb) and purchase order files (.po) in the specified directory.
10. If no .po or .jb files are found, raises a FileNotFoundError. - Ensures that the necessary files are present before proceeding.
11. Prompts the user to select .jb and .po files, loads the selected .po file into the checklist, and opens both files for preview in the selected editor. - Provides a graphical interface for the user to select and preview the required files.
12. If the –test argument is provided, loads the first .po file into the checklist and expands the window. - Automatically selects the first .po file for testing purposes and adjusts the window size.
13. Starts the Tkinter main loop to display the main window. - Enters the main event loop to keep the application running and responsive.

Raises **FileNotFoundError** – If no .po or .jb files are found in the revision directory.

CHAPTER

TWO

CONFIG_PARSER MODULE

config_parser.**get**(*key*)

Retrieve a value from the configuration.

This function ensures that the configuration is initialized and then retrieves the value associated with the specified key from the configuration.

Parameters **key** (*str*) – The key to look up in the configuration.

Returns The value associated with the specified key in the configuration.

Raises

- **ValueError** – If the key is not found in the configuration.
- **FileNotFoundError** – If the configuration file is not found.

config_parser.**initialize**()

Initialize and load the configuration file into memory.

This function searches for the configuration file named 'default_config.yaml' in the current directory and its subdirectories. If the file is found, it loads the configuration into the global variable 'full_configuration'. If the file is not found, it raises a FileNotFoundError.

Raises **FileNotFoundError** – If the configuration file is not found.

CHAPTER

THREE

ARGUMENT_PARSER MODULE

argument_parser.**get**()

Retrieve the parsed command-line arguments.

If the arguments have not been parsed yet, this function initializes the parser and parses the arguments. It then returns the parsed arguments.

Returns The parsed command-line arguments.

Return type argparse.Namespace

CHAPTER

FOUR

UTILS MODULE

class utils.**Type**(*value*)

Bases: enum.Enum

Enumeration for log message types.

This enum defines the types of log messages that can be used in the application. The types include: - INFO: Informational messages. - WARNING: Warning messages. - ERROR: Error messages. - QUESTION: Question messages.

ERROR = 3

INFO = 1

QUESTION = 4

WARNING = 2

utils.**custom_tab**(*size=None*)

Generate a string of spaces for tabulation to prevent "leave out too many symbols" behavior.

Parameters **size** (*int, optional*) – The number of tab lengths to generate. Defaults to None.

Returns A string of spaces for tabulation.

Return type str

utils.**debug**(*message*, *log=None*)

Print a debug message if debug mode is enabled.

This function formats and prints a debug message with a custom tag if debug mode is active. It also writes the message to the history log if a log argument is provided.

Parameters

- **message** (*str*) – The debug message to print.
- **log** (*log, optional*) – An optional log to write the message to.

Returns None

utils.**format_date**(*date*)

Format a date into a specific string format.

This function formats a given date into the format "ddMMMyy", where "MMM" is the abbreviated month name with the first letter capitalized.

Parameters **date** (*datetime.date*) – The date to be formatted.

Returns The formatted date string.

Return type str

utils.**format_line**(*text*, *fill_symbol=None*, *edge_symbol=None*)

Format a line of text with optional fill and edge symbols.

This function formats a line of text to fit within a predefined width, optionally filling the line with a specified symbol and adding edge symbols. It ensures that the text is centered within the line. It is meant to be used for a 'text in the box' kind of formatting for the CL UI.

Parameters

- **text** (*str*) – The text to format.
- **fill_symbol** (*str, optional*) – Symbol to fill the line. If provided, the entire line will be filled with this symbol. Defaults to None.
- **edge_symbol** (*str, optional*) – Symbol for the edges of the line. If provided, the line will be enclosed with this symbol. Defaults to None.

Returns The formatted line of text.

Return type str

utils.**gather_intel**(*root*)

Collect information and prompt the user for necessary actions.

This function prompts the user to confirm if they have run daps_trans. If not, it offers to run daps_trans. It also prompts the user to send an email and reveals additional options in the user interface.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

Returns None

utils.**generate_final_mask_folder**()

Create a new final mask folder by copying the 'mebes' directory.

This function logs the user's choice to create a final mask folder, attempts to copy the 'mebes' directory from the data preparation directory to a new directory, and returns the path to the newly created directory.

Returns The path to the newly created directory.

Return type str

Raises

- **PermissionError** – If there is a permission issue during the folder copying process.
- **Exception** – If any other unexpected error occurs.

utils.**get_full_name**()

Retrieve the full name of the current user.

This function gets the current username, retrieves the full name from the system's password database, and splits it into first name, last name, and username.

Returns A tuple containing the first name, last name, and username.

Return type tuple

utils.**get_release_version**()

Retrieve the release version from the releaseTag.txt file.

This function reads the releaseTag.txt file located in the $ONSEMI/cde/$CDE_REV directory and returns the release version.

Returns The release version.

Return type str

utils.**kill_processes**()

Terminate all running instances of the configured editor.

This function iterates through all running processes and terminates any process that matches the name or PID of the configured editor. It also performs cleanup actions such as removing swap files.

Returns None

Raises

- **OSError** – If no process with the name of the configured editor is found.
- **Exception** – For any other errors that occur while attempting to kill the processes.

utils.**last_resort**()

Execute the final set of actions for the application.

This function performs critical final steps including: 1. Setting paths for the final mask and dataprep forms directories. 2. Identifying the .po file within the final mask directory. 3. Running the daps_po2pdf command to convert the .po file to a .pdf. 4. Locating the generated .pdf file. 5. Moving the .pdf file to the 'forms' directory.

Returns None

utils.**open_for_preview**(*file_to_open*, *screen_position='left'*)

Launch a file for preview using the configured editor.

This function opens the specified file in the configured editor and tracks the process ID. It supports positioning the window on the left or right side of the screen.

Parameters

- **file_to_open** (*str*) – The path to the file to open.
- **screen_position** (*str*) – The screen position for the window ('left' or 'right').

Returns None

utils.**print_help**()

Display the comprehensive help message for the Last Resort application.

This function prints detailed usage instructions, available options, a description of the application, and examples of how to run the application. It is intended to guide users on how to effectively use the last_resort.py script.

The help message includes:

- Usage syntax
- List of command-line options with descriptions
- A detailed description of the application's purpose and functionality
- Step-by-step explanation of the application's workflow
- Examples demonstrating common usage scenarios

Usage: last_resort.py [OPTIONS]

Options:

-s, --silent	Enable silent mode (suppress output).
-d, --debug	Enable debug mode (detailed output for troubleshooting).
-v, --verbose	Enable verbose mode (more detailed output).
-t, --test	Run the application in test mode.
-i, --info	Show this help message and exit.

-c, –config [PATH] Path to provide a custom configuration file. -dc, –define_config Open a window to define a new temporary configuration for this session.

Description: The Last Resort application allows RDP engineers to perform a final check before sending data to the vendor, ensuring there are no typos, missing, or incorrect information.

The application performs the following steps:

1. Parses command-line arguments.
2. Initializes the Tkinter root window.
3. Prints an introductory message.

4. Generates a unique identifier (RUID).
5. Identifies all folders around the launch directory.
6. Initializes the main window of the user interface.
7. Initializes the folder structure.
8. Finds all *.jb* and *.po* files in the revision directory.
9. **Depending on whether the 'test' argument is provided:**
   - If 'test' is True, loads the first *.po* file into the checklist for testing purposes.
   - If 'test' is False, prompts the user to select *.jb* and *.po* files,

   loads the selected *.po* file into the checklist, and opens both selected files for preview.

Examples

Run the application normally: $ python last_resort.py

Run the application in test mode: $ python last_resort.py –test

Enable verbose output: $ python last_resort.py –verbose

Enable debug mode for troubleshooting: $ python last_resort.py –debug

Show this help message: $ python last_resort.py –info

Use a specific configuration file: $ python last_resort.py –config /path/to/config.yaml

Open a window to define a new temporary configuration for this session: $ python last_resort.py –define_config

For more information or if you encounter any issues, please contact Zdenek Lach.

utils.**print_intro**()
Print the introductory message for the application.

This function prints the application title, version, and usage instructions based on the command-line arguments provided.

utils.**printer**(*message*, *log_type=None*)
Print a formatted message with a specific log type.

Requirements for import: printer, custom_tab, Type

Parameters

- **message** (*str*) – The message to print.
- **log_type** (Type, *optional*) – The type of log message (INFO, WARNING, or None). Defaults to PRINTER.

Returns None

utils.**run_daps_trans**()
Execute the daps_trans command and manage its output.

This function runs the daps_trans command within the final mask directory, displays a reminder message, and logs the output for debugging purposes.

Returns True if the command executes successfully, False otherwise.

Return type bool

utils.**send_email**(*root*, *user_input_expedite=None*)
Initiate the email sending process using a specified script.

This function attempts to run an email script and handles any errors that occur. If multiple .po files cause the process to fail, it adjusts the directory and retries.

Parameters

- **root** (*tk.Tk*) – The root window of the Tkinter application.
- **user_input_expedite** – The functions stores the user-selected
- **attempt.** (*input in case it's recursively ran again after creating the tar file after a failed*) –

Returns None

utils.**simulate_daps_trans**()

Simulate the DAPS_TRANS process.

This function attempts to run the last_resort function. If a FileNotFoundError is caught, it checks if the final mask directory is "/mebes". If so, it prompts the user to create the final mask folder, attempts to copy the folder, set the new final mask directory, create a tar file, and rerun the last_resort function.

Raises

- **FileNotFoundError** – If the final mask directory is not found.
- **PermissionError** – If there is a permission error while copying the folder.
- **Exception** – If any other unexpected error occurs.

utils.**startup_dir_check**()

Validate the current directory for running the application.

This function checks if the current working directory starts with the configured masks project directory. If not, it prints error messages and exits the application.

Returns None

utils.**tar_file**()

Create a tar archive of the final mask directory.

This function constructs the path for the tar archive and creates a tar.gz file containing all files in the final mask directory, excluding any existing tar.gz files. It displays a success message upon completion or an error message if the process fails.

Raises **Exception** – If there is an error creating the tar file or running the tar command.

utils.**verbose**(*message*)

Print a verbose message if verbose or debug mode is enabled.

This function formats and prints a verbose message with a custom tag if either verbose or debug mode is active. It also writes the message to the history log.

Parameters **message** (*str*) – The verbose message to print.

Returns None

CHAPTER

FIVE

FILE_OPERATIONS MODULE

class file_operations.**PDF**(*orientation='P'*, *unit='mm'*, *format='A4'*)
Bases: fpdf.fpdf.FPDF

footer()
Add a footer to the PDF with the RUID.

This method sets the footer to 1.5 cm from the bottom, selects Arial italic font, and prints the RUID centered at the bottom of the page.

file_operations.**cleanup_swp_files**()
Delete all .swp files in the specified directory and its subdirectories.

This function finds and deletes all swap (.swp) files in the final mask directory.

Returns None

file_operations.**convert_lr_to_pdf**(*file_in_lr_format*)
Transform a .lr file into a .pdf file.

This function converts a specified .lr file to a .pdf file by invoking the txt_to_pdf function. It logs the conversion process for debugging purposes.

Parameters **file_in_lr_format** (*str*) – The path to the .lr file to be converted.

Returns True if the conversion is successful, False otherwise.

Return type bool

file_operations.**copy_folder**(*source_dir*, *destination_dir*)
Copy the contents of the source directory to the destination directory.

Parameters

- **source_dir** (*str*) – The path to the source directory.
- **destination_dir** (*str*) – The path to the destination directory.

Returns The path to the destination directory.

Return type str

Raises **Exception** – If an unexpected error occurs during the copying process.

file_operations.**find_files**(*base_name=None*, *extension=None*, *directory=None*)
Search for files in a directory that match a specific base name and/or extension.

This function walks through the directory tree starting from the specified directory and collects files that match the given base name and/or extension.

Parameters

- **base_name** (*str, optional*) – The base name of the files to find. Defaults to None.
- **extension** (*str, optional*) – The file extension to find. Defaults to None.
- **directory** (*str, optional*) – The directory to search in. Defaults to the current working directory.

Returns A list of matching file paths.

Return type list

file_operations.**find_folders**(*directory*)
Locate all folders within a specified directory.

This function walks through the directory tree and collects all folder paths.

Parameters **directory** (*str*) – The directory to search for folders.

Returns A list of folder paths.

Return type list

file_operations.**generate_checklist_filename**(*base_name*, *extension*, *checklist_dir=None*)
Generate a new filename with an incremented version number.

This function checks for existing files with the same base name and extension, increments the version number, and generates a new filename.

Parameters

- **base_name** (*str*) – The base name of the file.
- **extension** (*str*) – The file extension.
- **checklist_dir** (*str, optional*) – The directory to search for existing files. Defaults to None.

Returns The generated filename with the incremented version number.

Return type str

file_operations.**generate_final_folder_name**()
Generate the final folder name based on mask name, grade, and date.

This function constructs the final folder name using the mask name, mask grade (MS or MSW), and the current date. It logs the process for debugging purposes.

Returns The generated folder name, or None if the mask grade cannot be determined.

Return type str

file_operations.**generate_ruid**()
Create a unique run identifier (RUID) and generate a run report file.

This function generates a unique RUID, creates a run report file in the run_archive directory, and writes the RUID to the file.

Returns The generated RUID.

Return type str

Raises **OSError** – If the run report file cannot be written.

file_operations.**get_identified_folders**()

file_operations.**handle_multiple_final_mask_folders**(*root*)
Manage multiple final mask folders by prompting the user to select one.

This function retrieves the pattern from the configuration, finds folders matching the pattern in the dataprep directory, filters them, and prompts the user to select one if multiple matches are found.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

file_operations.**identify_folders**(*root*)
Determine and set paths for various project directories.

This function identifies the current working directory and matches it against predefined patterns to set paths for mask_name, revision, dataprep, and final_mask directories. It handles different launch scenarios based on the recognized directory.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

Returns A dictionary with paths for mask_name, revision, dataprep, and final_mask directories.

Return type dict

Raises **ValueError** – If the current directory does not match any expected patterns.

file_operations.**init_folder_structure**(*root*)
Set up the folder structure by identifying project directories.

This function initializes the folder structure by calling identify_folders and logging the identified directories.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

Returns A list of identified directory paths.

Return type list

file_operations.**lock_run_report**(*pdf_mode=True*)
Set the run report file to read-only.

This function changes the permissions of the run report file to read-only (0o444). If pdf_mode is True, it converts the run report to a PDF before setting it to read-only.

Parameters **pdf_mode** (*bool*) – Whether to convert the run report to a PDF before locking it. Defaults to True.

file_operations.**match_folder**(*folder_name*, *patterns*)
Check if a folder name matches any pattern in a given set.

This function iterates over a dictionary of patterns and checks if the folder name matches any of the regex patterns. If a match is found, it logs the match and returns the name of the matched pattern.

Parameters

- **folder_name** (*str*) – The name of the folder to check.
- **patterns** (*dict*) – A dictionary where keys are pattern names and values are regex patterns.

Returns The name of the matched pattern, or None if no match is found.

Return type str

file_operations.**navigate_directory**(*path*, *levels*, *pattern=None*)
Traverse directories up or down a specified number of levels.

This function navigates through directories starting from a given path. It can move up or down a specified number of levels and optionally match subdirectories against a regex pattern.

Parameters

- **path** (*str*) – The starting directory path.
- **levels** (*int*) – Number of levels to move. Positive for descending, negative for ascending.
- **pattern** (*re.Pattern, optional*) – Regex pattern to filter subdirectories. Defaults to None.

Returns The resulting directory path after navigation.

Return type str

Raises **FileNotFoundError** – If no matching subdirectory is found.

file_operations.**save_checklist_to_file**()
Save checklist items to a file in the 'docs' directory.

This function creates the docs directory if it doesn't exist, generates a new filename, and writes the checklist items to the file.

Returns The path to the saved checklist file, or None if an error occurs.

Return type str

file_operations.**save_checklist_to_pdf**()
Save checklist items to a PDF file.

This function saves the checklist items to a temporary file, converts it to a PDF, and sets the PDF file to read-only.

Returns The path to the saved PDF file, or None if an error occurs.

Return type str

file_operations.**search_string_in_file**(*file_path*, *search_string*)
Search for a specific string in a file.

This function reads the content of a file and checks if a given string is present. It handles file not found and other exceptions, logging errors as needed.

Parameters

- **file_path** (*str*) – The path to the file to be searched.
- **search_string** (*str*) – The string to search for in the file.

Returns True if the string is found, False otherwise.

Return type bool

file_operations.**txt_to_pdf**(*txt_file*, *pdf_file*)
Convert a text file to a PDF file.

This function reads the content of a text file, writes it to a PDF file, and handles any permission errors by generating a new filename.

Parameters

- **txt_file** (*str*) – The path to the text file.
- **pdf_file** (*str*) – The path for the PDF file to be created at.

Returns The path to the created PDF file.

Return type str

file_operations.**write_history**(*message*)
Append a message to the run report file with a timestamp.

This function writes a given message to the run report file, prefixed with the current timestamp.

Parameters **message** (*str*) – The message to append to the run report file.

Returns None

CHAPTER

SIX

USER_INTERFACE MODULE

user_interface.**center_window**(*window*)
Attempt to center the window on the screen.

Parameters **window** (*tk.Toplevel*) – The window to be centered.

user_interface.**close_app**(*root*)
Close the application and perform cleanup actions.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

user_interface.**confirm_button_action**(*root*)
Perform actions when the confirm button is clicked. 1. Closes subprocesses 2. saves checklist to the pdf 3. Starts prompting user with gather_intel function

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

user_interface.**expanded_view**(*root*)
Reveal additional buttons in the main window. To allow manual operations.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

user_interface.**init_main_window**(*root*)
Initialize the main window of the application.

Parameters **root** (*tk.Tk*) – The root window of the Tkinter application.

user_interface.**load_checklist**(*root*, *selected_po*)
Load the checklist items into the main window.

Parameters

- **root** (*tk.Tk*) – The root window of the Tkinter application.
- **selected_po** (*str*) – The selected .po file.

user_interface.**prompt_selection**(*root*, *options*, *title*, *is_file_list=None*, *is_folder_list=None*)
Prompt the user to select an option from a list.

Parameters

- **root** (*tk.Tk*) – The root window of the Tkinter application.
- **options** (*list*) – A list of options to choose from.
- **title** (*str*) – The title of the prompt window.
- **is_file_list** (*bool, optional*) – Indicates if the selection is meant for a list of files. Defaults to None.
- **is_folder_list** (*bool, optional*) – Indicates if the selection is meant for a list of folders. Defaults to None.

Returns The selected option.

Return type str

user_interface.**resize_based_on_input_length**(*root*, *options*)
Resize the window based on the length of the input options.

Parameters

- **root** (*tk.Tk*) – The root window of the Tkinter application.
- **options** (*list*) – A list of options to be displayed.

user_interface.**show_config_popup**(*root*)

CHAPTER

SEVEN

FILE_PRESENTER MODULE

file_presenter.**file_presenter**(*provided_file=None*, *screen_position='left'*)

Launch a read-only file presenter window.

This function creates a Tkinter window to display the contents of a specified file in read-only mode. The window can be positioned on the left or right side of the screen based on the provided argument.

Parameters

- **provided_file** (*str, optional*) – The path to the file to open. Defaults to None.
- **screen_position** (*str, optional*) – The screen position for the window ('left' or 'right'). Defaults to 'left'.

Returns None

CHAPTER

EIGHT

PO_PARSER MODULE

po_parser.**parse_po_file**(*file_to_parse_path*)
Parses the provided PO file and extracts relevant information.

Parameters **file_to_parse_path** (*str*) – The path to the PO file.

Returns A dictionary containing the extracted information.

Return type dict

po_parser.**print_po**(*po*)
Prints the parsed PO information in a nicely formatted way.

Parameters **po** (*dict*) – The parsed PO information.

CHAPTER

NINE

SEND_EMAIL MODULE

class send_email.**Person**(*first_name*, *last_name*, *phone*, *email*)
Bases: object

send_email.**create_email_body**(*contacts*, *cc_contacts*, *stepper*, *urgency*, *address*, *user_details*, *country*, *sent_file*, *grade*, *po_info*)
Prepare the body of the email using provided information.

This function constructs the HTML body of an email using the provided information such as contacts, stepper, urgency, address, user details, country, sent file, grade, and purchase order (PO) information.

Parameters

- **contacts** (*list*) – List of email addresses to send the email to.
- **cc_contacts** (*list*) – List of email addresses to CC.
- **stepper** (*str*) – The stepper type.
- **urgency** (*str*) – The urgency of the order.
- **address** (*str*) – The shipping address.
- **user_details** (*dict*) – Dictionary containing user details.
- **country** (*str*) – The country of the fab.
- **sent_file** (*str*) – The name of the sent file.
- **grade** (*str*) – The mask grade.
- **po_info** (*dict*) – Dictionary containing purchase order information.

Returns The HTML body of the email.

Return type str

Example

```
>>> create_email_body(
        contacts=["example@example.com"],
        cc_contacts=["cc@example.com"],
        stepper="Nikon",
        urgency="Standard",
        address="123 Street, City, Country",
        user_details={"full_name": "John Doe", "email": "john.doe@example.com"},
        country="Czech Republic",
        sent_file="file.tar.gz",
        grade="A",
        po_info={"vendor": "Vendor", "maskset": "MaskSet", "layers": [{"name":
↪"Layer1", "price": 100}]}
```

(continues on next page)

(continued from previous page)

```
    )
'<html>...</html>'
```

send_email.**determine_grade_and_parameters**(*po_line*)

Determine the stepper, fab, and magnification based on grade.

This function analyzes the purchase order (PO) line to determine the stepper, fab, and magnification based on predefined grades.

Parameters **po_line** (*list*) – A list of items from the purchase order line.

Returns A tuple containing the stepper, fab, magnification, and grade item.

Return type tuple

Raises **ValueError** – If the grade does not match any available grades.

Example

```
>>> determine_grade_and_parameters(['MS'])
('Nikon', 'ONCR', '5', 'MS')
```

send_email.**determine_shipping_address_string**(*fab*)

Determine the shipping address based on the fab type.

This function returns the shipping address and a description based on the specified fab type. It supports "ONCR" and "ISMF" fab types.

Parameters **fab** (*str*) – The fab type ("ONCR" or "ISMF").

Returns A tuple containing the shipping address string and a description.

Return type tuple

Raises **ValueError** – If an invalid fab type is specified.

Example

```
>>> determine_shipping_address_string("ONCR")
('Ship reticles to: CZ4 Fab, ON Semiconductor, Czech Republic<br>...',
 'CZ4 Fab the in Czech Republic')
```

send_email.**extract_mask_layers**(*po_lines*)

Extract mask layers from the given lines.

This function takes a list of lines (each line being a list of words) and extracts mask layers that match the pattern "Pd+[A-Z]?" (e.g., P1, P2A). It returns a list of all extracted mask layers.

Parameters **po_lines** (*list*) – A list of lines, where each line is a list of words.

Returns A list of extracted mask layers that match the pattern.

Return type list

Example

```
>>> extract_mask_layers([['P1', 'some', 'text'], ['P2A', 'more', 'text']])
['P1', 'P2A']
```

send_email.**get_cc_contacts**(*fab*)

Get CC contacts based on the fab.

This function returns a list of CC (carbon copy) email contacts based on the specified fab type. It includes common RDP_ONCR contacts and additional contacts specific to the fab type ("ONCR" or "ISMF").

Parameters **fab** (*str*) – The fab type ("ONCR" or "ISMF").

Returns A list of CC email addresses for the specified fab.

Return type list

Example

```
>>> get_cc_contacts("ISMF")
['zdenek.lach@onsemi.com', 'jiri.prokop@onsemi.com', ..., 'nuratiqahzafirah.
↪jaafar@onsemi.com']
```

send_email.**get_full_user_details**(*username*)

Get detailed information about a user.

This function retrieves detailed information about a user from the system's passwd entry and additional details from the RDP_ONCR_TEAM_MEMBERS dictionary. It returns a dictionary containing the user's details, including username, encrypted password, user ID, group ID, full name, first name, last name, home directory, shell, phone number, and email address.

Parameters **username** (*str*) – The username of the user to retrieve details for.

Returns

A dictionary containing the user's details, or None if the user does not exist or an error occurs.

Return type dict

Raises **subprocess.CalledProcessError** – If an error occurs while retrieving the passwd entry for the user.

Example

```
>>> get_full_user_details("username")
{
    'username': 'username',
    'encrypted_password': 'x',
    'user_id': 'XXXX',
    'group_id': 'XXXX',
    'full_name': 'first_name last_name',
    'first_name': 'first_name',
    'last_name': 'last_name',
    'home_directory': '/home/username',
    'shell': '/bin/bash',
    'phone': 'XXX XX XXXX',
    'email': 'first_name.last_name@onsemi.com'
}
```

send_email.**get_vendor_name**(*vendor*)

Determine the correct vendor name based on the provided vendor string.

This function maps a given vendor string to the correct vendor name based on predefined vendor lists.

Parameters **vendor** (*str*) – The vendor string to be mapped.

Returns The correct vendor name, or None if no match is found.

Return type str

Example

```
>>> get_vendor_name("COMPUGRAPHICS")
'Compugraphics'
```

send_email.**load_file**(*filename*)

Load a file and yield lines as lists of words.

This function checks if the specified file exists. If it does, it reads the file and returns a list of lines, where each line is split into a list of words. If the file does not exist, it logs an error message and returns an empty list.

Parameters **filename** (*str*) – The path to the file to be loaded.

Returns

A list of lines, where each line is a list of words. Returns an empty list if the file does not exist.

Return type list

Example

```
>>> load_file("example.txt")
[['This', 'is', 'a', 'line'], ['Another', 'line', 'here']]
```

send_email.**main**()

Main function to execute the script.

This function orchestrates the execution of the script, including retrieving user details, parsing purchase order (PO) files, determining shipping address, and sending an email with the prepared body and attachments.

Example

```
>>> if __name__ == "__main__":
        main()
```

send_email.**select_email_contacts**(*vendor*, *fab*)

Determine email contacts based on vendor and fab.

This function determines the appropriate email contacts based on the provided vendor and fab type. It returns a list of email addresses for the specified vendor and fab. If no matching contacts are found, it returns a message indicating the failure to determine email addresses.

Parameters

- **vendor** (*str*) – The vendor name.
- **fab** (*str*) – The fab type (“ONCR” or “ISMF”).

Returns A list of email addresses for the specified vendor and fab.

Return type list

Example

```
>>> select_email_contacts("Compugraphics", "ONCR")
['customer-support-uk@compugraphics-photomasks.com']
```

send_email.**select_vendor**(*po_lines*)

Determine the vendor based on PO content.

This function checks the content of the purchase order (PO) lines and determines the vendor based on predefined keywords.

Parameters **po_lines** (*list*) – A list of lines from the purchase order.

Returns The name of the vendor.

Return type str

Raises **ValueError** – If no matching vendor is found.

Example

```
>>> select_vendor([[], [], [], [], ['COMPUGRAPHICS']])
'Compugraphics'
```

send_email.**send_email**(*body*, *stepper*, *po_pdf_full_path*, *fab*, *user_details*, *po_info*)

Send the email with the prepared body and attachments.

This function sends an email with the provided HTML body and attachments, including a logo image and a PDF file. The email is sent to the user's email address.

Parameters

- **body** (*str*) – The HTML body of the email.
- **stepper** (*str*) – The stepper type.
- **po_pdf_full_path** (*str*) – The full path to the PO PDF file.
- **fab** (*str*) – The fab type ("ONCR" or "ISMF").
- **user_details** (*dict*) – Dictionary containing user details.
- **po_info** (*dict*) – Dictionary containing purchase order information.

Example

```
>>> send_email(
        body="<html>...</html>",
        stepper="Nikon",
        po_pdf_full_path="/path/to/po.pdf",
        fab="ONCR",
        user_details={"full_name": "John Doe", "email": "john.doe@example.com"},
        po_info={"vendor": "Vendor", "maskset": "MaskSet", "layers": [{"name":
↪"Layer1", "price": 100}]}
    )
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

N

O

P

Q

R

S

T

U

V

W