

OCR Extraction for Pre-Payment Processing

Developer's Guide

Overview

This guide is intended for developers and users who seek to modify or extend the functionality or features of this software. To do this, this guide contains information on important files to start with when developers/users are exploring the software, how they can configure the current settings of the program, and how they can add/remove/modify the functionality of the program. Those modifications can be split into two pieces, the first being changes to the frontend, and the second being changes to the backend.

Important Files

To begin with, users/developers would want to explore the *main.py* file, which is the file that runs the server of the software. This file is found in the *src/main* directory of the file structure. Inspecting this file in conjunction with *home.html*, which is found in the *src/main/templates* directory, will give the inspector an idea of the frontend of the system. When *main.py* is run, it starts the server, and when a user navigates to the URL of the program, *main.py* retrieves and displays *home.html* to the user, which is the interface that users see. The frontend of the application consists solely of those two files, and within *main.py* is a connection to the backend. This line of code `img, result = controller_entry_point(path)` calls *controller.py*, which is found in *src/main/backend*, and this file dictates the flow of control of the software. It sends the image to the preprocessing stage, where the user exploring the system would want to investigate the *preprocess_main.py*, which is in the *src/main/backend/preprocess* directory. This file handles all preprocessing of the check image, and returns it to the controller. Then the image is sent to the field extraction stage, where the primary file to inspect is the *field_extraction_main.py* in the *src/main/backend/field_extraction* directory. This file determines the field locations on the check, then returns them to the controller. Those files are then sent to the data extraction stage, where the core file to investigate is the *data_extraction_main.py* in the *src/main/backend/data_extraction* directory. This file extracts the data from the fields passed to it, and eventually the data returns to the controller again. Finally, this data is sent to the post processing stage, where the user should investigate the *postprocess_main.py* file in the *src/main/backend/postprocess* directory. This file adds the bounding boxes to the image and resizes it, then sends it back to the controller so that it is finally passed back to the front end.

Configuration

Within the validation stage of the program, the Number Amount and Written Out Amount fields are validated using an upper and lower threshold. These thresholds are configurable, and can be

set in the *config.yml* file in *src/main/config*. The *config.yml* file also contains the max age limit of a check (currently as 6 months), which is defined as the date difference between a checks' Date of Signing and Date of Submission (to the software). Developers can also add any other configurable details or settings such as the valid routing numbers of a check, minimum and maximum sizes of account and routing numbers, etc. Doing so allows for easy configurability of those settings later down the line.

Frontend Changes

Regarding changes to the front end, these can be simpler modifications, such as changing the title of the website, or major ones such as adding multiple new pages to the site, or anything in between. For any change, the developer/user would only need to modify the *main.py* file (the 'server'), and edit/add/modify/delete any template files (html files).

Any change to add a new page would simply require the user to create a new html page, place it in the *src/main/templates* directory, and add a new function to the *main.py*. The function would be similar to this code:

```
@app.route('/', methods=['GET'])
def render_home():
    return render_template('home.html')
```

As an example, we will go through an overview of the procedure to add a new page which simply displays 'Hello World!'. We first define the app route, which tells the server what code to execute when a certain URL is requested. The one shown above which is used in *main.py* is the default route, and the code in *render_home()* is what is called when a person navigates to the homepage of the software. To change that, or add a new page, the app route should be changed accordingly. For our example of adding a new page, we set the app route to '/hello', and the methods (which defines what kind of HTML methods are used) to 'GET'. We then define the function to be executed when the /hello is called as 'display_hello()'. Finally, we set the functions return value to be the new html file that we create and place in the templates directory, which we will call *hello.html*. This html file is very simple, as in its body tag it contains only one additional line: `<h1>Hello World!</h1>`. At the end of all of this, our addition to *main.py* looks like this:

```
@app.route('/hello', methods=['GET'])
def display_hello():
    return render_template('hello.html')
```

An example of a simple change would be to change the title of the website, from “Upload Your Check” to anything else, for example “Hello World!”. To make such a change, we go to the location of the text we want to change, which in our case is the *home.html* file in the *src/main/templates* directory, and change `<title>Upload Your Check</title>` to `<title>Hello World!</title>`.

Any server side change should also occur in *main.py*. For example, if the developer wants to return the data in non-JSON format, all we need to do is go to *main.py* and remove the `jsonify()` from the following code:

```
return jsonify(  
    image=image_to_data(path),  
    results=result  
).
```

Backend Changes

Regarding changes to the backend, we have simplified the process by splitting each stage of the software into different ‘modules’. By doing this, the developer only needs to modify the files pertaining to that specific module.

For example, to make a change to preprocessing, such as the developer no longer wants to use greyscaling, all they need to do is go to the *src/main/backend/preprocess* directory and remove the line of code within *preprocess_main.py* that performs the greyscaling. Likewise, in order to do the opposite, to add/apply another function to the image, or change the overall preprocess stage entirely, all the developer needs to do is change that module and everything outside of it will continue to run as it was.

Another potential change that could be made is to add another validation test for the Memo field. To do this, we go to the Validation ‘module’ that consists of a method within the field Python classes. For memo, we would go to *src/main/backend/data_extraction/field* directory and open *memo_field.py*, and under its `validate()` function, add an if statement that checks that the Memo string is not empty, like this:

```
def validate(self, data: FieldData):  
    print("Validating if the passed data is a valid memo number")  
    try:  
        str(data.extracted_data)  
    except ValueError:  
        print("Memo is not a string")
```

```

        return False
    if str(data.extracted_data) == "":
        return False

    print("The memo is valid.")
    return True

```

Again we see the modularity in play, as this change was made in the Validation ‘module’, which consists solely of the validate() functions within each field Python class. This change would not affect anything else except the outcome of the Memo validation.

The user/developer might also want to add a new stage to the software’s architecture. Thanks to the modularity with which the software was developed, this is also a simple process. To do this, we would create a new ‘module’ in the file structure, within the src/main/backend directory, and then appropriately call the entry point to that module in *controller.py*, using the correct function, and placing the function call in between the other module function calls corresponding to the new modules stage in the pipeline. An example of this is if we wanted to add another module after preprocessing and before field extraction, we would place the code between the corresponding function calls for those stages, like so:



```

#####
# PREPROCESS PASS
#####
# Save the original dimensions
height = img.shape[0]
width  = img.shape[1]
dim = (width, height)
# Process the image
# pre_image, old_image = prp.preprocessEntryPoint(img)
img, old_image = preprocessEntryPoint(img)

#####
# NEW MODULE PASS
#####
data = new_moduleEntryPoint(img)

#####
# FIELD EXTRACTION PASS
#####

```

```
# Returns a list of fields
# img, fields = fe.extractFieldsEntryPoint(old_image, pre_image)
, fields = extractFieldsEntryPoint(old_image, )
```

In this manner, a new module or even removal of a module does not affect other modules, and results in a cleaner and simpler process for modification of the software.

If the developer would like to modify the files for the open source Handwritten Text Recognition project that we have adapted for our software, they are present in the `src/main/backend/data_extraction/handwriting_extract` directory of the project. For example, the main file of this directory is `src/main/backend/data_extraction/handwriting_extract/src/main.py`. For further information, the developer should consult the original repository: <https://github.com/sushant097/Handwritten-Line-Text-Recognition-using-Deep-Learning-with-Tensorflow>.

Conclusion

In conclusion, as a guide for a developer or user to make modifications or extensions to the software, they should follow the general ideas demonstrated by the above examples for each category of change.