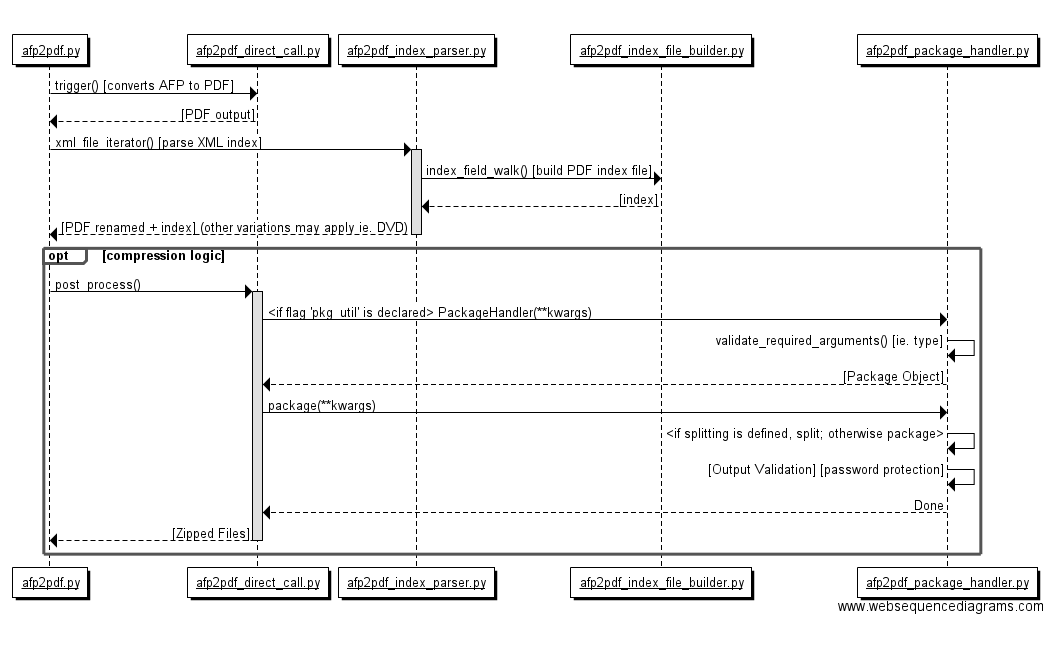
**Project Update – AFP2WEB Packaging Module**

**Business Requirement:**

* The creation of a Python packaging class that will integrate into the AFP2WEB services and perform packaging regardless of the PDF archival variation.
* The program should be capable of splitting a process into multiple packaged files based on either file size and file count.
* Initial support will be built for zip and 7z but should be easily scalable to accommodate future packaging type requirements (ie. tar should be easy to add).
* Allow command line processing for all of the above to act upon a folder of PDFs by giving calling the program with the correct arguments.

**Sequence Flow:**

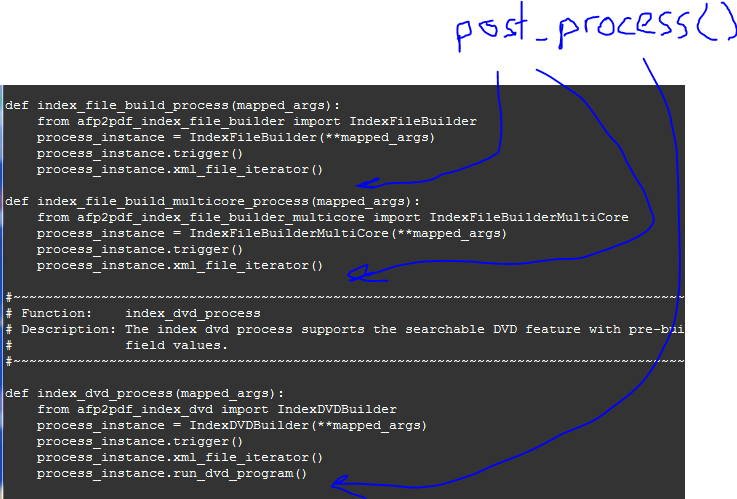


**Add a new “Type” argument to afp2pdf.py:**

This argument will be called “package” which will only apply packaging without AFP2PDF transformation. DP or IS may use this option to manually perform zipping of PDF files for jobs that are still manual.

**Class Design Direction:**

1. For each of the process functions in the entry point script afp2pdf.py, add a post process method call: process\_instance.post\_process(). This will trigger the post process step that you will introduce to handle the packaging class.



1. Write the post\_process() method in the base class afp2pdf\_direct\_call.py. This method will check the self.data dictionary for a key named ‘pkg\_util’:

**[Example: if ‘pkg\_util’ in self.data.keys(): ….code continues… ]**

The dictionary self.data contains all of the data that will come from the configuration file ${cid}${jid}\_pdf.control from the processing script. This means that this will check the values in the processing script for the packaging key.

1. If ‘pkg\_util’ exists (section #2 is true) import your new class PackageHandler

**[Example: from afp2pdf\_package\_handler.py import PackageHandler ]**

This is called “lazy loading”. You won’t load the necessary modules unless a job will require it.

1. Create an instance of PackageHandler and pass it the full self.data dictionary. We will have the PackageHandler constructor [ \_\_init\_\_() ] determine which fields from the self.data dictionary is needed assign the value configs into the kwargs when instantiating:

**[ Example: pkg\_handler = PackageHandler(configs = self.data) ]**

Your Package Handler will take the arguments as \*\*kwargs and parse out configs. **This will allow us to dynamically grow PackageHandler without modifying afp2pdf\_direct\_call.py.**

1. For each configuration that begins with “pkg” assign it to the object attribute (ie. self.pkg\_output\_name = ). Any configurations that can contain multiple values (ie. \_#), create a dictionary using the number as the key and the value as the nested dictionary under the dictionary **“pkg\_values”**. Each nested dictionary will have **“pkg\_src\_path”**, **“pkg\_output\_name”** and/or **“pkg\_src\_file\_pattern**” as the keys.

Example:

‘pkg\_util: ‘zip,

‘pkg\_values’ : {

‘1’ : {

‘pkg\_src\_path’ : ‘<< d\_dir >>/pdf\_afp/entc/entcms1/pdf/<< cycle >>\_disk01’

‘pkg\_output\_name’ : ‘<< d\_dir >>/pdf\_afp/entc/entcms1/zip/ ENTC\_COMBINED\_<< cycle >>\_<< segment >>\_<< sequence >>.zip’

‘pkg\_src\_file\_pattern’ : ‘entc\_archival\_\*\_\*.pdf | entc\_idx\_file.txt’

‘2’ : {

‘pkg\_src\_path’ : ‘<< d\_dir >>/pdf\_afp/entc/entcms1/pdf/<< cycle >>\_disk01’

‘pkg\_output\_name’ : ‘<< d\_dir >>/pdf\_afp/entc/entcms1/zip/ ENTC\_COMBINED\_<< cycle >>\_<< segment >>\_<< sequence >>.zip’

}

1. The PackageHandler constructor should call a validation method (similar to afp2pdf\_direct\_call’s check\_files() ) but call it validate\_arguments(). This method will perform validations on required fields for packaging. Currently there is only 3 required fields: **‘pkg\_util’**, **‘pkg\_src\_path’** (for all “pkg\_values”) and **‘pkg\_output\_name’** (for all “pkg\_values”)
2. ~~The constructor should also call the translation function translate\_arguments() which will resolve all fields that are nested within << >> with their corresponding field values. If a field value or field is declared in the configurations but not passed in, the program should exit out.~~
3. After \_\_init\_\_() runs, the object is returned back to afp2pdf\_direct\_call.py. At this point the PackageHandler instances should have all of the arguments it will need. Do NOT carry over the entire configs dictionary from self.data in afp2pdf\_direct\_call.
4. afp2pdf\_direct\_call will receive the instantiated object and call the package() function.
5. The package() function will perform the actual packaging. It should check self.pkg\_util to find out if it is using a 7z or a normal zip packaging. It will iterate through each self.pkg\_values’ \_# field. For each field that has a “self.pkg\_value.#.pkg\_src\_file\_pattern”, it should separate the fields by pipe to search for each value of “pkg\_src\_file\_pattern” recursively in each subdirectory starting from the pkg\_src\_path to generate the pkg\_output\_name file.
6. The package() function should also handle the split by count # or size as well as the password. These functionality requirements should preferably come from the Python library or an open source Python module. The package() function should also validate that the ‘pkg\_util’ has the capabilities to support the split by or password features. If it does not, it should error out and tell the user split by count/size or defining a password is not available for the selected ‘pkg\_util’.
7. At the end there should be a validation for the total count of files in the packaged file and the files in the system. This is to make sure that no files are left out (there has been cases were zip totals don’t match and we didn’t know until it got to the client).
8. If there is a selection, use the highest degree of compression by default.
9. If time allows, we may look into implementing the package() function in a multi-threading fashion such that multiple threads are packaging individual folders.

**Configuration Rules:**

The below is a representation of the configurations using ENTC as a placeholder CID.

* The fields **“pkg\_src\_file\_pattern”**, **“pkg\_src\_path”** and **“pkg\_output\_name”** should support multiple folders/outputs. This means that the \_# must to appended to each of the 3 fields listed.
* Packaging field **“pkg\_src\_file\_pattern”** is pipe delimited for all valid patterns. The program will need to iteratively look for all files for each field In all possible subdirectories starting from the **“pkg\_src\_path”.**
* If **“pkg\_src\_file\_pattern”** is not declared in the configs, the program should package the entire tree structure from **“pkg\_src\_path”** as the **“pkg\_output\_name”**
* Packaging fields **“pkg\_src\_path”**, **“pkg\_output\_name”** and **“pkg\_src\_file\_pattern”** will be in a 1:1:1 relationship.
* The other exception to the breakage of the 1:1:1 relationship is if the **“pkg\_src\_file\_pattern”** is not supplied. This means that the \_# without a **“pkg\_src\_file\_pattern”** will package all files within the folder path.
* Packaging fields **“pkg\_src\_path”** and **“pkg\_output\_name”** have the ability to use variable fields surrounded by << >>. These fields will be passed in when calling afp2pdf.py as a key value pair similar to the format used for **“InputFilename”** and **“OutputFilePath”**. These will be translated when executing the package() method call.
* **pkg\_src\_file\_pattern\_[#]:** If this field is not declared, the program will zip all files located in the pkg\_src\_path[#]
* If a packaging utility does not support a specified feature such as splitting, password protection, etc. exit out and issue a fatal error with the message “ERROR: Feature $feature not supported with $pkg\_util” .
* If a package is split, include the index file into the first split file. TODO: Accommodate split handling for segmented index files.

**Configuration Example:**

pkg\_util = zip

pkg\_src\_file\_pattern\_1 = entc\_archival\_\*\_\*.pdf | entc\_idx\_file.txt # Put \* for any variable arguments.

pkg\_src\_path\_1 = << d\_dir >>/pdf\_afp/entc/entcms1/pdf/<< cycle >>\_disk01/

pkg\_src\_path\_2 = << d\_dir >>/pdf\_afp/entc/entcms1/pdf/<< cycle >>\_disk02/

pkg\_output\_name\_1 = << d\_dir >>/pdf\_afp/entc/entcms1/zip/ ENTC\_ZIP\_DISK1\_<< cycle >>\_<< segment >>\_<< sequence >>.zip

pkg\_output\_name\_2 = << d\_dir >>/pdf\_afp/entc/entcms1/zip/ENTC\_ZIP\_DISK2\_<< cycle >>\_<< segment >>\_<< sequence >>.zip

pkg\_output\_ignore\_subdir = y # If yes, junk the path past “pkg\_src\_path”.

pkg\_output\_split\_size = count|10000 # Can accept size in similar format

pkg\_output\_password = 12345678 # Password value

**Supporting Updates:**

A translation class (**VariableTranslator** inside **afp2pdf\_variable\_translator.py**) will need to be built to handle the variables enclosed in << >>. The VariableTranslator will be created in a pre\_process() method created in **afp2pdf\_direct\_call.py** (similar to post\_process()).

* The translator class will check for << >> in the **self.data** fields and attempt to find a matching key inside self.data (ie. self.data[variable])
* If the variable does not exist, check the environment variables for the variable value.
* If the variable is not defined in self.data or as an environment variable, error out and let the user know that the variable cannot be resolved.

A supporting update must be made to the core program **afp2pdf\_index\_parser.py** and **afp2pdf\_index\_file\_builder.py** to support output generation into subdirectories.

* If output needs to be redirected into a subfolder, the GROUPINDEX field **OUTPUT\_FILE\_PATH** must be declared.
* If a variable cannot be resolved on the DFA level (ie. cycle or segment) the GROUPINDEX field **OUTPUT\_FILE\_PATH** can contain values enclosed in << >> to be resolved by the processing script. This should match with that is contained in the **“pkg\_src\_path”**.
* ~~Do a translation for all fields nested in << >> before moving the PDF file. If a field does not have a value, error out.~~
* When performing the PDF renaming step under **‘rename\_idx\_pdf\_after\_parse’**, rename the file under the path specified with **OUTPUT\_FILE\_PATH** if OUTPUT\_FILE\_PATH is declared. Otherwise, use the existing logic.
* If the path does not exist, create the path before renaming.

**Project Update – AFP2WEB Delivery Module**

**Business Requirements:**

* Develop encryption support:

The encryption feature should be able to encrypt a file into any extension (pgp by default). It should be able to act on files regards of type (ie. zip, pdf, xml).

* Support for placing files:

The feature can place the pdf archive files in the following destination:

* + InfoIMAGE FTP server (/d/infoftps#/CID/client\_pickup)
  + Remote FTP server outside of InfoIMAGE via the following FTP protocols
    - SFTP
    - FTPS (FTP overssl)
* Email notification default template informing the recipient that PDF archival has generated. There is an option to override the email title and content if client wants a non-standard template.
* To use the delivery module, the user must use the file delivery feature. The encryption and email notification features are optional depending on each application’s setup specifications.

**Technical Requirements:**

* Similar to the packaging module, there should be an option for afp2pdf.py that (when passed) will allow the user to use the delivery module without having to run an AFP2WEB conversion.
* The encryption feature will need to validate that an encryption key is supplied.
  + **Validate keyring**
  + **Validate passed via configuration**
* If FTP push is selected, it should check for a valid username@hostname and password fields.
* The email notification piece will utilize the existing Python mail program daily\_mail.py. The only thing it should do additionally is preparing the header/content of the email.

**Configuration Example:**

dlv\_Infile\_pattern = << d\_dir >>/pdf\_afp/demo/demoms1/zip/ DEMO\_ZIP\_DISK\*zip

dlv\_pickup\_server\_1 = infoftps3

dlv\_pickup\_full\_path\_1 = << d\_dir >>/infoftps3/osiw/demo/client\_pickup/

dlv\_delivery\_util\_2 = sftp

dlv\_delivery\_cred\_2 = infoimage@guidancefinancial.com

dlv\_delivery\_password\_2 = 123456

dlv\_encrypt\_ext = pgp

dlv\_encrypt\_key = guidance\_financial@guidancefinancial.com

dlv\_email\_mail\_list = external\_pdf\_maillist

dlv\_email\_custom\_header = “Guidance Financial PDF Archival Files are Ready”

dlv\_email\_custom\_content = “Custom content goes here…”

dlv\_email\_custom\_mail\_file = << home\_keep >>/keep/maillist/demoms1\_mail.lis

**Configuration Rules:**

* The **‘dlv\_infile\_pattern’** and a delivery method field (**‘dlv\_pickup\_server’**/**’dlv\_pickup\_full\_path’**/**’dlv\_delivery\_util’**) are the only two required fields. If either are missing, the program should error out.
* A **‘\_#’** can be applied to the **‘dlv\_pickup\*’** or **‘dlv\_delivery\*’** options to support multiple forms of delivery per application.
* The **‘dlv\_pickup\_full\_path’** may be used in place of the **‘dlv\_pickup\_server’**. This is to support cases where the **‘dlv\_pickup\_full\_path’** is not the standard ${d\_dir}/${server}/${cid}/client\_pickup/.
* If the **‘dlv\_delivery\_util’** is specified the **‘dlv\_delivery\_cred’** and the **‘dlv\_delivery\_password’** must be specified. If they are not, the program should error out without trying to establish a collection with the remote server.
* If an **‘dlv\_encrypt\_ext’** is provided a **‘dlv\_encrypt\_key’** must also be provided.
* The **‘dlv\_encrypt\_ext’** will be appended to each of the files matching the **‘dlv\_infile\_pattern’**.
* If the **‘dlv\_email\_mail\_list’** is provided, email notification is turned on for the application. Otherwise, the job will not have email notifications.
* By default, the program will check for the maillist recipients in /home/keep/maillist/${prefix}\_mail.lis. If an **‘dlv\_email\_custom\_mail\_file’** is declared, it will use this file instead. If no matching mail list file is found, the program should error out.
* **‘dlv\_email\_custom\_header’** and **‘dlv\_email\_custom\_content’** can be used to replace the default message.