Fiery JobFlow Cookbook

Fiery JobFlow Cookbook
- Introduction
- Fiery JobFlow Connect
- JobFlow Connect concepts
- Connect Scripting
- Connect advanced scripts
- Connect Packages
  - settings.json
  - Python script example
  - NodeJS script example
  - Powershell script example
- Connect Master Variables
  - JobFlow Vault
  - Master Variables examples
- Tips and tricks
  - Print a job to a JobFlow workflow
  - Check your batch code
  - Enfocus PitStop XML reports
  - ImageMagick and CPDF are bundled
- Fiery JobFlow Ticket
- Ticket examples

Introduction

The Connect module enables the use of third-party file processing solutions to address challenging workflows that need automation functions that JobFlow can't provide. This cookbook covers the best practices and tips on how to write scripts to extend workflow capabilities such as convert non-standard formats to PDF, edit PDF files, send notifications through third-party services, and much more.

Fiery JobFlow 2.2 introduced a new module called "Connect" with which JobFlow users can seamlessly connect 3rd party applications via scripting or hot folders. In addition JobFlow Rules can now also trigger a script to support notifications via 3rd party solutions. Fiery JobFlow Connect is available in the paid version of Fiery JobFlow.

With the release of JobFlow 2.3 managing scripts was improved via so "Connect Packages". A JobFlow Connect Package is an archive that contains all resources required to execute a script. Additional metadata can be added to allow customer input, provide developer info and detailed information that is visible in Fiery JobFlow Resources.

JobFlow 2.5 introduced Connect Master Variables with which Preflight, Correct, Connect and Output module settings could dynamically be updated to create more intelligent workflows.

With JobFlow 2.7 we made it easier to create your scripts using other batch language like Python and NodeJS. We also added the ImageMagick and Coherent PDF libraries to manipulate images and PDFs.

Fiery JobFlow Connect

Some examples the JobFlow Connect module can be used for:

- Convert native jobs to a format that Fiery JobFlow supports. For this the Connect module can be added to a workflow immediately after the Input module but before the Convert module.
- Modify PDF jobs as a normal step in a JobFlow workflow. The Connect Module can be placed anywhere in a workflow.
- Update Fiery job properties based on content found in a PDF.

JobFlow Connect concepts

JobFlow supports 3 methods to integrate with 3rd party applications.

1. JobFlow sends a job to a 3rd party application's hot folder and waits for the result.
2. Directly call a script that tells the 3rd party application what to do.
3. Create your own module with a JobFlow Connect Package.

The first method to send a Fiery JobFlow job is via hot folders. In the Connect module you can tell JobFlow where to move the job and where to wait for the result. An error time out can also be defined to prevent JobFlow from waiting eternally for a job to show up. Please note the input and output locations are relative to the JobFlow server.

In the following examples the input and output locations are hot folders on the JobFlow server.

```
c:\hotfolders\input
e:\hotfolders\output
```

Network locations are supported by using the standard Windows notation for network locations:

```
\\server\hotfolders\input
\\server\hotfolders\ouput
```

Connect Scripting

The second method for file conversion and manipulation is via scripting. But before we continue…

Scripting assumes a certain skill and knowledge level from a JobFlow user. Users need to be familiar with the basics of scripting and understand concepts such as arguments. We do not assume any responsibility for anything bad that might happen as a result of a poorly written script.

We have tried to make it as easy as possible for Fiery JobFlow users to implement scripts. In order to support user defined scripts, Fiery JobFlow will provide all the necessary information required to process a job via a script as effective as possible. This information is provided by JobFlow feeding the script with so-called arguments.

Fiery JobFlow provides the following arguments:

1. Input Location: a temporary location where JobFlow will make the file available for processing.
2. Output Location: a temporary location where JobFlow will wait for the processed file to be copied.
3. Job Name: the job name as shown in JobFlow.
4. Workflow: the name of the workflow where the script was initiated.
5. Preflight Report: location of the last preflight report generated in that workflow.
6. Workflow Vault: temporary location created by Fiery JobFlow that exists as long as a job exists in a workflow. If a workflow contains multiple scripts this location can be used to share data between scripts.
7. Next module ID: the ID for the next module that follows the Connect module. This ID is needed for working with JobFlow Master Variables.

Arguments don't have names and the script needs instructions to know the order to interpret the arguments.

Important note: arguments provided by the user always come first!

Most of the scripts shown in this document have a similar structure:

1. Set the main variables used in the script.
   These variables are set by the parameters that are provided to the script via JobFlow.
2. Check if utilities run via a script are installed.
   If they are not installed the script stops and logs the error in a log file that will be written in the folder where the script is running.
3. The main procedure that needs to be executed for this script.

An example of simple script.

```
:: Disclaimer: scripts are provided as-is and most of them use tools that are open source
:: or require a (small) license fee when used in a commercial environment. It is the
:: responsibility of the end user that use scripts to respect the license agreement for
:: these tools. EFI can not be held responsible in any way if scripts don't behave like
:: they should do resulting in any form of financial loss or downtime. Have a good day.

:: [Description, link to 3rd party app]
:: [Author], [Date]

:: JobFlow arguments, information that JobFlow makes available to the script
set INPUT=%~1
set OUTPUT=%~2
set JOBNAME=%~3
set WORKFLOW=%~4
set PREFLIGHT=%~5
set VAULT=%~6
set NEXTMODULEID=%~7

:: Static variables, information that we use to make the script easier to manege
:: Location of this script
set pathScript=%~dp0
:: Name of this script
set ScriptName=%~n0
:: File path where we want to save the log file. We like to use JobFlow Vault.
set pathLog=%VAULT%\%ScriptName%.log
:: File path where we will look for any additional applications we want to use
set pathInclude=%pathScript%\_include

:: Helper Apps
:: #scriptname#
:: Where in the _include path can we find the application we need?
set pathApp=%pathInclude%\app.exe
:: Make sure the application exists
if not exist %pathApp% (
    >>%pathLog% echo %DATE% %TIME% [ERROR] Cannot find application at %pathApp%. This is serious.
Exit script.
    exit /b 1
)
```

```
:: This is where we process the JobFlow job with the 3rd party app.
:: Note: this command is an _example_ and is different for every app

:do_something
%pathApp% %INPUT% %OUTPUT%\%JOBNAME%.pdf
:: Check if something worked
if %ERRORLEVEL% NEQ 0 (
  >>%pathLog% echo %DATE% %TIME% [ERROR] An error occured while doing something. Exit script.
  exit /b 1
)
>>%pathLog% echo %DATE% %TIME% [INFO] We processed %INPUT% with %pathApp% to
%OUTPUT%\%JOBNAME%.pdf

:: We're done. Back to JobFlow.
:success
exit /b 0
```

In JobFlow, the script would be called as follows in the Connect module script settings: `c:\scripts\simple.bat` The 3rd party application has to exist at `c:\scripts\_include\app.exe`

Connect advanced scripts

The following example introduces user provided arguments. This makes the script more powerful as the behavior can be changed without having to create or update the script.

As mentioned before, user provided arguments precede the JobFlow provided arguments so the order in which arguments are read by the script needs to reflect this.

In this script example a user provided value is checked and used to set the 3rd party app behavior.

```
:: Disclaimer: scripts are provided as-is and most of them use tools that are open source
:: or require a (small) license fee when used in a commercial environment. It is the
:: responsibility of the end user that use scripts to respect the license agreement for
:: these tools. EFI can not be held responsible in any way if scripts don't behave like
:: they should do resulting in any form of financial loss or downtime. Have a good day.

:: [Description, link to 3rd party app]
:: [Author], [Date]

:: User provided arguments
set txtValue=%~1
:: Manage user provided arguments. By using this trick we can manage arguments a lot easier
shift /1

:: JobFlow arguments, information that JobFlow makes available to the script
set INPUT=%~1
set OUTPUT=%~2
set JOBNAME=%~3
set WORKFLOW=%~4
set PREFLIGHT=%~5
set VAULT=%~6
set NEXTMODULEID=%~7

:: Static variables, information that we use to make the script easier to manage
:: Location of this script
set pathScript=%~dp0
:: Name of this script
set ScriptName=%~n0
:: File path where we want to save the log file. We like to use JobFlow Vault.
set pathLog=%VAULT%\%ScriptName%.log
:: File path where we will look for any additional applications we want to use
```

```
set pathInclude=%pathScript%\_include

:: Check user provided arguments
if /I "%txtValue%" EQU "" (
    >>%pathLog% echo %DATE% %TIME% [ERROR] User value cannot be blank. Exit script.
    exit /b 1
)

:: Helper Apps
:: #scriptname#
:: Where in the _include path can we find the application we need?
set pathApp=%pathInclude%\app.exe
:: Make sure the application exists
if not exist %pathApp% (
    >>%pathLog% echo %DATE% %TIME% [ERROR] Cannot find application at %pathApp%. This is serious.
Exit script.
    exit /b 1
)

:: This is where we process the JobFlow job with the 3rd party app.
:: Note: this command is an _example_ and is different for every app.

:: It is advisable to wrap all path variables with quotations to ensure path with spaces work
:do_something
"%pathApp%" "%INPUT%" %txtValue% "%OUTPUT%\%JOBNAME%.pdf"
:: Check if something worked
if %ERRORLEVEL% NEQ 0 (
  >>%pathLog% echo %DATE% %TIME% [ERROR] An error occurred while doing something. Exit script.
  exit /b 1
)
>>%pathLog% echo %DATE% %TIME% [INFO] We processed %INPUT% with %pathApp% to
%OUTPUT%\%JOBNAME%.pdf

:: We're done. Back to JobFlow.
:success
exit /b 0
```

In JobFlow, the script would be called as follows in the Connect module script settings: `c:\scripts\advanced.bat "user value"` The 3rd party application has to exist at `c:\scripts\_include\app.exe`

Connect Packages

JobFlow scripts are a powerful way to extend JobFlow functionality and make the workflow suit perfectly the requirement of the end user. As long as all the required components are copied in the correct location the end user understand command line arguments and every other detail related to the script, it works just fine.
If that is not the case, however things easily break and one spends a long time with theend user trying to get the script working.

To make JobFlow Connect scripting even more powerful and **much** easier to use, we make use of Connect Packages.

A JobFlow Connect Package is a zip archive that contains the following resources:

- The main script.
- The required, 3rd party applications needed to run the script.
- A JSON document called `settings.json` that contains additional metadata about the module. This document can also include variables that the end user can override via the module settings in JobFlow.

Sample connect package content:

- connect-name.zip
    - settings.json
    - script.bat
    - app-folder
        - app.exe
        - app.dll
        - app.readme

The JobFlow Connect Package can be shared with end users who can import the package in JobFlow via `Admin->Resources`. JobFlow will then import the resources and add the module to the Connect module "Connect Package" dropdown menu. Resources are copied to the `JobFlowImportedScripts` folder on the system where JobFlow is running.

settings.json

Connect Package metadata is defined in a JSON file called `settings.json`.

The JSON contains, at minimum, the following metadata properties:

1. The name of the Connect Package as it would be shown to the end user in the Connect Package dropdown menu in JobFlow.
2. A developer ID. There is no strict rule on how one defines a developer ID. The developer ID is used to separate Connect Packages from different developers when resources are copied to the `JobFlowImportedScripts` folder. If your developer ID is `jobflownerd`, the resources and script are copied to `JobFlowImportedScripts\jobflownerd`.
3. The main script that needs be executed when the module is processing a file. Path to the script is relative to the

`JobFlowImportedScripts\developer-id` folder.

4. An email address and website of the main developer. These are not mandatory but useful for the end user in case one has questions about a Connect Package.

An example of a simple JSON settings file:

```
{
 "name": "Description of JobFlow Connect package. This is what the end user sees in the dropdown menu",
 "dev_id": "efi.jobflow.[developerid]",
 "description": "More detailed description. If needed, inform the end user about licenses used in this module",
 "script": "script.bat",
 "email": "[developer email]",
 "url": "[developer url]"
}
```

Connect Package metadata can also contain user provided data that are passed on to the main scripts as arguments. In the following example we add parameters that show a dropdown menu with the values "Option 1", "Option 2" and "Option 3" and a text field with the predefined value "This is a test". Script arguments for this Connect Package are provided as `script.bat "Option 1" "This is a test"` and must be handled accordingly in `script.bat`.

```
{
 "name": "Description of JobFlow Connect package. This is what the end user sees in the dropdown menu",
 "dev_id": "efi.jobflow.[developerid]",
 "description": "More detailed description. If needed, inform the end user about licenses used in this module",
 "script": "script.bat",
 "email": "[developer email]",
 "url": "[developer url]",
 "params": {
  "Dropdown Menu": ["Option 1","Option 2","Option 3"],
  "Text Input": "This is a text"
 }
}
```

Script examples shared so far, are "basic" command line Windows batch files. To support more powerful and capable scripting languages `settings.json` can include an absolute or relative location of the script interpreter like Python, NodeJS or Powershell.

The following example calls the main script as follows: `powershell.exe -ExecutionPolicy Bypass -file pkgToCopy.ps1`

```
{
 "name": "Powershell Script",
 "dev_id": "efi.jobflow.cookbook",
 "description": "This is sample connect package using Powershell",
 "script": "pkgToCopy.ps1",
 "email": "jobflow@efi.com",
 "url": "http://efi.com/fieryjobflow",
 "params": {
  "RUNTIME": "Powershell.exe",
  "OPTION": "-ExecutionPolicy Bypass -file"
 }
}
```

Python script example

`settings.json` that accompanies script sample. Script runs as `c:\python-3.6.3\python copy.py`

```
{
 "name": "Python script example. Expects Python runtime pre-installed.",
 "dev_id": "efi.jobflow",
 "description": "This is sample Connect package using Python",
 "script": "copy.py",
 "email": "jobflow@efi.com",
 "url": "http://efi.com/fieryjobflow",
 "params": {
  "RUNTIME": "C:\\python-3.6.3\\python.exe"
 }
}
```

```
#!/usr/bin/python
import sys
print ("script name: ", sys.argv[0])
print ("number of arguments: ", len(sys.argv))
print ("arguments: " , str(sys.argv))

import os
import shutil

source = sys.argv[1]
target = sys.argv[2]
print ("source: ", source)
print ("target: ", target)

# adding exception handling
try:
    shutil.copy(source, target)
    print("File copy successful to copy file. %s",sys.argv[1] )
except IOError as e:
    print("Unable to copy file. %s" % e)
except:
    print("Unexpected error:", sys.exc_info())
```

Alternative `settings.json` when Connect Package has Python bundled.

```json
{
 "name": "Python script example. Python runtime is bundled",
 "dev_id": "efi.jobflow",
 "description": "This is sample Connect package using Python",
 "script": "copy.py",
 "email": "jobflow@efi.com",
 "url": "http://efi.com/fieryjobflow",
 "params": {
  "RELATIVE-RUNTIME": "\\python-3.6.3\\python.exe"
 }
}
```

NodeJS script example

`settings.json` that accompanies script sample. Script runs as `c:\Program Files\nodejs\node.exe copy.js`

```json
{
 "name": "Node script example. Expects Node runtime pre-installed.",
 "dev_id": "efi.jobflow",
 "description": "This is sample Connect package using Python",
 "script": "copy.js",
 "email": "jobflow@efi.com",
 "url": "http://efi.com/fieryjobflow",
 "params": {
  "RUNTIME": "c:\\Program Files\\nodejs\\node.exe"
 }
}
```

```javascript
// Sample Node js package to copy source file to destination

const fs = require('fs');
const path = require('path');

var myArgs = process.argv.slice(2);

console.log(`number of arguments: ${myArgs}`);

var source = myArgs[0];
// Append file name to output folder
var target = path.join(myArgs[1], path.basename(myArgs[0]));

console.log(`source : ${source}`);
console.log(`target : ${target}`);

// Output will be created or overwritten by default.
fs.copyFileSync(source, target);
console.log(`${source} was copied to ${target}`);
```

Powershell script example

`settings.json` that accompanies script sample. Script runs as `powershell -ExecutionPolicy Bypass -file copy.ps1`
Note: Windows 10 ships with Powershell 5.1 which is *not the latest version*. Microsoft currently promotes Powershell 7 which can run side by side with Powershell 7. More info at https://docs.microsoft.com/en-us/powershell/scripting/install/migrating-from-windows-powershell-51-to-powershell-7?view=powershell-7.1. After installing Powershell 7, use `pwsh.exe` as the runtime.

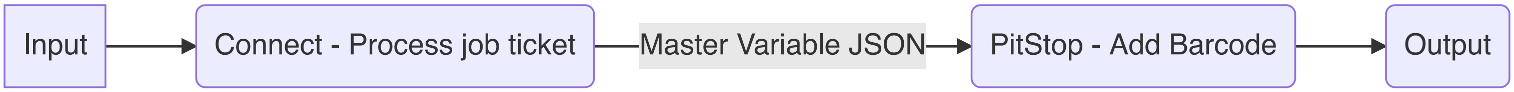
```
{
 "name": "Powershell script example.",
 "dev_id": "efi.jobflow",
 "description": "This is sample Connect package using Windows Powershell",
 "script": "copy.ps1",
 "email": "jobflow@efi.com",
 "url": "http://efi.com/fieryjobflow",
 "params": {
  "RUNTIME": "powershell.exe",
  "OPTION": "-ExecutionPolicy Bypass -file"
 }
}
```

```
write-host "script name: $($myInvocation.Path)"
write-host "number of arguments: $($args.count)"
write-host "arguments: $($args)"

$source = $args[0]
$target = $args[1]

write-host "source: $($source)"
write-host "target: $($target)"

Copy-Item -Path $source -Destination $target
if(Test-Path -Path $target) {
   write-host "File copied"
}else {
   write-host "File not copied"
}
```

Connect Master Variables

JobFlow Master Variables allow a Connect module to override variables for the *next* module.

A JobFlow Master Variable is a JSON file named `params_id.json` a script creates in the JobFlow Vault location. `id` is the JobFlow module ID of the next module which is provided by JobFlow automatically as a script argument.



JobFlow Vault

JobFlow Vault is a hidden temporary folder, automatically created by JobFlow, that exists as long as the job exists in JobFlow. The hidden folder called `.FieryJobFlowExternal` located on the system where JobFlow runs. When a user deletes a job in JobFlow, the temporary Vault folder is also deleted. JobFlow Vault is a great helper feature to temporarily store data that can be shared between modules. In this cookbook, script examples use JobFlow Vault to save log output of applications and scripts. JobFlow will also automatically save Enfocus PitStop reports created for a PDF in JobFlow Vault as an XML report. These XML reports are a great resource for getting additional data of a job that can be used in a Connect module script.

Master Variables examples

Example uses for using Master Variables are:

1. You have a PitStop Action module that adds a barcode to a page. The content for that barcode is an Enfocus PitStop Smart Variable called `txtText`. Use the Connect module to create a Master Variable that sets the barcode content for the PitStop barcode action by overriding the default value for `txtText`. The content of the JSON Master Variables file should be `{"txtText":"New barcode content"}`.
(Note: this guide does not cover how Enfocus PitStop Smart Variables work. For more info about this topic, please visit https://www.enfocus.com/Manuals/UserGuide/SW/11U4/Switch/en-us/concept/c_getting_started.html)
2. Master Variables can override Fiery Job Properties! By creating a JobFlow Master Variable with the correct PPD key (for instance `Notes1`) you can have JobFlow set the content for the notes field for that job on the Fiery. To override the content for the Notes 1 property, the content of the Master Variables JSON is `{"Notes1":"Custom notes content"}`
Other examples include having the ability to set a page range for a specific media or chapters based on job content.
3. JobFlow Connect modules can override settings of other JobFlow modules. For converting images, Master Variables can set additional parameters for the image conversion settings in the Convert module. Settings for page ranges in the Pages module can also be overridden by Master Variables.

Example of a script that overrides the page range for the Page module. **It is essential that this script runs immediately before the Pages module.**

```
:: Fiery JobFlow provided arguments
set INPUT=%~1
set OUTPUT=%~2
set JOBNAME=%~n3
set VAULT=%~6
set NEXTMODULEID=%~7

:: Script variables
set ScriptName=%~n0
:: Save script log output in vault
set pathLog=%VAULT%\%ScriptName%.log
:: Set master variables location and file name
set pathJSON=%VAULT%\params_%NEXTMODULEID%.json

:: Override page number range for pages module
@echo {"page_number_range":"1,3,5-10"}> %pathJSON%

:: Make sure the JSON file has been created
if not exist %pathJSON% (
    >>%pathLog% echo %DATE% %TIME% [ERROR] Cannot find %pathJSON%.
)

:: Move PDF to next module
copy /y "%INPUT%" "%OUTPUT%

:: Done
```

Tips and tricks

Print a job to a JobFlow workflow

In use cases where JobFlow needs to process print streams like PCL or Postscript, often a print-driver based workflow is required. The good news is that with a few simple steps one can add printer ports in Windows 10 that save the output in a JobFlow Smart Folder.

1. Install Multi File Monitor from https://sourceforge.net/projects/mfilemon/.
2. Open `printmanagement.msc` in Windows 10.
3. Add new port, select Multi File Port Monitor and name the port. Port name is arbitrary but wise to choose a descriptive name.
4. Port configuration settings:
   - Output path: `c:\SmartFolders\WorkflowName`. Needs to be an existing JobFlow workflow.
   - Filename pattern: `%t`.
     No other settings need to be configured.
5. Add printer with this new port.

Check your batch code

IDE's like Visual Studio Code have excellent support for script languages like Python, NodeJS and Powershell. While working with common windows batch scripts is fine in VSCode it lacks features like syntax checking. Fortunately Rob van der Woude, of the **fantastic https://ss64.com website has created a tool called BatCodeCheck that allows you to check your batch code. More info at: https://www.robvanderwoude.com/battech_batcodecheck.php
Run it as `BatCodeCheck.exe drive:\path\batchfile.bat /L /W` to get a report with suggested improvement or `BatCodeCheck.exe drive:\path\batchfile.bat /I` to run the tool in interactive mode.

Enfocus PitStop XML reports

JobFlow will save Enfocus PitStop XML based reports in JobFlow Vault automatically.

ImageMagick and CPDF are bundled

JobFlow 2.7 bundles ImageMagick and Coherent PDF libraries for image and PDF manipulation. The bundled binaries can be used in Connect modules.

Bundled ImageMagick sample script

```
:: Use the version of ImageMagick that ships with JobFlow 2.6 Note IM arguments can also
:: be set by a Master Variable. Sample is shared in case more complex processing is needed.

:: Fiery JobFlow provided arguments
set INPUT=%~1
set OUTPUT=%~2
set JOBNAME=%~n3
set WORKFLOW=%~4
set PREFLIGHT=%~5
set VAULT=%~6
set NEXTMODULEID=%~7

:: Static arguments
set pathScript=%~dp0
set nameScript=%~n0
set pathLog=%VAULT%\%nameScript%.log
set pathInclude=%pathScript%\_include

:: Let's find where ImageMagick is located
::
:: Step 1 - Get JobFlow Windows Registry Key data
if defined PROGRAMFILES(X86) (
 set KEY_NAME=HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\EFI\Fiery JobFlow
) else (
 set KEY_NAME=HKEY_LOCAL_MACHINE\SOFTWARE\EFI\Fiery JobFlow
)
:: Step 2 - Get install path
FOR /F "skip=2 tokens=1,2*" %%A IN ('REG QUERY "%KEY_NAME%" /v "Path" 2^>nul') DO (
 set ValueName=%%A
 set ValueType=%%B
 set ValueValue=%%C
)
set pathIM="%ValueValue%\Neem\Neem\app\plugins\convert\image_magick\bin\convert.exe"

if not exist %pathIM% (
 >>%pathLog% echo %DATE% %TIME% [ERROR] %pathIM% should exist but it doesn't! Exit script.
 exit /b 1
)

:: Run basic IM command
%pathIM% "%INPUT%" "%OUTPUT%\%JOBNAME%.pdf" > %VAULT%\imagemagick.log 2>&1
if %ERRORLEVEL% NEQ 0 (
  >>%pathLog% echo %DATE% %TIME% [ERROR] %pathIM% %argsIM% "%INPUT%" "%OUTPUT%\%JOBNAME%.pdf" failed. Exit script.
  exit /b 1
)

:success
echo %DATE% %TIME% [INFO] Done>>%pathLog%
exit /b 0
```

Bundled Coherent PDF sample script

```
:: Use the version of CPDF that ships with JobFlow 2.7

:: Fiery JobFlow provided arguments
set INPUT=%~1
set OUTPUT=%~2
set JOBNAME=%~n3
set WORKFLOW=%~4
set PREFLIGHT=%~5
set VAULT=%~6
set NEXTMODULEID=%~7

:: Static arguments
set pathScript=%~dp0
set nameScript=%~n0
set pathLog=%VAULT%\%nameScript%.log
set pathInclude=%pathScript%\_include

:: Check User Arguments
set argsCPDF=%argsCPDFsample%

if /i "%argsCPDFsample%" equ "use custom argument" (
 set argsCPDF=%argsCPDFcustom%
)
if /i "%argsCPDFcustom%" equ "" (
    >>%pathLog% echo %DATE% %TIME% [ERROR] Custom CPDF arguments are empty. Exit script.
    exit /b 1
)
>>%pathLog% echo %DATE% %TIME% [INFO] CPDF args: %argsCPDF%

:: Let's find where CPDF is located
::
:: Step 1 - Get JobFlow Windows Registry Key data
if defined PROGRAMFILES(X86) (
 set KEY_NAME=HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\EFI\Fiery JobFlow
) else (
 set KEY_NAME=HKEY_LOCAL_MACHINE\SOFTWARE\EFI\Fiery JobFlow
)
:: Step 2 - Get install path
FOR /F "skip=2 tokens=1,2*" %%A IN ('REG QUERY "%KEY_NAME%" /v "Path" 2^>nul') DO (
 set ValueName=%%A
 set ValueType=%%B
 set ValueValue=%%C
)
set pathCPDF="%ValueValue%\Neem\Neem\app\plugins\pages\bin\cpdf.exe"

if not exist %pathCPDF% (
 >>%pathLog% echo %DATE% %TIME% [ERROR] %pathCPDF% should exist but it doesn't!. Exit script.
 exit /b 1
 >>%pathLog% echo %DATE% %TIME% [ERROR] %pathCPDF% should exist but it doesnt. Exit script.
 exit /b 1
)

:: Run CPDF basic operation
%pathCPDF% "%INPUT%" -o "%OUTPUT%\%JOBNAME%.pdf" > %VAULT%\cpdf.log 2>&1
if %ERRORLEVEL% NEQ 0 (
  >>%pathLog% echo %DATE% %TIME% [ERROR] %pathCPDF% %argsCPDF% "%INPUT%" -o
"%OUTPUT%\%JOBNAME%.pdf" failed. Exit script.
  exit /b 1
)

:success
echo %DATE% %TIME% [INFO] Done>>%pathLog%
exit /b 0
```

Fiery JobFlow Ticket

Fiery JobFlow Ticket is a feature in Fiery JobFlow Base that allows users to submit jobs via a simple, text-based ticket with which they can define the order of jobs, number of copies per job and if jobs should be merged or not.

A JobFlow Ticket is a text file with comma separated values. Currently, JobFlow supports 2 columns: job location and number of copies. You do not need to add a header or name the columns. JobFlow always assumes job location is in column one and the number of copies is in column two. You can add comments in the ticket by beginning a line with '//'. Anything in a comment line will be ignored by JobFlow.

You can tell JobFlow to merge jobs in a ticket by adding an action. An action is prepended by the # character. Currently, JobFlow only supports the #merge command.

Ticket examples

Example of a ticket to retrieve jobs from a local folder or network location and set copies per job.

```
// Filename, NumCopies
c:\folder\Bedding Flowers.pdf,10
c:\folder\Direct Sow Flowers.pdf,10
c:\folder\Flowering Bulbs.pdf,5
\\networkdrive\Roses.pdf,20
```

Example of ticket which merges jobs in a ticket and set copies for merged job.

```
// Filename, NumCopies
#merge,10
c:\folder\Bedding Flowers.pdf
c:\folder\Direct Sow Flowers.pdf
c:\folder\Flowering Bulbs.pdf
\\networkdrive\Roses.pdf
```

Please note that if we merge jobs JobFlow automatically adds bookmarks for the first page of every job that is merged. The bookmark title is taken from the job name minus the file extension. These bookmarks can be used in Fiery JobMaster to automatically add tabs.

Example of ticket which merges jobs in ticket and set copies for merged job. Jobs and ticket are submitted as a folder or archive.

```
// Filename, NumCopies
#merge,10
Bedding Flowers.pdf
Direct Sow Flowers.pdf
Flowering Bulbs.pdf
Roses.pdf
```

Please note that if a folder or an archive with jobs is submitted, all jobs will be merged automatically. If you add the #merge command you can define the number of copies for the merged jobs.