DocumentBurster - Advanced report delivery scenarios

DocumentBurster - Advanced report delivery scenarios

Table of Contents

Manual Conventions

1. Path Separator

This manual uses slash character (/) to display directory and file components of a path.

Microsoft Windows can accept either the backslash (\) or slash (/) characters to separate directory and file components of a path, while the Microsoft convention is to use a backslash (\). Since *DocumentBurster* is intended to work on other operating systems (e.g. Linux) also, the convention is this manual is to use the slash character (/) to display directory and file components of a path.

Overview

The scope of this document is to show how *DocumentBurster* can be used to achieve more complex report delivery scenarios.

What to Expect

In this document, you'll learn how to

- Script *DocumentBurster* to achieve complex use cases
- Get yourself familiar with the sample scripts provided with *DocumentBurster*
- Upload reports using *cURL*

From time to time, some report distribution requirements might need to execute an external program during the report bursting lifecycle, distribute SMS messages, upload reports to enterprise portals, send reports by Fax or to print the output burst reports. In other situations it might be required to upload the reports using more secure protocols such as FTPs, SFTP or SCP. Before sending the reports, it is also possible to encrypt the output reports or stamp the distributed reports in much the same way that it is applied a rubber stamp to a paper document.

***DocumentBurster* scripting**

If required, *DocumentBurster* can be scripted in order to support more customized report distribution needs.

While it might look like an overkill to write scripts for doing report distribution, it is actually a very powerful and flexible approach. Further more, the default *DocumentBurster* software package it is coming with a set of already written scripts which can be used almost out of the box.

In most of the real life situations, just taking an existing script (appropriate for the task in hand) and doing very small adjustments (for example giving FTP host, user name and password) will be all that is required to achieve complex requirements.

***cURL* integration**

DocumentBurster does not reinvent the wheel and it integrates with *cURL* in order to achieve the most complex document distribution situations.

http://curl.haxx.se/

DocumentBurster does offer close integration with *cURL*, a command line tool for transferring data with URL syntax, supporting DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP. *cURL* supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password authentication (Basic, Digest, NTLM, Negotiate, kerberos...), file transfer resume, proxy tunneling and a busload of other useful tricks.

Quick & Professional Support

If you have any questions that aren't answered here or you need some special script to be developed, feel free to contact us: ***support@pdfburst.com***

Feedback

We welcome feedback on our products, including this manual. If you would like to make any suggestions for improving our products, please contact us: ***support@pdfburst.com***

Now let’s get started.

Chapter 1. Using Scripts to Achieve More

Scripts can help in squeezing more tailored functionality from *DocumentBurster*. For example, there is no GUI command to archive the output burst reports in a single compressed file, while with few lines of scripting it is easy to zip all the output files together.

DocumentBurster supports scripts written in Groovy, a scripting language for the Java platform. *DocumentBurster* Groovy scripts can make use of any existing Java code and library.

This chapter shows how to use the scripting capabilities of the software and how to customize *DocumentBurster* using some existing sample scripts which are provided with the package.

Scripting Scenarios

DocumentBurster has support for injecting tailored behavior during the normal bursting lifecycle. There are a set of predefined exit points in which, using scripting, it is possible to implement custom logic. For example there is an *endBursting* lifecycle phase in which, with few lines of code, it is possible to zip together all the burst files, which otherwise would have come separated in the output folder.

Following should give some ideas of the kind of things which are possible using *DocumentBurster* scripting capabilities:

File Related Capabilities

- *Copy* - Copy a file or a set of files to a new file or directory.
- *Delete* - Deletes a single file, all files and sub-directories in a specified directory, or a set of files specified with an wildcard (*) like file pattern.
- *Mkdir* - Creates a directory. Non-existent parent directories are created, when necessary.
- *Move* - Moves a file to a new file or directory, or a set(s) of file(s) to a new directory.
- *Archive* - Zip, GZip, BZip2 or Tar the burst reports.
- *Other file related capabilities* - Change the permissions and/or attributes of a file or all files inside the specified directories, generate or verify a checksum for a file or set of files and also touch the files.

Sample

For an example on how to zip or delete files, please see the existing *scripts/burst/samples/zip.groovy* sample script.

Execute External Programs

While integrating *DocumentBurster* with existing software, following capability will be of interest. It is possible to call any external executable in some pre-defined points during the report bursting and report distribution flow.

Exec - Execute a system command. When the OS attribute is specified, the command is only executed on one of the specified operating systems.

Sample

The external program to be demonstrated is *Pdftk - http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/*

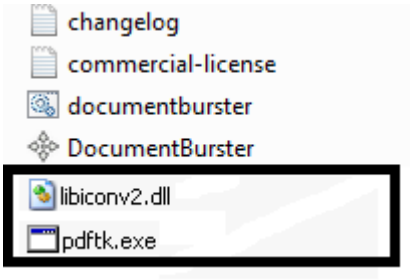
pdftk or the *pdf toolkit* is a cross-platform tool for manipulating PDF documents.

It is easy to execute *pdftk* from within *DocumentBurster* in order to achieve a wide range of additional powerful capabilities.

pdftk is capable of splitting, merging, encrypting, decrypting, uncompressing, recompressing, and repairing PDFs. It can also be used to manipulate watermarks, metadata, and to fill PDF Forms with FDF Data (Forms Data Format) or XFDF Data (XML Form Data Format).

Install *Pdftk*

- Please download *pdftk* from this location - *http://www.pdflabs.com/docs/install-pdftk/*
- Make sure to download the binaries which are specific to the target operating system.
- Copy the `pdftk.exe` and `libiconv2.dll` in the folder where *DocumentBurster* was installed, next to the `DocumentBurster.exe` file.



Under *Microsoft Windows*, `pdftk.exe` and `libiconv2.dll` should be placed next to the `DocumentBurster.exe` file.

For an example on how to execute *pdftk* during the report bursting lifecycle, please see the existing *scripts/burst/samples/exec_pdftk_background.groovy* sample script.

Publish Reports to Microsoft SharePoint Portal

Using scripting, it is possible to publish reports directly to enterprise portals. Think to the use case where there are few hundreds or thousands of customers and dealers and, with a single click, the relevant individual reports can be made available to each one of them on the portal.

DocumentBurster is distributing the reports to portals using the WebDAV protocol. Following products, they all support WebDAV, so that *DocumentBurster* is capable to distribute reports to the following:

- *Microsoft SharePoint*
- *IBM WebSphere Portal*

- *Oracle Portal*
- *SAP NetWeaver*
- *Tibco PortalBuilder*
- *Samsung ACUBE Portal*
- *Liferay Portal, Hippo portal, JBoss Enterprise Portal, eXo and Apache Portal*

Distribute by SMS and Fax

- *SMS messages* can be delivered, via email, through an online SMS gateway service. In such a scenario *DocumentBurster* is configured to send an email to the SMS gateway in which the text of the message and the destination number are specified. The SMS gateway will transform the email message and will deliver it further, using SMS, to the specified number. Using scripting, *DocumentBurster* can send configured SMS messages to any gateway service. For a list of available online SMS gateways just Google for *'list of SMS gateways'* . The SMS which is best fitting the needs can be selected and *DocumentBurster* will distribute SMS messages using it.
- *Fax the reports*

  There are various ways of sending documents by fax using the computer.

  The simplest way is to use an existing fax online gateway to which the reports are sent as an email attachment. The gateway will further forward the reports by fax to the specified number. For a list of available online fax gateways just Google for *'list of fax gateways'* .

  As an alternative, it is possible to send faxes by configuring a dial-up modem to work with specialized fax software. Microsoft Fax can be used as a fax software service on Windows. For instructions on enabling Microsoft Fax, please consult the appropriate Microsoft knowledgebase article from the Microsoft website. HylaFAX or AsterFax™ - Asterisk Fax are valid fax software solutions which can be used on UNIX/Linux systems. Using scripting, it is possible to integrate *DocumentBurster* with any of the previously enumerated fax products and this requires some customization effort to integrate with the specific fax vendor APIs.

Print Reports

DocumentBurster can print the output burst reports directly to physical printers.

Sample

For an example on how to print the output burst reports, please see the existing *scripts/burst/samples/print.groovy* sample script.

Mail, FTP, FTPs and SFTP

With a little bit of scripting it is possible to send reports by email, upload to FTP or FTPs and copy files to SFTP using SSH.

While sending the burst reports by email is available through the GUI interface, sometimes more flexibility can be achieved with the help of *DocumentBurster* scripting. One example is that using scripting it is possible, if required, to send emails without attachments to any SMS gateway - by default, through the GUI interface, all the emails which are sent will have a corresponding burst report attached.

Mail Sample

For an example on how to send an ad-hoc email during the report bursting flow, please see ***scripts/burst/samples/ant_mail.groovy*** sample script.

FTP/FTPs/SFTP Samples

For examples on how to FTP, FTPs or SFTP reports using scripting, please see ***Chapter 2. cURL Integration.***

Upload Reports to a Shared Location

DocumentBurster can upload the generated reports to a network shared location.

Sample

For an example on how to upload the burst reports to a shared location, please see the existing *scripts/burst/samples/copy_shared_folder.groovy* sample script.

Encrypt or Stamp the Output Reports

Using scripting, *DocumentBurster* can encrypt the output reports. This feature is commonly used to prevent unauthorized viewing, printing, editing, copying text from the document and doing annotations. It is also possible to ask the user for a password in order to view the report.

Sample

For an example on how to encrypt and password protect the burst reports, please see the existing *scripts/burst/samples/encrypt.groovy* sample script.

DocumentBurster can stamp the distributed reports in much the same way that it is applied a rubber stamp to a paper document. If required, it is possible to apply bates stamping, page numbering, text stamping, logo insertion or add headers/footers and watermarks to the reports.

Sample

For an example on how to stamp the burst reports, please see the existing *scripts/burst/samples/overlay.groovy* sample script.

Introduction to the Burst Lifecycle

During the report processing, *DocumentBurster* defines a set of exit points which can be used to customize the default software behavior. Before the bursting starts, the very first place which can be customized is the *controller.* Following the *controller* , and part of the bursting lifecycle, are coming a list of sequentially ordered burst phases. The burst lifecycle has the following ordered phases:

- *startBursting* - event triggered when the burst is starting
- *startParsePage* - event triggered before a page text is parsed
- *endParsePage* - event triggered after a page text was parsed
- *startExtractDocument* - event triggered before a burst report is extracted
- *endExtractDocument* - event triggered after a burst report was just extracted

- *startDistributeDocument* - event triggered before a burst report is distributed
- *endDistributeDocument* - event triggered after a burst report was just distributed
- *quarantineDocument* - event triggered whenever a report failed to be distributed and it is being quarantined
- *endBursting* - event triggered when the burst is finishing

Bursting Context

Bursting context is an object which is implicitly available for scripting throughout all the bursting lifecycle phases. The *bursting context* is available during scripting as a variable named *ctx.*

Following is the information which is available through the *bursting context.*

```
public List<String> burstTokens;

public String inputDocumentFilePath;

public String configurationFilePath;

public Settings settings;
public Variables variables;
public Scripts scripts;

public int currentPageIndex;
public String currentPageText;
public String previousPageText;

public String token;

public String outputFolder;
public String backupFolder;
public String quarantineFolder;

public String extractFilePath;

public int numberOfPages;

public int numberOfExtractedFiles;
public int numberOfDistributedFiles;
public int numberOfSkippedFiles;
public int numberOfQuarantinedFiles;

public boolean skipCurrentFileDistribution = false;

public List<String> attachments = new ArrayList<String>();
public String archiveFilePath;

public Object additionalInformation;
```

- ***ctx.inputDocumentFilePath*** - file path to the report which is being processed.

Lifespan - *ctx.inputDocumentFilePath* is available for all of the bursting lifecycle phases.

- ***ctx.configurationFilePath*** - file path to the configuration template which is being used.

  Lifespan - *ctx.configurationFilePath* is available for the *startExtractDocument, endExtractDocument, startDistributeDocument, endDistributeDocument, quarantineDocument* and *endBursting* lifecycle phases/events.

- ***ctx.settings*** - contains the settings used to process the current report. Following settings fields might present interest while scripting *burstFileName, outputFolder, backupFolder, quarantineFolder, sendFiles, deleteFiles, quarantineFiles* - with the last three fields being of type boolean.

  Lifespan - *ctx.settings* is available throughout all the bursting lifecycle starting with the first *startBursting* phase and up to the last *endBursting.*

- ***ctx.variables*** - Map<String, Object> which contains both the built-in and the user defined variables.

  The built-in variables are accessible using the *ctx.variables.get(variableName)* syntax.

  For instance, the syntax

  ctx.variables.get("input_document_name")

  will return the file name of the input report.

  The values for the following built-in variables can be returned similarly:

  input_document_name, burst_token, burst_index, output_folder, extracted_file_path, now, now_default_date, now_short_date, now_medium_date, now_long_date, now_full_date, now_default_time, now_short_time, now_medium_time, now_long_time, now_full_time and now_quarter.

  User defined variables are populated and are available per each separate burst token. The syntax to access the user variables is *ctx.variables.getUserVariables(ctx.token).get(variableName).*

  For example the code,

  ctx.variables.getUserVariables("clyde.grew@northridgehealth.org").get("var0")

  will return the first user variable for the token *clyde.grew@northridgehealth.org.*

  While the code,

  ctx.variables.getUserVariables(ctx.token).get("var0")

  will return the first user variable for the current burst token.

  Lifespan - Beside the *burst_token, burst_index, output_folder and extracted_file_path* all the other built-in variables are available throughout all the bursting lifecycle starting with the first *startBursting* phase up to the last *endBursting.*

  burst_token, burst_index and output_folder are populated during the time the burst reports are generated and are available in *startExtractDocument, endExtractDocument, startDistributeDocument, endDistributeDocument and quarantineDocument.*

  extracted_file_path is populated after each report is extracted and is available in *endExtractDocument, startDistributeDocument, endDistributeDocument and quarantineDocument.*

User variables are progressively populated during the time the report pages are being parsed and them become fully available for the *startExtractDocument, endExtractDocument, startDistributeDocument, endDistributeDocument, quarantineDocument and endBursting* phases.

- ***ctx.scripts*** - keeps track of the Groovy scripts to be executed for each of the bursting phases. *DocumentBurster* is coming with nine empty script templates found under the `scripts/burst` folder. The existing templates are suitable to be used for most of the scripting situations. For example, in order to put some custom behavior when the bursting is finished, than the simplest way to do this is to write the tailored logic by editing the existing empty template *endBursting.groovy* script.

  However, there might be cases in which it will be a need to associate totally new Groovy scripts to be executed when some bursting events are happening.

  The syntax to specify a custom script is *ctx.scripts.eventName = script_name.groovy*

  For example

  ctx.scripts.endExtractDocument = my_custom_script.groovy

  will assign the `my_custom_script.groovy` to be executed after each report is extracted.

  Following are all the phases/events for which custom scripts can be associated:

  - *ctx.scripts.startBursting*
  - *ctx.scripts.endBursting*
  - *ctx.scripts.startParsePage*
  - *ctx.scripts.endParsePage*
  - *ctx.scripts.startExtractDocument*
  - *ctx.scripts.endExtractDocument*
  - *ctx.scripts.startDistributeDocument*
  - *ctx.scripts.endDistributeDocument*
  - *ctx.scripts.quarantineDistributeDocument*

  If required, new scripts can be associated to be executed during the bursting lifecycle in the *controller*. For an example please see the *Controller* section.

  Lifespan - *ctx.scripts* is available throughout all the bursting lifecycle phases/events.

- ***ctx.currentPageIndex, ctx.currentPageText, ctx.previousPageText*** - the index of the current page which is being parsed and the text of the current and of the previous pages.

  Lifespan - *ctx.currentPageIndex, ctx.currentPageText, ctx.previousPageText* are available for the *startParsePage* and *endParsePage* phases/events.

- ***ctx.token*** - the token used to extract and process the current burst report

  Lifespan - *ctx.token* is available for the *startExtractDocument, endExtractDocument, startDistributeDocument, endDistributeDocument* and *quarantineDocument* phases/events.

- ***ctx.outputFolder, ctx.backupFolder, ctx.quarantineFolder*** - the output folder, backup folder and quarantine folder for the burst reports.

  Lifespan - *ctx.outputFolder, ctx.backupFolder, ctx.quarantineFolder* are available for the *startExtractDocument, endExtractDocument, startDistributeDocument, endDistributeDocument, quarantineDocument* and *endBursting* phases/events.

- ***ctx.extractFilePath*** - the path for current file which is being extracted

  Lifespan - *ctx.extractFilePath* is available for the *startExtractDocument, endExtractDocument, startDistributeDocument, endDistributeDocument* and *quarantineDocument* phases/events.

- ***ctx.numberOfPages*** - number of pages of the report which is being processed.

  Lifespan - *ctx.numberOfPages* is available for all the bursting lifecycle phases.

- ***ctx.numberOfExtractedFiles*** - number of extracted documents/reports.

  Lifespan - *ctx.numberOfExtractedFiles* is counting the number of extracted reports and is fully available during the *endBursting* report bursting phase.

- ***ctx.numberOfDistributedFiles*** - number of distributed documents/reports.

  Lifespan - *ctx.numberOfDistributedFiles* is counting the number of distributed reports and is fully available during the *endBursting* report bursting phase.

- ***ctx.numberOfSkippedFiles*** - number of skipped from distribution documents/reports.

  Lifespan - *ctx.numberOfSkippedFiles* is counting the number of skipped from distribution reports and is fully available during the *endBursting* report bursting phase.

- ***ctx.numberOfQuarantinedFiles*** - number of quarantined documents/reports.

  Lifespan - *ctx.numberOfQuarantinedFiles* is counting the number of quarantined reports and is fully available during the *endBursting* report bursting phase.

- ***ctx.skipCurrentFileDistribution*** - should the current file be skipped from distribution? Default value is *false*.

  Lifespan - *ctx.skipCurrentFileDistribution* can be used to skip some reports from being distributed and is available during *endExtractDocument* report bursting phase.

  For an example on how to use *skipCurrentFileDistribution,* please see ***scripts/burst/samples/skip_current_file_distribution_if.groovy*** sample script.

- ***ctx.attachments*** - list with the path(s) to the attachment(s) which are about to be distributed

  Lifespan - *ctx.attachments* is available for scripting during *startDistributeDocument* report bursting phase.

- ***ctx.archiveFilePath*** - path to the archive file which is generated and is about to be distributed. Available if the configuration to archive the attachments is enabled

  Lifespan - *ctx.archiveFilePath* is available for scripting during *startDistributeDocument, endDistributeDocument and quarantineDocument* report bursting phases.

- ***ctx.additionalInformation*** - additional information which might be required to store and use while scripting *DocumentBurster*.

Sample Scripts

DocumentBurster is coming with a number of sample scripts which can be used as a starting point for implementing other different custom requirements. All the sample scripts are available in the `scripts/burst/samples` folder.

zip.groovy

By default *DocumentBurster* is not archiving the output burst reports. By running few lines of script during the *endBursting* phase, it is possible to capture and zip together all the burst files in a single file.

Edit the script `scripts/burst/endBursting.groovy` with the content found in `scripts/burst/samples/zip.groovy` and then burst a new report. Now, every time a report is burst, the output files will be archived together in a single zip file.

Similarly, if required, the output files can be archived with different formats and algorithms such as gzip, bzip or tar. For a complete list and documentation of the available options please consult the help page of the *Ant Archive Tasks - http://ant.apache.org/manual/tasksoverview.html#archive*

The following code should be self explanatory. For customizing the name of the zip output file please change the value of the variable *zipFilePath* as per the needs.

```
    /*
 *
 * 1. This script should be used for zipping the output burst files
 *    in a single file.
 *
 * 2. The script should be executed during the endBursting report
 *    bursting lifecycle phase.
 *
 * 3. Please copy and paste the content
 *    of this script into the existing
 *    scripts/burst/endBursting.groovy script.
 *
 * 4. The script is doing basic archiving of all the output
 *    PDF files in a single zip file.
 *    Running multiple times the same input report will
 *    override the output zip file between the consecutive runs.
 *
 * 5. More complex archiving requirements can be achieved
 *    by modifying this starting script.
 *
 */

import com.smartwish.documentburster.variables.Variables

//zipFilePath variable keeps the name of the zip file.
//When bursting a report burst.pdf
//the output zip file will be named burst.pdf.zip and will
//contain inside all the generated reports
def zipFilePath = ctx.outputFolder+"/"+\
ctx.variables.get(Variables.INPUT_DOCUMENT_NAME)+".zip"
```

```
def ant = new AntBuilder()

//zip together all the individual burst reports
ant.zip(destfile: zipFilePath,
        basedir: ctx.outputFolder,
        includes: "**/*.pdf, **/*.xls, **/*.xlsx")

//finally, delete the individual burst reports
ant.delete {
  fileset(dir:ctx.outputFolder,
  includes: "**/*.pdf, **/*.xls, **/*.xlsx")
  }
```

encrypt.groovy

By default *DocumentBurster* is not encrypting or password protecting the output burst reports. By placing few lines of script during the *endExtractDocument* phase, it is possible to encrypt and password protect all the output files - *http://en.wikipedia.org/wiki/Portable_Document_Format#Security_and_signatures*

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/encrypt.groovy` and then burst a new report. Now, every time a report is burst, the output files will be encrypted to have both an owner and an user password.

The default user and owner passwords have the same value which is the value of the *$burst_token$* variable. For example, when bursting the sample report `samples/burst.pdf` two output files will be generated `doc1.pdf` and `doc2.pdf` . The password for the first report is *doc1* and for the second one is *doc2* with both passwords being generated from the *$burst_token$* variable.

Similarly, if required, the output files can be encrypted with the following additional possibilities:

- Certification file
- Set the assemble permission
- Set the extraction permission
- Set the fill in form permission
- Set the modify permission
- Set the modify annots permission
- Set the print permission
- Set the print degraded permission
- The number of bits for the encryption key

For a complete list and documentation of the available encrypt options please consult the help page of the *PDFBox encrypt utility - http://pdfbox.apache.org/commandlineutilities/Encrypt.html*

The following code should be self explanatory. For customizing the passwords, following syntax should be used to access the value of a variable - *ctx.variables.getUserVariables(ctx.token).get(variableName)* .

```
    /*
 *
```

```
 * 1. This script should be used for achieving PDF report
 *    encryption capabilities.
 *
 * 2. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 3. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 4. Following PDF encryption scenarios are possible:
 *
 *      4.1 -  Set the owner and user PDF passwords. Default is none.
 *      4.2 -  Digitally sign the report with a X.509 cert file.
 *      Default is none.
 *      4.3 -  Set the assemble permission. Default is true.
 *      4.4 -  Set the extraction permission. Default is true.
 *      4.5 -  Set the fill in form permission. Default is true.
 *      4.6 -  Set the modify permission. Default is true.
 *      4.7 -  Set the modify annots permission. Default is true.
 *      4.8 -  Set the print permission. Default is true.
 *      4.9 -  Set the print degraded permission. Default is true.
 *      4.10 - Sets the number of bits for the encryption key.
 *      Default is 40.
 *
 * 5. For a full list and documentation of the various PDF encryption
 *    capabilities please see
 *    http://pdfbox.apache.org/commandlineutilities/Encrypt.html
 *
 */

import com.smartwish.documentburster.variables.Variables

/*
 *
 * Warning:
 *
 * 1. Normally it should not be any need for you to modify
 *    the value of pdfBoxClassPath.
 *
 * 2. You should only double check that the values of
 *    the hard-coded jar paths/versions are still valid.
 *    With new releases of new software the jar paths/versions
 *    might become obsolete.
 *
 * 3. If required, modify the paths/versions with care.
 *    Having the pdfBoxClassPath wrong will result in the
 *    following ant.exec/pdfbox call to fail.
 *
 */

def pdfBoxClassPath="lib/burst/pdfbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/commons-logging-1.1.1.jar"
pdfBoxClassPath+=";lib/burst/jempbox-1.0.0.jar"
```

```
pdfBoxClassPath+=";lib/burst/fontbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/bcmail-jdk15-1.44.jar"
pdfBoxClassPath+=";lib/burst/bcprov-jdk15-1.44.jar"

/*
 *
 * 1. encryptOptions are the arguments which are passed for
 *    PDF encryption.
 *
 * 2. By default the encryptOptions is defining the
 *    owner (-O) and user (-U) passwords having the same
 *    value of the $burst_token$ system variable.
 *
 * 3. You can customize for different user and owner
 *    passwords which can be fetched from the values
 *    of any user variable such as $var0$, $var1$, etc.
 *
 */

def burstToken = ctx.token

/*
 *
 *  Following is an example to access the value of the first
 *  user defined variable $var0$.
 *
 *  def password = ctx.variables.getUserVariables(ctx.token).get("var0")
 *
 */

def password = burstToken

def inputFile = ctx.extractFilePath

/*
 *
 * 1. By changing the encryptOptions arguments you can
 *    achieve more PDF encryption features such as applying
 *    certification files, modifying the permissions on the report
 *    and modifying the length of the key which is used
 *    during encryption.
 *
 * 2. For a full list and documentation of the various
 *    PDF encryption capabilities please see
 *    http://pdfbox.apache.org/commandlineutilities/Encrypt.html
 *
 * 3. Gotchas: Take care if you want to pass an argument
 *    that contains white space since it will be split into
 *    multiple arguments. This is the reason why
 *    in encryptOptions all the string arguments are
 *    surrounded with the \" character.
 *
 *    For more details please read
 *    http://groovy.codehaus.org/Executing+External+Processes+From+Groovy
```

```
 *
 */

def encryptOptions =  "-O \"$password\" -U \"$password\" \"$inputFile\""

log.info("encryptOptions = $encryptOptions")

def ant = new AntBuilder()

ant.exec(outputproperty:"cmdOut",
  errorproperty: "cmdErr",
  resultproperty:"cmdExit",
  failonerror: "false",
  executable: 'java') {
   arg(line:"-cp $pdfBoxClassPath org.apache.pdfbox.Encrypt $encryptOptions")
  }

println "return code:  ${ant.project.properties.cmdExit}"
println "stderr:       ${ant.project.properties.cmdErr}"
println "stdout:       ${ant.project.properties.cmdOut}"
```

overlay.groovy

Using this sample script, *DocumentBurster* can stamp the output burst reports. The script should be executed during the *endExtractDocument* report bursting lifecycle phase. The script is using the `samples/Stamp.pdf` to overlay the output burst reports. It is easy to customize the overlay with a different custom stamp.

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/overlay.groovy` and then burst a new report. Now, every time a report is burst, the output files will be stamped with the `samples/Stamp.pdf` file.

The following code should be self explanatory. For customizing the overlay document please replace the existing `samples/Stamp.pdf` with a a different file.

```
    /*
*
* 1. This script should be used as a sample to overlay one document
*    as a stamp on top of the burst reports.
*
* 2. The script should be executed during the endExtractDocument
*    report bursting lifecycle phase.
*
* 3. Please copy and paste the content of this sample script
*    into the existing scripts/burst/endExtractDocument.groovy
*    script.
*
* 4. For a full documentation of the PDF overlay capability
*    please see
*    http://pdfbox.apache.org/commandlineutilities/Overlay.html
*
*/
```

```
import com.smartwish.documentburster.variables.Variables

/*
*
* Warning:
*
* 1. Normally it should not be any need for you to modify
*    the value of pdfBoxClassPath.
*
* 2. You should only double check that the values of
*    the hard-coded jar paths/versions are still valid.
*    With new releases of new software the jar paths/versions
*    might become obsolete.
*
* 3. If required, modify the paths/versions with care.
*    Having the pdfBoxClassPath wrong will result in the
*    following ant.exec/pdfbox call to fail.
*
*/

def pdfBoxClassPath="lib/burst/pdfbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/commons-logging-1.1.1.jar"
pdfBoxClassPath+=";lib/burst/jempbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/fontbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/bcmail-jdk15-1.44.jar"
pdfBoxClassPath+=";lib/burst/bcprov-jdk15-1.44.jar"

//apply the samples/Stamp.pdf as overlay
//for the extracted report
def inputFile = ctx.extractFilePath

def overlayOptions =  "samples/Stamp.pdf \"$inputFile\" \"$inputFile\""

log.info("overlayOptions = $overlayOptions")

def ant = new AntBuilder()

ant.exec(outputproperty:"cmdOut",
  errorproperty: "cmdErr",
  resultproperty:"cmdExit",
  failonerror: "false",
  executable: 'java') {
   arg(line:"-cp $pdfBoxClassPath org.apache.pdfbox.Overlay $overlayOptions")
  }

println "return code:  ${ant.project.properties.cmdExit}"
println "stderr:       ${ant.project.properties.cmdErr}"
println "stdout:       ${ant.project.properties.cmdOut}"
```

exec_pdftk_background.groovy

Using this sample script, *DocumentBurster* can apply a PDF watermark to the background of the output burst reports. The script should be executed during the *endExtractDocument* report bursting lifecycle phase. The script is using the `samples/Stamp.pdf` to be applied as a background to the output burst reports. It is easy to customize the background operation with a different custom stamp.

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/exec_pdftk_background.groovy` and then burst a new report. Now, every time a report is burst, the output files will be stamped with the `samples/Stamp.pdf` file.

The following code should be self explanatory. For customizing the background stamp please replace the existing `samples/Stamp.pdf` with a different custom file.

```
    /*
 *
 * 1. This script should be used:
 *
 *      1.1 - As a sample script to call an external executable
 *      during the report bursting life cycle.
 *      1.2 - As a sample for applying a PDF watermark to the
 *      background of the burst reports.
 *
 * 2. The external program to be demonstrated is pdftk
 *    http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/
 *
 * 3. pdftk or the pdf toolkit is a cross-platform tool for
 *    manipulating PDF documents. pdftk is basically a front
 *    end to the iText library (compiled to Native code using GCJ),
 *    capable of splitting, merging, encrypting, decrypting,
 *    uncompressing, recompressing, and repairing PDFs.
 *    It can also be used to manipulate watermarks, metadata,
 *    and to fill PDF Forms with FDF Data (Forms Data Format)
 *    or XFDF Data (XML Form Data Format).
 *
 * 4. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 5. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 6. For a full documentation of the PDF background capability
 *    please see
 *    http://www.pdflabs.com/docs/pdftk-man-page/#dest-op-background
 *
 */

import com.smartwish.documentburster.variables.Variables

def extractFilePath = ctx.extractFilePath
def stampedFilePath = ctx.extractFilePath + "_stamped.pdf"
```

```
//apply the samples/Stamp.pdf as a background
//to the extracted report
def execOptions =  "\"$extractFilePath\" background samples/Stamp.pdf "
execOptions += "output \"$stampedFilePath\""

/*
 *
 * 1. Please download and install pdftk from this location
 *    http://www.pdflabs.com/docs/install-pdftk/
 *
 * 2. Make sure to download the binaries which are
 *    specific to the target operating system.
 *
 * 3. Move the pdftk.exe and libiconv2.dll in the folder
 *    where DocumentBurster was installed, next
 *    to DocumentBurster.exe file.
 *
 */

def ant = new AntBuilder()

log.info("Executing pdftk.exe $execOptions")

//http://groovy.codehaus.org/Executing+External+Processes+From+Groovy
ant.exec(append: "true",
  failonerror: "true",
  output:"logs/pdftk.log",
  executable: 'pdftk.exe') {
   arg(line:"$execOptions")
  }

ant.move(file:"$stampedFilePath", tofile:"$extractFilePath")
```

print.groovy

Using this sample script, *DocumentBurster* can send the output burst reports to the printer. The script should be executed during the *endExtractDocument* report bursting lifecycle phase.

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/print.groovy` and then burst a new report. Now, every time a report is burst, the output files will be sent to the printer.

Using the *-silentPrint* switch it is possible to print the PDF reports without prompting for a printer.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used as a sample to print the burst reports.
 *
 * 2. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
```

```
 *
 * 3. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 4. For a full documentation of the PDF print capability
 *    please see
 *    http://pdfbox.apache.org/commandlineutilities/PrintPDF.html
 *
 */

import com.smartwish.documentburster.variables.Variables

/*
 *
 * Warning:
 *
 * 1. Normally it should not be any need for you to modify
 *    the value of pdfBoxClassPath.
 *
 * 2. You should only double check that the values of
 *    the hard-coded jar paths/versions are still valid.
 *    With new releases of new software the jar paths/versions
 *    might become obsolete.
 *
 * 3. If required, modify the paths/versions with care.
 *    Having the pdfBoxClassPath wrong will result in the
 *    following ant.exec/pdfbox call to fail.
 *
 */

def pdfBoxClassPath="lib/burst/pdfbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/commons-logging-1.1.1.jar"
pdfBoxClassPath+=";lib/burst/jempbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/fontbox-1.0.0.jar"
pdfBoxClassPath+=";lib/burst/bcmail-jdk15-1.44.jar"
pdfBoxClassPath+=";lib/burst/bcprov-jdk15-1.44.jar"

def extractFilePath = ctx.extractFilePath

//-silentPrint can be used to print the PDF without prompting for a printer.
def printOptions =  "\"$extractFilePath\""

log.info("printOptions = $printOptions")

def ant = new AntBuilder()

ant.exec(outputproperty:"cmdOut",
  errorproperty: "cmdErr",
  resultproperty:"cmdExit",
  failonerror: "false",
  executable: 'java') {
   arg(line:"-cp $pdfBoxClassPath org.apache.pdfbox.PrintPDF $printOptions")
  }
```

```
println "return code:  ${ant.project.properties.cmdExit}"
println "stderr:       ${ant.project.properties.cmdErr}"
println "stdout:       ${ant.project.properties.cmdOut}"
```

copy_shared_folder.groovy

Using this sample script, *DocumentBurster* can copy each individual output burst file to a shared folder (as long as the shared drive is mounted). The script should be executed during the *endExtractDocument* report bursting lifecycle phase.

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/copy_shared_folder.groovy` and then burst a new report. Now, every time a report is burst, the output files will be uploaded to the shared folder.

By default the script is getting the shared location path from the content of *$var0$* user variable (e.g // VBOXSVR/shareit).

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used for copying each individual
 *    output burst file to a shared folder
 *    (as long as the shared drive is mounted).
 *
 * 2. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 3. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 4. Ant copy task is used to upload the reports to the
 *    shared location
 *    - http://ant.apache.org/manual/Tasks/copy.html
 *
 */

import com.smartwish.documentburster.variables.Variables

def ant = new AntBuilder()

/*
 *    By default the script is getting the shared location path
 *    from the content of $var0$ user variable (e.g //VBOXSVR/shareit)
 *
 */
def sharedLocationPath = ctx.variables.getUserVariables(ctx.token).get("var0")

//ant.copy(file:ctx.extractFilePath, todir:'//VBOXSVR/shareit', overwrite:true)
ant.copy(file:ctx.extractFilePath, todir:"$sharedLocationPath", overwrite:true)
```

ant_ftp.groovy

Using this sample script, *DocumentBurster* can copy all the output burst files at once to a remote FTP server location.The script should be executed during the *endBursting* report bursting lifecycle phase.

Edit the script `scripts/burst/endBursting.groovy` with the content found in `scripts/burst/samples/ant_ftp.groovy` and then burst a new report. Now, every time a report is burst, the output files will be uploaded to the FTP server location.

By default the script is fetching the values of the FTP connect session, such as user, password and host from the values of *$var0$, $var1$ and $var2$* user report variables. If the burst reports are configured as such, then there is nothing more to do, and the FTP upload will work without any modification to the script. Otherwise, the FTP script should be modified as per the needs.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used for copying all the output burst
 *    files at once to a remote FTP server location.
 *
 * 2. The script should be executed during the endBursting report
 *    bursting lifecycle phase.
 *
 * 3. Please copy and paste the content of this script into the
 *    existing scripts/burst/endBursting.groovy script.
 *
 * 4. The scope of this script is to copy all the *.pdf files
 *    generated in the last burst session.
 *    Thus, in order for this script to really upload only
 *    the last generated files, it is required that each burst
 *    session will generate a new and unique burst output folder.
 *
 * 5. Ant FTP task is used to upload the reports
 *    - http://ant.apache.org/manual/Tasks/ftp.html
 *
 */

import com.smartwish.documentburster.variables.Variables

/*
 *    By default the script is getting the required FTP session information
 *    from the following sources:
 *
 *        userName - from the content of $var0$ user variable
 *        password - from the content of $var1$ user variable
 *
 *        hostName - from the content of $var2$ user variable
 *
 */
def userName = ctx.variables.getUserVariables(ctx.token).get("var0")
```

```
def password = ctx.variables.getUserVariables(ctx.token).get("var1")

def hostName = ctx.variables.getUserVariables(ctx.token).get("var2")

ant = new AntBuilder()

/*
 *    Copy all the *.pdf files generated in the last burst session.
 *    Thus, in order for this script to really upload only
 *    the last generated files, it is required that each burst
 *    session will generate a new and unique burst output folder.
 *
 */
ant.ftp(server: "$hostName",
  userid: "$userName",
  password: "$password",
  passive: 'yes',
  verbose: 'yes',
  binary: 'yes' ) {
   fileset(dir:ctx.outputFolder,includes: '**/*.pdf')
  }
```

ant_scp_sftp.groovy

Using this sample script, *DocumentBurster* can copy each individual output burst file to a remote SCP/SFTP server location. The script should be executed during the *endExtractDocument* report bursting life-cycle phase.

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/ant_scp_sftp.groovy` and then burst a new report. Now, every time a report is burst, the output files will be uploaded to the SFTP/SCP server location.

By default the script is fetching the values of the SCP/SFTP connect session, such as user, password, host and path from the values of *$var0$, $var1$, $var2$ and $var3$* user report variables. If the burst reports are configured as such, then there is nothing more to do, and the SFTP/SCP upload will work without any modification to the script. Otherwise, the SCP/SFTP script should be modified as per the needs.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used for copying each individual output burst file
 *    to a remote SCP/SFTP server location.
 *
 * 2. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 3. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 4. Ant SCP task is used to upload the reports
 *    - http://ant.apache.org/manual/Tasks/scp.html
```

```
 *
 */

import com.smartwish.documentburster.variables.Variables

/*
 *
 *    By default the script is getting the required SCP/SFTP session
 *    information from the following sources:
 *
 *        userName - from the content of $var0$ user variable
 *        password - from the content of $var1$ user variable
 *
 *        hostName - from the content of $var2$ user variable
 *        absolutePath - from the content of $var3$ user variable
 *
 */
def userName = ctx.variables.getUserVariables(ctx.token).get("var0")
def password = ctx.variables.getUserVariables(ctx.token).get("var1")

def hostName = ctx.variables.getUserVariables(ctx.token).get("var2")
def absolutePath = ctx.variables.getUserVariables(ctx.token).get("var3")

ant = new AntBuilder()

ant.scp(file: ctx.extractFilePath,
  todir: "$userName@$hostName:$absolutePath",
  password: "$password",
  trust:'true')
```

ant_vfs.groovy

DocumentBurster can distribute the output burst reports by using *Commons Virtual File System.* [http://commons.apache.org/vfs/index.html]

By scripting Commons VFS, *DocumentBurster* can upload the reports to any of the supported file systems such as FTP, Local Files, HTTP and HTTPS, SFTP, WebDAV and CIFS.

For example, following use cases are all achievable:

- Using HTTP POST, upload the burst reports to a cloud storage provider such as *Box.net* [http://www.box.net/] or *Dropbox.* [https://www.dropbox.com/]
- Using HTTP POST or WebDAV, upload the burst reports to a corporate portal such as Microsoft SharePoint, IBM WebSphere Portal, Oracle Portal, SAP NetWeaver, Tibco PortalBuilder or Samsung ACUBE Portal.
- Using CIFS, upload the burst reports to a CIFS server such as a Samba server, or a Windows share.

This script is showing how to copy the burst reports using the *file://* protocol and, with minimum effort, it can be adapted for any of the above listed protocols.

The script should be executed during the *endExtractDocument* report bursting lifecycle phase. Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/`

`burst/samples/ant_vfs.groovy` and then burst a new report. Now, every time a report is burst, the output files will be copied to the configured folder path.

By default the script is fetching the value of the destination folder from the value of *$var0$* user report variable. If the burst reports are configured as such, then there is nothing more to do, and the script will work without any other additional modification. Otherwise, the VFS script should be modified as per the needs.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used as a sample
 *    for copying/uploading each individual output burst file
 *    by using the Apache Commons VFS library.
 *    Commons VFS provides a single API for accessing various different
 *    file systems. It presents a uniform view of the files from various
 *    different sources, such as the files on local disk, on an HTTP server,
 *    or inside a Zip archive.
 *
 *    http://commons.apache.org/vfs/index.html
 *
 * 2. Commons VFS currently supports the following file systems:
 *    http://commons.apache.org/vfs/filesystems.html
 *
 * 3. This script is demonstrating the use of the V-Copy
 *    Commons VFS Ant task.
 *
 *    http://commons.apache.org/vfs/anttasks.html#V-Copy
 *
 * 4. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 5. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 */

import com.smartwish.documentburster.variables.Variables

/*
 *
 *    By default the script is getting the destination folder from the content
 *    of $var0$ user variable
 *
 */

//e.g. destDir = "file:///C:/test"
def destDir = ctx.variables.getUserVariables(ctx.token).get("var0")

ant = new AntBuilder()

ant.sequential{
```

```
    taskdef(name:"vfs_copy", classname:"org.apache.commons.vfs2.tasks.CopyTask")

    vfs_copy(src: ctx.extractFilePath,
        destdir: "$destDir",
        overwrite:'true')
}
```

add_and_format_page_numbers.groovy

As the name of the file suggests, this script can be used to add page numbers to the output burst reports. The script is numbering the pages of the output reports consecutively.

Each page of the output burst reports is stamped with the correct page number and both of the following two situations are supported:

- Add new page numbers when the initial input report does not have the pages numbered
- Replace and fix the existing page numbers when existing page numbering of the input reports becomes incorrect after the report is burst

The script should be executed during the *endExtractDocument* report bursting lifecycle phase. Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/add_and_format_page_numbers.groovy` and then burst a new report. Now, every time a report is burst, the pages of the output files will be properly stamped with a label similar with *Page i of n* ; where *i* is the index of the current page and *n* is the total number of pages.

The text, the font and the location of the page numbering label can be customized by doing small changes to the existing script. For example the following line of script will place the location of the numbering label at the bottom-left corner of the page.

```
over.setTextMatrix(30, 30);
```

The location of the label can be changed by altering the above coordinates. Please check the inline code comments for further details.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used for applying page numbers to
 *    the output burst files.
 *
 *    The script can:
 *
 *      1.1 - Place new numbers for pages of output burst reports
 *      which are not initially numbered.
 *      1.2 - Replace and fix the numbers for pages of burst reports
 *      for which the existing page numbering becomes incorrect
 *      after the report is split.
 *
```

```
 * 2. The text, the font and the location of the page numbering
 *    label can be customized by doing small changes to this script.
 *
 *    Please check the inline code comments for further details.
 *
 * 3. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 4. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 */

import java.io.FileOutputStream;
import java.awt.Color;

import org.apache.commons.io.FilenameUtils;
import com.lowagie.text.Element;
import com.lowagie.text.pdf.BaseFont;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfReader;
import com.lowagie.text.pdf.PdfStamper;
import com.lowagie.text.pdf.PdfGState;

/*
 *    Font of the label. Default value is BaseFont.HELVETICA
 *
 *    Other possible values are:
 *
 *      BaseFont.COURIER
 *      BaseFont.COURIER_BOLD
 *      BaseFont.COURIER_BOLDOBLIQUE
 *      BaseFont.COURIER_OBLIQUE
 *      BaseFont.HELVETICA
 *      BaseFont.HELVETICA_BOLD
 *      BaseFont.HELVETICA_BOLDOBLIQUE
 *      BaseFont.HELVETICA_OBLIQUE
 *      BaseFont.SYMBOL
 *      BaseFont.TIMES_BOLD
 *      BaseFont.TIMES_BOLDITALIC
 *      BaseFont.TIMES_ITALIC
 *      BaseFont.TIMES_ROMAN
 *      BaseFont.ZAPFDINGBATS
 *
 */
BaseFont bf = BaseFont.createFont(BaseFont.HELVETICA,
    BaseFont.WINANSI, BaseFont.EMBEDDED);

def numberedFilePath = ctx.outputFolder +
                       FilenameUtils.getBaseName(ctx.extractFilePath) +
                       "_numbered.pdf"

PdfReader reader = new PdfReader(ctx.extractFilePath);
```

```
//get the number of pages
int n = reader.getNumberOfPages();

PdfStamper stamp = new PdfStamper(reader,
                        new FileOutputStream(numberedFilePath));

PdfContentByte over;

PdfGState gs = new PdfGState();

//100% opacity
gs.setFillOpacity(1.0f);

//current page index
int i = 0;

while (i < n) {

    i++;

    over = stamp.getOverContent(i);

    //draw an "opaque" and white rectangle
    //which is used to hide the old/wrong page numbering
    over.setGState(gs);
    over.setColorFill(Color.WHITE);

    //the default label location is at the bottom left-corner
    //of the page

    //x, y, width, height
    over.rectangle(30, 30, 60, 20);

    over.fill();

    over.beginText();

    //Default text color is black

    //other possible color values
    //http://download.oracle.com/javase/1.4.2/docs/api/java/awt/Color.html

    over.setColorFill(Color.BLACK);

    //the default size of the font is 12
    over.setFontAndSize(bf, 12);

    //the default label location is at the bottom left-corner
    //of the page

    //x, y
    over.setTextMatrix(30, 30);
```

```
    //label text
    over.showText("Page $i of $n");

    over.endText();

}

stamp.close();

def ant = new AntBuilder()

//replace the original burst report
//with the numbered one
ant.delete(file:ctx.extractFilePath)
ant.move(file:"$numberedFilePath", tofile:ctx.extractFilePath)
```

merge_with_external_files.groovy

This script can be used to merge each of the output PDF burst files which is generated by *DocumentBurster* with other external reports. There isn't any restriction and the external reports can be generated by any of the existing proprietary reporting tools like Oracle Hyperion or Crystal Reports/SAP Business Objects.

Once the reports are merged, *DocumentBurster* flow will continue as normal.

By default the script is merging the external report first and the *DocumentBurster* output burst report second. Please see the inline script comments for details about how to change the merging order.

By default, for demonstration purposes, the script is merging as an external report the hard-coded `samples/Invoices-Dec.pdf.` With the help of user variables it is possible to define a configurable and dynamic external report to merge with.

For example, the external report to merge with can be dynamically defined with the help of the $var0$ user variable.

```
    def externalFilePath = ctx.variables.getUserVariables(ctx.token).get("var0")
```

The script should be executed during the *endExtractDocument* report bursting lifecycle phase. Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/merge_with_external_files.groovy` and then burst a new report. Now, every time a report is burst, the output files will be formed by merging the `samples/Invoices-Dec.pdf` with the original output burst files.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script can be used for merging the output PDF burst files
 *    with other external reports.
 *
 *    The script can:
 *
```

```
 *      1.1 - Merge each of the output burst files with other
 *      external and configurable report.
 *      1.2 - By default the external report is merged first
 *      and the burst report is appended second.
 *      1.3 - The merge order can be changed. Please see the
 *      inline code comments for further details.
 *      1.4 - Once the reports are merged, DocumentBurster
 *      flow will continue as normal
 *
 * 2. By default the script is merging as an external report
 *    the hard-coded samples/Invoices-Dec.pdf.
 *
 * 3. By using user variables it is possible to define a
 *    configurable and dynamic external report to merge with.
 *    For example, the external report to merge with can be
 *    dynamically defined with the help of the $var0$ user variable.
 *
 *    Please check the inline code comments for further details.
 *
 * 4. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 5. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 */

import com.smartwish.documentburster.engine.pdf.Merger;
import org.apache.commons.io.FilenameUtils;

def mergedFileName = FilenameUtils.getBaseName(ctx.extractFilePath)+"_merged.pdf"

/*
 *    External report to merge with. The default external report is
 *    defined to be "samples/Invoices-Dec.pdf"
 *
 *    The external report can be dynamically defined with the help
 *    of user variables.
 *
 *    For example
 *
 * def externalFilePath = ctx.variables.getUserVariables(ctx.token).get("var0")
 *
*/

def externalFilePath = "samples/Invoices-Dec.pdf"

//array with the two files to merge
def filePaths = new String[2]

//by default the external file is merged first
filePaths[0] = externalFilePath
//and the burst report is merged second
```

```
filePaths[1] = ctx.extractFilePath

def merger = new Merger(ctx.settings)

merger.doMerge(filePaths, mergedFileName)

def ant = new AntBuilder()

//replace the original burst report
//with the merged one
ant.delete(file:ctx.extractFilePath)
ant.copy(file: merger.getOutputFolder() + "/$mergedFileName",
         tofile:ctx.extractFilePath)

//clean the temporary folders/files
//this code assumes that the default program output/backup location
//is not changed
ant.delete(dir: "output/$mergedFileName",failonerror:false)
ant.delete(dir: "backup/$mergedFileName",failonerror:false)
```

ant_mail.groovy

This script can be used for sending various ad-hoc emails during the report bursting flow. Based on your needs, the script can be executed in any of the existing report bursting lifecycle phases (e.g. *endBursting*, *endExtractDocument* etc).

For example, this sample script can be used almost out of the box for sending an email notification when the bursting is successfully finished. To achieve this, please copy and paste the content of this sample script into the existing `scripts/burst/endBursting.groovy` script.

How to customize the script

- Change the first uncommented line of the script *(def to = "your.address@here.com")* with the email address where you need the email to be sent
- Optionally, the subject and the message of the notification email can be also changed

The following code should be self explanatory

```
    /*
 *
 * 1. This script can be used for sending various ad-hoc
 *    emails during the report bursting flow.
 *
 * 2. Based on your needs, the script can be executed in any of the existing
 *    report bursting lifecycle phases (e.g. endBursting, endExtractDocument etc).
 *
 * 3. For example, this script can be used almost out of the box for sending
 *    an email notification when bursting is successfully finished.
 *    To achieve this, please copy and paste the content of this sample script
 *    into the existing scripts/burst/endBursting.groovy
 *    script.
 *
```

```
 * 4. How to customize the script
 *
 *        4.1. Please change the first uncommented line of the script
 *        (def to = "your.address@here.com") with the email address where
 *        you need the email to be sent.
 *
 *        4.2. Optionally you can change the subject and the message of the
 *        email.
 *
 * 5. Ant Mail task is used
 *    - http://ant.apache.org/manual/Tasks/mail.html
 *
 */

//give a valid email address
def to = "your.address@here.com"

def host = ctx.settings.getEmailServerHost()
def port = ctx.settings.getEmailServerPort()

def user = ctx.settings.getEmailServerUserId()
def password = ctx.settings.getEmailServerUserPassword()

def from = ctx.settings.getEmailServerFrom()

//Optionally the subject can be changed
def subject = "DocumentBurster finished"

//The message can be also changed
def message = "Input file: " + ctx.inputDocumentFilePath +"\n\n"

message = message + "Number of pages: " +ctx.numberOfPages + "\n\n"

message = message + "Number of files extracted: "
message = message + ctx.numberOfExtractedFiles+"\n"

message = message + "Output folder: " + ctx.outputFolder+"\n\n"

message = message + "Number of files distributed: "
message = message + ctx.numberOfDistributedFiles+"\n\n"

message = message + "Number of files skipped from distribution: "
message = message  + ctx.numberOfSkippedFiles+"\n"

message = message + "Number of files quarantined: "
message = message + ctx.numberOfQuarantinedFiles+"\n"

message = message + "Quarantine folder: " + ctx.quarantineFolder+"\n\n"

def ssl="no"

if (ctx.settings.isEmailServerUseSSL())
 ssl="yes"
```

```
def enableStartTLS="no"

if (ctx.settings.isEmailServerUseTLS())
 enableStartTLS="yes"

ant = new AntBuilder()

ant.mail(mailhost:"$host",
         mailport:"$port",
         user:"$user",
         password:"$password",
         subject:"$subject",
         from:"$from",
         tolist:"$to",
         message:"$message",
         ssl:"$ssl",
         enableStartTLS:"$enableStartTLS")

log.info("Notification email sent successfully to email address $to ...")
```

skip_current_file_distribution_if.groovy

This sample script can be used to achieve complex *conditional* report delivery scenarios.

DocumentBurster has built-in support for implementing conditional report delivery and this is described in the *Appendix A. How to... -> Implement conditional report distribution* from *DocumentBurster User Guide [http://www.pdfburst.com/report-distribution-manual.pdf]* manual.

DocumentBurster's built-in support for conditional report distribution requires using a *<skip>true</skip>* instruction (or the shorter form *<s>true</s>)* for the reports which should not be distributed. Based on the specific business requirements, the report writer engine is expected to properly fill the *skip* instructions and this will be done by using a report formula (which will decide if the report should be distributed or not).

DocumentBurster's built-in capabilities *(skip* instruction) can be used to achieve many conditional distribution scenarios while this sample script, `scripts/burst/samples/skip_current_file_distribution_if.groovy`, should be used for achieving the remaining and more complex situations which cannot be implemented using the simple *skip* instruction approach.

This sample script can be used to achieve conditional report distribution in situations similar with the following

- The condition to skip the distribution cannot be achieved using a report formula (e.g. skip the delivery for files which are bigger than 20MB)
- The condition to skip the distribution is too complex and it might be more convenient to describe this in scripting than with a report formula
- For whatever reason the input report cannot be modified to accommodate the *<skip>true</skip>* instructions

The general code structure of the script is the following

```
//Pre-condition helper code

//The condition based on which the distribution will be skipped
if (skip-condition){

    //Skip the delivery of the current report
    ctx.skipCurrentFileDistribution = true

    //Other code which might be required

}
```

- ***ctx.skipCurrentFileDistribution = true*** is the line of code which is enabling *DocumentBurster* to skip the distribution for the current report
- ***skip-condition*** is the condition based on which the report will be skipped for distribution (will be different for each business scenario)

The sample `scripts/burst/samples/skip_current_file_distribution_if.groovy` has the same code structure and is skipping the distribution for reports which are bigger than the configurable 20MB file size threshold.

```
//configurable FILE_SIZE_THRESHOLD
final def FILE_SIZE_THRESHOLD = 20
```

The script must be executed during the *endExtractDocument* report bursting lifecycle phase. Please copy and paste the content of this sample script into the existing `scripts/burst/endExtractDocument.groovy` script.

```
    /*
 *
 * 1. This script can be used to implement more advanced conditional
 *    report delivery scenarios.
 *
 * 2. While the current script is a sample on how to skip the
 *    report distribution for reports > 20 MB (this is a configurable
 *    threshold since MS Exchange will bounce back for reports
 *    which are so big), similarly it is possible to skip the distribution
 *    based on any custom business situation which your organization
 *    might have.
 *
 * 3. "ctx.skipCurrentFileDistribution = true" is the line of code which
 *    is enabling DocumentBurster to skip the distribution
 *    for the current report.
 *
 * 4. The script must be executed during the endExtractDocument
 *    report bursting lifecycle phase.
```

```
 *
 * 5. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 6. How to customize the script to achieve other conditional
 *    report delivery scenarios
 *
 *        6.1. Replace the "if (currentFileSize >= FILE_SIZE_THRESHOLD)"
 *        with any custom condition which is appropriate for your
 *        scenario.
 *
 *        6.2. Beside the "ctx.skipCurrentFileDistribution = true"
 *        the rest of the code which is found in the IF block is just copying
 *        to quarantine the offending report (>20MB threshold).
 *
 *        Optionally you might want to change the code from within the IF block
 *        with something else which is better fitting your needs.
 *
 */

import com.smartwish.documentburster.utils.Utils

import org.apache.commons.io.FileUtils
import org.apache.commons.io.FilenameUtils

//configurable FILE_SIZE_THRESHOLD
final def FILE_SIZE_THRESHOLD = 20

def currentFile = new File(ctx.extractFilePath)

//get the size (in MEGABYTE) of the current report
def currentFileSize = Utils.getFileSize(currentFile.length(),
                                        Utils.FileSizeUnit.MEGABYTE);

//if the report is bigger than the defined threshold
if (currentFileSize > FILE_SIZE_THRESHOLD) {

    //skip the distribution
    ctx.skipCurrentFileDistribution = true

    //start - copy the report to quarantine
    File quarantineDir = new File(ctx.quarantineFolder);

    if (!quarantineDir.exists())
        FileUtils.forceMkdir(quarantineDir);

    File quarantineFile = new File(ctx.quarantineFolder + "/" +
                                   FilenameUtils.getName(ctx.extractFilePath));

    if (!quarantineFile.exists())
        FileUtils.copyFile(new File(ctx.extractFilePath), quarantineFile);

    ctx.numberOfQuarantinedFiles++;
```

```
    //end - copy the report to quarantine

    log.warn("The following file was skipped for distribution since its size - "+
             currentFileSize + " MB - is bigger than the " +
             FILE_SIZE_THRESHOLD + " MB file size threshold")

    log.warn("Associated burst token for the skipped file: " +
             ctx.token +", file path: '") + ctx.extractFilePath + "'"

    log.warn("The file was quarantined")

}
```

batch_pdf_print.groovy

Silent PDF batch printing

Using this sample script, *DocumentBurster* can silently print the output burst reports.

Foxit Reader

This script is using *Foxit Reader* in order to print the reports. *Foxit Reader* should be installed on your machine in order for this script to work properly.

www.foxitsoftware.com/Secure_PDF_Reader/

Foxit Reader - Command Line Switches

- Print a PDF file silently to the default printer : *"Foxit Reader.exe" /p <PDF Path>*
- Print a PDF file silently to an alternative printer: *"Foxit Reader.exe" /t <PDF Path> [Printer]*

The script should be executed during the *endExtractDocument* report bursting lifecycle phase. Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/batch_pdf_print.groovy` and then burst a new report. Now, every time a report is burst, the output files will be sent to the default printer.

The following code should be self explanatory.

```
    /*
 *
 * 1. This script should be used as a sample to silently batch
 *    print the burst (PDF) reports.
 *
 * 2. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 3. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 4. This script is using Foxit Reader in order to print the reports.
 *    Foxit Reader should be installed on your machine in order for
 *    this script to work properly.
```

```
 *
 *   - http://www.foxitsoftware.com/Secure_PDF_Reader/
 *
 * 5. Foxit Reader - Command Line Switches
 *
 *      5.1 Print a PDF file silently to the default printer:
 *
 *          "Foxit Reader.exe" /p <PDF Path>
 *
 *      5.2 Print a PDF file silently to an alternative printer:
 *
 *          "Foxit Reader.exe" /t <PDF Path> [Printer]
 *
 */
import java.io.File

def extractFilePath = (new File(ctx.extractFilePath)).getCanonicalPath()

def execOptions = "/p \"$extractFilePath\""

def ant = new AntBuilder()

log.info("Executing 'Foxit Reader.exe $execOptions'")

//If required, change the path to point to your installation of Foxit Reader
ant.exec(append: "true",
  failonerror: "true",
  output:"logs/foxit.log",
  executable: "C:/Program Files (x86)/Foxit Software/Foxit Reader/Foxit Reader.exe
   arg(line:"$execOptions")
  }
```

fetch_distribution_details_from_database.groovy

Fetch Bursting and Distribution Details from Database

Using this sample script, *DocumentBurster* can fetch the bursting and distribution details from an external database. Once fetched, the details are populated into the *var0*, *var1*, *etc* user variables in order to be further used by *DocumentBurster*.

This sample script is demonstrating how to connect to an HSQLDB database, however you can modify the connection details to point to an:

- Oracle database
- Microsoft SQL Server, Microsoft Access or Microsoft FoxPro database
- IBM DB2 or IBM AS/400 database
- PostgreSQL, MySQL, SQLite, Apache Derby or FireBird database
- Teradata database

JDBC driver

In order for this script to work it is mandatory to copy the correct JDBC driver jar file (corresponding to your database) into the existing `lib/burst` folder. For more details about available JDBC drivers please check

http://developers.sun.com/product/jdbc/drivers

Double check your customized SQL

In this script it is required to change the SQL query to meet your own need. It is mandatory to carefully check that your customized SQL query is correct and it is properly returning the unique details for the appropriate employee or customer (otherwise the risk is to send confidential information to the wrong employee/customer).

The following code should be self explanatory

```
    /*
 *
 * 1. This script should be used as a sample to fetch the
 *    bursting/distribution meta-data details from an external database.
 *
 * 2. The script can be executed (depending on the need) in either
 *    startExtractDocument, endExtractDocument or startDistributeDocument
 *    report bursting life-cycle phases.
 *
 * 3. Please copy and paste (if this is what you need) the content
 *    of this sample script into the existing
 *    scripts/burst/startExtractDocument.groovy script.
 *
 * 4. This sample script is connecting to an HSQLDB database, however
 *    you can modify the connection details to point to an
 *
 *        Oracle,
 *        Microsoft Access,
 *        Microsoft SQL Server,
 *        Microsoft FoxPro,
 *        IBM DB2,
 *        IBM AS/400,
 *        MySQL,
 *        PostgreSQL,
 *        Teradata,
 *        SQLite,
 *        Apache Derby or
 *        FireBird SQL database
 *
 * 5. In order for this script to work it is mandatory to copy the correct
 *    JDBC driver jar (corresponding to your database)
 *    file into the existing lib/burst folder
 *
 * 6. For more details about available JDBC drivers please check
 *
 *        http://developers.sun.com/product/jdbc/drivers
 *
 * 7. Groovy SQL resources
```

```
 *
 *        7.1 Groovy SQL - http://groovy.codehaus.org/Tutorial+6+-+Groovy+SQL
 *        7.2 Practically Groovy: JDBC programming with Groovy -
 *        http://www.ibm.com/developerworks/java/library/j-pg01115/index.html
 *
 */

import groovy.sql.Sql

//HSQLDB sample

//Replace localhost with your host
//Replace xdb with your own database name
//Replace sa and '' with your own database login details

def sql = Sql.newInstance('jdbc:hsqldb:hsql://localhost/xdb',
                          'sa', '','org.hsqldb.jdbcDriver')

//Oracle sample

//Replace localhost with your host
//Replace username and password with your database login details
//Change to your own database instance

//def sql = Sql.newInstance('jdbc:oracle:thin:@localhost:1521:orcl',
//                          'username', 'password',
//                          'oracle.jdbc.pool.OracleDataSource' )

//The burst token is used as a key to identify the details
//of the appropriate employee or customer
def token = ctx.token

//Change the SQL to your own need

//Double check your customized SQL is correct and is
//properly returning the unique details for the appropriate
//employee/customer (otherwise the risk is to send
//confidential information to the wrong employee or customer)

def employeeRow = sql.firstRow('SELECT employee_id, email_address,' +
                  'first_name, last_name FROM employees WHERE employee_id = ?',
                  [token])

def emailAddress = employeeRow.email_address

def firstName = employeeRow.first_name
def lastName = employeeRow.last_name

println "Employee: employee_id = ${employeeRow.employee_id} and " +
         "email_address = ${emailAddress} and first_name = ${firstName} " +
         "and last_name = ${lastName}"

//Populate the fetched information into var0, var1, etc user variables.
ctx.variables.setUserVariable(String.valueOf("${token}"),"var0",
```

```
                                String.valueOf("${emailAddress}"))

ctx.variables.setUserVariable(String.valueOf("${token}"),"var1",
                                String.valueOf("${firstName}"))

ctx.variables.setUserVariable(String.valueOf("${token}"),"var2",
                                String.valueOf("${lastName}"))
```

fetch_distribution_details_from_csv_file.groovy

Fetch Bursting and Distribution Details from an External (CSV) File

Using this sample script, *DocumentBurster* can fetch the bursting and distribution meta-data details from an external (CSV) file. Once fetched, the details are populated into the *var0*, *var1*, *etc* user variables in order to be further used by *DocumentBurster*. This script is reading the information from a CSV file, however you can modify the script to parse and read other plain text files which have a more custom format.

Following is a sample with how this script is expecting the CSV file

employee.csv

```
employee_id,email_address,first_name,last_name
1,email1@address1.com,firstName1,lastName1
2,email2@address2.com,firstName2,lastName2
3,email3@address3.com,firstName3,lastName3
4,email4@address4.com,firstName4,lastName4
```

The first column from the file is the employee identifier. The script is using this column to find the row which contains the details for each employee. Following is the code which is doing this

```
//The burst token is used as a key to identify the
details
//of the appropriate employee or customer
if
(employeeRow[0]== token)
{

...

}
```

If you have a file with a different structure then the script should be modified accordingly.

Double check your script logic

Most probably you will modify this script accordingly to your own custom file format. It is mandatory to carefully test that your script is properly parsing and reading the unique details for the appropriate employee or customer otherwise the risk is to send confidential information to the wrong employee/customer.

The following code should be self explanatory

```
    /*
 *
 * 1. This script should be used as a sample to fetch the
 *    bursting/distribution meta-data details from an external (CSV) file.
 *
 * 2. The script can be executed (depending on the need) in either
 *    startExtractDocument, endExtractDocument or startDistributeDocument
 *    report bursting life-cycle phases.
 *
 * 3. Please copy and paste (if this is what you need) the content
 *    of this sample script into the existing
 *    scripts/burst/startExtractDocument.groovy script.
 *
 * 4. This sample script is reading the information from a CSV file, however
 *    you can modify the script to parse and read other plain text files
 *    (which have your own custom format).
 *
 * 5. Following is a sample with how this script is expecting the CSV file
 *
 *        employee_id,email_address,first_name,last_name
 *        1,email1@address1.com,firstName1,lastName1
 *        2,email2@address2.com,firstName2,lastName2
 *        3,email3@address3.com,firstName3,lastName3
 *        4,email4@address4.com,firstName4,lastName4
 *
 * 6. If you have a file with a different structure then the script should
 *    be modified accordingly.
 *
 */

//The burst token is used as a key to identify the details
//of the appropriate employee or customer
def token = ctx.token

//Load and parse the CSV file - Change with the path of your own CSV file
def employees = new File("src/test/resources/input/unit/other/" +
                         "employees.csv").readLines()*.split(",")

println "Processed ${employees.size()} Lines"

def employeeId, emailAddress, firstName, lastName

for (employeeRow in employees) {

    //The burst token is used as a key to identify the details
    //of the appropriate employee or customer
    if (employeeRow[0] ==  token)
    {
        employeeId = employeeRow[0]
        emailAddress = employeeRow[1]
        firstName = employeeRow[2]
        lastName = employeeRow[3]
    }
```

```
}

println "Employee: employee_id = ${employeeId} and" +
        " email_address = ${emailAddress} and" +
        " first_name = ${firstName} and" +
        " last_name = ${lastName}"

//Populate the fetched information into var0, var1, etc.
ctx.variables.setUserVariable(String.valueOf("${token}"),"var0",
                              String.valueOf("${emailAddress}"))

ctx.variables.setUserVariable(String.valueOf("${token}"),"var1",
                              String.valueOf("${firstName}"))

ctx.variables.setUserVariable(String.valueOf("${token}"),"var2",
                              String.valueOf("${lastName}"))
```

Further Reading

- *Groovy documentation* [http://groovy.codehaus.org/] - general Groovy docs which will help for writing better *DocumentBurster* scripts.
- *Ant documentation* [http://ant.apache.org/manual/tasksoverview.html] - In case there is a need to *copy*, *mkdir*, *move*, *delete* files and folders. *Ant* can also be used for sending emails from within scripts or to FTP and SCP files using SSH.
- *AntBuilder documentation* [http://groovy.codehaus.org/Using+Ant+from+Groovy] - Using Ant from Groovy.
- *Commons VFS documentation* [http://commons.apache.org/vfs/filesystems.html] - WebDAV scripting, in case there is a need to upload reports to Microsoft SharePoint or to other portal product. Commons VFS can also be scripted to copy reports to a network shared drive or to upload the reports to FTP and SFTP servers.

Chapter 2. *cURL* Integration

The current chapter is related with both of the previously presented topics

- *Chapter 3. Distributing Reports* from *DocumentBurster User Guide*

  http://www.pdfburst.com/report-distribution-manual.pdf

- and *Chapter 1. Using scripts to achieve more*

This chapter is related with *Chapter 3. Distributing Reports* (User Guide document) in the sense that it will present some advanced report distribution capabilities of *DocumentBurster* and it is related with *Chapter 1. Using scripts to achieve more* in the sense that it is using scripting to achieve these capabilities.

DocumentBurster closely integrates with - *cURL*, a Swiss-army knife for doing data transfer. Through *cURL*, *DocumentBurster* can distribute reports via HTTP or FTP with or without authentication, it works over SSL, and it works without interaction. Actually *cURL* (and thus *DocumentBurster*) supports distributing files and data to a various range of common Internet protocols, currently including HTTP, HTTPS, FTP, FTPS, SCP, SFTP, TFTP, LDAP, LDAPS, DICT, TELNET, FILE, IMAP, POP3, SMTP and RTSP.

cURL - http://curl.haxx.se/

Cross platform

cURL is portable and works on many platforms, including Windows, Linux, Mac OS X, MS-DOS and more.

On Windows, *DocumentBurster* package distribution is bundling together a recent version of *cURL*. So, if your organization is running *DocumentBurster* under Windows, there is nothing more to download or install in regards with *cURL*.

For other UNIX like systems, such as Linux and Mac OS X, the appropriate *cURL* binaries distribution should be properly downloaded and installed. In addition, the *cURL* groovy scripts which are bundled together with *DocumentBurster* are written for Windows usage and should support small adjustments to be made ready for usage under Linux/UNIX.

Command line *cURL* examples

cURL is a tool for getting or sending files using URL syntax. The URL syntax is protocol-dependent. Along with the URL for the required protocol, *cURL* can take some additional options in the command line.

For complete *cURL* documentation you can follow

- *cURL Manual [http://curl.haxx.se/docs/manual.html]*
- *cURL Man Page [http://curl.haxx.se/docs/manpage.html]*
- *cURL Frequently Asked Questions [http://curl.haxx.se/docs/faq.html]*

Following are some sample *cURL* invocations to upload a file to a remote server (from *cURL* manual)

1. FTP / FTPS / SFTP / SCP

Upload data from a specified file, login with user and password

curl -T uploadfile -u user:passwd ftp://ftp.upload.com/myfile

Upload a local file to the remote site, and use the local file name remote too

curl -T uploadfile -u user:passwd ftp://ftp.upload.com/

cURL also supports ftp upload through a proxy, but only if the proxy is configured to allow that kind of tunneling. If it does, you can run *cURL* in a fashion similar to

curl --proxytunnel -x proxy:port -T localfile ftp.upload.com

--ftp-create-dirs

When integrated with *DocumentBurster*, following *cURL* option will be of interest

--ftp-create-dirs - (FTP/SFTP) When an FTP or SFTP URL/operation uses a path that doesn't currently exist on the server, the standard behavior of *cURL* is to fail. Using this option, *cURL* will instead attempt to create missing directories.

2. HTTP

Upload data from a specified file

curl -T uploadfile http://www.upload.com/myfile

Note that the http server must have been configured to accept PUT before this can be done successfully.

Debugging and tracing *cURL* - VERBOSE / DEBUG

If *cURL* fails where it isn't supposed to, if the servers don't let you in, if you can't understand the responses: use the -v flag to get verbose fetching. *cURL* will output lots of info and what it sends and receives in order to let the user see all client-server interaction (but it won't show you the actual data).

curl -v ftp://ftp.upload.com/

To get even more details and information on what *cURL* does, try using the --trace or --trace-ascii options with a given file name to log to, like this

curl --trace trace.txt www.haxx.se

***DocumentBurster* / *cURL* sample scripts**

While it is great to know that so many protocols are supported, *DocumentBurster* is coming with sample scripts to do *cURL* report distribution through the most commonly used protocols such as FTP, SFTP and FILE. Any other *cURL* supported protocol should be achievable by doing little changes to the scripts which are provided in the default *DocumentBurster* package distribution.

curl_ftp.groovy

`curl_ftp.groovy` script is an alternative to *DocumentBurster's* GUI FTP capability which was introduced in *Chapter 3. Distributing Reports* from *DocumentBurster User Guide*. While through the GUI it is possible to achieve common FTP report distribution use cases, using this FTP script is recommended for more advanced FTP scenarios which require the full *cURL* FTP capabilities. For example, using this script it is possible to instruct *DocumentBurster* to automatically create a custom hierarchy of directories on the FTP server, before uploading the reports.

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/curl_ftp.groovy`. By default the script is fetching the values for

the FTP connection , such as user, password, host and path from the values of *$var0$, $var1$, $var2$ and $var3$* user report variables. If the burst reports are configured as such, then there is nothing more to do, and the FTP upload will work without any modification to the script. Otherwise, the FTP script should be modified as per the needs.

While the script might look long, there are actually only few simple lines of active code - most of the content of the script are the comments which are appropriately describing the scope of each section of the script.

```
    /*
 *
 * 1. This script should be used:
 *
 *      1.1 - As a script to upload reports by FTP using cURL.
 *      1.2 - As a sample and starting script to invoke cURL during the
 *      report bursting life cycle.
 *
 * 2. curl is a tool to transfer data from or to a server, using one of the
 *    supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP,
 *    IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS,
 *    TELNET and TFTP).
 *
 *    The command is designed to work without user interaction.
 *
 * 3. curl offers a busload of useful tricks like proxy support,
 *    user authentication, FTP upload, HTTP post, SSL connections, cookies,
 *    file transfer resume and more.
 *
 * 4. The URL syntax is protocol-dependent. You'll find a detailed description
 *    in RFC 3986.
 *
 * 5. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 6. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 7. For a full documentation of the cURL and FTP please see
 *
 *      7.1. http://curl.haxx.se/docs/manual.html
 *      7.2. http://curl.haxx.se/docs/manpage.html
 *
 */

import com.smartwish.documentburster.variables.Variables

/*
 *
 *    The file to be uploaded is the file which has
 *    just been burst.
 *
 */
```

```
def uploadFilePath = ctx.extractFilePath

/*
 *    By default the script is extracting the required FTP
 *    session information from the following sources:
 *
 *      userName - from the content of $var0$ user variable
 *      password - from the content of $var1$ user variable
 *
 *      hostName - from the content of $var2$ user variable
 *      absolutePath - from the content of $var3$ user variable
 *
 */
def userName = ctx.variables.getUserVariables(ctx.token).get("var0")
def password = ctx.variables.getUserVariables(ctx.token).get("var1")

def hostName = ctx.variables.getUserVariables(ctx.token).get("var2")
def absolutePath = ctx.variables.getUserVariables(ctx.token).get("var3")

/*
 *
 *    $execOptions is the command line to be sent for execution to cURL
 *    - see http://curl.haxx.se/docs/manpage.html
 *
 *    --ftp-create-dirs -
 *
 *      (FTP/SFTP) When an FTP or SFTP URL/operation uses a path that
 *      doesn't currently exist on the server, the standard behavior
 *      of curl is to fail.
 *      Using this option, curl will instead attempt to create the
 *      missing directories.
 *
 *    -T, --upload-file <file>
 *
 *      This transfers the specified local file to the remote URL.
 *      If there is no file part in the specified URL, Curl will
 *      append the local file name.
 *      NOTE that you must use a trailing / on the last directory
 *      to really prove to Curl that there is no file name or curl
 *      will think that your last directory name is the remote file
 *      name to use. That will most likely cause the upload
 *      operation to fail.
 *      If this is used on a HTTP(S) server, the PUT command
 *      will be used.
 *
 *    -u, --user <user:password>
 *
 *      Specify the user name and password to use for server authentication.
 *
 *    --trace <file>
 *
 *      Enables a full trace dump of all incoming and outgoing data,
 *      including descriptive information, to the given output file.
 *      Use "-" as filename to have the output sent to stdout.
```

```
*       This option overrides previous uses of -v, --verbose or --trace-ascii.
*       If this option is used several times, the last one will be used.
*
*     --trace-ascii <file>
*
*       Enables a full trace dump of all incoming and outgoing data,
*       including descriptive information, to the given output file.
*       Use "-" as filename to have the output sent to stdout.
*       This is very similar to --trace, but leaves out the hex part
*       and only shows the ASCII part of the dump. It makes smaller
*       output that might be easier to read for untrained humans.
*       This option overrides previous uses of -v, --verbose or --trace.
*       If this option is used several times, the last one will be used.
*
*     --trace-time
*
*       Prepends a time stamp to each trace or verbose line that curl displays.
*       Added in curl 7.14.0)
*
*     -v, --verbose
*
*       Makes the fetching more verbose/talkative.
*       Mostly useful for debugging. A line starting with '>'
*       means "header data" sent by curl, '<' means "header data"
*       received by curl that is hidden in normal cases, and a
*       line starting with '*' means additional info provided
*       by curl.
*       Note that if you only want HTTP headers in the output,
*       -i, --include might be the option you're looking for.
*       If you think this option still doesn't give you enough details,
*       consider using --trace or --trace-ascii instead.
*       This option overrides previous uses of --trace-ascii or --trace.
*       Use -s, --silent to make curl quiet.
*
*     FTPS
*
*       It is just like for FTP, but you may also want to specify and use
*       SSL-specific options for certificates etc.
*       Note that using FTPS:// as prefix is the "implicit" way as
*       described in the standards while the recommended "explicit" way is
*       done by using FTP:// and the --ftp-ssl option.
*
*     SFTP / SCP
*
*       This is similar to FTP, but you can specify a private key to use
*       instead of a password.
*       Note that the private key may itself be protected by a password that is
*       unrelated to the login password of the remote system.
*       If you provide a private key file you must also provide a public key file.
*
*     For more details see:
*
*       1. http://curl.haxx.se/docs/manual.html
*       2. http://curl.haxx.se/docs/manpage.html
```

```
 *
 */
def execOptions =  "--ftp-create-dirs"
execOptions += " -T \"$uploadFilePath\""
execOptions += " -u $userName:$password"
execOptions += " ftp://$hostName/$absolutePath"

def ant = new AntBuilder()

/*
 *    The command executed by curl will be logged in
 *    the logs/DocumentBurster.log file
 */
log.info("Executing command: curl.exe $execOptions")

/*
 *
 *    1. http://groovy.codehaus.org/Executing+External+Processes+From+Groovy
 *    2. cURL is printing its logging operations to the logs/cURL.log file
 *
 */
ant.exec(
  append: "true",
  failonerror: "true",
  output:"logs/cURL.log",
  executable: 'curl/win/curl.exe') {
   arg(line:"$execOptions")
  }
```

curl_sftp.groovy

`curl_sftp.groovy` script can be used to upload the burst reports through Secure File Transfer Protocol or Secure FTP.

With minimum modifications to *$execOptions*, the script can be adapted to use other protocols such as FTPs or SCP. You can check *cURL Manual - cURL usage explained* for more details.

http://curl.haxx.se/docs/manual.html

Edit the script `scripts/burst/endExtractDocument.groovy` with the content found in `scripts/burst/samples/curl_sftp.groovy.` By default the script is fetching the values for the SFTP connection , such as user, password, host and path from the values of *$var0$, $var1$, $var2$ and $var3$* user report variables. If the burst reports are configured as such, then there is nothing more to do, and SFTP uploading will work without any additional modification to the script. Otherwise, this script should be modified as per the needs.

While the script might look long, there are actually only few simple lines of active code - most of the content of the script are the comments which are appropriately describing the scope of each section of the script.

```
    /*
 *
```

```
 * 1. This script should be used:
 *
 *      1.1 - As a script to upload reports by SFTP using cURL.
 *      1.2 - As a sample and starting script to invoke cURL during the
 *      report bursting life cycle.
 *
 * 2. curl is a tool to transfer data from or to a server, using one of the
 *    supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP,
 *    IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS,
 *    TELNET and TFTP).
 *
 *    The command is designed to work without user interaction.
 *
 * 3. curl offers a busload of useful tricks like proxy support,
 *    user authentication, FTP upload, HTTP post, SSL connections, cookies,
 *    file transfer resume and more.
 *
 * 4. The URL syntax is protocol-dependent. You'll find a detailed description
 *    in RFC 3986.
 *
 * 5. The script should be executed during the endExtractDocument
 *    report bursting lifecycle phase.
 *
 * 6. Please copy and paste the content of this sample script
 *    into the existing scripts/burst/endExtractDocument.groovy
 *    script.
 *
 * 7. For a full documentation of the cURL and FTP please see
 *
 *      7.1. http://curl.haxx.se/docs/manual.html
 *      7.2. http://curl.haxx.se/docs/manpage.html
 *
 */

import com.smartwish.documentburster.variables.Variables


/*
 *
 *    The file to be uploaded is the file which has
 *    just been burst.
 *
 */
def uploadFilePath = ctx.extractFilePath

/*
 *    By default the script is extracting the required SFTP
 *    session information from the following sources:
 *
 *      userName - from the content of $var0$ user variable
 *      password - from the content of $var1$ user variable
 *
 *      hostName - from the content of $var2$ user variable
 *      absolutePath - from the content of $var3$ user variable
```

```
 *
 */
def userName = ctx.variables.getUserVariables(ctx.token).get("var0")
def password = ctx.variables.getUserVariables(ctx.token).get("var1")

def hostName = ctx.variables.getUserVariables(ctx.token).get("var2")
def absolutePath = ctx.variables.getUserVariables(ctx.token).get("var3")

/*
 *
 *    $execOptions is the command line to be sent for execution to cURL
 *    - see http://curl.haxx.se/docs/manpage.html
 *
 *    --ftp-create-dirs -
 *
 *      (FTP/SFTP) When an FTP or SFTP URL/operation uses a path that
 *      doesn't currently exist on the server, the standard behavior
 *      of curl is to fail.
 *      Using this option, curl will instead attempt to create
 *      missing directories.
 *
 *    -T, --upload-file <file>
 *
 *      This transfers the specified local file to the remote URL.
 *      If there is no file part in the specified URL, Curl will
 *      append the local file name.
 *      NOTE that you must use a trailing / on the last directory
 *      to really prove to Curl that there is no file name or curl
 *      will think that your last directory name is the remote file
 *      name to use. That will most likely cause the upload
 *      operation to fail.
 *      If this is used on a HTTP(S) server, the PUT command
 *      will be used.
 *
 *    -u, --user <user:password>
 *
 *      Specify the user name and password to use for server authentication.
 *
 *    --trace <file>
 *
 *      Enables a full trace dump of all incoming and outgoing data,
 *      including descriptive information, to the given output file.
 *      Use "-" as filename to have the output sent to stdout.
 *      This option overrides previous uses of -v, --verbose or --trace-ascii.
 *      If this option is used several times, the last one will be used.
 *
 *    --trace-ascii <file>
 *
 *      Enables a full trace dump of all incoming and outgoing data,
 *      including descriptive information, to the given output file.
 *      Use "-" as filename to have the output sent to stdout.
 *      This is very similar to --trace, but leaves out the hex part
 *      and only shows the ASCII part of the dump. It makes smaller
 *      output that might be easier to read for untrained humans.
```

```
 *      This option overrides previous uses of -v, --verbose or --trace.
 *      If this option is used several times, the last one will be used.
 *
 *    --trace-time
 *
 *      Prepends a time stamp to each trace or verbose line that curl
 *      displays.
 *      Added in curl 7.14.0)
 *
 *    -v, --verbose
 *
 *      Makes the fetching more verbose/talkative.
 *      Mostly useful for debugging. A line starting with '>'
 *      means "header data" sent by curl, '<' means "header data"
 *      received by curl that is hidden in normal cases, and a
 *      line starting with '*' means additional info provided by curl.
 *      Note that if you only want HTTP headers in the output,
 *      -i, --include might be the option you're looking for.
 *      If you think this option still doesn't give you enough details,
 *      consider using --trace or --trace-ascii instead.
 *      This option overrides previous uses of --trace-ascii or --trace.
 *      Use -s, --silent to make curl quiet.
 *
 *    FTPS
 *
 *      It is just like for FTP, but you may also want to specify and use
 *      SSL-specific options for certificates etc.
 *      Note that using FTPS:// as prefix is the "implicit" way as
 *      described in the standards while the recommended "explicit" way is
 *      done by using FTP:// and the --ftp-ssl option.
 *
 *    SFTP / SCP
 *
 *      This is similar to FTP, but you can specify a private key to use
 *      instead of a password.
 *      Note that the private key may itself be protected by a password that is
 *      unrelated to the login password of the remote system.
 *      If you provide a private key file you must also provide
 *      a public key file.
 *
 *    For more details see:
 *
 *      1. http://curl.haxx.se/docs/manual.html
 *      2. http://curl.haxx.se/docs/manpage.html
 *
 */
def execOptions =  "-T \"$uploadFilePath\""
execOptions += " -u $userName:$password"
execOptions += " sftp://$hostName/$absolutePath"

def ant = new AntBuilder()

/*
 *
```

```
 *    The command executed by curl will be logged in
 *    the logs/DocumentBurster.log file
 *
 */
log.info("Executing command: curl.exe $execOptions")

/*
 *
 *    1. http://groovy.codehaus.org/Executing+External+Processes+From+Groovy
 *    2. cURL is printing its logging operations to the logs/cURL.log file
 *
 */
ant.exec(
  append: "true",
  failonerror: "true",
  output:"logs/cURL.log",
  executable: 'curl/win/curl.exe') {
   arg(line:"$execOptions")
  }
```