PharmaSUG 2019 - Paper PO-274

A Macro to Expand Encrypted Zip Files on the SAS LSAF Environment

Steven Hege, Alexion Pharmaceuticals, Inc.

ABSTRACT

This paper will introduce a short SAS macro that uses java objects to expand zip archive files located on the SAS Life Science Analytics Framework (LSAF). This macro runs on LSAF storing files directly on the environment in a specified folder and can expand password encrypted archives. Our group has found this macro useful when handling data archives uploaded directly into LSAF.

INTRODUCTION

A common practice in the pharmaceutical industry is for CRO's and other vendors will submit raw data in zipped archive files often password protected for security reasons. The process for using the data requires the recipient organization to decrypt and extract the contents of the zip files onto a SAS environment.

Until recently, we would download these files onto a user's laptop computer to decrypt and expand the archive contents. After expansion, the programmer uploads the individual files into the LSAF system into the appropriate folder. This method adds extra steps to the process requiring programmer time for each step and a constant and reliable internet connection during the download and upload parts. Also, this method brought up questions regarding data integrity and file security. The %GLSAFUNZIP macro presented in this paper provides a solution that helps resolve these issues.

The %GLSAFUNZIP macro is a means for expanding zip archive files with an option of decrypting password protected files. The macro employs the SAS LSAF macro 1.8 API, and a data step java object or JavaObj (DeVenezia, 2005) built using open source ZIP4J Java library (Lingala 2013) and JDK 1.7 for a SAS 9.4 environment. The java object is contained in a fat JAR file along with all other required code and is usually located in the same folder with the macro. The java object provides several methods to set and return parameters values, perform decrypting and archive expansion, and return exception messages. While much of the macro is unique to the LSAF environment with a repository and our folder structure, a programmer can revise it for other systems.

MACRO PARAMETERS AND CALL

Within our LSAF environment, the %GLSAFUNZIP is non-specific to a particular project, and is callable within a wrapper program, a LSAF workspace job, or part of a data extraction program. The macro has 3 parameters where only the first one listed is required.

ZIPFILE:	Name with path of the zip archive to process (Required).
OUTPUTDEST:	Path of the destination folder for writing the files. If not given, the files are written to the same folder containing the zip file. (Optional)
ZIPPASSWORD:	Password for decrypting a protected archive if needed. If not present, the file is processed as unencypted. (Optional)

An example of calling it in a LSAF environment is:

```
%glsafunzip(zipfile=%str(/alxn/TestJava/test_glsafunzip/Files/glsafunzip example.zip),
            outputdest=%str(/alxn/TestJava/test_glsafunzip/Files),
            zippassword=%str(sasisgreat));
```

In this example, the password protected archive "…/glsafunzip example.zip" is decrypted with password "sasisgreat" and expanded into the "…/Files" folder. See Figure 1. Example folder showing before and after of using the %GLSAFUNZIP macro.

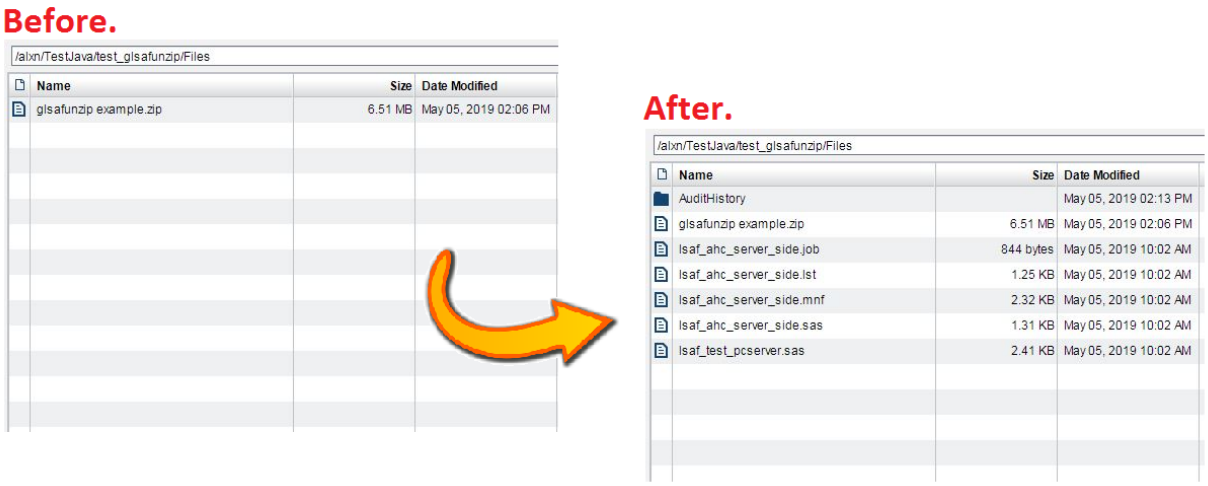

Figure 1. Example folder showing before and after of using the %GLSAFUNZIP macro.

CONCLUSION

The SAS macro %GLSAFUNZIP provides a solution for expanding zip archive files on the LSAF environment. Our use of this macro solves several issues within our organization. In addition to resolving data integrity and file security issues, we found it useful for speeding up uploading data and other files especially large datasets, and, coupled with its counterpart %GLSAFZIP, for saving space in our online SAS services. The %GLSAFZIP macro testing was not complete before submitting this paper. Once we complete testing, we anticipate both macros will be an integral part of our SAS programming environment saving both programmer time and disk space.

REFERENCES

Richard A. DeVenezia, April 2005, *Java in SAS® - JavaObj, a DATA Step Component Object*, SUGI 30 Proceedings, Philadelphia, PA. SAS Institute, Inc., https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/241-30.pdf

Srikanth Reddy Lingala. 2013. Zip4J – Java library to handle Zip files, http://www.lingala.net/zip4j/about.php

ACKNOWLEDGMENTS

The author would like to thank Greg Weber for help during testing and his suggestions for improving the macro.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Steve Hege
Alexion Pharmaceuticals, Inc.
Steven.Hege@alexion.com

APPENDIX

SAS MACRO CODE

```
/*************************************************************************************
Program : glsafunzip
Purpose : Utility to unzip a zip archive within LSAF with ability to expand password protected files
Date    : 19Sep2018
Author  : Steve Hege
SAS Vers: 9.2 or higher
Inputs  : Zip archive file
          glsafunzip-2.0.jar
Outputs : All files in zip archive
*************************************************************************************
Notes:

Parameter:
   zipfile: Full path including extension of zip file to expand (required).
   outputdest: Destination LSAF output folder (required).
   zippassword: Password of zip file if password protected (optional).

*************************************************************************************/

%macro glsafunzip(zipfile=,outputdest=,zippassword=);

%if %sysfunc(fileexist(&_sasws_.&zipfile)) = 0 %then %do;
   %lsaf_objectexists(lsaf_path=&zipfile.);
   %if &_lsafObjectExists_ = 1 %then %do;
      %lsaf_syncfiletoworkspace(lsaf_path=&zipfile., lsaf_version=);
   %end;
   %else %do;
      %let ER1=ER;
      %put ============================================================================================================;
      %put &ER1.ROR (GLSAFUNZIP): The external zip file &zipfile does not exist in repository or workspace.;
      %put ============================================================================================================;
      %abort;
   %end;
%end;

%else %if %sysfunc(fileexist(&_sasws_.&zipfile)) %then %do;

data _null;
   length tmpstr $2048;
   tmpstr="&_sasws_.";
   if index(tmpstr,".transient") gt 0 then
      call symput("reporun","Y");
   else
      call symput("reporun","N");
run;

%if "&outputdest." eq "" %then %do;
data _null_;
   length temppath $200;
   temppath="&zipfile.";
   tmpindx=find(temppath,"/",-length(temppath));
   temppath=substr(temppath,1,tmpindx-1);
   call symput("outputdest",strip(temppath));
run;
%put &outputdest;
%end;

data _null_;
   length tmpdest $2048;
   tmpdest="&outputdest.";
   if substr(tmpdest,length(tmpdest),1) eq '/' then do;
      tmpdest=substr(tmpdest,1,length(tmpdest)-1);
      call symput("outputdest",trim(tmpdest));
   end;
run;
%put &outputdest;

%if %index(&outputdest,/Files) = 0 %then %do;
   %let ER1=ER;
   %put &ER1.ROR: The folder output destination path does not allow folder creation. It must be within a
'/Files/' area on LSAF;
   %abort;
%end;

DATA _null_;
   length  path_separator $ 2 orig_classpath $ 500;
```

```
   declare JavaObj f("java.io.File", "");
   f.getStaticStringField("pathSeparator", path_separator);

   orig_classpath = strip(SYSGET("CLASSPATH"));

   if _error_ = 1 or length(orig_classpath) = 0 then do;
      put "NOTE: Ignore any messages from the next statement(s)";
      orig_classpath = "";
   end;

   call symputx('CP_orig_classpath', strip(orig_classpath), 'GLOBAL');
   call symputx('CP_path_separator', compress(path_separator), 'GLOBAL');
RUN;

%let cp_addition=&_sasws_/alxn/Files/macros/glsafunzip-3.0.jar;

DATA _null_;
   length  current_classpath $ 500 new_classpath $ 500;

   current_classpath = strip(SYSGET("CLASSPATH"));

   if _error_ = 1 or length(current_classpath) = 0 then do;
      put "NOTE: Ignore any messages from the nearby statement(s)";
      new_classpath = "&cp_addition";
    end;
   else do;
      new_classpath = COMPRESS(current_classpath) || "&CP_path_separator" || "&cp_addition";
   end;

   call symputx('CP_new_classpath', strip(new_classpath), 'GLOBAL');
run;

%PUT NOTE: Setting Java classpath to &CP_new_classpath;
options set=classpath "&CP_new_classpath";

%global maxfile;

/* DATA step code */
data _null_;

   length ExtractException OpenException $2048;

   dcl javaobj j("glsafunzip/GLsafUnzip");

   rc1=j.callVoidMethod("setZipFilePath", "&_sasws_.&zipfile.");

   if "&reporun." eq "Y" then do;
      rc3=j.callVoidMethod("setOutputPath", "&_sasws_.&outputdest./0");
    end;
   else do;
      rc3=j.callVoidMethod("setOutputPath", "&_sasws_.&outputdest.");
   end;

   if not missing("&zippassword") then do;
      rc4=j.callVoidMethod("setZipFilePwd", "&zippassword.");
   end;

   rc5=j.callVoidMethod("openZipFile");

   rc5exc=j.callStringMethod("getZipException",OpenException);

   length er1 $2100 testresult 8;
   testresult = .;
   call symput("ExceptAction","GO");
   if missing(OpenException) then do;
      rc6=j.callBooleanMethod("IsFileEncrypted",IsFileEncrypted);
      if IsFileEncrypted = 1 and not missing("&zippassword") then do;
         rc61=j.callIntMethod("verifyZipPassword",testresult);
      end;
      if (missing("&zippassword") and IsFileEncrypted = 1) then do;
         put "================================================================================================";
         er1=cats("ER","ROR (GLSAFUNZIP): Zip file is encrypted but password parameter is missing.");
         put er1;
         put "================================================================================================";
         call symput("ExceptAction","ABORT");
       end;
      else if IsFileEncrypted = 1 and not missing("&zippassword") and testresult = 0 then do;
         put "================================================================================================";
         er1=cats("ER","ROR (GLSAFUNZIP): Zip file is encrypted but wrong password parameter given.");
         put er1;
```

```
         put "================================================================================================";
         call symput("ExceptAction","ABORT");
       end;
      else do;
         rc7=j.callVoidMethod("ExtractAllFiles");
         rc8=j.callStringMethod("getZipException",ExtractException);
         if not missing(ExtractException) then do;
            put "================================================================================================";
            er1=cats("ER","ROR (GLSAFUNZIP): ZipException while extracting Zip File.");
            put er1;
            er1=cat("ER","ROR (GLSAFUNZIP): ",left(scan(ExtractException,4,':')));
            put er1;
            put "================================================================================================";
            call symput("ExceptAction","ABORT");
         end;
      end;
    end;
   else do;
      put "================================================================================================";
      er1=cats("ER","ROR (GLSAFUNZIP): Exception while opening Zip File.");
      put er1;
      er1=cat("ER","ROR (GLSAFUNZIP): ",left(scan(OpenException,4,':')));
      put er1;
      put "================================================================================================";
      call symput("ExceptAction","ABORT");
   end;

   rc8=j.callVoidMethod("resetZipParams");
run;

%PUT NOTE: Setting Java classpath back to its original state: &CP_orig_classpath;
options set=classpath "&CP_orig_classpath";

%if "&ExceptAction" = "ABORT" %then %do;
   %abort;
%end;

%macro driveList(dir);
   %local filrf filrf2 rc did did2 memcnt name i;

   /* Assigns a fileref to the directory and opens the directory */
   %let rc=%sysfunc(filename(filrf,&dir));
   %let did=%sysfunc(dopen(&filrf));

   /* Make sure directory can be open */
   %if &did eq 0 %then %do;
      %put Directory &dir cannot be open or does not exist;
      %return;
   %end;

   /* Loops through entire directory */
   %do i = 1 %to %sysfunc(dnum(&did));

     /* Retrieve name of each file */
     %let name=%qsysfunc(dread(&did,&i));
     %let rc=%sysfunc(filename(filrf2,&dir/&name));
     %let did2=%sysfunc(dopen(&filrf2));

     /* Checks to see if the extension matches the parameter value */
     /* If condition is true print the full name to the log        */
      %if &did2 eq 0 %then %do;
        data temp;
           length fullpath $2048 name $256;
           fullpath = "&dir/&name";
           name = "&name";
        run;
        data _allUnzippedfiles;
           set _allUnzippedfiles temp;
        run;
      %end;
     /* If directory name call macro again */
      %else %if &did2 gt 0 %then %do;
        %driveList(&dir/%unquote(&name));
        %let rc=%sysfunc(dclose(&did2));
        %let rc=%sysfunc(filename(filrf2));
      %end;

   %end;

   proc datasets lib=work nolist;
      delete temp;
```

```
   quit;

   /* Closes the directory and clear the fileref */
   %let rc=%sysfunc(dclose(&did));
   %let rc=%sysfunc(filename(filrf));
 %mend driveList;

%if "&reporun." eq "Y" %then %do;

data _allUnzippedfiles;
   length fullpath $2048 name $256;
   fullpath = ""; name="";
   delete;
run;

%driveList(dir=%str(&_sasws_.&outputdest./0));

data _allUnzippedfiles2;
   set _allUnzippedfiles;
   length newpath workpath $2048;
   newpath=tranwrd(fullpath,'/0/','/');
   workpath=trim(tranwrd(newpath,"&_sasws_",""));
run;

%macro copyOverFiles(fullpath=,newpath=);
filename in "&fullpath";
filename out "&newpath";

/* copy the file byte-for-byte  */
data _null_;
  length filein 8 fileid 8;
  filein = fopen('in','I',1,'B');
  fileid = fopen('out','O',1,'B');
  rec = '20'x;
  do while(fread(filein)=0);
     rc = fget(filein,rec,1);
     rc = fput(fileid, rec);
     rc =fwrite(fileid);
  end;
  rc = fclose(filein);
  rc = fclose(fileid);
run;

filename in clear;
filename out clear;

%mend copyOverFiles;

data _null_;
   set _allUnzippedfiles2;
   length cmdstr $4096;
   cmdstr='%copyOverFiles(fullpath='||trim(fullpath)||',newpath='||trim(newpath)||');';
   call execute(cmdstr);
run;

data delfldrs(keep=folder);
   set _allUnzippedfiles2 end=lastone;
   length folder $2048;
   folder=substr(fullpath,1,find(fullpath,'/',-length(fullpath))-1);
   fname="deleteme";
   rc=filename(fname,fullpath);
   if rc = 0 and fexist(fname) then
      rc=fdelete(fname);
   rc=filename(fname);
   output;
   if lastone then do;
      folder="&_sasws_.&outputdest./0";
      output;
   end;
run;

proc sort data=delfldrs out=delfldrs2 nodupkeys;
   by descending folder;
run;

data _null;
   set delfldrs2;
   fname="delfldr";
   rc=filename(fname,folder);
   if rc = 0 then rc2=fdelete(fname);
   rc=filename(fname);
```

```
   msg=sysmsg();
   if rc2 ^= 0 then put msg=;
run;

%end; /* %if "&reporun." eq "Y" %then %do;*/

%end; /* %if %sysfunc(fileexist(&_sasws_.&zipfile)) %then %do;*/
%else %do;
   %let ER1=ER;
   %put ============================================================================================;
   %put &ER1.ROR(GLSAFUNZIP): The external zip file &zipfile does not exist or there was an issue copying.;
   %put ============================================================================================;
   **this is to generate actual red err message in LSAF job so that job actually fails;
   %abort;
%end;

%mend glsafunzip;
```

JAVA SOURCE CODE

```
package glsafunzip;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;

import net.lingala.zip4j.core.ZipFile;
import net.lingala.zip4j.exception.ZipException;
import net.lingala.zip4j.model.FileHeader;

/**
* @author Steve Hege, 19Sep2018
 */

public class GLsafUnzip {

    private String ZipFilePath = "";
    private String ZipFilePwd = "";
    private String OutputPath = "";
    private List fileHeaderList;
    private ZipFile zipFile;
    private FileHeader fileHeader;
    private boolean isEncrypted;
    private boolean isSplitArchive;
    private boolean isValidZipFile;
    private String zipException = "";

    public boolean IsFileEncrypted () throws ZipException {
        return isEncrypted;
    }
    public boolean IsFileSplitArchive () throws ZipException {
        return isSplitArchive;
    }
    public boolean IsFilevalidZip () throws ZipException {
        return isValidZipFile;
    }

    public void openZipFile () {
        this.zipException = "";
        try {
            // Initiate ZipFile object with the path/name of the zip file.
            zipFile = new ZipFile(ZipFilePath);
            isEncrypted = zipFile.isEncrypted();
            isSplitArchive = zipFile.isSplitArchive();
            isValidZipFile = zipFile.isValidZipFile();
            if (isEncrypted) {
                zipFile.setPassword(ZipFilePwd);
            }
        } catch (ZipException x) {
            this.zipException = x.toString();
        }
    }

    public int numberFilesInZipFile() {
        this.zipException = "";
        int numberFound = 0;
        try {
            // Get the list of file headers from the zip file
            fileHeaderList = zipFile.getFileHeaders();

            numberFound = fileHeaderList.size();

        } catch (ZipException x) {
            this.zipException = x.toString();
        }
        return numberFound;
    }
```

```
    public int verifyZipPassword() {
        int testResult = -1;
        try {
            List<FileHeader> fileHeaders = zipFile.getFileHeaders();

//            System.out.println("Zip Password: "+this.ZipFilePwd);

            for(FileHeader fileHdr : fileHeaders) {
                try {
                    InputStream is = zipFile.getInputStream(fileHdr);
                    byte[] b = new byte[4 * 4096];
                    while (is.read(b) != -1) {
                        //Do nothing as we just want to verify password
                    }
                    is.close();
                    testResult=1;
                } catch (ZipException x) {
                    this.zipException = x.toString();
                    testResult=0;
/*                    if (x.getCode() == ZipExceptionConstants.WRONG_PASSWORD) {
                      System.out.println("Wrong password: ");
                    }*/
                } catch (IOException x) {
                    this.zipException = x.toString();
                    testResult=0;
//                    System.out.println("Most probably wrong password.");
                }
            }
        } catch (Exception x) {
            this.zipException = x.toString();
            testResult=0;
//            System.out.println("Some other exception occurred");
//            x.printStackTrace();
        }
        return testResult;
    }

    public String getFileNameInZipFile (String istr) {
        this.zipException = "";
        try {
            // Get the list of file headers from the zip file
            fileHeaderList = zipFile.getFileHeaders();

            int i = Integer.parseInt(istr);

            fileHeader = (FileHeader)fileHeaderList.get(i);

        } catch (ZipException x) {
            this.zipException = x.toString();
        }
        return fileHeader.getFileName();
    }

    public void ExtractAllFiles() {
        this.zipException = "";
        try {
            // Check to see if the zip file is password protected
            if (zipFile.isEncrypted()) {
                    // if yes, then set the password for the zip file
                    zipFile.setPassword(ZipFilePwd);
            }

            // Extracts all files to the path specified
            zipFile.extractAll(OutputPath);

        } catch (ZipException x) {
            this.zipException = x.toString();
        }
    }
```

```
    public void resetZipParams() {
        this.zipException = "";
        this.ZipFilePath = "";
        this.OutputPath = "";
        this.ZipFilePwd = "";
    }

    public void setZipFilePath(String filepath) {
        this.ZipFilePath = filepath;
    }

    public String getZipFilePath() {
        return this.ZipFilePath;
    }

    public void setZipFilePwd(String Pword) {
        this.ZipFilePwd = Pword;
    }

    public String getZipFilePwd() {
        return this.ZipFilePwd;
    }

    public void setOutputPath(String outPath) {
        this.OutputPath = outPath;
    }

    public String getOutputPath() {
        return this.OutputPath;
    }

    public String getZipException() {
        return this.zipException;
    }

    public void clearZipException() {
        this.zipException = "";
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
            GLsafUnzip gu = new GLsafUnzip();
    }
}
```