IES PRINT WRITER: PDF MAIL RUNS – SHELL PROGRAM

Purpose

The purpose of this Document is to provide the Developer with a basic shell program that includes the steps for setting up a pdf mail run, and which may be expanded by inclusion of the necessary logic to fit your requirements.

Introduction

To set up a “PDF mail run” in which various documents are created and dispatched by e-mail to multiple addresses, you will write a program which is then placed in the Menu Process Library, from where it may be offered on a standard User Menu for execution.

The program will perform all required steps, and in this document, we show such a shell program. The shell program is incomplete, i.e. you have to expand the logic to include what you wish to achieve, but it is quite easy to do so when using the shell or ‘template’ program as a basis. This program is called ‘do.pdf.mail.run’ and we can find the source in an IES Business Datamart in the file “Client.bp”. From here, we can copy it to a new name in “client.bp”, modify and expand the program, and then place it in the Menu Process Library for use.

The Shell Program

```
subroutine do.pdf.mail.run
*
  $include subcommon.t
*
*
*     this is a shell or template program only, and is intended
*     as a guide to the developer to develop a 'pdf mail run'
*     program that employs printwriter and docwriter to create
*     documents (statements or other reports), which are then
*     converted to pdf and dispatched seamlessly by e-mail.
*
*
*     for this program to work properly, it will be a requirement
*     that pdfcreator is present as a printer on the workstation
*     where this is executed, and the mail client (e.g. outlook, express
*     or other) should be set with permission to send e-mail on behalf
*     of an application without asking permission each time ...
*
*** *** * declare the files that will be written or read

***
***
***
***
```

```
*** *** * determine the usercode and port, which will be the key
*        to use on file "docwritespre"

***
***
***
  open '','DOCWRITESPRE' to fbxdocwritespre else
    vfbx = status() : ":" : "docwritespre"
    call file_fail_exit(vfbx)
    end
  open '','CUSTFILE' to fbxcustfile else
    vfbx = status() : ":" : "custfile"
    call file_fail_exit(vfbx)
    end
  open '','FAILFILE' to fbxfailfile else
    vfbx = status() : ":" : "failfile"
    call file_fail_exit(vfbx)
    end
  open '','DATACAP' to fbxdatacap else
    vfbx = status() : ":" : "datacap"
    call file_fail_exit(vfbx)
    end
***
  param = '26'
  port = ''
  call s.peek(param,port)
  param = '6'
  usercode = ''
  call s.peek(param,usercode)


*** *** * if there is a need to offer a screen for the user to capture
*        some required values, we may call the screen 1st - in our example
*        we are using a fictional screen called 'screen1' which is also
*        present in the menu process library as 'screen1'

  ref = 'screen1'
  ermes = ''
  call s.logi(ref,ermess)
  * and maybe we read back the result of what is captured; in this
  * example, we suggest there is a 'statement header' that we need
  read datacap from fbxdatacap,port then
    stateheader = ''
    end else
    * the user never saved the screen
    go mainexit
    end


*** *** * it may be necessary to offer the user a data selector to select
*        only those records intended for use in this run; for example
*        we could say that we select records from 'custfile', and there
*        is a data selector also that is called 'custfile'
```

```
  dselector = 'custfile'
  call makelist.t(dselector)
  command = 'get-list portlist*':port
  execute command
  if system(11) else
    * no records selected
    go mainexit
    end
  selecte to klist


*** *** * now we can use the selected records and loop though them to
*       produce and mail the document in each case

nexrec: *
  readnext custkey from klist then
    gosub perform.doc
    go nexrec
    end

*** *** * transfer to program exit
go mainexit

*** *** * beginning of subroutine section

perform.doc: *
  * for each document we perform, we will call a printwriter process,
  * which is already present in the menu process library

  * in each case, printwriter will be calling the designated docwriter
  * process, which may already include a prerun program to format
  * some data and record it in "docwritespre" to be accessed by docwriter
  * to produce the resulting report, or we may call a program from here,
  * for example a program that formats the necessary customer data into
  * docwritespre, and when docwriter is called, the data will already
  * be there

  * before we call the printwriter for the current customer, we need
  * to pass instructions into 'docwritespre' to provide the current
  * e-mail address and other parameters for the mail to be dispatched

  * and after calling printwriter we may want to check the result, i.e.
  * if there was an error or not


  * we call another program we have already written (not supplied here)
  * and which will format some cust data into docwritespre using key=
  * usercode
  call myformatprog(custkey)

  * now we read the customer record to determine the e-mail address
  read custfile from fbxcustfile,custkey then
    email = ''
    end else
    * how can the rec not be there?
    go slip.perform.doc
    end

  * the docwriter object will use data on the rec in 'docwritespre' where
  * the key=usercode (and this data was already formatted when we called
  * 'myformatprog', and now we will pass the necessary instructions for
  * printwriter, also into 'docwritespre', but using key=port.

  x = ''
  x<1> = email ; * email address
  x<2> = 'your statement: ':stateheader ; * mail title
  * x<3> may be used for message text, and may be multi-valued, i.e.
  *      multiple lines of text - we are not doing it here
  x<4> = 'statement' ; * this is the name that will be assigned to
                   * to the attached pdf, it needs to be a valid
                   * filename, and the system will put the '.pdf'
  x<8> = 'mailing statement for customer ':custkey ; * message to be
      * displayed on screen while processing

  docwritespre = x
  write docwritespre on fbxdocwritespre,port

  * at this stage we can call printwriter, which process we have
  * predefined in the menu process library as 'cstate-pdf'
  ref = 'cstate-pdf'
  ermes = ''
  call s.logi(ref,ermes)

  * by now, the mail has gone out or has failed
  *
  read docwritespre from fbxdocwritespre,port then
    x = docwritespre
    * if there is a value on field 5, it was successful
    senddate = x<5>
    if senddate = '' then
      * failed, the error will be on field 7
      failmes = x<7>
      * we can record the failure on some error file and provide
      * a report later if we want - code not shown
      end else
      sendtime = x<6>
      * we can record the success perhaps on the customer rec
      * for future enquiry - code not shown
      end
    end

slip.perform.doc: *

return
```

```
*** *** * end of subroutine section

*
mainexit: *
*
return
```