LEXMARK™

Document Distributor

Script Examples

To view or download this or other Lexmark Document Solutions publications, **click here.**

October 2004 **www.lexmark.com**

Edition: October 2004

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in later editions. Improvements or changes in the products or the programs described may be made at any time.

Comments about this publication may be addressed to Lexmark International, Inc., Department F95/032-2, 740 West New Circle Road, Lexington, Kentucky 40550, U.S.A. In the United Kingdom and Eire, send to Lexmark International Ltd., Marketing and Services Department, Westhorpe House, Westhorpe, Marlow Bucks SL7 3RQ. Lexmark may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. You can purchase additional copies of publications related to this product by calling 1-800-553-9727. In the United Kingdom and Eire, call +44 (0)8704 440 044. In other countries, contact your point of purchase.

References in this publication to products, programs, or services do not imply that the manufacturer intends to make these available in all countries in which it operates. Any reference to a product, program, or service is not intended to state or imply that only that product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any existing intellectual property right may be used instead. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by the manufacturer, are the user’s responsibility.

ImageQuick, Optra, Lexmark, and Lexmark with diamond design are trademarks of Lexmark International, Inc. registered in the United States and/or other countries.

PostScript® is a registered trademark of Adobe Systems Incorporated.

Other trademarks are the property of their respective owners.

Table of contents

1 Introduction

This book provides examples of the constructs and objects that make up the script language used by the Lexmark Document Solutions suite. The examples have been divided into three chapters; each chapter corresponds to a different member of the application suite. You may choose to review all of them or only those that apply to your application.

All the examples in this book are included with the Workflow Composer software. It is recommended that you have access to the Workflow Composer and the Server Manager or Document Portal Admin so that you can work with the scripts as you follow along in the book. When uploading the scripts to the Document Server, you will be prompted for any environment-specific information necessary to execute the scripts. For example, when the IP address of the SMTP server is necessary, the script will show "local.mailserver". This will be replaced by the value you provide.

2 Document Distributor Scripts

Example 1: Saving to a file

Overview: Save the scanned document to a file.

Details: A worker needs to scan invoices at an MFP and have them stored on the local file system.

Script:

```
with SaveToFile
   .input=original.document
   .Path="c:\lexmark\invoice.tif"
   .AppendTimestamp=TRUE
   .CreateDirectory=TRUE
   .Go()
endwith
```

To run this script:

1. Upload it to the server using the Workflow Composer or the Server Manager.
2. Using the Server Manager, create a profile on an MFP.

Note: By setting AppendTimestamp to TRUE, a unique filename is guaranteed for each saved file. The filename generated will be:
`c:\lexmark\invoice_MON_DD_YYYY_HH_MM_SS_MS.tif`

It is important to note that the same timestamp is used for the entire script; so, calling SaveToFile multiple times in the same script execution will use the same timestamp. If this is necessary, the timestamp can be built manually using the usertime property of the original object and appending some unique value. For an example of this, see Example 5: Separating Pages Using Barcodes.

By setting the CreateDirectory property to TRUE, the action will create any directories in the path that do not exist. This is especially useful if the timestamp itself will be used as part of the directory name.

Example 2: Printing a document

Overview: Print a scanned document to another printer.

Details: A worker needs to send an internal fax; a fax from one company location to another. Since there is a company-wide LAN and printers are available in each location, this script prints directly to the printers instead of sending a fax. This provides a higher quality output and saves on phone charges. In addition, the timestamp is added to the printout to make it more "fax-like."

Script:

1. Use the separate method to break up the usertime value into individual parts. Then build up the information string that is stamped on each page of the print out. The format of the string is "Sent by `IP address` on `date` at `time`"

```
stringarray time = original.usertime.Separate("_")
string stamp = "Sent by " + original.userip
stamp.Concatenate(" on " + time[2] + " " + time[3] + " " + time[4])
stamp.Concatenate(" at " + time[5] + ":" + time[6])
```

2. Use the ImageText action to add the timestamp information on the bottom right corner of all pages being sent.

```
with ImageText
   .input=original.document
   .Orientation=LDD_ORIENTATION_PORTRAIT
   .Text=stamp
   .Direction=LDD_IMGTEXTDIR_HORIZONTAL
   .Font=LDD_IMGTEXTFONT_ARIAL
   .Color=LDD_IMGTEXTCOLOR_BLACK
   .Position=LDD_IMGTEXTPOS_BOTTOMRIGHT
   .AllPages=TRUE
   .Size=10
   .Bold=TRUE
   .Italic=TRUE
   .Underline=FALSE
   .Go()
endwith
```

3. Use the ConvertToDocument action to convert the TIFF image to PostScript so that it can be printed. Note that we are using the output from the ImageText action so that we get the TIFF image with the timestamp information.

```
with ConvertToDocument
   .input=ImageText.output
   .Format=LDD_DOCFORMAT_PS
   .AllPages=TRUE
   .PaperSize=LDD_PAPERSIZE_LETTER
   .Orientation=LDD_ORIENTATION_AUTO
   .Go()
endwith
```

4. Use the PrintJob action to send the PostScript file to the specified printer. If the printer is installed with an ImageQuick Option, the TIFF file could be sent directly to the printer without converting it to PostScript first. A local setting, with a name of `printerip` and a "Type" set equal to "Text", was defined to be used within the `.IPAddress` property of the PrintJob action. For more information, see the “Local Settings” help in the Workflow Composer application.

```
with PrintJob
   .input=ConvertToDocument.output
   .IPAddress=local.printerip
   .Copies=1
   .SetWaitTimeout(FALSE)
   .PaperSource=LDD_PRINTER_DEFAULT
   .PaperSize=LDD_PRINTER_DEFAULT
   .OutputBin=LDD_PRINTER_DEFAULT
   .Duplex=LDD_DUPLEX_OFF
   .Orientation=LDD_PRINTER_DEFAULT
   .Nup=LDD_NUP_OFF
   .Collate=FALSE
   .Staple=FALSE
   .HolePunch=FALSE
   .TonerSaver=FALSE
   .SetConfidentialPrint(FALSE)
   .Go()
endwith
```

To run this script:

1. Upload it to the server using the Workflow Composer or the Server Manager.
2. Configure the printer IP address to use by way of the Local Settings.
3. Create a profile on an MFP.

Example 3: Prompting and sending an e-mail

Overview: OCR a document and send it to the user by way of e-mail.

Details: A company wants to make it easy for its employees to OCR hard copy documents without having to install OCR software on company computers. This script is used to prompt the user for their e-mail address and then ask if they want the document in RTF format (used by most word processors) or in PDF format. The script converts it and then sends it to the user by way of e-mail.

Script:

1. This script asks the end-user questions, so we need to set those up. The first question is a string and asks for an e-mail address. The second is a single-select list that asks for the format of the file.

```
stringprompt email
email.SetPromptText("Enter your email address")

stringprompt format
with format
   .SetPromptText("Select the file format")
   .AddPromptOption("RTF")
   .AddPromptOption("PDF")
endwith
```

2. The questions have been prepared. Now the prompt block is used to ask the questions. When the user selects this script, this is the part of the script that is executed before the scanning occurs. Since the answers to the questions do not affect later questions and since the answers are not verified, the questions can be asked in one step.

```
prompt
    step "prompts"
      ask(email)
      ask(format)
      laststep
endprompt
```

3. Once the questions have been asked, the document is scanned and submitted like all other non-prompting jobs. The script will resume execution at this point when the job is restarted.

4 Look at the format selected and convert the input document to PDF or RTF. The output is stored in the variable `ocr` so that it can be used for the e-mail action.

```
doc ocr
if (format == "PDF")
   with ImageToPDF
      .input=original.document
      .Language=LDD_LANG_ENGLISH
      .TextType=LDD_TEXTTYPE_NORMAL
      .Format=LDD_PDFFORMAT_IMAGEONTEXT
      .PictureResolution=300
      .PictureQuality=75
      .AllPages=TRUE
      .Go()
      ocr = .output
   endwith
else
    with ImageToRTF
      .input=original.document
      .Language=LDD_LANG_ENGLISH
      .TextType=LDD_TEXTTYPE_NORMAL
      .PaperSize=LDD_PAPERSIZE_LETTER
      .KeepColor=TRUE
      .KeepPictures=TRUE
      .AllPages=TRUE
      .Go()
      ocr = .output
   endwith
endif
```

5 The EmailSMTP action is now used to send the e-mail with the OCR output as the attachment. The e-mail address used was obtained from the end-user when the job was submitted. The IP address of the SMTP server was defined as a local setting; when the script is uploaded to the server, the Server Manager prompts for the value to use for this property.

```
with EmailSMTP
   .Server=local.emailserver
   .To=email
   .From=email
   .Subject="Your scanned document"
   .Message="Attached is your scanned document in the format you"_
            "requested."
   .CharacterSet=LDD_SMTPCHARSET_US
   .Attachments=ocr
   .Go()
endwith
```

To run this script:

1. Upload it to the server using the Workflow Composer or the Server Manager.
2. Configure the SMTP server address by way of the Local Settings.
3. In order to use this script on an MFP, it must have a touch screen. If one is not available, use Select'N'Send or Print'N'Send instead. If one is available, create a profile using a TIFF, resolution of 300 dpi, and a darkness of 4. This will provide the best OCR results.

    Note: The EmailSMTP action can be replaced with specific GroupWare-supported objects, such as Lotus Notes (EmailNotes), Microsoft Exchange (EmailExchange), and Novell GroupWise (GWMail). The major advantage of using these objects is that the script can take advantage of the address book and GroupWare-specific features; however, more setup requirements are involved in using those objects. In most cases, the GroupWare servers can be configured to accept SMTP mail. If the script has no need for the special features, the more general EmailSMTP action should suffice.

Example 4: Storing data in a database

Overview: Store the scanned document in a database.

Details: A Human Resources department is responsible for keeping track of all resumes. This script will be used to store the resume in a database. In addition, the resume will be converted to text which will also be stored in the database. By storing the text, the database can be searched for certain keywords and then a list of all resumes that match the search can be returned. Since the resume will be there as well, it can be viewed, printed, or routed to another location.

- Script 4A should be used by databases that use Binary Large Objects (BLOBs). BLOBs are used by databases that permit files to be stored in it. Examples of such databases are Oracle, Microsoft SQL Server, and IBM DB2.
- Script 4B should be used by databases such as Microsoft Access that do not support BLOBs. To accomplish the same task, the document must be saved to the local file system. The location of the file is then saved in the database so it can be retrieved later.

Script 4A:

1. Convert the submitted document to text; this text will be stored in the database and linked with this resume, thus making it searchable.

```
with ImageToText
   .input=original.document
   .Language=LDD_LANG_ENGLISH
   .TextType=LDD_TEXTTYPE_NORMAL
   .OneLine=TRUE
   .SeparateParagraphs=TRUE
   .PageBreaks=FALSE
   .AppendEOF=FALSE
   .AllPages=TRUE
   .Go()
endwith
```

2. Using the textfile object, read in the text produced by the ImageToText action and store it in a string.

```
textfile tf
string ocrText
tf.Open(ImageToText.output)
ocrText = tf.Read(tf.Size())
tf.Close()
```

3. Insert the resume along with the text into the database so it can be retrieved later. The properties for the database information are configured via Local Settings when the script is uploaded to the server.

```
with ODBCWrite
   .DataSource=local.DataSource
   .TableName=local.TableName
   .LogonName=local.LogonName
   .LogonPassword=local.LogonPassword
   .AssignValue("ScanDate","")
```

```
    .AssignValue("OCRText",ocrText)
    .AssignValue("Resume",original.document)
    .Go()
  endwith
```

To run this script:

1. On the server machine, setup the database to be used. The files, LDDOracleTable.sql and LDDSQLServerTable.sql, are provided to create the sample tables in Oracle and Microsoft SQL Server. These files are located in the scripts example directory installed with Workflow Composer. After setting up the database, configure the ODBC connection. For details on this, please refer to Appendix A: Configuring ODBC Data Source Names (DSN).
2. Upload it to the server using the Workflow Composer or the Server Manager.
3. Configure the ODBC connection information by way of the Local Settings.
4. Create a profile on an MFP.

Script 4B:

1. Convert the submitted document to text; this text will be stored in the database and linked with this resume, thus making it searchable.

```
with ImageToText
   .input=original.document
   .Language=LDD_LANG_ENGLISH
   .TextType=LDD_TEXTTYPE_NORMAL
   .OneLine=TRUE
   .SeparateParagraphs=TRUE
   .PageBreaks=FALSE
   .AppendEOF=FALSE
   .AllPages=TRUE
   .Go()
endwith
```

2. Using the textfile object, read in the text produced by the ImageToText action and store it in a string.

```
textfile tf
string ocrText
tf.Open(ImageToText.output)
ocrText = tf.Read(tf.Size())
tf.Close()
```

3. Setup the path for saving the file. The file will be saved to the local file system in a directory called `c:\lexmark`. The filename will be `resume_timestamp.tif`.

```
string filepath = "c:\lexmark\resume" + original.usertime + ".tif"
```

4. Since the document itself cannot be inserted into the database, save the document to the local file system. AppendTimestamp is set to FALSE since we manually added the timestamp in the previous step. CreateDirectory is set to TRUE so `c:\lexmark` will be created if it does not exist.

```
with SaveToFile
   .input=original.document
   .AppendTimestamp=FALSE
   .Overwrite=FALSE
   .Path=filepath
   .CreateDirectory=TRUE
   .Go()
endwith
```

5 Insert the filename along with the text into the database so it can be retrieved later. The properties for the database information are configured by way of the Local Settings when the script is uploaded to the server.

```
with ODBCWrite
   .DataSource=local.DataSource
   .TableName=local.TableName
   .LogonName=local.LogonName
   .LogonPassword=local.LogonPassword
   .AssignValue("ScanDate","")
   .AssignValue("OCRText",ocrText)
   .AssignValue("ResumeFile",filepath)
   .Go()
endwith
```

To run this script:

1 On the server machine, set up the database to be used. The Microsoft Access database, resumes.mdb, is provided to use for this example. This file is located in the scripts example directory installed with Workflow Composer. After setting up the database or using the provided one, configure the ODBC connection. For details on this, please refer to Appendix A.

2 Upload it to the server using the Workflow Composer or the Server Manager.

3 Configure the ODBC connection information by way of the Local Settings.

4 Create a profile on an MFP.

Example 5: Separating pages using barcodes

Overview: Use a barcode separator page to break jobs of varying page length into separate files.

Details: A worker needs to scan and process several sets of documents. Each set could be scanned separately, but it is easier (and more desirable) to scan them all at one time. This can be accomplished by using a barcode separator page to indicate the start of a new set. The use of the separator page allows each set to contain any number of pages. This script will look for a barcode on each page of the scanned document. When one is found, it will mark the end of the previous set which will be saved to a file and mark the beginning of a new set.

Script:

1. Declare all the variables used in this script.

```
doc separated
int i, startpage, lastpage
bool found
string fullpath, bcode
```

2. Separate the scanned document so that each page is in its own file. This is necessary so that each group of pages that form a set can later be combined into a single file.

```
with ImageSeparate
   .input=original.document
   .AllPages=TRUE
   .Go()
   separated =.output
endwith
```

3. Use the startpage variable to indicate the first page of the current set. Initialize it to the first page in the scanned set.

```
startpage = 1
```

4. Loop through all the pages that were submitted

```
loop i from 1 to separated.GetNumberFiles()
```

5. The found variable indicates a new group of pages is ready to be saved. Reset before processing each page.

```
found = false
```

6 Use the BarcodeRead action to determine whether a barcode is on this page.

```
with BarcodeRead
   .Reset()
   .Input=separated.GetFile(i)
   .Type=LDD_BARCODE_CODE128
   .Direction=LDD_BCDIR_HORIZ
   .Quality=LDD_BCQUALITY_GOOD
   .Fixed=FALSE
   .Validate=TRUE
   .Length=0
   .PageNumber=1
   .WholePage=TRUE
   .Go()
   bcode = .Results
endwith
```

7 For this example (and sample file provided), the barcode is the word "BREAK". Check to see if this has been found on this page.

```
if (bcode.Contains("BREAK")) then
```

8 A barcode has been found. If the startpage is equal to the current page (for example, startpage is 1 and the first page of the scanned document is a separator page), then no pages need to be saved. In this case the startpage of the next set is set to be the next page. Otherwise, the lastpage of the current set is set to the previous page. (Don't set equal to this page or the separator page itself will be saved, too.) Set the found flag to indicate we have a set of pages that needs to be saved.

```
if (startpage == i)
    startpage = i + 1
else
   lastpage = i - 1
   found = true
endif
```

9 No barcode has been found. Check to see if this is the lastpage of submitted document. If it is, this has to be the lastpage of the current set. Set lastpage to the current page and set the found flag to indicate we have a set of pages that needs to be saved.

```
else
   if (i == separated.GetNumberFiles()) then
     lastpage = i
      found = true
   endif
endif
```

10 If a new set of pages has been found, they need to be saved.

```
if (found == true) then
```

11 Use the ImageCombine action to combine the set of pages into one document.

```
with ImageCombine
   .input=separated.GetFileRange(startpage, lastpage)
   .Format=LDD_IMGFORMAT_TIFG4
   .ColorDepth=LDD_IMGDEPTH_BW
   .Go()
endwith
```

12 Determine the name of the file to use for saving the new document. The sets will be saved in the `c:\lexmark` directory. The file name will be `set_timestamp_page.tif`. Since all files saved in one job will have the same timestamp, the lastpage number of the set is added to get a unique filename.

```
fullpath = "c:\lexmark\set" + original.usertime + "_" + _
            lastpage.AsString() + ".tif"
```

13 Save the new document to disk. AppendTimestamp is set to FALSE since we added it manually in the previous step. CreateDirectory is set to TRUE so it will create the directory `c:\lexmark,` if it doesn't already exist.

```
with SaveToFile
   .input = ImageCombine.output
   .Path=fullpath
   .AppendTimestamp=FALSE
   .Overwrite=FALSE
   .Go()
endwith
```

14 The current set has now been saved. Set the startpage of the next set to be the next page in the document. Don't set it to the current page; otherwise, the next set will include the separator page as a part of the saved document.

```
startpage = i + 1;
   endif
endloop
```

To run this script:

1 Upload it to the server using the Workflow Composer or the Server Manager.

2 Create a profile on an MFP using TIFF, resolution of 300dpi, and a darkness of 3 or 4.

3 The file, fullset.tif, is provided in the scripts example directory installed with the Worfkflow Composer. This file can be printed and scanned or submitted by way of Select'N'Send. When used with this script, it will produce three separate files with 3, 2, and 4 pages respectively.

Example 6: Advanced Prompting I

Overview: Use the answers from prompt questions to determine the next question to ask

Details: This script allows a user to scan a document and send the original or an editable version of it by way of e-mail. This is an extension of script in Example 3: Prompting and sending an e-mail. The user will be asked for an e-mail address to send the document and then be asked if he wants to OCR the document. Based on that answer, the next question will ask for the appropriate format. While this is a simple script, it shows the power of the prompts and how the answer to one question can help drive the next question to ask.

Script:

1. This script asks the end-user questions, so we need to set those up here. The first question is a string and asks for an e-mail address. The second is a boolean prompt; the user will check it to OCR the document; otherwise, it will be sent as an image.

```
stringprompt email
email.SetPromptText("Email address to send document")

boolprompt ocr
ocr.SetPromptText("Click to OCR the document.")
```

2. If the user chooses OCR, this question is used. This is a single-select list that lets the format of the OCR document to be chosen.

```
stringprompt ocrformat
  with ocrformat
     .SetPromptText("Select the OCR format")
     .AddPromptOption("RTF")
     .AddPromptOption("PDF (Searchable)")
  endwith
```

3. If the user chooses not to OCR, this question is used. This is a single-select list that lets the format of the image document to be chosen.

```
stringprompt imgformat
  with imgformat
     .SetPromptText("Select the Image format")
     .AddPromptOption("TIFF")
     .AddPromptOption("PDF (Not Searchable)")
  endwith
```

4. The questions have been prepared. Now the prompt block is used to ask the questions. When the user selects this script, the prompting part of the script is executed before the scanning occurs. Since the answers to the questions are used to pick the next question, multiple steps are used.

```
prompt
```

5 This step contains two questions: what is the e-mail address and whether to OCR the document. Both questions are asked in one step since they don't affect each other. After the OCR question is asked, the return value is checked. If OCR is selected, the next question to execute is `oformat`; otherwise, the next step is `iformat`. Every step must have a nextstep or laststep directive.

```
step "email"
     ask(email)
     ask(ocr)

    if (ocr == TRUE)
       nextstep "oformat"
    else
       nextstep "iformat"
    endif
```

6 This step is chosen when the user chooses to OCR. It asks the user to choose the format of the OCR document. This is the last question in this path so the laststep directive is used.

```
step "oformat"
    ask(ocrformat)
    laststep
```

7 This step is chosen when the user chooses to not OCR. It will ask the user to choose the format of the image. This is the last question in this path so the laststep directive is used.

```
step "iformat"
   ask(imgformat)
   laststep
endprompt
```

8 Once the questions have been asked, the document is scanned and submitted like all other non-prompting jobs. The script will resume execution at this point when the job is restarted. Check the `ocr` prompt and see if the user wants to OCR the document.

```
doc attach
if (ocr == TRUE)
```

9 The user wants to OCR, check the format that is desired, and perform the conversion. Store the output of the conversion in the `attach` variable.

```
if (ocrformat == "RTF")
    with ImageToRTF
      .input=original.document
      .Language=LDD_LANG_ENGLISH
      .TextType=LDD_TEXTTYPE_NORMAL
      .PaperSize=LDD_PAPERSIZE_LETTER
      .KeepColor=TRUE
      .KeepPictures=TRUE
      .AllPages=TRUE
      .Go()
      attach =.output
    endwith
 else
    with ImageToPDF
      .input=original.document
      .Language=LDD_LANG_ENGLISH
      .TextType=LDD_TEXTTYPE_NORMAL
      .Format=LDD_PDFFORMAT_IMAGEONTEXT
      .PictureResolution=72
      .PictureQuality=50
      .AllPages=TRUE
      .Go()
      attach = .output
   endwith
endif
```

10 The user does not want to OCR the document; check the image format that is desired and perform the conversion if necessary. Since the document was scanned as a TIFF, no conversion is necessary if that format is chosen. Store the output of the conversion in the `attach` variable.

```
else
   if (imgformat == "TIFF")
     attach = original.document
   else
     with ConvertToDocument
       .input=original.document
       .Format=LDD_DOCFORMAT_PDF
       .AllPages=TRUE
       .PaperSize=LDD_PAPERSIZE_LETTER
       .Orientation=LDD_ORIENTATION_AUTO
       .Go()
        attach = .output
     endwith
   endif
endif
```

11 The EmailSMTP action is now used to send the e-mail with the selected output as the attachment. The e-mail address used was obtained from the end-user when the job was submitted. The IP address of the SMTP server was defined as a Local Setting; when the script is uploaded to the server, the Server Manager prompts for the value to use for this property.

```
with EmailSMTP
   .Server=local.emailserver
   .To=email
   .cc=""
   .bcc=""
   .From=email
   .ReplyTo=""
   .Sender=""
   .Subject="Scanned Document"
   .Message="Here's your scanned document(s)"
   .CharacterSet=LDD_SMTPCHARSET_US
   .Attachments=attach
   .Go()
endwith
```

To run this script:

1 Upload it to the server using the Workflow Composer or the Server Manager.

2 Configure the SMTP server address by way of the Local Settings.

3 In order to use this script on an MFP, it must have a touch screen. If one is not available, use Select'N'Send or Print'N'Send instead. If one is available, create a profile using a TIFF, resolution of 300dpi, and a darkness of 4. This will provide the best OCR results.

Example 7: Advanced Prompting II

Overview: Perform data validation as the end-user answers dynamic prompts.

Details: A worker needs to scan documents but wants to separate them based on the store that needs the document and then based on the department at the store. This script will prompt for the store number. If the store number is valid, a list of the departments for that store is displayed to the user. Otherwise, an error message is displayed and the user is prompted again for the store number. Once all the needed information is gathered, the document is scanned and saved to the appropriate location on the local file system.

Script:

1. This script asks the end-user questions, so we need to set those up first. The first prompt is an integer and asks for a store number. The second is an error message to display in case the store number entered is invalid. The third prompt is declared here but will be set up after the question has been answered; all prompts and any variables used in the prompt block must be declared before the prompt block is entered.

```
intprompt store
store.SetPromptText("Enter the store number:")

messageprompt errmsg
errmsg.SetPromptText("Invalid store number; please reenter.")

stringprompt dept
string storename, item
```

2. Some questions have been prepared and the others have been declared. Now the prompt block is used to ask the questions. When the user selects this script, this is the part of the script that is executed before the scanning occurs. Since the answers are being validated and used to help form other questions, multiple steps are used.

```
prompt
```

3 This step contains one question: what is the store number. After the store number is provided, it is looked up in the database to make sure it is a valid store number. If it is, the store name is retrieved from the database and the nextstep is set to be `dept`. If it is not valid, the nextstep is set to be `error`.

```
step "storenum"
     ask(store)
with ODBCRead
   .Reset()
   .DataSource=local.DataSource
   .TableName="stores"
   .LogonName=local.LogonName
   .LogonPassword=local.LogonPassword
   .DataColumns="StoreName"
   .WhereClause="StoreNumber=" + store.AsString()
   .Go()
   if (.GetNextRow())
   then
     storename = .GetString("StoreName")
     nextstep "dept"
   else
     nextstep "error"
   endif
   .Finished()
endwith
```

4 This step is used if an invalid store number is provided. It simply displays an error message and tells the user to re-enter. The user will acknowledge the message and then the nextstep is set to be `storenum`; in other words, the first question will be asked again.

```
step "error"
     ask(errmsg)
     nextstep "storenum"
```

5 This step is used once a valid store number has been given. The prompt text is set here so that it can include the name of the store that was selected. The valid departments for that store are then retrieved from the database.

```
step "dept"
     dept.SetPromptText("Select the Department at " + storename)
     with ODBCRead
       .Reset()
       .DataSource=local.DataSource
       .TableName="departments"
       .LogonName=local.LogonName
       .LogonPassword=local.LogonPassword
       .DataColumns="Department"
       .WhereClause="StoreNumber=" + store.AsString()
       .Go()
```

6 Each department is added to the single-select list as an option.

```
    repeat until (.GetNextRow() == FALSE)
         item = .GetString("Department")
         dept.AddPromptOption(item)
    endrepeat
    .Finished()
  endwith
```

7 The question has now been completely setup. The user will be able to select which department the scanned document is for. This is the final question so the laststep directive is used.

```
  ask(dept)
  laststep

endprompt
```

8 Once the questions have been asked, the document is scanned and submitted like all other non-prompting jobs. The script will resume execution at this point when the job is restarted. Build the full path and filename used for saving the file. The file is being saved in the `c:\lexmark` directory under a directory using the store number and then in a subdirectory for that department. The final name looks as follows: `c:\lexmark\storenum\department name\scan_timestamp.tif`.

```
string filepath = "c:\lexmark\" + store.AsString() + "\" + dept +
                    "\scan.tif"
```

9 The file is saved to disk using the SaveToFile action and the filepath specified in the previous step. AppendTimestamp is set to TRUE so a unique filename is guaranteed. CreateDirectory is set to TRUE so any of the directories that do not exist will be created.

```
with SaveToFile
   .input=original.document
   .Path=filepath
   .AppendTimestamp=TRUE
   .CreateDirectory=TRUE
   .Go()
endwith
```

To run this script:

1 On the server machine, set up the database to be used. The Microsoft Access database, stores.mdb, is provided to use for this example. This file is located in the scripts example directory installed with Workflow Composer. Configure the ODBC connection. For details on this, please refer to Appendix A.

2 Upload it to the server using the Workflow Composer or the Server Manager.

3 Configure the ODBC connection information by way of the Local Settings.

4 In order to use this script on an MFP, it must have a touch screen. If one is not available, use Select'N'Send or Print'N'Send instead. If one is available, create a profile.

5 The database has three store numbers: 10, 20, and 30. If you provide an invalid number, an error message will be displayed.

This script demonstrates the power of the prompts to validate data as the user enters it and to use the answers given to dynamically provide selections to the user. While this example saves the file to disk, it could be used to extend Example 2: Printing a document. In that case, the printer IP address that corresponds to the selected store and department would be retrieved from the database and used to send the print job.

3 Document Producer Scripts

Example 8: Merge data and print a form

Overview: Merge data onto a form and print it.

Details: Using Document Producer, a company is going to redirect their ASCII print stream from a printer loaded with preprinted forms to the Document Server. This script is used to merge the incoming data with a FormSet created with the Forms Composer and print it on blank paper.

Script:

1. Use the incoming data, original.dataset, with the MergeForm action to produce a PDF of the filled-in form.

```
with MergeForm
   .input=original.dataset
   .Go()
endwith
```

2. The print options such as duplex, paper source, and paper size are configured when the FormSet is created and passed by way of the printoptions property. Finishing options such as staple and hole punch cannot be set in the FormSet. To add additional settings or override the FormSet settings, create a printoptions object and specify the desired settings. In this case, hole punch is being turned on.

```
printoptions po = MergeForm.printoptions
po.SetJobOption(LDD_PRINTOPTION_HOLEPUNCH, LDD_HOLEPUNCH_ON)
```

3. Use the PrintForm action to print the PDF form. The IP address of the printer and type of printer are configured by way of the Local Settings. If the printer being used is equipped with an ImageQuick Option, the PDF form is sent directly to the printer. Otherwise, the PDF form is converted to PostScript and then sent to the printer. The `PrinterType` field must have it's local variable (`PrinterType`) set up as an enumeration variable. See the "Local Settings" topic in the Workflow Composer application for more information.

```
with PrintForm
   .input=MergeForm.output
   .IPAddress=local.PrinterAddress
   .PrinterType=local.PrinterType
   .PrintOptions=po
   .WaitForCompletion=FALSE
   .Go()
endwith
```

To run this script:

1. Upload it to the server using the Workflow Composer or the Server Manager.
2. Configure the printer information by way of the Local Settings.
3. Upload the debit memo example included with the Forms Composer. Create a named pipe called "test". For more information on this, see Appendix B.
4. Open a command prompt window on the machine the Document Server is installed. Using the sample data file included with the Forms Composer, emulate a print stream by typing the following:

```
copy /b debittext.txt \\.\pipe\test
```

The /b option performs a binary copy. The file debittext.txt contains a sample data file formatted as if being sent to a printer. The \\.\pipe\test copies the file to the named pipe created in Appendix B. If running from a different computer than the one the Document Server is installed, replace the "." (period) with the IP address of the machine where the server is installed.

Note: For information on configuring the Windows printer queues, refer to the online help in Forms Composer and the Server Manager.

Example 9: Merge data and fax a form

Overview: Merge data onto a form and fax it.

Details: Using Document Producer, a company is going to redirect their ASCII print stream from a printer loaded with preprinted forms to the Document Server. This script is used to merge the incoming data with a FormSet created with the Forms Composer and fax the resulting document.

Script:

1. Use the incoming data, original.dataset, with the MergeForm action to produce a PDF of the filled-in form.

```
with MergeForm
   .input=original.dataset
   .Go()
endwith
```

2. The output of the MergeForm action is a PDF. It needs to be converted into a TIFF image so that it can be converted to PostScript.

```
with ConvertImageFormat
   .input=MergeForm.output
   .Format=LDD_IMGFORMAT_TIFG4
   .ColorDepth=LDD_IMGDEPTH_BW
   .AllPages=TRUE
   .Go()
endwith
```

3. Convert the TIFF image to PostScript so it can be faxed.

```
with ConvertToDocument
   .input=ConvertImageFormat.output
   .Format=LDD_DOCFORMAT_PS
   .AllPages=TRUE
   .PaperSize=LDD_PAPERSIZE_LETTER
   .Orientation=LDD_ORIENTATION_AUTO
   .Go()
endwith
```

4. Get the fax phone number out of the submitted dataset. Any value of any variable in a FormSet can be retrieved and used in a script.

```
string faxnumber = original.dataset.GetDataItem("FAX_NUMBER")
```

5. Use the FaxByPrinter action to send the fax. The fax is sent through an MFP with fax capabilities. The IP address of the MFP is configured through a Local Setting.

```
with FaxByPrinter
   .input=ConvertToDocument.output
   .IPAddress=local.IpAddress
   .StationID="Fax Station ID"
   .FaxNumber=faxnumber
   .WaitForCompletion=FALSE
   .RedialMinutes=5
```

```
    .RedialTimes=3
    .Go()
  endwith
```

To run this script:

1. Upload it to the server using the Workflow Composer or the Server Manager.
2. Configure the printer information by way of the Local Settings.
3. Upload the fax cover example included with the Forms Composer. Create a named pipe called "test". For more information on this, see Appendix B.
4. Open a command prompt window on the machine the Document Server is installed. Using the sample data file included with the Forms Composer, emulate a print stream by typing the following:

```
copy /b faxcover.txt \\.\pipe\test
```

The /b option performs a binary copy. The file faxcover.txt contains a sample data file formatted as if being sent to a printer. The \\.\pipe\test copies the file to the named pipe created in Appendix B. If running from a different computer than the one the Document Server is installed, replace the "." (period) with the IP address of the machine where the server is installed.

Note: While the script only retrieves the fax number, it is also possible to get an email address, customer number, or any other variable in a FormSet. These values can then be used in the script. This script chooses to fax by way of an MFP; it is possible, to write the script so that it faxes through a fax server that supports SMTP e-mail. The form could also be archived for later use by any other actions supported by the server.

4 Document Portal Scripts

Example 10: Print a blank form

Overview: Print a blank form

Details: This is a basic script used by the Document Portal to print blank forms using a PostScript printer.

Script:

1. Create a dataset to pass to the MergeForm action. This dataset will only contain the name of the FormSet to be used; no data will be passed since a blank form is desired. The FormSet to be used is passed in by way of the formsetname property of the original object.

```
dataset merge_input
merge_input.SetFormset(original.formsetname)
```

2. Use the MergeForm action to generate the blank PDF form.

```
with MergeForm
   .input=merge_input
   .Go()
endwith
```

3. Use the PrintForm action to print the PDF generated by the MergeForm action. The IP address of the printer and the print options to be used are passed in by way of the printerip and printoptions properties of the original object. These settings are configured in the Document Portal Admin. In this case, the printer type is set to PostScript; if the printer being used has the ImageQuick Option, change the printer type to LDD_PRINTERTYPE_LEXIQ.

```
with PrintForm
   .input=MergeForm.output
   .IPAddress=original.printerip
   .PrinterType=LDD_PRINTERTYPE_LEXPS
   .PrintOptions=original.printoptions
   .WaitForCompletion=FALSE
   .Go()
endwith
```

To run this script:

1. Upload the script to the server using the Document Portal Admin. Do NOT use Workflow Composer or Server Manager to upload the script.
2. If no PDFs or FormSets have been uploaded to the server, upload one. Associate the script with an uploaded FormSet.
3. Using a touchscreen MFP, select the Forms icon and then select the form associated with this script.

Example 11: Prompt the user and print a blank form

Overview: Prompt the user and print the selected form.

Details: This script is useful for color printers. It prompts the user for whether the form should be printed in color or black and white. This allows one script to be used for both types of output.

Script:

1. This script asks the end-user a question, so we need to set it up here. The question is a single-select list that lets the user select Black and White or Color output. As with all scripts that ask questions, all prompts and any variables used in the prompt block must be declared before the prompt block is entered.

```
stringprompt color
with color
   .SetPromptText("Select the desired color output:")
   .AddPromptOption("Black & White")
   .AddPromptOption("Color")
endwith
```

2. The question has been prepared. Now the prompt block is used to ask the question. When the user selects this script, this is the part of the script that is executed before the job is sent to the server. Since there is only one question, only one step is necessary.

```
prompt
   step "prompts"
      ask(color)
      laststep
endprompt
```

3. Once the question has been asked, the job is submitted like all other non-prompting jobs. The script will resume execution at this point when the job is restarted.

4. Create a dataset to pass to the MergeForm action. This dataset will only contain the name of the FormSet to be used; no data will be passed since a blank form is desired. The FormSet to be used is passed in by way of the formsetname property of the original object.

```
dataset merge_input
merge_input.SetFormset(original.formsetname)
```

5. Use the MergeForm action to generate the blank PDF form.

```
with MergeForm
   .input=merge_input
   .Go()
endwith
```

6 The script will modify the print options sent along with the job. Based on the user selection, the printer will be set to print using the color cartridges or only the black cartridge.

```
printoptions opts = original.printoptions
if (color == "Color")
   opts.SetJobOption(LDD_PRINTOPTION_COLORMODEL, LDD_COLORMODEL_CMYK)
else
   opts.SetJobOption(LDD_PRINTOPTION_COLORMODEL, LDD_COLORMODEL_BLACK)
endif
```

7 Use the PrintForm action to print the PDF generated by the MergeForm action. The IP address of the printer is passed by way of the printerip property of the original object. The print options are the modified version created in the previous step. The original print settings are configured in the Document Portal Admin. In this case, the printer type is set to PostScript; if the printer being used has the ImageQuick Option, change the printer type to LDD_PRINTERTYPE_LEXIQ.

```
with PrintForm
   .input=MergeForm.output
   .IPAddress=original.printerip
   .PrinterType=LDD_PRINTERTYPE_LEXPS
   .PrintOptions=opts
   .WaitForCompletion=FALSE
   .Go()
endwith
```

To run this script:

1 Upload the script to the server using the Document Portal Admin. Do NOT use Workflow Composer or Server Manager to upload the script.

2 If no PDFs or FormSets have been uploaded to the server, upload one. Associate the script with an uploaded FormSet.

3 Using a touchscreen MFP connected to a color printer, select the Forms icon and then select the form associated with this script.

Appendix A: ODBC Setup

Follow these steps to create ODBC DSN connections used for Examples 4 and 7. The DSNs are created on the same machine that the Document Server is installed on.

1. On Windows NT: Go to **Settings** ▸ **Control Panel**.
   On Windows 2000/XP: Go to **Settings** ▸ **Control Panel** ▸ **Administrative Tools**
2. Select Data Sources (ODBC).
3. Select the System DSN tab.Click **Add**. This displays a list of ODBC drivers that have been installed on this machine. Choose "Microsoft Access Driver (*.mdb)" from the list and click **Finish**.
4. The ODBC Microsoft Access Setup dialog is displayed. For Example 4, enter the Data Source Name of "LDD1"; for Example 7, enter "LDD2". Click **Select** button to browse to the database to be used by this connection. For Example 4, select the "resumes.mdb" database. For Example 7, select "stores.mdb". Click **OK**.
5. Once both have been added, they will appear in the System DSN list.
6. Click **OK** to close the dialog. The connections have now been configured and can be used from within a script.

   When uploading scripts, Local Settings are available for Examples 4 and 7 that let the ODBC settings to be configured. In general, two pieces of data are necessary for all ODBC connections; while two are optional.

   - The DSN must be provided. The data source name is the one provided in step 4 above.
   - The table name must be provided. This is the name of the table that contains the data used in the script. If using the databases included with the examples, the default table names provided with the Local Settings can be used.
   - The username is optional. If no logon is required, this can be left blank. (By default, no logon is required for Access.)
   - The password is optional. If no logon is required, this can be left blank.

Appendix B: FormSet Setup

This appendix details the steps necessary to upload a FormSet and configure the named pipe to work with it. This information is used to run Examples 8 and 9.

In the Server Manager:

1. If not already done, upload the script to be used for this FormSet. For these examples, upload the scripts "example08.xdd" or "example09.xdd".
2. Click **Add a FormSet** from the Home Page. This starts the Add FormSet Wizard. The first step in the Wizard is to select the FormSet to upload. For Example 8, select "debitmemo.fdd"; for Example 9, select "faxcover.fdd". Both of these are included in the samples directory installed with the Forms Composer. Click **Next**.The next page of the Wizard lets you select what action is performed when data is received for this FormSet. There are three options:
   - the action will be specified by the pipe or in the datastream,
   - one of the predefined actions should be used, or
   - a custom script should be used.

   For our examples, we want to use a custom script. From the list select the appropriate script (example08 or example09).
3. Click **Next**. The last page of the Wizard provides a summary. If you have not previously created the named pipe to use for the examples, check the box that will start the Named Pipe Wizard.
4. Click **Finish**. If you need to create the named pipe, the Named Pipe Wizard will start; otherwise, you are finished and can proceed with running the script.
5. For the Named Pipe Wizard, the first step is to name the pipe. For our examples, we will create only one pipe. It can be used by both Examples 8 and 9. Call the pipe "test" and enter a description if desired. Click **Next**.
6. The next page of the Wizard configures the pipe to accept data for a specific FormSet or multiple FormSets. Since our pipe is being created for both Examples 8 and 9, select "This pipe will be used by multiple FormSets". Click **Next**.
7. The next page of the Wizard configures the action to take when data is received in the pipe. This is similar to the page of the Add FormSet Wizard which lets you configure what happens when data is received for the FormSet. Since we already configured what happens with the FormSet (it runs a custom script), select "Action will be specified in print stream". Click **Next**.

8 The next page of the Wizard lets you configure the type of data that is received by the pipe. In our case, we will be sending text files; so, the data type is always the same. Select "FormSets are always this type" and choose "a stream of plain text" from the list. Click **Next**.

9 The last page of the Wizard presents a summary of the selected options. Click **Finish**. You are now ready to run the script.

For more information on the various options when uploading FormSets and when creating named pipes, please refer to the online help available in the Server Manager.

740 West New Circle Road
Lexington, Kentucky 40550