# A Method for the Part Extraction of Handwritten Scores

Version 3 - 7 July 2012

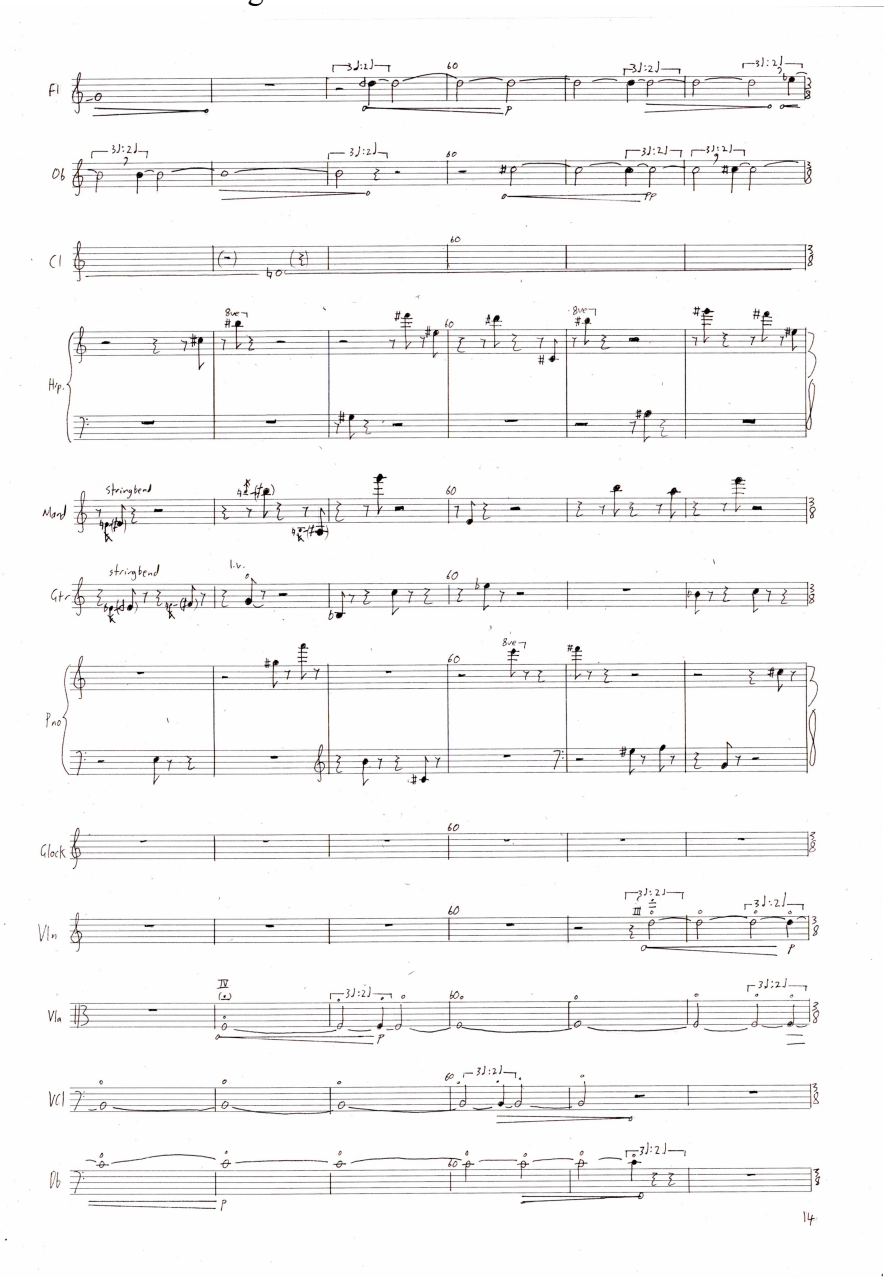
David Pocknee (david [at] davidpocknee.com)

This method is designed for the easy extraction of instrumental parts from handwritten scores.
The process works by taking the scanned images of pages of your scores, copying the area containing a particular instrument into new files, and then compiling it into an OpenOffice word file.

In order for this method of part extraction to work successfully, your score must:

- Have the same instrument layout on every page.
- Have a space between each instrument staff.
- The staffs between instruments should not be linked with barlines.

An ideal score layout would be something like this:



## *Programs used:*

This method uses two, free, open source programs, available for both mac and PC. These programs need to be downloaded and installed before any of the steps below can be followed:

- GIMP image editing program
  - Available from www.gimp.org
  - This method definitely works with version 2.6.8.
- OpenOffice
  - Available from www.openoffice.org
  - This method definitely works with version 3.0.0

This process utilises two scripts:

1. A script for GIMP which extracts the parts into separate files.
2. A script for OpenOffice which compiles them.

Before the first time you extract a score, a script for GIMP and a script for OpenOffice must be installed, but this only needs to be done once:

## *To Install The GIMP Script (this only needs to be done once):*

1. Paste the text in ***Appendix I*** into a text editor (e.g. Notepad for Windows or Textedit for Macs) and save it as a .scm file in the GIMP scripts directory (this can be found by opening GIMP and going to: Edit/Preferences and then clicking Folders/scripts).

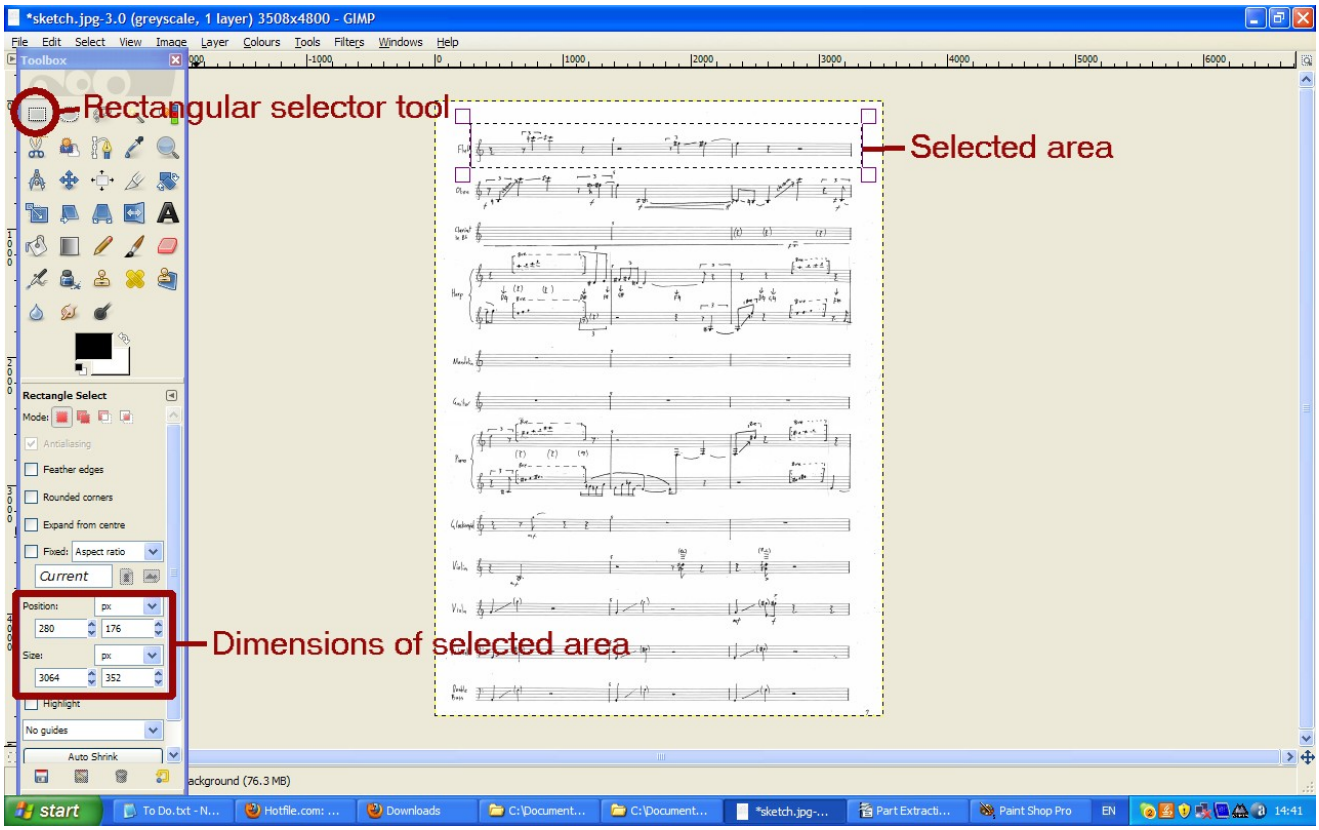
## *To Install the OpenOffice Script (this only needs to be done once)*

1. Open up OpenOffice's 'Writer' application, start a new document and go to: Tools/Macros/Organize Dialogs...
2. Click on the 'Modules' Tab.
3. Click on the 'My Macros' icon in the left hand pane.
4. Click on the 'New...' button.
5. Enter the name 'Extractor' into the box that appears.
6. Click on the 'Edit' button.
7. A text window will open. Delete any code that is there and paste in all the code from ***Appendix II***.
8. Go to File/Save.
9. Close the window.

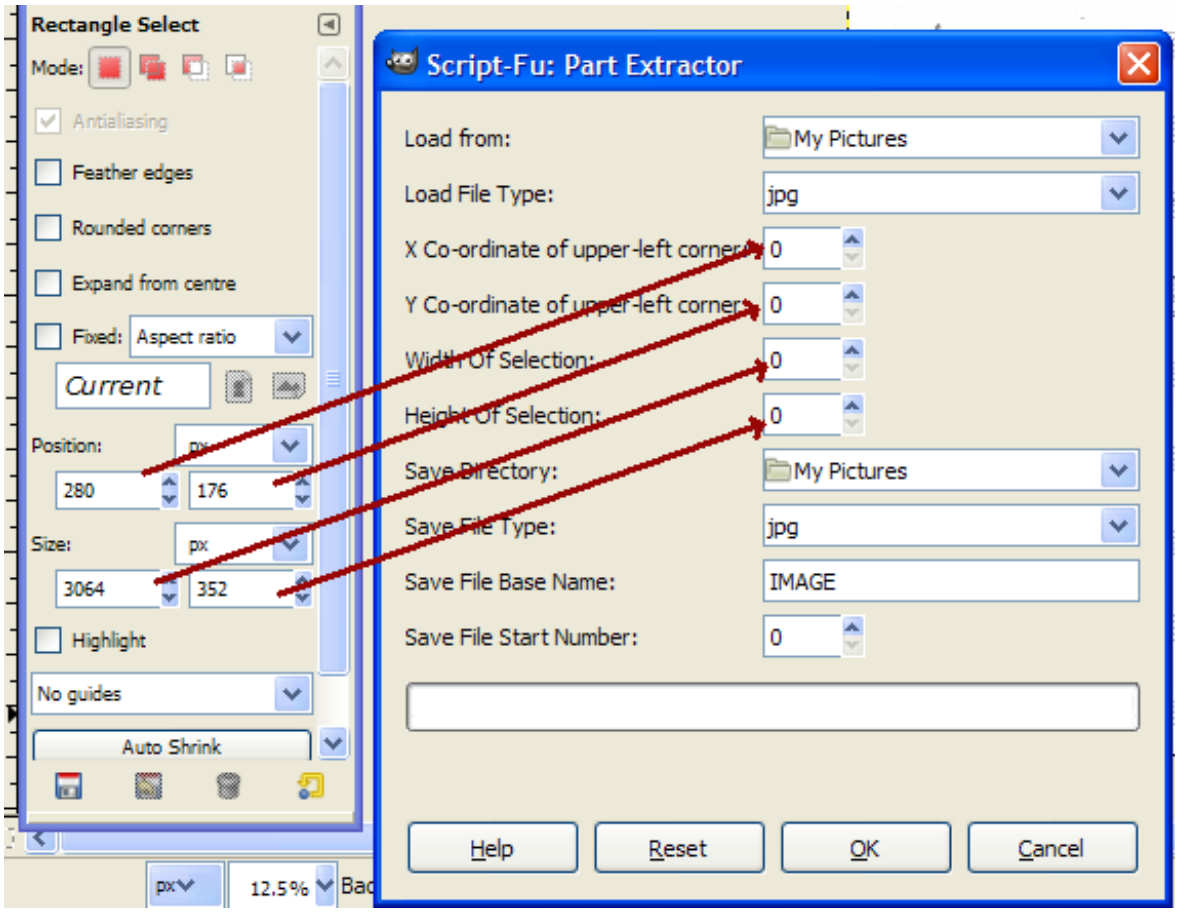
# Extracting Parts with GIMP:

Once the scripts for GIMP and OpenOffice have been installed, you can follow the steps below:

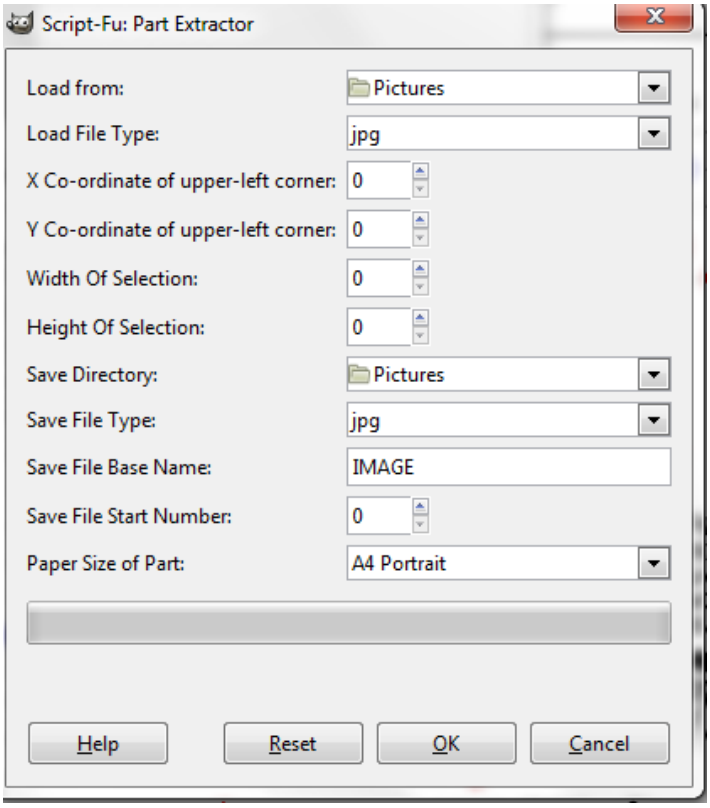
1. Scan in all pages of your score and save them as .jpg .png, .or .bmp.
2. Place all the files in the same folder and number the files as 'page01', 'page02' etc.
3. Load up GIMP, click File/Open and find a page of your score.
4. Use the Rectangular selector tool to highlight the area containing the staff of the instrument you want to extract.
5. Make a note of the dimensions of the rectangular area you selected. These are displayed in the box at the bottom left of the window (highlighted below).




6. Go to File/Part Extractor... and copy the values for the rectangular box into the boxes of the Part Extractor window, shown below. (If 'Part Extractor...' does not show up in the File menu, it means you have not installed the GIMP script correctly):




## Guide to the options:




- The directory your score pages are in.
- The file type of the score pages.

- The directory you want the parts saved in.
- The file type you want the parts saved as.

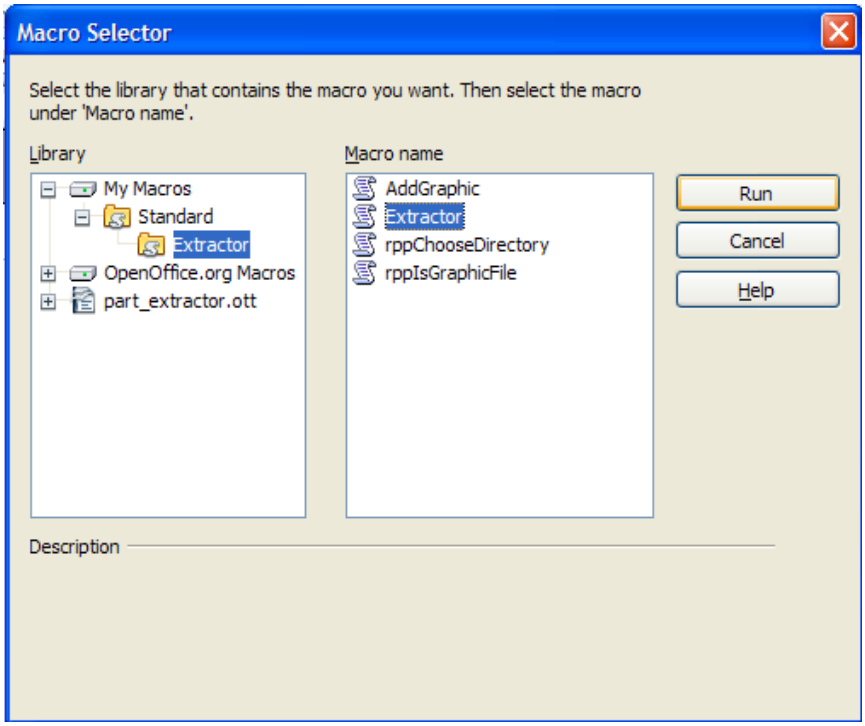
- The filenames you want the parts saved as. Here I have used 'Flute' as this is the flute part.
- The paper size and orientation of the paper the part will be placed on. The program will automatically scale the width of the staff to the width of the paper size.

**NOTE: You need to use a separate folder for every instrument you extract as the OpenOffice macro compiles all files in a selected folder into one part.**

8. Click OK and the program will extract the area of your rectangular selection from every file in your selected folder to the output folder you have selected.

# Compiling Parts In OpenOffice:

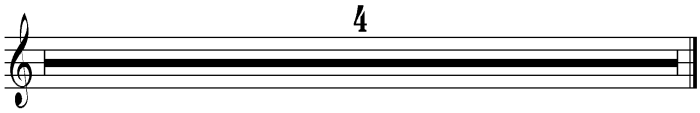
18. Go back to your new text document and go to:
    Tools/Macros/Run Macro...
19. Click on the + symbol next to the 'My Macros' icon in the left pane to expand the folder and then click on the + symbol next to the 'Standard' icon that appears beneath it.
20. Click on the 'Extractor' folder that appears below it.
21. Click on the 'Extractor' symbol in the right pane.
22. Click on the 'Run' button.




24. A dialog box will appear. Select the folder in which you extracted your files to with the previous script and click OK. Every file in the folder will be compiled into your document.

## Things to consider:

If a part needs a multirest symbol, such as the one below, then this will need to be drawn separately, scanned in, and added by hand:




Any comments, suggestions or improvements can be directed to david [at] davidpocknee.com

# Appendix I

Paste this into a text editor and save it as a .scm file in the GIMP script directory which can be found by opening GIMP and going to: Edit/Preferences and then clicking Folders/scripts

```
; Part Extractor Version 3
; by David Pocknee 5 July 2012
; Based on:
; DivideScannedImages.scm
; and
; by Rob Antonishen
; http://ffaat.pointclark.net

(define (script_fu_PartExtractor img inLayer inX inY inWidth inHeight inSaveFiles inDir
inSaveType inFileName inFileNumber inPaper)
  (let*
    (
      (width (car (gimp-image-width img)))
      (height (car (gimp-image-height img)))
      (newpath 0)
      (strokes 0)
      (tempVector 0)
      (tempImage 0)
      (tempLayer 0)
      (bounds 0)
      (count 0)
      (numextracted 0)
      (saveString "")
      (newFileName "")
      (tempdisplay 0)
      (buffname "dsibuff")
      (pathchar (if (equal? (substring gimp-dir 0 1) "/") "/" "\\"))
      (newdpi 0)
      (tempwidth 0)
      (tempheight 0)
      (brandwidth 0)
      (brandheight 0)
      (ratio 0)
      (paperchoose 0)
    )
(gimp-rect-select img inX inY inWidth inHeight REPLACE 0 0)


    (gimp-context-push)
    (gimp-image-undo-disable img)

    (if (= inSaveFiles TRUE)
      (set! saveString
      (cond
        (( equal? inSaveType 0 ) ".jpg" )
        (( equal? inSaveType 1 ) ".bmp" )
        (( equal? inSaveType 2 ) ".png" )
      )
))
(begin
          (set! buffname (car (gimp-edit-named-copy inLayer buffname)))
          (set! tempImage (car (gimp-edit-named-paste-as-new buffname)))
          (set! tempLayer (car (gimp-image-get-active-layer tempImage)))
          (gimp-image-undo-disable tempImage)
          (set! tempdisplay (car (gimp-display-new tempImage)))
      (gimp-layer-flatten tempLayer)
          (gimp-image-undo-enable tempImage)
(set! newdpi (car (gimp-image-get-resolution img)))
          (gimp-image-set-resolution tempImage newdpi newdpi)


(set! paperchoose
      (cond
        (( equal? inPaper 0 ) 7.5 )
        (( equal? inPaper 1 ) 11.5 )
      (( equal? inPaper 2 ) 15.5 )
      (( equal? inPaper 3 ) 9.5 )
      (( equal? inPaper 4 ) 13.5)      )
)

      (set! tempwidth (car (gimp-image-width tempImage)))
      (set! tempheight (car (gimp-image-height tempImage)))
```

```
      (set! ratio (/ paperchoose (/ tempwidth newdpi)))
      (set! brandwidth      (* ratio tempwidth))
      (set! brandheight     (* ratio tempheight))
      (gimp-image-scale tempImage brandwidth brandheight)

          ;save file
          (if (= inSaveFiles TRUE)
          (begin
            (set! newFileName (string-append inDir pathchar inFileName
                                             (substring "00000" (string-length (number->string
(+ inFileNumber numextracted))))
                                               (number->string (+ inFileNumber numextracted))
saveString))
                    (gimp-file-save  RUN-NONINTERACTIVE  tempImage  tempLayer  newFileName
newFileName)
            (gimp-display-delete tempdisplay)

          (set! numextracted (+ numextracted 1))

      (set! count (+ count 1))


    ;input drawable name should be set to 1919191919 if in batch
        (if  (and  (>  numextracted  0)  (equal?  (car  (gimp-drawable-get-name  inLayer))
"1919191919"))
      (gimp-drawable-set-name inLayer (number->string (+ 1919191919 numextracted))))

    ;delete temp path
    (gimp-selection-none img)

    ;done
    (gimp-image-undo-enable img)
    (gimp-progress-end)
    (gimp-displays-flush)
    (gimp-context-pop)
  )
)


(define  (script_fu_BatchPartExtractor  inSourceDir  inLoadType  inX  inY  inWidth  inHeight
inDestDir inSaveType inFileName inFileNumber inPaper)
(let*
    (
      (varLoadStr "")
      (varFileList 0)
      (varCounter inFileNumber)
      (pathchar (if (equal? (substring gimp-dir 0 1) "/") "/" "\\"))
    )

    (define split
      (lambda (ls)
        (letrec ((split-h (lambda (ls ls1 ls2)
                            (cond
                              ((or (null? ls) (null? (cdr ls)))
                               (cons (reverse ls2) ls1))
                              (else (split-h (cddr ls)
                                             (cdr ls1) (cons (car ls1) ls2)))))))
          (split-h ls ls '()))))

    (define merge
      (lambda (pred ls1 ls2)
        (cond
          ((null? ls1) ls2)
          ((null? ls2) ls1)
          ((pred (car ls1) (car ls2))
           (cons (car ls1) (merge pred (cdr ls1) ls2)))
          (else (cons (car ls2) (merge pred ls1 (cdr ls2)))))))

    ;pred is the comparison, i.e. <= for an ascending numeric list, or
    ;string<=? for a case sensitive alphabetical sort,
    ;string-ci<=? for a case insensitive alphabetical sort,
    (define merge-sort
      (lambda (pred ls)
        (cond
          ((null? ls) ls)
          ((null? (cdr ls)) ls)
          (else (let ((splits (split ls)))
                  (merge pred
                    (merge-sort pred (car splits))
```

```
                    (merge-sort pred (cdr splits))))))))

    ;begin here
    (set! varLoadStr
    (cond
    (( equal? inLoadType 0 ) ".[jJ][pP][gG]" )
    (( equal? inLoadType 1 ) ".[bB][mM][pP]" )
    (( equal? inLoadType 2 ) ".[pP][nN][gG]" )
    ))

     (set! varFileList (merge-sort string<=? (cadr (file-glob (string-append inSourceDir
pathchar "*" varLoadStr)  1))))
    (while (not (null? varFileList))
      (let* ((filename (car varFileList))
             (image (car (gimp-file-load RUN-NONINTERACTIVE filename filename)))
             (drawable (car (gimp-image-get-active-layer image))))

        ;flag for batch mode
        (gimp-drawable-set-name drawable "1919191919")
        (gimp-progress-set-text (string-append "Working on ->" filename))

         (script_fu_PartExtractor image drawable inX inY inWidth inHeight TRUE inDestDir
inSaveType inFileName varCounter inPaper)

        ;increment by number extracted.
          (set! varCounter (+ varCounter (- (string->number (car (gimp-drawable-get-name
drawable))) 1919191919)))
        (gimp-image-delete image)
      )
      (set! varFileList (cdr varFileList))
    )
  )
)

(script-fu-register "script_fu_BatchPartExtractor"
                    "<Toolbox>/File/Part Extractor..."
                    "Extracts parts from hand-written music scores"
                    "Rob Antonishen modified by David Pocknee"
                    "Rob Antonishen, modified by David Pocknee"
                    "April 2010"
                    ""
                    SF-DIRNAME    "Load from" ""
                    SF-OPTION     "Load File Type" (list "jpg" "bmp" "png")
                     SF-ADJUSTMENT "X Co-ordinate of upper-left corner"  (list 0 0 9000 1
100 0 SF-SPINNER)
                     SF-ADJUSTMENT "Y Co-ordinate of upper-left corner"  (list 0 0 9000 1
100 0 SF-SPINNER)
                    SF-ADJUSTMENT "Width Of Selection"           (list  0  0  9000  1  100  0
SF-SPINNER)
                    SF-ADJUSTMENT "Height Of Selection"                (list  0  0  9000  1
100 0 SF-SPINNER)
                    SF-DIRNAME    "Save Directory"                       ""
                    SF-OPTION     "Save File Type"                       (list "jpg" "bmp"
"png")
                    SF-STRING     "Save File Base Name"                  "IMAGE"
                     SF-ADJUSTMENT "Save File Start Number"               (list 0 0 9000 1
100 0 SF-SPINNER)
                  SF-OPTION      "Paper Size of Part" (list "A4 Portrait" "A4 Landscape"
"A3 Landscape" "B4 Portrait" "B4 Landscape")
)
```

# Appendix II

```
'Part Extractor V2 09/10/10
'by David Pocknee
'Based on PhotoAlbum code by Russel Phillips and uses elements of
'code derived from code in Andrew Pitonyak's excellent book,
'OpenOffice.org Macros Explained (http://www.pitonyak.org/book/)
'The author, Russell Phillips, can be contacted at avantman42@users.sourceforge.net
'It also uses elements from codes by Andrew Pitonyak from
'http://documentation.openoffice.org/HOW_TO/various_topics/AndrewMacro.odt


Sub Extractor

      dim oSlides as object, oAlbum as object
      dim sDir as string, sFile as string
rem define variables
dim document   as object
dim dispatcher as object

rem ----------------------------------------------------------------------
rem get access to the document
document   = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

      'Get directory with graphics to be imported
      sDir = rppChooseDirectory (False, True)
      if sDir = "" then
            'User cancelled directory dialogue box
            exit sub
      end if

'Get first file in directory
      sFile = dir(sDir)
      'Go through all files in directory
      While Not (sFile = "")
            if rppIsGraphicFile (sFile) then

oDoc = ThisComponent
  ' Create a table and attach/insert it at the current cursor position
  vViewCursor = oDoc.getCurrentController().getViewCursor()
  oTable = oDoc.createInstance("com.sun.star.text.TextTable")
  oTable.initialize(1, 1)

  REM Now insert the text table at the end of the document.
  oDoc.getText.insertTextContent( vViewCursor, oTable, False )
  Dim x 'represents each BorderLine
  Dim v 'represents the TableBorder Object as a whole
  v = oTable.TableBorder
  x = v.TopLine        : x.OuterLineWidth = 0 : v.TopLine = x
  x = v.LeftLine       : x.OuterLineWidth = 0 : v.LeftLine = x
  x = v.RightLine      : x.OuterLineWidth = 0 : v.RightLine = x
  x = v.TopLine        : x.OuterLineWidth = 0 : v.TopLine = x
  x = v.VerticalLine   : x.OuterLineWidth = 0 : v.VerticalLine = x
  x = v.HorizontalLine : x.OuterLineWidth = 0 : v.HorizontalLine = x
  x = v.BottomLine     : x.OuterLineWidth = 0 : v.BottomLine = x

  oTable.TableBorder = v


  oImage = oDoc.createInstance("com.sun.star.text.GraphicObject")
  With oImage
     .GraphicURL = ConvertToURL(sDir+sFile)
     .AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
      End With

  ' create a text cursor in the left cell
  oCursor = oTable.getCellByPosition(0,0).createTextCursor()


oTable.getCellByPosition(0,0).insertTextContent(oCursor,oImage,False)

            end if
            'Get next file
            sFile = dir
      Wend
```

```
End Sub

Function rppIsGraphicFile (FileName as string) as Boolean
'Function to determine if file is a graphic file
'Returns True if file is graphics file, False if not
	dim asGraphicExt
	dim sLFile as string
	dim iExt as integer

	'asGraphicExt is array of graphics file extensions
	asGraphicExt = Array (".bmp", ".dxf", ".emf", ".eps", ".gif", ".jpg", ".jpeg",
".met", _
		".pbm", ".pcd", ".pct", ".pcx", ".pgm", ".png", ".ppm", ".psd", ".ras",
".sgf", _
		".sgv", ".svm", ".tga", ".tif", ".tiff", ".wmf", ".xbm", ".xpm")

	'Initialise return value to False
	rppIsGraphicFile = False

	'Convert FileName to lower-case, to make comparison simpler
	sLFile = LCase (FileName)

	'Loop through asGraphicExt
	for iExt = LBound (asGraphicExt) to UBound (asGraphicExt)
		'Check file extension against element iExt of asGraphicExt
		If Right (sLFile, Len (asGraphicExt (iExt))) = asGraphicExt (iExt) then
			'Match: file is graphic file. Set return value to True
			rppIsGraphicFile = True
		end if
	next iExt

End Function

Function rppChooseDirectory (asURL as Boolean, incSeperator as Boolean) as string
'Function to allow user to choose a directory via a dialogue box
'Returns path to directory, or empty string if user cancelled
'If asURL is true, returns as a URL
'If incSeperator is true, includes seperator (\ or /) at end of string

	dim dlgDirectory as Object
	dim sReturn as string

	'Set up FolderPicker object & initialise return value
	dlgDirectory = CreateUnoService ("com.sun.star.ui.dialogs.FolderPicker")
	sReturn = ""

	'Display dialogue box
	if dlgDirectory.Execute () = 1 then
		'User did not cancel dialogue box. Get path to directory
		sReturn = CStr (dlgDirectory.GetDirectory ())
		if asURL and incSeperator then
			'Append / at end if not already present
			if not (Right (sReturn, 1) = "/") then
				sReturn = sReturn & "/"
			end if
		elseif not (asURL) then
			'Convert sReturn from URL format
			sReturn = ConvertFromURL (sReturn)
			'Append seperator if not already present
			if incSeperator then
				if not (Right (sReturn, 1) = GetPathSeparator ()) then
					sReturn = sReturn & GetPathSeparator ()
				end if
			end if
		end if
	end if

	'Set function return value
	rppChooseDirectory = sReturn

End Function
```