

wxPython の画像とビットマップは通常、ディスクからファイルをロードすることによって作成されます。メモリ内の画像を操作することも、特定の種類のアプリケーションでは一般的なタスクです。このレシピセットでは、wxPython アプリケーションで使用するための画像のメモリ内操作 (特に生成) について説明します。

目次：

目次

1. PIL イメージ、wx.Bitmap、wx.Image 間の変換
 1. 画像ファイルの読み込み
 2. 2進値のアルファ透明度を真の (256 値) アルファ透明度に変換する
2. wx.Image、wx.Bitmap、wx.Cursor、wx.Icon と文字列データ間の変換
3. PIL (Python イメージング ライブラリ)
4. ナンビ
 1. スライス
 2. 例
5. スクリーンキャプチャ
6. 柔軟なスクリーンキャプチャアプリ
7. ビットマップにテキストを書き込む
8. PythonMagick (16 ビット イメージング ライブラリ)
2. ありがとう

PIL イメージ、wx.Bitmap、wx.Image 間の変換

私が最も頻繁に使用する Image クラスは、wx.Image、wx.Bitmap、および PIL Image です。ここでは、これらのクラス間の変換に使用する関数をすべての組み合わせで示します。wx.Windows API での使用法と名前付けはわかりにくいため、関連する名前を持つ 6 つの個別の関数としてこれらを分類しています。また、API は比較的頻繁に変更されます。このセクションの古いテキストは時代遅れだと思いますが、確認する時間がありませんでした。-- Robb Shecter

行番号表示/非切替表示

```
1 # wxPython 2.3.4.2 および PIL 1.1.3 でテスト済み
  2 wx
  3 をインポート Image # PIL モジュールをインポートします。PIL
ライブラリを使用する場合のみ
  4
```

```

5  WxBitmapToPILImage ( myBitmap ) を定義します:
6      WxImageToPILImage ( WxBitmapToWxImage ( myBitmap ) )
7  を返す
8  WxBitmapToWxImage ( myBitmap ) の定義:
9      wx.ImageFromBitmap ( myBitmap )
10 を返す
11 #-----
12
13  PILImageToWxBitmap ( myPILImage ) を定義します:
14      WxImageToWxBitmap ( PILImageToWxImage ( myPILImage ) )
15 を返す
16  PILImageToWxImage ( myPILImage ) を定義します:
17      myWxImage = wx.EmptyImage ( myPILImage.size [ 0 ] ,
myPILImage.size [ 1 ] )
18      myWxImage.SetData ( myPILImage.convert ( ' RGB ' ) . tostring (
)
) 19      myWxImage
20 を返す
21 # または、アルファチャンネルもコピーしたい場合 (wxPython 2.5以降で利用可能)
22 # このルーチンを使用するには、ソース PIL イメージにアルファは必要ありません
23 # ただし、アルファ付きの wx.Image を取得するには、アルファ付きの PIL 画像が
必要です
24
25  PILImageToWxImage を定義します ( myPILImage、copyAlpha = True ) :
26
27      hasAlpha = myPILImage.mode [ - 1
] == ' A ' 28      if copyAlpha and hasAlpha : # アルファレイヤーのコ
ピーがあることを確認します
29
30          myWxImage = wx.EmptyImage ( * myPILImage.size )
31          myPILImageCopyRGBA = myPILImage.copy ( )
32          myPILImageCopyRGB = myPILImageCopyRGBA.convert ( ' RGB ' )
#RGBA -- > RGB
33          myPILImageRgbData = myPILImageCopyRGB.tostring ( )
34          myWxImage.SetData ( myPILImageRgbData )
35  です。 myWxImage.SetAlphaData ( myPILImageCopyRGBA.tostring (
) [ 3 :: 4 ] ) #レイヤーを作成し、アルファ値を挿入します
36
37      else : # 結果の画像にはアルファは含まれません
38
39          myWxImage = wx.EmptyImage ( * myPILImage.size )
40          myPILImageCopy = myPILImage.copy ( )
41          myPILImageCopyRGB = myPILImageCopy.convert ( ' RGB ' )
# PIL イメージからアルファをすべて破棄します
42          myPILImageRgbData = myPILImageCopyRGB.tostring ( )
43          myWxImage.SetData ( myPILImageRgbData )
44  です。
45      myWxImage
46 を返す
47 #-----
48
49 def imageToPIL ( myWxImage ):
50     myPILImage = 画像 . new ( ' RGB ' , ( myWxImage . GetWidth ( ) ,
myWxImage . GetHeight ( ) ) )
51     myPILImage.fromstring ( myWxImage.GetData ( )
) 52     myPILImage
53 を返す
54  WxImageToWxBitmap ( myWxImage ) を定義します:
55      myWxImage を返します。 ConvertToBitmap ( )

```

でも、待ってください...まだあります! wx.Image または wx.Bitmap には2種類の透明度があります: 1) すべてのピクセルは完全に透明か完全に不透明です。このバイナリ透明度は透明マスクです。 2) 各ピクセルは完全に透明 (アルファ値 = 0) から完全に

不透明 (値 = 255) までの可変量の透明度を持つことができます。各ピクセルの透明度は 0 から 255 までの値で定義されます。これはアルファ透明度です。たとえば、GIF ファイルには透明マスクがあることも、透明度がまったくないこともあります。ただし、PNG ファイルには透明マスク、アルファ透明度があることも、どちらもありません。比較すると、JPG ファイルにはいかなる種類の透明度も持ってません。

画像ファイルの読み込み

画像ファイルから `wx.Bitmap` を作成する:

```
wxBmap = wx.EmptyBitmap( 1, 1 ) # ビットマップコンテナオブジェクトを作成します。サイズ値はダミーです。  
wxBmap.LoadFile( filename, wx.BITMAP_TYPE_ANY ) # ファイルイメージで読み込みます。
```

ビットマップにマスクまたはアルファ透明度があるかどうかを判断します。

```
bmapHasMask = wxBmap.GetMask() # "GetMask()" であって、"HasMask()" ではありません!  
bmapHasAlpha = wxBmap.HasAlpha()
```

画像ファイルから `wx.Image` を作成する方法は、`wx.Bitmap` の場合とまったく同じです。

```
wxImg = wx.EmptyBitmap( 1, 1 ) # ビットマップコンテナを作成する  
wxImg.LoadFile( ファイル名, wx.BITMAP_TYPE_ANY )
```

ビットマップにマスクまたはアルファ透明度があるかどうかを判断するには:

```
imgHasMask = wxImg.HasMask() # 「HasMask()」であって、「GetMask()」ではありません! うわあ...  
imgHasAlpha = wxImg.HasAlpha()
```

画像ファイルをPILイメージに読み込む:

```
画像をインポート  
pilImage = Image.open( ファイル名 )
```

`PIL open()` 関数は、ファイルのイメージタイプを自動的に判別します。`wxPython` では、必要なくても「`wx.BITMAP_TYPE_ANY`」パラメータを削除することはできません。`PIL` イメージの透明度はアルファタイプのみであることに注意してください。`PIL` イメージに透明度があるかどうかを確認するには、次のようにします。

```
pilImageHasAlpha = pilImg.mode[ -1 ] == 'A'
```

}PIL はマスク付きの PNG ファイルを読み込むと、マスクをバイナリ値のアルファ透明度に自動的に変換します。その値は、完全に透明な場合は 0、完全に不透明な場合は 255 になります。通常、これは望ましい動作ではありません。



2 進値のアルファ透明度を真の (256 値) アルファ透明度に変換する

PIL イメージの透明度は、真の 256 値のアルファ タイプのみになりますが、`wx.Bitmap` と `wx.Image` はどちらの種類でもかまいません。`wx.Image` の透明度マスクを真のアルファ透明度に変換するには、次のようにします。

```
imgHasMask = wxImg.HasMask()  
wxImg.HasMask() の場合:  
    wxImg.InitAlpha()
```

さまざまな画像タイプの変換をすべて処理し、同時に透明度を適切にコピーするには、次の **ImgConv.py** モジュールを使用できます。フラグパラメータをデフォルト値で呼び出します。つまり、呼び出しパラメータ リストにフラグパラメータを含めないでください。透明マスクまたはアルファ透明度を持つ `wx.Bitmap` または `wx.Image` から開始する場合、これらのルーチンはその透明度をアルファ付きの PIL 画像に引き継ぎます。透明マスクは常にアルファ透明度に変換されます。PIL 画像には透明度としてアルファのみを含めることができ、マスクを含めることはできません。したがって、このパッケージが透明度を持つ PIL 画像を変換すると、結果の `wx.Image` または `wx.Bitmap` にはマスクではなくアルファ透明度が含まれます。

フラグパラメータ **addAlphaLayer** および **delAlphaLayer** を使用すると、オプションで次の操作を実行できます。1) ソース イメージにアルファ透明度がない場合にアルファ透明度のあるイメージを作成する (`addAlphaLayer=True`)、または逆に、2) ソース イメージにアルファがある場合に新しいイメージでアルファが自動的に作成されないようにする (`delAlphaLayer=True`)。

 **ImgConv.py**  **Win32IconImagePlugin.py**

このモジュールには 6 つの変換関数が含まれています。デフォルトでは、入力画像に存在する透明度が自動的に保持されます。これは通常、アプリで画像を表示するときに必要なことです。

```
WxBitmapFromPilImage()  
WxImageFromPilImage()  
  
WxBitmapFromWxImage()  
PilImageFromWxImage()  
  
WxImageFromWxBitmap()  
PilImageFromWxBitmap()
```

BitmapManip.py モジュールには、高度な画像操作に役立つマスクを抽出および結合するためのユーティリティ関数があります。完全に不透明で動的に変化するテキスト

トが重ねられた半透明の背景画像を持つ、自由形式の非長方形の `Wx.Frames` を作成できます。

 ビットマップマニピュレータ

`wx.Image`、`wx.Bitmap`、`wx.Cursor`、`wx.Icon` と文字列データ間の変換

`wx.Image` から `wx.Bitmap` へ:

```
myWxBmap = myWxImg.ConvertToBitmap()
```

または :

```
myWxBmap = wxBitmapFromImage( myWxImg )
```

`wxImage` から文字列データへ:

```
myStrData = myWxImg.GetData()
```

これは、長さ (幅 * 高さ * 3) のバイナリ データ Python 文字列を返します。

文字列データを `wx.Image` に変換します。

```
wxImg = wx.Image()  
wxImg.SetData( myStrData )
```

バイナリ データの Python 文字列を `wx.Bitmap` に変換します。 `wx.Image` を経由して `wx.Bitmap` を取得します。

`DATA` を `wx.Icon` に: 可能なはずですが、オーバーロードされたコンストラクター名が見つかりません。

`wx.Icon` から `wxBitmap` へ:

```
myWxBitmap = wx.EmptyBitmap( myIcon.GetWidth(), myIcon.GetHeight() )  
myWxBitmap.CopyFromIcon( myIcon )
```

いくつか `wx` 関数が欠落しているようです。C ユーザー向けにこれらが低レベルの抽象化で利用できるかどうかはわかりませんが、いずれにしても、いつか登場することを期待しているものを次に示します。

`wxBitmap` から文字列データへ:

`wx.Image` に変換してから `GetData()` を呼び出す必要があります。

バイナリデータの Python 文字列または `wx.Image` を `wxCursor` に変換します。

オーバーライドされたコンストラクターが利用可能になるはずですが、まだ利用できません。

`wx.Icon` をバイナリ データの Python 文字列または `wx.Image` に変換します。

アイコンの編集を可能にするには、これが必要になります。

`wxCursor` をバイナリデータの Python 文字列または `wxImage` に変換します。

繰り返しになりますが、カーソルの編集を可能にするためです。現在のホットスポットを取得するメソッドも必要です。 `wxImage` から `wxIcon` へ -- 上記の DATA から `wxIcon` を参照してください。どちらでも問題ありませんが、完全なセットがあれば便利です。

PIL (Python イメージング ライブラリ)

`wxPython` に組み込まれているものよりも高度な画像処理が必要な場合は、`wxPython` で PIL を使用できます。バイナリ文字列画像データは、以下の例に示すように、`.convert()`および`.tostring()`を含む PIL Image オブジェクトを使用して作成できます。適切なサイズの空の `wxImage` オブジェクトを作成するために、PIL Image の`.size`属性を使用していることに注意してください。

行番号表示/非切替表示

```

1 import PIL.Image          # 基本的なPILライブラリモジュール
2 import Win32IconImagePlugin # PIL が .ICO ファイル内の透明度を
 適切に読み取るために必要です
3
4 wx
5をインポート
6 #-----
7
8 def BitmapFromFile (画像ファイル名) :
9     """ 次の PIL 画像変換は、まず wx.Image に渡される必要があります
10
11 wx.Imageは、 11   かどうかに関係なく、常に最終的にwx.Bitmapに変換されます。
 処理すべき画像の透明度情報が存在する
12   これは、wxBitmap だけが直接表示できるからです。wxImage は直接表示でき
 ません
13
14   PILが正しく読み取れるようにするには、モジュールWin32IconImagePlugin.py
 をインポートする必要があります
15   マスク付きパレット画像（バイナリ値の透明度）。すべての .ICO ファイルと一部
 の .PNG ファイル
16   マスクの透明度を持つパレット画像データがある場合があります。参照：
17
18   Win32IconImagePlugin - Microsoft .ico ファイルを処理するための代替
  PIL プラグイン
19
20   http://code.google.com/p/casadebender/wiki/Win32IconImagePlugin
  「
```

```

21     pilImg = PIL.Image.open ( imgFilename ) 22

23     # 以下は「wxImg = wx.EmptyImage( pilImg.size[0], pilImg.size[1]
)」と同等です
    24     wxImg = wx.EmptyImage ( * pilImg.size )      # 常に透明面なしで作
成されます
    25
26     # 画像ファイルに固有の透明度があるかどうかを判断します
    27     pilMode = pilImg . mode      # 通常は「RGB」または「RGBA」ですが、
他の場合もあります
    28     pilHasAlpha = pilImg。モード[- 1 ] == ' A '
29     pilHasAlphaの場合:
30
31     # まずデータ文字列から RGB データだけを抽出し、それを wx.Image に挿
入します
    32     pilRgbStr = pilImg.convert ( ' RGB ' ) . tostring ( )
33     wxImg.SetData ( pilRgbStr )
34です。
35     # アルファ付きの wx.Image に変換するには、pilImg モードを "RGBA"
にする必要があります
    36     # したがって、元のファイル画像に透明度情報がない場合でも、アルファ
レイヤーを追加します
    37     # ファイル画像に透明度がない場合、結果のwx.Image (そして最終的には
wx.Bitmap)
38     # ファイル画像と同様に 100% 不透明になります
    39     pilImgStr = pilImg.convert ( ' RGBA ' ) . tostring ( )
# 元の画像モードがすでに「RGBA」の場合は無害です
    40
41     # 次に、アルファデータだけを抽出して挿入します
    42     pilAlphaStr = pilImgStr [ 3 :: 4 ]      # バイトインデックス3
から開始し、ストライド (バイトスキップ) は4です
    43     wxImg . SetAlphaData ( pilAlphaStr )
44
45
#46の場合終了
47     wxBmap = wxImg . ConvertToBitmap ( )      # 同等の結果: wxBmap =
wx.BitmapFromImage( wxImg )
48     wxBmap
49を返す
50 #定義終了

```

注: マスク付きの GIF ファイルが読み込まれると、PIL はマスクをバイナリ値のアルファ透明度に自動的に変換します。これは通常、表示目的に適しています。wx も PIL も、マスク透明度付きの .ICO ファイルを適切に書き込むことはできません。写真以外の画像ファイルはすべて .PNG 形式で書き込む方がよいため、おそらくこれで問題ありません。

ナンピ

[🔗](#) この例では、NumPy配列をデータ ソースとして使用してビットマップを作成する方法を示します。NumPyはwxPython と比較して逆の列と行の順序を使用するので、幅と高さの座標ではなく、高さと幅の座標を使用して画像を生成する必要があります。また、RGB 形式の画像データには 'uint8' データ型が使用されることにも注意してください。


```

1 wx、numpy
2をインポートします。
3 def GetBitmap ( self、幅= 32、高さ= 32、色= ( 0、0、0 ) ):
4     配列= numpy.zeros ((高
さ,幅, 3 ) , ' uint8 ' ) 5     配列[:, :, ] =色
6     image = wx.EmptyImage (幅,高さ)
7     画像.SetData (配列.tostring (
) ) 8     wxBitmap = image.ConvertToBitmap ( ) # または:
wx.BitmapFromImage (image)
9     wxBitmapを返す

```

スライス

NumPyのスライス表記は、ここでは次のように機能します: [rowStart: rowEnd, columnStart: columnEnd, colourPlanes] 画像全体に単一の色を割り当てるには:
array[:, :] = (r,g,b) 赤のビットプレーン全体に値を割り当てるには:
array[:, :, 0] = 255 最初の行に色を割り当てるには: array[0] = (r,g,b)

例

この例では、NumPyの(拡張)スライス表記法を使用してメモリ内のイメージを変更する方法を示します。スライス表記法では、標準の負の値のインデックスセマンティクスが使用可能であることに注意してください。また、割り当て操作の適応性にも注意してください。行に3つのタプルを割り当てると、行内のすべての値が設定されますが、3つのタプルのリストを割り当てると、各値が個別に設定されます(項目の数が正しくない場合はエラーが発生します)。

行番号表示/非切替表示

```

1 wx、numpy
2をインポートします。
3 def GetBitmap ( self、幅= 32、高さ= 32、色=( 128、128、128 )、境界=
5、境界色=( 255、255、255 ) ):
4     「」
5     境界線付きのビットマップを作成します
6     「」
7     配列= numpy.zeros ((高
さ,幅, 3 ) , ' uint8 ' ) 8     配列[ border :- border , border :-
border , : ] =色
9     配列[: border , :, : ] = borderColour
10    配列[ -境界線:, :, : ] = borderColour
11    配列[:, : border ,:] = borderColour
12    配列[ :, -境界線:, : ] = borderColour
13    image = wx.EmptyImage (幅,
高さ) 14    イメージ.SetData (配列.tostring (
) ) 15    imageを返します。ConvertToBitmap ( ) # または:
wx.BitmapFromImage( image )

```

この例では、2つの色の間に水平グラデーションを作成します。スライス インデックスのため、NumPyは配列への割り当てで float から uint8 データ型に自動的に変換す

ることに注意してください。インデックスがないと、配列は既存の配列内でデータを割り当ててのではなく、別の numpy 配列を指すように上書きされます。

行番号表示/非切替表示

```

1  numpyを
   np2としてインポートする
3  def GetBitmap ( self、幅= 640、高さ= 480、左色=( 255、128、0 )、右色=(
64、0、255 ) ):
4      「
5      水平グラデーションを作成する
6      「
7      配列= np . zeros ( (高さ、幅、3 )、' uint8 ' )
8
9      # アルファは0.0から1.0までの線形グラデーションを持つ1次元配列です
10     アルファ= np .線形空間( 0. , 1. ,幅)
11
12     # これはアルファを使用してleftColourとrightColour
13の間を線形補間します colourGradient = np.outer (アルファ、左色) +
np.outer ( ( 1.-アルファ)、右
色) 14
15     # NumPyのブロードキャストルールは、宛先配列
16のすべての行にcolourGradientを割り当てます。 配列[:, :, :] =
colourGradient
17     image = wx.EmptyImage (幅、
高さ) 18     イメージ.SetData (配列.tostring (
) ) 19     wxBitmap = wx.BitmapFromImage ( image ) #または:
image.ConvertToBitmap ( )
20     wxBitmapを返す

```

スクリーンキャプチャ

この例では、フレームのクライアント領域を wxBitmap オブジェクト (およびそこからファイル) にキャプチャします。ここではエラー チェックが行われていないことに注意してください。各コンテキストを確認するには、おそらく Ok() か何かを使用する必要があります。画面の最初の描画の前にこれ呼び出した場合、何が起るかはわかりませんが、コンテキストにはデスクトップか何かが含まれている可能性があります。注: 以下のコードは、ウィンドウのクライアント領域のみをキャプチャします。Tagore は、wx.WindowDC でウィンドウ全体を取得し、wx.ScreenDC で画面の任意の部分を取得できると述べています。

行番号表示/非切替表示


```


1  デフ onSaveToFile (自分、イベント):
2      コンテキスト= wx.ClientDC ( self )
3      メモリ= wx.MemoryDC ( )
4      x , y =自分.クライアントサイズ
5      ビットマップ= wx.EmptyBitmap ( x , y
, - 1 ) 6      メモリ.SelectObject (ビットマップ)
7      メモリ.Blit ( 0 , 0 , x , y ,コンテキスト, 0 , 0 )
8      メモリ.SelectObject ( wx.NullBitmap )
9      ビットマップ.SaveFile ( ' test.bmp ' , wx.BITMAP_TYPE_BMP )

```

柔軟なスクリーンキャプチャアプリ

これは、一般的なスクリーンキャプチャモジュールと、最初にメイン画面全体をキャプチャし、次にその4つの小さな部分をキャプチャするデモアプリです。

 スクリーンショットWX.py

 スクリーンショットWX_Demo.py

行番号表示/非切替表示

```

1 デフ スクリーンキャプチャ (キャプチャ開始位置、キャプチャマップサイズ、デバッグ=
False ):
2     「
3     一般的なデスクトップ画面部分のキャプチャ - デスクトップの一部または全体
4
5     私の特定の画面ハードウェア構成:
6     wx.Display( 0 ) はプライマリデスクトップディスプレイモニター画面を
参照します
7     wx.Display( 1 ) は、拡張デスクトップディスプレイモニター画面（プ
ライマリ画面の上）を参照します
8
9     特定のデスクトップ画面サイズは:
10    スクリーンRect = wx.Display( n ).GetGeometry()
11
12    単一のシステム内の異なる wx.Display は異なる寸法を持つ場合があります
13    「
14
15    # wx.ScreenDC はデスクトップ全体へのアクセスを提供します
16
17    # これには、OS で有効になっている拡張デスクトップ モニター画面も含まれます。
scrDC = wx . ScreenDC ( ) # MSW にはバグがあります: 拡張画面が含まれて
いません
18    scrDcSize = scrDC .サイズ
19    scrDcSizeX、scrDcSizeY = scrDcSize
20
21    # クロスプラットフォーム適応:
22    scrDcBmap = scrDC.GetAsBitmap ( )
23    scrDcBmapSize = scrDcBmap.GetSize ( )
24    デバッグの場合:
25    ' DEBUG: scrDC.GetAsBitmap() のサイズ'
を印刷します。scrDcBmapSize 26
27    # scrDC.GetAsBitmap() メソッドがこのプラットフォームに実装されているか
どうかを確認します
28    if not scrDcBmap.IsOk ( ): # 未実装: 長い方法で画面ビッ
トマップを取得します
29
30    デバッグの場合:
31    print ' DEBUG: scrDC.GetAsBitmap() が機能しないため、
memDC.Blit() を使用します。'
32
33    # scrDC のサイズの新しい空の (黒の) 宛先ビットマップを作成します
34    scrDcBmap = wx . EmptyBitmap ( * scrDcSize ) # 無効な
元の割り当てを上書きします
35    scrDcBmapSizeX、scrDcBmapSizeY = scrDcSize
36
37    # scrDcBmapに関連付けられたDCツールを作成します
38    memDC = wx.MemoryDC ( scrDcBmap )
39
40    # 画面ビットマップの一部をコピー (blit、「ブロックレベル転送」) する
41    # 返されたキャプチャビットマップに挿入します

```

```

。 42          # memDC (scrDcBmap) に関連付けられたビットマップがブリティ先です
。 43
44          memDC . Blit ( 0 , 0 ,                                # この開始
座標にコピーします
。 45                                scrDcBmapSizeX, scrDcBmapSizeY, # このサイズの領
域をコピーします
。 46                                scrDC,                                # この DC の
ビットマップからコピーします
。 47                                0 , 0 , )                        # この開始座標からコピーします
。 48
49          memDC . SelectObject ( wx . NullBitmap )            # これで
wx.MemoryDC の使用を終了します。
50                                # scrDcBmap を他の用
途に解放します
。 51          その他:
52
53          デバッグの場合:
54          print 'デバッグ: scrDC.GetAsBitmap() の使用'
55
56          # このプラットフォームには scrDC.GetAsBitmap() が実装されています
。 57          scrDcBmap = scrDC.GetAsBitmap ()                # とても簡単です! デス
クトップビットマップ全体をコピーし
ます 。 58
59          デバッグの場合:
60          ' DEBUG: scrDcBmap.GetSize () 'を印刷、
scrDcBmap.GetSize ( )
61
62
#63の場合終了
64          scrDcBmap.GetSubBitmap ( wx.RectPS ( captureStartPos ,
captureBmapSize ) )
65を返す
66 #スクリーンキャプチャの定義終了

```

アプリケーションでは、キャプチャされた 5 つの wxBitmap オブジェクトすべてが PNG ファイルに保存されます。

wx.ScreenDC は、デスクトップ画面全体を、存在するすべての wx.Display() 領域の 1 つのシームレスな集合体として扱います。メインのデスクトップ画面に対する負の座標値が許可されます。プライマリ モニター画面 (タスクバーがある画面) の左または上にある拡張モニター画面の任意の部分をキャプチャするには、負の座標値を使用する必要があります。たとえば、拡張モニターがメイン モニターの真上になるように構成されているモニターが 2 つあります。両方のモニターの解像度を 1280x800 に設定しました。拡張画面全体だけをキャプチャするには、次の操作を行います。


行番号表示/非切替表示

```

1          screenWid = wx.SystemSettings.GetMetric ( wx.SYS_SCREEN_X ) 2
          screenHgt = wx.SystemSettings.GetMetric ( wx.SYS_SCREEN_Y ) 3
          captureSize = ( screenWid , screenHgt * 2 )    # 両方の画面は同じサ
イズに設定されます
。 4          wxBitmap = ScreenCapture (開始位置、キャプチャサイズ)

```

ImageMagickや PILなどの他の wxPython アドオン パッケージでも、任意のサイズのスクリーン ショットを撮ることができます。Python [🔗](https://wiki.wxpython.org/WorkingWithImages) イメージング ライブラリ:

 DesktopScreenShotPIL.py -- Ray Pasco (別名 WinCrazy)

ビットマップにテキストを書き込む

ビットマップにテキストを書き込むには、コンテキストを作成し、そのコンテキストのDrawTextメソッドを使用して実際の描画を行います。簡単な例:

行番号表示/非切替表示

```

1  wx
2  をインポート
3  def SetDcContext ( memDC、フォント=なし、色=なし ):
4      フォントの場合:
5          memDC.SetFont (フォント)
6      その他:
7          memDC.SetFont ( wx.NullFont )
8  です。
9      色の場合:
10         memDC.SetTextForeground (色)
11
12  #終了定義
13
14  def WriteTextOnBitmap (テキスト、ビットマップ、位置=( 0、0 )、フォント=
None、色= None ) :
15      「
16      ビットマップへの単純な書き込みでは、チェックは行われません
。17      「
18      memDC = wx.MemoryDC ( )
19      SetDcContext ( memDC、フォント、色)
20      memDC.SelectObject (ビット
マップ) 21      トライ:
22          memDC.DrawText (テキスト, pos [ 0 ] , pos [ 1
] ) 23      除く:
24
25  をパス
26      memDC.SelectObject ( wx.NullBitmap )
27  です。
28      ビットマップ
29  を返す
30  #定義終了

```

これを使用する目的の1つは、wx.BitmapButtonsの「ボタン ラベル」を作成することです。次のユーティリティ関数は、ビットマップの基本的な中央揃えのテキスト(複数行の可能性のある)キャプションを提供します。また、コンテキストのGetTextExtentメソッドを使用してテキストを中央揃えする方法も示します。

行番号表示/非切替表示

```

1  wx
2  をインポート
3  最小フォントサイズ= 4
4
5  def WriteCaptionOnBitmap (テキスト、ビットマップ、フォント= None、余白=
( 2、2 )、色= None ) :
6      「
7      指定されたキャプション (テキスト) をビットマップ

```

```

8に書き込む
9    のフォント (指定されていない場合はデフォルト) を使用する      余白は与えられ
ています。テキストが
10に収まるように努めます      ビットマップに
。11    「」
12    メモリ= wx.MemoryDC ( )
13    font =フォント または メモリ。GetFont ( )
14    textLines =テキスト.split ( ' \
n ' ) 15    フィット=偽
16    適合し ない:
17        合計幅= 0
18        合計高さ= 0
19        行数= []
20        テキスト内の行 数:
21            行 と 行[- 1 ] == ' \r 'の場合:
22                行=行[:- 1 ]
23
24        幅、高さ=範囲=メモリ。GetTextExtent (行)
25        合計幅=最大(合計幅, 幅)
26        合計高さ+=高さ
27        setLines . append ( (行、範囲))
28
29の終了
30    if ( totalWidth > ( bitmap.GetWidth ( ) - 2 * margins [ 0
]) または \
31        ( totalHeight > ( bitmap.GetHeight ( )- 2 * margins [ 0
]))
: 32
33        サイズ=フォント.GetPointSize ( ) - 1
34        サイズがMINIMUMFONTSIZE未満の場合:
35            fit = True # オーバードローします!!!
36        その他:
37            フォント.SetPointSize (サイズ)
38            メモリ.SetFont (フォント)
39        その他:
40            フィット=真
41
42の場合終了
43
44の間終了
45    設定されていない 場合は行数:
46        ビットマップ
47を返す
48    centreX , centreY = ( bitmap . GetWidth ()/ 2 ), ( bitmap .
GetHeight ()/ 2 )
49    x、y =センターX -(合計幅/ 2 )、センターY -(合計高さ/ 2 )
50    メモリ.SelectObject (ビット
マップ) 51    SetMemDcContext (メモリ、フォント、色)
52    setLines内のline、 ( deltaX、deltaY )の場合:
53        x =中心X - (デルタX / 2 )
54        メモリ.DrawText (行, x , y , )
55        y +=デルタY
56    メモリ.SelectObject ( wx.NullBitmap )
57    ビットマップ
58を返す
59
60 def SetMemDcContext (メモリ、フォント= None、色= None ) :
61
62    フォントの場合:
63        メモリ.SetFont (フォント)
64    その他:
65        メモリ.SetFont ( wx.NullFont )
66
67    色の場合:
68        メモリ.SetTextForeground (色)

```

69
70 #定義終了

透明性は少し難しいので、別の例を示します。(上記とは異なり、私の場合はうまくいきます)。

この例では、2つのビットマップが作成されます。1つは完全に白 (oldPix)、もう1つは完全に黒 (newPix) です。次に、黒いビットマップに赤いボックスが描画されます。次に、newPix の黒い色がマスクされ、newPix が oldPix (白いもの) の上にコピーされます。結果として、赤いボックスのみがコピーされます。

行番号表示/非切替表示

```

1  wx
2  をインポート
3  クラス Test ( wx.App ) :
4      デフ OnInit ( 自己 ):
5          oldPix  = wx.EmptyBitmap ( 300,300 )
6          newPix  = wx.EmptyBitmap ( 300,300 )
7          mem = wx.MemoryDC ( )
8          メモリSelectObject ( oldPix )
9          mem.Clear ( ) #画像をクリアする必要がある
10         mem.SelectObject ( newPix ) # wxEmptyBitmapは11のみ
11         なので mem.SetBackground ( wx.BLACK_BRUSH ) # スペース12を
12         割り当てます メモリクリア ( )
13
14         # これで白黒画像ができました
15
16         # 次に、黒い画像16の中央に白いボックスを描画します。
17         mem.SetPen ( wx.RED_PEN )
18         ペンの設定 mem.DrawLines ((( 100 , 200 ),( 100 , 100
19         ),( 200 , 100 ),( 200 , 200 ),( 100, 200 ) ) ) 19
20         メモリSelectObject ( oldPix )
21         newPix.SetMask ( wxMaskColour ( newPix , wx.BLACK ) )
22         mem.DrawBitmap ( newPix , 0,0,1 ) 23
23
24         oldPix.SaveFile ( " oldPix.bmp " , wx.BITMAP_TYPE_BMP )
25         newPix.SaveFile ( " newPix.bmp " , wx.BITMAP_TYPE_BMP )
26         True
27     を返す
28     アプリ=テスト (リダイレクト= False )
29     アプリのメインループ ( )

```

これは完全な wxPython アプリなので、実行して作成されたビットマップ oldPix.bmp と newPix.bmp で結果を確認してください。 -- Nikolai Hlubek

PythonMagick (16 ビット イメージング ライブラリ)

以下は、短いですが完全な PythonMagick/wxPython プログラムです。このプログラムは wxPython GUI を使用して、画像を読み込み、画像を表示し、簡単な画像処理操作 (しきい値) を実行します。PythonMagick、PIL、wxPython 画像間のさまざまな変換は、PythonMagick [Webサイト](https://www.pythonmagick.org/) で入手できます。

PythonMagickの詳しい説明については、PythonMagickページを参照してください。

PythonMagickには現在 Windows インストーラーしかないことに注意してください。

行番号表示/非切替表示

```

1  wxPython からwx
2をインポート  PythonMagick
3をインポートする
4 ID_FILE_OPEN = wx.wxNewId ( )
5 ID_FILE_EXIT  = wx.wxNewId ( )
6 ID_THRESHOLD = wx.wxNewId ( )
7
8 クラス ImagePanel ( wx . wxPanel ) :
9     def __init__ ( 自己、親、id ) :
10         wx.wxPanel .__init__ ( 自己、親、id )
11         self.image = None # wxPython
イメージ12         wx.EVT_PAINT ( self , self.OnPaint )
13です。
14     def display ( 自己、magickimage ) :
15         self.image = self.convertMGtoWX ( magickimage )
16です。         自己.リフレッシュ ( True )
17
18     デフ OnPaint ( 自己、イベント ) :
19         dc = wx.wxPaintDC ( 自己 )
20         自分なら.画像:
21             dc.DrawBitmap ( self.image.ConvertToBitmap
( ) , 0 , 0 ) 22
23     def convertMGtoWX ( 自己、magickimage ) :
24         img = PythonMagick.Image ( magickimage ) #コピーを作成
25         img . depth = 8 # ディスプレイ
26の深度のみ変更         img . magick = " RGB "
27         データ=画像.データ
28         wximg = wx.wxEmptyImage ( 画像.列 ( ) , 画像.行 ( ) )
29         wximg.SetData ( データ )
30         wximg
31を返す
32
33 クラス mtFrame ( wx.wxFrame ) :
34     def __init__ ( 自分、親、ID、タイトル ) :
35         wx.wxFrame .__init__ ( self , parent , ID , title ,
wx.wxDefaultPosition , wx.wxSize ( 500 , 400 )
) 36
37         self.iPanel = ImagePanel ( self
, - 1 ) 38         self . im = None # マジックイメージ
39
40         ## 「ファイル」メニュー
41を作成する         self.menuBar = wx.wxMenuBar ( )
42です。         self.menuFile = wx.wxMenu ( )
43         self.menuFile.Append ( ID_FILE_OPEN , "画像を開く ( & O ) " ,
" " )
44         wx.EVT_MENU ( self , ID_FILE_OPEN , self.OnOpen )
45         自己.menuFile.AppendSeparator ( )
46         self.menuFile.Append ( ID_FILE_EXIT , "終了 ( & X ) " , " "
)
47         wx.EVT_MENU ( self , ID_FILE_EXIT , self.OnExit )
48         self.menuBar.Append ( self.menuFile , "
& File " ) ; 49
50         ## 「プロセス」メニュー
51を構築         self.menuProcess = wx.wxMenu ( )
52         self.menuProcess.Append ( ID_THRESHOLD , "しきい値" , " " )
53         wx.EVT_MENU ( self , ID_THRESHOLD , self.OnThreshold )
54
55         self.menuBar.Append ( self.menuProcess , " & Process " )
56         自分自身. SetMenuBar ( self . menuBar )

```



```

57
58     デフ OnOpen (自分、イベント):
59         fd = wx.wxFileDialog ( self , "画像を開く" , " " , " " , " *
. * " , wx.wxOPEN )
60
61         fd.ShowModal
( ) == wx.wxID_OKの場合: 62             self.loadImage ( fd.GetPath ( )
) 63です。             fd .破壊()
64
65     def loadImage (自己、パス):
66         トライ:
67             self.im = PythonMagick.Image (パス)
68             自己.iPanel .ディスプレイ(自己.im )
69             IOError を除く:
70                 印刷 「ファイルを開けません」
71
72     ##----- プロセス -----
73
74     定義 OnThreshold (自分、イベント):
75         自己.im =自己.しきい値(自己.im , 0.5 )
76         自己.iPanel .ディスプレイ(自己.im )
77         #self.im.write('d:/threshold.tif')
78
79     定義 しきい値(自己、画像、しきい値):
80         「
81         閾値画像。入力閾値は正規化される (0-1.0)
82         「
83         img = PythonMagick.Image ( image ) #
コピー84         画像.しきい値(しきい値* 65535.0 )
85         画像
86     を返す
87     ##-----
88
89     デフ OnCloseWindow (自分、イベント):
90         自分を破壊する ( )
91
92     デフ OnExit (自分、イベント):
93         自分.閉じる( True )
94
95     #-----
-----
96
97     クラス mtApp ( wx.wxApp ) :
98         デフ OnInit (自己):
99             フレーム= mtFrame ( wx . NULL , - 1 , " MagickSimple1 " )
100             フレーム.表示 (True)
101             self.SetTopWindow (フレーム)
102             True
103     を返す
104     アプリ= mtApp ( 0 )
105     アプリのメインループ()

```

-ボブ・クリメック 2003年9月23日

ありがとう

もう一度お礼を申し上げます。このページから学んだアルファトリックを使用して、pil --> image を使用しました。\\d

WorkingWithImages (最終更新日時 2011-06-20 15:32:02 更新者pool-71-244-98-82)

注意:この Wiki のページを編集するには、TrustedEditorsGroupのメンバーである必要があります。