BREW & EZアプリ
ゲームプログラミング

○　第８回　画像の読み込み　○

画像の読み込み

BREWでは、画像はファイルまたはリソースから読み込むことができます。対応している形式は、BMP、PNG、JPEGです。ただし、BMPを１とすると、PNGは４、JPEGは１００の時間が掛かります。また、透過色をサポートしているインタフェースはIBITMAPだけなので、キャラクターなどで透明色を扱うには、BMP形式しか選択肢がありません。なお、BREWで推奨されているBMPの形式は、圧縮なしの色深度１，２，４，８ビットのものです(大きさには特に制限がなく、容量の限り読み込めます)。

ファイルは、BREWがサポートするファイルシステムのことです。携帯Javaと違い、BREWにはPCのようなファイルシステムが存在します。このファイルシステムは、ファイル名の最大長が64文字であるという制限を除けば、ディレクトリも作成できます。ファイルを扱うインタフェースにはIFileMgrやIFileなどがあります。

リソースは、アプリで使う表示文字列や画像など、実行コード以外の資源をひとつにまとめたものです。リソースは、ひとつのアプリにひとつである必要はなく、リソースを持たないアプリや２つ以上のリソースを使用するアプリを作成することもできます。BREWのリソースは、拡張子が.barで、BREW SDKに含まれるBREWリソースエディタで作成することができます。

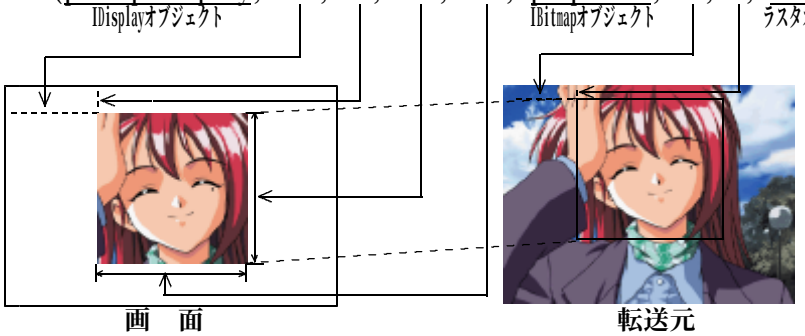
リソースから画像を読み込む関数には、ISHELL_LoadResBitmap関数とISHELL_LoadResImage関数があります。ISHELL_LoadResBitmap関数は、リソースからBMP形式の画像を読み込み、それを制御するためのIBitmapインタフェースを返します。ISHELL_LoadResImage関数は、リソースから画像を読み込み、それを制御するためのIImageインタフェースを返します。
IBitmapインタフェースはBMP形式しか扱えませんが、IImageはすべての形式を扱うことができます。ただし、IBitmapインタフェースはビットマップの管理と描画に、IImageインタフェースはアニメーションに重点を置いており、２つのインタフェースがサポートする関数は、かなりの違いがあります。

```
IBitmap   *pBmpChara;       // ビットマップを格納するインタフェース
IImage    *pImgAniChara;    // イメージを格納するインタフェース

// リソースからビットマップを読み込む
pBmpChara = ISHELL_LoadResBitmap(pMe->pIShell, "Resource.bar", IDB_CHARA);
                                 IShellオブジェクト  リソース    リソースID

// リソースから画像を読み込む
pImgAniChara = ISHELL_LoadResImage(pMe->pIShell, "Resource.bar", IDI_ANICHARA);
```

画像の描画

ビットマップを画面に表示する関数には、IDISPLAY_BitBlt関数があります。

```
// ビットマップの描画
IDISPLAY_BitBlt(pMe->pIDisplay, 160, 50, 150, 160, pBmpChara, 82, 9, AEE_RO_COPY);
                IDisplayオブジェクト                 IBitmapオブジェクト       ラスタオペレーションコード
```



最後の引数は、「ラスタオペレーションコード」と呼ばれるもので、転送元と転送先のビットマップデータをどのように結合するかという指定です。通常はAEE_RO_COPYを使用して単純に転送元から転送先にコピーするだけですが、画像を反転、くり抜き、合成といった効果を付加することができます。

ラスタオペレーションコード	演　　算	意　　味
AEE_RO_COPY	転送先 = 転送元	転送元を上書き
AEE_RO_NOT	転送先 = ~転送元	転送元のビット列を反転して上書き
AEE_RO_OR	転送先 = 転送先 | 転送元	転送先と転送元のビット列のORをとる
AEE_RO_XOR	転送先 = 転送先 ^ 転送元	転送先と転送元のビット列のXORをとる
AEE_RO_MERGENOT	転送先 = ~転送元 | 転送先	転送元のビット列を反転し、転送先とORをとる
AEE_RO_ANDNOT	転送先 = ~転送元 & 転送先	転送元のビット列を反転し、転送先とANDをとる
AEE_RO_TRANSPARENT		転送元の指定された色が転送されなくなる

▲ラスタオペレーションコード

IBitmapインタフェースは、ビットマップ間で転送を行うIBITMAP_BltIn関数とIBITMAP_BltOut関数をサポートしているほか、IBITMAP_QueryInterface関数でIDIBインタフェースを取得することによりピクセルに直接アクセスすることもできるので、ビットマップに様々な加工を行うことができます。IImageインタフェースには、このような機能はありません。

IImageインタフェースでは、IIMAGE_Draw関数またはIIMAGE_DrawFrame関数で画面に描画します。IImageインタフェースは、アニメーション機能に重点を置いているので、IBitmapインタフェースに比べ、用意されている関数はかなり異なります。

透過色

IBitmapインタフェースは、透過色をサポートしています。透過色を適用するには、はIBITMAP_SetTransparencyColor関数で透明にしたい色(転送したくない色)を指定し、ビットマップの描画および転送時にラスタオペレーションコード「AEE_RO_TRANSPARENT」を指定すれば、設定した色が転送されなくなります。ただし、IBITMAP_RGBToNative関数に渡す色は、デバイスに読み込まれた後の、それぞれのデバイスに合わせた色でなければならないので、MAKE_RGBマクロではなく、IBITMAP_RGBToNative関数で作成します。

```
NativeColor   color;

// 透過色設定(何度も行う必要はありません)
color = IBITMAP_RGBToNative (pBmpChara, MAKE_RGB(0, 0, 0));
IBITMAP_SetTransparencyColor(pBmpChara, color);

// 透過色を用いた転送
IDISPLAY_BitBlt(pMe->pIDisplay, 0, 0, 60, 60, pBmpChara, 0, 0, AEE_RO_TRANSPARENT);
```

画像の解放

IBitmapインタフェースに読み込まれているビットマップが不用になったら、IBITMAP_Release関数で解放します。同じように、IImageインタフェースに読み込まれている画像が不用になったら、IImage_Release関数で解放します。
解放されたインタフェースは、再び生成または初期化するまで使用できないので、インタフェースを格納していた変数にはNULLを代入し、無効であることを明示するようにします。こうしておくと、解放したインタフェースを初期化しないで不正使用したことによる、不安定な動作を起こすという深刻なバグを防ぐことができます。

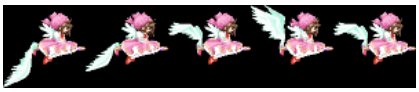
```
// 画像の解放
if(pBmpChara != NULL) {          // ポインタチェック。NULLの場合は解放しない(できない)
    IBITMAP_Release(pBmpChara); // インタフェースの解放
    pBmpChara = NULL;            // NULLを代入し、無効なインタフェースであることを示す
}
```

課　題

画像を読み込み、描画してみましょう。

(１)以下のような背景とキャラクターを作成し、それぞれ256色以下のビットマップ形式で保存しましょう。

背景(BG.bmp:240×298)

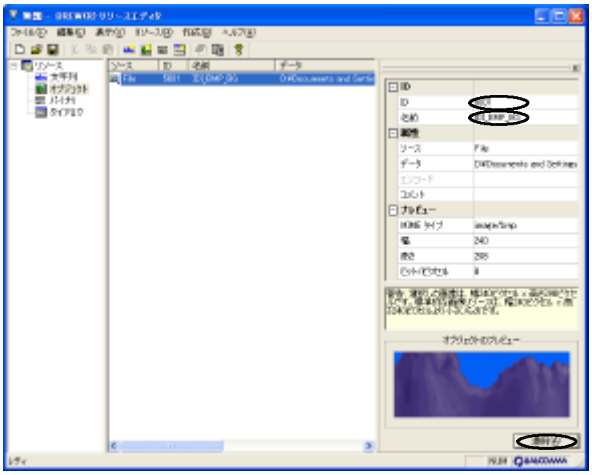


キャラクター(Chara.bmp)

(２)リソースを作成します。BREWリソースエディタを起動しましょう。

(３)リソースにビットマップを登録します。メニューから「リソース(R)→新規オブジェクト(O)」を選択しましょう。リソースに、画像を登録する領域(File 5001 IDI_OBJECT_5001)が作成されます。

(４)画像を登録します。「File 5001 IDI_OBJECT_5001」という文字列をダブルクリックしましょう。画像ファイルを選択するダイアログが表示されるので、背景となるビットマップファイルを選択します。



選択した画像によっては、警告になる場合があります。画像サイズ以外の警告は、画像の形式がBREW準拠ではないということになり、実行時に画像が正しく扱われるかどうかは機種依存となります。

(５)画像のプロパティを設定します。

IDには数字を、名前には英数字と'_'を、ほかのリソースと重複しないように入力します(ただし、リソースファイルが異なる場合は、重複してもかまいません)。IDに「5001」、名前に「IDI_BMP_BG」と入力し、下の適用ボタンをクリックしましょう。

これらは、リソースのヘッダファイル(.brh)で以下のように定義され、ISHELL_LoadResBitmap関数とISHELL_LoadResImage関数で必要になります。

```
#define IDI_BMP_BG        5001
```

(６)(３)から(５)を参考に、キャラクターとなる画像をリソースに登録しましょう。

(７)brxファイルを保存します。メニューから「ﾌｧｲﾙ(F)→保存(S)」を選択し、プロジェクトのフォルダに保存しましょう(名前は適当でかまいませんが、プロジェクトと同じ名前が無難です)。

※brxファイルは、リソースエディタが読み込み可能なリソースのファイルです。このファイルをリソースエディタで読み込めば、リソースを編集することができます。

(８)リソースのヘッダファイルとbarファイルを作成します。メニューから「作成(B)→リソーススクリプトのコンパイル(C)」を選択しましょう。選択後、(７)で指定したフォルダにリソースのヘッダファイル(*brxのファイル名*.brh)とリソースファイル(*brxのファイル名*.bar)が作成されます。

(９)ビットマップを扱うために必要なヘッダファイルをインクルードします。以下のプログラムを適切な場所に追加しましょう。

```
#include "AEEBitmap.h"          // Bitmap interface definitions
#include "AEEStdLib.h"

#include "brxのファイル名.brh"
```

(10)アプレット構造体で、ビットマップを制御するための変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
IBitmap   *pBmpBG;
```

(11)ビットマップを読み込みます。以下のプログラムを完成させ、*アプリケーション名*_InitSceneData関数の適切な場所に追加しましょう。

```
// 背景読み込み
pMe->pBmpBG = ISHELL_?????????????(pMe->???????, "brxのファイル名.bar", IDI_BMP_BG);
if(pMe->pBmpBG == NULL)
    return FALSE;
```

※画像は、ゲームで必要となるデータなので、InitSceneData関数で読み込みます。

(12)ゲーム終了時に、ビットマップが解放されるようにします。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
// 背景解放
if(pMe->pBmpBG != NULL) {
    IBITMAP_???????(pMe->pBmpBG);
    pMe->pBmpBG = ????;
}
```

※以下のようなマクロを定義しておくと、解放処理を１行にまとめることができます。

```
#define BITMAP_RELEASE(x) {if(x!=NULL) {IBITMAP_Release(x); x=NULL;}}
```

(13)背景を描画してみましょう。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
// 背景描画
IDISPLAY_??????(pMe->pIDisplay, 0, 0, 240, 298, pMe->pBmpBG, 0, 0, AEE_RO_????);
```

(14)(10)～(13)を参考に、キャラクターを読み込み、１コマ目だけ描画しましょう。

(15)キャラクターに透過色を設定しましょう。

(16)キャラクターをアニメーションさせましょう。アニメーションに必要な変数は、自由に追加してかまいません。ただし、グローバル変数とstatic変数が使用できないことに注意しましょう。