

パス名の操作 (IV)

指定ファイルの有無をチェックおよびパス名からドライブ番号を取得します。

PathFileExists 指定したファイルの有無をチェック

PathGetDriveNumber パス名からドライブ番号を取得



```
'=====
'= パス名の操作 (IV)
'= (PathFileExists.bas)
'=====
#include "Windows.bi"

' 指定のファイルの有無をチェック
Declare Function Api_PathFileExists& Lib "shlwapi" Alias "PathFileExistsA" (ByVal lpszPath$)

' パス名からドライブ番号を取得
Declare Function Api_PathGetDriveNumber& Lib "shlwapi" Alias "PathGetDriveNumberA" (ByVal pszPath$)

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "C:¥Temp¥Test.txt"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    If Api_PathFileExists(Edit1.GetWindowText) Then
        Text1.SetWindowText "存在します！"
    Else
        Text1.SetWindowText "存在しません！"
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Text1.SetWindowText "[" & Chr$(65 + Api_PathGetDriveNumber(Edit1.GetWindowText)) &
    "]"ドライブです！"
End Sub
```

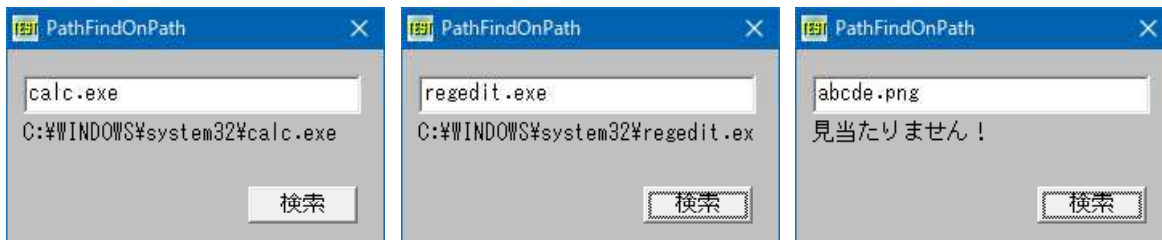
```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

パス名の操作 (V)

指定したファイルがWindows/system/system32に存在するかどうかを判断します。
PathFindOnPath 標準のパスおよび指定したパスからファイルを検索



```

'=====
'= パス名の操作 (V)
'= (PathFindOnPath.bas)
'=====
#include "Windows.bi"

' 標準のパスおよび指定したパスからファイルを検索
Declare Function Api_PathFindOnPath& Lib "shlwapi" Alias "PathFindOnPathA" (ByVal
pszPath$, ByVal ppszOtherDirs$)

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Edit1.SetWindowText "calc.exe"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var pPath As String
    Var Ret As Long

    Text1.SetWindowText ""

    pPath = Edit1.GetWindowText & String$(255, Chr$(0))

    Ret = Api_PathFindOnPath(pPath, ByVal 0)
    If Ret Then
        Text1.SetWindowText pPath
    Else
        Text1.SetWindowText "見当たりません！"
    End If
End Sub

```

```

End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

パス名の操作 (VI)

パス名操作に関するAPIをテストしています。

PathIsDirectory 指定したディレクトリが存在するかどうか判定

PathIsDirectoryEmpty バスで示されるディレクトリが空かどうか確認

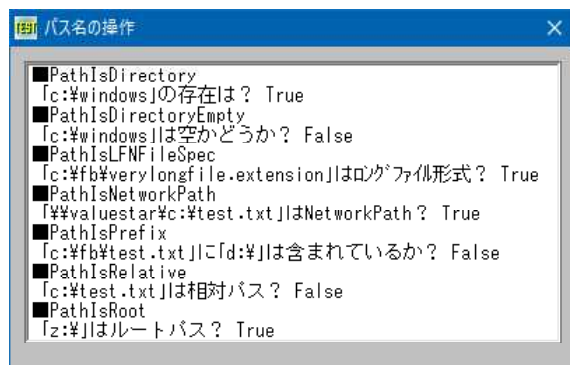
PathIsLFNFileSpec ファイル名前が長いフォーマットにあるかどうか判断

PathIsNetworkPath ネットワークパスかどうかの判断

PathIsPrefix 有効なプレフィックスを含むかどうか判断

PathIsRelative パスが相対かどうかを判断

PathIsRoot ルートパスかどうかを判断



```

'=====
'= パス名の操作 (VI)
'= (PathFunctions3.bas)
'=====
#include "Windows.bi"

```

' 指定したディレクトリが存在するかどうか判定

```

Declare Function Api_PathIsDirectory& Lib "shlwapi" Alias "PathIsDirectoryA" (ByVal
pszPath$)

```

' バスで示されるディレクトリが空かどうか確認

```

Declare Function Api_PathIsDirectoryEmpty& Lib "shlwapi" Alias "PathIsDirectoryEmptyA"
(ByVal pszPath$)

```

' ファイル名前が長いフォーマットにあるかどうか判断

```

Declare Function Api_PathIsLFNFileSpec& Lib "shlwapi" Alias "PathIsLFNFileSpecA" (ByVal
lpName$)

```

' ネットワークパスかどうかの判断

```

Declare Function Api_PathIsNetworkPath& Lib "shlwapi" Alias "PathIsNetworkPathA" (ByVal
pszPath$)

```

' 有効なプレフィックスを含むかどうか判断

```

Declare Function Api_PathIsPrefix& Lib "shlwapi" Alias "PathIsPrefixA" (ByVal
pszPrefix$, ByVal pszPath$)

```

' パスが相対かどうかを判断

```

Declare Function Api_PathIsRelative& Lib "shlwapi" Alias "PathIsRelativeA" (ByVal
pszPath$)

```

・ ルートパスかどうかを判断

```
Declare Function Api_PathIsRoot Lib "shlwapi" Alias "PathIsRootA" (ByVal pszPath$)
```

```
Var Shared List1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var Temp0 As String
```

```
    Var Temp1 As String
```

```
    Var State As String
```

```
    Var Msg As String
```

```
    Var Ret As Long
```

```
    Temp1 = "c:¥windows"
```

```
    List1.AddString "■PathIsDirectory"
```

```
    Ret = Api_PathIsDirectory(Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"の存在は? " & Msg
```

```
    List1.AddString "■PathIsDirectoryEmpty"
```

```
    Ret = Api_PathIsDirectoryEmpty(Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"は空かどうか? " & Msg
```

```
    Temp1 = "c:¥fb¥verylongfile.extension"
```

```
    List1.AddString "■PathIsLFNFileSpec"
```

```
    Ret = Api_PathIsLFNFileSpec(Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"はロングファイル形式? " & Msg
```

```
    Temp1 = "¥¥valuestar¥c:¥test.txt"
```

```
    List1.AddString "■PathIsNetworkPath"
```

```
    Ret = Api_PathIsNetworkPath(Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"はNetworkPath? " & Msg
```

```
    Temp0 = "d:¥"
```

```
    Temp1 = "c:¥fb¥test.txt"
```

```
    List1.AddString "■PathIsPrefix"
```

```
    Ret = Api_PathIsPrefix(Temp0, Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"に[" & Temp0 & "]"は含まれているか? " & Msg
```

```
    Temp1 = "c:¥test.txt"
```

```
    List1.AddString "■PathIsRelative"
```

```
    Ret = Api_PathIsRelative(Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"は相対パス? " & Msg
```

```
    Temp1 = "z:¥"
```

```
    List1.AddString "■PathIsRoot"
```

```
    Ret = Api_PathIsRoot(Temp1)
```

```
    GoSub *Hantei
```

```
    List1.AddString "[" & Temp1 & "]"はルートパス? " & Msg
```

```
Exit Sub
```

```
*Hantei
```

```
    If Ret <> 0 Then
```

```
        Msg = "True"
```

```
    Else
```

```
        Msg = "False"
```

```
    End If
```

```
Return
```



```
End Sub
```

```
' =====  
' =  
' =====  
While 1  
    WaitEvent  
Wend  
Stop  
End
```

パス名の振り分け

パス名をドライブ名・フォルダ名・ベースファイル名・拡張子に分けます。

引数

<パス名>パス名

<ドライブ名部分>ドライブ名を格納する変数。フォルダ名にはルートフォルダを表す¥、およびベースファイル名直前の¥が含まれます。

<ベースファイル名部分>ベースファイル名を格納する変数

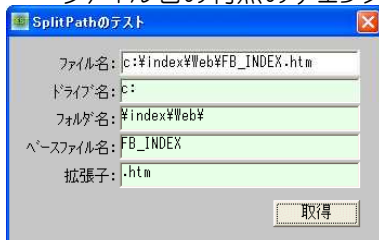
<拡張子部分>拡張子を格納する変数。拡張子にはベースファイル名直後の .が含まれます。

相対パス↔フルパスの変換は行いません。

ロングファイルネーム↔ショートファイルネームの変換は行いません。

ファイル名の有効性のチェックは行いません。

ファイル名の有無のチェックは行いません。



注) 5Ch (¥) を含む全角文字が混じったパスや . が2つ以上あるパスを正しく分割できないそうです。

情報源:<http://www.geocities.jp/fbasic63/fbn007.htm>

```
' =====  
' = パス名をドライブ名・フォルダ名・ベースファイル名・拡張子に分ける  
' =====  
#include "Windows.bi"  
#include "File.bi"  
  
Var Shared Edit1 As Object  
Var Shared Text(8) As Object  
Var Shared Button1 As Object  
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14  
For i = 0 To 8  
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))  
    Text(i).SetFontSize 14  
Next i  
  
' =====  
' =  
' =====  
Declare Sub Button1_on edec1 ()  
Sub Button1_on()  
    File$ = GetDlgItemText("Edit1")  
  
    SplitPath File$, Drv$, Fld$, Bfn$, Ext$  
  
    Edit1.SetWindowText File$  
    Text(5).SetWindowText Drv$  
    Text(6).SetWindowText Fld$  
    Text(7).SetWindowText Bfn$  
    Text(8).SetWindowText Ext$  
End Sub
```

```

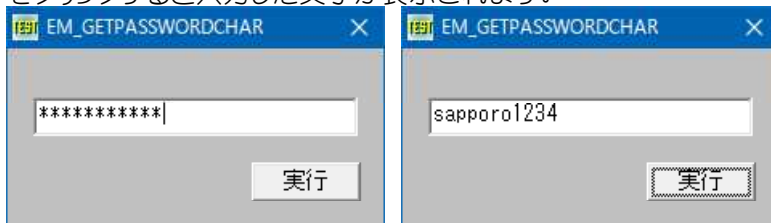
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

パスワード文字を取得 (I)

SendMessage ウィンドウにメッセージを送信
ShowWindow 指定されたウィンドウの表示状態を設定
EM_GETPASSWORDCHAR (&HD2) エディットコントロールのパスワード文字を取得
EM_SETPASSWORDCHAR (&HCC) エディットコントロールのパスワード文字を設定または削除
SW_HIDE (0) 指定のウィンドウを非表示にし他のウィンドウをアクティブ化
SW_SHOW (5) ウィンドウをアクティブ化し現在の位置とサイズで表示

EditBoxのプロパティで「パスワードあり」に設定しています。パスワードを入力すると「*」が表示され、「実行」ボタンをクリックすると入力した文字が表示されます。



```

'=====
'= パスワード文字を取得 ( I )
'= (EM_GETPASSWORDCHAR.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, ByVal lParam&)

' 指定されたウィンドウの表示状態を設定
Declare Function Api_ShowWindow& Lib "user32" Alias "ShowWindow" (ByVal hWnd&, ByVal
nCmdShow&)

#define EM_GETPASSWORDCHAR &HD2
#define EM_SETPASSWORDCHAR &HCC

#define SW_HIDE 0
#define SW_SHOW 5

' エディットコントロールのパスワード文字を取得する
' エディットコントロールのパスワード文字を設定または削除
' する
' 指定のウィンドウを非表示にし他のウィンドウをアクティブ化
' ウィンドウをアクティブ化し現在の位置とサイズで表示

Var Shared Edit1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Ret As Long

    Ret = Api_SendMessage(Edit1.GethWnd, EM_GETPASSWORDCHAR, 0, 0)
    Ret = Api_SendMessage(Edit1.GethWnd, EM_SETPASSWORDCHAR, 0, 1)

    'Edit1リフレッシュ
    Ret = Api_ShowWindow(Edit1.GethWnd, SW_HIDE) 'Edit1.ShowWindow 0

```

```

Ret = Api_ShowWindow(Edit1.GethWnd, SW_SHOW) 'Edit1.ShowWindow -1
End Sub

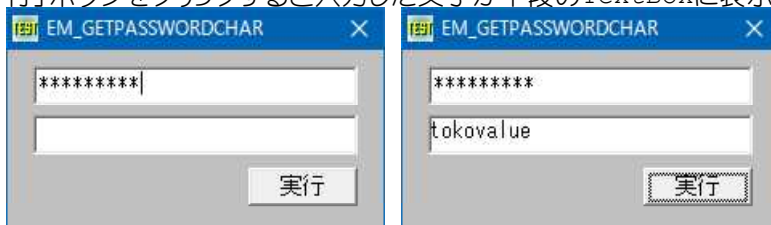
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

パスワード文字を取得 (II)

SendMessage ウィンドウにメッセージを送信
EM_GETPASSWORDCHAR(&HD2) エディットコントロールのパスワード文字を取得
WM_GETTEXT(&HD) コントロールのキャプション・テキストをバッファにコピー
WM_GETTEXTLENGTH(&HE) WM_GETTEXTの前に文字数を調べる

上段のEditBox(プロパティで「パスワードあり」に設定しています。)パスワードを入力すると「*」が表示され、「実行」ボタンをクリックすると入力した文字が下段のTextBoxに表示されます。



```

'=====
'= パスワード文字を取得 (II)
'= (EM_GETPASSWORDCHAR4.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

#define WM_GETTEXT &HD
#define WM_GETTEXTLENGTH &HE
#define EM_GETPASSWORDCHAR &HD2

'コントロールのキャプション・テキストをバッファにコピー
'WM_GETTEXTの前に文字数を調べる
'エディットコントロールのパスワード文字を取得する

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var length As Long
    Var strtmp As String
    Var Ret As Long

    If Api_SendMessage(Edit1.GethWnd, EM_GETPASSWORDCHAR, ByVal CLng(0), ByVal CLng(0))
<> 0 Then
        length = Api_SendMessage(Edit1.GethWnd, WM_GETTEXTLENGTH, ByVal 0, ByVal 0) + 1
        strtmp = Space$(length - 1)
        Ret = Api_SendMessage(Edit1.GethWnd, WM_GETTEXT, ByVal length, strtmp)
    End If

```

```

    Text1.SetWindowText strtmp
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

パスを表す文字列の最後にファイル拡張子を付加

PathAddExtension パスを表す文字列の最後にファイル拡張子を付け加える。既に付けられている場合は付加されない



```

'=====
'= パスを表す文字列の最後にファイル拡張子を付加
'= (PathAddExtension.bas)
'=====
#include "Windows.bi"

' パスを表す文字列の最後にファイル拡張子を付け加える。既に付けられている場合は付加されない
Declare Function Api_PathAddExtension& Lib "shlwapi" Alias "PathAddExtensionA" (ByVal
pszPath$, ByVal pszExt$)

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'= Chr$(0)を取り除く
'=====
Declare Function TrimNull (item As String) As String
Function TrimNull(item As String) As String
    Var ePos As Integer

    ePos = Instr(item, Chr$(0))
    If ePos Then
        TrimNull = Left$(item, ePos - 1)
    Else
        TrimNull = item
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "c:\testpath\dir\myfile"
End Sub

```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sSave As String
    Var Ret As Long

    sSave = Edit1.GetWindowText & String$(100, 0)

    Ret = Api_PathAddExtension(sSave, ".tst")

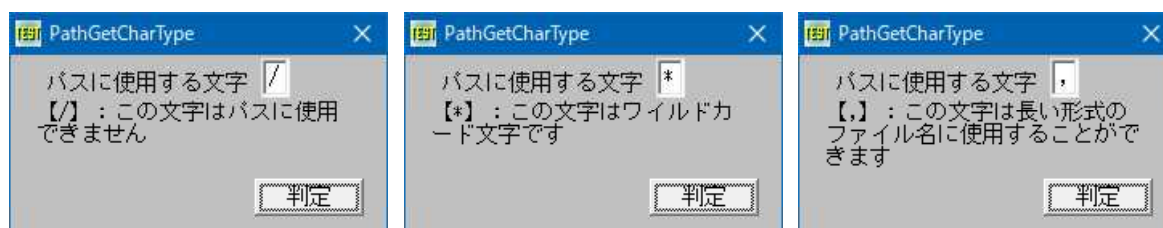
    Text1.SetWindowText TrimNull(sSave)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

パスを構成する文字の種別を判定

PathGetCharType パスを構成する文字の種別を判定



```

'=====
'= パスを構成する文字の種別を判定
'= (PathGetCharType.bas)
'=====
#include "Windows.bi"

#define GCT_INVALID &H0
#define GCT_LFNCHAR &H1
#define GCT_SEPARATOR &H8
#define GCT_SHORTCHAR &H2
#define GCT_WILD &H4

' パスに使用できない
' 長い形式のファイル名に使用することができる
' パスの区切りの文字である
' 短い形式のファイル名に使用することができる
' ワイルドカード文字である

' パスを構成する文字の種別を判定
Declare Function Api_PathGetCharType& Lib "shlwapi" Alias "PathGetCharTypeA" (ByVal ch
As Byte)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()

```

```

Sub Button1_on()

    Var MyStr As String
    Var txt As String
    Var Ret As Long

    MyStr = Edit1.GetWindowText

    Select Case Api_PathGetCharType(Asc(MyStr))
        Case GCT_INVALID
            txt = "この文字はパスに使用できません"
        Case GCT_LFNCHAR
            txt = "この文字は長い形式のファイル名に使用することができます"
        Case GCT_SHORTCHAR
            txt = "この文字は短い形式のファイル名に使用することができます"
        Case GCT_WILD
            txt = "この文字はワイルドカード文字です"
        Case GCT_SEPARATOR
            txt = "この文字はパスの区切りの文字です"
    End Select

    Text2.SetWindowText "[" & MyStr & "]: " & txt
End Sub

' =====
' =
' =====
Declare Sub Edit1_SetFocus edec1 ()
Sub Edit1_SetFocus()
    Text2.SetWindowText ""
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

パターン画像を使って転送

LoadImage 画像ファイルの読み込み

BitBlt コピー元からコピー先のデバイスコンテキストへ、指定された長方形内の各ピクセルの色データをコピー

PatBlt 指定の範囲内を現在のパターンブラシで塗りつぶす

CreatePatternBrush 指定されたビットマップパターンを持つ論理ブラシを作成

CreateCompatibleDC 指定されたDCに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成

SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

GetObject オブジェクト取得

DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放

GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```

'=====
'= パターン画像を使って転送
'= (PatBlt.bas)
'=====

#include "Windows.bi"

#define SRCCOPY &HCC0020      'そのまま転送
#define SRCPAINT &HEE0086     '転送先の画像とOR演算して転送
#define SRCAND &H8800C6      '転送先の画像とAND演算して転送
#define SRCINVERT &H660046    '転送先の画像とXOR演算して転送
#define SRCERASE &H440328     '転送先ビットマップを反転、その結果と転送元ビットマップを
                                論理AND演算子で結合
#define NOTSRCCOPY &H330008   '色を反転して転送
#define NOTSRCERASE &H1100A6  'コピー元の色と、コピー先の色を論理OR演算子で結合し、さ
                                らに反転
#define MERGECOPY &HC000CA    'コピー元の色と、コピー先の色を論理AND演算子で結合
#define MERGEPAIN &HBB0226    '反転した転送元ビットマップとパターンビットマップを論理
                                OR演算子で結合
#define PATCOPY &HF00021      '指定のパターンで描画先へコピー
#define PATINVERT &H5A0049    'XORで指定のパターンと描画先の色を組み合わせる
#define PATPAINT &HF80A09     '指定パターンの色と、コピー元の色を反転した色を論理OR
                                演算子で結合し、結果をコピー先の色と論理OR演算子
#define DSTINVERT &H550009    'コピー先を反転してコピー
#define BLACKNESS &H42        'すべてを黒にしてコピー
#define WHITENESS &HFF0062    'すべてを白にしてコピー
#define LR_LOADFROMFILE &H10 '外部ファイルからロードする
#define IMAGE_BITMAP 0        'ビットマップ

Type BITMAP
    bmType As Long
    bmWidth As Long
    bmHeight As Long
    bmWidthBytes As Long
    bmPlanes As Integer
    bmBitsPixel As Integer
    bmBits As Long
End Type

' 画像ファイルの読み込み
Declare Function Api_LoadImage& Lib "user32" Alias "LoadImageA" (ByVal hInst&, ByVal
lpzName$, ByVal uType&, ByVal cxDesired&, ByVal cyDesired&, ByVal fuLoad&)

' ビットブロック転送を行う。コピー元からコピー先のデバイスコンテキストへ、指定された長方形内の各ピクセルの色
データをコピー
Declare Function Api_BitBlt& Lib "gdi32" Alias "BitBlt" (ByVal hDestDC&, ByVal X&, ByVal
Y&, ByVal nWidth&, ByVal nHeight&, ByVal hSrcDC&, ByVal xSrc&, ByVal ySrc&, ByVal dwRop&)

' 指定の範囲内を現在のパターンブラシで塗りつぶす
Declare Function Api_PatBlt& Lib "gdi32" Alias "PatBlt" (ByVal hDC&, ByVal x&, ByVal y&,
ByVal nWidth&, ByVal nHeight&, ByVal dwRop&)

' 指定されたビットマップパターンを持つ論理ブラシを作成
Declare Function Api_CreatePatternBrush& Lib "gdi32" Alias "CreatePatternBrush" (ByVal
hBitmap&)

' 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
Declare Function Api_CreateCompatibleDC& Lib "gdi32" Alias "CreateCompatibleDC" (ByVal
hDC&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' オブジェクト取得
Declare Function Api_GetObject& Lib "gdi32" Alias "GetObjectA" (ByVal hObject&, ByVal
nCount&, lpObject As Any)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

```

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
Var Shared Picture(1) As Object
```

```
Var Shared Button(1) As Object
```

```
Var Shared Comb1 As Object
```

```
For i = 0 To 1
```

```
    Picture(i).Attach GetDlgItem("Picture" & Trim$(Str$(i + 1)))
```

```
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1))) : Button(i).SetFontSize 14
```

```
Next i
```

```
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
```

```
Var Shared hBitmap As Long
```

```
Var Shared ItemData(3) As Long
```

```
Var Shared hDC1 As Long
```

```
Var Shared hDC2 As Long
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var bmp As BITMAP
```

```
    Var mhDC As Long
```

```
    Var Ret As Long
```

```
    Comb1.AddString "PATCOPY" : ItemData(0) = PATCOPY
```

```
    Comb1.AddString "PATINVERT" : ItemData(1) = PATINVERT
```

```
    Comb1.AddString "WHITENESS" : ItemData(2) = WHITENESS
```

```
    Comb1.AddString "BLACKNESS" : ItemData(3) = BLACKNESS
```

```
    Comb1.SetCursel 0
```

'ファイルからBitmapをロード

```
hBitmap = Api_LoadImage(0, "TEST7.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE)
```

'Picture1のデバイスコンテキストを取得

```
hDC1 = Api_GetDC(Picture(0).GethWnd)
```

'メモリデバイスコンテキストを作成

```
mhDC = Api_CreateCompatibleDC(hDC1)
```

'Object取得

```
Ret = Api_GetObject(hBitmap, Len(bmp), bmp)
```

'Object選択

```
Ret = Api_SelectObject(mhDC, hBitmap)
```

'指定の(PictureBox)のデバイスコンテキストにメモリデバイスコンテキストのデータを転送

```
Ret = Api_BitBlt(hDC1, 0, 0, bmp.bmWidth, bmp.bmHeight, mhDC, 0, 0, SRCCOPY)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Picture(1).Cls
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button2_on edec1 ()
```

```
Sub Button2_on()
```

```
    Var Rop As Long
```



```

Var hObject As Long
Var hOrgObject As Long
Var Ret As Long

hDC2 = Api_GetDC(Picture(1).GethWnd)

'dwRop取得
Rop = ItemData(Combo1.GetCursel)

hObject = Api_CreatePatternBrush(hBitmap)
hOrgObject = Api_SelectObject(hDC2, hObject)

Ret = Api_PatBlt(hDC2, 0, 0, Picture(1).GetWidth, Picture(1).GetHeight, Rop)
Ret = Api_SelectObject(hDC2, hOrgObject)

Ret = Api_DeleteObject(hObject)
Ret = Api_ReleaseDC(Picture(0).GethWnd, hDC1)
Ret = Api_ReleaseDC(Picture(1).GethWnd, hDC2)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

バックスラッシュの無い場合¥を追加

PathAddBackslash パスを表す文字列の最後にバックスラッシュ¥を付け加える。含まれる場合は付け加えられない



```

'=====
'= バックスラッシュの無い場合¥を追加
'= (PathAddBackslash.bas)
'=====
#include "Windows.bi"

' パスを表す文字列の最後にバックスラッシュ¥を付け加える。含まれる場合は付け加えられない
Declare Function Api_PathAddBackslash& Lib "shlwapi" Alias "PathAddBackslashA" (ByVal pszPath$)

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'= Chr$(0)を取り除く
'=====
Declare Function TrimNull (item As String) As String
Function TrimNull(item As String) As String
    Var ePos As Integer

```

```

ePos = Instr(item, Chr$(0))
If ePos Then
    TrimNull = Left$(item, ePos - 1)
Else
    TrimNull = item
End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "c:\testpath\dir"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sSave As String
    Var Ret As Long

    sSave = Edit1.GetWindowText & String$(100, 0)

    Ret = Api_PathAddBackslash(sSave)

    Text1.SetWindowText TrimNull(sSave)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ハッチパターン作成とブラシの原点

CreateHatchBrush ハッチパターンの論理ブラシを作成

GetBrushOrgEx 現在のブラシの原点を取得

SetBrushOrgEx ブラシの原点を設定

SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

Rectangle 長方形の描画

DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放

GetDC 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイDCのハンドルを取得

ReleaseDC デバイスコンテキストを解放

HS_BDIAGONAL(3) 45度(右上から左下)

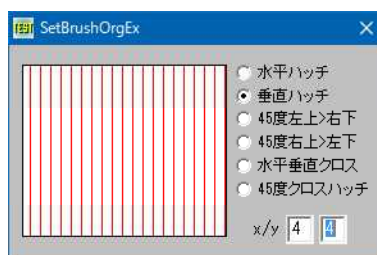
HS_CROSS(4) 水平と垂直クロスハッチ

HS_DIAGCROSS(5) 45度のクロスハッチ

HS_FDIAGONAL(2) 45度(左上から右下)

HS_HORIZONTAL(0) 水平ハッチ

HS_VERTICAL(1) 垂直ハッチ



```

'=====
'= ハッチパターン作成とブラシの原点
'= (SetBrushOrgEx.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type

' ハッチパターンの論理ブラシを作成
Declare Function Api_CreateHatchBrush& Lib "gdi32" Alias "CreateHatchBrush" (ByVal
nIndex&, ByVal crColor&)

' 現在のブラシの原点を取得
Declare Function Api_GetBrushOrgEx& Lib "gdi32" Alias "GetBrushOrgEx" (ByVal hDC&,
lpPoint As POINTAPI)

' ブラシの原点を設定
Declare Function Api_SetBrushOrgEx& Lib "gdi32" Alias "SetBrushOrgEx" (ByVal hDC&, ByVal
nXOrg&, ByVal nYOrg&, lppt As POINTAPI)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 長方形の描画
Declare Function Api_Rectangle& Lib "gdi32" Alias "Rectangle" (ByVal hDC&, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define HS_BDIAGONAL 3 '45度(右上から左下)
#define HS_CROSS 4 '水平と垂直クロスハッチ
#define HS_DIAGCROSS 5 '45度のクロスハッチ
#define HS_FDIAGONAL 2 '45度(左上から右下)
#define HS_HORIZONTAL 0 '水平ハッチ
#define HS_VERTICAL 1 '垂直ハッチ

Var Shared hBrushNew As Long
Var Shared hBrushOld As Long
Var Shared OldPt As POINTAPI
Var Shared NewPt As POINTAPI
Var Shared hDC As Long

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Radio(5) As Object

For i = 0 To 5
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1)))
    Radio(i).SetFontSize 12
Next i
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14

'=====
'=
'=====
Declare Function Index bdecl () As Integer

```

```

Function Index()
    Index = Val (Mid$(GetDlgRadioSelect("Radiol"), 6)) - 1
End Function

'=====
'=
'=====
Declare Sub SetHatch ()
Sub SetHatch()
    Var Hatch(5) As Long
    Var Ret As Long

    Hatch(0) = HS_HORIZONTAL
    Hatch(1) = HS_VERTICAL
    Hatch(2) = HS_FDIAGONAL
    Hatch(3) = HS_BDIAGONAL
    Hatch(4) = HS_CROSS
    Hatch(5) = HS_DIAGCROSS

    'ブラシの原点を取得保持
    Ret = Api_GetBrushOrgEx(hDC, OldPt)

    'ブラシの原点を設定
    NewPt.x = Val(Edit1.GetWindowText)
    NewPt.y = Val(Edit2.GetWindowText)
    Ret = Api_SetBrushOrgEx(hDC, NewPt.x, NewPt.y, NewPt)

    'ハッチパターンの論理ブラシを作成
    hBrushNew = Api_CreateHatchBrush(Hatch(Index), RGB(255, 0, 0))
    hBrushOld = Api_SelectObject(hDC, hBrushNew)

    Ret = Api_Rectangle(hDC, 10, 15, 170, 150)
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    hDC = Api_GetDC(GetHWnd)
    SetHatch
End Sub

'=====
'=
'=====
Declare Sub Edit1_Change edec1 ()
Sub Edit1_Change()
    Edit2.SetSelText 0, 0
    Edit1.SetSelText 0, -1
    SetHatch
End Sub

'=====
'=
'=====
Declare Sub Edit2_Change edec1 ()
Sub Edit2_Change()
    Edit1.SetSelText 0, 0
    Edit2.SetSelText 0, -1
    SetHatch
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 (Cancel As Integer, Mode As Integer)
Sub MainForm_QueryClose(Cancel As Integer, Mode As Integer)
    If Cancel = 0 Then
        Ret = Api_DeleteObject(Api_SelectObject(hDC, hBrushOld))
    
```

```

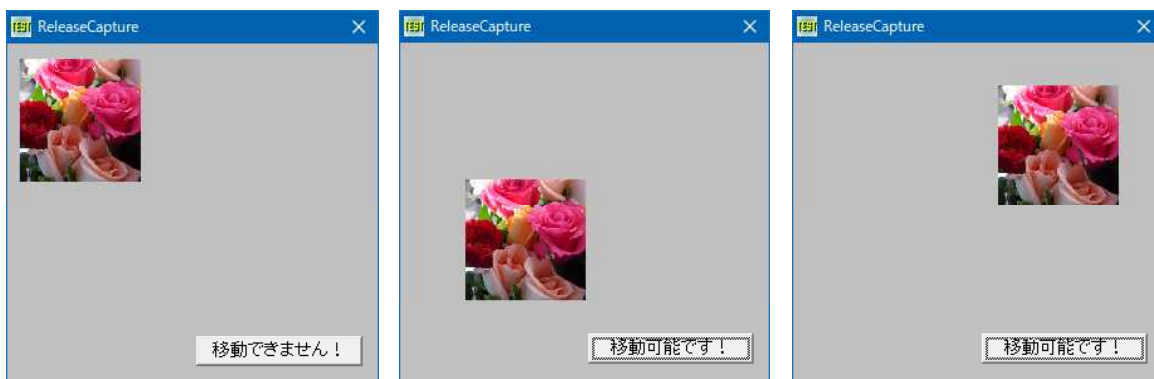
Ret = Api_SetBrushOrgEx (hDC, OldPt.x, OldPt.y, OldPt)
Ret = Api_ReleaseDC (GethWnd, hDC)
End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ピクチャボックスの移動

ReleaseCapture マウスのキャプチャを解放
SendMessage ウィンドウにメッセージを送信
WM_NCLBUTTONDOWN (&HA1) 非クライアント領域で左マウスボタンを押す
HTCAPTION (2) タイトルバーをクリックしたことを示す



```

'=====
'= ピクチャボックスの移動
'= (ReleaseCapture.bas)
'=====
#include "Windows.bi"

' マウスのキャプチャを解放
Declare Function Api_ReleaseCapture& Lib "user32" Alias "ReleaseCapture" ()

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal wMsg&, ByVal wParam&, lParam As Any)

#define WM_NCLBUTTONDOWN &HA1
#define HTCAPTION 2
#define vbArrow 1
#define vbDefault 0

Var Shared Picture1 As Object
Var Shared Button1 As Object
Var Shared Bitmap As Object

Picture1.Attach GetDlgItem("Picture1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
BitmapObject Bitmap

Var Shared mMove As Integer

'=====
'=
'=====

```

```

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Bitmap.LoadFile "flower.bmp"
    Picture1.DrawBitmap Bitmap, 0, 0
    Bitmap.DeleteObject

    Picture1.SetMousePointer vbArrow
    Button1.SetWindowText "移動できません！"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Picture1.SetMousePointer vbDefault
    Button1.SetWindowText "移動可能です！"
End Sub

'=====
'=
'=====
Declare Sub Picture1_MouseDown edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal x As Single, ByVal y As Single)
Sub Picture1_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single,
ByVal y As Single)
    Var Ret As Long

    If Picture1.GetMousePointer = vbDefault Then
        Ret = Api_ReleaseCapture
        Ret = Api_SendMessage(Picture1.GethWnd, WM_NCLBUTTONDOWN, ByVal HTCAPTION, ByVal
0)
        mMove = True
    End If
End Sub

'=====
'=
'=====
Declare Sub Picture1_MouseMove edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal x As Single, ByVal y As Single)
Sub Picture1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single,
ByVal y As Single)
    If mMove Then
        Picture1.SetMousePointer vbArrow
        mMove = False

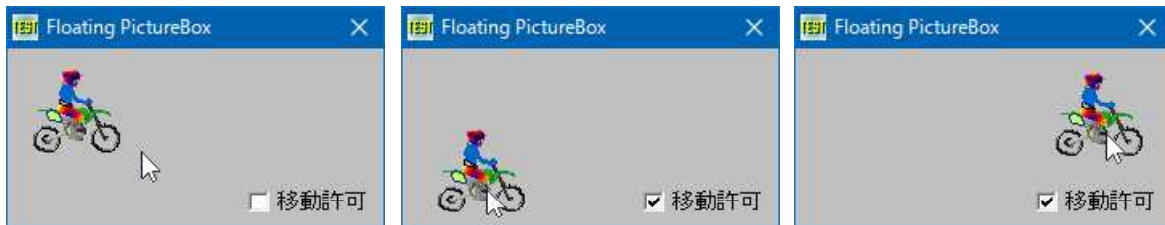
        Button1.SetWindowText "移動できません！"
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ピクチャボックスをフローティング表示

ReleaseCapture マウスのキャプチャを解放
SendMessage ウィンドウにメッセージを送信



```

'=====
' = ピクチャボックスをフローティング表示
' = (WM_NCLBUTTONDOWN.bas)
'=====
#include "Windows.bi"

#define WM_NCLBUTTONDOWN &H01
#define HTCAPTION 2

' マウスのキャプチャを解放
Declare Function Api_ReleaseCapture Lib "user32" Alias "ReleaseCapture" ()

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

Var Shared Picture1 As Object
Var Shared Check1 As Object
Var Shared Bitmap As Object

Picture1.Attach GetDlgItem("Picture1")
Check1.Attach GetDlgItem("Check1") : Check1.SetFontSize 14
BitmapObject Bitmap

'=====
' =
'=====
Declare Sub Picture1_MouseDown edec1 (Button As Integer, Shift As Integer, x As Single, y
As Single)
Sub Picture1_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
    Var Ret As Long

    If Check1.GetCheck = 1 Then
        Ret = Api_ReleaseCapture()
        Ret = Api_SendMessage(Picture1.GetHwnd, WM_NCLBUTTONDOWN, ByVal HTCAPTION, ByVal
0)
    End If
End Sub

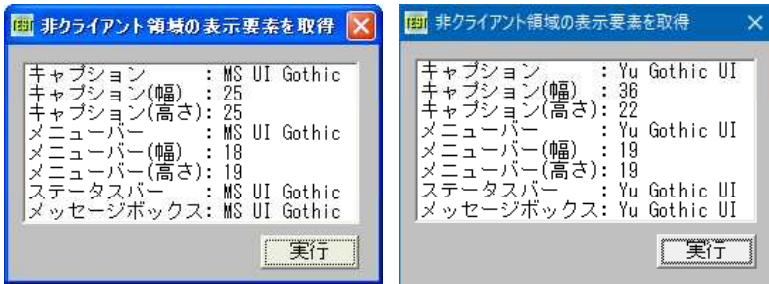
'=====
' =
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Bitmap.LoadFile "bike.bmp"
    Picture1.DrawBitmap Bitmap, 0, 0
    Bitmap.DeleteObject
End Sub

'=====
' =
'=====
While 1
    WaitEvent
Wend
Stop
End

```

非クライアント領域の表示要素を取得

SystemParametersInfo システム全体に関するパラメータを取得・設定
SPI_GETNONCLIENTMETRICS (41) NONCLIENTMETRICS構造体を取得



```
'=====
'= 非クライアント領域の表示要素を取得
'= (NonClientMetrics.bas)
'=====

#include "Windows.bi"

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight          As Long
    lfWidth           As Long
    lfEscapement       As Long
    lfOrientation      As Long
    lfWeight           As Long
    lfItalic           As Byte
    lfUnderline        As Byte
    lfStrikeOut        As Byte
    lfCharSet          As Byte
    lfOutPrecision     As Byte
    lfClipPrecision    As Byte
    lfQuality          As Byte
    lfPitchAndFamily   As Byte
    lfFaceName         As String * LF_FACESIZE
End Type

Type NONCLIENTMETRICS
    cbSize             As Long
    iBorderWidth       As Long
    iScrollWidth       As Long
    iScrollHeight      As Long
    iCaptionWidth      As Long
    iCaptionHeight     As Long
    lfCaptionFont       As LOGFONT
    iSmCaptionWidth    As Long
    iSmCaptionHeight   As Long
    lfSmCaptionFont    As LOGFONT
    iMenuWidth         As Long
    iMenuHeight        As Long
    lfMenuFont         As LOGFONT
    lfStatusFont       As LOGFONT
    lfMessageFont      As LOGFONT
End Type

#define SPI_GETNONCLIENTMETRICS 41      '非クライアント領域に関する情報を定義する
                                       NONCLIENTMETRICS構造体を取得

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
```



```

Button1.Attach GetDlgItem("Button1") : Button1.setFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var ncm As NONCLIENTMETRICS
    Var ContainFontName As String
    Var MenuBarFontName As String
    Var StatusBarFontName As String
    Var MsgBoxFontName As String
    Var Ret As Long

    'リストボックスをクリア
    List1.ResetContent

    '非クライアント領域の表示要素を定義する構造体を初期化
    ncm.cbSize = Len(ncm)

    '非クライアント領域の表示要素を取得
    Ret = Api_SystemParametersInfo(SPI_GETNONCLIENTMETRICS, Len(ncm), ncm, 0)

    '非クライアント領域の表示要素を表示
    ContainFontName = ncm.lfCaptionFont.lfFaceName
    List1.AddString "キャプション" : " & Left$(ContainFontName, InStr(ContainFontName,
Chr$(0)) - 1)
    List1.AddString "キャプション(幅) : " & Trim$(Str$(ncm.iCaptionWidth))
    List1.AddString "キャプション(高さ) : " & Trim$(Str$(ncm.iCaptionHeight))

    StatusBarFontName = ncm.lfStatusFont.lfFaceName
    MenuBarFontName = ncm.lfMenuFont.lfFaceName

    List1.AddString "メニューバー" : " & Left$(MenuBarFontName, InStr(MenuBarFontName,
Chr$(0)) - 1)
    List1.AddString "メニューバー(幅) : " & Trim$(Str$(ncm.iMenuWidth))
    List1.AddString "メニューバー(高さ) : " & Trim$(Str$(ncm.iMenuHeight))
    List1.AddString "ステータスバー" : " & Left$(StatusBarFontName,
InStr(StatusBarFontName, Chr$(0)) - 1)

    MsgBoxFontName = ncm.lfMessageFont.lfFaceName
    List1.AddString "メッセージボックス : " & Left$(MsgBoxFontName, InStr(MsgBoxFontName,
Chr$(0)) - 1)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

日付・時刻の書式を取得

日付表示・時刻表示の書式を取得します。
GetDateFormat 日付表示の書式を取得
GetTimeFormat 時刻表示の書式を取得

地域と言語のオプションで設定された書式(フォーマット)に準じます。
 例では、today\$で取得した日付・時刻をシステムタイム構造体に代入し地域オプションで設定したフォーマットで出力しています。



```
'=====
'= 日付・時刻の書式を取得
'= (GetDateFormat.bas)
'=====
#include "Windows.bi"
```

```
Type SYSTEMTIME
    wYear      As Integer
    wMonth     As Integer
    wDayOfWeek As Integer
    wDay       As Integer
    wHour      As Integer
    wMinute    As Integer
    wSecond    As Integer
    wMilliseconds As Integer
End Type
```

' 日付表示の書式を取得

```
Declare Function Api_GetDateFormat Lib "Kernel32" Alias "GetDateFormatA" (ByVal
    Locale&, ByVal dwFlags&, lpDate As SYSTEMTIME, ByVal lpFormat$, ByVal lpDateStr$, ByVal
    cchDate&)
```

' 時刻をフォーマットし、指定された地域に対応する時刻文字列を作成

```
Declare Function Api_GetTimeFormat Lib "kernel32" Alias "GetTimeFormatA" (ByVal
    Locale&, ByVal dwFlags&, lpTime As SYSTEMTIME, ByVal lpFormat As Any, ByVal lpTimeStr$,
    ByVal cchTime&)
```

```
Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Timer1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 20
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 20
Timer1.Attach GetDlgItem("Timer1")
```

```
'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.SetInterval 100
    Timer1.Enable -1
End Sub
```

```
'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var Buffer As String
    Var st As SYSTEMTIME
```

```

Var Ret As Long

st.wDay = Val (Right$ (Today$, 2))
st.wMonth = Val (Mid$ (Today$, 6, 2))
st.wYear = Val (Left$ (Today$, 4))

st.wHour = Val (Left$ (Time$, 2))
st.wMinute = Val (Mid$ (Time$, 4, 2))
st.wSecond = Val (Right$ (Time$, 2))

Buffer = String$(255, 0)
Ret = Api_GetDateFormat (ByVal 0, 0, st, ByVal 0, Buffer, Len(Buffer))
Buffer = Left$(Buffer, InStr(1, Buffer, Chr$(0)) - 1)
Text1.SetWindowText Buffer

Buffer = String$(255, 0)
Ret = Api_GetTimeFormat (ByVal 0, 0, st, ByVal 0, Buffer, Len(Buffer))
Buffer = Left$(Buffer, InStr(1, Buffer, Chr$(0)) - 1)
Text2.SetWindowText Buffer

End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ビットマップの画像情報を取得

CreateCompatibleDC 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成

GetObject オブジェクト取得

GetType 指定のオブジェクトのタイプを取得

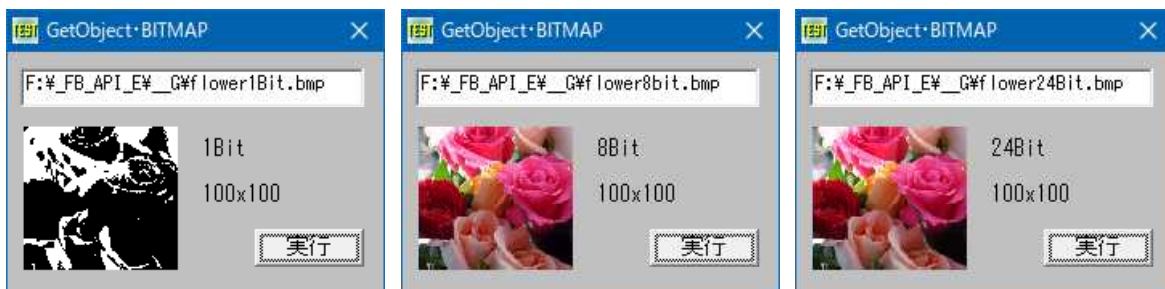
LoadImage 画像ファイルの読み込み

SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

BitBlt ビットブロックを転送

GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```

'=====
'= ビットマップの画像情報を取得
'= (GetObject2.bas)
'=====
#include "Windows.bi"

```

Type BITMAP

bmType As Long

bmWidth As Long

bmHeight As Long

bmWidthBytes As Long

bmPlanes As Integer

bmBitsPixel As Integer

'ビットマップ タイプを指定 (0)

'ビットマップの幅を指定

'ビットマップの高さを指定

'1 走査線あたりのバイト数を指定

'カラープレーンを指定 (通常1)

'1 ピクセルを定義するのに必要なビット数を指定

```

        bmBits           As Long           'ビット データが格納されているの配列へのポインタを指定
End Type

#define OBJ_DC 3           'デバイスコンテキスト
#define OBJ_ENHMETADC 12   '拡張メタファイルのデバイスコンテキスト
#define OBJ_ENHMETAFILE 13 '拡張メタファイル
#define OBJ_EXTPEN 11      '拡張ペン
#define OBJ_FONT 6         'フォント
#define OBJ_MEMDC 10       'メモリデバイスコンテキスト
#define OBJ_METADC 4        'メタファイルのデバイスコンテキスト
#define OBJ_METAFILE 9     'メタファイル
#define OBJ_PAL 5          'パレット
#define OBJ_PEN 1          'ペン
#define OBJ_REGION 8       '領域
#define OBJ_BITMAP 7

#define IMAGE_BITMAP 0     'ビットマップ
#define IMAGE_CURSOR 2     'カーソル
#define IMAGE_ENHMETAFILE 3 '拡張メタファイル
#define IMAGE_ICON 1       'アイコン
#define LR_LOADFROMFILE &H10 '外部ファイルからロードする
#define LR_CREATEDIBSECTION &H2000 'デバイス独立ビットマップとしてロードする
#define SRCCOPY &HCC0020   'そのまま転送

' 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
Declare Function Api_CreateCompatibleDC& Lib "gdi32" Alias "CreateCompatibleDC" (ByVal
hDC&)

' オブジェクト取得
Declare Function Api_GetObject& Lib "gdi32" Alias "GetObjectA" (ByVal hObject&, ByVal
nCount&, lpObject As Any)

' 指定のオブジェクトのタイプを取得
Declare Function Api_GetObjectType& Lib "gdi32" Alias "GetObjectType" (ByVal hgdiobj&)

' 画像ファイルの読み込み
Declare Function Api_LoadImage& Lib "user32" Alias "LoadImageA" (ByVal hInst&, ByVal
lpzName$, ByVal uType&, ByVal cxDesired&, ByVal cyDesired&, ByVal fuLoad&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' ビットブロック転送を行う。コピー元からコピー先のデバイスコンテキストへ、指定された長方形内の各ピクセルの色
データをコピー
Declare Function Api_BitBlt& Lib "gdi32" Alias "BitBlt" (ByVal hDestDC&, ByVal X&, ByVal
Y&, ByVal nWidth&, ByVal nHeight&, ByVal hSrcDC&, ByVal xSrc&, ByVal ySrc&, ByVal dwRop&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Picture1 As Object
Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Picture1.Attach GetDlgItem("Picture1")
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared FileName As String
Var Shared hDC As Long

```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var bm As BITMAP
    Var hBitmap As Long
    Var BitmapDC As Long
    Var Ret As Long

    hDC = Api_GetDC(Picture1.GethWnd)

    Picture1.Cls

    'ファイルからBitmapをロード
    hBitmap = Api_LoadImage(0, FileName, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE Or
LR_CREATEDIBSECTION)

    '呼び出し失敗の場合終了
    If (hBitmap = 0) Then
        A% = MsgBox("Bitmap Load Error", "Error, Bitmapファイルではありません！", 0, 2)
        End
    End If

    'ロードしたBitmapのデバイスコンテキストを作成
    BitmapDC = Api_CreateCompatibleDC(0)

    '失敗した場合プロシージャを抜ける
    If (BitmapDC = 0) Then
        A% = MsgBox("Device Context Error", "Error, デバイスコンテキストを作成できません！",
0, 2)
        Exit Sub
    End If

    '作成されたデバイスコンテキストにBitmapをAttach
    Ret = Api_SelectObject(BitmapDC, hBitmap)

    'イメージ情報を取得
    Ret = Api_GetObject(hBitmap, Len(bm), bm)

    'Pictureに描画
    Ret = Api_BitBlt(hDC, 0, 0, bm.bmWidth, bm.bmHeight, BitmapDC, 0, 0, SRCCOPY)

    Text1.SetWindowText Trim$(Str$(bm.bmBitsPixel)) & "Bit"
    Text2.SetWindowText Trim$(Str$(bm.bmWidth)) & "x" & Trim$(Str$(bm.bmHeight))

    Ret = Api_ReleaseDC(hDC, Picture1.GethWnd)
End Sub

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
Sub Edit1_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount(DF)
    FileName = GetDropFileName(DF, 0)
    Edit1.SetWindowText FileName
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

    Ret = Api_ReleaseDC(Picture1.GethWnd, hDC)
End

```

```
End Sub
```

```
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ビットマップパターンを持つ論理ブラシを作成

CreatePatternBrush 指定されたビットマップパターンを持つ論理ブラシを作成

FillRect ブラシで矩形領域を塗りつぶす

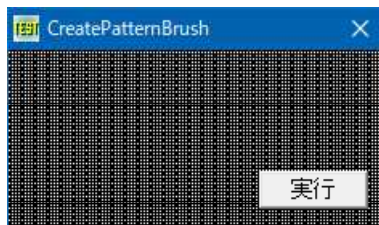
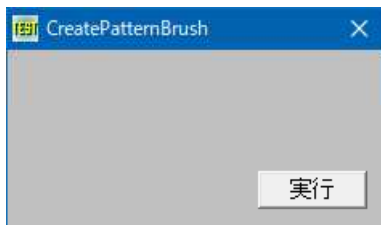
SetRect RECT構造体の値を設定

DeleteObject システムリソースを解放

CreateBitmap 指定された幅・高さ・色形式を持つビットマップを作成

GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```
'=====
'= ビットマップパターンを持つ論理ブラシを作成
'= (CreatePatternBrush.bas)
'=====
#include "Windows.bi"
```

```
Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type
```

' 指定されたビットマップパターンを持つ論理ブラシを作成

```
Declare Function Api_CreatePatternBrush& Lib "gdi32" Alias "CreatePatternBrush" (ByVal hBitmap&)
```

' ブラシで矩形領域を塗りつぶす

```
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hDC&, lpRect As RECT, ByVal hBrush&)
```

' RECT構造体の値を設定

```
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&, ByVal Y1&, ByVal X2&, ByVal Y2&)
```

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放

```
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

' 指定された幅・高さ・色形式を持つビットマップを作成

```
Declare Function Api_CreateBitmap& Lib "gdi32" Alias "CreateBitmap" (ByVal nWidth&, ByVal nHeight&, ByVal nPlanes&, ByVal nBitCount&, lpBits As Any)
```

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```

Var bBytes(8) As Integer

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var rc As RECT
    Var hDC As Long
    Var mBrush As Long
    Var hBitmap As Long
    Var Ret As Long

    hDC = Api_GetDC(GetHwnd)

    For mBrush = 0 To 7 Step 2
        bBytes(mBrush) = 170
        bBytes(mBrush + 1) = 85
    Next

    'ビットマップを作成
    hBitmap = Api_CreateBitmap(8, 8, 1, 1, bBytes(1))

    'パターンブラシの作成
    mBrush = Api_CreatePatternBrush(hBitmap)
    Ret = Api_SetRect(rc, 0, 0, GetWidth, GetHeight)

    'フォームを塗りつぶす
    Ret = Api_FillRect(hDC, rc, mBrush)

    '解放
    Ret = Api_DeleteObject(mBrush)
    Ret = Api_DeleteObject(hBitmap)
    Ret = Api_ReleaseDC(hWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

表計算風？項目入力例

表計算のような項目に入力する場合、FlexGridのようなものがないので文字数固定のセルを2頁に分けて入力する方法を考えてみました。

エディットボックスで入力しフォーカスが外れるとピクチャボックスに書き出しています。

マウスを置いたセルは淡い黄色の背景色になり、文字入力後【Enter】押下で次のセルに移動します。

図の例では、住所1のセル入力後【Enter】で次のセルに移動しますが、住所2のセルは途中で切れているので2頁目に切り替わります。

セルに文字が入っている場合は文字の最後尾にカーソルが移動し、数値が入っている場合は全範囲選択状態で入力待ちになります。

手っ取り早く試用できるようGRID.LZHにテストデータ共々入れてあります。

名前	ヨミ	敬称	連名1	連名2	連名3	会社名	役職	〒No	住所1	住所2
相川 欽助	アウキンソク	様	ミドリ			札幌岡村センター前		007-0021	根室市幸町1丁目3-1	
安達 邦明	アタ'ブン'アキ	様						003-0022	札幌市白石区南郷通14丁目2-1	山田マンション302
井川 国安	イケン'クニヤ	様	あずき	琢郎	優香			080-0039	北見市とん田	
江藤 五郎	エウ'ゴ'ロウ	様						002-8028	札幌市北区篠路6条8丁目	
大山 大寒	オヤマ'ダイ'カン	様						081-2273	札幌市南区豊港234	
鎌田 浩介	カマタ'ヒロ'サイ	様						080-0030	帯広市西8条北3丁目	入力後『Enter』で確定
亀田 勇作	カメタ'ユウ'サク	様	こころ					040-0001	函館市五稜郭	
川谷 光男	カハタ'ミツ'オ	様				北海製菓販売社	専務取締役	081-2285	札幌市南区豊野5条4丁目1-2	
川中 美由紀	カハナ'ミユ'キ	様	康夫					005-0841	札幌市南区石山1条6丁目	
北村 末治	キタムラ'スエ'チ	様	吉香					003-0022	札幌市白石区南郷通13丁目南5-1	山田パレス205
古泉 順見郎	コイズミ'ノブ'ロウ	様						083-0021	網走市南十一条西1丁目	

名前	ヨミ	敬称
相川 欽助	アウキンソク	様
安達 邦明	アタ'ブン'アキ	様
井川 国安	イケン'クニヤ	様
江藤 五郎	エウ'ゴ'ロウ	様
大山 大寒	オヤマ'ダイ'カン	様
鎌田 浩介	カマタ'ヒロ'サイ	様

住所2	電話	金額	文字1	文字2	文字3	数値1	数値2	数値3	数値4
山田マンション302	011-123-4567	208,000			文字列指定にカーソル	25	30	11	33
		225,000				15	18	22	18
		10,500			数値の場合は	25	21	35	18
		33,800			金銭を範囲指定→	11	8	88	1,250
		0			『Enter』で変更なし	0	0	0	0
入力後『Enter』で確定項目移動します		123,545				0	0	0	0
		89,524				0	33	0	0
		0				0	0	0	0
		0				0	0	0	0
		0				2,585	156	21	4,067
山田パレス205	011-063-1234	0				0	0	0	0
		20,506	okaten			0	0	0	0

住所2	電話	金額
		208,000
山田マンション302	011-123-4567	225,000
		10,500
		33,800
		0
入力後『Enter』で確定項目移動します		123,545

メインフォームにエディットボックス、ピクチャボックス、水平スクロールバー、コマンドボタンを貼り付けます。
 エディットボックス:複数行入力あり(Enter移動部の方法にる)、垂直オートスクロールあり、3D表示なし、可視なしに設定します。

枠無し、背景色を淡い黄色に設定するためリソースを書き換える必要があります。

- フォーム :ピクセルモード
- ピクチャボックス :ピクセルモード、境界線あり、境界線あり、可視なしに設定。保存ボタンは確認のために設けています。
- スクロールバー :あまり意味はありません。テスト用です。

MAINFORM

保存

リソースの書き換え(初期状態)

```

BEGIN
__CREATE3, XWT_EDIT, 0x40810144L, 0x0L

```



```

__ATTRDATA3, 0x0L
__WINDOWTITLE, "¥0"
__CONTROLNAME, "EDIT1¥0"
__LIMITTEXT3, 0x12L
__CTRLTEXTCOLOR, 0x0L
__CTRLBACKCOLOR, 0xfffffL
__FONTSIZE, 8
__FONTNAME2, "MS ゴシック¥0", 1, 186
__IMEMODE3, 0x1L
__WINDOWSIZE3, 0x0L, 0x10L, 0x22L, 0x5eL, 0x16L
__EVENT

```

上記赤文字の部分を下記のとおり書き換えます。

イ.境界線なし

```

__CREATE3, XWT_EDIT, 0x40810144L, 0x0L
      ↓ 8を0に
__CREATE3, XWT_EDIT, 0x40010144L, 0x0L

```

ロ.白を薄い黄色に(あまり薄色に設定すると液晶では判別しにくいようです)

```

__CTRLBACKCOLOR, 0xfffffL
      ↓
__CTRLBACKCOLOR, 0xe0ffffL

```

```

'=====
'= EXCEL風入力テスト
'=====
#include "Windows.bi"

```

```

Var Shared MainForm As Object
Var Shared Edit As Object
Var Shared Picture As Object
Var Shared HScroll As Object
Var Shared Button As Object

```

```

Var Shared CrLf$ As String
Var Shared EPos As Integer
Var Shared Col As Integer
Var Shared Row As Integer
Var Shared ColMax As Integer
Var Shared ColMax1 As Integer
Var Shared ColMax2 As Integer
Var Shared RowMax As Integer
Var Shared RowHeight As Integer
Var Shared YPos As Integer
Var Shared XPos(21) As Integer
Var Shared Mx As Integer
Var Shared My As Integer
Var Shared Dt$(21,30) As String
Var Shared Item$(21,3) As String
Var Shared CellWidth(21) As Integer
Var Shared FName$ As String
Var Shared FrameTop As Integer
Var Shared FrameBottom As Integer
Var Shared FrameLeft As Integer
Var Shared FrameRight As Integer
Var Shared FrameWidth As Integer
Var Shared FrameWidth2 As Integer
Var Shared FontSize As Integer
Var Shared FrmWidth As Integer
Var Shared FrmHeight As Integer
Var Shared DataMax As Long
Var Shared PageFlg As byte
Var Shared ColOfst As Integer
Var Shared OffSet As Integer

```

```

'
'CRLF取得位置(Enter移動用)
'列番号(項目)
'行番号
'セル最大値
'1面表示(1~Col1)
'2面表示(ColMax2~ColMax)
'行最大値
'行間隔
'Y軸(横罫線、文字表示位置)
'X軸(縦罫線、文字表示位置)
'X軸(マウスクリック位置)
'Y軸(マウスクリック位置)
'データ格納(最大20×30)
'0:項目名 1:文字数 2:IME制御 3:数値扱
'セル幅
'フォント
'枠上位置
'枠下位置
'枠左位置
'枠右位置
'1面+2面項目総幅
'2面項目総幅
'フォントサイズ
'フォーム幅
'フォーム高
'
'1面/2面
'
'

```

```

MainForm.Attach GethWnd
Edit.Attach GetDlgItem("Edit1")
Picture.Attach GetDlgItem("Picture1")
HScroll.Attach GetDlgItem("HScroll1")
Button.Attach GetDlgItem("Button1")

```

[illegible]

```

For i = 1 To ColMax
    For J = 0 To 3
        Read Item$(I, J)
    Next J
    If i > 0 Then CellWidth(I) = Val(Item$(i, 1)) * FontSize / 2 + 8 ' +8:文字前後4ドット
    XPos(i) = XPos(i - 1) + CellWidth(i) + 1
    FrameWidth = FrameWidth + CellWidth(i) + 1 'グリッド幅
Next I

'-----
'ピクチャボックス用意
'-----

Picture.SetWindowSize FrmWidth - 8 - FrameLeft * 2, FrameBottom ' -8 (4 * 2:フォーム枠
                                     立体部)

Picture.MoveWindow FrameLeft, FrameTop
Button.SetWindowSize 30, 24 '保存(テスト用)
Button.MoveWindow 4, FrmHeight - 60
Button.ShowWindow -1
Separate_Point '面/2面区切位置取得

If RowMax * RowHeight + FrameTop + 20 > FrmHeight Then
    A% = MessageBox(GetWindowText, "行数を減らしてください!", 0, 2) : Stop : End
Else If FrameWidth2 > FrmWidth - 8 - FrameLeft * 2 Then
    A% = MessageBox(GetWindowText, "フィールド数を減らすかフォントサイズを小さくしてください!",
0, 2) : Stop : End
Else If FontSize Mod 2 <> 0 Then
    A% = MessageBox(GetWindowText, "フォントサイズは偶数で指定してください!", 0, 2) : Stop :
End
End If

PageFlg = 0
Offset = 0
Col = 1
Row = 1
Grid_Draw
Data_Read
Data_Dsp
Picture.ShowWindow -1
EditBox_Move
End sub

'=====
'= 切替位置取得
'=====

Sub Separate_Point()
    WD = 0
    For i = 1 To ColMax
        WD = WD + CellWidth(i) + 1
        If WD > FrmWidth - FrameLeft * 2 Then
            ColMax1 = i - 1 '★1面(1～ColMax1)
            Exit For
        End If
    Next i

    WD = 0
    For i = ColMax To 1 Step -1
        WD = WD + CellWidth(i) + 1
        If WD >= FrmWidth - FrameLeft * 2 Then
            ColMax2 = i '★2面(ColMax2～ColMax)
            Exit For
        End If
    Next i

    WD = 0
    For i = ColMax1 + 1 To ColMax
        WD = WD + CellWidth(i) + 1
    Next i

    FrameWidth2 = WD '2面項目総幅
    WD = 0

```

```

For i = 1 To ColMax2
    WD = WD + CellWidth(i) + 1
Next i
ColOfst = WD                                'オフセット
End sub

'=====
'= Picture1描画
'=====
Sub Grid_Draw()
    Picture.Cls

    '項目名下罫線
    Picture.Line(XPos(0), 0) - (XPos(ColMax) - 1, RowHeight), , 14, bf

    For i = 0 To ColMax

        '縦罫線(項目)
        Picture.Line(XPos(i) - OffSet, 0) - (XPos(i) - OffSet, RowHeight), , 1

        '縦罫線
        Picture.Line(XPos(i + 1) - OffSet, RowHeight) - (XPos(i + 1) - OffSet,
FrameBottom), , 14

        '縦罫線
        If i < ColMax Then Picture.Line(XPos(i) - OffSet + 1, 0) - (XPos(i) - OffSet + 1,
RowHeight), , 15
        Next

        '項目名下罫線
        Picture.Line(XPos(0), RowHeight) - (XPos(ColMax) - 1, RowHeight), , 1

        '横罫線
        Picture.Line(XPos(0), 1) - (XPos(ColMax), 1), , 15

        '横罫線
        Picture.Line(XPos(0), RowHeight) - (XPos(ColMax), RowHeight), , 1

    For i = RowHeight * 2 To RowHeight * RowMax Step RowHeight

        '横罫線
        Picture.Line(XPos(0), i) - (XPos(ColMax), i), , 14
    Next

    '外枠(↑°↓°↑°↓°↑°↓°↑°境界線なし時)
    Picture.Line(XPos(0), 0) - (FrmWidth - FrameLeft * 2 - 1, FrameBottom), , 0, b

    HScroll1.SetWindowSize FrmWidth - 8 - FrameLeft * 2, 16
    HScroll1.MoveWindow FrameLeft, FrameBottom + RowHeight + 4
    HScroll1.ShowWindow -1
    HScroll1.SetScrollRange 0, ColMax + 1
    HScroll1.SetScrollPos Col
    HScroll1.SetScrollStep 1, 1

    For i = 1 To ColMax
        Picture.Symbol(XPos(i - 1) - OffSet + (CellWidth(i) - Len(Item$(i, 0)) * FontSize /
2) / 2 + 4, 6), Item$(i, 0), 1, 1, 0
    Next

    Data_Dsp
End sub

'=====
'= データ表示
'=====
Sub Data_Dsp()
    YPos = RowHeight
    If PageFlg = 0 Then
        ST = 1
        ED = ColMax

```

```

Else
    ST = ColMax2 + 1
    ED = ColMax
End If

For i = 1 To RowMax
    For j = ST To ED
        If Val(Item$(J, 3)) = 1 Then '数値表示
            Num$ = Right$("###,###,###,###",Val(Item$(J, 1)))
            If Left$(Num$, 1) = "," Then Num$ = " " & Right$(Num$, Len(Num$) -1)
            Picture.Symbol(XPos(J - 1) - OffSet + 4, YPos + (i - 1) * RowHeight + 4),
Format$(Val(Dt$(j, i)), Num$), 1, 1
        Else '文字表示
            Picture.Symbol(XPos(J - 1) - OffSet + 4, YPos+(I - 1) * RowHeight + 4),
Left$(Dt$(j, i), Val(Item$(j, 1))), 1, 1
        End If
    Next j
Next i
End sub

'=====
'= スクロールバーでの移動
'=====
Declare Sub HScroll11_Change edec1 ()
Sub HScroll11_Change()
    Col = HScroll1.GetScrollPos '位置取得

    If PageFlg = 0 Then '★1面
        If Col > ColMax1 Then '最大項目を超過
            Edit.ShowWindow 0
            PageFlg = 1
            OffSet = ColOfst
            Grid_Draw
        Else If Col < 1 Then
            Edit.ShowWindow 0
            Col = ColMax
            Row = Row - 1 : If Row < 1 Then Row = 1
            PageFlg = 1
            OffSet = ColOfst
            Grid_Draw
        End If
    Else '★2面
        If Col > ColMax Then
            Edit.ShowWindow 0
            Col = 1
            Row = Row + 1 : If Row > RowMax Then Row = 1 '行制限
            PageFlg = 0
            OffSet = 0
            Grid_Draw
        Else If Col <= ColMax2 Then
            Edit.ShowWindow 0
            Row = Row - 1 : If Row < 1 Then Row = 1
            PageFlg = 0
            OffSet = 0
            Grid_Draw
        End If
    End If

    EditBox_Move
End sub

'=====
'= マウスによる行位置取得
'=====
Sub Picture1_Click()
    Mx = Mouse(0)
    My = Mouse(1)

    If PageFlg = 0 Then ColMaxEnd = ColMax1 Else ColMaxEnd = ColMax

```

```

For i = 1 To ColMaxEnd
    If Mx + OffSet > XPos(i - 1) And Mx + OffSet < XPos(i) Then
        Col = i
        Exit for
    Else
        Col = 0
    End If
Next i
Select Case My
    Case RowHeight To FrameBottom
        Row = (My - FrameTop) ¥ RowHeight + 1
        YPos = FrameTop + (Row - 1) * RowHeight
    Case Else
        Row = 0
End Select
EditBox_Move
End sub

'=====
'= EditBox移動
'=====
Sub EditBox_Move()

    '面・カーソル位置確認
    txt$ = AKCnv$(Trim$(Str$(PageFlg + 1))) & "面/Col=" & Format$(Col, "##")
    txt$ = txt$ & "/Row=" & Format$(Row, "##") & "/Mx=" & Format$(Mx, "####") & "/My=" &
Format$(My, "####")
    MainForm.SetWindowText txt$

    'マウスのセル範囲外クリック時
    If Col = 0 Or Row = 0 Then
        Edit.ShowWindow 0
        Exit Sub
    End If

    YPos = FrameTop + RowHeight + (Row - 1) * RowHeight
    Edit.SetImeMode Val(Item$(Col, 2)) 'IMEモード設定
    Edit.SetWindowSize CellWidth(Col), RowHeight - 1 '-1:罫線消去防止
    Edit.MoveWindow XPos(Col - 1) - OffSet + FrameLeft + 2, YPos + 2
    Edit.ShowWindow -1
    Edit.SetLimitText Val(Item$(Col, 1)) + 2'+2:CrLf$(Enter移動方法による)
    Edit.SetWindowText RTrim$(Left$(Dt$(Col, Row), Val(Item$(Col, 1))))

    If Val(Item$(Col, 3)) = 1 Then '数値選択の場合
        Edit.SetSelText 0, -1
    Else '文字選択の場合
        Edit.SetSelText Len(RTrim$(Dt$(Col, Row))), Len(RTrim$(Dt$(Col, Row)))
    End If

    HScroll.SetScrollPos Col
    Edit.SetFontName FtName$
    Edit.SetFontSize FontSize
    Edit.SetFontBold 0
    Edit.SETFOCUS
End sub

'=====
'= イベント
'=====
While 1
    WaitEvent
Wend

'=====
'= 文字入力
'=====
Declare Sub Edit1_Change edec1 ()
Sub Edit1_Change()
    If PageFlg = 0 Then OffSet = 0 Else OffSet = ColOfst

```

```

OldDt$ = Dt$(Col, Row)
Dt$(Col, Row) = Edit.GetWindowText()
EPos = InStr(Dt$(Col, Row), CrLf$)

If EPos <> 0 Then
    Dt$(Col, Row) = Mid$(Dt$(Col, Row), 1, EPos - 1) & Mid$(Dt$(Col, Row), EPos + 2)
    Edit.SetWindowText Dt$(Col, Row)
    If Dt$(Col, Row) = "" Then Dt$(Col, Row) = OldDt$
    OldDt$ = Dt$(Col, Row)
    Picture.Line(XPos(Col - 1) - Offset + 2, YPos - FrameTop + 1) - (XPos(Col - 1) -
Offset + CellWidth(Col), YPos - FrameTop + RowHeight - 1), , 15, bf

    If Val(Item$(Col, 3)) = 1 Then                                '数値表示
        Num$ = Right$("###,###,###,###", Val(Item$(Col, 1)))
        If Left$(Num$, 1) = "," Then Num$ = " " + Right$(Num$, Len(Num$) - 1)
        Picture.Symbol(XPos(Col - 1) - Offset + 4, YPos - FrameTop + 4),
Format$(Val(Dt$(Col, Row)), Num$), 1, 1
    Else                                                         '文字表示
        Picture.Symbol(XPos(Col - 1) - Offset + 4, YPos - FrameTop + 4), Dt$(Col, Row),
1, 1
    End If

    '-----
    Col = Col + 1                                                '次列
    If PageFlg = 0 Then                                          '★1面
        If Col > ColMax1 Then
            PageFlg = 1
            Offset = ColOfst
            Edit.ShowWindow 0
            Grid_Draw
        End If
        If Col > ColMax Then                                     '最終列の場合
            Row = Row + 1                                         '次行の
            Col = 1                                               '最初の列
            If Row > RowMax Then Row = RowMax                     '最終行の場合
        End If
        EditBox_Move                                           'EditBox移動
    Else                                                         '★2面
        If Col < ColMax1 Or Col > ColMax Then
            PageFlg = 0
            Offset = 0
            Edit.ShowWindow 0
            Grid_Draw
        End If
        If Col > ColMax Then                                     '最終列の場合
            Col = 1                                               '最初の列
            Row = Row+1 : If Row > RowMax Then Row = 1           '最終行の場合
        End If
        EditBox_Move                                           'EditBox移動
    End If
End If
End sub

'=====
'= CSVデータ読込
'=====
Sub Data_Read()
    On Error Goto *Er_Trap

    FlNo = FreeFile
    Open "TEST.CSV" For Input As #FlNo
    i = 0
    SetMousePointer 2

    Do Until Eof(FlNo)
        i = i + 1
        Input #FlNo, Dt$(1, i)
        Input #FlNo, Dt$(2, i)
        Input #FlNo, Dt$(3, i)
        Input #FlNo, Dt$(4, i)
    
```

```

        Input #FlNo, Dt$(5, i)
        Input #FlNo, Dt$(6, i)
        Input #FlNo, Dt$(7, i)
        Input #FlNo, Dt$(8, i)
        Input #FlNo, Dt$(9, i)
        Input #FlNo, Dt$(10, i)
        Input #FlNo, Dt$(11, i)
        Input #FlNo, Dt$(12, i)
        Input #FlNo, Dt$(13, i)
        Input #FlNo, Dt$(14, i)
        Input #FlNo, Dt$(15, i)
        Input #FlNo, Dt$(16, i)
        Input #FlNo, Dt$(17, i)
        Input #FlNo, Dt$(18, i)
        Input #FlNo, Dt$(19, i)
        Input #FlNo, Dt$(20, i)
    Loop
    Close #FlNo

    DataMax = i
    SetMousePointer 0
    Exit Sub

*Er_Trap
    SetMousePointer 0
    Close #FlNo
    A% = MsgBox(GetWindowText, "読込に失敗しました！『TEST.CSV』が用意されていないか、データ形式が違います！(i=" & Trim$(Str$(i)) & ")", 0, 0)
    Resume *Er_Return

*Er_Return
    On Error Goto 0
End sub

'=====
' = CSVデータ保存(DataMax固定)
'=====
Declare Sub Button1_ON edec1 ()
Sub Button1_ON()
    FlNo = freefile

    On Error Goto *Er_Trap

    If DataMax = 0 Then A% = MsgBox(GetWindowText, "保存するデータがありません！", 0, 2) :
Exit Sub
    Open "TEST.CSV" For Create As #FlNo
    SetMousePointer 2

    For i = 1 To DataMax
        ZD$ = ""
        ZD$ = ZD$ & RTrim$(Dt$(1, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(2, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(3, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(4, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(5, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(6, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(7, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(8, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(9, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(10, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(11, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(12, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(13, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(14, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(15, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(16, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(17, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(18, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(19, i)) & ","
        ZD$ = ZD$ & RTrim$(Dt$(20, i))
    
```



```

        Print #FlNo,ZD$
    Next

    Close #FlNo

    SetMousePointer 0
    A% = MsgBox(GetWindowText, "CSV型式で保存しました！", 0, 2)
    Exit Sub

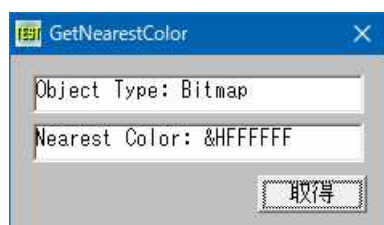
*Er_Trap
    SetMousePointer 0
    Close #FlNo
    A% = MsgBox(GetWindowText, "保存に失敗しました！(Erl=" & Str$(Erl) & "/Err=" &
Str$(Err) & "/i=" & Trim$(Str$(i)) & ")", 0, 2)
    Resume *Er_Return

*Er_Return
    On Error Goto 0
End sub

```

表示できる色に最も近いカラー値を取得(1)

GetNearestColor 指定のカラー値に対して、デバイスが表示できる色に最も近いカラー値を取得
GetObjectType 指定のオブジェクトのタイプを取得
CreateCompatibleDC 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
CreateBitmap 指定された幅・高さ・色形式を持つビットマップを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
DeleteDC 指定されたデバイスコンテキストを削除
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```

'=====
'= 表示できる色に最も近いカラー値を取得
'= (GetNearestColor.bas)
'=====
#include "Windows.bi"

' 指定のカラー値に対して、デバイスが表示できる色に最も近いカラー値を取得
Declare Function Api_GetNearestColor& Lib "gdi32" Alias "GetNearestColor" (ByVal hDC&,
ByVal crColor&)

' 指定のオブジェクトのタイプを取得
Declare Function Api_GetObjectType& Lib "gdi32" Alias "GetObjectType" (ByVal hgdibj&)

' 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
Declare Function Api_CreateCompatibleDC& Lib "gdi32" Alias "CreateCompatibleDC" (ByVal
hDC&)

' 指定された幅・高さ・色形式を持つビットマップを作成
Declare Function Api_CreateBitmap& Lib "gdi32" Alias "CreateBitmap" (ByVal nWidth&,
ByVal nHeight&, ByVal nPlanes&, ByVal nBitCount&, lpBits As Any)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

```

```

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたデバイスコンテキストを削除
Declare Function Api_DeleteDC& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define OBJ_BITMAP 7
#define OBJ_BRUSH 2
#define OBJ_DC 3
#define OBJ_ENHMETADC 12
#define OBJ_ENHMETAFILE 13
#define OBJ_EXTPEN 11
#define OBJ_FONT 6
#define OBJ_MEMDC 10
#define OBJ_METADC 4
#define OBJ_METAFILE 9
#define OBJ_PAL 5
#define OBJ_PEN 1
#define OBJ_REGION 8
#define vbYellow &H00FFFF

'ビットマップ
'ブラシ
'デバイスコンテキスト
'拡張メタファイルのデバイスコンテキスト
'拡張メタファイル
'拡張ペン
'フォント
'メモリデバイスコンテキスト
'メタファイルのデバイスコンテキスト
'メタファイル
'パレット
'ペン
'領域
'黄色のカラーコード

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var hDC As Long
    Var nDC As Long
    Var nBitmap As Long
    Var MyStr As String
    Var Ret As Long

    '親ウィンドウのデバイスコンテキストを取得
    hDC = Api_GetDC(GethWnd)

    'デバイスコンテキストを作成
    nDC = Api_CreateCompatibleDC(hDC)

    'ビットマップを作成
    nBitmap = Api_CreateBitmap(10, 10, 1, 1, ByVal 0)

    'オブジェクトを選択
    Ret = Api_SelectObject(nDC, nBitmap)

    'オブジェクトタイプを取得
    Select Case Api_GetObjectType(nBitmap)
        Case OBJ_BITMAP
            MyStr = "Object Type: Bitmap"
        Case OBJ_BRUSH
            MyStr = "Object Type: Brush"
        Case OBJ_FONT
            MyStr = "Object Type: Font"
        Case OBJ_PAL
            MyStr = "Object Type: Pal"
        Case OBJ_PEN
            MyStr = "Object Type: Pen"
    End Select
End Sub

```

```

Case OBJ_EXTPEN
    MyStr = "Object Type: ExtPen"
Case OBJ_REGION
    MyStr = "Object Type: Region"
Case OBJ_DC
    MyStr = "Object Type: Device Context"
Case OBJ_MEMDC
    MyStr = "Object Type: Memory Device Context"
Case OBJ_METAFILE
    MyStr = "Object Type: Metafile"
Case OBJ_METADC
    MyStr = "Object Type: Metafile DC"
Case OBJ_ENHMETAFILE
    MyStr = "Object Type: Enhanced Metafile"
Case OBJ_ENHMETADC
    MyStr = "Object Type: Enhanced Metafile DC"
End Select

```

'オブジェクトタイプを表示

```
Text1.SetWindowText MyStr
```

'最も近いカラー値を取得

```
Text2.SetWindowText "Nearest Color: &H" & Hex$(Api_GetNearestColor(nDC, vbYellow))
```

'解放

```
Ret = Api_DeleteDC(nDC)
```

```
Ret = Api_DeleteObject(nBitmap)
```

```
Ret = Api_ReleaseDC(GetHwnd, hDC)
```

```
End Sub
```

```

'=====
'=
'=====

```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

表示できる色に最も近いカラー値を取得 (II)

GetPixel 指定された座標のピクセルのRGB値を取得

GetNearestColor 指定のカラー値に対して、デバイスが表示できる色に最も近いカラー値を取得

CreateCompatibleDC 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成

DeleteDC 指定されたデバイスコンテキストを削除

GetDC 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放

「実行」ボタンクリックで24ビットの画像をトレースし、そのGetNearestColorの値をPicture2に(1ビット)描画しています。

Picture1上のマウス指定位置のRGB値と、その時のGetNearestColor値を表示させています。



```

'=====
'= 表示できる色に最も近いカラー値を取得 (11)
'= (GetNearestColor2.bas)
'=====
#include "Windows.bi"

' 指定された座標のピクセルのRGB値を取得
Declare Function Api_GetPixel& Lib "gdi32" Alias "GetPixel" (ByVal hDC&, ByVal X&, ByVal Y&)

' 指定のカラー値に対して、デバイスが表示できる色に最も近いカラー値を取得
Declare Function Api_GetNearestColor& Lib "gdi32" Alias "GetNearestColor" (ByVal hDC&, ByVal crColor&)

' 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
Declare Function Api_CreateCompatibleDC& Lib "gdi32" Alias "CreateCompatibleDC" (ByVal hDC&)

' 指定されたデバイスコンテキストを削除
Declare Function Api_DeleteDC& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared MainForm As Object
Var Shared Text(3) As Object
Var Shared Picture(1) As Object
Var Shared Button1 As Object
Var Shared Bitmap As Object
BitmapObject Bitmap

MainForm.Attach GethWnd
For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    If i < 2 Then
        Text(i).SetFontSize 14
        Picture(i).Attach GetDlgItem("Picture" & Trim$(Str$(i + 1)))
    Else
        Text(i).SetFontSize 12
    End If
Next i
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared hDC As Long

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Bitmap.LoadFile "flower.bmp"
    Picture(0).DrawBitmap Bitmap , 0, 0
    Bitmap.DeleteObject

    'Picture1のデバイスコンテキストを取得
    hDC = Api_GetDC(Picture(0).GethWnd)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var DotCol As String
    Var lColor As Long
    Var rgbR As Integer
    Var rgbG As Integer

```

```

Var rgbB As Integer
Var nDC As Long
Var x As Integer
Var y As Integer
Var Ret As Long

Picture(1).Cls

'デバイスコンテキストを作成
nDC = Api_CreateCompatibleDC(hDC)

For x = 0 To Picture(0).GetWidth - 1
    For y = 0 To Picture(0).GetHeight - 1

        'マウス指定位置のRGB値を取得
        lColor = Api_GetPixel(hDC, x, y)

        rgbR = lColor And &HFF
        rgbG = (lColor And &HFF00) / 256
        rgbB = (lColor And &HFF0000) / 65536

        If Api_GetNearestColor(nDC, RGB(rgbR, rgbG, rgbB)) = &HFFFFFF Then
            Picture(1).Pset(x, y), 15
        Else
            Picture(1).Pset(x, y), 0
        End If
    Next y
Next x

'解放
Ret = Api_DeleteDC(nDC)
End Sub

'=====
'=
'=====
Declare Sub Picture1_MouseMove edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
Sub Picture1_MouseMove (ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single,
ByVal Y As Single)
    Var DotCol As String
    Var lColor As Long
    Var rgbR As Integer
    Var rgbG As Integer
    Var rgbB As Integer
    Var nDC As Long
    Var Ret As Long

    'デバイスコンテキストを作成
    nDC = Api_CreateCompatibleDC(hDC)

    'マウス指定位置のRGB値を取得
    lColor = Api_GetPixel(hDC, X, Y)

    'Picture1指定座標と同位置に赤のドットで描画
    Picture(1).Pset(X, Y), 5

    rgbR = lColor And &HFF
    rgbG = (lColor And &HFF00) / 256
    rgbB = (lColor And &HFF0000) / 65536

    'RGBを表示
    DotCol = "(" & Format$(rgbR, "###") & ", " & Format$(rgbG, "###") & ", " & Format$(rgbB,
"###") & ")"
    Text(0).SetWindowText DotCol

    '最も近いカラー値を取得
    Text(1).SetWindowText "&H" & Hex$(Api_GetNearestColor(nDC, RGB(rgbR, rgbG, rgbB)))

```

```

'解放
Ret = Api_DeleteDC (nDC)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    Ret = Api_ReleaseDC (Picture (0) .GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルからBitmapをロード

ファイルからBitmapをロードし、クリップボードに転送。ピクチャボックスに貼り付けてみます。

LoadImage 画像ファイルの読み込み
OpenClipboard クリップボードをオープン
CloseClipboard クリップボードをクローズ
EmptyClipboard クリップボードを空にする
SetClipboardData クリップボードにデータを設定
IsClipboardFormatAvailable 指定したフォーマットがクリップボードにあるかどうかを判断
GetClipboardData クリップボードから指定のフォーマットデータを検索
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
CreateCompatibleDC メモリデバイスコンテキストを作成
DeleteDC 指定されたデバイスコンテキストを削除
BitBlt ビットブロックの転送
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

LoadImageの第一引数にhInstを指定した場合は、リソースを読み込み、「0」を指定した場合はファイルから画像を読み込みます。

EditBoxにBitmapファイルをドラッグするとPictureBoxにBitmap画像が表示されます。



```

'=====
'= ファイルからBitmapをロード
'= (LoadImage.bas)
'=====
#include "Windows.bi"

#define LR_LOADFROMFILE &H10
#define IMAGE_BITMAP 0
#define IMAGE_CURSOR 2
#define IMAGE_ICON 1
#define IMAGE_ENHMETAFILE 3

'外部ファイルからロードする
'ビットマップ
'カーソル
'アイコン
'拡張メタファイル

```

```

#define CF_BITMAP 2                                'ビットマップのデータ (HBITMAP)
#define SRCCOPY &HCC0020                          'そのまま転送

' 画像ファイルの読み込み
Declare Function Api_LoadImage& Lib "user32" Alias "LoadImageA" (ByVal hInst&, ByVal
lpzName$, ByVal uType&, ByVal cxDesired&, ByVal cyDesired&, ByVal fuLoad&)

' クリップボードを開く
Declare Function Api_OpenClipboard& Lib "user32" Alias "OpenClipboard" (ByVal hWnd&)

' クリップボードを閉じる
Declare Function Api_CloseClipboard& Lib "user32" Alias "CloseClipboard" ()

' クリップボードを空にする
Declare Function Api_EmptyClipboard& Lib "user32" Alias "EmptyClipboard" ()

' クリップボードにデータを設定
Declare Function Api_SetClipboardData& Lib "user32" Alias "SetClipboardData" (ByVal
wFormat&, ByVal hMem&)

' 指定したフォーマットがクリップボードにあるかどうか判定
Declare Function Api_IsClipboardFormatAvailable& Lib "user32" Alias
"IsClipboardFormatAvailable" (ByVal wFormat&)

' クリップボードから指定フォーマットのデータを検索
Declare Function Api_GetClipboardData& Lib "user32" Alias "GetClipboardData" (ByVal
wFormat&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
Declare Function Api_CreateCompatibleDC& Lib "gdi32" Alias "CreateCompatibleDC" (ByVal
hDC&)

' 指定されたデバイスコンテキストを削除
Declare Function Api_DeleteDC& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)

' ビットブロック転送を行います。コピー元からコピー先のデバイスコンテキストへ、指定された長方形内の各ピクセル
の色データをコピーします。
Declare Function Api_BitBlt& Lib "gdi32" Alias "BitBlt" (ByVal hDestDC&, ByVal X&, ByVal
Y&, ByVal nWidth&, ByVal nHeight&, ByVal hSrcDC&, ByVal xSrc&, ByVal ySrc&, ByVal dwRop&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared hBitmap As Long
Var Shared phDC As Long
Var Shared FileName As String

Var Shared Picture1 As Object
Var Shared Edit1 As Object

Picture1.Attach GetDlgItem("Picture1")
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()

    'ピクチャボックスのデバイスコンテキストを取得
    phDC = Api_GetDC(Picture1.GetHwnd)
End Sub

```

```

'=====
'=
'=====
Declare Sub PicturePaste edec1 ()
Sub PicturePaste ()
    Var hBuf As Long
    Var Ret As Long

    'ファイルからBitmapをロード
    hBitmap = Api_LoadImage(0, FileName, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE)

    If hBitmap = 0 Then
        A% = MsgBox(GetWindowText, "ビットマップファイルをロードできません！", 0, 2)
        Exit Sub
    End If

    'クリップボードをオープン
    Ret = Api_OpenClipboard(GetHwnd)

    'クリップボードを初期化
    Ret = Api_EmptyClipboard()

    'クリップボードにBitmapをロード
    Ret = Api_SetClipboardData(CF_BITMAP, hBitmap)

    'クリップボードにBitmapが存在するかどうかを判断
    If Api_IsClipboardFormatAvailable(CF_BITMAP) = 0 Then
        A% = MsgBox(GetWindowText, "クリップボードにBitmapがありません！", 0, 2)
    End If

    'クリップボードをクローズ
    Ret = Api_CloseClipboard()

    Picture1.Cls

    'クリップボードに指定したフォーマットのデータがあるかどうかを調べる
    Ret = Api_GetClipboardData(CF_BITMAP)

    'メモリデバイスコンテキストを作成
    hBuf = Api_CreateCompatibleDC(Ret)

    'オブジェクトを選択
    Ret = Api_SelectObject(hBuf, hBitmap)

    'ビットブロックの転送
    Ret = Api_BitBlt(phDC, 0, 0, Picture1.GetWidth, Picture1.GetHeight, hBuf, 0, 0,
SRCOPY)

    'デバイスコンテキストを削除
    Ret = Api_DeleteDC(hBuf)
End Sub

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
Sub Edit1_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount(DF)
    FileName = GetDropFileName(DF, 0)
    Edit1.SetWindowText FileName
    PicturePaste
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()

```



```

Var Ret As Long

Ret = Api_ReleaseDC(Picture1.GethWnd, phDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイル検索とコピー

例ではC:\Tempフォルダ内の指定した拡張子を持つファイルを検索し、C:\Temp\Testフォルダへコピーします。フォルダが存在しない場合は作成してコピーします。

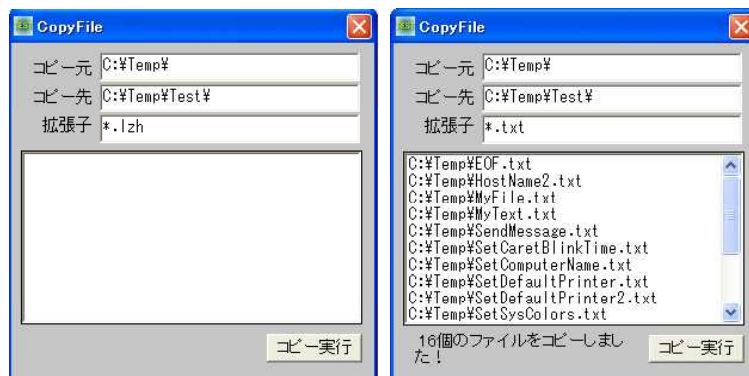
CreateDirectory ディレクトリの新規作成

CopyFile ファイルのコピー

FindFirstFile 指定したファイル名に一致するファイルやディレクトリを検索

FindNextFile FindFirstFile() 関数で検出したファイルの次を検出

FindClose ファイル検索ハンドルをクローズ



Tempにtestフォルダが作成され、ファイルがコピーされた状態を示しています。



```

'=====
'= ファイルコピー
'= (CopyFile.bas)
'=====
#include "Windows.bi"

```

```

#define INVALID_HANDLE_VALUE -1
#define MAX_PATH 260

```

```

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

```

```

Type WIN32_FIND_DATA

```

```

        dwFileAttributes As Long
        ftCreationTime    As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime  As FILETIME
        nFileSizeHigh    As Long
        nFileSizeLow     As Long
        dwReserved0      As Long
        dwReserved1      As Long
        cFileName        As String * MAX_PATH
        cAlternate        As String * 14
End Type

Type SECURITY_ATTRIBUTES
    nLength          As Long
    lpSecurityDescriptor As Long
    bInheritHandle   As Long
End Type

' ディレクトリの新規作成
Declare Function Api_CreateDirectory& Lib "kernel32" Alias "CreateDirectoryA" (ByVal lpPathName$, lpSecurityAttributes As SECURITY_ATTRIBUTES)

' ファイルコピー
Declare Function Api_CopyFile& Lib "kernel32" Alias "CopyFileA" (ByVal lpExistingFileName$, ByVal lpNewFileName$, ByVal bFailIfExists&)

' 指定したファイル名に一致するファイルやディレクトリを検索
Declare Function Api_FindFirstFile& Lib "kernel32" Alias "FindFirstFileA" (ByVal lpFileName$, lpFindFileData As WIN32_FIND_DATA)

' FindFirstFile() 関数で検出したファイルの次を検出
Declare Function Api_FindNextFile& Lib "kernel32" Alias "FindNextFileA" (ByVal hFindFile&, lpFindFileData As WIN32_FIND_DATA)

' ファイル検索ハンドルをクローズ
Declare Function Api_FindClose& Lib "kernel32" Alias "FindClose" (ByVal hFindFile&)

Var Shared List1 As Object
Var Shared Edit(2) As Object
Var Shared Text(3) As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
For i = 0 To 3
    If i < 3 Then Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) :
Edit(i).SetFontSize 14
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
Next

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit(0).SetWindowText "C:¥Temp¥"
    Edit(1).SetWindowText "C:¥Temp¥Test¥"
    Edit(2).SetWindowText "*.lzh"
End Sub

' =====
' =
' =====
Declare Function CopyFiles(sSourcePath As String, sDestination As String, sFiles As String) As Long
Function CopyFiles(sSourcePath As String, sDestination As String, sFiles As String) As Long
    Var WFD As WIN32_FIND_DATA
    Var SA As SECURITY_ATTRIBUTES
    Var r As Long
    Var Ret As Long
    Var hFile As Long

```

```

Var bNext As Long
Var copied As Long
Var currFile As String

'フォルダが存在しない場合は作成する
Ret = Api_CreateDirectory(sDestination, SA)

'目的フォルダ内を検索
hFile = Api_FindFirstFile(sSourcePath & sFiles, WFD)

'ファイルが見あたらない場合
If hFile = INVALID_HANDLE_VALUE Then
    Text(3).SetWindowText sFiles & " は、ありません！"
    Beep
    Wait 100
    Exit Function
End If

'新しいディレクトリにファイルをコピー
If hFile Then
    Do
        'ファイル名を取得
        currFile = Left$(WFD.cFileName, InStr(WFD.cFileName, Chr$(0)))

        'ファイルコピー開始
        Ret = Api_CopyFile(sSourcePath & currFile, sDestination & currFile, False)
        copied = copied + 1

        'コピーしたファイルのチェック
        List1.AddString sSourcePath & currFile

        '次のファイルを検索
        bNext = Api_FindNextFile(hFile, WFD)
    Loop until bNext = 0
End If

'クローズ
Ret = Api_FindClose(hFile)

'コピーされたファイル数
CopyFiles = copied
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sSourcePath As String
    Var sDestination As String
    Var sFiles As String
    Var numCopied As Long

    '元フォルダ・複写先フォルダ・複写するファイルの種類
    sSourcePath = Edit(0).GetWindowText
    sDestination = Edit(1).GetWindowText
    sFiles = Edit(2).GetWindowText

    List1.ResetContent

    'コピーしたファイルをカウント
    numCopied = CopyFiles(sSourcePath, sDestination, sFiles)

    Text(3).SetWindowText Str$(numCopied) & "個のファイルをコピーしました！"
End Sub

'=====
'=
'=====
Declare Sub Edit3_SetFocus edec1 ()

```

```

Sub Edit3_SetFocus()
    List1.ResetContent
    Text(3).SetWindowText ""
End Sub

'=====
' =
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルサイズの取得(1)

ファイルサイズを取得します。

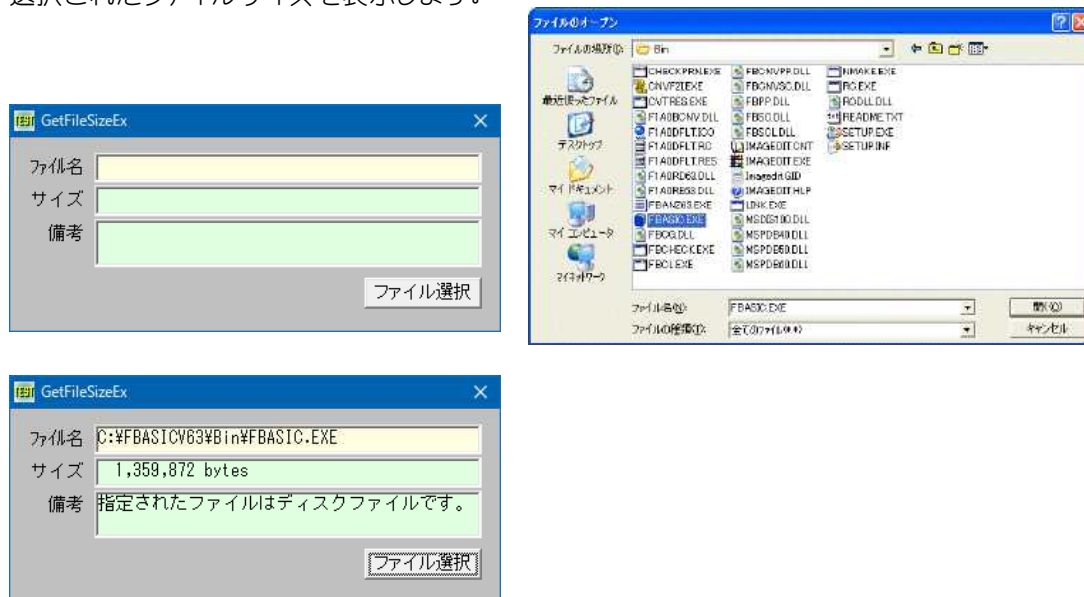
GetFileType オープンしているファイルのタイプを取得

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す

GetFileSizeEx ファイルサイズの取得 (64bit関数であり、F-BASICにはCurrencyがないので、
ULARGE_INTEGER構造体によりHigh、Lowに分けて取り出します。)

CloseHandle 開かれているオブジェクトハンドルを閉じる

ファイル選択ボタンでファイルのオープンダイアログを呼び出し、ファイルを選択します。
選択されたファイルサイズを表示します。



```

'=====
' = ファイルサイズの取得
' = (GetFileSizeEx.bas)
'=====
#include "Windows.bi"

```

```

Type ULARGE_INTEGER
    LowPart As Long
    HighPart As Long
End Type

```

' オープンしているファイルのタイプを取得

```
Declare Function Api_GetFileType& Lib "kernel32" Alias "GetFileType" (ByVal hFile&)
```

' 指定したファイルをオープンし、デバイスハンドルを返す

```
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal
dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)
```

```

' ファイルサイズの取得
Declare Function Api_GetFileSizeEx Lib "kernel32" Alias "GetFileSizeEx" (ByVal hFile&,
lpFileSize As ULARGE_INTEGER)

' 開かれているオブジェクトハンドルを閉じる
Declare Function Api_CloseHandle Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

#define GENERIC_READ -2147483648          '&H80000000
#define FILE_SHARE_READ &H1             '後続のオープン操作で読み取りアクセスが要求された場合、そのオープンを許可
#define FILE_Type_CHAR &H2              'キャラクタ型デバイス
#define FILE_Type_DISK &H1              'ディスクファイル
#define FILE_Type_PIPE &H3              'パイプ
#define FILE_Type_UNKNOWN &H0           '不明
#define OPEN_EXISTING 3                  'ファイルをオープンする

Var Shared Text(5) As Object
For i = 0 To 5
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
Next

Var Shared sFile As String

'=====
'=
'=====
Declare Sub FileSizeGet edec1 ()
Sub FileSizeGet ()
    Var hFile As Long
    Var nSize As ULARGE_INTEGER
    Var sSave As String
    Var Ret As Long

    'ファイルを開く
    hFile = Api_CreateFile(sFile, GENERIC_READ, FILE_SHARE_READ, ByVal 0&,
OPEN_EXISTING, ByVal 0&, ByVal 0&)

    'ファイルサイズ取得
    Ret = Api_GetFileSizeEx(hFile, nSize)

    'ファイルの種類を検索
    Select Case Api_GetFileType(hFile)
        Case FILE_Type_UNKNOWN
            sSave = "指定されたファイルのタイプは未知です。"
        Case FILE_Type_DISK
            sSave = "指定されたファイルはディスクファイルです。"
        Case FILE_Type_CHAR
            sSave = "指定されたファイルは、キャラクタファイル、通常LPTデバイスまたはコンソールです。"
        Case FILE_Type_PIPE
            sSave = "指定されたファイルは命名されたか匿名のパイプです。"
    End Select

    'ファイルを閉じる
    Ret = Api_CloseHandle(hFile)

    Text(4).SetWindowText Format$(nSize.HighPart * 2 ^ 32 + nSize.LowPart) / (1000 ^ 2) *
1000000, "###,###,###" & " bytes"
    Text(5).SetWindowText sSave
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    For i = 3 To 5
        Text(i).SetWindowText ""
    Next

    sFile = WinOpenDlg("ファイルのオープン", "C:¥*.*", "全てのファイル(*)", 0)

```

```

        If sFile <> Chr$(&H1B) Then
            Text(3).SetWindowText sFile
            FileSizeGet
        End If
    End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

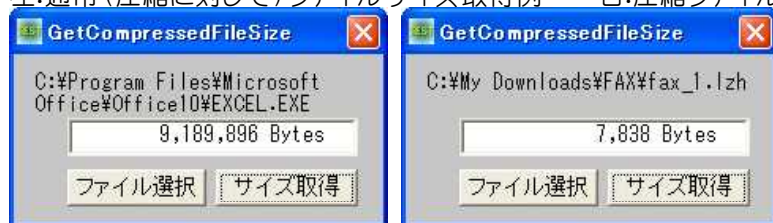
ファイルサイズを取得 (II)

ファイルサイズを取得します。

GetCompressedFileSize ファイルサイズを取得

例は、選択したファイルのサイズを取得します。

左:通常 (圧縮に対して) ファイルサイズ取得例 右:圧縮ファイルのサイズ取得例



```

'=====
'= ファイルサイズを取得
'= (GetCompressedFileSize.bas)
'=====
#include "Windows.bi"

' ファイルサイズを取得
Declare Function Api_GetCompressedFileSize& Lib "kernel32" Alias
"GetCompressedFileSizeA" (ByVal lpFileName$, lpFileSizeHigh&)

Var Shared Text1 As Object
Var Shared Text2 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14

Var Shared FileName As String

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Text1.SetWindowText ""
    Text2.SetWindowText ""

    FileName = WinOpenDlg("ファイルのオープン", "C:¥*.*", "全てのファイル (*.*)", 0)
    If FileName <> Chr$(&H1B) Then
        Text1.SetWindowText FileName
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()

```

```

Sub Button2_on()
    Var Ret As Long

    Ret = Api_GetCompressedFileSize(FileName, ByVal 0)

    Text2.SetWindowText Format$(Ret, " ###,###,###,### Bytes")
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

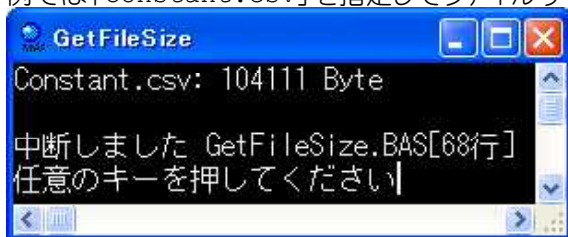
```

ファイルサイズの取得(Ⅲ)

ファイルサイズを取得します。

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す
GetFileSize ファイルのサイズをバイト単位で取得
CloseHandle オープンされているオブジェクトハンドルをクローズ

例では「Constant.csv」を指定してファイルサイズを取得しています。



```

'=====
'= ファイルサイズの取得(Ⅲ)
'= (GetFileSize.bas)
'=====

' セキュリティを定義する構造体
Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type

' 指定したファイルをオープンし、デバイスハンドルを返す
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As
SECURITY_ATTRIBUTES, ByVal dwCreationDistribution&, ByVal dwFlagsAttributes&, ByVal
hTemplateFile&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

' ファイルのサイズをバイト単位で取得
Declare Function Api_GetFileSize& Lib "kernel32" Alias "GetFileSize" (ByVal hFile&,
lpFileSizeHigh&)

#define INVALID_HANDLE_VALUE -1
#define GENERIC_READ -2147483648
#define OPEN_EXISTING 3

'見つからない場合
'読み込みモード(&H80000000)
'ファイルをオープンする(存在しない場合失敗)

Var FileName As String
Var hFileName As Long
Var sa As SECURITY_ATTRIBUTES

```

```

Var FileSizeLow As Long
Var FileSizeHigh As Long
Var Ret As Long

'ファイル名を指定
FileName = "Constant.csv"

'セキュリティ構造体を初期化
sa.nLength = len(sa)

'ファイルハンドル取得
hFileName = Api_CreateFile(FileName, GENERIC_READ, 0, sa, OPEN_EXISTING, 0, 0)

'ファイルハンドルの値が有効(見つかったとき)
If hFileName <> INVALID_HANDLE_VALUE Then

    'ファイルサイズを取得
    FileSizeLow = Api_GetFileSize(hFileName, FileSizeHigh)

    'ファイルサイズの取得に失敗したときは
    If FileSizeLow = &HFFFFFFFF Then

        '拡張エラー情報を表示
        Print "取得に失敗しました。"
    Else

        'ファイルサイズを表示
        Print FileName ":" FileSizeLow "Byte"
    End If

    'ファイルをクローズ
    Ret = Api_CloseHandle(hFileName)

'有効でないとき
Else

    'エラーを表示
    Print "ファイルを開けませんでした。"
End If

Stop
End

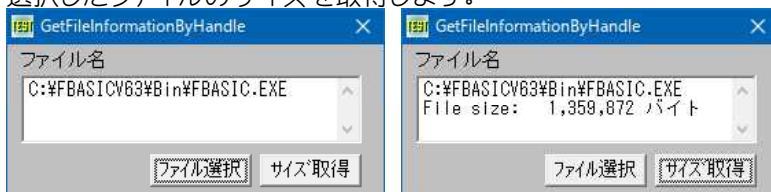
```

ファイルサイズの取得(Ⅳ)

ファイルサイズを取得します。

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す
GetFileInformationByHandle 指定されたファイルに関する情報を取得
CloseHandle オープンされているオブジェクトハンドルをクローズ

選択したファイルのサイズを取得します。



```

'=====
'= ファイルサイズの取得(Ⅲ)
'= (GetFileInformationByHandle.bas)
'=====
#include "Windows.bi"

#define OPEN_EXISTING 3

'ファイルをオープンする(存在しない場合失敗)

```



```

Type FILETIME
    dwLowDateTime    As Long
    dwHighDateTime   As Long
End Type

Type BY_HANDLE_FILE_INFORMATION
    dwFileAttributes    As Long
    ftCreationTime       As FILETIME
    ftLastAccessTime     As FILETIME
    ftLastWriteTime      As FILETIME
    dwVolumeSerialNumber As Long
    nFileSizeHigh        As Long
    nFileSizeLow         As Long
    nNumberOfLinks       As Long
    nFileIndexHigh       As Long
    nFileIndexLow        As Long
End Type

' 指定したファイルをオープンし、デバイスハンドルを返す
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal
dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' 指定されたファイルに関する情報を取得
Declare Function Api_GetFileInformationByHandle& Lib "kernel32" Alias
"GetFileInformationByHandle" (ByVal hFile&, lpFileInformation As
BY_HANDLE_FILE_INFORMATION)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

Var Shared FileName As String

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Edit1.SetWindowText ""

    FileName = WinOpenDlg("ファイルのオープン", "C:¥*.*", "全てのファイル (*.*)", 0)
    If FileName <> Chr$(&H1B) Then
        Edit1.SetWindowText FileName
    End If
End Sub

' =====
' =
' =====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var hFile As Long
    Var FileInfo As BY_HANDLE_FILE_INFORMATION
    Var Ret As Long

    ' ファイルをオープンし、デバイスハンドルを返す
    hFile = Api_CreateFile(FileName, 0, 0, ByVal 0, OPEN_EXISTING, 0, ByVal 0)

    ' ファイルに関する情報を取得
    Ret = Api_GetFileInformationByHandle(hFile, FileInfo)

```

'ハンドルをクローズ

```
Ret = Api_CloseHandle(hFile)
```

'結果表示

```
Edit1.SetWindowText FileName & Chr$(13, 10) & "File size: " &  
format$(FileInfo.nFileSizeLow, "###,###,### バイト")  
End Sub
```

```
'=====
```

```
'=
```

```
'=====
```

```
While 1  
    WaitEvent  
Wend  
Stop  
End
```

ファイル情報を取得

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す

GetFileInformationByHandle 指定されたファイルに関する情報を取得

CloseHandle オープンされているオブジェクトハンドルをクローズ

FileTimeToLocalFileTime 世界協定時刻形式ファイル時間をローカルファイル時間に変換

FileTimeToSystemTime ファイルタイムをシステムタイムに変換



```
'=====
```

```
'= ファイル情報を取得
```

```
'= (GetFileInformationByHandle2.bas)
```

```
'=====
```

```
#include "Windows.bi"
```

```
#define INVALID_HANDLE_VALUE -1  
#define GENERIC_READ -2147483648  
#define OPEN_EXISTING 3  
#define FILE_ATTRIBUTE_ARCHIVE &H20  
#define FILE_ATTRIBUTE_COMPRESSED &H800  
#define FILE_ATTRIBUTE_DIRECTORY &H10  
#define FILE_ATTRIBUTE_HIDDEN &H2  
#define FILE_ATTRIBUTE_NORMAL &H80  
#define FILE_ATTRIBUTE_READONLY &H1  
#define FILE_ATTRIBUTE_SYSTEM &H4  
#define FILE_ATTRIBUTE_TEMPORARY &H100
```

```
Type SECURITY_ATTRIBUTES  
    nLength As Long  
    lpSecurityDescriptor As Long  
    bInheritHandle As Long  
End Type
```

```
Type FILETIME  
    dwLowDateTime As Long  
    dwHighDateTime As Long  
End Type
```

```
Type SYSTEMTIME  
    wYear As Integer
```

'見つからない場合

'読み込みモード(&H80000000)

'ファイルをオープンする(存在しない場合失敗)

'アーカイブ属性を示す定数

'圧縮属性を示す定数

'ディレクトリ属性

'隠しファイル属性

'他のファイル属性を持たない

'読み込み専用属性

'システムファイル属性

'一時ファイル属性を示す定数

```

    wMonth           As Integer
    wDayOfWeek       As Integer
    wDay             As Integer
    wHour            As Integer
    wMinute          As Integer
    wSecond          As Integer
    wMilliseconds    As Integer
End Type

Type BY_HANDLE_FILE_INFORMATION
    dwFileAttributes As Long
    ftCreationTime   As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime  As FILETIME
    dwVolumeSerialNumber As Long
    nFileSizeHigh    As Long
    nFileSizeLow     As Long
    nNumberOfLinks   As Long
    nFileIndexHigh   As Long
    nFileIndexLow    As Long
End Type

' 指定したファイルをオープンし、デバイスハンドルを返す
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal
dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' 指定されたファイルに関する情報を取得
Declare Function Api_GetFileInformationByHandle& Lib "kernel32" Alias
"GetFileInformationByHandle" (ByVal hFile&, lpFileInformation As
BY_HANDLE_FILE_INFORMATION)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

' 世界協定時刻形式ファイル時間をローカルファイル時間に変換
Declare Function Api_FileTimeToLocalFileTime& Lib "kernel32" Alias
"FileTimeToLocalFileTime" (lpFileTime As FILETIME, lpLocalFileTime As FILETIME)

' ファイルタイムをシステムタイムに変換
Declare Function Api_FileTimeToSystemTime& Lib "kernel32" Alias "FileTimeToSystemTime"
(lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME)

Var Shared FileName As String

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared List1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14 : List1.SetWindowSize 290, 74
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Edit1.SetWindowText "ファイルをドラッグ&ドロップしてください"
End Sub

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var hFile As Long
    Var sa As SECURITY_ATTRIBUTES

```

```

Var bhfi As BY_HANDLE_FILE_INFORMATION
Var ft As FILETIME
Var st As SYSTEMTIME
Var txt As String
Var Ret As Long

'リストボックスを初期化
List1.ResetContent

'ファイル名を表示
Edit1.SetWindowText FileName

'セキュリティ構造体を初期化
sa.nLength = Len(sa)

'ファイルをオープン
hFile = Api_CreateFile(FileName, GENERIC_READ, 0, sa, OPEN_EXISTING, 0, 0)

'ファイルハンドルの値が有効であるとき
If hFile <> INVALID_HANDLE_VALUE Then

    'ファイルに関する情報を取得
    Ret = Api_GetFileInformationByHandle(hFile, bhfi)

    'ファイルに関する情報を表示
    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_READONLY Then
        List1.AddString "属性 : " & "ReadOnly"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_HIDDEN Then
        List1.AddString "属性 : " & "Hidden"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_SYSTEM Then
        List1.AddString "属性 : " & "System"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY Then
        List1.AddString "属性 : " & "Directory"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_ARCHIVE Then
        List1.AddString "属性 : " & "Archive"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_NORMAL Then
        List1.AddString "属性 : " & "Normal"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_TEMPORARY Then
        List1.AddString "属性 : " & "Temporary"
    End If

    If bhfi.dwFileAttributes And FILE_ATTRIBUTE_COMPRESSED Then
        List1.AddString "属性 : " & "Compressed"
    End If

    'ファイルサイズを表示
    List1.AddString "サイズ:" & Trim$(Format$(bhfi.nFileSizeLow, "#,###,###"))

    'ファイルの作成日時をローカルファイル日付へ変換
    Ret = Api_FileTimeToLocalFileTime(bhfi.ftCreationTime, ft)

    'ファイルの作成日時をシステムファイル日付へ変換
    Ret = Api_FileTimeToSystemTime(ft, st)

    'ファイルの作成日時を表示
    txt = "作成日:" & Trim$(Str$(st.wYear)) & "年" & Trim$(Str$(st.wMonth)) & "月" &
Trim$(Str$(st.wDay)) & "日" & "(" & KMid$("日月火水木金土", st.wDayOfWeek + 1, 1) & ")"
    txt = txt & Trim$(Str$(st.wHour)) & "時" & Trim$(Str$(st.wMinute)) & "分" &

```

```

Trim$(Str$(st.wSecond)) & "秒"
List1.AddString txt

' ファイルの更新日時をローカルファイル日付へ変換
Ret = Api_FileTimeToLocalFileTime(bhfi.ftLastWriteTime, ft)

' ファイルの更新日時をシステムファイル日付へ変換
Ret = Api_FileTimeToSystemTime(ft, st)

txt = "更新日:" & Trim$(Str$(st.wYear)) & "年" & Trim$(Str$(st.wMonth)) & "月" &
Trim$(Str$(st.wDay)) & "日" & "(" & KMid$("日月火水木金土", st.wDayOfWeek + 1, 1) & ")"
txt = txt & Trim$(Str$(st.wHour)) & "時" & Trim$(Str$(st.wMinute)) & "分" &
Trim$(Str$(st.wSecond)) & "秒"
List1.AddString txt

' ファイルのアクセス日時をローカルファイル日付へ変換
Ret = Api_FileTimeToLocalFileTime(bhfi.ftLastAccessTime, ft)

' ファイルのアクセス日時をシステムファイル日付へ変換
Ret = Api_FileTimeToSystemTime(ft, st)

txt = "アクセス日:" & Trim$(Str$(st.wYear)) & "年" & Trim$(Str$(st.wMonth)) & "月" &
Trim$(Str$(st.wDay)) & "日" & "(" & KMid$("日月火水木金土", st.wDayOfWeek + 1, 1) & ")"
txt = txt & Trim$(Str$(st.wHour)) & "時" & Trim$(Str$(st.wMinute)) & "分" &
Trim$(Str$(st.wSecond)) & "秒"
List1.AddString txt

' ファイルをクローズ
Ret = Api_CloseHandle(hFile)

' 有効でないとき
Else

' エラーを表示
List1.AddString "ファイルを開けませんでした。"
End If
End Sub

' =====
' = シェルドロップされたファイル名を取得
' =====
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
Sub Edit1_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount (DF)
    FileName = GetDropFileName (DF, 0)
    Edit1.SetWindowText FileName
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

ファイル所有者の取得

GetFileSecurity ファイルのセキュリティ情報取得

GetSecurityDescriptorOwner 指定されたセキュリティ記述子からオーナーを表すSID(セキュリティ識別子)を取得

LookupAccountSid SID(セキュリティ識別子)に対応する名前の取得

IsValidSid リビジョン番号が既知の範囲内にあるか、副権限の数が最大値以下であるかどうかを検証することにより、SID構造体を確認



```
'=====
'= ファイル所有者の取得
'= (GetFileSecurity.bas)
'=====
#include "Windows.bi"

' ファイルのセキュリティ情報取得
Declare Function Api_GetFileSecurity& Lib "advapi32" Alias "GetFileSecurityA" (ByVal
lpFileName$, ByVal RequestedInformation&, pSecurityDescriptor As Byte, ByVal nLength&,
lpnLengthNeeded&)

' 指定されたセキュリティ記述子からオーナーを表すSID (セキュリティ識別子) を取得
Declare Function Api_GetSecurityDescriptorOwner& Lib "advapi32" Alias
"GetSecurityDescriptorOwner" (pSecurityDescriptor As Any, pOwner&, lpbOwnerDefaulted&)

' SID (セキュリティ識別子) に対応する名前の取得
Declare Function Api_LookupAccountSid& Lib "advapi32" Alias "LookupAccountSidA" (ByVal
lpSystemName$, ByVal lpSid&, ByVal Name$, cbName&, ByVal ReferencedDomainName$,
cbReferencedDomainName&, peUse&)

' リビジョン番号が既知の範囲内にあるか、副権限の数が最大値以下であるかどうかを検証することにより、SID構
造体を確認
Declare Function Api_IsValidSid& Lib "advapi32" Alias "IsValidSid" (ByVal pSid&)

#define OWNER_SECURITY_INFORMATION &H1

Var Shared Edit1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared FileName As String

'=====
'=
'=====
Declare Function SidToString (ByVal Sid As Long) As String
Function SidToString (ByVal Sid As Long) As String
    Var cbAccountName As Long
    Var cbDomainName As Long
    Var strAccountName As String
    Var strDomainName As String
    Var ComputerName As String
    Var Use As Long
    Var Ret As Long

    If Api_IsValidSid(Sid) <> 0 Then
        Ret = Api_LookupAccountSid(ComputerName, Sid, ByVal 0, cbAccountName, ByVal 0,
cbDomainName, Use)
        strAccountName = Space$(cbAccountName + 1)
        strDomainName = Space$(cbDomainName + 1)
        Ret = Api_LookupAccountSid(ComputerName, Sid, strAccountName, cbAccountName,
strDomainName, cbDomainName, Use)

        If Ret <> 0 Then
            SidToString = Left$(strDomainName, cbDomainName) & "¥" &
Left$(strAccountName, cbAccountName)
        Else
            SidToString = "<UNKNOWN SID>"
        End If
    Else

```

```

        SidToString = "<INVALID SID>"
    End If
End Function

'=====
'=
'=====
Declare Function GetFileOwner(FullFileName As String) As String
Function GetFileOwner(FullFileName As String) As String
    Var Size As Long
    Var Sid As Long
    Var Dummy As Long
    Var Ret As Long

    Ret = Api_GetFileSecurity(FullFileName, OWNER_SECURITY_INFORMATION, ByVal 0, 0,
Size)

    If Size <> 0 Then
        Var SecurityDescr(Size - 1) As Byte

        If Api_GetFileSecurity(FullFileName, OWNER_SECURITY_INFORMATION, SecurityDescr
(0), Size, Size) <> 0 Then
            Ret = Api_GetSecurityDescriptorOwner(SecurityDescr(0), Sid, Dummy)
            If Sid <> 0 Then
                GetFileOwner = SidToString(Sid)
            Else
                A% = MsgBox(GetWindowText, "SIDを取得できません！", 0, 2)
            End If
        Else
            A% = MsgBox(GetWindowText, "ファイルセキュリティ情報を取得できません！", 0, 2)
        End If
    Else
        A% = MsgBox(GetWindowText, "ファイルセキュリティ情報を取得できません！", 0, 2)
    End If
End Function

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
Sub Edit1_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount (DF)
    FileName = GetDropFileName (DF, 0)
    Edit1.SetWindowText FileName
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    A% = MsgBox(GetWindowText, GetFileOwner(FileName), 0, 2)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

指定したファイルの属性を取得・設定を実行し確認します。

`GetFileAttributes` 属性を取得するAPI

`SetFileAttributes` 属性を設定するAPI

あらかじめ 同じフォルダに適当なファイル(例:`TEST.TXT`)を作成しておきます。起動するとファイル名と現在の属性を表示します。

コンボボックス内の目的行を選択しクリックするとその属性が変更され、変更された属性が表示されます。



APIでなければならないわけではありません。

`SETATTR` - ファイルの属性を設定と解除を行います。

SETATTR ファイルディスクリプタ, 属性

ファイルディスクリプタ → ファイル名

属性 → 文字列で設定

"+R" 読取専用属性を設定

"+A" アーカイブ属性を設定

"-R" 読取専用属性を解除

"-A" アーカイブ属性を解除

例:

```
#include "File.bi"
SETATTR "C:\Temp\test.txt", "+R"
```

```
'=====
'= ファイルの属性取得と変更の確認
'= (GetFileAttributes.bas)
'=====
#include "Windows.bi"
```

' 指定されたファイルまたはディレクトリの属性を取得

```
Declare Function Api_GetFileAttributes& Lib "Kernel32" Alias "GetFileAttributesA" (ByVal lpFileName$)
```

' ファイルの属性を変更

```
Declare Function Api_SetFileAttributes& Lib "Kernel32" Alias "SetFileAttributesA" (ByVal lpFileName$, ByVal dwFileAttributes&)
```

```
#define FILE_ATTRIBUTE_ARCHIVE &H20
#define FILE_ATTRIBUTE_DIRECTORY &H10
#define FILE_ATTRIBUTE_HIDDEN &H2
#define FILE_ATTRIBUTE_NORMAL &H80
#define FILE_ATTRIBUTE_READONLY &H1
#define FILE_ATTRIBUTE_SYSTEM &H4
```

'アーカイブ属性
'ディレクトリ属性
'隠しファイル属性
'ファイル属性を持たない
'読み込み専用属性
'システムファイル属性

```
#define FILE_ATTRIBUTE_COMPRESSED &H800
#define FILE_ATTRIBUTE_TEMPORARY &H100
```

'圧縮属性を示す定数
'一時ファイル属性を示す定数

```
Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Combo1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
```

```
Var Shared nFile As String
Var Shared WK$(5,1) As String
```



```

'=====
'=
'=====
Declare Sub Attr_Dsp edec1 ()
Sub Attr_Dsp()
    Var Ret As Long

    Text1.SetWindowText nFile

    Ret = Api_GetFileAttributes(nFile)

    Select Case Ret
        Case FILE_ATTRIBUTE_ARCHIVE
            Text2.SetWindowText "アーカイブ属性"
        Case FILE_ATTRIBUTE_DIRECTORY
            Text2.SetWindowText "ディレクトリ属性"
        Case FILE_ATTRIBUTE_HIDDEN
            Text2.SetWindowText "隠しファイル属性"
        Case FILE_ATTRIBUTE_NORMAL
            Text2.SetWindowText "ファイル属性持たない"
        Case FILE_ATTRIBUTE_READONLY
            Text2.SetWindowText "読み込み専用属性"
        Case FILE_ATTRIBUTE_SYSTEM
            Text2.SetWindowText "システムファイル属性"
    End Select
End Sub

'=====
'=
'=====
declare sub MainForm_Start edec1 ()
sub MainForm_Start()
    nFile = "test.txt"

    For i = 0 To 5
        For j = 0 To 1
            Read WK$(i, j)
        Next
        Combo1.AddString WK$(i, 0)
    Next

    data FILE_ATTRIBUTE_ARCHIVE , "&h20"
    data FILE_ATTRIBUTE_DIRECTORY , "&h10"
    data FILE_ATTRIBUTE_HIDDEN , "&h2"
    data FILE_ATTRIBUTE_NORMAL , "&h80"
    data FILE_ATTRIBUTE_READONLY , "&h1"
    data FILE_ATTRIBUTE_SYSTEM , "&h4"
    Attr_Dsp
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    Var NO As Long
    Var Ret As Long

    NO = Combo1.GetCursel

    Ret = Api_SetFileAttributes(nFile, Val(WK$(NO, 1)))
    Attr_Dsp
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend

```

Stop
End

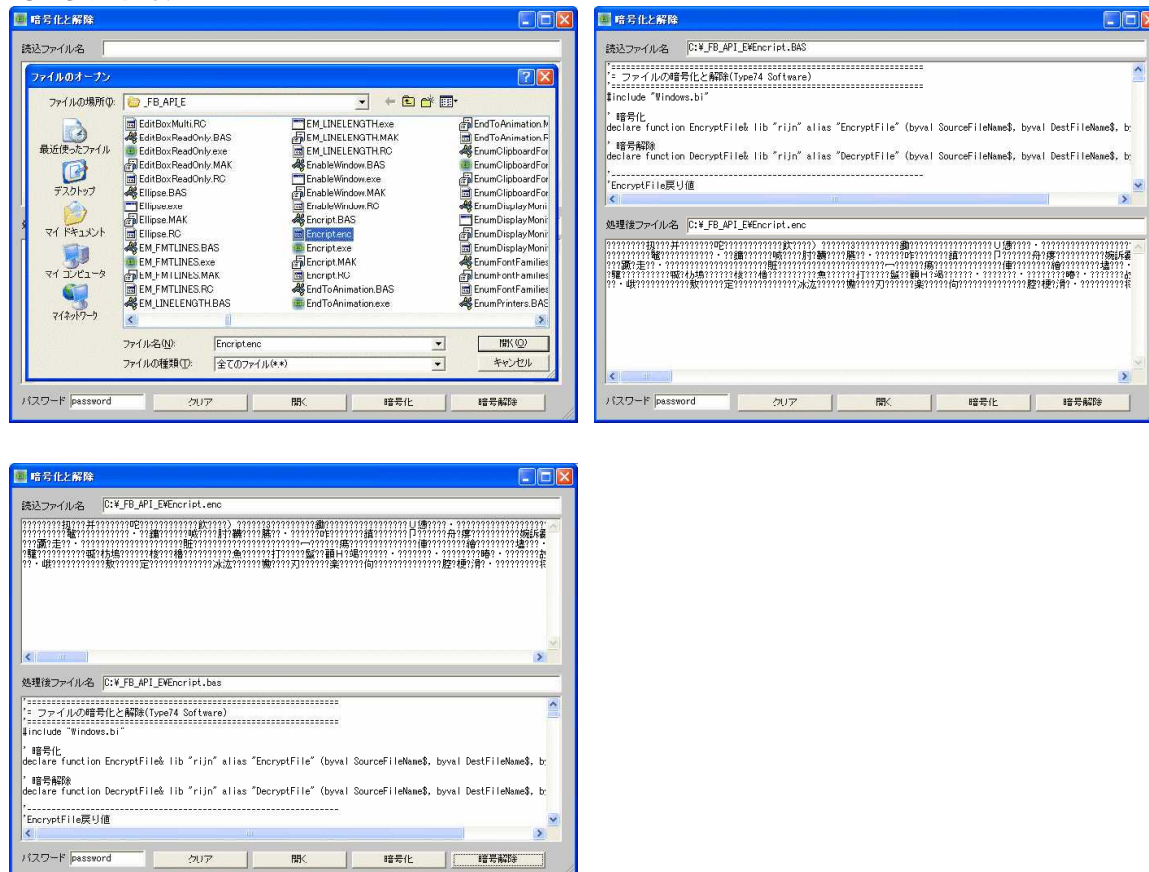
ファイルの暗号化と解除(1)

フリーソフトの**ラインドール(Rijn.dll)**を使用して、ファイルの暗号化および解除をします。

EncryptFile ファイルの暗号化

DecryptFile ファイルの暗号を解除

テスト例では、読み込んだファイルを暗号化し、拡張子を `enc` に変更しバイナリ保存しています。
また、暗号化されたファイルを読み込み、解除後ファイル名を指定し保存しています。
EditBoxには、そのまま表示させていますので、テキスト形式以外は正常に表示できません。(エラー処理部は入れておりません)



参考

<http://type74.org/>

ハッシュ、MD5、暗号などでWeb検索中、**ラインドール(Rijn.dll)**を見つけました。上記URLからダウンロードできます。

```
'=====
'= ファイルの暗号化と解除 (Type74 Software)
'= (Encrypt.bas)
'=====
#include "Windows.bi"

' 暗号化
Declare Function EncryptFile& Lib "rijn" Alias "EncryptFile" (ByVal SourceFileName$,
ByVal DestFileName$, ByVal key$)

' 暗号解除
Declare Function DecryptFile& Lib "rijn" Alias "DecryptFile" (ByVal SourceFileName$,
ByVal DestFileName$, ByVal key$)

'-----
'EncryptFile戻り値
' 0 正常終了
'-1 暗号化元ファイルオープン時エラー
'-2 暗号化先ファイル作成時エラー
```

```

'-3      暗号化時エラー

'DecryptFile戻り値
' 0      正常終了
'-1      復号元ファイルオープン時エラー
'-2      復号先ファイル作成時エラー
'-3      復号時エラー
'-----

' パス名の拡張子だけを変更
Declare Function Api_PathRenameExtension& Lib "shlwapi" Alias "PathRenameExtensionA"
(ByVal pszPath$, ByVal pszExt$)

' マウスのキャプチャを解放
Declare Function Api_ReleaseCapture& Lib "user32" Alias "ReleaseCapture" ()

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

' 表示要素の寸法とシステム構成の設定を取得
Declare Function Api_GetSystemMetrics& Lib "user32" Alias "GetSystemMetrics" (ByVal
nIndex&)

#define WM_NCLBUTTONDOWN &H41      '非クライアント領域で左マウスボタンを押す
#define HTBOTTOMRIGHT 17           'ウィンドウ境界の右下隅
#define SM_CXFRAME 32              'サイズ可変ウィンドウの境界線のx方向の幅
#define SM_CYFRAME 33              'サイズ可変ウィンドウの境界線のy方向の幅
#define SM_CYSIZE 31               'タイトルバー内のビットマップの高さ
#define SM_CYBORDER 6              'サイズ固定ウィンドウの境界線のy方向の幅

#define vbCrLf (Chr$(13) & Chr$(10)) 'キャリッジリターンとラインフィード(¥r¥n)

Var Shared Edit(4) As Object
Var Shared Text(3) As Object
Var Shared Button(3) As Object

For i = 0 To 4
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
    Edit(i).SetFontSize 12
Next
For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    If i < 2 Then
        Text(i).SetFontSize 12
    End If
Next
For i = 0 To 3
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1)))
    Button(i).SetFontSize 12
Next

Var Shared sFname As String      '元ファイル名
Var Shared dFname As String      '新ファイル名
Var Shared kChar As String       'パスワード
Var Shared zX As Integer         'サイズグリップx軸
Var Shared zY As Integer         'サイズグリップy軸

'=====
'= サイズグリップ位置計算
'=====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var fs As Integer

    fs = 18
    Text(3).SetFontSize fs
    Text(3).SetWindowSize fs, fs
    Text(3).SetFontName "Marlett"
    Text(3).SetWindowText "o"

```

```

ShowWindow -1

zX = Api_GetSystemMetrics(SM_CXFRAME) * 2 + 1
zY = Api_GetSystemMetrics(SM_CYFRAME) * 2 + Api_GetSystemMetrics(SM_CYBORDER) +
Api_GetSystemMetrics(SM_CYSIZE) + 1
End Sub

'=====
'= サイズグリップ処理
'=====
Declare Sub MainForm_MouseDown edec1 (ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
Sub MainForm_MouseDown (ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
    Var Ret As Long

    If Button% = 1 Then
        Ret = Api_ReleaseCapture()
        Ret = Api_SendMessage(Text(3).GethWnd, WM_NCLBUTTONDOWN, HTBOTTOMRIGHT, 0)
    End If
End Sub

'=====
'= リサイズ
'=====
Declare Sub MainForm_Resize edec1 ()
Sub MainForm_Resize ()

    '最小基本サイズに戻す
    If GetWidth < 700 Or GetHeight < 512 Then
        SetWindowSize 700, 512
    End If

    Text(0).MoveWindow 12, 16
    Text(1).MoveWindow 12, (GetHeight - 78) / 2 + 15
    Text(2).MoveWindow 12, GetHeight - 64

    Edit(0).MoveWindow 10, 36
    Edit(0).SetWindowSize GetWidth - 34, (GetHeight - 78) / 2 - 37

    Edit(1).MoveWindow 10, (GetHeight - 78) / 2 + 11 + 26
    Edit(1).SetWindowSize GetWidth - 34, (GetHeight - 78) / 2 - 37

    Edit(2).MoveWindow 110, 10
    Edit(2).SetWindowSize GetWidth - 134, 22

    Edit(3).MoveWindow 110, (GetHeight - 78) / 2 + 11
    Edit(3).SetWindowSize GetWidth - 134, 22

    Edit(4).MoveWindow 70, GetHeight - 68

    Button(0).MoveWindow 174, GetHeight - 66
    Button(1).MoveWindow 296, GetHeight - 66
    Button(2).MoveWindow 418, GetHeight - 66
    Button(3).MoveWindow 540, GetHeight - 66

    'サイズグリップ(右下)
    Text(3).MoveWindow (GetWidth - Text(3).GetWidth) - zX, GetHeight - Text(3).GetHeight
- zY
End Sub

'=====
'= EditTextクリア
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    For i = 0 To 3
        Edit(i).SetWindowText ""
    Next
End Sub

```

```

'=====
'= ファイル読込
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var hFile As Byte
    Var Buff As String
    Var Temp As String
    Var Ret As Long
    Button1_on

    sFName = WinOpenDlg("ファイルのオープン", "*.*", "全てのファイル (*.*)", 0)
    If sFName <> Chr$(&H1B) Then
        Edit(2).SetWindowText sFName
        hFile = FreeFile
        open sFName For BinInp As hFile
        Buff = Space$(Lof(hFile))
        FRead hFile, Buff
        Buff = JConv$(Buff, 0, 2)
        Edit(0).SetWindowText Buff
        Close hFile

        '拡張子をhtmに変換
        dFName = sFName & String$(260, Chr$(0))
        Ret = Api_PathRenameExtension(dFName, ".enc")
        Edit(3).SetWindowText dFName
    Else
        sFName = ""
    End If
End Sub

```

```

'=====
'= 暗号化
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on()
    Var hFile As Byte
    Var Buff As String
    Var Ret As Long

    dFName = Edit(3).GetWindowText
    kChar = Edit(4).GetWindowText

    If dFName = "" Then
        A% = MsgBox("", "ファイル名を設定してください！", 0, 2)
        Exit Sub
    End If

    Ret = EncryptFile(sFName, dFName, kChar)

    hFile = FreeFile
    Open dFName For BinInp As hFile
    Buff = Space$(Lof(hFile))
    FRead hFile, Buff
    Buff = JConv$(Buff, 0, 2)
    Edit(1).SetWindowText Buff
    Close hFile
End Sub

```

```

'=====
'= 暗号解除
'=====
Declare Sub Button4_on edec1 ()
Sub Button4_on()
    Var hFile As Byte
    Var Buff As String
    Var Ret As Long

    dFName = Edit(3).GetWindowText
    kChar = Edit(4).GetWindowText

```

```

If dFName = "" Then
    A% = MsgBox(" ", "ファイル名を設定してください！", 0, 2)
    Exit Sub
End If

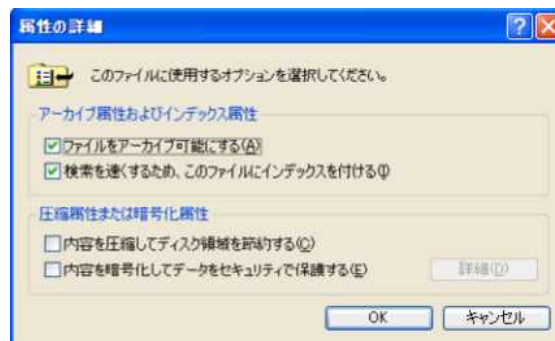
Ret = DecryptFile(sFName, dFName, kChar)

hFile = FreeFile
Open dFName For BinInp As hFile
Buff = Space$(Lof(hFile))
FRead hFile, Buff
Buff = JConv$(Buff, 0, 2)
Edit(1).SetWindowText Buff
Close hFile
End Sub
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルの暗号化と解除 (II)

EncryptFile ファイルまたはディレクトリを暗号化する
DecryptFile 暗号化されたファイルまたはディレクトリを解読



他のユーザーが暗号化されたファイルにアクセスしようとした場合アクセスが拒否されます。



```

'=====
'= ファイルの暗号化と解読
'= (EncryptFile.bas)
'=====
#include "Windows.bi"

```

' ファイルまたはディレクトリを暗号化する。ファイルのデータストリームはすべて暗号化。暗号化されたディレクトリ内で新しく作成するファイルはすべて暗号化される

```

Declare Function Api_EncryptFile& Lib "advapi32" Alias "EncryptFileA" (ByVal lpFileName$)

```

' 暗号化されたファイルまたはディレクトリを解読

```

Declare Function Api_DecryptFile& Lib "advapi32" Alias "DecryptFileA" (ByVal lpFileName$, ByVal dwReserved&)

```

```

Var Shared Edit1 As Object

```

```

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

Var Shared FileName As String

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    'ファイルが選択されているときは
    If Len(FileName) Then

        '対象のファイル名を指定
        FileName = Edit1.GetWindowText

    'ファイルが選択されていないときは
    Else
        Text2.SetWindowText "ファイルを選択してください！"
        Exit Sub
    End If

    '暗号化を設定
    Ret = Api_EncryptFile(FileName)

    '設定できたときは
    If Ret <> 0 Then
        Text2.SetWindowText "設定しました。"
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    'ファイルが選択されているときは
    If Len(FileName) Then

        '対象のファイル名を指定
        FileName = Edit1.GetWindowText

    'ファイルが選択されていないときは
    Else
        Text2.SetWindowText "ファイルを選択してください！"
        Exit Sub
    End If

    '暗号化を設定
    Ret = Api_DecryptFile(FileName, 0)

    '設定できたときは
    If Ret <> 0 Then
        Text2.SetWindowText "解除しました。"
    End If
End Sub

```

```

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
Sub Edit1_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount (DF)
    FileName = GetDropFileName (DF, 0)
    Edit1.SetWindowText FileName
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルの終わり (EOF) を移動

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す
WriteFile データをファイルに書き出す
SetFilePointer 開いているファイルのポインタを移動
SetEndOfFile 指定されたファイルの終わり (EOF) ポジションをファイルポインタの現在のポジションに移動
CloseHandle オープンされているオブジェクトハンドルをクローズ

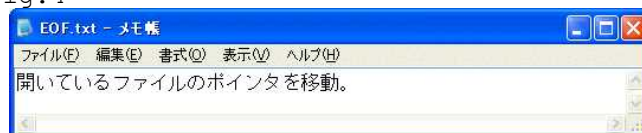
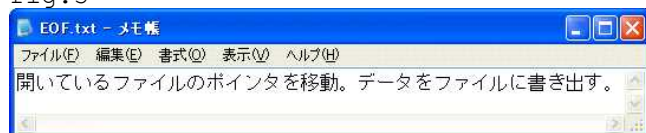
左:起動時、C:\¥Temp¥EOF.txtにMyStrを書き込んでいます。確認(fig.3) 右:Length=36をEOFとして再度書き込みます。確認(fig.4)



fig.3



fig.4



```

'=====
'= ファイルの終わり (EOF) を移動
'= (SetEndOfFile.bas)
'=====
#include "Windows.bi"

' 指定したファイルをオープンし、デバイスハンドルを返す
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal
dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' データをファイルに書き出す
Declare Function Api_WriteFile& Lib "kernel32" Alias "WriteFile" (ByVal hFile&, lpBuffer
As Any, ByVal nNumberOfBytesToWrite&, lpNumberOfBytesWritten&, lpOverlapped As Any)

' 開いているファイルのポインタを移動
Declare Function Api_SetFilePointer& Lib "kernel32" Alias "SetFilePointer" (ByVal
hFile&, ByVal lDistanceToMove&, lpDistanceToMoveHigh&, ByVal dwMoveMethod&)

' 指定されたファイルの終わり (EOF) ポジションをファイルポインタの現在のポジションに移動
Declare Function Api_SetEndOfFile& Lib "kernel32" Alias "SetEndOfFile" (ByVal hFile&)

```


・オープンされているオブジェクトハンドルをクローズ

```
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

#define GENERIC_READ -2147483648          '読み込みモード(&H80000000)
#define GENERIC_WRITE &H40000000        '書き込みモード
#define FILE_SHARE_READ &H1             '後続のオープン操作で読み取りアクセスが要求された場合、そのオープンを許可
#define FILE_SHARE_WRITE &H2            '後続のオープン操作で書き込みアクセスが要求された場合、そのオープンを許可
#define OPEN_ALWAYS 4                    'ファイルをオープンする(存在しない場合作成)
#define FILE_BEGIN 0                     'ファイルポインタを先頭に

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var hFile As Long
    Var MyStr As String
    Var Length As Long
    Var BytesWritten As Long
    Var Path As String
    Var Ret As Long

    Path = "C:¥Temp¥EOF.txt"
    MyStr = "開いているファイルのポインタを移動。データをファイルに書き出す。"
    Length = Len(MyStr)

    hFile = Api_CreateFile(Path, GENERIC_WRITE, FILE_SHARE_READ Or FILE_SHARE_WRITE,
ByVal 0, OPEN_ALWAYS, 0, 0)
    If hFile = -1 Then End

    Ret = Api_WriteFile(hFile, MyStr, Length, BytesWritten, ByVal 0)
    A% = MsgBox("", "' " & "OK 'をクリックする前に 'C:¥Temp¥EOF.txt' を確認してください!",
0, 2)

    Ret = Api_SetFilePointer(hFile, 36, 0, FILE_BEGIN)
    Ret = Api_SetEndOfFile(hFile)
    Ret = Api_CloseHandle(hFile)

    A% = MsgBox("", "' " & Path & "' を、もう一度確認してください!", 0, 2)
    End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ファイルのコピーと削除

SHFileOperation ファイルシステムオブジェクトのコピー・移動・名前の変更・削除を行う

例では、Test1.txt を Test2.txt としてコピー、また、Test2.txt を削除しています。



```
'= ファイルのコピーと削除
'= (SHFileOperation2.bas)
#include "Windows.bi"
```

- 'ファイルをコピー
- 'ファイルを削除
- 'ファイルを移動
- 'ファイル名を変更
- 'できる限りアンドゥ情報を保持。pFrom で指定されたファイル名がフルパスで指定されている必要がある。削除
- '現在使用されない
- 'ワイルドカードファイル名(*.*)が指定された場合にのみ操作を実行
- 'pTo メンバが複数のファイル名を指定していることを示す
- '表示されるダイアログボックスで「はい」または「すべて」を選択するようにする
- '新しいディレクトリを作成する必要がある場合に、作成するかどうかの確認をしない
- 'エラーが発生した場合に、ユーザーインターフェースを表示しない
- 'ローカルディレクトリのように操作を行なう。サブディレクトリに対して再帰的に操作を行なわない
- '移動、コピー、名前の変更の操作において、指定したファイル名がすでに存在していた場合には、操作対象のフ
- '経過を表すダイアログボックスを表示しない
- '経過を表すダイアログボックスにファイル名を表示しない
- '
- 'NULL文字(¥0)

- 'フォームのウィンドウハンドル
- '処理内容(コピー・削除・移動・名前の変更)
- '元ファイル名(ファイル名の後ろには2つのヌル文字を入れる)
- '新ファイル名(削除の場合は不要・ファイル名の後ろには2つのヌル文字を入れる)
- '動作オプション
- '処理終了前にキャンセルしたときは[1]
- 'ファイルネームマッピングオブジェクト
- 'ダイアログボックスのキャプション

```
Declare Function Api_SHFileOperation Lib "shell32" Alias "SHFileOperationA" (lpFileOp
As SHFILEOPSTRUCT)
```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Text(0).SetWindowText "Test.txt"
    Text(1).SetWindowText "Test2.txt"
End Sub

'=====
'= ファイルコピー
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var SrcFile As String
    Var DestFile As String
    Var Ret As Long
    SrcFile = "Test.txt" & vbNullChar & vbNullChar
    DestFile = "Test2.txt" & vbNullChar & vbNullChar

    sfos.hwnd = GethWnd
    sfos.fFlags = FOF_ALLOWUNDO
    sfos.wFunc = FO_COPY
    sfos.pFrom = StrAdr(SrcFile)
    sfos.pTo = StrAdr(DestFile)

    Ret = Api_SHFileOperation(sfos)
End Sub

'=====
'= ファイル削除
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Var DelFile As String
    Var Ret As Long

    DelFile = "Test2.txt" & vbNullChar & vbNullChar

    sfos.hwnd = GethWnd
    sfos.fFlags = FOF_ALLOWUNDO
    sfos.wFunc = FO_DELETE
    sfos.pFrom = StrAdr(DelFile)
    sfos.pTo = 0

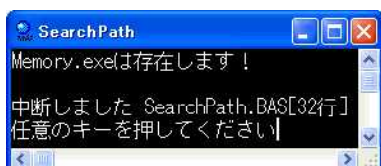
    Ret = Api_SHFileOperation(sfos)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルの存在を検索

SearchPath ファイルの存在を検索します。



```

'=====
'= ファイルの存在を検索
'= (SearchPath.bas/P)
'=====

' ファイルやフォルダを検索する
Declare Function Api_SearchPath Lib "kernel32" Alias "SearchPathA" (ByVal lpPath$,
ByVal lpFileName$, ByVal lpExtension$, ByVal nBufferLength, ByVal lpBuffer$, ByVal
lpFilePart$)

Var sDrive As String
Var sName As String
Var sExtn As String
Var nName As String
Var nFile As String
Var nLeng As Long
Var Ret As Long

sDrive = "C:¥_FB_API_P"
sName = "Memory"
sExtn = ".exe"

nName = string$(250, Chr$(0))
nFile = string$(250, Chr$(0))
nLeng = len(nName)

Ret = Api_SearchPath(sDrive, sName, sExtn, nLeng, nName, nFile)

If Ret <> 0 Then
    Print sName & sExtn & "は存在します！"
Else
    Print sName & sExtn & "は存在しません！"
End If

Stop
End

```

ファイルのタイムスタンプを取得

選択したファイルのタイムスタンプを取得します。新規作成年月日・時間・曜日、最終更新年月日・時間・曜日および最終アクセス年月日・時間・曜日を取得します。

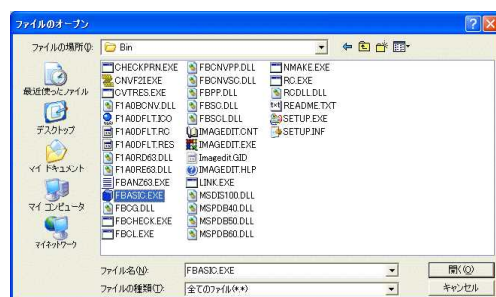
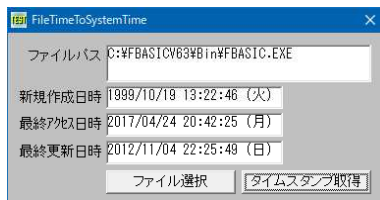
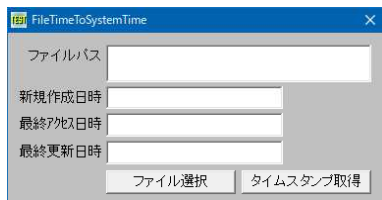
CreateFile ファイルのハンドルを取得

GetFileTime ファイルスタンプを取得

FileTimeToSystemTime システムファイルタイムに変換

FileTimeToLocalFileTime ローカルタイムに変換

FindClose ファイルのハンドルを閉じる



```

'=====
'= ファイルのタイムスタンプを取得
'= (FileTime.bas)
'=====

#include "Windows.bi"

```

```

Type SECURITY_ATTRIBUTES
    nLength As Long

```

```

    lpSecurityDescriptor    As Long
    bInheritHandle          As Long
End Type

Type SYSTEMTIME
    wYear                   As Integer    '年
    wMonth                  As Integer    '月 (1月=1、2月=2...)
    wDayOfWeek              As Integer    '曜日 (日曜=0、月曜=1...)
    wDay                    As Integer    '日
    wHour                   As Integer    '時
    wMinute                 As Integer    '分
    wSecond                 As Integer    '秒
    wMilliseconds           As Integer    'ミリ秒
End Type

Type FILETIME
    dwLowDateTime           As Long        '下位32ビット値
    dwHighDateTime          As Long        '上位32ビット値
End Type

#define GENERIC_READ -2147483648          '&H80000000
#define GENERIC_WRITE &H40000000         '
#define FILE_SHARE_READ &H1              '後続のオープン操作で読み取りアクセスが要求された場合、そのオープンを許可
#define FILE_SHARE_WRITE &H2              '後続のオープン操作で書き込みアクセスが要求された場合、そのオープンを許可

#define CREATE_NEW 1                       '
#define CREATE_ALWAYS 2                   '
#define OPEN_EXISTING 3                   'ファイルをオープンする
#define OPEN_ALWAYS 4                     '
#define TRUNCATE_EXISTING 5               '
#define FILE_ATTRIBUTE_ARCHIVE &H20      'アーカイブ属性を示す定数
#define FILE_ATTRIBUTE_HIDDEN &H2        '隠しファイル属性
#define FILE_ATTRIBUTE_NORMAL &H80       '他のファイル属性を持たない
#define FILE_ATTRIBUTE_READONLY &H1      '読み込み専用属性
#define FILE_ATTRIBUTE_SYSTEM &H4        'システムファイル属性
#define FILE_ATTRIBUTE_TEMPORARY &H100    '一時ファイル属性を示す定数
#define FILE_FLAG_WRITE_THROUGH -2147483648 'キャッシュに書き込まれたデータをそのまま直接ディスクに書き込むようにする(&H8000
#define FILE_FLAG_OVERLAPPED &H40000000  '時間のかかる処理に対してReadFile関数やWriteFile関数でERROR_IO_PENDING拡張エラ
#define FILE_FLAG_NO_BUFFERING &H20000000 'システムキャッシュを使用せずにファイルをオープンするように指定
#define FILE_FLAG_RANDOM_ACCESS &H10000000 'ファイルにランダムアクセスすることをシステムに示す
#define FILE_FLAG_SEQUENTIAL_SCAN &H80000000 'ファイルにシーケンシャルアクセスすることをシステムに示す
#define FILE_FLAG_DELETE_ON_CLOSE &H40000000 'そのファイルを指すすべてのファイルのハンドルがクローズされた時に、そのファイルを
#define FILE_FLAG_POSIX_SEMANTICS &H10000000 'POSIXの規則に従ってファイルにアクセス

' ファイルのハンドルを閉じる
Declare Function Api_FindClose& Lib "kernel32" Alias "FindClose" (ByVal hFindFile&)

' ファイルのハンドルをGET
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$, ByVal dwDesiredAccess&, ByVal dwShareMode&, ByVal lpSecurityAttributes&, ByVal dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' ファイルスタンプを取得する
Declare Function Api_GetFileTime& Lib "kernel32" Alias "GetFileTime" (ByVal hFile&, CreationTime As FILETIME, LastAccessTime As FILETIME, LastWriteTime As FILETIME)

' システムファイルタイムに変換
Declare Function Api_FileTimeToSystemTime& Lib "kernel32" Alias "FileTimeToSystemTime" (lpFileTime As FILETIME, SystemTime As SYSTEMTIME)

' ローカルタイムに変換
Declare Function Api_FileTimeToLocalFileTime& Lib "kernel32" Alias "FileTimeToLocalFileTime" (lpFileTime As FILETIME, LocalFileTime As FILETIME)

Var Shared Text (7) As object

```

```

Var Shared Button(1) As Object

For i = 0 To 7
    If i < 2 Then
        Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1)))
        Button(i).SetFontSize 14
    End If
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next

Var Shared FileName As String

'=====
'=
'=====
Declare Function WeekDay(Str As Integer) As String
Function WeekDay(Str As Integer) As String
    Select Case Str
        Case 0
            WeekDay = "日"
        Case 1
            WeekDay = "月"
        Case 2
            WeekDay = "火"
        Case 3
            WeekDay = "水"
        Case 4
            WeekDay = "木"
        Case 5
            WeekDay = "金"
        Case 6
            WeekDay = "土"
    End Select
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    For i = 4 To 7
        Text(i).SetWindowText ""
    Next

    FileName = WinOpenDlg("ファイルのオープン", "*.*", "全てのファイル(*.*)", 0)
    If FileName <> Chr$(&H1B) Then
        Text(4).SetWindowText FileName
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var CreationTime As FILETIME
    Var LastAccessTime As FILETIME
    Var LastWriteTime As FILETIME
    Var LocalFileTime As FILETIME
    Var SystemTime As SYSTEMTIME
    Var Ret As Long

    'ファイルのハンドル取得する
    Ret = Api_CreateFile(FileName, GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0)

    If Ret <> -1 Then

```

```

'ファイルスタンプを取得
Ret = Api_GetFileTime(Ret, CreationTime, LastAccessTime, LastWriteTime)

If Ret <> 0 Then

    'ファイルの新規作成日を取得
    Ret = Api_FileTimeToLocalFileTime(CreationTime, LocalFileTime)

    'システムタイムに変換
    Ret = Api_FileTimeToSystemTime(LocalFileTime, SystemTime)

    If Ret <> 0 Then
        ymd$ = Trim$(Str$(SystemTime.wYear)) & "/" & Right$(Str$(100 +
SystemTime.wMonth), 2) & "/" & Right$(Str$(100 + SystemTime.wDay), 2) & " "
        hms$ = Right$(Str$(100 + SystemTime.wHour), 2) & ":" & Right$(Str$(100 +
SystemTime.wMinute), 2) & ":" & Right$(Str$(100 + SystemTime.wSecond), 2) & " "
        week$ = "(" & WeekDay(SystemTime.wDayOfWeek) & ")"
        Text(5).SetWindowText ymd$ & hms$ & week$
    End If

    '最終アクセス日を取得
    Ret = Api_FileTimeToLocalFileTime(LastAccessTime, LocalFileTime)

    'システムタイムに変換
    Ret = Api_FileTimeToSystemTime(LocalFileTime, SystemTime)

    If Ret <> 0 Then
        ymd$ = Trim$(Str$(SystemTime.wYear)) & "/" & Right$(Str$(100 +
SystemTime.wMonth), 2) & "/" & Right$(Str$(100 + SystemTime.wDay), 2) & " "
        hms$ = Right$(Str$(100 + SystemTime.wHour), 2) & ":" & Right$(Str$(100 +
SystemTime.wMinute), 2) & ":" & Right$(Str$(100 + SystemTime.wSecond), 2) & " "
        week$ = "(" & WeekDay(SystemTime.wDayOfWeek) & ")"
        Text(6).SetWindowText ymd$ & hms$ & week$
    End If

    'ファイルの最終更新日を取得
    Ret = Api_FileTimeToLocalFileTime(LastWriteTime, LocalFileTime)

    'システムタイムに変換
    Ret = Api_FileTimeToSystemTime(LocalFileTime, SystemTime)

    If Ret <> 0 Then
        ymd$ = Trim$(Str$(SystemTime.wYear)) & "/" & Right$(Str$(100 +
SystemTime.wMonth), 2) & "/" & Right$(Str$(100 + SystemTime.wDay), 2) & " "
        hms$ = Right$(Str$(100 + SystemTime.wHour), 2) & ":" & Right$(Str$(100 +
SystemTime.wMinute), 2) & ":" & Right$(Str$(100 + SystemTime.wSecond), 2) & " "
        week$ = "(" & WeekDay(SystemTime.wDayOfWeek) & ")"
        Text(7).SetWindowText ymd$ & hms$ & week$
    End If
End If

Ret = Api_FindClose(Ret)
End Sub

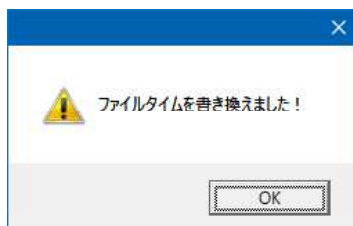
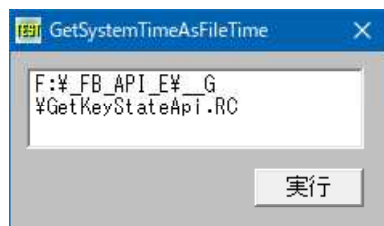
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルのタイムスタンプを設定(1)

OpenFile ファイルを開く

CloseHandle オープンされているオブジェクトハンドルをクローズ
 GetSystemTimeAsFileTime 現在のシステム日付とシステム時刻を世界協定時刻 (UTC) 形式で取得
 GetFileTime ファイルスタンプを取得
 SetFileTime ファイルの作成日時などを設定
 OF_WRITE(&H1) ファイルを書き込みで開く



名前	更新日時
GetKeyStateApi.RC	2004/11/22 16:21
名前	更新日時
GetKeyStateApi.RC	2017/04/24 20:50

更新日時が変更されています。

```

'=====
' = ファイルのタイムスタンプを設定 (II)
' = (GetSystemTimeAsFileTime.bas)
'=====
#include "Windows.bi"

#define OFS_MAXPATHNAME 128

Type OFSTRUCT
    cBytes           As Byte
    fFixedDisk       As Byte
    nErrCode         As Integer
    Reserved1        As Integer
    Reserved2        As Integer
    szPathName       As String * OFS_MAXPATHNAME
End Type

Type FILETIME
    dwLowDateTime    As Long
    dwHighDateTime   As Long
End Type

' ファイルを開く
Declare Function Api_OpenFile& Lib "kernel32" Alias "OpenFile" (ByVal lpFileName$,
lpReOpenBuff As OFSTRUCT, ByVal wStyle&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)

' 現在のシステム日付とシステム時刻を世界協定時刻 (UTC) 形式で取得
Declare Sub Api_GetSystemTimeAsFileTime Lib "kernel32" Alias "GetSystemTimeAsFileTime"
(lpSysTimeAsFileTime As FILETIME)

' ファイルスタンプを取得
Declare Function Api_GetFileTime& Lib "Kernel32" Alias "GetFileTime" (ByVal hFile&,
lpCreationTime As FILETIME, lpOldTime As FILETIME, lpLastWriteTime As FILETIME)

' ファイルの作成日時などを設定
Declare Function Api_SetFileTime& Lib "Kernel32" Alias "SetFileTime" (ByVal hFile&,
lpCreationTime As FILETIME, lpOldTime As FILETIME, lpLastWriteTime As FILETIME)

#define OF_WRITE &H1                                'ファイルを書き込みで開く

Var Shared Edit1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared FileName As String

'=====
' =
'=====
Declare Sub Button1_on edec1 ()
  
```



```

Sub Button1_on()
    Var os As OFSTRUCT
    Var ft As FILETIME
    Var NewTime As FILETIME
    Var OldTime As FILETIME
    Var ModifiedTime As FILETIME
    Var hFile As Long
    Var Ret As Long

    FileName = Edit1.GetWindowText
    hFile = Api_OpenFile(FileName, os, OF_WRITE)

    'FILETIME構造体より現在の時間を取得
    Api_GetSystemTimeAsFileTime ft

    'ファイル独自の時間を取得
    Ret = Api_GetFileTime(hFile, NewTime, OldTime, ModifiedTime)

    '新しい時間を書き込む
    Ret = Api_SetFileTime(ByVal hFile, NewTime, OldTime, ft)
    If Ret <> 0 Then
        A% = MessageBox("", "ファイルタイムを書き換えました！", 0, 2)
    Else
        A% = MessageBox("", "ファイルタイムの書き換えはできませんでした！", 0, 2)
    End If

    Ret = Api_CloseHandle(hFile)
End Sub

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
Sub Edit1_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount(DF)
    FileName = GetDropFileName(DF, 0)
    Edit1.SetWindowText FileName
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルのタイムスタンプを設定(II)

ファイルのタイムスタンプを設定します。

CreateFile ファイルなどの作成やオープンや切り捨てを行う
CloseHandle オープンされているオブジェクトハンドルをクローズ
GetSystemTime システムの現在の日付と時刻を取得
SystemTimeToFileTime システム時間をファイル時間に変換
SetFileTime ファイルの作成日時などを設定



ファイルオープンダイアログでファイルを選択、「日時設定」で作成日時を現在時間に書き換えています。



結果をプロパティで確認しています。SysTimeは世界標準時なので日本時間に変換(+9時間)しています。

```
Ret = Api_SystemTimeToFileTime(SysTime, CreateTime)  作成日時
Ret = Api_SystemTimeToFileTime(SysTime, AccessTime)  アクセス日時
Ret = Api_SystemTimeToFileTime(SysTime, WriteTime)  更新日時

'=====
'= ファイルのタイムスタンプを設定
'= (SetFileTime.bas)
'=====
#include "Windows.bi"

Type SECURITY_ATTRIBUTES
    nLength          As Long
    lpSecurityDescriptor As Long
    bInheritHandle    As Long
End Type

#define INVALID_HANDLE_VALUE (-1)      ' ファイルハンドルの値が無効であることを示す定数の宣言
#define GENERIC_WRITE &H40000000      ' オブジェクトへのアクセスの種類を指定する定数の宣言
#define OPEN_EXISTING 3                ' ファイルへの動作を指定する定数の宣言

' ファイルなどの作成やオープンや切り捨てを行う関数の宣言
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lFileName$,
ByVal dDesiredAccess&, ByVal dShareMode&, lSecurityAttributes As SECURITY_ATTRIBUTES,
ByVal dCreationDisposition&, ByVal dFlagsAndAttributes&, ByVal hTemplateFile&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

' 日付と時刻を定義する構造体
Type SYSTEMTIME
    wYear          As Integer
    wMonth         As Integer
    wDayOfWeek     As Integer
    wDay           As Integer
    wHour          As Integer
    wMinute        As Integer
    wSecond        As Integer
    wMilliseconds  As Integer
End Type

' システムの現在の日付と時刻を取得する関数の宣言
Declare Sub Api_GetSystemTime Lib "kernel32" Alias "GetSystemTime" (lpSystemTime As
SYSTEMTIME)

' ファイル時間を定義する構造体
Type FILETIME
    dwLowDateTime   As Long
    dwHighDateTime  As Long
End Type

' システム時間をファイル時間に変換する関数の宣言
Declare Function Api_SystemTimeToFileTime& Lib "kernel32" Alias "SystemTimeToFileTime"
(lpSystemTime As SYSTEMTIME, lpFileTime As FILETIME)

' ファイルの作成日時などを設定する関数の宣言
Declare Function Api_SetFileTime& Lib "kernel32" Alias "SetFileTime" (ByVal hFile&,
lpCreationTime As FILETIME, lpLastAccessTime As FILETIME, lpLastWriteTime As FILETIME)

Var Shared Edit1 As Object
```

```

Var Shared Text3 As Object
Edit1.Attach GetDLgItem("Edit1") : Edit1.SetFontSize 14
Text3.Attach GetDLgItem("Text3") : Text3.SetFontSize 14

Var Shared FileName As String

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    FileName = WinOpenDlg("ファイルのオープン", "*.*", "全てのファイル(*)", 0)
    If FileName <> Chr$(&H1B) Then
        Edit1.SetWindowText FileName
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var FileHandle As Long
    Var SecAttr As SECURITY_ATTRIBUTES
    Var SysTime As SYSTEMTIME
    Var LocalTime As SYSTEMTIME
    Var CreateTime As FILETIME
    Var AccessTime As FILETIME
    Var WriteTime As FILETIME
    Var ymd As String
    Var hms As String
    Var Ret As Long

    'セキュリティ構造体を初期化
    SecAttr.nLength = Len(SecAttr)

    'ファイルをオープン
    FileHandle = Api_CreateFile(FileName, GENERIC_WRITE, 0, SecAttr, OPEN_EXISTING, 0, 0)

    'ファイルハンドルの値が有効であるときは
    If FileHandle <> INVALID_HANDLE_VALUE Then

        'システムの日時を取得
        Api_GetSystemTime SysTime

        'ローカルの日時を表示
        ymd = Trim$(Str$(SysTime.wYear)) & "/" & Right$(Str$(100 + SysTime.wMonth), 2) &
        "/" & Right$(Str$(100 + SysTime.wDay), 2)

        'SysTime (GMT) を日本時間 (JMT:Hour) に変換
        If SysTime.wHour < 16 Then Hour = SysTime.wHour + 9 Else Hour = SysTime.wHour - 15
        hms = Right$(Str$(100 + Hour), 2) & ":" & Right$(Str$(100 + SysTime.wMinute), 2) &
        ":" & Right$(Str$(100 + SysTime.wSecond), 2)
        Text3.SetWindowText ymd & " " & hms

        'システム時間をファイル時間に変換
        Ret = Api_SystemTimeToFileTime(SysTime, CreateTime)

        'ファイルの作成日時を設定          作成日時      アクセス日時  更新日時
        Ret = Api_SetFileTime(FileHandle, CreateTime, AccessTime, WriteTime)

        'ファイルをクローズ
        Ret = Api_CloseHandle(FileHandle)
    End If
End Sub

'=====
'=
'=====

```

```
While 1
    WaitEvent
Wend
Stop
End
```

ファイルのタイムスタンプを変更(1)

ファイルのタイムスタンプを変更します。

TouchFileTimes 指定されたファイルが最後に変更されたことを示す日付と時刻を更新

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す

CloseHandle オープンされているオブジェクトハンドルをクローズ

Windows95/98には対応していません。



変更後の確認



```
'=====
'= タイムスタンプの書き換え
'= Windows 95/98:非対応
'= (TouchFileTimes.bas)
'=====

#define GENERIC_WRITE &H40000000      '書き込みモード
#define OPEN_EXISTING 3                'ファイルをオープンする(存在しない場合失敗)
#define FILE_SHARE_READ &H1           '後続のオープン操作で読み取りアクセスが要求された場合、そのオープンを許可
#define FILE_SHARE_WRITE &H2          '後続のオープン操作で書き込みアクセスが要求された場合、そのオープンを許可

' 指定されたファイルが最後に変更されたことを示す日付と時刻を更新
Declare Function Api_TouchFileTimes& Lib "imagehlp" Alias "TouchFileTimes" (ByVal FileHandle&, ByRef pSystemTime As Any)

' 指定したファイルをオープンし、デバイスハンドルを返す
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$, ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)

Var sFile as string
Var lHandle As Long

sFile = "c:\¥test¥myfile.txt"

lHandle = Api_CreateFile(sFile, GENERIC_WRITE, FILE_SHARE_READ Or FILE_SHARE_WRITE,
```

```

ByVal 0, OPEN_EXISTING, 0, 0)
If Api_TouchFileTimes(lHandle, ByVal 0) = 0 Then
    Print "ファイルのタイムスタンプ変更は失敗しました！"
Else
    Print "ファイルのタイムスタンプを変更しました！"
End If

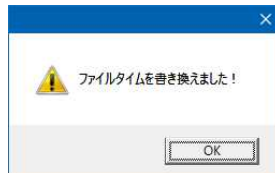
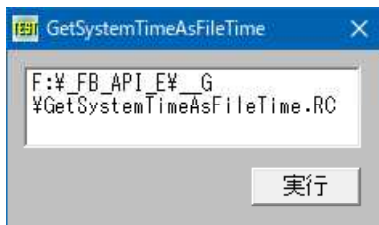
Ret = Api_CloseHandle(hWnd)

Stop
End

```

ファイルのタイムスタンプを設定 (II)

OpenFile ファイルを開く
CloseHandle オープンされているオブジェクトハンドルをクローズ
GetSystemTimeAsFileTime 現在のシステム日付とシステム時刻を世界協定時刻 (UTC) 形式で取得
GetFileTime ファイルスタンプを取得
SetFileTime ファイルの作成日時などを設定
OF_WRITE (&H1) ファイルを書き込みで開く



プロパティで確認:更新日時等が書き換わっています。

作成日時:	2014年2月9日、19:52:03	作成日時:	2014年2月9日、19:52:03
更新日時:	2008年5月19日、18:45:28	更新日時:	2017年4月15日、16:28:25
アクセス日時:	2014年2月9日、19:52:03	アクセス日時:	2017年4月15日、16:27:49

```

'=====
' = ファイルのタイムスタンプを設定 (II)
' =   (GetSystemTimeAsFileTime.bas)
'=====
#include "Windows.bi"

#define OFS_MAXPATHNAME 128

Type OFSTRUCT
    cBytes           As Byte
    fFixedDisk       As Byte
    nErrCode         As Integer
    Reserved1        As Integer
    Reserved2        As Integer
    szPathName       As String * OFS_MAXPATHNAME
End Type

Type FILETIME
    dwLowDateTime    As Long
    dwHighDateTime   As Long
End Type

' ファイルを開く
Declare Function Api_OpenFile& Lib "kernel32" Alias "OpenFile" (ByVal lpFileName$,
lpReOpenBuff As OFSTRUCT, ByVal wStyle&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)

' 現在のシステム日付とシステム時刻を世界協定時刻 (UTC) 形式で取得
Declare Sub Api_GetSystemTimeAsFileTime Lib "kernel32" Alias "GetSystemTimeAsFileTime"
(lpSysTimeAsFileTime As FILETIME)

' ファイルスタンプを取得
Declare Function Api_GetFileTime& Lib "Kernel32" Alias "GetFileTime" (ByVal hFile&,
lpCreationTime As FILETIME, lpOldTime As FILETIME, lpLastWriteTime As FILETIME)

```

' ファイルの作成日時などを設定

```
Declare Function Api_SetFileTime& Lib "Kernel32" Alias "SetFileTime" (ByVal hFile&, lpCreationTime As FILETIME, lpOldTime As FILETIME, lpLastWriteTime As FILETIME)
```

```
#define OF_WRITE &H1
```

' ファイルを書き込みで開く

```
Var Shared Edit1 As Object  
Var Shared Button1 As Object
```

```
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
Var Shared FileName As String
```

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var os As OFSTRUCT
```

```
    Var ft As FILETIME
```

```
    Var NewTime As FILETIME
```

```
    Var OldTime As FILETIME
```

```
    Var ModifiedTime As FILETIME
```

```
    Var hFile As Long
```

```
    Var Ret As Long
```

```
    FileName = Edit1.GetWindowText
```

```
    hFile = Api_OpenFile(FileName, os, OF_WRITE)
```

' FILETIME構造体より現在の時間を取得

```
    Api_GetSystemTimeAsFileTime ft
```

' ファイル独自の時間を取得

```
    Ret = Api_GetFileTime(hFile, NewTime, OldTime, ModifiedTime)
```

' 新しい時間を書き込む

```
    Ret = Api_SetFileTime(ByVal hFile, NewTime, OldTime, ft)
```

```
    If Ret <> 0 Then
```

```
        A% = MessageBox("", "ファイルタイムを書き換えました！", 0, 2)
```

```
    Else
```

```
        A% = MessageBox("", "ファイルタイムの書き換えはできませんでした！", 0, 2)
```

```
    End If
```

```
    Ret = Api_CloseHandle(hFile)
```

```
End Sub
```

```
'=====
```

'= シェルドロップされたファイル名を取得

```
'=====
```

```
Declare Sub Edit1_DropFiles edec1 (ByVal DF As Long)
```

```
Sub Edit1_DropFiles (ByVal DF As Long)
```

```
    Var CN As Long
```

```
    CN = GetDropFileCount(DF)
```

```
    FileName = GetDropFileName(DF, 0)
```

```
    Edit1.SetWindowText FileName
```

```
End Sub
```

```
'=====
```

```
'=
```

```
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

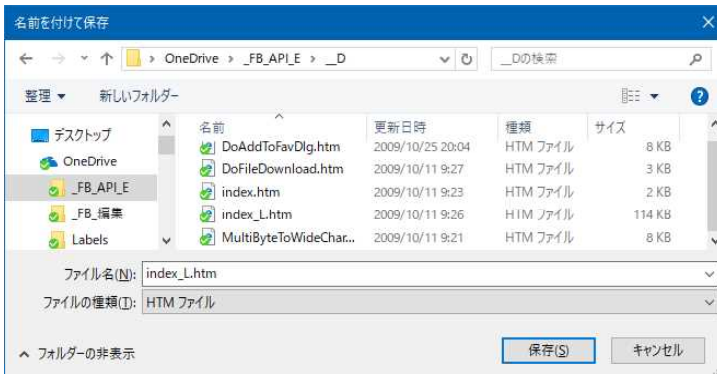
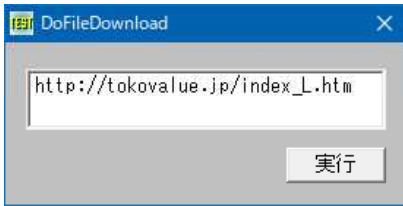
```
Stop
```

```
End
```

ファイルのダウンロード

DoFileDownload ファイルのダウンロードを呼び出す

MultiByteToWideChar ファイルのダウンロード



```
'=====
'= ファイルのダウンロード
'= (DoFileDownload.bas)
'=====

#include "Windows.bi"

' ファイルのダウンロード
Declare Function Api_DoFileDownload& Lib "shdocvw" Alias "DoFileDownload" (ByVal
lpSzFile$)

' ANSI文字列をUnicode文字列に変換
Declare Function Api_MultiByteToWideChar& Lib "Kernel32" Alias "MultiByteToWideChar"
(ByVal CodePage&, ByVal dwFlags&, ByVal lpMultiByteStr$, ByVal cchMultiByte&, ByVal
lpWideCharStr$, ByVal cchWideChar&)

#define CP_ACP 0
#define CP_MACP 2
#define CP_OEMCP 1
#define CP_SYMBOL 42
#define CP_THREAD_ACP 3
#define CP_UTF7 65000
#define CP_UTF8 65001

'ANSIコードページ
'Macintoshコードページ
'OEMコードページ
'シンボルコードページ (Windows2000・XP)
'呼び出しスレッドのANSIコードページ (Windows2000・XP)
'UTF-7を使用して変換 (Windows98・Me・NT4.0以降)
'UTF-8を使用して変換 (Windows98・Me・NT4.0以降) これ
を指定した場合、dwFlagsパラメータは0

Var Shared Edit1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====

Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Buff As String
    Var wBuff As String
    Var wBufflen As Long
    Var Ret As Long

    Buff = String$(256, Chr$(0))
    Buff = Edit1.GetWindowText
    wBuff = String$(256, Chr$(0))
```

```

' 必要なバッファサイズを取得
wBufflen = Api_MultiByteToWideChar(CP_ACP, 0, Buff, -1, wBuff, 0)
Ret = Api_MultiByteToWideChar(CP_ACP, 0, Buff, -1, wBuff, wBufflen)

wBuff = Left$(wBuff, (wBufflen - 1) * 2)

Ret = Api_DoFileDownload(wBuff)
End Sub

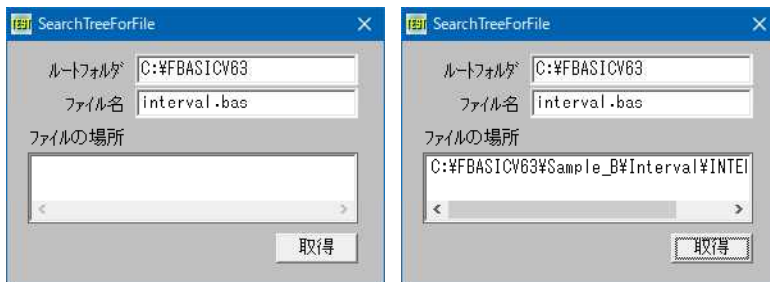
' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルのディレクトリツリーを検索

指定されたファイルのディレクトリツリーを検索します。フォルダはC:¥だけでも良いのですが、絞り込んだ方が検索が速くなります。

SearchTreeForFile 指定されたファイルのディレクトリツリーを検索



```

' =====
' = ファイルのパスを取得
' = (SearchTreeForFile.bas)
' =====
#include "Windows.bi"

' 指定されたファイルのディレクトリツリーを検索
Declare Function Api_SearchTreeForFile& Lib "imagehlp" Alias "SearchTreeForFile" (ByVal RootPath$, ByVal InputPathName$, ByVal OutputPathBuffer$)

#define MAX_PATH 260

Var Shared Edit(2) As Object
Var Shared Text(2) As Object

For i = 0 To 2
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
Next

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var RootFolder As String
    Var FileName As String
    Var Buff As String
    Var Ret As Long

    RootFolder = Edit(0).GetWindowText
    FileName = Edit(1).GetWindowText

```



```

Buff = String$(MAX_PATH, Chr$(0))      'バッファ初期化

Ret = Api_SearchTreeForFile(RootFolder, FileName, Buff)

If Ret <> 0 Then
    Edit(2).SetWindowText Left$(Buff, InStr(1, Buff, Chr$(0)) - 1)
Else
    Edit(2).SetWindowText "File not found!"
End If
End Sub

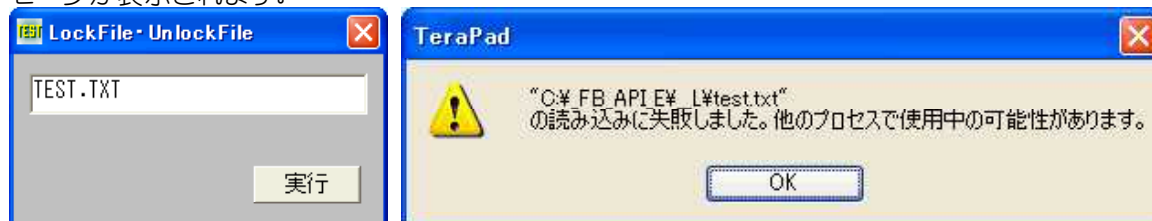
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルの特定領域をロック

CreateFile 指定したファイルをオープンし、デバイスハンドルを返す
LockFile オープンしているファイルの特定の領域を、他のプロセスがアクセスできないようにロック
UnlockFile 他のプロセスがアクセスできるようにロックを解除
CloseHandle オープンされているオブジェクトハンドルをクローズ
Sleep カレントスレッドの実行を指定の時間だけ中断

例では、実行をクリックすると20秒間ロックされます。その間に指定したファイルをアクセスした場合右図のようなメッセージが表示されます。



```

'=====
'= ファイルの特定領域をロック
'= (LockFile.bas)
'=====
#include "Windows.bi"

' 指定したファイルをオープンし、デバイスハンドルを返す
Declare Function Api_CreateFile& Lib "Kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal
dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' オープンしているファイルの特定の領域を、他のプロセスがアクセスできないようにロック
Declare Function Api_LockFile& Lib "kernel32" Alias "LockFile" (ByVal hFile&, ByVal
dwFileOffsetLow&, ByVal dwFileOffsetHigh&, ByVal nNumberOfBytesToLockLow&, ByVal
nNumberOfBytesToLockHigh&)

' 他のプロセスがアクセスできるようにロックを解除
Declare Function Api_UnlockFile& Lib "kernel32" Alias "UnlockFile" (ByVal hFile&, ByVal
dwFileOffsetLow&, ByVal dwFileOffsetHigh&, ByVal nNumberOfBytesToUnlockLow&, ByVal
nNumberOfBytesToUnlockHigh&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)

' カレントスレッドの実行を指定の時間だけ中断
Declare Sub Api_Sleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds&)

#define GENERIC_READ -2147483648      '読み込みモード(&H80000000)

```

```

#define GENERIC_WRITE &H40000000
#define FILE_ATTRIBUTE_NORMAL &H80
#define OPEN_EXISTING 3

'書き込みモード
'他のファイル属性を持たない
'ファイルをオープンする(存在しない場合失敗)

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Var Shared PathName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    PathName = Edit1.GetWindowText
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var hFile As Long
    Var Ret As Long

    'ファイルのオープン
    hFile = Api_CreateFile(PathName, GENERIC_READ, 0, ByVal 0, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, ByVal 0)

    'ファイルのロック
    Ret = Api_LockFile(hFile, 0, 0, GENERIC_READ, 0)
    Text1.SetWindowText "ファイルをロックしました。"

    'ファイルの書き込み処理などを実行
    Api_Sleep 20000

    'ファイルのアンロック
    Ret = Api_UnlockFile(hFile, 0, 0, GENERIC_READ, 0)
    Text1.SetWindowText "ファイルをアンロックしました。"

    'ファイルのクローズ
    Ret = Api_CloseHandle(hFile)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルのファイルタイムを比較

ファイルのファイルタイムを比較します。

GetSystemTime 現在の世界協定時刻を取得

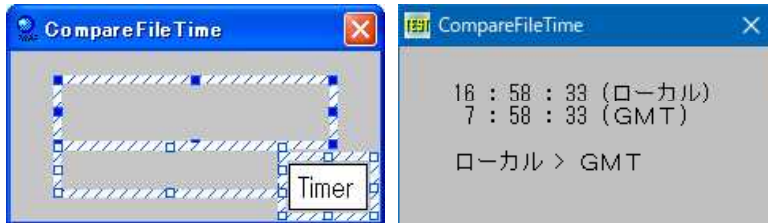
LocalFileTimeToFileTime ローカルファイルタイムをUTCファイルタイム形式で取得

SystemTimeToFileTime システムタイムをファイルタイムに変換

CompareFileTime ファイルのファイルタイムを比較

FileTimeToSystemTime ファイルタイムをシステムファイルに変換

FileTimeToLocalFileTime 世界協定時刻形式ファイル時間をローカルファイル時間に変換



```
'=====
'= ファイルのファイルタイムを比較
'= (CompareFileTime.bas)
'=====
#include "Windows.bi"

#define vbCrLf (Chr$(13) & Chr$(10))      'キャリッジリターンとラインフィード(¥r¥n)

Type FILETIME
    dwLowDateTime    As Long
    dwHighDateTime   As Long
End Type

Type SYSTEMTIME
    wYear            As Integer
    wMonth           As Integer
    wDayOfWeek       As Integer
    wDay             As Integer
    wHour            As Integer
    wMinute          As Integer
    wSecond          As Integer
    wMilliseconds    As Integer
End Type

' 現在の世界協定時刻の取得
Declare Sub Api_GetSystemTime Lib "kernel32" Alias "GetSystemTime" (lpSystemTime As
SYSTEMTIME)

' ローカルファイルタイムをUTCファイルタイム形式で取得
Declare Function Api_LocalFileTimeToFileTime& Lib "kernel32" Alias
"LocalFileTimeToFileTime" (lpLocalFileTime As FILETIME, lpFileTime As FILETIME)

' システムタイムをファイルタイムに変換
Declare Function Api_SystemTimeToFileTime& Lib "kernel32" Alias "SystemTimeToFileTime"
(lpSystemTime As SYSTEMTIME, lpFileTime As FILETIME)

' ファイルのファイルタイムを比較
Declare Function Api_CompareFileTime& Lib "kernel32" Alias "CompareFileTime"
(lpFileTime1 As FILETIME, lpFileTime2 As FILETIME)

' ファイルタイムをシステムファイルに変換
Declare Function Api_FileTimeToSystemTime& Lib "kernel32" Alias "FileTimeToSystemTime"
(lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME)

' 世界協定時刻形式ファイル時間をローカルファイル時間に変換
Declare Function Api_FileTimeToLocalFileTime& Lib "kernel32" Alias
"FileTimeToLocalFileTime" (lpFileTime As FILETIME, lpLocalFileTime As FILETIME)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Timer1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Timer1.Attach GetDlgItem("Timer1")

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
```

```

        Timer1.SetInterval 100
        Timer1.Enable -1
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var LocalTime As FILETIME
    Var GMTTime As FILETIME
    Var STimeLocal As SYSTEMTIME
    Var STimeGMT As SYSTEMTIME
    Var TmpTime As FILETIME
    Var txt As String
    Var Ret As Long

    Api_GetSystemTime STimeLocal

    Ret = Api_SystemTimeToFileTime(STimeLocal, TmpTime)

    Ret = Api_FileTimeToLocalFileTime(TmpTime, LocalTime)

    Ret = Api_LocalFileTimeToFileTime(LocalTime, GMTTime)
    Ret = Api_FileTimeToSystemTime(GMTTime, STimeGMT)
    Ret = Api_FileTimeToSystemTime(LocalTime, STimeLocal)

    Ret = Api_CompareFileTime(LocalTime, GMTTime)

    Select Case Ret
        Case -1
            Text2.SetWindowText "ローカル < GMT"
        Case 0
            Text2.SetWindowText "ローカル = GMT"
        Case 1
            Text2.SetWindowText "ローカル > GMT"
    End Select

    txt = Format$(STimeLocal.wHour, "## : ") & Format$(STimeLocal.wMinute, "## : ") &
    Format$(STimeLocal.wSecond, "## ") & "(ローカル)"
    txt = txt & vbCrLf & Format$(STimeGMT.wHour, "## : ") & Format$(STimeGMT.wMinute, "## : ") &
    Format$(STimeGMT.wSecond, "## ") & "(GMT)"
    Text1.SetWindowText txt
End Sub

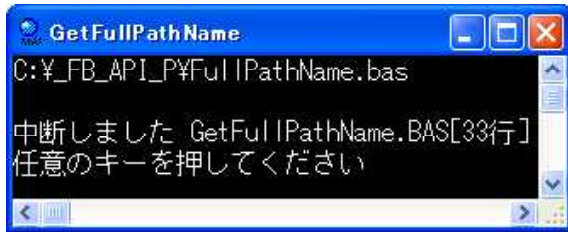
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルのフルパス名を取得

ファイルのフルパス名を取得します。
GetFullPathName ファイルのフルパス名を取得

現在のカレントパスを付加したファイル名を作成するもので、指定したファイル名が存在しない場合も、現在のカレントパスを付加したファイル名を作成します。



```
'=====
' = ファイルのフルパス名を取得
' = (現在のカレントパスを付加したファイル名を作成)
' = (GetFullPathName.bas)
'=====

' ファイルのフルパス名を取得
Declare Function Api_GetFullPathName& Lib "kernel32" Alias "GetFullPathNameA" (ByVal
lpFileName$, ByVal nBufferLength&, ByVal lpBuffer$, ByVal lpFilePart&)

'=====
' = Null[Chr$(0)]を取り除く
'=====
Declare Function TrimNull (item As String) As String
Function TrimNull(item As String) As String
    Var ePos As Integer

    ePos = InStr(item, Chr$(0))
    If ePos Then
        TrimNull = Left$(item, ePos - 1)
    Else
        TrimNull = item
    End If
End Function

'-----

Var FileName As String
Var FullPathName As String * 256
Var Ret As Long

FileName = "FullPathName.bas"

' フルパス名とファイル名を取得
Ret = Api_GetFullPathName(FileName, len(FullPathName), FullPathName, 0)

Print TrimNull(FullPathName)

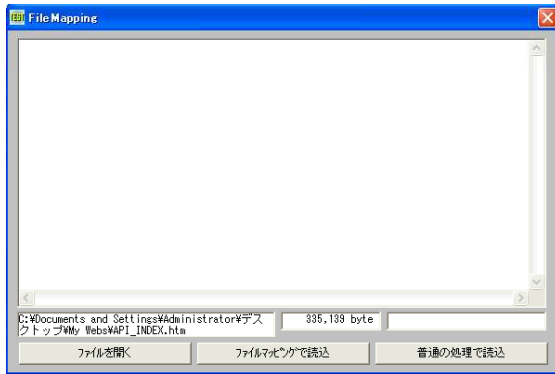
Stop
End
```

ファイル読込の速度比較

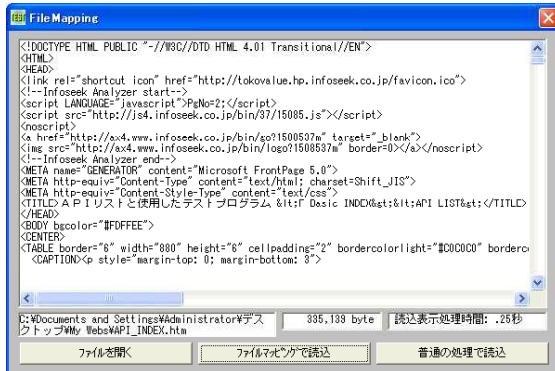
ファイルの読込速度 (通常のテキストファイル読込と、ファイルマッピングを使った読込) を比較します。

GetFileTitle パスからファイル名を取得
GetFileSize ファイルのサイズをバイト単位で取得
CloseHandle オープンされているオブジェクトハンドルをクローズ
CreateFile 指定したファイルをオープンし、デバイスハンドルを返す (ファイルサイズ取得用)
CreateFileLong 指定したファイルをオープンし、デバイスハンドルを返す (マッピング読込用)
CreateFileMapping ファイルマッピングオブジェクトを作成
MapViewOfFile ファイルビューをマッピング
OpenFileMapping 名前付きファイルマッピングオブジェクトをオープン
UnmapViewOfFile ファイルのマッピングビューをアンマップ
GetTickCount システムが起動してからの経過時間を取得
MoveMemory メモリの指定領域をコピー

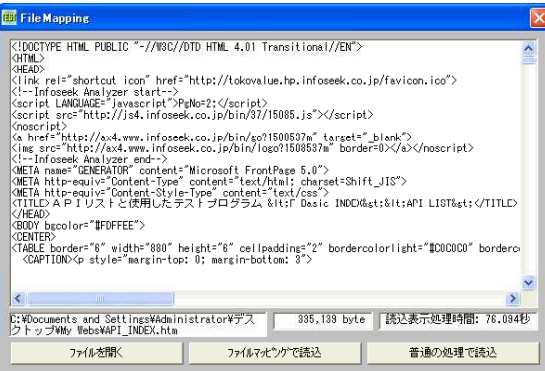
Text1にドラッグ & ドロップ、または「ファイルを開く」でテキストファイルを選択します。それぞれの読込ボタンでその処理時間を取得しています。



ファイルマッピングで読込



普通の処理で読込



```
'=====
'= ファイル読込の速度比較
'= (OpenFileMapping.bas)
'=====
```

```
#include "Windows.bi"

#define FILE_MAP_COPY &H1
#define FILE_MAP_WRITE &H2
#define FILE_MAP_READ &H4

#define PAGE_READONLY &H2
#define PAGE_READWRITE &H4

#define PAGE_WRITECOPY &H8

#define GENERIC_READ -2147483648
#define GENERIC_WRITE &H40000000

#define INVALID_HANDLE_VALUE -1

#define OPEN_EXISTING 3
#define OPEN_ALWAYS 4
#define vbCrLf (Chr$(13) & Chr$(10))
```

' セキュリティを定義する構造体

```
Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type
```

' パスからファイル名を取得

```
Declare Function Api_GetFileTitle% Lib "comdlg32" Alias "GetFileTitleA" (ByVal lpzFile$, ByVal lpzTitle$, ByVal cbBuf%)
```

' ファイルのサイズをバイト単位で取得

```
Declare Function Api_GetFileSize& Lib "kernel32" Alias "GetFileSize" (ByVal hFile&, lpFileSizeHigh&)
```

' オープンされているオブジェクトハンドルをクローズ

```
Declare Function Api_CloseHandle& Lib "kernel32" Alias "CloseHandle" (ByVal hObject&)
```

' 上書きモード
' 読み書きモード
' 読み込み専用モード

' コミット済みのページ領域に対する読み取りアクセスを許可
' コミット済みのページ領域に対する読み取りアクセスと書き込みアクセスの両方を許可
' コピー可能アクセス (hFileは、GENERIC_READ属性を持たなければならない)

' 読み込みモード (&H80000000)
' 書き込みモード

' 見つからない場合

' ファイルをオープンする (存在しない場合失敗)
' ファイルをオープンする (存在しない場合作成)
' キャリッジリターンとラインフィード (¥r¥n)

```

' 指定したファイルをオープンし、デバイスハンドルを返す (ファイルサイズ取得用)
Declare Function Api_CreateFile& Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName$,
ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As
SECURITY_ATTRIBUTES, ByVal dwCreationDisposition&, ByVal dwFlagsAttributes&, ByVal
hTemplateFile&)

' 指定したファイルをオープンし、デバイスハンドルを返す (マッピング読込用)
Declare Function Api_CreateFileLong& Lib "kernel32" Alias "CreateFileA" (ByVal
lpFileName$, ByVal dwDesiredAccess&, ByVal dwShareMode&, ByVal lpSecurityAttributes&,
ByVal dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)

' ファイルマッピングオブジェクトを作成
Declare Function Api_CreateFileMapping& Lib "kernel32" Alias "CreateFileMappingA" (ByVal
hFile&, ByVal lpFileMappigAttributes&, ByVal flProtect&, ByVal dwMaximumSizeHigh&, ByVal
dwMaximumSizeLow&, ByVal lpName$)

' ファイルビューをマッピング
Declare Function Api_MapViewOfFile& Lib "kernel32" Alias "MapViewOfFile" (ByVal
hFileMappingObject&, ByVal dwDesiredAccess&, ByVal dwFileOffsetHigh&, ByVal
dwFileOffsetLow&, ByVal dwNumberOfBytesToMap&)

' 名前付きファイルマッピングオブジェクトをオープン
Declare Function Api_OpenFileMapping& Lib "kernel32" Alias "OpenFileMappingA" (ByVal
dwDesiredAccess&, ByVal bInheritHandle&, ByVal lpName$)

' ファイルのマッピングビューをアンマップ
Declare Function Api_UnmapViewOfFile& Lib "kernel32" Alias "UnmapViewOfFile" (ByVal
lpBaseAddress&)

' システムが起動してからの経過時間を取得
Declare Function Api_GetTickCount& Lib "kernel32" Alias "GetTickCount" ()

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (Dest As Any, ByVal Source&,
ByVal length&)

Var Shared Edit1 As Object
Var Shared Text(2) As Object
Var Shared Button(2) As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12
For i = 0 To 2
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 12
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1))) : Button(i).SetFontSize 12
Next

Var Shared PathName As String
Var Shared FileLength As Long
Var Shared sFile As Long

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.EnableWindow 0
    Button(1).EnableWindow 0
    Button(2).EnableWindow 0
End Sub

' =====
' = ファイルマッピングを用いてファイルを高速に読み込む
' =====
Declare Function FileReadByFileMapping(hFileName As String) As String
Function FileReadByFileMapping(hFileName As String) As String
    Var hFileMap As Long
    Var hFile As Long
    Var hOpened As Long
    Var nAddress As Long
    Var buffer As String

```

```

Var Buffer2 As String
Var Ret As Long

hFile = Api_CreateFileLong(hFileName, GENERIC_READ, 0, 0, OPEN_EXISTING, 0, 0)
hFileMap = Api_CreateFileMapping(hFile, 0, PAGE_READONLY, 0, 0, "test")
If hFileMap = 0 Then Goto *ReadEnd

hOpened = Api_OpenFileMapping(FILE_MAP_READ, 0, "test")
If hOpened = 0 Then Goto *ReadEnd

nAddress = Api_MapViewOfFile(hOpened, FILE_MAP_READ, 0, 0, 0)
If nAddress = 0 Then Goto *ReadEnd

buffer = String$(FileLength, Chr$(0))
MoveMemory buffer, nAddress, FileLength
Buffer2 = Left$(buffer, InStr(buffer, Chr$(0)))
Ret = Api_UnmapViewOfFile(nAddress)

*ReadEnd
Ret = Api_CloseHandle(hOpened)
Ret = Api_CloseHandle(hFileMap)
Ret = Api_CloseHandle(hFile)

FileReadByFileMapping = Buffer2
End Function

'=====
'= ファイルサイズ取得
'=====
Declare Sub FileSizeGet edec1 ()
Sub FileSizeGet ()
    Var hFileName As Long
    Var sa As SECURITY_ATTRIBUTES
    Var FileSizeLow As Long
    Var FileSizeHigh As Long
    Var Ret As Long

    Edit1.SetWindowtext ""
    Text(1).SetWindowtext ""
    Text(2).SetWindowtext ""

    'セキュリティ構造体を初期化
    sa.nLength = Len(sa)

    'ファイルハンドル取得
    hFileName = Api_CreateFile(PathName, GENERIC_READ, 0, sa, OPEN_EXISTING, 0, 0)

    'ファイルハンドルの値が有効(見つかったとき)
    If hFileName <> INVALID_HANDLE_VALUE Then

        'ファイルサイズを取得
        FileSizeLow = Api_GetFileSize(hFileName, FileSizeHigh)

        'ファイルサイズの取得に失敗したときは
        If FileSizeLow = &HFFFFFFF Then

            '拡張エラー情報を表示
            A% = MsgBox("", "取得に失敗しました。", 0, 2)
            Exit Sub
        Else

            'ファイルサイズ
            FileLength = FileSizeLow + 1
        End If

        'ファイルをクローズ
        Ret = Api_CloseHandle(hFileName)

    '有効でないとき
    Else

```



```

        'エラーを表示
        A% = MessageBox("", "ファイルを開けませんでした。", 0, 2)
    End If
End Sub

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Text1_DropFiles edec1 (ByVal DF As Long)
Sub Text1_DropFiles (ByVal DF As Long)
    Var CN As Long

    Edit1.EnableWindow -1
    Button(1).EnableWindow -1
    Button(2).EnableWindow -1

    CN = GetDropFileCount (DF)
    PathName = GetDropFileName (DF, 0)
    Text(0).SetWindowText PathName

    'ファイルサイズ取得
    FileSizeGet
    Text(1).SetWindowText format$(FileLength - 1, "###,###,### byte ")
End Sub

'=====
'= ファイルを開く
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Buffer As String
    Var Ret As Integer

    Edit1.SetWindowText ""
    Text(0).SetWindowText ""
    Text(1).SetWindowText ""
    Text(2).SetWindowText ""

    PathName = WinOpenDlg("ファイルのオープン", "*.txt;*.csv;*.bas", "テキストファイル (*.txt);csvファイル (*.csv);basファイル (*.bas)", 0)
    If PathName <> Chr$(&H1B) Then
        Edit1.EnableWindow -1
        Button(1).EnableWindow -1
        Button(2).EnableWindow -1

        Buffer = String$(255, 0)
        Ret = Api_GetFileTitle(PathName, Buffer, Len(Buffer))
        Buffer = Left$(Buffer, InStr(1, Buffer, Chr$(0)) - 1)
        Text(0).SetWindowText " " & Buffer

        'ファイルサイズ取得
        FileSizeGet
        Text(1).SetWindowText format$(FileLength - 1, "###,###,### byte ")
    Else
        Edit1.EnableWindow 0
        Button(1).EnableWindow 0
        Button(2).EnableWindow 0
    End If
End Sub

'=====
'= ファイルマッピングによる読み込み
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var t As Long
    Var buffer As String

    Edit1.SetWindowText ""
    Text(2).SetWindowText ""

```

```

'計測開始
SetMousePointer 2
t = Api_GetTickCount

buffer = FileReadByFileMapping(PathName)

Edit1.SetWindowText buffer

'計測終了
SetMousePointer 0
Text(2).SetWindowText " 読込表示処理時間:" & Str$( (Api_GetTickCount - t) / 1000) & "秒"
End Sub

'=====
'= 通常の読込
'=====
Declare Sub Button3_on edecl ()
Sub Button3_on()
    Var ff As Integer
    Var t As Long
    Var txt As String
    Var temp As String

    Edit1.SetWindowText ""
    Text(2).SetWindowText ""

    '計測開始
    SetMousePointer 2
    t = Api_GetTickCount

    ff = FreeFile
    Open PathName For Input As #ff
    Do While Not Eof(#ff)
        Line Input #ff, temp
        txt = txt & temp & vbCrLf
    Loop
    Close #ff

    Edit1.SetWindowText txt

    '計測終了
    SetMousePointer 0
    Text(2).SetWindowText " 読込表示処理時間:" & Str$( (Api_GetTickCount - t) / 1000) & "秒"
End Sub

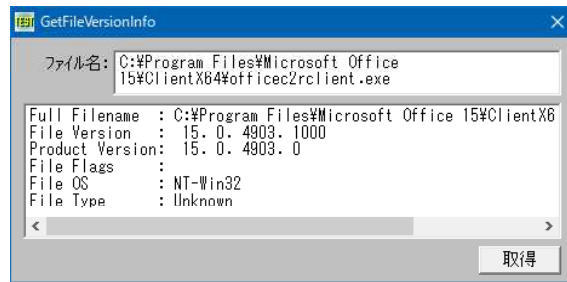
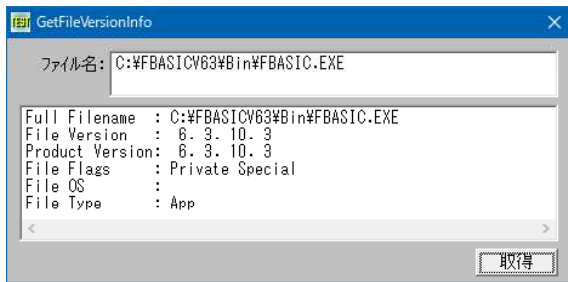
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルバージョン情報の取得

GetFileVersionInfo ファイルのバージョン情報の取得
GetFileVersionInfoSize ファイルからバージョン情報を取得できるかどうかを判定
VerQueryValue バージョン情報リソースから選択されたバージョン情報を取得

ファイルを Drag&Drop し、ファイルのバージョン情報を取得します。



```
'=====
'= ファイルバージョン情報の取得
'= (GetFileVersionInfo2.bas)
'=====

#include "Windows.bi"
```

```
#define VS_FFI_SIGNATURE &HFEF04BD
#define VS_FFI_STRUCVERSION &H10000
#define VS_FFI_FILEFLAGSMASK &H3F&
' dwFileFlags
#define VS_FF_DEBUG &H1
#define VS_FF_PRERELEASE &H2
#define VS_FF_PATCHED &H4
#define VS_FF_PRIVATEBUILD &H8
#define VS_FF_INFOINFERRED &H10
#define VS_FF_SPECIALBUILD &H20
' dwFileOS
#define VOS_UNKNOWN &H0
#define VOS_DOS &H10000
#define VOS_OS216 &H20000
#define VOS_OS232 &H30000
#define VOS_NT &H40000
#define VOS__BASE &H0
#define VOS__WINDOWS16 &H1
#define VOS__PM16 &H2
#define VOS__PM32 &H3
#define VOS__WINDOWS32 &H4
#define VOS_DOS_WINDOWS16 &H10001
#define VOS_DOS_WINDOWS32 &H10004
#define VOS_OS216_PM16 &H20002
#define VOS_OS232_PM32 &H30003
#define VOS_NT_WINDOWS32 &H40004
' dwFileType
#define VFT_UNKNOWN &H0
#define VFT_APP &H1
#define VFT_DLL &H2
#define VFT_DRV &H3
#define VFT_FONT &H4
#define VFT_VXD &H5
#define VFT_STATIC_Lib &H7
' dwFileSubtype
#define VFT2_UNKNOWN &H0
#define VFT2_DRV_PRINTER &H1
#define VFT2_DRV_KEYBOARD &H2
#define VFT2_DRV_LANGUAGE &H3
#define VFT2_DRV_DISPLAY &H4
#define VFT2_DRV_MOUSE &H5
#define VFT2_DRV_NETWORK &H6
#define VFT2_DRV_SYSTEM &H7
#define VFT2_DRV_INSTALLABLE &H8
#define VFT2_DRV_SOUND &H9
#define VFT2_DRV_COMM &HA
```

```
Type VS_FIXEDFILEINFO
    dwSignature           As Long
    dwStrucVersion1       As Integer
    dwStrucVersionh       As Integer
    dwFileVersionMS1      As Integer
    dwFileVersionMSH      As Integer
```

```
' 構造体の識別子
' 構造体のバイナリバージョン番号 0x00000029以上の値
'
' デバッグ情報が含まれている
' ファイルは開発バージョン
' ファイルは修正されたバージョン
' 標準的なリリースされたファイルではない
' 構造体は動的に作成されている
' 標準ファイルのバリエーション
'
' MS-DOS
' OS/2 16ビット
' OS/2 32ビット
' WindowsNT
'
' in16
' Win32s
'
' 設計対象となったOS
'
' アプリケーション
' ダイナミックリンクライブラリ
' デバイスドライバ
' フォント
' 仮想デバイス
' スタティックリンクライブラリ
'
' プリンタドライバ
' キーボードドライバ
' 言語ドライバ
' ディスプレイドライバ
' マウスドライバ
' ネットワークドライバ
' ファイルはシステムドライバ
' インストール可能なドライバ
' サウンドドライバ
```

```

dwFileVersionLS1      As Integer
dwFileVersionLSH      As Integer
dwProductVersionMS1  As Integer
dwProductVersionMSH  As Integer
dwProductVersionLS1  As Integer
dwProductVersionLSH  As Integer
dwFileFlagsMask      As Long
dwFileFlags           As Long
dwFileOS              As Long
dwFileType            As Long
dwFileSubType         As Long
dwFileDateMS          As Long
dwFileDateLS          As Long
End Type

```

' ファイルに関するバージョン情報を取得

```

Declare Function Api_GetFileVersionInfo& Lib "Version" Alias "GetFileVersionInfoA"
(ByVal lptstrFilename$, ByVal dwhandle&, ByVal dwlen&, lpData As Any)

```

' ファイルからバージョン情報を取得できるかどうかを判定

```

Declare Function Api_GetFileVersionInfoSize& Lib "Version" Alias
"GetFileVersionInfoSizeA" (ByVal lptstrFilename$, lpdwHHandle&)

```

' バージョン情報リソースから選択されたバージョン情報を取得

```

Declare Function Api_VerQueryValue& Lib "Version" Alias "VerQueryValueA" (pBlock As Any,
ByVal lpSubBlock$, lpBuffer As Any, puLen&)

```

' メモリの指定領域をコピー

```

Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, ByVal Source&,
ByVal length&)
Var Shared Button1 As Object
Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Edit2 As Object

```

```

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

```

Var Shared Filename As String
Var Shared Directory As String
Var Shared FullFileName As String
Var Shared StrucVer As String
Var Shared FileVer As String
Var Shared ProdVer As String
Var Shared FileFlags As String
Var Shared FileOS As String
Var Shared FileType As String
Var Shared FileSubType As String

```

```

' =====
' =
' =====

```

```

Declare Sub DisplayVerInfo edec1 ()
Sub DisplayVerInfo()

```

```

    Var lDummy As Long
    Var BuffLen As Long
    Var VerPointer As Long
    Var VerBuff As VS_FIXEDFILEINFO
    Var VerBuffLen As Long
    Var Ret As Long

```

```

    BuffLen = Api_GetFileVersionInfoSize(Filename, lDummy)
    If BuffLen < 1 Then
        Res = MsgBox("", "バージョン情報は有りません！", 0, 2)
        Exit Sub
    End If

```

```

    Var sBuffer(BuffLen) As Byte

```

```

Ret = Api_GetFileVersionInfo(FileName, 0&, BuffLen, sBuffer(0))
Ret = Api_VerQueryValue(sBuffer(0), "¥", VerPointer, VerBuffLen)
MoveMemory VerBuff, VerPointer, Len(VerBuff)

```

```

StrucVer = Format$(VerBuff.dwStrucVersionh) & "." & Format$(VerBuff.dwStrucVersionl)
FileVer = Format$(VerBuff.dwFileVersionMSh) & "." &
Format$(VerBuff.dwFileVersionMSl) & "." & Format$(VerBuff.dwFileVersionLSh) & "." &
Format$(VerBuff.dwFileVersionLSl)
ProdVer = Format$(VerBuff.dwProductVersionMSh) & "." &
Format$(VerBuff.dwProductVersionMSl) & "." & Format$(VerBuff.dwProductVersionLSh) & "." &
Format$(VerBuff.dwProductVersionLSl)

```

```

FileFlags = ""
If VerBuff.dwFileFlags And VS_FF_DEBUG Then FileFlags = "Debug "
If VerBuff.dwFileFlags And VS_FF_PRERELEASE Then FileFlags = FileFlags & "PreRel "
If VerBuff.dwFileFlags And VS_FF_PATCHED Then FileFlags = FileFlags & "Patched "
If VerBuff.dwFileFlags And VS_FF_PRIVATEBUILD Then FileFlags = FileFlags & "Private "
If VerBuff.dwFileFlags And VS_FF_INFOINFERRE Then FileFlags = FileFlags & "Info "
If VerBuff.dwFileFlags And VS_FF_SPECIALBUILD Then FileFlags = FileFlags & "Special "
If VerBuff.dwFileFlags And VFT2_UNKNOWN Then FileFlags = FileFlags & "Unknown "

```

```

Select Case VerBuff.dwFileOS
Case VOS_DOS_WINDOWS16
FileOS = "DOS-Win16"
Case VOS_DOS_WINDOWS32
FileOS = "DOS-Win32"
Case VOS_OS216_PM16
FileOS = "OS/2-16 PM-16"
Case VOS_OS232_PM32
FileOS = "OS/2-16 PM-32"
Case VOS_NT_WINDOWS32
FileOS = "NT-Win32"
Case other
FileOS = "Unknown"
End Select

```

```

Select Case VerBuff.dwFileType
Case VFT_APP
FileType = "App"
Case VFT_DLL
FileType = "DLL"
Case VFT_DRV
FileType = "Driver"
Select Case VerBuff.dwFileSubtype
Case VFT2_DRV_PRINTER
FileSubType = "Printer drv"
Case VFT2_DRV_KEYBOARD
FileSubType = "Keyboard drv"
Case VFT2_DRV_LANGUAGE
FileSubType = "Language drv"
Case VFT2_DRV_DISPLAY
FileSubType = "Display drv"
Case VFT2_DRV_MOUSE
FileSubType = "Mouse drv"
Case VFT2_DRV_NETWORK
FileSubType = "Network drv"
Case VFT2_DRV_SYSTEM
FileSubType = "System drv"
Case VFT2_DRV_INSTALLABLE
FileSubType = "Installable"
Case VFT2_DRV_SOUND
FileSubType = "Sound drv"
Case VFT2_DRV_COMM
FileSubType = "Comm drv"
Case VFT2_UNKNOWN
FileSubType = "Unknown"
End Select
Case VFT_FONT
FileType = "Font"
Select Case VerBuff.dwFileSubtype

```

```

        Case VFT_FONT_RASTER
            FileSubType = "Raster Font"
        Case VFT_FONT_VECTOR
            FileSubType = "Vector Font"
        Case VFT_FONT_TRUETYPE
            FileSubType = "TrueType Font"
    End Select
    Case VFT_VXD
        FileType = "VxD"
    Case VFT_STATIC_Lib
        FileType = "Lib"
    Case Else
        FileType = "Unknown"
    End Select
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    FileName = Edit2.GetWindowText

    FileVer = ""
    ProdVer = ""
    FileFlags = ""
    FileOS = ""
    FileType = ""

    DisplayVerInfo
    txt$ = txt$ & "Full Filename   : " & FileName & Chr$(13,10)
    txt$ = txt$ & "File Version    : " & FileVer & Chr$(13,10)
    txt$ = txt$ & "Product Version: " & ProdVer & Chr$(13,10)
    txt$ = txt$ & "File Flags      : " & FileFlags & Chr$(13,10)
    txt$ = txt$ & "File OS        : " & FileOS & Chr$(13,10)

    If FileSubType = "" Then
        txt2$ = ""
    Else
        txt2$ = " (" & FileSubType & ")"
    End If

    txt$ = txt$ & "File Type      : " & FileType & txt2$
    Edit1.SetWindowText txt$
End Sub

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit2_DropFiles edec1 (ByVal DF As Long)
Sub Edit2_DropFiles (ByVal DF As Long)
    Var CN As Long

    CN = GetDropFileCount (DF)
    FileName = GetDropFileName (DF, 0)
    Edit2.SetWindowText FileName
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイルバージョン・製品バージョン取得

ファイルバージョンと製品バージョンを取得します。

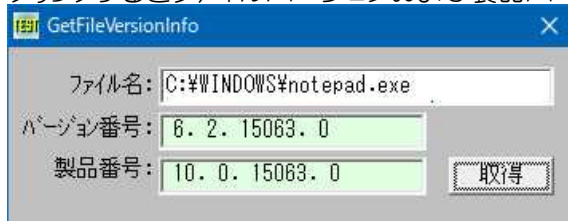
GetFileVersionInfo ファイルに関するバージョン情報を取得

GetFileVersionInfoSize ファイルからバージョン情報を取得できるかどうかを判定

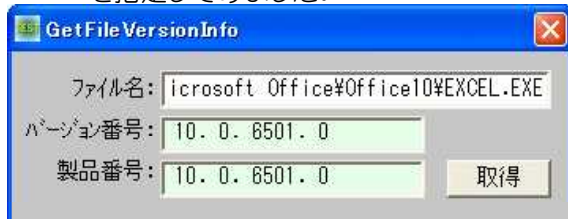
VerQueryValue バージョン情報リソースから選択されたバージョン情報を取得

RtlMoveMemory ある位置から別の位置にメモリーブロックを移動

初期状態では、既知のnotepad.exeファイルをしていしています。ファイル名欄に、フルパスを指定し『取得』ボタンをクリックするとファイルバージョンおよび製品バージョンが表示されます。



EXCELを指定してみました。



```
'=====
'= ファイル・製品バージョン取得
'= (GetFileVersionInfo.bas)
'=====
#include "Windows.bi"
```

```
Type VS_FIXEDFILEINFO
    dwSignature           As Long
    dwStrucVersion        As Long
    dwFileVersionMS       As Long
    dwFileVersionLS       As Long
    dwProductVersionMS    As Long
    dwProductVersionLS    As Long
    dwFileFlagsMask       As Long
    dwFileFlags           As Long
    dwFileOS              As Long
    dwFileType            As Long
    dwFileSubType         As Long
    dwFileDateMS          As Long
    dwFileDateLS          As Long
End Type
```

' ファイルに関するバージョン情報を取得

```
Declare Function Api_GetFileVersionInfo& Lib "Version" Alias "GetFileVersionInfoA"
(ByVal lptstrFilename$, ByVal dwhandle&, ByVal dwlen&, lpData As Any)
```

' ファイルからバージョン情報を取得できるかどうかを判定

```
Declare Function Api_GetFileVersionInfoSize& Lib "Version" Alias
"GetFileVersionInfoSizeA" (ByVal lptstrFilename$, lpdwHandle&)
```

' バージョン情報リソースから選択されたバージョン情報を取得

```
Declare Function Api_VerQueryValue& Lib "Version" Alias "VerQueryValueA" (pBlock As Any,
ByVal lpSubBlock$, lpBuffer As Any, puLen&)
```

' メモリの指定領域をコピー

```
Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (dest As Any, ByVal Source&,
ByVal length&)
```

```
Var Shared Edit1 As Object
Var Shared Text(4) As Object
Var Shared Button1 As Object
```

```

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
For i = 0 To 4
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
Next i

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var FileName As String
    Var SizeVerInfo As Long
    Var DmyHandle As Long
    Var PtrVerInfo As Long
    Var LenVerInfo As Long
    Var FileInfo As VS_FIXEDFILEINFO
    Var FileVersion As String
    Var ProductVersion As String
    Var Ret As Long

    FileName = GetDlgItemText("Edit1") '対象ファイルを設定
    SizeVerInfo = Api_GetFileVersionInfoSize(FileName, DmyHandle) 'バージョン情報のサイズを取得
    'バージョン情報のサイズを取得成功
    If SizeVerInfo > 0 Then
        Var DmyVerInfo(SizeVerInfo - 1) 'ダミー変数の領域を確保
        Ret = Api_GetFileVersionInfo(FileName, 0, SizeVerInfo, DmyVerInfo(0)) 'バージョン情報を取得
        Ret = Api_VerQueryValue(DmyVerInfo(0), "¥", PtrVerInfo, LenVerInfo) 'バージョン情報リソースのルートブロックを取得
        MoveMemory FileInfo, ByVal PtrVerInfo, Len(FileInfo) '取得したルートブロックをバージョン情報の構造体へ移動

        'ファイルバージョンを取り出し
        FileVersion = Str$( (FileInfo.dwFileVersionMS ¥ 2 ^ 16) And &HFFFF) & "." &
Str$(FileInfo.dwFileVersionMS And &HFFFF) & "." & Str$( (FileInfo.dwFileVersionLS ¥ 2 ^ 16) And &HFFFF) & "." & Str$(FileInfo.dwFileVersionLS And &HFFFF)

        '製品バージョンを取り出し
        ProductVersion = Str$( (FileInfo.dwProductVersionMS ¥ 2 ^ 16) And &HFFFF) & "." &
Str$(FileInfo.dwProductVersionMS And &HFFFF) & "." & Str$( (FileInfo.dwProductVersionLS ¥ 2 ^ 16) And &HFFFF) & "." & Str$(FileInfo.dwProductVersionLS And &HFFFF)
        Text(3).SetWindowText FileVersion 'ファイルバージョンを表示
        Text(4).SetWindowText ProductVersion '製品バージョンを表示
    Else 'バージョン情報のサイズを取得失敗
        Text(3).SetWindowText "取得できません。"
        Text(4).SetWindowText "取得できません。"
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ファイル またはディレクトリを移動する(1)

MoveFile 既存のファイルを移動します。

第一引数:移動(名前変更)させたいファイルのパスを指定し、
第二引数:ファイルの移動先をパスで指定します。
ファイルの移動先が同じディレクトリの場合、
ファイルは移動せず、ファイルの名前が変更されます。


```

'=====
'= ファイルを移動する
'= (MoveFile.bas)
'=====
#include "Windows.bi"

' 既存のファイルを移動
Declare Function Api_MoveFile& Lib "Kernel32" Alias "MoveFileA" (ByVal
lpExistingFileName$, ByVal lpNewFileName$)

Var OldName As String
Var NewName As String
Var Ret As Long

OldName = "C:¥Temp¥Hoge.txt"
NewName = "C:¥Temp2¥Hoge.txt"

Ret = Api_MoveFile (OldName, NewName)

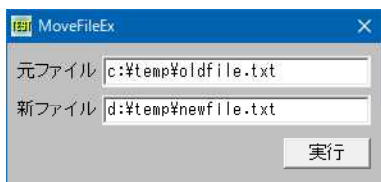
If Ret <> 0 Then
    Print "移動しました！"
Else
    Print "移動できません！"
End If

Stop : End

```

ファイルまたはディレクトリを移動(II)

MoveFileEx 既存のファイルまたはディレクトリを移動



```

'=====
'= ファイルまたはディレクトリを移動(II)
'= (MoveFileEx.bas)
'=====
#include "Windows.bi"

' 既存のファイルまたはディレクトリを移動
Declare Function Api_MoveFileEx& Lib "kernel32" Alias "MoveFileExA" (ByVal
lpExistingFileName$, ByVal lpNewFileName$, ByVal dwFlags&)

#define MOVEFILE_REPLACE_EXISTING &H1
#define MOVEFILE_COPY_ALLOWED &H2
#define MOVEFILE_DELAY_UNTIL_REBOOT &H4
#define MOVEFILE_WRITE_THROUGH &H8

' 上書き可能にする
' 異なるボリューム間での移動を、ファイルの複製と削除を行
' うことによって可能にする
' システムの次回起動時に移動を実行する
' 移動が完全に終了するまで、MoveFileEx関数は終了しな
' い

Var Shared Edit1 As Object : Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Var Shared Edit2 As Object : Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Var Shared Text1 As Object : Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Var Shared Text2 As Object : Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Var Shared Button1 As Object : Button1.Attach GetDlgItem("Button1") :
Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Mainform_Start edec1 ()

```

```

Sub MainForm_Start()
    Edit1.SetWindowText "c:¥temp¥oldfile.txt"
    Edit2.SetWindowText "d:¥temp¥newfile.txt"
End Sub

'=====
'==
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var ExistingFile As String
    Var NewFile As String
    Var Ret As Long

    ExistingFile = Edit1.GetWindowText
    NewFile = Edit2.GetWindowText

    If ExistingFile = "" Or NewFile = "" Then Exit Sub

    Ret = Api_MoveFileEx(ExistingFile, NewFile, MOVEFILE_COPY_ALLOWED Or
MOVEFILE_REPLACE_EXISTING)
End Sub

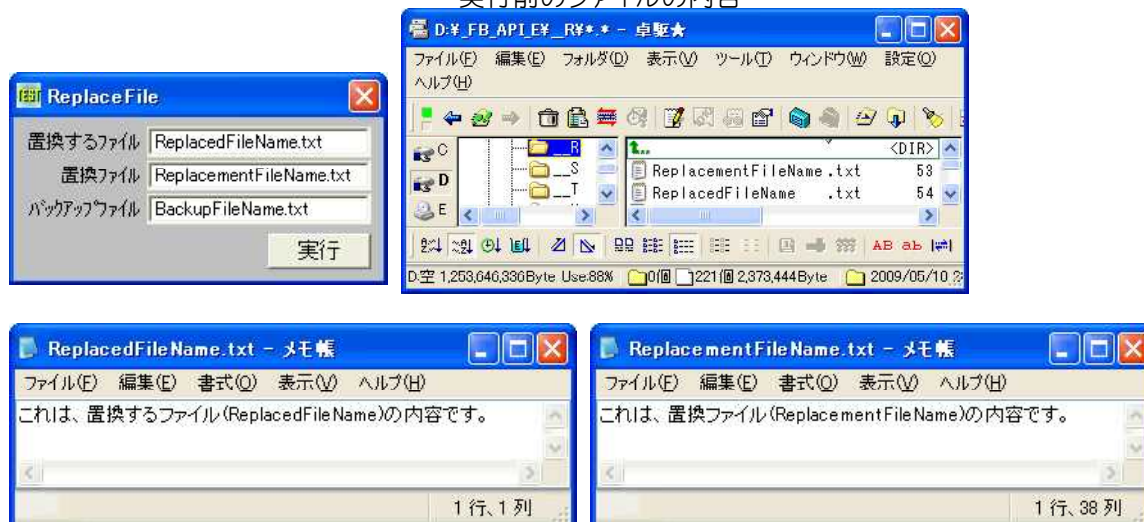
'=====
'==
'=====
While 1
    WaitEvent
Wend
Stop
End

```

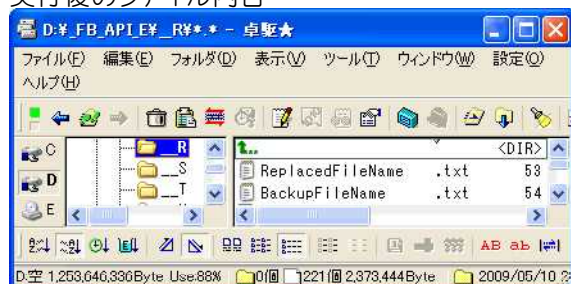
ファイル名の置換

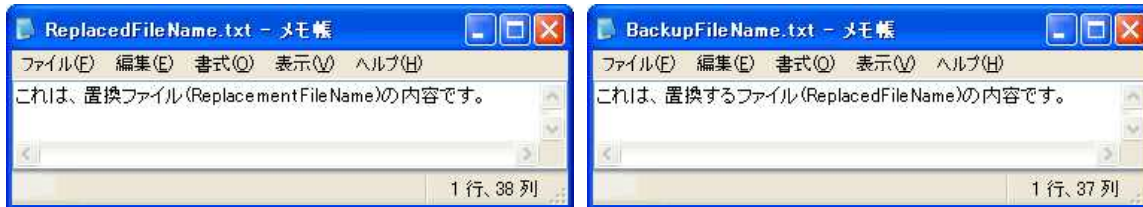
ReplaceFile ファイルを別のファイルに置換

実行前のファイルの内容



実行後のファイル内容





```
'=====
'= ファイル名の置換
'= (ReplaceFile.bas)
'=====
#include "Windows.bi"

' ファイルを別のファイルに置換
Declare Function Api_ReplaceFile& Lib "kernel32" Alias "ReplaceFileA" (ByVal
lpReplacedFileName$, ByVal lpReplacementFileName$, ByVal lpBackupFileName$, ByVal
dwReplaceFlags&, ByVal lpExclude&, ByVal lpReserved&)

#define REPLACEFILE_WRITE_THROUGH &H1
#define REPLACEFILE_IGNORE_MERGE_ERRORS &H2

Var Shared Text(2) As Object
Var SHared Edit(2) As Object
Var Shared Button1 As Object

For i = 0 To 2
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 12
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 12
Next i
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

ShowWindow -1

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sFile As String
    Var tFile As String
    Var bFile As String
    Var Ret As Long

    sFile = "ReplacedFileName.txt" & Chr$(0)
    tFile = "ReplacementFileName.txt" & Chr$(0)
    bFile = "BackupFileName.txt" & Chr$(0)

    Ret = Api_ReplaceFile(sFile,tFile, bFile, REPLACEFILE_WRITE_THROUGH, ByVal 0, ByVal
0)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

' ディスクフラッシュ強制を示す
' 置換されたファイルと置換ファイルの情報を併合
するときに発生するエラーを無視

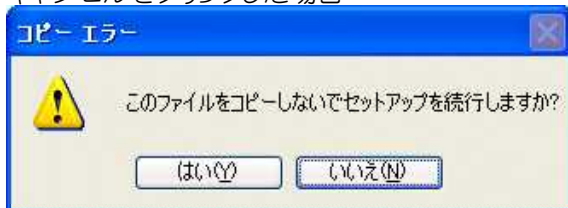
' 置換するファイル
' 置換ファイル
' バックアップファイル

ファイル名変更エラーダイアログボックス

SetupRenameError ファイルの名前変更エラーをユーザーへ通知するためのダイアログボックスを生成します。



キャンセルをクリックした場合



```
'=====
'= ファイル名変更エラーダイアログボックス
'= (SetupRenameError.bas)
'=====
#include "Windows.bi"

'ファイルの名前変更エラーをユーザーへ通知するためのダイアログボックスを生成
Declare Function Api_SetupRenameError Lib "setupapi" Alias "SetupRenameErrorA" (ByVal
hwndParent&, ByVal DialogTitle$, ByVal SourceFile$, ByVal TargetFile$, ByVal
Win32ErrorCode&, ByVal Style&)

Var Shared Button1 As Object
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Ret As Long

    Ret = Api_SetupRenameError(GetHwnd, "Rename Error", "file.ex_", "file.ext", 123, 0)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ファイルやフォルダの監視

指定したフォルダ (サブフォルダを含む) やファイルの追加・削除等の変更を通知します。

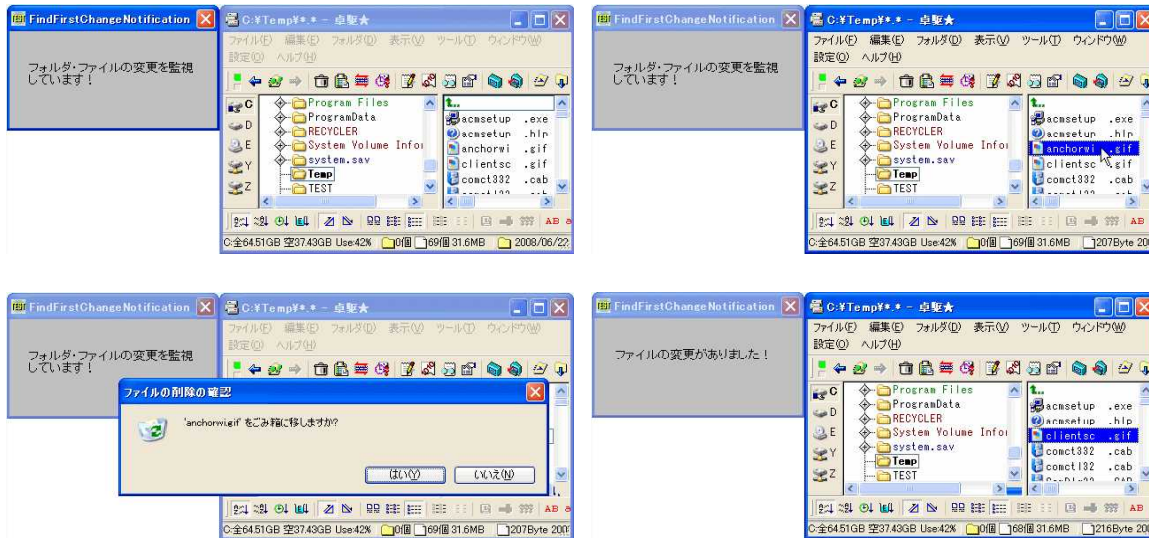
FindFirstChangeNotification ファイル変更通知オブジェクトのハンドルを作成

FindNextChangeNotification 次のファイル通知を要求

FindCloseChangeNotification 変更通知オブジェクトのハンドルを解放

WaitForMultipleObjects 複数のオブジェクトがシグナル状態になるか、一定時間を経過するまで待機

エクスプローラ等でフォルダおよびファイルの追加、削除を実行することにより下記のように表示されます。



```
'=====
'= フォルダ・ファイルの監視
'= (FindFirstChangeNotification.bas)
'=====
#include "Windows.bi"
```

' ファイル変更通知オブジェクトのハンドルを作成

```
Declare Function Api_FindFirstChangeNotification& Lib "kernel32" Alias
"FindFirstChangeNotificationA" (ByVal lpPathName$, ByVal bWatchSubtree&, ByVal
dwNotifyFilter&)
```

' 次のファイル変更通知を要求

```
Declare Function Api_FindNextChangeNotification& Lib "kernel32" Alias
"FindNextChangeNotification" (ByVal hChangeHandle&)
```

' 変更通知オブジェクトのハンドルを開放

```
Declare Function Api_FindCloseChangeNotification& Lib "kernel32" Alias
"FindCloseChangeNotification" (ByVal hChangeHandle&)
```

' 複数のオブジェクトがシグナル状態になるか、一定時間を経過するまで待機

```
Declare Function Api_WaitForMultipleObjects& Lib "kernel32" Alias
"WaitForMultipleObjects" (ByVal nCount&, lpHandles&, ByVal bWaitAll&, ByVal
dwMilliseconds&)
```

```
#define FILE_NOTIFY_CHANGE_ATTRIBUTES &H4 '属性の変更
#define FILE_NOTIFY_CHANGE_DIR_NAME &H2 'ディレクトリの追加・削除、ディレクトリ名の変更
#define FILE_NOTIFY_CHANGE_FILE_NAME &H1 'ファイルの追加・削除、ファイル名の変更
#define FILE_NOTIFY_CHANGE_LAST_WRITE &H10 '最終書き込み時刻の変更
#define FILE_NOTIFY_CHANGE_SECURITY &H100 'セキュリティ属性の変更
#define FILE_NOTIFY_CHANGE_SIZE &H8 'サイズの変更
```

```
#define INVALID_HANDLE_VALUE -1 '見つからない場合
#define INFINITE &HFFFFFF
```

```
#define WAIT_ABANDONED &H80
```

```
#define WAIT_FAILED &HFFFFFFF
#define WAIT_OBJECT_0 &H0
```

```
#define WAIT_TIMEOUT &H102
```

```
Var Shared flg As Integer
```

```
Var Shared Text1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
```

```
'=====
'=
'=====
```

'全オブジェクトを待っている場合には、すべてのオブジェクトがシグナル状態になり、その中に放棄されたためにシグナル状態になったミューテックスが含まれていることを示す
'エラーが発生したことを示す
'全オブジェクトを待っている場合には、すべてのオブジェクトがシグナル状態になったことを示す
'タイムアウト時間が経過したことを示す

```

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var hNotify(1) As Long
    Var WaitStatus As Long
    Var Ret As Long

    Text1.SetWindowText "フォルダ・ファイルの変更を監視しています！"

    'ファイルの追加・削除、ファイル名の変更
    hNotify(0) = Api_FindFirstChangeNotification("C:¥", True,
FILE_NOTIFY_CHANGE_FILE_NAME)
    If hNotify(0) = INVALID_HANDLE_VALUE Then
        Text1.SetWindowText "ファイルが見つかりません！"
        GoSub *Msg
        Exit Sub
    End If

    'ディレクトリの追加・削除、ディレクトリ名の変更の監視
    hNotify(1) = Api_FindFirstChangeNotification("C:¥", True,
FILE_NOTIFY_CHANGE_DIR_NAME)
    If hNotify(1) = INVALID_HANDLE_VALUE Then
        Text1.SetWindowText "ディレクトリが見つかりません！"
        GoSub *Msg
        Exit Sub
    End If

    Do
        CallEvent

        '指定のフォルダが変更されるまで待機
        WaitStatus = Api_WaitForMultipleObjects(2, hNotify(0), False, 10)

        Select Case WaitStatus
            Case WAIT_OBJECT_0 'すべてのオブジェクトがシグナル状態
                Text1.SetWindowText "ファイルの変更がありました！"
                GoSub *Msg

                'ファイル変更通知オブジェクトのハンドルの作成と解放
                Ret = Api_FindCloseChangeNotification(hNotify(0))
                hNotify(0) = Api_FindFirstChangeNotification("C:¥", True,
FILE_NOTIFY_CHANGE_FILE_NAME)

                Case WAIT_OBJECT_0 + 1 'WAIT_OBJECT_0 + nCount に設定した条件で復帰
                    Text1.SetWindowText "フォルダの変更がありました！"
                    GoSub *Msg

                    'フォルダ変更通知オブジェクトのハンドルの作成と解放
                    Ret = Api_FindCloseChangeNotification(hNotify(1))
                    hNotify(1) = Api_FindFirstChangeNotification("C:¥", True,
FILE_NOTIFY_CHANGE_DIR_NAME)
                End Select
            Loop Until flg = True

            'ハンドルの解放
            Ret = Api_FindCloseChangeNotification(hNotify(0))
            Ret = Api_FindCloseChangeNotification(hNotify(1))
            Exit Sub

        *Msg
        Wait 100
        Text1.SetWindowText "フォルダ・ファイルの変更を監視しています！"
        Return
    End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()
    flg = True

```

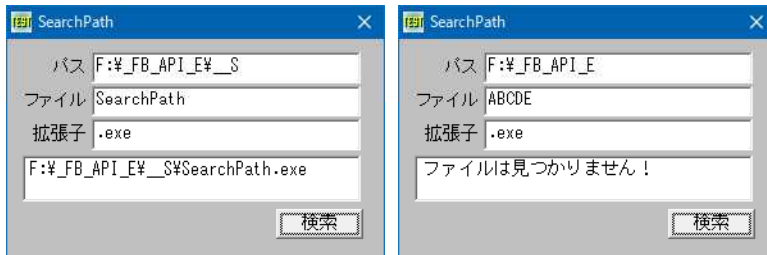


```
End Sub
```

```
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ファイルやフォルダの検索

ファイルやフォルダを検索します。
SearchPath ファイルやフォルダを検索



```
'=====
'= ファイルやフォルダの検索
'= (SearchPath.bas)
'=====
#include "Windows.bi"
```

ファイルやフォルダを検索する

```
Declare Function Api_SearchPath& Lib "Kernel32" Alias "SearchPathA" (ByVal lpPath$,
ByVal lpFileName$, ByVal lpExtension$, ByVal nBufferLength&, ByVal lpBuffer$, ByVal
lpFilePart$)
```

```
#define MAX_PATH 260
#define vbNullString ByVal 0 '値0の文字列。値0を持つ文字列。空文字列ではない
```

```
Var Shared Text(2) As Object
Var Shared Edit(3) As Object
Var Shared Button1 As Object
```

```
For i = 0 To 3
    If i < 3 Then Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) :
Text(i).SetFontSize 14
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Path As String
    Var FileName As String
    Var Extension As String
    Var FullPath As String
    Var Ret As Long

    Path = Edit(0).GetWindowText
    FileName = Edit(1).GetWindowText
    Extension = Edit(2).GetWindowText

    FullPath = Space$(MAX_PATH)
```

```

If Path = "" And Extension = "" Then
    Ret = Api_SearchPath(vbNullString, FileName, vbNullString, MAX_PATH, FullPath,
vbNullString)
Else
    Ret = Api_SearchPath(Path, FileName, Extension, MAX_PATH, FullPath, vbNullString)
End If

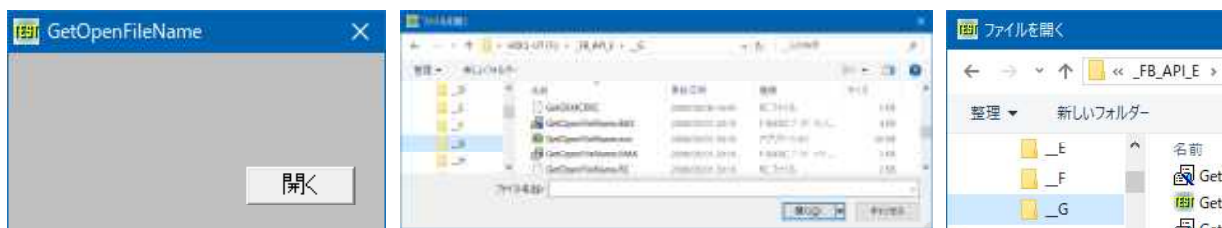
If Ret = 0 Then
    Edit(3).SetWindowText "ファイルは見つかりません！"
Else
    Edit(3).SetWindowText FullPath
End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

「ファイルを開く」ダイアログの作成

GetOpenFileName 「ファイルを開く」ダイアログを表示
lstrcat ある文字列の末尾に別の文字列を結合



```

'=====
'='ファイルを開く」ダイアログを表示
'=(GetOpenFileName.bas)
'=====
#include "Windows.bi"

```

Type OPENFILENAME

lStructSize As Long
hwndOwner As Long

hInstance As Long
lpstrFilter As Long

lpstrCustomFilter As Long

nMaxCustFilter As Long
nFilterIndex As Long
lpstrFile As Long
nMaxFile As Long
lpstrFileTitle As Long

nMaxFileTitle As Long
lpstrInitialDir As Long
lpstrTitle As Long

Flags As Long

nFileOffset As Integer

nFileExtension As Integer

'構造体のサイズをバイト単位で指定
'ダイアログボックスを所有するウィンドウへのハンドルを指定
'メモリオブジェクトへのハンドルを指定
'任意の数のNULL文字で終わる文字列のペアを保持するバッファへのポインタを指定
'ユーザーによって選択されたファイル フィルター文字列のペアが保存される
'lpstrCustomFilterメンバのバッファ サイズを指定
'フィルターの初期インデックスを指定
'文字列バッファを指定
'lpstrFileメンバのバッファ サイズを指定
'選択されたファイルの名前、および拡張子を保存するための文字列バッファを指定
'lpstrFileTitle メンバのバッファ サイズを指定
'初期ディレクトリのパスを指定
'ダイアログボックスのタイトルバーに表示する文字列を指定
'ダイアログボックスの初期化フラグを、「OFN_...」の定数を組み合わせて指定
'lpstrFileメンバが示すパス内の、ファイル名の位置がバッファの先頭から何バイト目であるかを示す
'lpstrFileメンバが示すパス内の、拡張子の位置がバッファ


```
lpstrDefExt      As Long
lCustData        As Long

lpfnHook         As Long
lpTemplateName  As Long
```

アの先頭から何バイト目であるかを示す
 'デフォルト拡張子の文字列を示す
 'lpfnHookメンバが示すフック プロシージャへ渡すデータを指定
 'フック プロシージャへのポインタを指定
 'hInstanceメンバが示すモジュール内にあるダイアログテンプレートを指定

```
End Type
```

' 「ファイルを開く」ダイアログボックスを作成

```
Declare Function Api_GetOpenFileName& Lib "comdlg32" Alias "GetOpenFileNameA"
  (pOpenfilename As OPENFILENAME)
```

' ある文字列の末尾に別の文字列を結合

```
Declare Function Api_lstrcat& Lib "kernel32" Alias "lstrcatA" (ByVal lpString1$, ByVal
lpString2$)
```

```
Var Shared Button1 As Object
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var ofn As OPENFILENAME
```

```
    Var Ret As Long
```

```
    ofn.lStructSize = Len(ofn)
```

```
    ofn.hwndOwner = GethWnd
```

```
    ofn.hInstance = GethInst
```

```
    ofn.lpstrFilter = Api_lstrcat("テキストファイル (*.txt)" & Chr$(0) & "*.txt" & Chr$(0) & "
全てのファイル (*.*)" & Chr$(0) & "*.*)" & Chr$(0), "")
```

```
    ofn.lpstrFile = Api_lstrcat(Space$(254) & Chr$(0), "")
```

```
    ofn.nMaxFile = 255
```

```
    ofn.lpstrFileName = StrAdr(Space$(254) & Chr$(0))
```

```
    ofn.nMaxFileName = 255
```

```
    ofn.lpstrInitialDir = StrAdr("C:¥" & Chr$(0))
```

```
    ofn.lpstrTitle = StrAdr("ファイルを開く" & Chr$(0))
```

```
    ofn.flags = 0
```

```
    Ret = Api_GetOpenFileName(ofn)
```

```
    If Ret = 0 Then
```

```
        A% = MsgBox("", "「キャンセル」が押されました！", 0, 2)
```

```
    Else
```

```
        A% = MsgBox("", "「開く」が押されました！", 0, 2)
```

```
    End If
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

「ファイルを開く」「名前を付けて保存」ダイアログの作成

GetOpenFileName 「ファイルを開く」ダイアログボックスを作成

GetSaveFileName 「保存」ダイアログボックスを作成



```
'=====
' = 「ファイルを開く」「名前を付けて保存」ダイアログの作成
' = (OpenSave.bas)
'=====
#include "Windows.bi"

Type OPENFILENAME
    lStructSize      As Long          '構造体のサイズをバイト単位で指定
    hwndOwner        As Long          'ダイアログボックスを所有するウィンドウへのハンドルを指定
    hInstance        As Long          'メモリオブジェクトへのハンドルを指定
    lpstrFilter       As Long          '任意の数のNULL文字で終わる文字列のペアを保持するバッファへのポインタを指定
    lpstrCustomFilter As Long          'ユーザーによって選択されたファイル フィルター文字列のペアが保存される
    nMaxCustFilter    As Long          'lpstrCustomFilterメンバのバッファ サイズを指定
    nFilterIndex      As Long          'フィルターの初期インデックスを指定
    lpstrFile         As Long          '文字列バッファを指定
    nMaxFile          As Long          'lpstrFileメンバのバッファ サイズを指定
    lpstrFileTitle    As Long          '選択されたファイルの名前、および拡張子を保存するための文字列バッファを指定
    nMaxFileTitle     As Long          'lpstrFileTitle メンバのバッファ サイズを指定
    lpstrInitialDir   As Long          '初期ディレクトリのパスを指定
    lpstrTitle        As Long          'ダイアログボックスのタイトルバーに表示する文字列を指定
    Flags             As Long          'ダイアログボックスの初期化フラグを、「OFN_...」の定数を組み合わせて指定
    nFileOffset       As Integer       'lpstrFileメンバが示すパス内の、ファイル名の位置がバッファの先頭から何バイト目であるかを示す
    nFileExtension    As Integer       'lpstrFileメンバが示すパス内の、拡張子の位置がバッファの先頭から何バイト目であるかを示す
    lpstrDefExt       As Long          'デフォルト拡張子の文字列を示す
    lCustData         As Long          'lpfnHookメンバが示すフック プロシージャへ渡すデータを指定
    lpfnHook          As Long          'フック プロシージャへのポインタを指定
    lpTemplateName    As Long          'hInstanceメンバが示すモジュール内にあるダイアログテンプレートを指定
End Type

' 「ファイルを開く」ダイアログボックスを作成
Declare Function Api_GetOpenFileName& Lib "comdlg32" Alias "GetOpenFileNameA"
    (pOpenfilename As OPENFILENAME)

' 「保存」ダイアログボックスを作成
Declare Function Api_GetSaveFileName& Lib "comdlg32" Alias "GetSaveFileNameA"
    (pOpenfilename As OPENFILENAME)

#define OFN_HIDEREADONLY &H4          '読み取り専用のチェックボックスを非表示
#define OFN_PATHMUSTEXIST &H800      '有効なパスだけをうけつける
#define OFN_FILEMUSTEXIST &H1000     '存在しないファイル名は入力不可

Var Shared Text1 As Object
```

```

Var Shared Button1 As Object
Var Shared Button2 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Function ShowOpen(Filter As String, Flags As Long, hWnd As Long) As String
Function ShowOpen(Filter As String, Flags As Long, hWnd As Long) As String
    Var Buffer As String
    Var ofn As OPENFILENAME
    Var Ret As Long

    Buffer = String$(128, Chr$(0))

    ofn.lStructSize = Len(ofn)
    ofn.hwndOwner = hWnd
    ofn.Flags = Flags
    ofn.nFilterIndex = 1
    ofn.nMaxFile = Len(Buffer)
    ofn.lpstrFile = StrAdr(Buffer & Chr$(0))
    ofn.lpstrFilter = StrAdr(Filter & Chr$(0))

    Ret = Api_GetOpenFileName(ofn)

    If Ret <> 0 Then
        ShowOpen = Left$(Filter, InStr(Filter, Chr$(0)) - 1)
    End If
End Function

'=====
'=
'=====
Declare Function ShowSave(Filter As String, Flags As Long, hWnd As Long, FileName As
String) As String
Function ShowSave(Filter As String, Flags As Long, hWnd As Long, FileName As String) As
String
    Var ofn As OPENFILENAME
    Var Buffer As String
    Var Ret As Long

    Buffer = FileName & String$(128 - Len(FileName), 0)

    ofn.lStructSize = Len(ofn)
    ofn.hwndOwner = hWnd
    ofn.Flags = OFN_HIDEREADONLY Or OFN_PATHMUSTEXIST
    ofn.nFilterIndex = 1
    ofn.nMaxFile = Len(Buffer)
    ofn.lpstrFile = StrAdr(Buffer & Chr$(0))
    ofn.lpstrFilter = StrAdr(Filter & Chr$(0))

    Ret = Api_GetSaveFileName(ofn)

    If Ret <> 0 Then
        ShowSave = Left$(Filter, InStr(Filter, Chr$(0)) - 1)
    End If
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Filter As String
    Var FileName As String
    Var Flags As Long

```

```

Flags = OFN_FILEMUSTEXIST Or OFN_HIDEREADONLY Or OFN_PATHMUSTEXIST

Filter = "全てのファイル (*.*)" & Chr$(0) & "*.*" & Chr$(0) & "モジュール・クラス (*.bas ,
*.cls)" & Chr$(0) & "*.bas;*.cls" & Chr$(0) & Chr$(0)

FileName = ShowOpen(Filter, Flags, GethWnd)

Text1.SetWindowText FileName
End Sub

' =====
' =
' =====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Filter As String
    Var FileName As String
    Var Flags As Long

    Flags = OFN_FILEMUSTEXIST Or OFN_HIDEREADONLY Or OFN_PATHMUSTEXIST

    Filter = "全てのファイル (*.*)" & Chr$(0) & "*.*" & Chr$(0) & "モジュール・クラス (*.bas ,
*.cls)" & Chr$(0) & "*.bas;*.cls" & Chr$(0) & Chr$(0)

    FileName = ShowSave(Filter, Flags, GethWnd, "Test.txt")

    Text1.SetWindowText FileName
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

フェードイン・フェードアウト

AlphaBlend 透過ピクセルと半透過ピクセルを持つビットマップを表示

GetDC デバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```

' =====
' = フェードイン・フェードアウト
' = (Fading2.bas)
' =====
#include "Windows.bi"

' 透過ピクセルと半透過ピクセルを持つビットマップを表示
Declare Function Api_AlphaBlend Lib "msimg32" Alias "AlphaBlend" (ByVal hdcDest&, ByVal
nXDest&, ByVal nYDest&, ByVal nWidthDest&, ByVal nHeightDest&, ByVal hdcSrc&, ByVal
nXSrc&, ByVal nYSrc&, ByVal nWidthSrc&, ByVal nHeightSrc&, ByVal nBlendFunc&)

```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define SRCAND &H8800C6
#define SRCCOPY &HCC0020

' 転送先の画像とAND演算して転送
' そのまま転送

Var Shared Picture1 As Object
Var Shared Picture2 As Object
Var Shared Radio1 As Object
Var Shared Radio2 As Object
Var Shared Button1 As Object
Var Shared Bitmap As Object

Picture1.Attach GetDlgItem("Picture1")
Picture2.Attach GetDlgItem("Picture2")
Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 12
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
BitmapObject Bitmap

' =====
' =
' =====

Declare Function Index bdecl () As Integer
Function Index()
 Index = Val(Mid\$(GetDlgRadioSelect("Radio1"), 6)) - 1
End Function

' =====
' =
' =====

Declare Sub Radio1_on edecl ()
Sub Radio1_on()
 Bitmap.LoadFile "FadeIn.bmp"
 Picture1.StretchBitmap Bitmap, 0, 0, Picture1.GetWidth, Picture1.GetHeight
 Bitmap.DeleteObject
End Sub

' =====
' =
' =====

Declare Sub Radio2_on edecl ()
Sub Radio2_on()
 Bitmap.LoadFile "FadeOut.bmp"
 Picture1.StretchBitmap Bitmap, 0, 0, Picture1.GetWidth, Picture1.GetHeight
 Bitmap.DeleteObject
End Sub

' =====
' =
' =====

Declare Sub Pic2DrawBitmap()
Sub Pic2DrawBitmap()
 Bitmap.LoadFile "flower.bmp"
 Picture2.StretchBitmap Bitmap, 0, 0, Picture2.GetWidth, Picture2.GetHeight
 Bitmap.DeleteObject
End Sub

' =====
' =
' =====

Declare Sub MainForm_Start edecl ()
Sub MainForm_Start()
 Pic2DrawBitmap
 Radio1_on
End Sub

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC1 As Long
    Var hDC2 As Long
    Var i As Long
    Var Ret As Long

    hDC1 = Api_GetDC(Picture1.GethWnd)
    hDC2 = Api_GetDC(Picture2.GethWnd)

    Button1.EnableWindow 0

    For i = 0 To 100
        If Index = 0 Then
            Ret = Api_AlphaBlend(hDC1, 0, 0, Picture1.GetWidth, Picture1.GetHeight, hDC2,
0, 0, Picture2.GetWidth, Picture2.GetHeight, i * &H8000)
        Else
            Ret = Api_AlphaBlend(hDC2, 0, 0, Picture2.GetWidth, Picture2.GetHeight, hDC1,
0, 0, Picture1.GetWidth, Picture1.GetHeight, i * &H8000)
        End If
        Wait 3
    Next

    Button1.EnableWindow -1

    Ret = Api_ReleaseDC(Picture1.GethWnd, hDC1)
    Ret = Api_ReleaseDC(Picture2.GethWnd, hDC2)

    Pic2DrawBitmap
End Sub

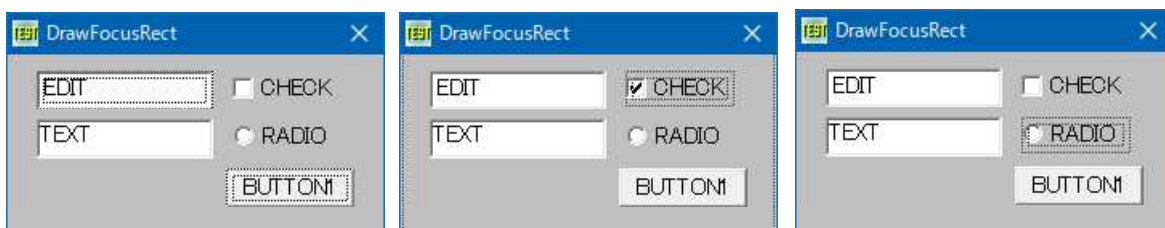
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォーカスを得たとき点線の枠を描画

DrawFocusRect フォーカスを得たとき、点線の枠を描画
InflateRect RECT構造体の座標を拡大・縮小

DrawFocusRectは、同じ矩形領域を再描画した場合は、その矩形(点線)は消去されます。Xorの関係
InflateRect(rc, -1, -1)を繰り返すことにより点線の太さを変えています。



関連

MiniMiniSpy--は、領域を赤枠で描画していますが、DrawFocusRectで点線で描画する方法もあります。

```

'=====
'= フォーカスを得た時、点線の枠を描く
'= (DrawFocusRect2.bas)
'=====
#include "Windows.bi"

```

```

Type POINTAPI
    x    As Long
    y    As Long
End Type

Type RECT
    Left    As Long
    Top     As Long
    Right   As Long
    Bottom  As Long
End Type

' ウィンドウの座標をスクリーン座標系で取得
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,
lpRect As RECT)

' カーソルの現在のスクリーン座標の取得
Declare Function Api_GetCursorPos& Lib "user32" Alias "GetCursorPos" (lpPoint As
POINTAPI)

' 指定の座標位置にあるウィンドウハンドルを取得
Declare Function Api_WindowFromPoint& Lib "user32" Alias "WindowFromPoint" (ByVal
xPoint&, ByVal yPoint&)

' ウィンドウ全体のデバイスコンテキストを取得
Declare Function Api_GetWindowDC& Lib "user32" Alias "GetWindowDC" (ByVal hWnd&)

' 指定されたデバイスコンテキストを削除
Declare Function Api_Deletedc& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
'Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' フォーカスを得た時、点線の枠を描く
Declare Function Api_DrawFocusRect& Lib "user32" Alias "DrawFocusRect" (ByVal hDC&,
lpRect As RECT)

' RECT構造体の座標値を拡大・縮小
Declare Function Api_InflateRect& Lib "user32" Alias "InflateRect" (lpRect As RECT, ByVal
x&, ByVal y&)

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Check1 As Object
Var Shared Radiol As Object
Var Shared Button1 As Object
Var Shared Timer1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Check1.Attach GetDlgItem("Check1") : Check1.SetFontSize 14
Radiol.Attach GetDlgItem("Radiol") : Radiol.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Timer1.Attach GetDlgItem("Timer1")

' =====
' =
' =====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.SetInterval 40
    Timer1.Enable -1
End Sub

```

```

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var hWnd As Long
    Var hDC As Long
    Var rc As RECT
    Var pa As POINTAPI
    Var Ret As Long

    'カーソル位置のスクリーン座標を取得
    Ret = Api_GetCursorPos(pa)

    '座標を含むウィンドウのハンドルを取得
    hWnd = Api_WindowFromPoint(pa.x, pa.y)

    '矩形領域を取得
    Ret = Api_GetWindowRect(hWnd, rc)

    'デバイスコンテキストを取得
    hDC = Api_GetWindowDC(hWnd)
    Ret = Api_SetRect(rc, 0, 0, rc.Right - rc.Left, rc.Bottom - rc.Top)

    If hWnd <> 0 Then

        '太さ(lw)を設定し点線枠を描画
        For lw = 0 To 1
            Ret = Api_DrawFocusRect(hDC, rc)
            Ret = Api_InflateRect(rc, -1, -1)
        Next
    End If

    Ret = Api_DeleteDC(hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォームが指定領域内にあるかを判断

フォームが指定した矩形領域内にあるかどうかを判断します。

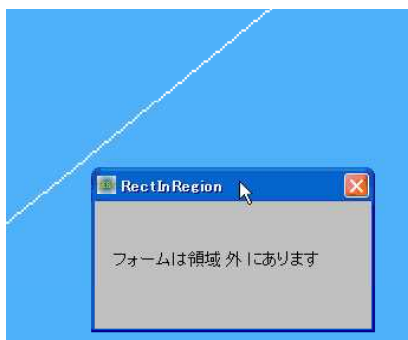
RectInRegion 長方形の一部または全部が、リージョンの内側にあるかどうかを判断

GetSystemMetrics さまざまなシステムメトリックスの値とシステムの現在の構成を取得

CreatePolygonRgn 多角形のリージョンを作成

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

DeleteObject 論理オブジェクトを削除



ディスプレイの右上から左下に線を引き、フォームがその左上に相当する領域内にあるかどうかを判断しています。


```

'=====
'= フォームが指定領域内にあるかを判断
'= (RectInRegion.bas)
'=====
#include "Windows.bi"

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

Type POINTAPI
    x As Long
    y As Long
End Type

' 長方形の一部または全部が、リージョンの内側にあるかどうかを判断
Declare Function Api_RectInRegion& Lib "gdi32" Alias "RectInRegion" (ByVal hRgn&, lpRect As RECT)

' さまざまなシステムメトリックの値とシステムの現在の構成を取得
Declare Function Api_GetSystemMetrics& Lib "user32" Alias "GetSystemMetrics" (ByVal nIndex&)

' 多角形のリージョンを作成
Declare Function Api_CreatePolygonRgn& Lib "gdi32" Alias "CreatePolygonRgn" (lppt As POINTAPI, ByVal nCount&, ByVal nPolyFillMode&)

' ウィンドウの座標をスクリーン座標系で取得
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&, lpRect As RECT)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

#define SM_CXSCREEN 0           'ディスプレイの幅
#define SM_CYSCREEN 1         'ディスプレイ高さ
#define INDING 2              '全域モード(塗りつぶし)

Var Shared hRgn As Long

Var Shared Timer1 As Object
Var Shared Text1 As Object

Timer1.Attach GetDlgItem("Timer1")
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14

'=====
'=
'=====
Declare Sub Timer1_Timer edecl ()
Sub Timer1_Timer()
    Var sWidth As Long
    Var sHeight As Long
    Var rc As RECT
    Var pts(2) As POINTAPI
    Var IsInside As Long
    Var Ret As Long

    sWidth = Api_GetSystemMetrics(SM_CXSCREEN)
    sHeight = Api_GetSystemMetrics(SM_CYSCREEN)

    pts(0).x = 0
    pts(0).y = 0
    pts(1).x = sWidth
    pts(1).y = 0
    pts(2).x = 0
    pts(2).y = sHeight

```



```

hRgn = Api_CreatePolygonRgn(pts(0), 3, INDING)

Ret = Api_GetWindowRect(GethWnd, rc)

IsInside = Api_RectInRegion(hRgn, rc)

If IsInside = 0 Then
    Text1.SetWindowText "フォームは領域 外 にあります"
Else
    Text1.SetWindowText "フォームは領域 内 にあります"
End If

Ret = Api_DeleteObject(hRgn)
End Sub

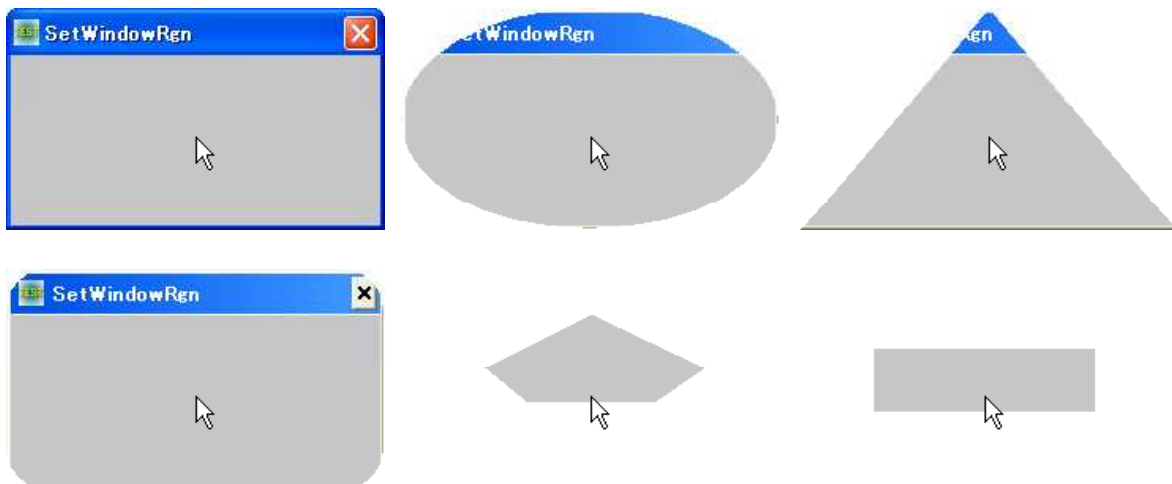
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォームの形状いろいろ

フォームの形状を変化させてみます。

CreateRoundRectRgn 角の丸い長方形のリージョンを作成
CreateEllipticRgnIndirect 指定の領域をウィンドウ領域として設定
GetWindowRgn 指定のウィンドウのウィンドウ領域を取得
SetWindowRgn 指定の領域をウィンドウ領域として設定
DeleteObject 論理オブジェクトを削除
CreatePolygonRgn 多角形のリージョンを作成
GetWindowRect ウィンドウの座標をスクリーン座標系で取得
CreateRectRgnIndirect 矩形領域をRECT構造体にも続いて作成
InflateRect RECT構造体の座標値を拡大・縮小
CopyRect RECT構造体の座標値を別の構造体にコピー
SetRect RECT構造体の値を設定



```

'=====
'= フォームの形状いろいろ
'=   (CopyRect.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type

```

```

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

' 角の丸い長方形のリージョンを作成
Declare Function Api_CreateRoundRectRgn& Lib "gdi32" Alias "CreateRoundRectRgn" (ByVal
nLeftRect&, ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&, ByVal
nWidthEllipse&, ByVal nHeightEllipse&)

' 指定の領域をウィンドウ領域として設定
Declare Function Api_CreateEllipticRgnIndirect& Lib "gdi32" Alias
"CreateEllipticRgnIndirect" (lpRect As RECT)

' 指定のウィンドウのウィンドウ領域を取得
Declare Function Api_GetWindowRgn& Lib "user32" Alias "GetWindowRgn" (ByVal hWnd&, ByVal
hRgn&)

' 指定の領域をウィンドウ領域として設定
Declare Function Api_SetWindowRgn& Lib "user32" Alias "SetWindowRgn" (ByVal hWnd&, ByVal
hRgn&, ByVal bRedraw&)

' 論理オブジェクトを削除
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 多角形のリージョンを作成
Declare Function Api_CreatePolygonRgn& Lib "gdi32" Alias "CreatePolygonRgn" (lppt As
POINTAPI, ByVal nCount&, ByVal nPolygonFillMode&)

' ウィンドウの座標をスクリーン座標系で取得
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,
lpRect As RECT)

' 矩形領域をRECT構造体に基づいて作成
Declare Function Api_CreateRectRgnIndirect& Lib "gdi32" Alias "CreateRectRgnIndirect"
(lpRect As RECT)

' RECT構造体の座標値を拡大・縮小
Declare Function Api_InflateRect& Lib "user32" Alias "InflateRect" (lpRect As RECT, ByVal
x&, ByVal y&)

' RECT構造体の座標値を別のRECT構造体にコピー
Declare Function Api_CopyRect& Lib "user32" Alias "CopyRect" (lpDestRect As RECT,
lpSourceRect As RECT)

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

#define ALTERNATE 1
#define WINDING 2

Var Shared RectRgn As Long
Var Shared EllipticRgn As Long
Var Shared PolygonRgn As Long
Var Shared PolygonRgn2 As Long
Var Shared RoundRectRgn As Long
Var Shared BufferRgn As Long
Var Shared n As Long

' =====
' =
' =====

Declare Function CreateRgns () As Long
Function CreateRgns () As Long
    Var rct As RECT
    Var trct As RECT
    Var Polygon(2) As POINTAPI

```

```

Var Polygon2(4) As POINTAPI
Var Ret As Long

Ret = Api_GetWindowRect(GethWnd, rct)
Ret = Api_SetRect(rct, 0, 0, rct.Right - rct.Left, rct.Bottom - rct.Top)

'矩形領域をRECT構造体に基づいて作成
BufferRgn = Api_CreateRectRgnIndirect(trct)

'楕円形
EllipticRgn = Api_CreateEllipticRgnIndirect(rct)

'三角形
Polygon(0).x = rct.Right / 2
Polygon(0).y = 0
Polygon(1).x = 0
Polygon(1).y = rct.Bottom
Polygon(2).x = rct.Right
Polygon(2).y = rct.Bottom
PolygonRgn = Api_CreatePolygonRgn(Polygon(0), 3, WINDING)

'五角形
Polygon2(0).x = rct.Right / 2
Polygon2(0).y = rct.Bottom / 5
Polygon2(1).x = rct.Right / 4.5
Polygon2(1).y = 2 * rct.Bottom / 4.5
Polygon2(2).x = rct.Right / 3
Polygon2(2).y = 3 * rct.Bottom / 5
Polygon2(3).x = 3 * rct.Right / 4.5
Polygon2(3).y = 3 * rct.Bottom / 5
Polygon2(4).x = 4 * rct.Right / 5
Polygon2(4).y = 2 * rct.Bottom / 4.5
PolygonRgn2 = Api_CreatePolygonRgn(Polygon2(0), 5, WINDING)

'角を丸める
RoundRectRgn = Api_CreateRoundRectRgn(rct.Left, rct.Top, rct.Right, rct.Bottom, 50,
50)

'RECT構造体の座標値を別のRECT構造体にコピーして縮小
Ret = Api_CopyRect(trct, rct)
Ret = Api_InflateRect(trct, -50, -50)
RectRgn = Api_CreateRectRgnIndirect(trct)
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    n = 0
    Ret = CreateRgns
End Sub

'=====
'=
'=====
Declare Sub MainForm_Click edec1 ()
Sub MainForm_Click()
    Var Ret As Long

    Ret = Api_GetWindowRgn(GethWnd, BufferRgn)

    n = n + 1
    If n > 6 Then n = 1

    Select Case n
        Case 1
            Ret = Api_SetWindowRgn(GethWnd, EllipticRgn, 1)
        Case 2
            Ret = Api_SetWindowRgn(GethWnd, PolygonRgn, 1)

```

```

Case 3
    Ret = Api_SetWindowRgn (GethWnd, PolygonRgn2, 1)
Case 4
    Ret = Api_SetWindowRgn (GethWnd, RoundRectRgn, 1)
Case 5
    Ret = Api_SetWindowRgn (GethWnd, RectRgn, 1)
Case 6
    Ret = Api_SetWindowRgn (GethWnd, 0, 1)
    Ret = CreateRgns
End Select
End Sub

' =====
' =
' =====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    Ret = Api_DeleteObject (EllipticRgn)
    Ret = Api_DeleteObject (PolygonRgn)
    Ret = Api_DeleteObject (PolygonRgn2)
    Ret = Api_DeleteObject (RoundRectRgn)
    Ret = Api_DeleteObject (StdRgn)
    Ret = Api_DeleteObject (BufferRgn)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

フォームの形状(ミッキーマウス)

フォームをミッキーマウスにしてみます。

CombineRgn 既存の二つの領域を結合し、新しい領域を作成

SetWindowRgn 指定の領域をウィンドウ領域として設定

CreateEllipticRgn 楕円形のリージョンを作成

DeleteObject オブジェクトを削除

SetCapture 指定のウィンドウにマウスキャプチャを設定

ReleaseCapture マウスキャプチャの解放

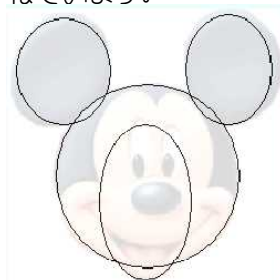
SetWindowPos ウィンドウのサイズ、位置、及びzオーダーを設定

GetCursorPos マウスカーソルの現在の位置に相当するスクリーン座標を取得

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

例では、下図のように4個の領域を結合しています。

MainFormのプロパティで境界線を選択、サイズは240×240ピクセル。同サイズのミッキーマウスのbmpファイルを重ねています。



```

' =====
' = フォームの形状(ミッキーマウス)
' =   (CombineRgn.bas)
' =====

```

```

#include "Windows.bi"

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

Type POINTAPI
    x As Long
    y As Long
End Type

' 既存の二つの領域を結合して新しい領域を作成
Declare Function Api_CombineRgn& Lib "gdi32" Alias "CombineRgn" (ByVal hRgnDest&, ByVal hRgnSrc1&, ByVal hRgnSrc2&, ByVal nCombineMode&)

' 指定の領域をウィンドウ領域として設定
Declare Function Api_SetWindowRgn& Lib "user32" Alias "SetWindowRgn" (ByVal hWnd&, ByVal hRgn&, ByVal bRedraw&)

' 楕円形のリージョンを作成
Declare Function Api_CreateEllipticRgn& Lib "gdi32" Alias "CreateEllipticRgn" (ByVal nLeftRect&, ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定のウィンドウにマウスキャプチャを設定
Declare Function Api_SetCapture& Lib "user32" Alias "SetCapture" (ByVal hWnd&)

' マウスのキャプチャを解放
Declare Function Api_ReleaseCapture& Lib "user32" Alias "ReleaseCapture" ()

' ウィンドウのサイズ、位置、および Z オーダーを設定
Declare Function Api_SetWindowPos& Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal uFlags&)

' マウスカーソル (マウスポインタ) の現在の位置に相当するスクリーン座標を取得
Declare Function Api_GetCursorPos& Lib "user32" Alias "GetCursorPos" (lpPoint As POINTAPI)

' ウィンドウの座標をスクリーン座標系で取得
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&, lpRect As RECT)

#define PS_SOLID 0
#define SWP_NOZORDER &H4
#define SWP_NOSIZE &H1
#define RGN_AND 1
#define RGN_COPY 5
#define RGN_DIFF 4
#define RGN_OR 2
#define RGN_XOR 3

' ウィンドウリスト内での現在位置を保持する
' ウィンドウの現在のサイズを保持する
' リージョン同士のAND結合
' HRGNSRC1のコピーを作成
' HRGNSRC1からHRGNSRC2を除いた領域
' リージョン同士のOR結合
' リージョン同士のXOR結合

Var Shared StartCursor As POINTAPI
Var Shared StartPos As RECT
Var Shared Capture As Integer

Var Shared Bitmap As Object
BitmapObject Bitmap

' =====
' =
' =====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var Rgn0 As Long
    Var Rgn1 As Long

```

```

Var Rgn2 As Long
Var Rgn3 As Long
Var Ret As Long

Bitmap.LoadFile "Mickey.bmp"
DrawBitmap Bitmap, 0, 0
Bitmap.DeleteObject

Rgn0 = Api_CreateEllipticRgn(42, 65, 202, 226)
Rgn1 = Api_CreateEllipticRgn(8, 8, 90, 100)
Rgn2 = Api_CreateEllipticRgn(155, 5, 230, 100)
Rgn3 = Api_CreateEllipticRgn(80, 100, 160, 236)

Ret = Api_CombineRgn(Rgn0, Rgn0, Rgn1, RGN_OR)
Ret = Api_CombineRgn(Rgn0, Rgn0, Rgn2, RGN_OR)
Ret = Api_CombineRgn(Rgn0, Rgn0, Rgn3, RGN_OR)

Ret = Api_SetWindowRgn(GethWnd, Rgn0, True)
ShowWindow -1
End Sub

'=====
'=
'=====
Declare Sub MainForm_DblClick edec1 ()
Sub MainForm_DblClick()
    End
End Sub

'=====
'=
'=====
Declare Sub MainForm_MouseDown edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal x As Single, ByVal y As Single)
Sub MainForm_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single,
ByVal y As Single)
    Var Ret As Long

    If Button = 1 Then
        'ドラッグ
        Capture = True
        Ret = Api_GetCursorPos(StartCursor)
        Ret = Api_GetWindowRect(GethWnd, StartPos)
        Ret = Api_SetCapture(GethWnd)
    End If
End Sub

'=====
'=
'=====
Declare Sub MainForm_MouseMove edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal x As Single, ByVal y As Single)
Sub MainForm_MouseMove(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single,
ByVal y As Single)
    Var CurrentCursor As POINTAPI
    static Event As Integer
    Var Ret As Long

    If Capture And Button = 1 And Not Event Then
        Event = True
        CallEvent

        Ret = Api_GetCursorPos(CurrentCursor)

        Ret = Api_SetWindowPos(GethWnd, 0, StartPos.Left + CurrentCursor.x -
StartCursor.x, StartPos.Top + CurrentCursor.y - StartCursor.y, 0, 0, SWP_NOZORDER Or
SWP_NOSIZE)
        Event = False
    End If

```

```

End Sub

'=====
'=
'=====
Declare Sub MainForm_MouseUp edec1 (ByVal Button As Integer, ByVal Shift As Integer, ByVal
x As Single, ByVal y As Single)
Sub MainForm_MouseUp (ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single,
ByVal y As Single)
    Var Ret As Long

    If Button = 1 And Capture Then

        'ドラッグ停止
        Capture = False
        Ret = Api_ReleaseCapture
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォームの最大化2題

SetWindowPosを使いFrameを含めた最大化とMaximizeWindowでの最大化

SetWindowPos ウィンドウのサイズ、位置、zオーダーを設定

GetSystemMetrics システムメトリック値とシステム現在の構成を取得

左:初期状態 中:SetWindowPosによる最大化(2倍に拡大) 右:MaximizeWindowによる最大化(2倍に拡大)



```

'=====
'= フォーム最大化
'= (Maximize.bas)
'=====
#include "Windows.bi"

' ウィンドウのサイズ、位置、および z オーダーを設定
Declare Function Api_SetWindowPos& Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal
hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal uFlags&)

' さまざまなシステムメトリックの値とシステムの現在の構成を取得
Declare Function Api_GetSystemMetrics& Lib "user32" Alias "GetSystemMetrics" (ByVal
nIndex&)

#define SM_CXSCREEN 0
#define SM_CYSCREEN 1
#define HWND_TOP 0
#define SWP_SHOWWINDOW &H40

' ディスプレイの幅
' ディスプレイ高さ
' ウィンドウをzオーダーの一番上に配置する
' ウィンドウを表示する

'=====
'= SetWindowPosでの最大化
'=====
Declare Sub Button1_on edec1 ()

```



```

Sub Button1_on()
    Var lWidth As Long
    Var lHeight As Long
    Var Ret As Long

    If IsMaximized Then
        RestoreWindow
    End If

    lWidth = Api_GetSystemMetrics(SM_CXSCREEN)
    lHeight = Api_GetSystemMetrics(SM_CYSCREEN)

    Ret = Api_SetWindowPos(GethWnd, HWND_TOP, 0, 0, lWidth, lHeight, SWP_SHOWWINDOW)
End Sub

'=====
'= F-BASIC通常の最大化
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    MaximizeWindow
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

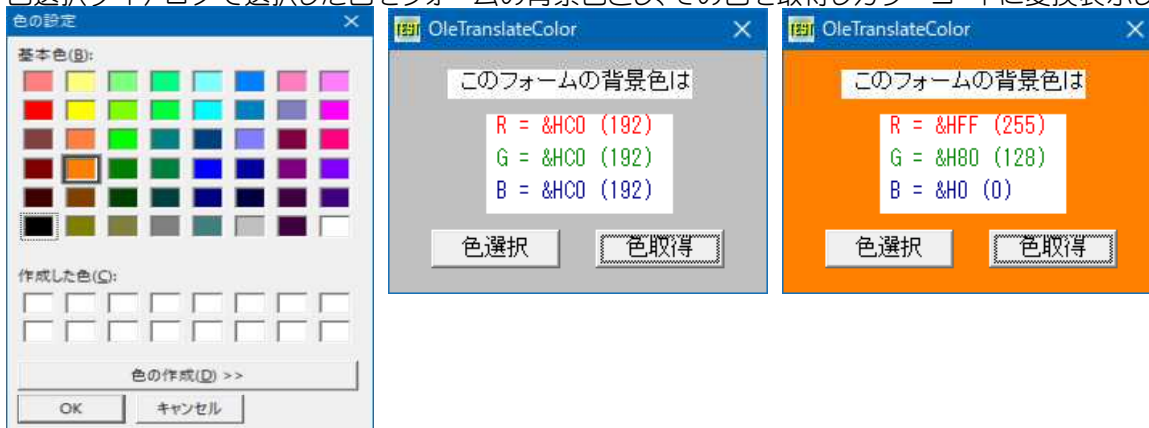
```

フォームの背景色を取得

例ではフォームの背景色を取得します。

OleTranslateColor カラーコードへの変換 (OLE_COLORタイプをCOLORREFに変換)

色選択ダイアログで選択した色をフォームの背景色とし、その色を取得しカラーコードに変換表示しています。



```

'=====
'= フォームの背景色を取得
'= (TranslateColor.bas)
'=====
#include "Windows.bi"

'カラーコードへの変換 (OLE_COLORタイプをCOLORREFに変換)
Declare Function Api_OleTranslateColor& Lib "olepro32" Alias "OleTranslateColor" (ByVal lOleColor&, ByVal lHPalette&, ByRef lColorRef&)

Var Shared Text(3) As Object
For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))

```

```

Next i

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var RealColor As Long

    SetMapMode 2
    If ChooseColor(RealColor) Then
        SetBackColor RealColor
        Cls
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long
    Var RealColor As Long

    Ret = Api_OleTranslateColor(GetBackColor, 0, RealColor)

    Text(1).SetWindowText " R = " & "&H" & Hex$(RealColor And &HFF) & " (" &
Trim$(Str$(RealColor And &HFF) & ") " & Chr$(13,10)
    Text(2).SetWindowText " G = " & "&H" & Hex$( (RealColor And &HFF00) / 2 ^ 8) & " (" &
Trim$(Str$( (RealColor And &HFF00) / 2 ^ 8) & ") " & Chr$(13,10)
    Text(3).SetWindowText " B = " & "&H" & Hex$( (RealColor And &HFF0000) / 2 ^ 16) & " (" &
Trim$(Str$( (RealColor And &HFF0000) / 2 ^ 16) & ") "
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォームの縁が沈んで見える境界線を指定

GetWindowLong 指定されたウィンドウに関する情報を取得
SetWindowLong 指定されたウィンドウの属性を変更
SetWindowPos ウィンドウのサイズ、位置、および Z オーダーを設定



Windows10ではサイズが変化するだけのようです。

```

'=====
'= フォームの縁が沈んで見える境界線を指定
'= (SetWindowLong3.bas)
'=====
#include "Windows.bi"

' 指定されたウィンドウに関する情報を取得。また、拡張ウィンドウメモリから、指定されたオフセットにある32ビット値
取得することもできる
Declare Function Api_GetWindowLong Lib "user32" Alias "GetWindowLongA" (ByVal hWnd&,

```

```
ByVal nIndex&)
```

' 指定されたウィンドウの属性を変更。また、拡張ウィンドウメモリの指定されたオフセットの32ビット値を書き換えることができる

```
Declare Function Api_SetWindowLong& Lib "user32" Alias "SetWindowLongA" (ByVal hWnd&, ByVal nIndex&, ByVal dwNewLong&)
```

' ウィンドウのサイズ、位置、および z オーダーを設定

```
Declare Function Api_SetWindowPos& Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal uFlags&)
```

```
#define GWL_EXSTYLE -20
#define WS_EX_CLIENTEDGE &H200
#define SWP_NOSIZE &H1
#define SWP_NOMOVE &H2
#define SWP_NOZORDER &H4
#define SWP_FRAMECHANGED &H20
```

```
#define SWP_DRAWFRAME &H20
```

' 拡張ウィンドウスタイル

' 縁が沈んで見える境界線を持つウィンドウを指定

' ウィンドウの現在のサイズを保持する

' ウィンドウの現在位置を保持する

' ウィンドウリスト内での現在位置を保持する

' ウィンドウのサイズ変更中であってもWM_NCCALCSIZEを送る

' 再描画のときウィンドウを囲む枠も描画

```
Var Shared Button1 As Object
Var Shared Button2 As Object
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
```

```
Var Shared OldStyle As Long
```

```
' =====
' =
' =====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var NewStyle As Long
```

```
    Var Ret As Long
```

' 拡張ウィンドウスタイルを取得

```
OldStyle = Api_GetWindowLong(GethWnd, GWL_EXSTYLE)
```

' 拡張ウィンドウスタイルに縁が沈んで見える境界線のフラグを追加

```
NewStyle = OldStyle Or WS_EX_CLIENTEDGE
```

' 新しい拡張ウィンドウスタイルを設定

```
Ret = Api_SetWindowLong(GethWnd, GWL_EXSTYLE, NewStyle)
```

' 新しいウィンドウスタイルを適用

```
Ret = Api_SetWindowPos(GethWnd, 0, 0, 0, 0, 0, 0, SWP_NOSIZE Or SWP_NOMOVE Or SWP_NOZORDER Or SWP_DRAWFRAME)
```

```
End Sub
```

```
' =====
' =
' =====
```

```
Declare Sub Button2_on edec1 ()
```

```
Sub Button2_on()
```

```
    Var Ret As Long
```

' 拡張ウィンドウスタイルを設定

```
Ret = Api_SetWindowLong(GethWnd, GWL_EXSTYLE, 0)
```

' ウィンドウスタイルを適用

```
Ret = Api_SetWindowPos(GethWnd, 0, 0, 0, 0, 0, 0, SWP_NOSIZE Or SWP_NOMOVE Or SWP_NOZORDER Or SWP_DRAWFRAME)
```

```
End Sub
```

```
' =====
' =
' =====
```

```
While 1
```

```
    WaitEvent
```

Wend
Stop
End

フォームのアニメーション移動(最小化・戻る時)

フォームを最小化(アイコン化)するとき、タスクバーから元に戻すときアニメーション表示...

SystemParametersInfo システムワイドパラメータを問い合わせや設定

SPI_SETANIMATION(73) アニメーション効果を設定

SPIF_UPDATEINIFILE(&H1) ユーザープロファイルの更新を指定

SPIF_SENDWININICHANGE(&H2) 全てのアプリケーションに通知して更新



```
'=====
'= アニメーション効果(フォームの最小化・戻る時)
'= (SysParamInfo.bas)
'=====
#include "Windows.bi"

Type tagANIMATIONINFO
    cbSize          As Long
    iMinAnimate     As Long
End Type

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo& Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

#define SPI_SETANIMATION 73
#define SPIF_UPDATEINIFILE &H1
#define SPIF_SENDWININICHANGE &H2
#define SPIF_SENDCHANGE SPIF_SENDWININICHANGE

Var Shared AnimFlag As Byte

'=====
'=
'=====
Declare Sub AnimeSetSelect edec1 ()
Sub AnimeSetSelect ()
    Var AnimInfo As tagANIMATIONINFO
    Var Ret As Long

    'アニメーション情報の構造体を初期化
    AnimInfo.cbSize = Len(AnimInfo)

    '最小化時のアニメーションのオンオフを指定
    AnimInfo.iMinAnimate = AnimFlag

    '最小化時などのアニメーションの有効無効を取得
    Ret = Api_SystemParametersInfo(SPI_SETANIMATION, Len(AnimInfo), AnimInfo,
    SPIF_UPDATEINIFILE Or SPIF_SENDCHANGE)
End Sub

'=====
'=
'=====
Declare Sub Radiol_on edec1 ()
Sub Radiol_on ()
```

```

        AnimFlag = 0
        AnimeSetSelect
End Sub

'=====
'=
'=====
Declare Sub Radio2_on edec1 ()
Sub Radio2_on ()
    AnimFlag = 1
    AnimeSetSelect
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

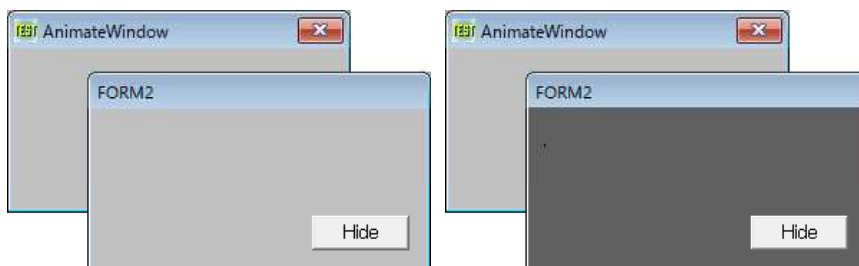
```

フォームをアニメーション起動・終了(1)

通常のフォーム表示じゃ物足りない？ではフェード効果を...

例では、メインフォームをフェードイン表示、さらにフォーム2をセンターからアニメート表示

AnimateWindow アニメートウィンドウ



```

'=====
'= フォームをアニメーション起動・終了
'= (AnimateWindow.bas)
'=====
#include "Windows.bi"

' フォームをアニメーション表示
Declare Function Api_AnimateWindow& Lib "user32" Alias "AnimateWindow" (ByVal hWnd&,
ByVal dwTime&, ByVal dwFlags&)

#define AW_HOR_POSITIVE &H1           ' 左から右に向かってアニメート
#define AW_HOR_NEGATIVE &H2          ' 右から左に向かってアニメート
#define AW_VER_POSITIVE &H4           ' 上から下に向かってアニメート
#define AW_VER_NEGATIVE &H8           ' 下から上に向かってアニメート
#define AW_CENTER &H10                ' 中心からアニメート
#define AW_HIDE &H10000              ' 非表示にする。既定値は表示
#define AW_ACTIVATE &H20000           ' アクティブに
#define AW_SLIDE &H40000              ' スライドアニメーション。既定値はロールアニメーション
#define AW_BLEND &H80000              ' フェードイン効果
#define FORM_HIDE &H80000 Or &H10000 ' 複合操作

Var Shared MainForm As Object
Var Shared Form2 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

MainForm.attach GethWnd
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

```

Var Shared hWnd2 As Long

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_start()
    Var Ret As Long

    Ret = Api_AnimateWindow(GethWnd, 1000, AW_BLEND)

    Cls
    SetFocus
    Form2.CreateWindow "Form2", 0
    hWnd2 = Form2.GethWnd
    Button2.Attach Form2.GetDlgItem("Button2") : Button2.SetFontSize 14
End Sub

'=====
'= Form2 ShowWindow
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_AnimateWindow(hWnd2, 1000, AW_CENTER)

    If Not (Ret <> 0) Then
        Ret = MessageBox("", "失敗", 0, 2)
    End If

    Form2.Cls
    Form2.SetFocus
End Sub

'=====
'= Form2 HideWindow
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    Ret = Api_AnimateWindow(hWnd2, 1000, FORM_HIDE)
    If Not (Ret <> 0) Then
        Ret = MessageBox("", "失敗", 0, 2)
    End If

    Form2.ShowWindow 0
    MainForm.SetFocus
End Sub

'=====
'= 終了
'=====
Declare Sub MainForm_QueryClose edec1 (Cancel%, ByVal Mode%)
Sub MainForm_QueryClose (Cancel%, ByVal Mode%)
    Var Ret As Long

    If Cancel% = 0 Then
        Ret = Api_AnimateWindow(GethWnd, 1000, FORM_HIDE)
    End
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend

```

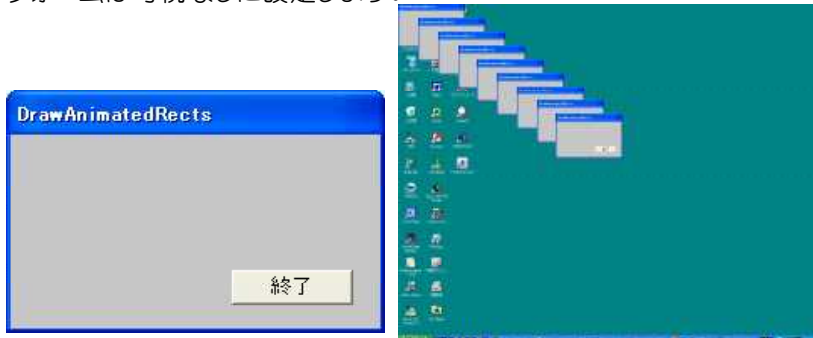
Stop
End

フォームをアニメーション起動・終了(II)

起動時フォームは画面左上から中央に、終了時はタスクバー右端に移動して終了します。

DrawAnimatedRects アニメートウィンドウ

フォームは可視なしに設定します。



```
'=====
'= フォームをアニメーション起動・終了(II)
'= (DrawAnimatedRects.bas)
'=====
#include "Windows.bi"

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

' ウィンドウの矩形寸法を取得
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,
lpRect As RECT)

' 矩形のアニメーションを実行
Declare Function Api_DrawAnimatedRects& Lib "user32" Alias "DrawAnimatedRects" (ByVal
hWnd&, ByVal idAni&, lpRect As RECT, lpRectEnd As RECT)

' クラス名またはキャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,
ByVal lpWindowName$)

' クラス名、または キャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindowEx& Lib "user32" Alias "FindWindowExA" (ByVal
hWndParent&, ByVal hWndChildAfter&, ByVal lpszClass$, ByVal lpszWindow$)

' アニメーションモード
#define IDANI_OPEN &H1
#define IDANI_CLOSE &H2
#define IDANI_CAPTION &H3

' クローズ状態からオープン状態へのアニメーション
' オープン状態からクローズ状態へのアニメーション
' アニメーション矩形にタイトルバーを含める

#define CLASS_TASKBAR "Shell_TrayWnd"
#define CLASS_TASKTRAY "TrayNotifyWnd"

Var Shared MainForm As Object
Var Shared Button1 As Object

MainForm.Attach GethWnd
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var rct As RECT
    Var rctEnd As RECT
    Var frmLeft As Long
    Var frmTop As Long
    Var Ret As Long

    ' アニメーションを開始する (Close状態) 矩形を指定
    rct.Top = 0
    rct.Bottom = 0
    rct.Left = 0
    rct.Right = 0

    ' アニメーションを終了する (Open状態) 矩形を指定
    ' メインフォームの境界矩形の寸法を取得
    Ret = Api_GetWindowRect (GethWnd, rctEnd)

    ' アニメーションの実行
    Ret = Api_DrawAnimatedRects (GethWnd, IDANI_OPEN Or IDANI_CAPTION, rct, rctEnd)

    frmLeft = (GetdeviceCaps (8) - GetWidth) / 2
    frmTop = (GetDeviceCaps (10) - GetHeight) / 2

    MainForm.MoveWindow frmLeft, frmTop
    MainForm.ShowWindow -1
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var rct As RECT
    Var rctEnd As RECT

    Var hTaskBar As Long
    Var hTaskTray As Long
    Var Ret As Long

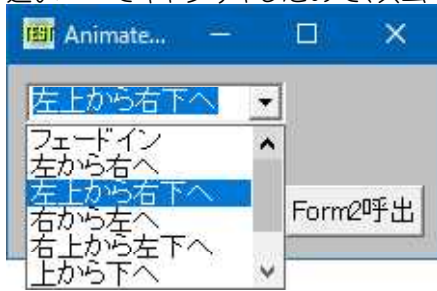
    hTaskBar = Api_FindWindow (CLASS_TASKBAR, ByVal 0)
    If hTaskBar Then
        hTaskTray = Api_FindWindowEx (hTaskBar, 0&, CLASS_TASKTRAY, ByVal 0)
        If hTaskTray Then
            Ret = Api_GetWindowRect (hTaskTray, rctEnd)
            Ret = Api_GetWindowRect (GethWnd, rct)
            Ret = Api_DrawAnimatedRects (GethWnd, IDANI_CLOSE Or IDANI_CAPTION, rct,
rctEnd)
        End If
    End If
    End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```


8通りのアニメーション方法をテストします。
 AnimateWindow フォームをアニメーション表示

「Form2」ボタンクリックをクリックすると、選択したアニメーションでForm2を起動します。フォームをアニメーション起動・終了(Ⅰ)と同じAPIを使いますが、8通りの方法を切り替えてテストできます。
 遅いCPUでキャプチャしたので、スムーズではありませんが...



```
'=====
'= フォームをアニメーション起動・終了(Ⅲ)
'= (AnimateWindow2.bas)
'=====

#include "Windows.bi"

' フォームをアニメーション表示
Declare Function Api_AnimateWindow& Lib "user32" Alias "AnimateWindow" (ByVal hWnd&,
ByVal dwTime&, ByVal dwFlags&)

#define AW_ACTIVATE &H20000                                'ウィンドウをアクティブにする。AW_HIDEと同時にセットしないこと
#define AW_BLEND &H80000                                    'フェードイン効果。AW_HIDEと同時にセットしないこと
#define AW_CENTER &H10                                     '中心からアニメート
#define AW_HIDE &H10000                                     '非表示にする。既定値は表示
#define AW_HOR_NEGATIVE &H2                                '右から左に向かってアニメート
#define AW_HOR_POSITIVE &H1                                '左から右に向かってアニメート
#define AW_SLIDE &H40000                                    'スライドアニメーションを使う。既定値はロールアニメーション
#define AW_VER_NEGATIVE &H8                                '下から上に向かってアニメート
#define AW_VER_POSITIVE &H4                                '上から下に向かってアニメート

Var Shared MainForm As Object
Var Shared Form2 As Object
Var Shared Comb1 As Object

MainForm.Attach GethWnd
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14

Var Shared AnimType As Long 'アニメートタイプ

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
  Form2.CreateWindow "Form2", 0

  Comb1.AddString "フェードイン"
  Comb1.AddString "左から右へ"
  Comb1.AddString "左上から右下へ"
  Comb1.AddString "右から左へ"
  Comb1.AddString "右上から左下へ"
  Comb1.AddString "上から下へ"
  Comb1.AddString "下から上へ"
  Comb1.AddString "中央から外側へ"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
```

```

Var Index As Long
Var Ret As Long

Index = Combo1.GetCursel + 1

Select Case Index
    Case 0
        Exit Sub
    Case 1
        AnimType = AW_BLEND
    Case 2
        AnimType = AW_SLIDE Or AW_HOR_POSITIVE
    Case 3
        AnimType = AW_VER_POSITIVE Or AW_HOR_POSITIVE
    Case 4
        AnimType = AW_SLIDE Or AW_HOR_NEGATIVE
    Case 5
        AnimType = AW_HOR_NEGATIVE Or AW_VER_POSITIVE
    Case 6
        AnimType = AW_SLIDE Or AW_VER_POSITIVE
    Case 7
        AnimType = AW_SLIDE Or AW_VER_NEGATIVE
    Case 8
        AnimType = AW_CENTER
End Select

MainForm.EnableWindow 0
Ret = Api_AnimateWindow(Form2.GetWnd, 1000, AnimType)
Form2.Cls
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    Ret = Api_AnimateWindow(Form2.GetWnd, 1000, AW_BLEND Or AW_HIDE)
    MainForm.EnableWindow -1
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォームをアニメーション起動・終了(Ⅳ)

起動時は、タスクバー右端から画面左上に向かってアニメーション起動し、終了時はタスクバー右端に向かってアニメーション終了します。

SetRect RECT構造体の値を設定

DrawAnimatedRects 矩形のアニメーションを実行



```

'=====
'= フォームをアニメーション起動・終了 (IV)
'= (SetRect.bas)
'=====
#include "Windows.bi"
#define IDANI_OPEN &H1
#define IDANI_CLOSE &H2
#define IDANI_CAPTION &H3

'クローズ状態からオープン状態へのアニメーション
'オープン状態からクローズ状態へのアニメーション
'アニメーション矩形にタイトルバーを含める

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' 矩形のアニメーションを実行
Declare Function Api_DrawAnimatedRects& Lib "user32" Alias "DrawAnimatedRects" (ByVal
hWnd&, ByVal idAni&, lprctStart As RECT, lprctEnd As RECT)

Var Shared scWidth As Long
Var Shared scHeight As Long

'=====
'= タスクバー右から画面左上方向へアニメート起動
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var rSource As RECT
    Var rDest As RECT
    Var Ret As Long

    'スクリーンサイズ取得
    scWidth = GetDeviceCaps(8)
    scHeight = GetDeviceCaps(10)

    'スクリーン(rSource) /作成するウインドウサイズ(rDest)
    Ret = Api_SetRect(rSource, scWidth, scHeight, scWidth, scHeight)
    Ret = Api_SetRect(rDest, 0, 0, 240, 140)

    'アニメート
    Ret = Api_DrawAnimatedRects(GetHwnd, IDANI_CLOSE Or IDANI_CAPTION, rSource, rDest)

    'フォームの位置設定
    SetWindowSize 240, 140
    MoveWindow 0, 0
    ShowWindow -1
End Sub

'=====
'= タスクバー右端に向かってアニメート終了
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var rSource As RECT
    Var rDest As RECT
    Var Ret As Long

    'スクリーン(rSource) /作成するウインドウサイズ(rDest)
    Ret = Api_SetRect(rSource, 0, 0, 240, 140)
    Ret = Api_SetRect(rDest, scWidth, scHeight, scWidth, scHeight)

    'アニメート
    Ret = Api_DrawAnimatedRects(GetHwnd, IDANI_OPEN Or IDANI_CAPTION, rSource, rDest)
End Sub

```

```
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

フォームをアニメーションで終了

フォームをアニメーションで終了させます。

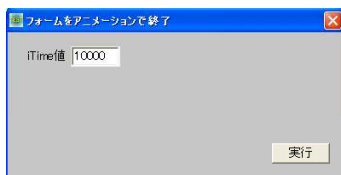
SystemParametersInfo システム全体に関するパラメータを取得・設定

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

Rectangle 長方形の描画

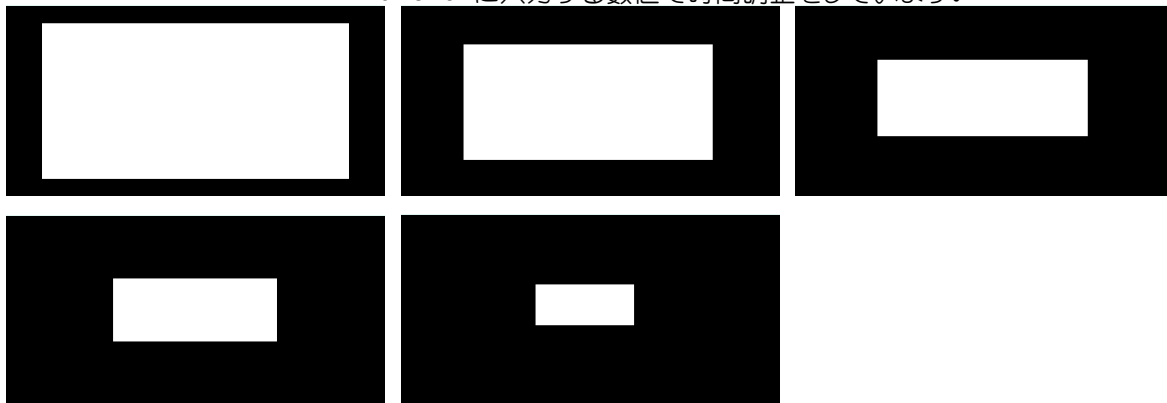
GetDC デバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



Rectangleでアニメーション風にフォームを徐々に小さく描画しています。

TextBoxに入力する数値で時間調整をしています。



```
'=====
'= フォームをアニメーションで終了
'= (EndToAnimation.bas)
'=====
#include "Windows.bi"
```

```
Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type
```

```
#define SPI_GETWORKAREA 48 '主モニターの有効なスクリーンのサイズを取得
```

' システム全体に関するパラメータを取得・設定

```
Declare Function Api_SystemParametersInfo& Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)
```

' ウィンドウの座標をスクリーン座標系で取得

```
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,
    lpRect As RECT)
```

' 長方形の描画

```
Declare Function Api_Rectangle& Lib "gdi32" Alias "Rectangle" (ByVal hDC&, ByVal X1&,
    ByVal Y1&, ByVal X2&, ByVal Y2&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====

Declare Sub EndToAnimation edec1 ()

Sub EndToAnimation()
 Var Scrn As Long
 Var DelObj As Long
 Var rct As RECT
 Var iWidth As Integer
 Var iHeight As Integer
 Var Cnt As Integer
 Var iLeft As Integer
 Var iTop As Integer
 Var x As Integer
 Var y As Integer
 Var iTime As Long
 Var Ret As Long

' 描画時間

iTime = Val(Edit1.GetWindowText)
If iTime > 32767 Then Exit Sub

' ウィンドウ(フォーム)のサイズ取得

Ret = Api_GetWindowRect(GetHwnd, rct)
iWidth = (rct.Right - rct.Left)
iHeight = rct.Bottom - rct.Top

' スクリーンのデバイスコンテキスト取得(0を指定)

Scrn = Api_GetDC(0)

For Cnt = iTime To 1 Step -1
 x = iWidth * (Cnt / iTime)
 y = iHeight * (Cnt / iTime)
 iLeft = rct.Left + (iWidth - x) / 2
 iTop = rct.Top + (iHeight - y) / 2

' 長方形を描画

Ret = Api_Rectangle(Scrn, iLeft, iTop, iLeft + x, iTop + y)
Next Cnt

Ret = Api_ReleaseDC(0, Scrn)
End

End Sub

'=====

Declare Sub Button1_on edec1 ()

Sub Button1_on()
 EndToAnimation

End Sub

'=====

Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()

```

EndToAnimation
End Sub

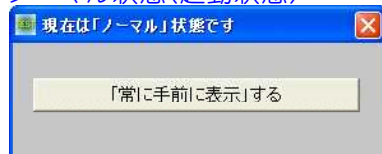
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

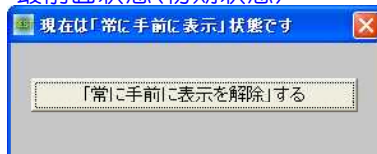
フォームを最前面に (1)

SetWindowPos 指定のウィンドウサイズ・位置・zオーダー (ウィンドウの重なる順) を設定する

ノーマル状態 (起動状態)

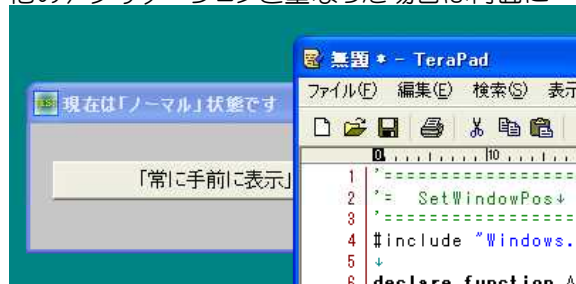


最前面状態 (初期状態)



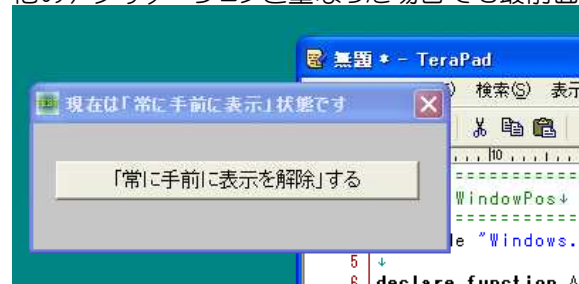
ノーマル状態

他のアプリケーションと重なった場合は背面に



最前面状態

他のアプリケーションと重なった場合でも最前面に



```

'=====
'= フォームを最前面に
'= (SetWindowPos.bas)
'=====
#include "Windows.bi"

```

' ウィンドウのサイズ、位置、および z オーダーを設定。(ウィンドウの重なり順のことを「zオーダー」といいzオーダーのトップに置くと一番手前に表示される)

Declare Function Api_SetWindowPos& Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal uFlags&)

```

#define HWND_BOTTOM 1
#define HWND_NOTOPMOST (-2)

#define HWND_TOP 0
#define HWND_TOPMOST (-1)

```

```

#define SWP_SHOWWINDOW &H40
#define SWP_NOSIZE &H1
#define SWP_NOMOVE &H2

```

```

Var Shared Button1 As Object
Button1.Attach GetDlgItem("Button1")

```

```

Declare Sub MainForm_Start edec1 ()
Declare Sub Form_State edec1 ()

```

```

Var Shared FLG As Byte

```

'ウィンドウを最背面に配置
'ウィンドウを常に最前面に配置 (他のウィンドウが HWND_TOPMOSTに配置されている場合はその配下)
'ウィンドウを最前面に配置
'ウィンドウを常に最前面に配置

'ウィンドウを表示する
'ウィンドウの現在のサイズを保持する
'ウィンドウの現在位置を保持する

```

'=====
'=
'=====
Sub MainForm_Start()
    FLG = 0
    Form_State
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    FLG = Not FLG
    Form_State
End Sub

'=====
'=
'=====
Sub Form_State()
    Var Ret As Long

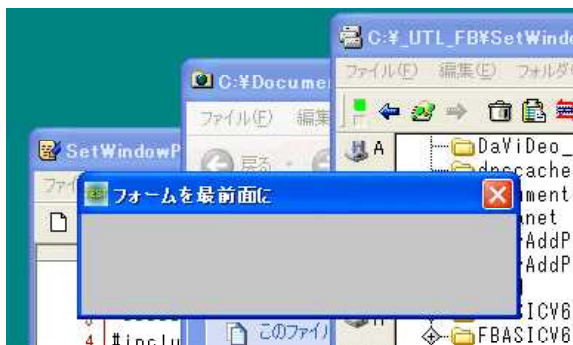
    If FLG = 0 Then
        Ret = Api_SetWindowPos(GethWnd, HWND_NOTOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW Or
SWP_NOMOVE Or SWP_NOSIZE)
        Button1.SetWindowText "「常に手前に表示」する"
        SetWindowText "現在は「ノーマル」状態です"
    Else
        Ret = Api_SetWindowPos(GethWnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW Or
SWP_NOMOVE Or SWP_NOSIZE)
        Button1.SetWindowText "「常に手前に表示を解除」する"
        SetWindowText "現在は「常に手前に表示」状態です"
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォームを最前面に(II)

SetWindowPos APIを使ってフォームを最前面に表示



```

'=====
'= フォームを最前面に(II・III)
'=====
#include "Windows.bi"

```

' 指定のウィンドウのサイズ・位置・zオーダーを設定する

```
Declare Function API_SETWINDOWPOS Lib "user32" Alias "SetWindowPos" (ByVal HWND&, ByVal
HWNDINSERTAFTER&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal WFLAGS&)
```

```
#define HWND_TOPMOST -1
```

```
#define SWP_NOSIZE &H1
```

```
#define SWP_NOMOVE &H2
```

'ウィンドウをウィンドウリストの一番上に配置

'ウィンドウの現在のサイズを保持

'ウィンドウの現在位置を保持

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
Var RES As Long
```

' 現在の位置とサイズでウィンドウリストの一番上に配置

```
RES = API_SETWINDOWPOS (MAINFORM.GETHWND, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE Or
SWP_NOSIZE)
```

```
End Sub
```

```
'=====
```

```
'=
```

```
'=====
```

```
While 1
```

```
WaitEvent
```

```
Wend
```

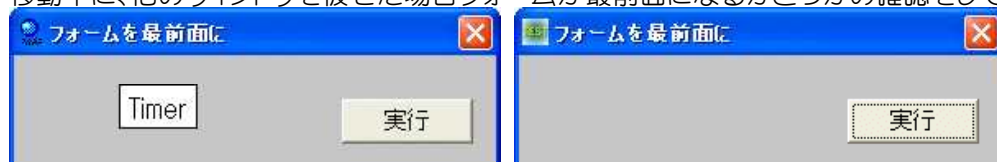
```
Stop
```

```
End
```

フォームを最前面に(III)

SetWindowPos APIを使ってフォームを最前面に表示

②と同じですが、実行ボタンをクリックするとフォームが左上から中央に向かって移動します。
移動中に、他のウィンドウを被せた場合フォームが最前面になるかどうかの確認をしています。それだけ..(--;)



```
'=====
```

```
'= フォームを最前面に(III)
```

```
'=====
```

```
#include "Windows.bi"
```

' 指定のウィンドウのサイズ・位置・zオーダーを設定する

```
Declare Function Api_SetWindowPos Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal
hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal uFlags&)
```

```
#define HWND_TOP 0
```

```
#define HWND_TOPMOST -1
```

```
#define SWP_NOSIZE &H1
```

```
#define SWP_NOMOVE &H2
```

'zオーダーのトップ

'ウィンドウをウィンドウリストの一番上に配置

'ウィンドウの現在のサイズを保持

'ウィンドウの現在位置を保持

```
Var Shared Timer1 As Object
```

```
Timer1.Attach GetDlgItem("Timer1")
```

```
Var Shared cnt As Long
```

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```



```

    Timer1.SetInterval 10
    Timer1.Enable 0
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Timer1.Enable -1
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var Ret As Long

    cnt = cnt + 1
    If cnt < 30 Then
        Ret = Api_SetWindowPos (GethWnd, HWND_TOPMOST, cnt * 10, cnt * 10, 0, 0,
SWP_SHOWWINDOW Or SWP_NOSIZE)
    Else
        Ret = Api_SetWindowPos (GethWnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW Or
SWP_NOSIZE)
        Timer1.Enable 0
        cnt = 0
    End If
End Sub

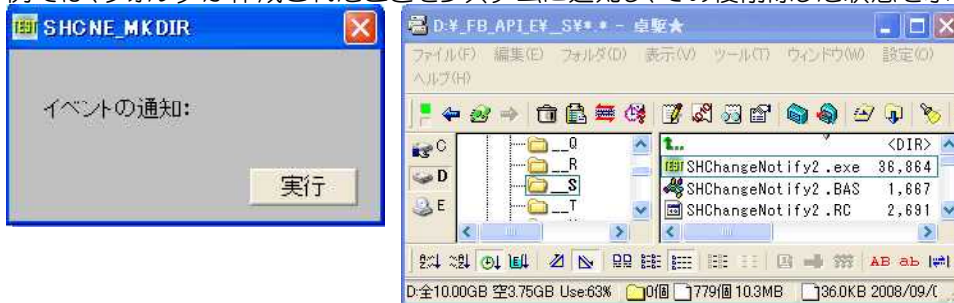
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォルダ作成をシステムに通知

SHChangeNotify イベントをシステムに通知
SHCNE_MKDIR(&H8) フォルダが作成された
SHCNE_RMDIR(&H10) フォルダが削除された
SHCNF_PATH(&H1) パス名SHCNF_PATHA
SHCNF_PATH(&H5) パス名SHCNF_PATHW(UNICODE)

例では、フォルダが作成されたことをシステムに通知し、その後削除した状態を示しています。



```

'=====
'= フォルダ作成をシステムに通知
'= (SHChangeNotify2.bas)
'=====
#include "Windows.bi"
#include "File.bi"

```

```

#define SHCNE_MKDIR &H8          'フォルダが作成された
#define SHCNE_RMDIR &H10        'フォルダが削除された
#define SHCNF_PATH &H1         'パス名SHCNF_PATHA
'#define SHCNF_PATH &H5        'パス名(UNICODE) SHCNF_PATHW

' イベントをシステムに通知
Declare Sub Api_SHChangeNotify Lib "shell32" Alias "SHChangeNotify" (ByVal wEventId&,
ByVal uFlags&, dwItem1 As Any, dwItem2 As Any)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var FolderName As String

    ' フォルダを作成
    FolderName = "Test"
    MkDir FolderName

    ' イベントをシステムに通知
    Api_SHChangeNotify SHCNE_MKDIR, SHCNF_PATH, FolderName, ByVal CLng(0)
    ' Api_SHChangeNotify SHCNE_RMDIR, SHCNF_PATH, FolderName, ByVal CLng(0)

    ' 結果を表示
    Text2.SetWindowText "通知しました。"

    ' フォルダ作成・削除確認のため
    A% = MsgBox("フォルダ作成・削除確認", "通知しました。", 0, 2)

    ' フォルダを削除
    RmDir FolderName
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

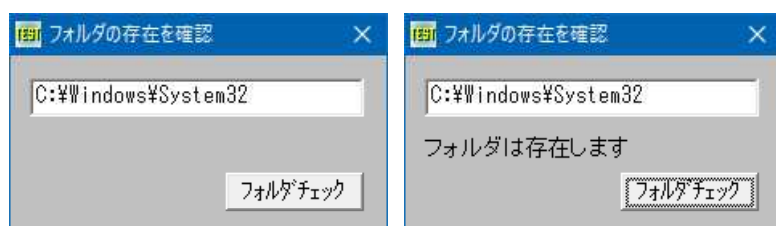
```

フォルダ存在の確認

指定したフォルダが存在するかどうかを確認します。

FindFirstFile ファイルやディレクトリを検索

FindClose ファイル検索ハンドルを解放



```

'=====
'= フォルダを検索
'= (FindFirstFile2.bas)
'=====
#include "Windows.bi"

#define FILE_ATTRIBUTE_DIRECTORY &H10 'ディレクトリ属性
#define INVALID_HANDLE_VALUE -1 '見つからない場合
#define MAX_PATH 260

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type

' 指定したファイル名に一致するファイルやディレクトリを検索
Declare Function Api_FindFirstFile& Lib "Kernel32" Alias "FindFirstFileA" (ByVal lpFileName$, lpFindFileData As WIN32_FIND_DATA)

' ファイル検索ハンドルをクローズ
Declare Function Api_FindClose& Lib "Kernel32" Alias "FindClose" (ByVal hFindFile&)

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function UnQualifyPath (Folder As String) As String
Function UnQualifyPath(Folder As String) As String

    Folder = Trim$(Folder) '空白削除

    If Right$(Folder, 1) = "¥" Then
        UnQualifyPath = Left$(Folder, Len(Folder) - 1) '¥削除
    Else
        UnQualifyPath = Folder
    End If
End Function

'=====
'=
'=====
Declare Function FolderExists (Folder As String) As Integer
Function FolderExists(Folder As String) As Integer
    Var hFile As Long
    Var wfd As WIN32_FIND_DATA
    Var Ret As Long

    '空白・¥があれば削除
    Folder = UnQualifyPath(Folder)

```

```

'フォルダ取得
hFile = Api_FindFirstFile(Folder, wfd)

'有効なディレクトリ属性が設定されるファイルハンドルが戻されたならばフォルダは存在
FolderExists = (hFile <> INVALID_HANDLE_VALUE) And (wfd.dwFileAttributes And
FILE_ATTRIBUTE_DIRECTORY)

Ret = Api_FindClose(hFile)
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Integer

    If Len(Edit1.GetWindowText) > 0 Then
        Ret = FolderExists(Edit1.GetWindowText)
        If Ret Then
            Text1.SetWindowText "フォルダは存在します"
        Else
            Text1.SetWindowText "フォルダは存在しません"
        End If
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

「フォルダの参照」-「C:¥」ダイアログを開く

「フォルダの参照」ダイアログを開きます。ダイアログ`内の選択したフォルダパスをEditBoxに表示させています。
 lstrcat("C:¥", "")とC:¥を指定していますので、他のドライブを指定した場合はEditBoxに表示されません。

SHBrowseForFolder 「フォルダの参照」ダイアログを開く

SHGetPathFromIDList アイテムIDリストをファイルシステムのパス名に変換

CoTaskMemFree アイテムIDリストの解放

lstrcat ある文字列の末尾に別の文字列を結合



```

'=====
'= 「フォルダの参照」ダイアログを開く
'= (SHBrowseForFolder.bas)
'=====
#include "Windows.bi"

```

```

Type BROWSEINFO
    hWndOwner      As Long
    pIDLRoot       As Long
    pszDisplayName  As Long

    lpszTitle      As Long

    ulFlags        As Long
    lpfnCallback   As Long
    lParam         As Long
    iImage         As Long
End Type

#define BIF_RETURNONLYFSDIRS 1
#define BIF_DONTGOBELOWDOMAIN 2

#define BIF_STATUSTEXT 4
#define BIF_RETURNFSANCESTORS 8

#define BIF_EDITBOX 10

#define MAX_PATH 260

' 「フォルダの参照」ダイアログを開き、選択されたフォルダのID値を取得
Declare Function Api_SHBrowseForFolder& Lib "Shell32" Alias "SHBrowseForFolder" (lpbi As BROWSEINFO)

' アイテムIDリストをファイルシステムのパス名に変換
Declare Function Api_SHGetPathFromIDList& Lib "Shell32" Alias "SHGetPathFromIDList" (ByVal pidList&, ByVal lpBuffer$)

' アイテムIDリストの解放
Declare Sub Api_CoTaskMemFree Lib "ole32" Alias "CoTaskMemFree" (ByVal hMem&)

' ある文字列の末尾に別の文字列を結合
Declare Function Api_lstrcat& Lib "kernel32" Alias "lstrcatA" (ByVal lpString1$, ByVal lpString2$)

Var Shared Edit1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var iNull As Integer
    Var lpIDList As Long
    Var Ret As Long
    Var sPath As String
    Var bi As BROWSEINFO

    bi.hWndOwner = GwthWnd
    bi.lpszTitle = Api_lstrcat("C:¥", "")
    bi.ulFlags = BIF_DONTGOBELOWDOMAIN

    lpIDList = Api_SHBrowseForFolder(bi)

    If lpIDList Then
        sPath = String$(MAX_PATH, 0)
        Ret = Api_SHGetPathFromIDList(lpIDList, sPath)
        Api_CoTaskMemFree lpIDList
        iNull = InStr(sPath, Chr$(0))
        If iNull Then
            sPath = Left$(sPath, iNull - 1)
        End If
    End If

    Edit1.SetWindowText sPath

```

```
End Sub
```

```
'=====
' =
'=====
```

```
While 1
    WaitEvent
Wend
Stop
End
```

「フォルダの参照」->「ネットワークの参照」ダイアログを開く

「フォルダの参照」-「ネットワークの参照」ダイアログを開きます。ダイアログ内の選択したフォルダパスをEditBoxに表示させています。

lstrcat("C:¥", "")とc:¥を指定していますので、他のドライブを指定した場合はEditBoxに表示されません。

SHBrowseForFolder 「フォルダの参照」ダイアログを開く

SHGetPathFromIDList アイテムIDリストをファイルシステムのパス名に変換

lstrcat ある文字列の末尾に別の文字列を結合



```
'=====
' = 「フォルダの参照」-「ネットワークの参照」を開く
' = (SHBrowseForFolder2.bas)
'=====
```

```
#include "Windows.bi"
```

```
Type BROWSEINFO
```

```
    hWndOwner      As Long
    pIDLRoot        As Long
    pszDisplayName  As Long
```

```
    lpszTitle       As Long
```

```
    ulFlags         As Long
```

```
    lpfnCallback    As Long
```

```
    lParam          As Long
```

```
    iImage          As Long
```

```
End Type
```

```
#define CSIDL_NETWORK &H12
```

```
#define BIF_RETURNONLYIFSDIRS &H1
```

```
#define MAX_PATH 260
```

'表示されるダイアログボックスのオーナーウィンドウを指定
'ルートフォルダを表すアイテムIDリストへのポインタを指定
'ユーザーに選択されたアイテムの表示名を格納するバッファのアドレスを指定

'ダイアログボックスのツリービューコントロールの上部に表示される文字列のアドレスを指定

'ダイアログボックスのオプションフラグを指定

'ネットワークコンピュータ (仮想フォルダ)

'ファイルシステムディレクトリのみを返す

```

' 「フォルダの参照」ダイアログを開き、選択されたフォルダのID値を取得
Declare Function Api_SHBrowseForFolder& Lib "Shell32" Alias "SHBrowseForFolder" (lpbi As BROWSEINFO)

' アイテムIDリストをファイルシステムのパス名に変換
Declare Function Api_SHGetPathFromIDList& Lib "Shell32" Alias "SHGetPathFromIDList" (ByVal pidList&, ByVal lpBuffer$)

' ある文字列の末尾に別の文字列を結合
Declare Function Api_lstrcat& Lib "kernel32" Alias "lstrcatA" (ByVal lpString1$, ByVal lpString2$)

Var Shared Edit1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var lpIDList As Long
    Var Buffer As String
    Var NetPath As String
    Var bi As BROWSEINFO
    Var Ret As Long

    bi.hWndOwner = GethWnd
    bi.pIDLRoot = CSIDL_NETWORK
    bi.lpszTitle = Api_lstrcat("ネットワークの参照", "")
    bi.ulFlags = BIF_RETURNONLYFSDIRS

    lpIDList = Api_SHBrowseForFolder(bi)

    If lpIDList Then
        Buffer = Space$(MAX_PATH)
        Ret = Api_SHGetPathFromIDList(lpIDList, Buffer)

        If Ret Then
            NetPath = Left$(Buffer, InStr(Buffer, Chr$(0)) - 1)
        Else
            A% = MsgBox("", "/パス取得失敗", 0, 2)
        End If
    End If

    Edit1.SetWindowText NetPath
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

フォルダ・ファイル数の取得

FindFirstFile 指定したファイル名に一致するファイルやディレクトリを検索
FindNextFile FindFirstFile() 関数で検出したファイルの次を検出
FindClose ファイル検索ハンドルをクローズ
GetFileAttributes 指定されたファイルまたはディレクトリの属性を取得
lstrlenW (Null文字で終了する) UNICODE文字列の文字数を返す



```
'=====
'= フォルダ・ファイル数の取得
'= (DirFileCount.bas)
'=====
#include "Windows.bi"
```

```
#define FILE_ATTRIBUTE_ARCHIVE &H20
#define FILE_ATTRIBUTE_COMPRESSED &H800
#define FILE_ATTRIBUTE_DIRECTORY &H10
#define FILE_ATTRIBUTE_HIDDEN &H2
#define FILE_ATTRIBUTE_NORMAL &H80
#define FILE_ATTRIBUTE_READONLY &H1
#define FILE_ATTRIBUTE_SYSTEM &H4
#define FILE_ATTRIBUTE_TEMPORARY &H100
#define INVALID_HANDLE_VALUE -1
#define MAX_PATH 260
#define vbDot 46
```

'アーカイブ属性
'圧縮属性
'ディレクトリ属性
'隠しファイル属性
'ファイル属性を持たない
'読み込み専用属性
'システムファイル属性
'一時ファイル属性
'見つからない場合

```
Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type
```

```
Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type
```

' 指定したファイル名に一致するファイルやディレクトリを検索

```
Declare Function Api_FindFirstFile& Lib "Kernel32" Alias "FindFirstFileA" (ByVal lpFileName$, lpFindFileData As WIN32_FIND_DATA)
```

' FindFirstFile() 関数で検出したファイルの次を検出

```
Declare Function Api_FindNextFile& Lib "Kernel32" Alias "FindNextFileA" (ByVal hFindFile&, lpFindFileData As WIN32_FIND_DATA)
```

' ファイル検索ハンドルをクローズ

```
Declare Function Api_FindClose& Lib "Kernel32" Alias "FindClose" (ByVal hFindFile&)
```

' 指定されたファイルまたはディレクトリの属性を取得

```
Declare Function Api_GetFileAttributes& Lib "Kernel32" Alias "GetFileAttributesA" (ByVal lpFileName$)
```

' (Null文字で終了する) UNICODE文字列の文字数を返す

```
Declare Function Api_lstrlenW& Lib "Kernel32" Alias "lstrlenW" (lpString As Any)
```

```
Var Shared List1 As Object
Var Shared Text1 As Object
Var Shared Edit(1) As Object
Var Shared Button(2) As Object
```



```

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
List1.SetWindowSize 258, 132
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
For i = 0 To 2
    If i < 2 then
        Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
        Edit(i).SetFontSize 14
    End If
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1)))
    Button(i).SetFontSize 12
Next

'=====
'= フォルダ・ファイル
'=====
Declare Function FilesCountAll(sSource As String, sFileType As String) As Long
Function FilesCountAll(sSource As String, sFileType As String) As Long
    Var wfd As WIN32_FIND_DATA
    Var hFile As Long
    Var Cnt As Long
    Var fCount As Long

    hFile = Api_FindFirstFile(sSource & sFileType, wfd)

    If hFile <> INVALID_HANDLE_VALUE Then
        Do
            If (Asc(wfd.cFileName) <> vbDot) Then
                Cnt = Cnt + 1
                List1.AddString Format$(Cnt, "#####:") & wfd.cFileName
                FilesCountAll = Cnt
            End If
            Loop Until Api_FindNextFile(hFile, wfd) = 0
        End If

        Ret = Api_FindClose(hFile)
    End Function

'=====
'=
'=====
Declare Function FilesCountByAttribute(sSource As String, sFileType As String,
dwAttributes As Long) As Long
Function FilesCountByAttribute(sSource As String, sFileType As String, dwAttributes As
Long) As Long
    Var wfd As WIN32_FIND_DATA
    Var hFile As Long
    Var Cnt As Long
    Var Ret As Long

    hFile = Api_FindFirstFile(sSource & sFileType, wfd)

    If (hFile <> INVALID_HANDLE_VALUE) Then
        Do
            If (dwAttributes And Api_GetFileAttributes(sSource & wfd.cFileName)) And
(Asc(wfd.cFileName) <> vbDot) Then
                Cnt = Cnt + 1
                List1.AddString Format$(Cnt, "#####:") & wfd.cFileName
                FilesCountByAttribute = Cnt
            End If
            Loop Until Api_FindNextFile(hFile, wfd) = 0
        End If

        Ret = Api_FindClose(hFile)
    End Function

'=====
'=
'=====
Declare Function FilesListByAttribute(Source As String, FileType As String, dwAttributes
As Long) As Long

```

```

Function FilesListByAttribute (Source As String, FileType As String, dwAttributes As Long)
As Long
    Var wfd As WIN32_FIND_DATA
    Var hFile As Long
    Var Cnt As Long
    Var Ret As Long

    hFile = Api_FindFirstFile (Source & FileType, wfd)

    If (hFile <> INVALID_HANDLE_VALUE) Then
        Do
            If (dwAttributes And Api_GetFileAttributes (Source & wfd.cFileName)) =
dwAttributes And (Asc(wfd.cFileName) <> vbDot) Then
                Cnt = Cnt + 1
                List1.AddString Format$(Cnt, "####:") & wfd.cFileName
                FilesListByAttribute = Cnt
            End If
            Loop Until Api_FindNextFile (hFile, wfd) = 0
        End If

        Ret = Api_FindClose (hFile)
    End Function

'=====
'=
'=====
Declare Function GetAttributeString (Attr As Long) As String
Function GetAttributeString (Attr As Long) As String
    Var Tmp As String

    If Attr And FILE_ATTRIBUTE_ARCHIVE      Then Tmp = Tmp & "ARCHIVE "
    If Attr And FILE_ATTRIBUTE_NORMAL        Then Tmp = Tmp & "NORMAL "
    If Attr And FILE_ATTRIBUTE_HIDDEN        Then Tmp = Tmp & "HIDDEN "
    If Attr And FILE_ATTRIBUTE_READONLY     Then Tmp = Tmp & "READONLY "
    If Attr And FILE_ATTRIBUTE_SYSTEM        Then Tmp = Tmp & "SYSTEM "
    If Attr And FILE_ATTRIBUTE_TEMPORARY    Then Tmp = Tmp & "TEMPORARY "
    If Attr And FILE_ATTRIBUTE_COMPRESSED   Then Tmp = Tmp & "COMPRESSED "
    If Attr And FILE_ATTRIBUTE_DIRECTORY    Then Tmp = Tmp & "DIRECTORY "

    GetAttributeString = Tmp
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Edit(0).SetWindowText "D:¥_FB_API_E¥"
    Edit(1).SetWindowText "*.*"
End Sub

'=====
'= Folder·File All Count
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Source As String
    Var FileType As String
    Var numFiles As Long

    Source = Edit(0).GetWindowText
    If Right$(Source, 1) <> "¥" Then
        Source = Source & "¥"
        Edit(0).SetWindowText Source
    End If

    FileType = Edit(1).GetWindowText

    List1.Resetcontent

```

```

        numFiles = FilesCountAll(Source, FileType)

        Text1.SetWindowText Str$(numFiles) & "個の フォルダ・ファイル"
End Sub

'=====
'= Archive Normal
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Source As String
    Var FileType As String
    Var dwAttributes As Long
    Var numFiles As Long

    Source = Edit(0).GetWindowText
    If Right$(Source, 1) <> "¥" Then
        Source = Source & "¥"
        Edit(0).SetWindowText Source
    End If

    FileType = Edit(1).GetWindowText

    dwAttributes = FILE_ATTRIBUTE_NORMAL Or FILE_ATTRIBUTE_ARCHIVE

    List1.Resetcontent

    numFiles = FilesCountByAttribute(Source, FileType, dwAttributes)

    Text1.SetWindowText Str$(numFiles) & "個の " & GetAttributeString(dwAttributes)
End Sub

'=====
'= Archive
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on()
    Var Source As String
    Var FileType As String
    Var dwAttributes As Long
    Var numFiles As Long

    Source = Edit(0).GetWindowText
    If Right$(Source, 1) <> "¥" Then
        Source = Source & "¥"
        Edit(0).SetWindowText Source
    End If

    FileType = Edit(1).GetWindowText

    dwAttributes = FILE_ATTRIBUTE_ARCHIVE

    List1.Resetcontent

    numFiles = FilesListByAttribute(Source, FileType, dwAttributes)

    Text1.SetWindowText Str$(numFiles) & "個の " & GetAttributeString(dwAttributes)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

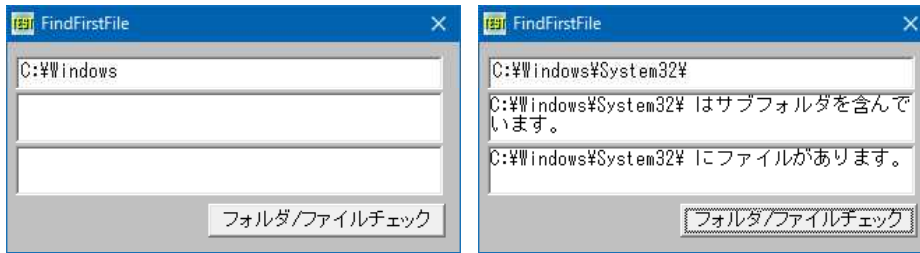
フォルダ・ファイルのチェック

フォルダやファイルの存在をチェックします。

FindFirstFile 指定したファイル名に一致するファイルやディレクトリを検索

FindNextFile FindFirstFile() 関数で検出したファイルの次を検出

FindClose ファイル検索ハンドルをクローズ



```
'=====
'= フォルダ・ファイルのチェック
'= (FindFirstFile.bas)
'=====
```

```
#include "Windows.bi"
```

```
#include "File.bi"
```

```
#define MAX_PATH 260
```

```
#define INVALID_HANDLE_VALUE -1
```

```
#define vbDirectory 16
```

'見つからない場合
'フォルダ

```
Type FILETIME
```

```
    dwLowDateTime    As Long
```

```
    dwHighDateTime   As Long
```

```
End Type
```

```
Type WIN32_FIND_DATA
```

```
    dwFileAttributes As Long
```

```
    ftCreationTime    As FILETIME
```

```
    ftLastAccessTime  As FILETIME
```

```
    ftLastWriteTime   As FILETIME
```

```
    nFileSizeHigh     As Long
```

```
    nFileSizeLow       As Long
```

```
    dwReserved0        As Long
```

```
    dwReserved1        As Long
```

```
    cFileName          As String * MAX_PATH
```

```
    cAlternate          As String * 14
```

```
End Type
```

' 指定したファイル名に一致するファイルやディレクトリを検索

```
Declare Function Api_FindFirstFile& Lib "kernel32" Alias "FindFirstFileA" (ByVal lpFileName$, lpFindFileData As WIN32_FIND_DATA)
```

' FindFirstFile() 関数で検出したファイルの次を検出

```
Declare Function Api_FindNextFile& Lib "kernel32" Alias "FindNextFileA" (ByVal hFindFile&, lpFindFileData As WIN32_FIND_DATA)
```

' ファイル検索ハンドルをクローズ

```
Declare Function Api_FindClose& Lib "kernel32" Alias "FindClose" (ByVal hFindFile&)
```

```
Var Shared Edit1 As Object
```

```
Var Shared Text1 As Object
```

```
Var Shared Text2 As Object
```

```
Var Shared Button1 As Object
```

```
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
```

```
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```

'=====
'=
'=====
Declare Function FolderContainsSubfolders(sRoot As String) As Integer
Function FolderContainsSubfolders(sRoot As String) As Integer
    Var wfd As WIN32_FIND_DATA
    Var hFile As Long
    Var Ret As Long

    hFile = Api_FindFirstFile(sRoot & "*.*", wfd)

    If hFile <> INVALID_HANDLE_VALUE Then
        Do
            If (wfd.dwFileAttributes And vbDirectory) Then

                If (Left$(wfd.cFileName, 1) <> ".") And (Left$(wfd.cFileName, 2) <> "..")
Then
                    FolderContainsSubfolders = True
                    Exit Do
                End If
            End If

            Loop While Api_FindNextFile(hFile, wfd)
        End If

        Ret = Api_FindClose(hFile)
End Function

'=====
'=
'=====
Declare Function FolderContainsFiles(sRoot As String) As Integer
Function FolderContainsFiles(sRoot As String) As Integer
    Var wfd As WIN32_FIND_DATA
    Var hFile As Long
    Var Ret As Long

    hFile = Api_FindFirstFile(sRoot & "*.*", wfd)

    If hFile <> INVALID_HANDLE_VALUE Then
        Do
            If (Not (wfd.dwFileAttributes And vbDirectory) = vbDirectory) Then

                FolderContainsFiles = True
                Exit Do
            End If

            Loop While Api_FindNextFile(hFile, wfd)
        End If

        Ret = Api_FindClose(hFile)
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "C:¥Windows"
    Button1.SetWindowText "フォルダ/ファイルチェック"
End Sub

'=====
'= パスの後尾に「¥」が無ければ付加
'=====
Declare Function qPath(sPath As String) As String
Function qPath(sPath As String) As String
    If Len(sPath) > 0 Then
        If Right$(sPath, 1) <> "¥" Then
            qPath = sPath & "¥"

```

```

        Else
            qPath = sPath
        End If
    Else
        qPath = ""
    End If
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sRootPath As String
    Var RetVal1 As Integer
    Var RetVal2 As Integer

    sRootPath = qPath(Edit1.GetWindowText)
    Edit1.SetWindowText sRootPath

    RetVal1 = FolderContainsSubfolders(sRootPath)
    RetVal2 = FolderContainsFiles(sRootPath)

    Select Case RetVal1
        Case True
            Text1.SetWindowText sRootPath & " はサブフォルダを含んでいます。"
        Case Else
            Text1.SetWindowText sRootPath & " 配下にサブフォルダはありません。"
        End Select

    Select Case RetVal2
        Case True
            Text2.SetWindowText sRootPath & " にファイルがあります。"
        Case Else
            Text2.SetWindowText sRootPath & " にファイルはありません。"
        End Select
    End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

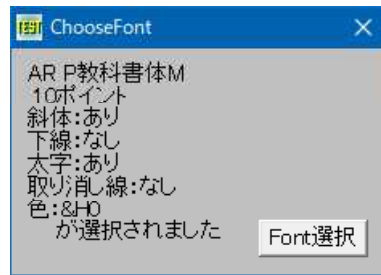
```

フォント選択ダイアログを開く

フォント選択ダイアログを開きます。

ChooseFont フォント選択のコモンダイアログボックスを表示
CopyMemory ある位置から別の位置にメモリブロックを移動
GlobalLock ヒープに確保されたメモリをロック
GlobalUnlock メモリブロックのロックを解除
GlobalAlloc メモリブロックを確保しハンドルを取得
GlobalFree メモリブロックのロックを解放

左:起動時・中:「フォント選択」ボタンクリックでダイアログ表示・右:フォントを選択し「OK」クリックで選択フォントを表示
 「摘要」「ヘルプ」ボタン、文字飾り、色、サイズ選択に表示されるサイズの制限などいろいろ設定が可能です。



```
'デバイス名の長さを示す定数
'フォーム名の長さを示す定数
'「適用」ボタンを追加する
'スクリーンフォント・するプリンタフォントを選択可能に
'取り消し線付き、下線付き、文字色選択可能にする
'フックプロシーヤを使う
'存在しないフォントを選択しようとしたときにエラーとなる
'LOGFONT構造体を使ってダイアログを初期化する
'フォントサイズの範囲を制限する
'プリンタで使えるフォントのみを表示
'スクリーンフォントのみを選択可能に
'ヘルプボタンを表示
'lpzStyleメンバを使用
'グリッピング精度
'デフォルトキャラクタセット
'デフォルトのピッチ
'デフォルト
'両面印刷
,
'可変ストローク幅を持つ、セリフ付きフォント
,
,
,
,
'デフォルト精度マッピング
'標準体
'スクリーンフォント
```

```
Type CHOOSEFONT
    lStructSize      As Long
    hDC              As Long
    lpLogFont        As Long
    iPointSize       As Long
```

```

    flags                As Long
    rgbColors            As Long
    lCustData            As Long
    lpfnHook            As Long
    lpTemplateName      As Long
    hInstance           As Long
    lpszStyle            As Long
    nFontType           As Integer
    MISSING_ALIGNMENT As Integer
    nSizeMin            As Long
    nSizeMax            As Long
End Type

```

' フォント選択のコモンダイアログボックスを表示

```

Declare Function Api_ChooseFont& Lib "comdlg32" Alias "ChooseFontA" (pChoosefont As CHOOSEFONT)

```

' ある位置から別の位置にメモリブロックを移動する関数の宣言

```

Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (hpvDest As Any, hpvSource As Any, ByVal cbCopy As Long)

```

' ヒープに確保されたメモリをロック

```

Declare Function Api_GlobalLock& Lib "kernel32" Alias "GlobalLock" (ByVal hMem&)

```

' メモリブロックのロックを解除

```

Declare Function Api_GlobalUnlock& Lib "kernel32" Alias "GlobalUnlock" (ByVal hMem&)

```

' メモリブロックを確保しハンドルを取得

```

Declare Function Api_GlobalAlloc& Lib "kernel32" Alias "GlobalAlloc" (ByVal wFlags&, ByVal dwBytes&)

```

' メモリブロックのロックを解放

```

Declare Function Api_GlobalFree& Lib "kernel32" Alias "GlobalFree" (ByVal hMem&)

```

```

Var Shared cf As CHOOSEFONT
Var Shared lf As LOGFONT

```

```

Var Shared Text1 As Object
Var Shared Button1 As Object

```

```

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

```

' =====
' =
' =====

```

```

Declare Function ShowFont() As String
Function ShowFont() As String

```

```

    Var hMem As Long
    Var pMem As Long
    Var Ret As Long

```

```

    lf.lfHeight = 13                ' デフォルト高さ
    lf.lfWidth = 0                  ' デフォルト幅
    lf.lfEscapement = 0             ' ベースラインと文字送りベクトルの間の角度
    lf.lfOrientation = 0           ' ベースラインとオリエンテーションベクトルの間の角度
    lf.lfWeight = FW_NORMAL        ' 標準
    lf.lfCharSet = DEFAULT_CHARSET ' デフォルトキャラクタセット
    lf.lfOutPrecision = OUT_DEFAULT_PRECIS ' デフォルト精度マッピング
    lf.lfClipPrecision = CLIP_DEFAULT_PRECIS ' デフォルトクリッピング精度
    lf.lfQuality = DEFAULT_QUALITY  ' デフォルト品質
    lf.lfPitchAndFamily = DEFAULT_PITCH Or FF_ROMAN ' デフォルトピッチ
    lf.lfFaceName = "MS ゴシック" & Chr$(0) ' デフォルト指定フォント名

```

```

    hMem = Api_GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(lf))
    pMem = Api_GlobalLock(hMem) ' ポインタを取得しロック
    CopyMemory ByVal pMem, lf, Len(lf) ' 構造体の内容をブロックにコピー

```

```

    cf.lStructSize = Len(cf) ' 構造体サイズ
    cf.hwndOwner = GethWnd ' ダイアログボックスのウィンドウ

```



```

cf.hDC = 0
cf.lpLogFont = pMem
cf.iPointSize = 100
cf.flags = CF_SCREENFONTS Or CF_EFFECTS Or CF_INITTOLOGFONTSTRUCT
cf.rgbColors = &H0
cf.lpTemplateName = 0
cf.hInstance = GethInst
cf.lpszStyle = StrAdr(String$(256, Chr$(0)))
cf.nFontType = REGULAR_FONTTYPE
cf.MISSING_ALIGNMENT = 0
cf.nSizeMin = 10
cf.nSizeMax = 72

Ret = Api_ChoseFont(cf)
If Ret <> 0 Then
    CopyMemory lf, ByVal pMem, Len(lf)
    ShowFont = Left$(lf.lfFaceName, InStr(lf.lfFaceName, Chr$(0)) - 1)
End If

Ret = Api_GlobalUnlock(hMem)
Ret = Api_GlobalFree(hMem)
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var txt As String

    txt = ""
    txt = ShowFont & Chr$(13, 10)
    txt = txt & Str$(cf.iPointSize / 10) & "ポイント" & Chr$(13, 10)
    If lf.lfItalic Then
        txt = txt & "斜体:あり" & Chr$(13, 10)
    Else
        txt = txt & "斜体:なし" & Chr$(13, 10)
    End If
    If lf.lfUnderline Then
        txt = txt & "下線:あり" & Chr$(13, 10)
    Else
        txt = txt & "下線:なし" & Chr$(13, 10)
    End If
    If lf.lfWeight >= 700 Then
        txt = txt & "太字:あり" & Chr$(13, 10)
    Else
        txt = txt & "太字:なし" & Chr$(13, 10)
    End If
    If lf.lfStrikeOut Then
        txt = txt & "取り消し線:あり" & Chr$(13, 10)
    Else
        txt = txt & "取り消し線:なし" & Chr$(13, 10)
    End If
    txt = txt & "色:&&H" & Hex$(cf.rgbColors) & Chr$(13, 10)
    Text1.SetWindowText txt & "    が選択されました"
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フォントの回転

フォントを回転及び拡大描画します。

CreateFontIndirect 論理フォントを作成

SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

TextOut 文字列を描画



```
'=====
'= 文字列の回転
'= (RotateFont.bas)
'=====
#include "Windows.bi"
```

```
#define LF_FACESIZE 32
```

```
Type LOGFONT
```

lfHeight	As Long	'文字セルまたは文字の高さ
lfWidth	As Long	'平均文字幅
lfEscapement	As Long	'文字送りの方向とx軸との角度
lfOrientation	As Long	'ベースラインとx軸との角度
lfWeight	As Long	'フォントの太さ
lfItalic	As Byte	'イタリック体指定
lfUnderline	As Byte	'下線付き指定
lfStrikeOut	As Byte	'打ち消し線付き指定
lfCharSet	As Byte	'キャラクタセット
lfOutPrecision	As Byte	'出力精度
lfClipPrecision	As Byte	'クリッピングの精度
lfQuality	As Byte	'出力品質
lfPitchAndFamily	As Byte	'ピッチとファミリ
lfFaceName(LF_FACESIZE)	As Byte	'フォント名

```
End Type
```

```
' 論理フォントを作成
```

```
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)
```

```
' 指定されたデバイスコンテキストのオブジェクトを選択
```

```
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)
```

```
' 文字列を描画
```

```
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal x&, ByVal
y&, ByVal lpString$, ByVal nCount&)
```

```
' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
```

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

```
' デバイスコンテキストを解放
```

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
Var Shared Text1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
```

```
Var Shared Txt As String
```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Txt = ":-)"

    Point(10, 10)
    Print Txt
End Sub

'=====
'=
'=====
Declare Sub Button1_On edec1 ()
Sub Button1_On ()
    Var lf As LOGFONT
    Var rFont As Long
    Var hDC As Long
    Var Deg As Integer
    Var Size As Integer
    Var i As Long
    Var Ret As Long

    Deg = 10
    Size = 60

    'デバイスコンテキスト取得
    hDC = Api_GetDC (GethWnd)

    For i = 1 To 360

        'オブジェクトを選択
        Ret = Api_SelectObject (hDC, rFont)

        '文字高さ設定
        lf.lfHeight = Size

        '角度設定
        lf.lfEscapement = Deg * i

        '論理フォントの作成
        rFont = Api_CreateFontIndirect (lf)

        '文字列描画
        Ret = Api_TextOut (hDC, 110, 80, Txt, Len (Txt))

        Text1.SetWindowText Str$ (Deg * i)

        Wait 1
    Next

    'デバイスコンテキスト解放
    Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

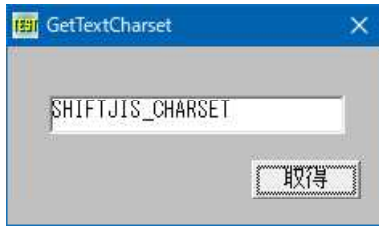
```

フォントの文字セット識別子を取得

GetTextCharset 指定されたデバイスコンテキストの現在のフォントの文字セット 識別子を取得

GetDC デバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストの解放



```
'=====
'= フォントの文字セット 識別子を取得
'= (GetTextCharset.bas)
'=====

#include "Windows.bi"

' 指定されたデバイスコンテキストの現在のフォントの文字セット識別子を取得
Declare Function Api_GetTextCharset& Lib "gdi32" Alias "GetTextCharset" (ByVal hDC&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define ANSI_CHARSET 0                'Windows標準文字セット
#define DEFAULT_CHARSET 1            'デフォルト(指定なし)
#define SYMBOL_CHARSET 2             'シンボル文字セット
#define MAC_CHARSET 77               '
#define SHIFTJIS_CHARSET 128         'シフトJIS文字セット
#define HANGEUL_CHARSET 129          'ハングル文字セット
#define JOHAB_CHARSET 130            '韓国版Windows文字セット
#define GB2312_CHARSET 134           '
#define CHINESEBIG5_CHARSET 136      '中国語文字セット
#define GREEK_CHARSET 161            '
#define TURKISH_CHARSET 162          '
#define VIETNAMESE_CHARSET 163       '
#define HEBREW_CHARSET 177           '中東語版Windows(ヘブライ語)文字セット
#define ARABIC_CHARSET 178           '中東語版Windows(アラビア語)文字セット
#define BALTIC_CHARSET 186           '
#define RUSSIAN_CHARSET 204          '
#define THAI_CHARSET 222             'タイ版Windows文字セット
#define EASTEUROPE_CHARSET 238       '
#define OEM_CHARSET 255              'OEM文字セット

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC As Long
    Var hChr As String
    Var Ret As Long

    hDC = Api_GetDC(GethWnd)

    Ret = Api_GetTextCharset(hDC)
```

```

Select Case Ret
    Case 0
        hChr = "ANSI_CHARSET"
    Case 1
        hChr = "DEFAULT_CHARSET"
    Case 2
        hChr = "SYMBOL_CHARSET"
    Case 128
        hChr = "SHIFTJIS_CHARSET"
    Case 129
        hChr = "HANGEUL_CHARSET"
    Case 134
        hChr = "GB2312_CHARSET"
    Case 136
        hChr = "CHINESEBIG5_CHARSET"
    Case 255
        hChr = "OEM_CHARSET"
End Select

Text1.SetWindowText hChr

Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

フォントリソースをロード

ディスクからフォントリソースをシステムフォントテーブルにロードします。

AddFontResource フォントリソースをディスクファイルからシステムフォントテーブルにロード

RemoveFontResource システムからフォントを削除

例では、Quake書体をロードし表示させています。当然 **Quake.ttf** が、指定フォルダになければなりません。



```

' =====
' = フォントリソースをロード
' = (AddFontResource.bas)
' =====
#include "Windows.bi"
#include "File.bi"

' フォントリソースをディスクファイルからシステムフォントテーブルにロード
Declare Function Api_AddFontResource& Lib "gdi32" Alias "AddFontResourceA" (ByVal lpFileName$)

' システムからフォントを削除
Declare Function Api_RemoveFontResource& Lib "gdi32" Alias "RemoveFontResourceA" (ByVal lpFilename$)

Var Shared Text1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

```

```

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 30
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

Var Shared fFileName As String

'=====
'= ロードするパス名とフォント名取得
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    fFileName = CrDir$("")
    If Right$(fFileName, 1) <> "¥" Then fFileName = fFileName & "¥"
    fFileName = fFileName & "quake.ttf"
    Text1.SetWindowText "QUAKE"
End Sub

'=====
'= フォントテーブルにロード
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_AddFontResource(fFileName)
    Text1.SetFontName "Quake"
    Text1.SetWindowText "QUAKE"
End Sub

'=====
'= テーブルから削除(元のフォントに戻しておく)
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    Ret = Api_RemoveFontResource(fFileName)
    Text1.SetFontName "MS ゴシック"
    Text1.SetWindowText "QUAKE"
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

複数コントロールの接続

同種のコントロールが沢山ある場合、個別に設定するのは大変です。
 例では、テキストボックス、エディットボックスのコントロール接続、フォント名、フォントサイズおよびIMEモードを一度に設定しています。

```

'=====
'= 複数コントロールの接続(フォントサイズも明示的に指定)
'=====
#include "Windows.bi"

Var Shared Edit(9) As Object
Var Shared Text(9) As Object

'=====
'=
'=====
Declare Sub MainForm_Start edecl ()
Sub MainForm_Start()
    For i = 0 To 9
        Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
        Edit(i).SetFontName "MS ゴシック"
        Edit(i).SetFontSize 14
        Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
        Text(i).SetFontName "MS ゴシック"
        Text(i).SetFontSize 14
        Select Case i
            Case 0, 3, 4, 5, 9
                Edit(i).SetImeMode 1           '仮名漢字ON
            Case 1
                Edit(i).SetImeMode 6           '半角カタカナ
            Case Else
                Edit(i).SetImeMode 2           '仮名漢字OFF
        End Select
    Next i
    Edit(1).SetFocus
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

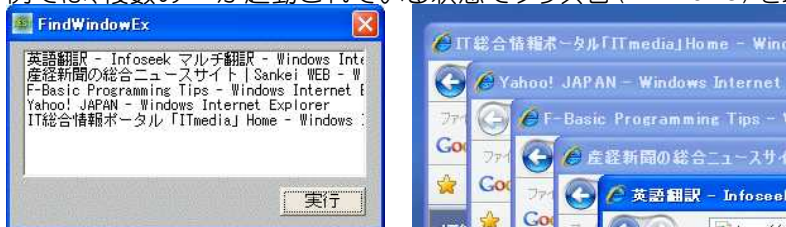
複数のウィンドウを検索

複数のウィンドウを検索します。

FindWindowEx クラス名、または キャプションを与えてウィンドウのハンドルを取得

GetWindowText ウィンドウのタイトル文字列を取得

例では、複数のIEが起動されている状態でクラス名(IEFrame)を取得しています。



```

'=====
'= 複数のウィンドウを検索
'= (FindWindowEx.bas)
'=====
#include "Windows.bi"

' クラス名、または キャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindowEx& Lib "user32" Alias "FindWindowExA" (ByVal
hWndParent&, ByVal hWndChildAfter&, ByVal lpzClass$, ByVal lpzWindow$)

```

・ ウィンドウのタイトル文字列を取得

```
Declare Function Api_GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hWnd&,
ByVal lpString$, ByVal cch&)

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hWnd As Long
    Var Title As String * 512
    Var Ret As Long

    List1.Resetcontent

    hWnd = 0
    Do
        hWnd = Api_FindWindowEx(0, hWnd, "IEFrame", ByVal 0)
        If hWnd = 0 Then Exit Do

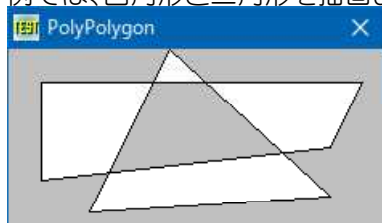
        Ret = Api_GetWindowText(hWnd, Title, Len(Title))
        List1.AddString Left$(Title, InStr(Title, Chr$(0)) - 1)
    Loop
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

複数の閉じた多角形を描画

PolyPolygon 複数の閉じた多角形を描画
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

例では、四角形と三角形を描画しています。



```
'=====
'= 複数の閉じた多角形を描画
'= (PolyPolygon.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type
```


' 複数の閉じた多角形を描画

```
Declare Function Api_PolyPolygon Lib "gdi32" Alias "PolyPolygon" (ByVal hdc As  
POINTAPI, lpPolyCounts As, ByVal nCount As)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC Lib "user32" Alias "GetDC" (ByVal hWnd As)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC Lib "user32" Alias "ReleaseDC" (ByVal hWnd As, ByVal hdc As)
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start ()
    Var pts(6) As POINTAPI
    Var numpoints(1) As Long
    Var hdc As Long
    Var Ret As Long
```

' デバイスコンテキスト取得

```
hdc = Api_GetDC (GethWnd)
```

' □のポイント

```
pts(0).x = 20 : pts(0).y = 20
pts(1).x = 220 : pts(1).y = 20
pts(2).x = 200 : pts(2).y = 60
pts(3).x = 20 : pts(3).y = 80
numpoints(0) = 4
```

' △のポイント

```
pts(4).x = 100 : pts(4).y = 0
pts(5).x = 50 : pts(5).y = 100
pts(6).x = 200 : pts(6).y = 90
numpoints(1) = 3
```

' □と△のポリゴン描画

```
Ret = Api_PolyPolygon(hdc, pts(0), numpoints(0), 2)
```

' デバイスコンテキスト解放

```
Ret = Api_ReleaseDC (GethWnd, hdc)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

二つのパスで共通しているプレフィックスを取得

PathCommonPrefix 二つのパスで共通しているプレフィックスを取得



```

'=====
'= 二つのパスで共通しているプレフィックスを取得
'= (PathCommonPrefix.bas)
'=====
#include "Windows.bi"

' 二つのパスで共通しているプレフィックスを取得
Declare Function Api_PathCommonPrefix& Lib "shlwapi" Alias "PathCommonPrefixA" (ByVal
pszFile1$, ByVal pszFile2$, ByVal achPath$)

Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'= Chr$(0)を取り除く
'=====
Declare Function TrimNull (item As String) As String
Function TrimNull (item As String) As String
    Var ePos As Integer

    ePos = Instr(item, Chr$(0))
    If ePos Then
        TrimNull = Left$(item, ePos - 1)
    Else
        TrimNull = item
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "c:¥mydir¥test¥myfile.ext"
    Edit2.SetWindowText "c:¥mydir¥testing¥hello.txt"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Buff As String
    Var path1 As String
    Var path2 As String
    Var Ret As Long

    Buff = String$(100, 0)
    path1 = Edit1.GetWindowText
    path2 = Edit2.GetWindowText

    Ret = Api_PathCommonPrefix(path1, path2, Buff)

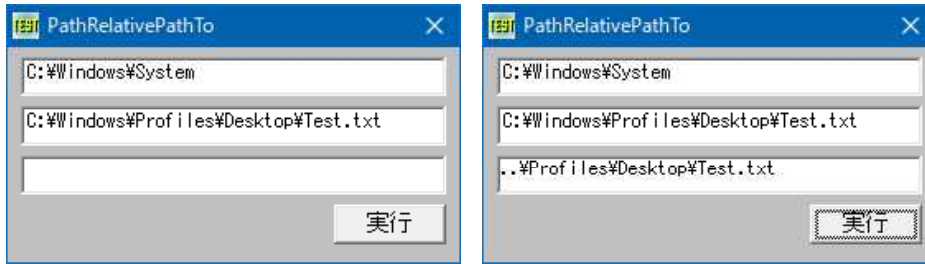
    Text1.SetWindowText TrimNull(Buff)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop : End

```

二つのファイル名の相対パスを取得

PathRelativePathTo 二つのファイル名の相対パスを取得
FILE_ATTRIBUTE_DIRECTORY (&H10) ディレクトリ属性
FILE_ATTRIBUTE_NORMAL (&H80) 他のファイル属性を持たない



```
'=====
'= 二つのファイル名の相対パスを取得
'= (PathRelativePathTo.bas)
'=====
#include "Windows.bi"

#define FILE_ATTRIBUTE_DIRECTORY &H10      'ディレクトリ属性
#define FILE_ATTRIBUTE_NORMAL &H80        '他のファイル属性を持たない

' 二つのパス間の相対パスを取得
Declare Function Api_PathRelativePathTo Lib "shlwapi" Alias "PathRelativePathToA"
    (ByVal pszPath$, ByVal pszFrom$, ByVal dwAttrFrom&, ByVal pszTo$, ByVal dwAttrTo&)

Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 12
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "C:\Windows\System"
    Edit2.SetWindowText "C:\Windows\Profiles\Desktop\Test.txt"
    Text1.SetWindowText ""
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var strPath As String
    Var Ret As Long

    strPath = Space$(256)

    Ret = Api_PathRelativePathTo(strPath, Edit1.GetWindowText,
FILE_ATTRIBUTE_DIRECTORY, Edit2.GetWindowText, FILE_ATTRIBUTE_NORMAL)

    Text1.SetWindowText strPath
End Sub

'=====
'=
'=====
```

```
While 1
    WaitEvent
Wend
Stop
End
```

二つの領域を結合し塗りつぶす

二つの領域を結合し、指定したブラシで塗りつぶします。

PaintRgn 現在のブラシで領域を塗りつぶす
GetStockObject スtockオブジェクトのハンドルを取得
CreateRectRgn 長方形のリージョンを作成
CombineRgn 既存の二つの領域を結合して新しい領域を作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
DeleteObject 論理オブジェクトを削除
CreateBrushIndirect LOGBRUSH構造体を定義して論理ブラシを作成
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

例では、二つの領域 (20, 20)-(110, 110) と (50, 50)-(140, 140) を結合 (CombineRgn) し、ランダムに作成したブラシ (CreateBrushIndirect) で塗りつぶしています。

RGN_AND、RGN_OR、RGN_XORでの状態も確認しています。



```
'=====
'= 二つの領域を結合し塗りつぶす
'= (PaintRgn.bas)
'=====
#include "Windows.bi"
```

```
Type LOGBRUSH
    lbStyle    As Long
    lbColor    As Long
    lbHatch    As Long
End Type
```

' 現在のブラシで領域を塗りつぶす

```
Declare Function Api_PaintRgn& Lib "gdi32" Alias "PaintRgn" (ByVal hDC&, ByVal hRgn&)
```

' スtockオブジェクトのハンドルを取得

```
Declare Function Api_GetStockObject& Lib "gdi32" Alias "GetStockObject" (ByVal nIndex&)
```

' 長方形のリージョンを作成

```
Declare Function Api_CreateRectRgn& Lib "gdi32" Alias "CreateRectRgn" (ByVal nLeftRect&,
ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&)
```

' 既存の二つの領域を結合して新しい領域を作成

```
Declare Function Api_CombineRgn& Lib "gdi32" Alias "CombineRgn" (ByVal hRgnDest&, ByVal
hRgnSrc1&, ByVal hRgnSrc2&, ByVal nCombineMode&)
```

' 指定されたデバイスコンテキストのオブジェクトを選択

```
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)
```

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる

```
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

' LOGBRUSH構造体を定義して論理ブラシを作成

```
Declare Function Api_CreateBrushIndirect& Lib "gdi32" Alias "CreateBrushIndirect"  
(lpLogBrush As LOGBRUSH)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
#define BS_SOLID 0  
#define WHITE_BRUSH 0  
#define LTGRAY_BRUSH 1  
#define GRAY_BRUSH 2  
#define BLACK_BRUSH 4  
#define RGN_AND 1  
#define RGN_OR 2  
#define RGN_XOR 3
```

'ソリッドブラシ
'白
'明るい灰色
'灰色
'黒
'リージョン同士のAND結合
'リージョン同士のOR結合
'リージョン同士のXOR結合

```
Var Shared Picture1 As Object  
Var Shared Radio1 As Object  
Var Shared Radio2 As Object  
Var Shared Radio3 As Object  
Var Shared Button1 As Object
```

```
Picture1.Attach GetDlgItem("Picture1")  
Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 14  
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14  
Radio3.Attach GetDlgItem("Radio3") : Radio3.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
```

```
'=  
'=====
```

```
Declare Function Rgn bdecl () As Integer  
Function Rgn()  
    Rgn = Val(Mid$(GetDlgRadioSelect("Radio1"), 6))  
End Function
```

```
'=====
```

```
'=  
'=====
```

```
Declare Sub Button1_on edec1 ()  
Sub Button1_on()
```

```
    Var LB As LOGBRUSH  
    Var hDC As Long  
    Var hRgn1 As Long  
    Var hRgn2 As Long  
    Var hOldBrush As Long  
    Var hBrush As Long  
    Var Ret As Long
```

```
    Picture1.cls
```

```
    Randomize Val(Right$(time$, 1))  
    LB.lbColor = RGB(Int(Rnd * 256), Int(Rnd * 256), Int(Rnd * 256))  
    LB.lbStyle = BS_SOLID
```

'ピクチャボックスのハンドル取得

```
hDC = Api_GetDC(Picture1.GethWnd)
```

'ブラシの取得

```
' hBrush = Api_GetStockObject(GRAY_BRUSH) '灰色ブラシの場合  
hBrush = Api_CreateBrushIndirect(LB)
```

'リージョンの作成

```
hRgn1 = Api_CreateRectRgn(20, 20, 110, 110)  
hRgn2 = Api_CreateRectRgn(50, 50, 140, 140)
```

リージョンの結合

```
Ret = Api_CombineRgn(hRgn1, hRgn1, hRgn2, Rgn)
```

リージョンの描画

```
hOldBrush = Api_SelectObject(hDC, hBrush)
```

```
Ret = Api_PaintRgn(hDC, hRgn1)
```

```
Ret = Api_SelectObject(hDC, hOldBrush)
```

後処理

```
Ret = Api_DeleteObject(hRgn1)
```

```
Ret = Api_DeleteObject(hRgn2)
```

```
Ret = Api_ReleaseDC(Picture1.GethWnd, hDC)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

フラッシュウィンドウ(1)

タイトルバーが点滅し目立つようにしたい

FlashWindow ウィンドウタイトル点滅



```
'=====
'= タイトルバーを点滅させる
'= (FlashWindow.bas)
'=====
```

```
#include "Windows.bi"
```

指定されたウィンドウを一度フラッシュする

```
Declare Function Api_FlashWindow& Lib "user32" Alias "FlashWindow" (ByVal hWnd&, ByVal bInvert&)
```

```
Var Shared Timer1 As Object
```

```
Var Shared Flag As Integer
```

```
Timer1.Attach GetDlgItem("Timer1")
```

```
'=====
'= フォームロード時の処理 (インターバル:0~65535)
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Timer1.SetInterval 50
```

```
    Flag = 1
```

```
End Sub
```

```
'=====
'= タイマーの処理
'=====
```

```
Declare Sub Timer1_Timer edec1 ()
```

```
Sub Timer1_Timer()
```

```
    Var Ret As Long
```

```

Ret = Api_FlashWindow (GethWnd, Flag)

If Flag = 0 Then
    Flag = 1
Else if Flag = 1 Then
    Flag = 0
End if
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

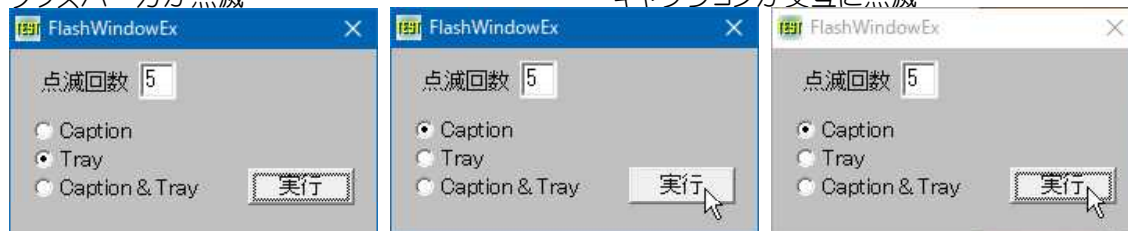
フラッシュウィンドウ(II)

タイトルバー・タスクバーを点滅させたい
点滅する箇所を選択できます。

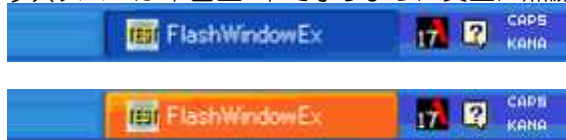
FlashWindowEx ウィンドウタイトル点滅

タスクバーが点滅

キャプションが交互に点滅



タスクバーが下図上・下で示すように交互に点滅



```

'=====
'= タイトルバー・タスクバーを点滅させる
'= (FlashWindowEx.bas)
'=====
#include "Windows.bi"

Type FLASHWINFO
    cbSize      As Long
    hWnd        As Long
    dwFlags      As Long
    uCount       As Long
    dwTimeout    As Long
End Type

' 点滅回数
' 点滅間隔 (ms)

' 指定されたウィンドウを一度フラッシュする
Declare Function Api_FlashWindowEx& Lib "user32" Alias "FlashWindowEx" (pfwi As FLASHWINFO)

#define FLASH_STOP &H0
#define FLASH_CAPTION &H1
#define FLASH_TRAY &H2
#define FLASH_ALL &H1 OR &H2
#define FLASH_TIMER &H4
#define FLASH_TIMERNOFG &H

' 点滅停止
' タイトルバーのみ点滅
' タスクバーのみ点滅
' FLASH_STOPが呼び出されるまで点滅
' FLASH_TIMERが呼び出されるまで点滅
' ウィンドウがアクティブになるまで点滅

Var Shared Radiol As Object

```

```

Var Shared Radio2 As Object
Var Shared Radio3 As Object
Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 14
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14
Radio3.Attach GetDlgItem("Radio3") : Radio3.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    ShowWindow -1
End Sub

'=====
'=
'=====
declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var fw As FLASHWINFO
    Var Ret As Long

    '点滅の設定
    fw.cbSize = Len(fw)
    fw.hWnd = GethWnd

    If Radio1.GetCheck = 1 Then
        fw.dwFlags = FLASH_CAPTION
    Else If Radio2.GetCheck = 1 Then
        fw.dwFlags = FLASH_TRAY
    Else
        fw.dwFlags = FLASH_ALL
    End If

    fw.uCount = Val(Edit1.GetWindowText)
    fw.dwTimeout = 500

    '点滅実行
    Ret = Api_FlashWindowEx(fw)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

フリガナ取得

変換中の文字列の情報を取得します。住所録などの作成においては必要不可欠

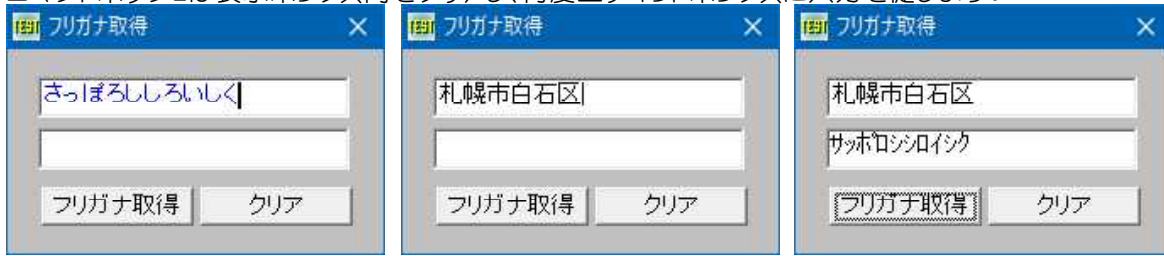
ImmGetCompositionString 変換中の文字列を取得

ImmGetContext IMEのコンテキストを取得

ImmReleaseContext IMEのコンテキストを解放

フォームにエディットボックス、テキストボックス、コマンドボタンを2個貼り付けます。
 エディットボックス内で文字列を仮名漢字変換しコマンドボタン1をクリックするとテキストボックスにフリガナが入ります。

コマンドボタン2は表示ボックス内をクリアし、再度エディットボックスに入力を促します。



```
'=====
'= フリガナ取得
'= (Hurigana.bas)
'=====
#include "Windows.bi"

' 変換中の文字列の情報を取得
Declare Function Api_ImmGetCompositionString& Lib "imm32" Alias
"ImmGetCompositionStringA" (ByVal hIMC&, ByVal dwIndex&, ByVal lpBuf$, ByVal dwBufLen&)

' 指定のウィンドウの入力コンテキストのハンドルを取得
Declare Function Api_ImmGetContext& Lib "imm32" Alias "ImmGetContext" (ByVal hWnd&)

' 入力コンテキストのハンドルを解放。
Declare Function Api_ImmReleaseContext& Lib "imm32" Alias "ImmReleaseContext" (ByVal
hWnd&, ByVal hIMC&)

#define GCS_RESULTREADSTR &H200

#define vbNullString ByVal 0
#define vbNullChar Chr$(0)

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Text1.SetWindowText ""
    Edit1.SetWindowText ""
    Edit1.SetFocus
End Sub

'=====
'= フリガナ取得
'=====
Declare Sub Button1_On edec1 ()
Sub Button1_On()
    Var hImc As Long
    Var hWnd As Long
    Var nSize As Long
    Var Buff As String
    Var Ret As Long

    hWnd = Edit1.GethWnd
    hImc = Api_ImmGetContext(hWnd)
    nSize = Api_ImmGetCompositionString(hImc, GCS_RESULTREADSTR, vbNullString, 0)
    Buff = String$(nSize, vbNullChar)

    Ret = Api_ImmGetCompositionString(hImc, GCS_RESULTREADSTR, Buff, Len(Buff))
    Text1.SetWindowText Buff
```

```

Ret = Api_ImmReleaseContext (hWnd, hImc)
End Sub

'=====
'= クリア
'=====

Declare Sub Button2_On edec1 ()
Sub Button2_On ()
    Text1.SetWindowText ""
    Edit1.SetWindowText ""
    Edit1.SetFocus
End Sub

'=====
'= イベントループ
'=====

While 1
    WaitEvent
Wend
Stop
End

```

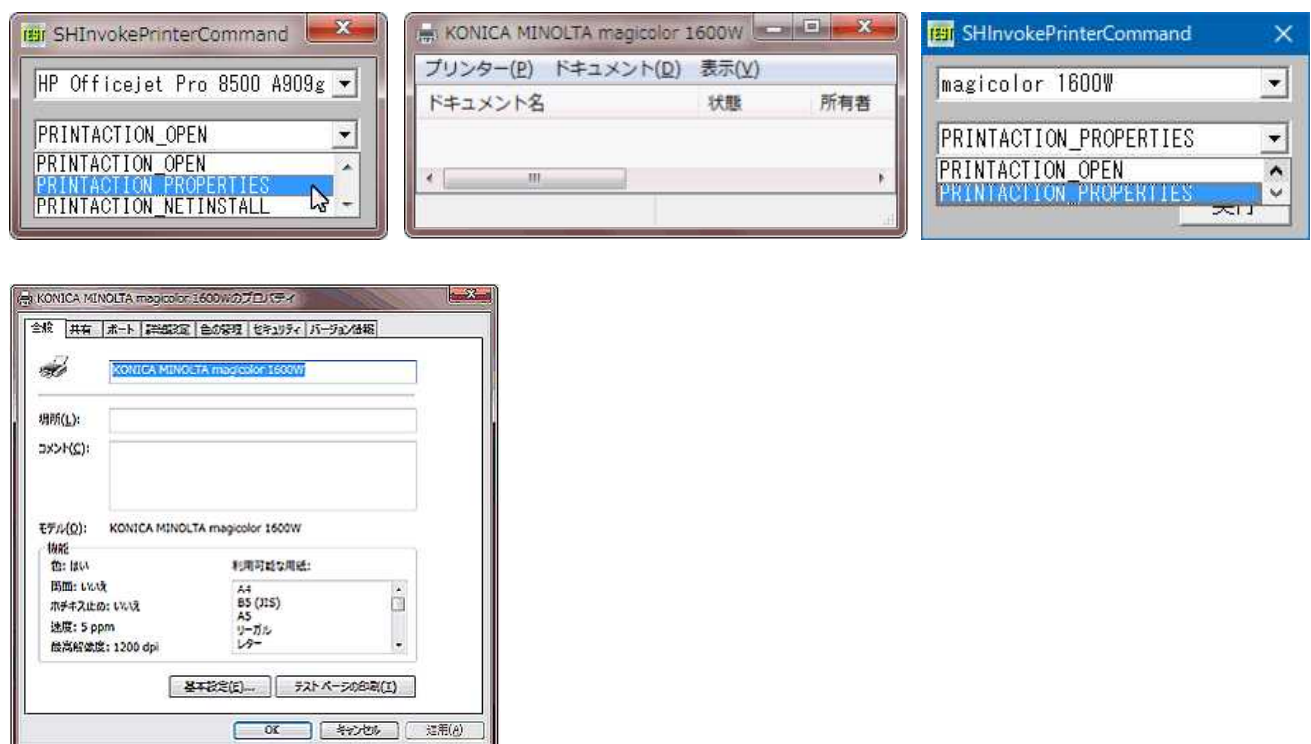
プリンタ設定のダイアグラム等を表示

プリンタの印刷設定用ダイアグラムを呼び出します。

CopyMemory メモリブロックの移動

EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙

SHInvokePrinterCommand プリンタのプロパティ等の取得



```

'=====
'= プリンタの設定ダイアグラムの表示
'= (SHInvokePrinterCommand3.bas)
'=====

```

```
#include "Windows.bi"
```

```
#define PRINTACTION_DOCUMENTDEFAULTS 6
```

```
#define PRINTACTION_NETINSTALL 2
```

```
#define PRINTACTION_NETINSTALLLINK 3
```

'lpBuf1で指定されたプリンタのプロパティを表示
(lpBuf2 パラメータは無視される)

'lpBuf1で指定されたネットワークプリンタをインストール
(lpBuf2 パラメータは無視される)

'lpBuf1で指定されたネットワークプリンタへのショートカット

```
#define PRINTACTION_OPEN 0
#define PRINTACTION_OPENNETPRN 5
#define PRINTACTION_PROPERTIES 1
#define PRINTACTION_SERVERPROPERTIES 7
#define PRINTACTION_TESTPAGE 4

#define PRINTER_ENUM_DEFAULT &H1
#define PRINTER_ENUM_LOCAL &H2
#define PRINTER_ENUM_NAME &H8
#define PRINTER_ENUM_SHARED &H20
#define MAX_DEVICENAME 64
```

トを作成 (lpBuf2 パラメータは、ショートカットを作成するフォルダのドライブ、パスを指定)
 'lpBuf1で指定されたプリンタを開く (lpBuf2パラメータは無視される)
 'lpBuf1で指定されたネットワークプリンタを開く (lpBuf2 パラメータは無視される)
 'lpBuf1で指定されたプリンタ用ディスプレイのプロパティページを表示
 'lpBuf1で指定されたプリンタサーバーのプロパティを表示 (lpBuf2 パラメータは無視される)
 'lpBuf1で指定されたプリンタのテストページを印刷 (lpBuf2 パラメータは無視される)
 'デフォルトのプリンタに関する情報を列挙
 'Nameの設定を無視して、ローカルプリンタを列挙
 'Nameで指定されたプリンタを列挙
 '共有属性を持つプリンタを列挙

```
Type PRINTER_INFO_5
    pPrinterName          As Long
    pPortName              As Long
    Attributes             As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type
```

' ある位置から別の位置にメモリブロックを移動する関数の宣言

```
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length&)
```

' 使用可能なプリンタ・プリントサーバーなどを列挙する

```
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&, pcReturned&)
```

' プリンタのプロパティ取得等のコマンドを実行

```
Declare Function Api_SHInvokePrinterCommand& Lib "Shell32" Alias "SHInvokePrinterCommandA" (ByVal hWnd&, ByVal uAction&, ByVal lpBuf1$, ByVal lpBuf2$, ByVal fModal&)
```

```
Var Shared PrinterName As String
```

```
Var Shared Combo1 As Object
Var Shared Combo2 As Object
Var Shared Button1 As Object
```

```
Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Combo2.Attach GetDlgItem("Combo2") : Combo2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long
```

```
'ローカルプリンタから検索
PrintServer = ""
```

```
'PRINTER_INFO_5構造体を受け取る
Level = 5
```

```
'バッファに必要なバイト数を調べる
```

```

Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
If Needed = 0 Then End

'全プリンタ情報を得る
Var Buffer(Needed - 1) As byte
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)

'構造体リストの準備
Var PI_5(Returned - 1) As PRINTER_INFO_5

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
    PrinterName = KLeft$(PrinterName, KInStr(1, PrinterName, Chr$(0)) - 1)

    Combo1.AddString PrinterName
Next

Combo2.AddString "PRINTACTION_OPEN"
Combo2.AddString "PRINTACTION_PROPERTIES"
Combo2.AddString "PRINTACTION_NETINSTALL"
Combo2.AddString "PRINTACTION_NETINSTALLLINK"
Combo2.AddString "PRINTACTION_TESTPAGE"
Combo2.AddString "PRINTACTION_OPENNETPRN"
Combo2.AddString "PRINTACTION_DOCUMENTDEFAULTS"
Combo2.AddString "PRINTACTION_SERVERPROPERTIES"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Index As Long
    Var Ret As Long

    '選択されているプリンタ名を取得
    PrinterName = Combo1.GetWindowText

    Index = Combo2.GetCursel

    'プリンタ設定に関するダイアログを表示
    Ret = Api_SHInvokePrinterCommand(GethWnd, Index, PrinterName, ByVal 0, 1)
End Sub

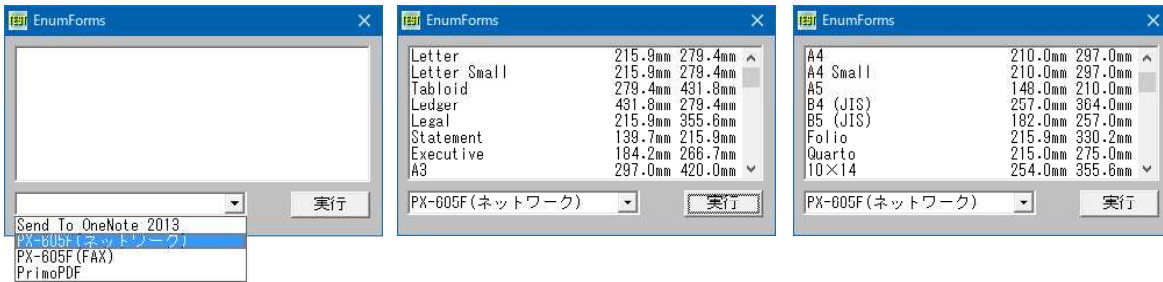
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタで利用できる用紙とサイズを取得

EnumPrinters プリンタの詳細な情報を取得
OpenPrinter プリンタオブジェクトをオープン
EnumForms プリンタで利用できる用紙を列挙
MoveMemory メモリの指定領域をコピー

lstrcpy 文字列をコピーする
ClosePrinter プリンタオブジェクトを閉じる



```
'=====
'= プリンタで使える用紙とサイズを取得
'= (EnumForms.bas)
'=====
```

```
#include "Windows.bi"
```

```
Type PRINTER_DEFAULTS
```

```
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
```

```
End Type
```

```
Type SIZE
```

```
    cx As Long
    cy As Long
```

```
End Type
```

```
Type RECT
```

```
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
```

```
End Type
```

```
Type FORM_INFO_1
```

```
    Flags As Long
    pName As Long
    Size As SIZE
    ImageableArea As RECT
```

```
End Type
```

```
Type PRINTER_INFO_5
```

```
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
```

```
End Type
```

```
#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙
#define MAX_DEVICENAME 64
```

```
' プリンタの詳細な情報を取得
```

```
Declare Function Api_EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cdBuf&, pcbNeeded&, pcReturned&)
```

```
' プリンタオブジェクトをオープン
```

```
Declare Function Api_OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)
```

・ プリンタで利用できる用紙を列挙

```
Declare Function Api_EnumForms& Lib "winspool.drv" Alias "EnumFormsA" (ByVal hPrinter&,
ByVal Level&, pForm As Any, ByVal cbBuf&, pcbNeeded&, pcReturned&)
```

・ メモリの指定領域をコピー

```
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal Length&)
```

・ 文字列をコピーする

```
Declare Function Api_lstrcpy& Lib "Kernel32" Alias "lstrcpy" (lpszString1 As Any,
lpszString2 As Any)
```

・ プリンタオブジェクトを閉じる

```
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)
```

```
Var Shared List1 As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var PrinterServer As String
    Var PrinterName As String
    Var PortName As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var cnt As Long
    Var Ret As Long
```

```
    PrinterServer = ""
    Level = 5
```

```
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrinterServer, Level, Chr$(0), 0, Needed,
Returned)
    If Needed = 0 Then End
```

```
    Var Buffer(Needed - 1) As Byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrinterServer, Level, Buffer(0), Needed,
Needed, Returned)
```

```
    Var pi5(Returned - 1) As PRINTER_INFO_5
```

```
    For cnt = 0 To Returned - 1
        MoveMemory pi5(cnt), Buffer(cnt * Len(pi5(cnt))), Len(pi5(cnt))
```

・ プリンタ名取得

```
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    MoveMemory PrinterName, ByVal pi5(cnt).pPrinterName, Len(PrinterName)
    PrinterName = Left$(PrinterName, InStr(PrinterName, Chr$(0)) - 1)
    Comb1.AddString PrinterName
```

```
Next
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var DeviceName As String
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
```

```

Var Level As Long
Var Needed As Long
Var Returned As Long
Var FormName As String * 64
Var cnt As Long
Var txt As String
Var Ret As Long

'リストビューを初期化
List1.Resetcontent

'プリンタ名を指定
DeviceName = Combo1.GetWindowText

'プリンタアクセス権を指定
pd.DesiredAccess = PRINTER_ALL_ACCESS

'プリンタのオブジェクトハンドルを取得
Ret = Api_OpenPrinter(DeviceName, hPrinter, pd)

'構造体のレベルを指定
Level = 1

'バッファに必要なサイズを取得
Ret = Api_EnumForms(hPrinter, Level, 0, 0, Needed, Returned)

'用紙情報が取得できないときは
If Needed <= 0 Then
    List1.AddString "取得できません。"

    '取得できないときの終了処理へ分岐
    GoTo *ClosePrt
End If

'バッファを確保
Var Buffer(Needed - 1) As Byte

'用紙情報を取得
Ret = Api_EnumForms(hPrinter, Level, Buffer(0), Needed, Needed, Returned)

'取得した用紙情報の構造体を確保
Var Info(Returned - 1) As FORM_INFO_1

'取得した用紙情報を構造体へ移動
MoveMemory Info(0), Buffer(0), Len(Info(0)) * Returned

'バッファに用紙情報がある間は
For cnt = 0 To Returned - 1

    '用紙情報を列挙(用紙名を複写)
    Ret = Api_lstrcpy(FormName, ByVal Info(cnt).pName)

    '用紙名を取得
    txt = Left$(Left$(FormName, InStr(FormName, Chr$(0)) - 1) & Space$(24), 24)

    '幅を取得
    txt = txt & " " & Format$(Info(cnt).Size.cx / 1000, "###.##mm")

    '高さを取得
    txt = txt & " " & Format$(Info(cnt).Size.cy / 1000, "###.##mm")

    List1.AddString txt
Next cnt

*ClosePrt                                'プリンタオブジェクトをクローズ

Ret = Api_ClosePrinter(hPrinter)
End Sub

```

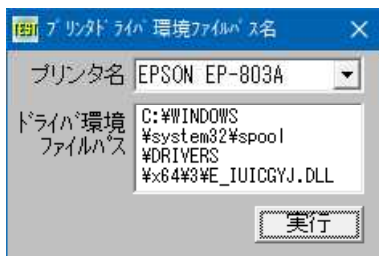
```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタドライバ環境ファイルパス名を取得

OpenPrinter プリンタオブジェクトをオープン
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
SHInvokePrinterCommand プリンタのプロパティ取得等のコマンドを実行
DeviceCapabilities プリンタデバイスドライバの能力を取得
GetPrinterDriver プリンタのドライバ情報を取得
MoveMemory メモリの指定領域をコピー
lstrcpy 文字列をコピー
ClosePrinter プリンタオブジェクトを閉じる



```

'=====
'= プリンタドライバ環境ファイルパス名を取得
'= (PrtDriverConfigFile.bas)
'=====
#include "Windows.bi"

```

```

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

```

```

Type DRIVER_INFO_2
    cVersion As Long
    pName As Long
    pEnvironment As Long
    pDriverPath As Long
    pDataFile As Long
    pConfigFile As Long
End Type

```

```

Type PRINTER_INFO_5
    pDeviceName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionColorDeviceryTimeOut As Long
End Type

```

```

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '

```

```

#define PRINTER_ENUM_DEFAULT &H1 'デフォルトのプリンタに関する情報を列挙
#define PRINTER_ENUM_LOCAL &H2 'Nameの設定を無視して、ローカルプリンタを列挙
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙

```



```

#define PRINTER_ENUM_SHARED &H20          '共有属性を持つプリンタを列挙
#define MAX_DEVICENAME 64
#define DC_COLORDEVICE 32

' プリントオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pDeviceName$, phPrinter&, pDefault As Any)

' 使用可能なプリンタ・プリントサーバーなどを列挙する
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)

' プリンタのプロパティ取得等のコマンドを実行
Declare Function Api_SHInvokePrinterCommand& Lib "Shell32" Alias
"SHInvokePrinterCommandA" (ByVal hWnd&, ByVal uAction&, ByVal lpBuf1$, ByVal lpBuf2$,
ByVal fModal&)

' プリンタデバイスドライバの能力を取得
Declare Function Api_DeviceCapabilities& Lib "winspool.drv" Alias "DeviceCapabilitiesA"
(ByVal lpDeviceName$, ByVal lpPort$, ByVal iIndex&, lpOutput As Any, lpDevMode As Any)

' プリンタのドライバ情報を取得
Declare Function Api_GetPrinterDriver& Lib "winspool.drv" Alias "GetPrinterDriverA"
(ByVal hPrinter&, ByVal pEnvironment$, ByVal Level&, pDriverInfo As Any, ByVal cdBuf&,
pcbNeeded&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal Length&)

' 文字列をコピーする
Declare Function Api_lstrcpy& Lib "Kernel32" Alias "lstrcpy" (lpszString1 As Any,
lpszString2 As Any)

' プリントオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

Var Shared Text(2) As Object
Var Shared Comb1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

For i = 0 To 1
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared DeviceName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var ColorDevice As Long

    'ローカルプリンタから検索
    PrintServer = ""

```

```

'PRINTER_INFO_5構造体を受け取る
Level = 5

'バッファに必要なバイト数を調べる
ColorDevice = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0,
Needed, Returned)
If Needed = 0 Then End

'全プリンタ情報を得る
Var Buffer(Needed - 1) As byte
ColorDevice = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0),
Needed, Needed, Returned)

'構造体リストの準備
Var PI_5(Returned - 1) As PRINTER_INFO_5

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    MoveMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

    'プリンタ名を得る
    DeviceName = String$(MAX_DEVICENAME, Chr$(0))
    MoveMemory DeviceName, ByVal PI_5(CT).pDeviceName, Len(DeviceName)
    DeviceName = KLeft$(DeviceName, KInStr(1, DeviceName, Chr$(0)) - 1)

    Combol.AddString DeviceName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var Level As Long
    Var Needed As Long
    Var di2 As DRIVER_INFO_2
    Var ConfigFile As String * 516
    Var Ret As Long

    DeviceName = Combol.GetWindowText
    If DeviceName = "" Then
        A% = MsgBox(GetWindowText, "プリンタ名を指定してください!", 0, 2)
        Exit Sub
    End If

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(DeviceName, hPrinter, pd)

    '構造体のレベルを指定
    Level = 2

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinterDriver(hPrinter, ByVal 0, Level, ByVal 0, 0, Needed)

    'バッファを確保
    Var Buffer(Needed - 1)

    'プリンタのドライバ情報を取得
    Ret = Api_GetPrinterDriver(hPrinter, ByVal 0, Level, Buffer(0), Needed, Needed)

    '取得したプリンタのドライバ情報を構造体へ移動
    MoveMemory di2, Buffer(0), Len(di2)

```

```

'ドライバ環境ファイルパス名を表示
'ドライバ環境ファイルパス名を複写
Ret = Api_lstrcpy(ConfigFile, ByVal di2.pConfigFile)

'ドライバ環境ファイルパス名を切り出し
Edit1.SetWindowText Left$(ConfigFile, InStr(ConfigFile, Chr$(0)) - 1)

'プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'=
'=====
Declare Sub Combo1_Click edec1 ()
Sub Combo1_Click()
    Edit1.SetWindowText ""
End Sub

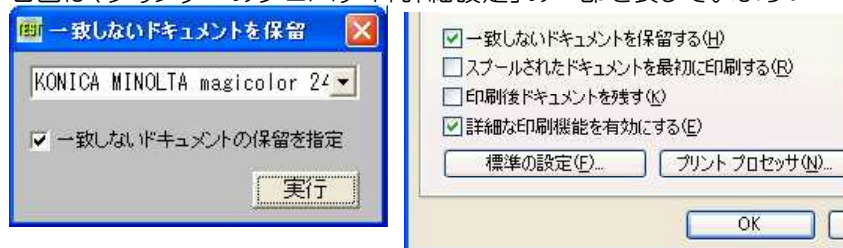
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの一致しないドキュメントの保留を設定

EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
OpenPrinter プリンタオブジェクトをオープン
GetPrinter プリンタの詳細な情報を取得
MoveMemory メモリの指定領域をコピー
SetPrinter プリンタの詳細な情報を設定
ClosePrinter プリンタオブジェクトを閉じる

右図は、プリンタのプロパティ「詳細設定」の一部を表しています。



```

'=====
'= プリンタの一致しないドキュメントの保留を設定
'= (PRINTER_ATTRIBUTE_ENABLE_DEVQ.bas)
'=====

```

```
#include "Windows.bi"
```

```

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

```

```

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

```

```

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define PRINTER_ATTRIBUTE_ENABLE_DEVQ &H80 '一致しないドキュメントの保留
#define ERROR_INVALID_SHARENAME 1215 '共有名無効の拡張エラーを示す定数
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙
#define MAX_DEVICENAME 64

' 使用可能なプリンタ・プリントサーバーなどを列挙する
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal Length&)

' プリンタの詳細な情報を設定
Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)
Var Shared Combo1 As Object
Var Shared Check1 As Object
Var Shared Button1 As Object

Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Check1.Attach GetDlgItem("Check1") : Check1.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    'ローカルプリンタから検索(サーバー名 ¥¥xxx 指定可)
    PrintServer = ""

    'PRINTER_INFO_5構造体を受け取る
    Level = 5

    'バッファに必要なバイト数を調べる
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
    If Needed = 0 Then End

    '全プリンタ情報を得る
    Var Buffer(Needed - 1) As byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)

```

```

'構造体リストの準備
Var pi5(Returned - 1) As PRINTER_INFO_5
For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    MoveMemory pi5(CT), Buffer(CT * Len(pi5(CT))), Len(pi5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    MoveMemory PrinterName, ByVal pi5(CT).pPrinterName, Len(PrinterName)
    PrinterName = kLeft$(PrinterName$, KInStr(1, PrinterName, Chr$(0)) - 1)

    Combol.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var PrinterDeviceName As String
    Var pd As PRINTER_DEFAULTS
    Var pi5 As PRINTER_INFO_5
    Var hPrinter As Long
    Var Level As Long
    Var Needed As Long
    Var Ret As Long

    'プリンタ名を指定
    PrinterDeviceName = Combol.GetWindowText

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(PrinterDeviceName, hPrinter, pd)

    '構造体のレベルを指定
    Level = 5

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Needed)

    'バッファを確保
    Var Buffer(Needed - 1) As Byte

    '詳細なプリンタ情報を取得
    Ret = Api_GetPrinter(hPrinter, Level, Buffer(0), Needed, Needed)

    '取得した詳細なプリンタ情報を構造体へ移動
    MoveMemory pi5, Buffer(0), Len(pi5)

    '設定する条件によって分岐
    Select Case Check1.GetCheck

        '指定する
        Case 1

            '属性との論理和を指定
            pi5.Attributes = pi5.Attributes Or PRINTER_ATTRIBUTE_ENABLE_DEVQ

        '指定しない
        Case 0

            '属性との排他的論理和を指定
            pi5.Attributes = pi5.Attributes Xor PRINTER_ATTRIBUTE_ENABLE_DEVQ
    End Select

```

```

' 詳細なプリンタ情報を設定
Ret = Api_SetPrinter(hPrinter, Level, pi5, 0)

' プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの印刷ジョブを取得 (1)

プリンタの印刷ジョブを列挙します。

OpenPrinter プリンタオブジェクトをオープン

ClosePrinter プリンタオブジェクトを閉じる

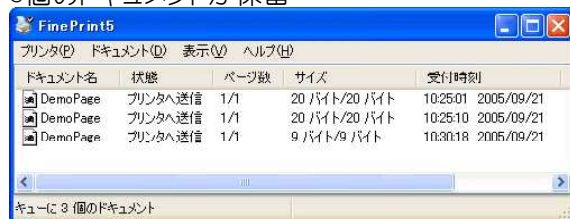
EnumJobs プリンタの印刷ジョブを列挙

例では、デフォルトプリンタ名「FinePrint5」を直接指定しています。

確認の為、「FinePrint5」をダブルクリック



3個のドキュメントが保留



```

' =====
' = プリンタの印刷ジョブを列挙
' = (EnumJobs.bas)
' =====

```

' プリンタオブジェクトをオープン

```

Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

```

' プリンタオブジェクトを閉じる

```

Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

```

' プリンタの印刷ジョブを列挙

```

Declare Function Api_EnumJobs& Lib "winspool.drv" Alias "EnumJobsA" (ByVal hPrinter&,
ByVal FirstJob&, ByVal NoJobs&, ByVal Level&, pJob As Any, ByVal cdBuf&, pcb1Needed&,
pcReturned&)

```

```

Var hPrinter As Long
Var lNeeded As Long
Var lReturned As Long

```

```

Var lJobCount As Long
Var Ret As Long

Ret = Api_OpenPrinter("FinePrint5", hPrinter, ByVal 0)
Ret = Api_EnumJobs(hPrinter, 0, 99, 1, ByVal 0, 0, lNeeded, lReturned)
If lNeeded > 0 Then
    dim JobsBuffer(lNeeded - 1) As byte
    Ret = Api_EnumJobs(hPrinter, 0, 99, 1, JobsBuffer(0), lNeeded, lNeeded, lReturned)
    If lReturned > 0 Then
        lJobCount = lReturned
    Else
        lJobCount = 0
    End If
Else
    lJobCount = 0
End If

Ret = Api_ClosePrinter(hPrinter)
Print "キューに" + Str$(lJobCount) & "個のドキュメントがあります！"

Stop
End

```

プリンタの印刷ジョブを取得(II)

プリンタジョブ(保留状態)を取得します。

MoveMemory メモリブロックを別の領域に移動する

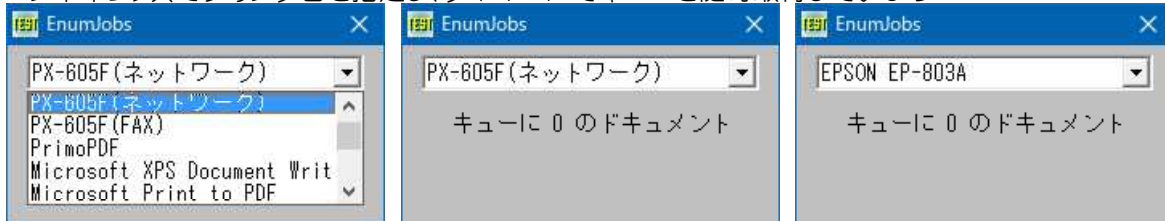
OpenPrinter プリンタオブジェクトをオープン

ClosePrinter プリンタオブジェクトを閉じる

EnumJobs プリンタの印刷ジョブを列挙

EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙

コンボボックスでプリンタ名を指定し、タイマーにてキューを随時取得しています。



```

'=====
' プリンタジョブの状態を表示
' (EnumJobs.bas)
'=====
#include "Windows.bi"

```

```

Type PRINTER_INFO_5
    pPrinterName           As Long
    pPortName              As Long
    Attributes              As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

```

' ある位置から別の位置にメモリブロックを移動する関数の宣言

```

Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)

```

' プリンタオブジェクトをオープン

```

Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

```

' プリンタオブジェクトを閉じる

```

Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

```

・プリンタの印刷ジョブを列挙

```
Declare Function Api_EnumJobs& Lib "winspool.drv" Alias "EnumJobsA" (ByVal hPrinter&,
ByVal FirstJob&, ByVal NoJobs&, ByVal Level&, pJob As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)
```

・使用可能なプリンタ・プリントサーバーなどを列挙する

```
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)
```

```
#define PRINTER_ENUM_DEFAULT &H1
#define PRINTER_ENUM_LOCAL &H2
#define PRINTER_ENUM_NAME &H8
#define PRINTER_ENUM_SHARED &H20
#define MAX_DEVICENAME 64
```

'デフォルトのプリンタに関する情報を列挙
'Nameの設定を無視して、ローカルプリンタを列挙
'Nameで指定されたプリンタを列挙
'共有属性を持つプリンタを列挙

```
Var Shared Timer1 As Object
Var Shared Text1 As Object
Var Shared Combo1 As Object
```

```
Timer1.Attach GetDlgItem("Timer1")
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Var Shared PrinterName As String
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long
```

'ローカルプリンタから検索

```
PrintServer = ""
```

'PRINTER_INFO_5構造体を受け取る

```
Level = 5
```

'バッファに必要なバイト数を調べる

```
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
If Needed = 0 Then End
```

'全プリンタ情報を得る

```
Var Buffer(Needed - 1) As byte
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)
```

'構造体リストの準備

```
Var PI_5(Returned - 1) As PRINTER_INFO_5
```

```
For CT = 0 To Returned - 1
```

'バッファから構造体1つ分を抜き取る

```
CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))
```

'プリンタ名を得る

```
PrinterName = String$(MAX_DEVICENAME, Chr$(0))
CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
PrinterName = KLeft$(PrinterName, KInStr(1, PrinterName, Chr$(0)) - 1)
```

```
    Combo1.AddString PrinterName
Next
Timer1.SetInterval 50
Timer1.Enable -1
```



```

End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    Var hPrinter As Long
    Var Ret As Long

    PrinterName = Combo1.GetText (Combo1.GetCursel)
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var hPrinter As Long
    Var Needed As Long
    Var Returned As Long
    Var JobCount As Long
    Var Ret As Long

    Ret = Api_OpenPrinter (PrinterName, hPrinter, ByVal 0)
    Ret = Api_EnumJobs (hPrinter, 0, 99, 1, ByVal 0, 0, Needed, Returned)
    If Needed > 0 Then
        Var JobsBuffer (Needed - 1) As Byte
        Ret = Api_EnumJobs (hPrinter, 0, 99, 1, JobsBuffer(0), Needed, Needed, Returned)
        If Returned > 0 Then
            JobCount = Returned
        Else
            JobCount = 0
        End If
    Else
        JobCount = 0
    End If

    Ret = Api_ClosePrinter (hPrinter)

    Text1.SetWindowText "キューに" & Str$(JobCount) & " のドキュメント"
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

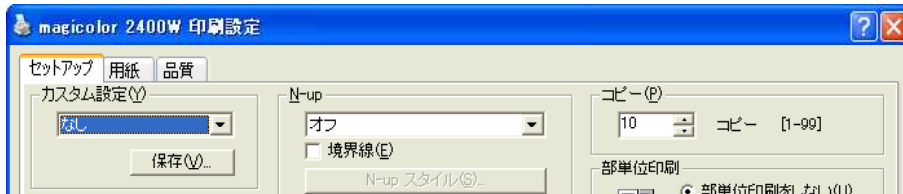
```

プリンタの印刷部数を設定

指定したプリンタの印刷部数を設定します。

- EnumPrinters** 使用可能なプリンタ・プリントサーバーなどを列挙する
- CopyMemory** ある位置から別の位置にメモリブロックを移動
- OpenPrinter** プリンタオブジェクトをオープン
- GetPrinter** プリンタの詳細な情報を取得
- SetPrinter** プリンタの詳細な情報を設定
- ClosePrinter** プリンタオブジェクトを閉じる

プリンタを選択し、印刷部数を入力。その結果を表示しています。



```
'=====
'= プリンタの印刷部数を設定
'= (PrinterDmCopies.bas)
'=====
#include "Windows.bi"
```

```
#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数の宣言
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す定数の宣言
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザ権限を示す定数の宣言
#define PRINTER_ALL_ACCESS (&HF0000 Or &H4 Or &H8) '
#define PRINTER_ENUM_DEFAULT &H1 'デフォルトのプリンタに関する情報を列挙
#define PRINTER_ENUM_LOCAL &H2 'Nameの設定を無視して、ローカルプリンタを列挙
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙
#define PRINTER_ENUM_SHARED &H20 '共有属性を持つプリンタを列挙
#define MAX_DEVICENAME 64

#define CCHDEVICENAME 32 'デバイス名の長さを示す定数
#define CCHFORMNAME 32 'フォーム名の長さを示す定数
#define DM_COPIES &H100 '用紙が有効であることを示す定数
```

```
Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type
```

```
Type DEVMODE
    dmDeviceName (CCHDEVICENAME - 1) As Byte
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    dmFormName (CCHFORMNAME - 1) As Byte
    dmLogPixels As Integer
    dmBitsPerPel As Long
    dmPelsWidth As Long
    dmPelsHeight As Long
```

```

        dmDisplayFlags      As Long
        dmDisplayFrequency As Long
End Type

```

```

Type PRINTER_INFO_2
    pServerName      As Long
    pPrinterName     As Long
    pShareName       As Long
    pPortName        As Long
    pDriverName      As Long
    pComment         As Long
    pLocation        As Long
    pDevMode         As Long
    pSepFile         As Long
    pPrintProcessor  As Long
    pDatatype        As Long
    pParameters      As Long
    pSecurityDescriptor As Long
    Attributes       As Long
    Priority          As Long
    DefaultPriority   As Long
    StartTime        As Long
    UntilTime        As Long
    Status            As Long
    cJobs            As Long
    AveragePPM       As Long
End Type

```

```

Type PRINTER_INFO_5
    pPrinterName      As Long
    pPortName         As Long
    Attributes        As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

```

・ 使用可能なプリンタ・プリントサーバーなどを列挙する

```

Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)

```

・ ある位置から別の位置にメモリブロックを移動

```

Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)

```

・ プリンタオブジェクトをオープン

```

Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

```

・ プリンタの詳細な情報を取得

```

Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

```

・ プリンタの詳細な情報を設定

```

Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)

```

・ プリンタオブジェクトを閉じる

```

Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

```

```

Var Shared Text1 As Object
Var Shared text2 As Object
Var Shared Edit1 As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

```

```

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14

```

```

Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterName As String

'=====
'= 接続プリンタを列挙
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    'ローカルプリンタから検索
    PrintServer = ""

    'PRINTER_INFO_5構造体を受け取る
    Level = 5

    'バッファに必要なバイト数を調べる
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
    If Needed = 0 Then End

    '全プリンタ情報を得る
    Var Buffer(Needed - 1) As Byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)

    '構造体リストの準備
    Var PI_5(Returned - 1) As PRINTER_INFO_5

    For CT = 0 To Returned - 1

        'バッファから構造体1つ分を抜き取る
        CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

        'プリンタ名を得る
        PrinterName = String$(MAX_DEVICENAME, Chr$(0))
        CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
        PrinterName = KLeft$(PrinterName, kInStr(1, PrinterName, Chr$(0)) - 1)

        Combo1.AddString PrinterName
    Next
End Sub

'=====
'= 印刷部数を設定
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var PrtDevName As String
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var Level As Long
    Var Needed As Long
    Var pi2 As PRINTER_INFO_2
    Var dm As DEVMODE
    Var Ret As Long

    'プリンタ名を指定
    PrinterName = Combo1.GetText(Combo1.GetCursel)
    If PrinterName = "" Then
        A% = MsgBox(GetWindowText, "プリンタを選択してください!", 0, 2)
        Exit Sub
    End If

```

```

'プリンタアクセス権を指定
pd.DesiredAccess = PRINTER_ALL_ACCESS

'プリンタのオブジェクトハンドルを取得
Ret = Api_OpenPrinter(PrinterName, hPrinter, pd)

'構造体のレベルを指定
Level = 2

'バッファに必要なサイズを取得
Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Needed)

'バッファを確保
Var Buffer(Needed - 1)

'詳細なプリンタ情報を取得
Ret = Api_GetPrinter(hPrinter, Level, Buffer(0), Needed, Needed)

'取得した詳細なプリンタ情報を構造体へ移動
CopyMemory pi2, Buffer(0), Len(pi2)

'取得した詳細なプリンタデバイス情報を構造体へ移動
CopyMemory dm, ByVal pi2.pDevMode, Len(dm)

'部数が指定できるときは
If dm.dmFields And DM_COPIES Then

    '部数を指定
    dm.dmCopies = Val(Edit1.GetWindowText)

    '構造体を取得した詳細なプリンタデバイス情報へ移動
    CopyMemory ByVal pi2.pDevMode, dm, Len(dm)

    '詳細なプリンタ情報を書き戻し
    Ret = Api_SetPrinter(hPrinter, Level, pi2, 0)
End If

'プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'= 文字入力 + Enter
'=====
Declare Sub Edit1_Change edec1 ()
Sub Edit1_Change()
    Var EPos As Integer

    Ed$ = Edit1.GetWindowText

    EPos = InStr(Ed$, Chr$(13, 10))
    If EPos <> 0 Then
        Ed$ = Mid$(Ed$, 1, EPos - 1) & Mid$(Ed$, EPos + 2)
        Edit1.SetWindowText Ed$
        If Val(Ed$) < 1 Or Val(Ed$) > 99 Then
            Edit1.SetWindowText ""
            Edit1.SetFocus
        Else
            Button1.SetFocus
        End If
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop : End

```

プリンタの印刷方向を設定

プリンタの印刷方向を設定します。

OpenPrinter プリンタのオブジェクトをオープン

GetPrinter プリンタの詳細な情報を取得

MoveMemory ある位置から別の位置へメモリブロックを移動

SetPrinter プリンタの詳細な情報を設定

ClosePrinter プリンタのオブジェクトハンドルを解放



実行結果の確認



```
'=====
'= プリンタの印刷方向を設定
'= (DM_ORIENTATION.bas)
'=====
#include "Windows.bi"
```

```
Type PRINTER_DEFAULTS
    pDatatype As Long
    pdm As Long
    DesiredAccess As Long
End Type
```

```
#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す定数
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す定数
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
```

```
#define CCHDEVICENAME 32 'デバイス名の長さを示す定数
#define CCHFORMNAME 32 'フォーム名の長さを示す定数
```

```
Type DEVMODE
    dmDeviceName As String * CCHDEVICENAME 'ドライバがサポートするデバイス名
    dmSpecVersion As Integer '構造体の基準になった初期化データ仕様のバージョン番号
    dmDriverVersion As Integer 'プリンタドライバのバージョン番号
    dmSize As Integer 'この構造体のサイズ(バイト単位)
    dmDriverExtra As Integer 'この構造体に続くドライバ データのバイト数
    dmFields As Long
    dmOrientation As Integer 'DMORIENT_PORTRAIT、DMORIENT_LANDSCAPE
    dmPaperSize As Integer '用紙サイズ
    dmPaperLength As Integer 'dmPaperSizeメンバで指定した用紙の長さをオーバーライド
    dmPaperWidth As Integer 'dmPaperSizeメンバで指定した用紙の幅をオーバーライド
    dmScale As Integer '印刷出力をスケールリングするときの、スケールリング係数
    dmCopies As Integer 'デバイスが複数の部数に対応する場合、印刷する部数
    dmDefaultSource As Integer '予約済み(0)
    dmPrintQuality As Integer 'プリンタの解像度(ドット/インチ)
    dmColor As Integer 'カラープリンタの場合
    ' (DMCOLOR_COLOR・DMCOLOR_MONOCHROME)
```

dmDuplex	As Integer	'両面印刷が可能なプリンタ (DMDUP_SIMPLEX・DMDUP_HORIZONTAL・ DMDUP_VERTICAL)
dmYResolution	As Integer	'プリンタのy方向の解像度(ドット/インチ)
dmTTOption	As Integer	'TrueTypeフォントの印刷方法
dmCollate	As Integer	'複数部数を印刷するときにページ順にそろえるかどうか
dmFormName	As String * CCHFORMNAME	'フォーム名を指定
dmUnusedPadding	As Integer	'使用しない
dmBitsPerPixel	As Integer	'ディスプレイ デバイスの解像度をピクセルあたりのビット数 で指定
dmPelsWidth	As Long	'可視のデバイスの表面の幅をピクセル単位で指定
dmPelsHeight	As Long	'可視のデバイスの表面の高さをピクセル単位で指定
dmDisplayFlags	As Long	'デバイスのディスプレイ モードを指定
dmDisplayFrequency	As Long	'ディスプレイデバイスのリフレッシュレート(垂直同期周波 数)を1秒当たりのサイクル数(Hz)で指定
dmICMMethod	As Long	'非ICMアプリケーションの場合に、ICMが使用可能かどうか を指定
dmICMIntent	As Long	'カラーマッチング方法のデフォルトを指定
dmMediaType	As Long	'印刷メディアのタイプを指定
dmDitherType	As Long	'ディザリング方法を指定
dmReserved1	As Long	'予約済み(0)
dmReserved2	As Long	'予約済み(0)
dmPanningWidth	As Long	'NT系(0)
dmPanningHeight	As Long	'NT系(0)

End Type

#define DM_ORIENTATION &H1 '用紙の向きを指定
#define DMORIENT_PORTRAIT 1 '用紙縦置き
#define DMORIENT_LANDSCAPE 2 '用紙横置き

Type PRINTER_INFO_2

pServerName	As Long
pPrinterName	As Long
pShareName	As Long
pPortName	As Long
pDriverName	As Long
pComment	As Long
pLocation	As Long
pDevMode	As Long
pSepFile	As Long
pPrintProcessor	As Long
pDatatype	As Long
pParameters	As Long
pSecurityDescriptor	As Long
Attributes	As Long
Priority	As Long
DefaultPriority	As Long
StartTime	As Long
UntilTime	As Long
Status	As Long
cJobs	As Long
AveragePPM	As Long

End Type

' プリントオブジェクトをオープン

Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName\$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得

Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' ある位置から別の位置にメモリブロックを移動

Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)

' プリンタの詳細な情報を設定

Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)

' プリンタのオブジェクトハンドルを破棄

```
Declare Function Api_ClosePrinter Lib "winspool.drv" Alias "ClosePrinter" (ByVal  
hPrinter&)
```

```
Var Shared Text1 As Object  
Var Shared Radiol As Object  
Var Shared Radio2 As Object  
Var Shared Button1 As Object
```

```
Text1.Attach getDlgItem("Text1") : Text1.SetFontSize 14  
Radiol.Attach getDlgItem("Radiol") : Radiol.SetFontSize 14  
Radio2.Attach getDlgItem("Radio2") : Radio2.SetFontSize 14  
Button1.Attach getDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
```

```
Declare Function Index bdecl () As Integer  
Function Index()  
    Index = val (Mid$(GetDlgRadioSelect("Radiol"), 6)) -1  
End Function
```

```
'=====
```

```
Declare Sub Button1_on edec1 ()  
Sub Button1_on()
```

```
    Var PrtDevName As String  
    Var pd As PRINTER_DEFAULTS  
    Var hPrinter As Long  
    Var Level As Long  
    Var Need As Long  
    Var pi2 As PRINTER_INFO_2  
    Var dm As DEVMODE  
    Var Ret As Long
```

' プリンタ名を指定

```
PrtDevName = "EPSON PX-G920" 'Printer.DeviceName
```

' プリンタアクセス権を指定

```
pd.DesiredAccess = PRINTER_ALL_ACCESS
```

' プリンタのオブジェクトハンドルを取得

```
Ret = Api_OpenPrinter(PrtDevName, hPrinter, pd)
```

' 構造体のレベルを指定

```
Level = 2
```

' バッファに必要なサイズを取得

```
Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Need)
```

' バッファを確保

```
Var Buffer(Need - 1) As Byte
```

' 詳細なプリンタ情報を取得

```
Ret = Api_GetPrinter(hPrinter, Level, Buffer(0), Need, Need)
```

' 取得した詳細なプリンタ情報を構造体へ移動

```
MoveMemory pi2, Buffer(0), Len(pi2)
```

' 取得した詳細なプリンタデバイス情報を構造体へ移動

```
MoveMemory dm, ByVal pi2.pDevMode, Len(dm)
```

' 印刷方向が指定できる場合

```
If dm.dmFields And DM_ORIENTATION Then
```

' 印刷方向を指定

```
If Index = 0 Then  
    dm.dmOrientation = DMORIENT_PORTRAIT  
Else
```



```

        dm.dmOrientation = DMORIENT_LANDSCAPE
    End If

    ' 構造体を取得した詳細なプリンタデバイス情報へ移動
    MoveMemory ByVal pi2.pDevMode, dm, Len(dm)

    ' 詳細なプリンタ情報を書き戻し
    Ret = Api_SetPrinter(hPrinter, Level, pi2, 0)
    End If

    ' プリンタオブジェクトをクローズ
    Ret = Api_ClosePrinter(hPrinter)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタのオンライン・オフライン切替

通常使うプリンタを「オンライン・オフライン」で使用するかを切り替えます。

OpenPrinter プリンタオブジェクトをオープン
GetPrinter プリンタの詳細な情報を取得
MoveMemory メモリの指定領域をコピー
SetPrinter プリンタの詳細な情報を設定
ClosePrinter プリンタオブジェクトを閉じる



```

' =====
' = プリンタのオンライン・オフライン切替
' = (PrinterOffline.bas)
' =====

#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type PRINTER_INFO_2
    pServerName As Long
    pPrinterName As Long
    pShareName As Long
    pPortName As Long
    pDriverName As Long
    pComment As Long
    pLocation As Long
    pDevMode As Long
    pSepFile As Long
    pPrintProcessor As Long
    pDatatype As Long

```

```

    pParameters          As Long
    pSecurityDescriptor As Long
    Attributes           As Long
    Priority              As Long
    DefaultPriority       As Long
    StartTime            As Long
    UntilTime            As Long
    Status               As Long
    cJobs                As Long
    AveragePPM           As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4    'プリンタアクセス権の管理者権限を示す定数宣言
#define PRINTER_ACCESS_USE &H8           'プリンタアクセス権のユーザー権限の定数宣言
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define PRINTER_ATTRIBUTE_WORK_OFFLINE &H400 'プリンタがオフライン

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal length&)

' プリンタの詳細な情報を設定
Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

Var Shared Comb1 As Object
Var Shared Button1 As Object

Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Comb1.AddString "オフライン"
    Comb1.AddString "オンライン"
End Sub

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var PrtDevName As String
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var Level As Long
    Var Need As Long
    Var pi2 As PRINTER_INFO_2
    Var Ret As Long

    ' プリンタ名を指定
    PrtDevName = "KONICA MINOLTA magicolor 2400W" 'Printer.DeviceName

```

```

'プリンタアクセス権を指定
pd.DesiredAccess = PRINTER_ALL_ACCESS

'プリンタのオブジェクトハンドルを取得
Ret = Api_OpenPrinter(PrtDevName, hPrinter, pd)

'構造体のレベルを指定
Level = 2

'バッファに必要なサイズを取得
Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Need)

'バッファを確保
Var Buffer(Need - 1) As Byte

'詳細なプリンタ情報を取得
Ret = Api_GetPrinter(hPrinter, Level, Buffer(0), Need, Need)

'取得した詳細なプリンタ情報を構造体へ移動
MoveMemory pi2, Buffer(0), Len(pi2)

'オフライン設定が選択されているとき
If Combol.GetWindowText = "オフライン" Then

    'オフライン属性を設定
    pi2.Attributes = PRINTER_ATTRIBUTE_WORK_OFFLINE

'オフライン設定が選択されていないとき
Else

    'オフライン属性を解除
    pi2.Attributes = pi2.Attributes Xor PRINTER_ATTRIBUTE_WORK_OFFLINE
End If

'詳細なプリンタ情報を書き戻し
Ret = Api_SetPrinter(hPrinter, Level, pi2, 0)

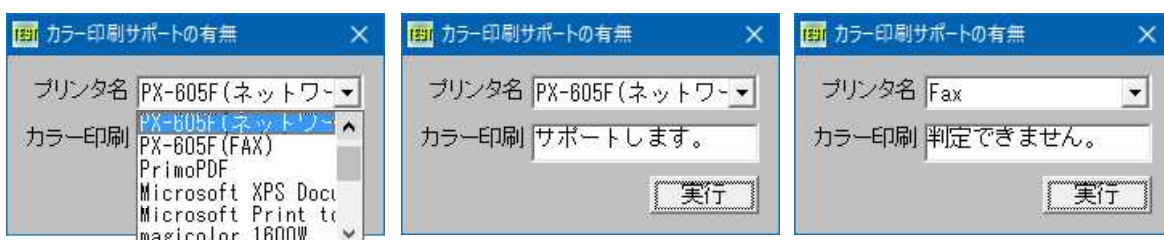
'プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタのカラー印刷サポートを判定

CopyMemory マウスホイールのスクロール量を取得・設定します。ある位置から別の位置にメモリブロックを移動
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
SHInvokePrinterCommand プリンタのプロパティ取得等のコマンドを実行
DeviceCapabilities プリンタデバイスドライバの能力を取得



```

'=====
'= プリンタのカラー印刷サポートを判定
'= (DC_COLORDEVICE.bas)
'=====
#include "Windows.bi"

#define PRINTER_ENUM_DEFAULT &H1
#define PRINTER_ENUM_LOCAL &H2
#define PRINTER_ENUM_NAME &H8
#define PRINTER_ENUM_SHARED &H20
#define MAX_DEVICENAME 64
#define DC_COLORDEVICE 32

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionColorDeviceryTimeOut As Long
End Type

' ある位置から別の位置にメモリブロックを移動する関数の宣言
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)

' 使用可能なプリンタ・プリントサーバーなどを列挙する
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)

' プリンタのプロパティ取得等のコマンドを実行
Declare Function Api_SHInvokePrinterCommand& Lib "Shell32" Alias
"SHInvokePrinterCommandA" (ByVal hWnd&, ByVal uAction&, ByVal lpBuf1$, ByVal lpBuf2$,
ByVal fModal&)

' プリンタデバイスドライバの能力を取得
Declare Function Api_DeviceCapabilities& Lib "winspool.drv" Alias "DeviceCapabilitiesA"
(ByVal lpDeviceName$, ByVal lpPort$, ByVal iIndex&, lpOutput As Any, lpDevMode As Any)

Var Shared Text(2) As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

For i = 0 To 2
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var ColorDevice As Long

    'ローカルプリンタから検索
    PrintServer = ""

    'PRINTER_INFO_5構造体を受け取る
    Level = 5

```

```

'バッファに必要なバイト数を調べる
ColorDevice = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0,
Needed, Returned)
If Needed = 0 Then End

'全プリンタ情報を得る
Var Buffer(Needed - 1) As byte
ColorDevice = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0),
Needed, Needed, Returned)

'構造体リストの準備
Var PI_5(Returned - 1) As PRINTER_INFO_5

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
    PrinterName = KLeft$(PrinterName, KInStr(1, PrinterName, Chr$(0)) - 1)
    Combo1.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var DeviceCapa As Long
    Var txt As String
    Var ColorDevice As Long

    PrinterName = Combo1.GetWindowText
    If PrinterName = "" Then
        A% = MsgBox(GetWindowText, "プリンタ名を指定してください!", 0, 2)
        Exit Sub
    End If

    '問い合わせる能力を指定
    DeviceCapa = DC_COLORDEVICE

    'カラー印刷サポートを判定
    ColorDevice = Api_DeviceCapabilities(PrinterName, "", DeviceCapa, ByVal 0, ByVal 0)

    'カラー印刷サポートを表示
    Select Case ColorDevice
        Case 0
            txt = "サポートしません。"
        Case 1
            txt = "サポートします。"
        Case Else
            txt = "判定できません。"
    End Select

    Text(2).SetWindowText txt
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの利用できる給紙方法・最大部数取得

選択したプリンタの利用できる給紙方法を取得します。

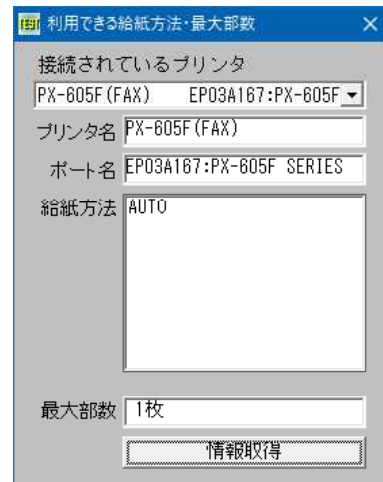
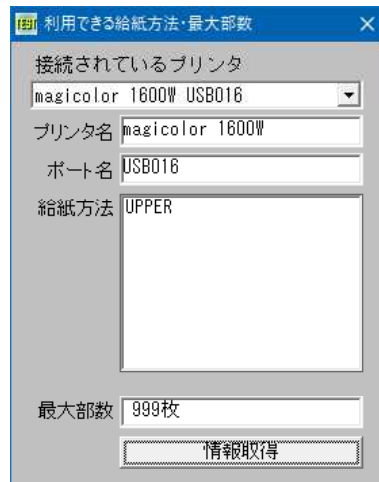
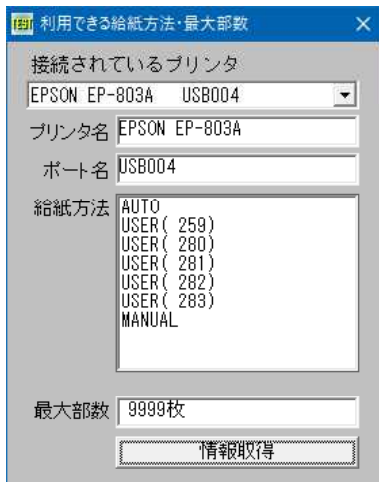
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙

RtlMoveMemory メモリブロックを別の領域に移動

DeviceCapabilities プリンタデバイスドライバの能力を取得

接続されているプリンタ名およびポート名がコンボボックスにリスト表示されます。

リスト行を選択し取得ボタンをクリックすることにより使用できる用紙の種類が表示されます。



```
'=====
'= プリンタの利用できる給紙方法・最大部数取得
'= (Devicecapabilities2.bas)
'=====
#include "Windows.bi"
```

```
Type PRINTER_INFO_5
    pPrinterName      As Long
    pPortName         As Long
    Attributes        As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type
```

' 使用可能なプリンタ・プリントサーバーなどを列挙する

```
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags& , ByVal Name$ , ByVal Level& , pPrinterEnum As Any , ByVal cbBuf& , pcbNeeded& ,
pcReturned& )
```

```
#define PRINTER_ENUM_NAME &H8
#define MAX_DEVICENAME 64
```

' メモリブロックを別の領域に移動する

```
Declare Sub Api_MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any,
Source As Any, ByVal Length&)
```

' プリンタデバイスドライバの能力を取得する関数の宣言

```
Declare Function Api_DeviceCapabilities& Lib "winspool.drv" Alias "DeviceCapabilitiesA"
(ByVal lpDeviceName$, ByVal lpPort$, ByVal iIndex&, lpOutput As Any, lpDevMode As Any)
```

```
#define DC_ACTIVE 1
#define DC_BINS 6
```

' アクティブなときの色

' プリンタで利用できる給紙方法を取得することを示す定数の宣言

```

#define DC_COPIES 18
#define DC_GRADIENT &H20
#define DC_ICON 4
#define DC_INBUTTON &H10
#define DC_NOTACTIVE &H2
#define DC_SMALLCAP 2
#define DC_TEXT 8

' 最大部数を取得することを示す定数の宣言
' グラデーション
' アイコン付き
' ボタンとして描画
'
' 幅の狭いタイプ
' タイトルテキスト付き

' 給紙方法を示す定数の宣言
#define DMBIN_CASSETTE 14
#define DMBIN_ENVELOPE 5
#define DMBIN_ENVMANUAL 6
#define DMBIN_FORMSOURCE 15
#define DMBIN_LARGECAPACITY 11
#define DMBIN_LARGEFORMAT 10
#define DMBIN_LOWER 2
#define DMBIN_MANUAL 4
#define DMBIN_MIDDLE 3
#define DMBIN_ONLYONE 1
#define DMBIN_SMALLFORMAT 9
#define DMBIN_TRACTOR 8
#define DMBIN_UPPER 1
#define DMBIN_USER 256

' 用紙カセット
' 封筒フィーダ
' 手差し封筒フィーダ
'
' 大型用紙ソース
' 下用紙トレイ
' 手差し用紙フィーダ
' 中用紙トレイ
' 単一用紙ソース
' 小型用紙ソース
' トラクタフィーダ
' 上用紙フィーダ
' ユーザ定義

Var Shared List1 As Object
Var Shared Combo1 As Object
Var Shared Text(7) As Object

Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
For i = 0 To 7
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next i

Var Shared PrinterName As String
Var Shared PortName As String

' =====
' =
' =====
Declare Sub Mainform_Start edec1 ()
Sub Mainform_Start()
    Var PrintServer As String
    Var Need As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    ' ローカルプリンタから検索
    PrintServer = ""

    ' PRINTER_INFO_5 構造体を受け取る
    Level = 5

    ' バッファに必要なバイト数を調べる
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Need,
Returned)
    If Need = 0 Then End

    ' 全プリンタ情報を得る
    Var Buffer(Need - 1) As byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Need, Need,
Returned)

    ' 構造体リストの準備
    Var PI_5(Returned-1) As PRINTER_INFO_5

    For CT = 0 To Returned - 1

```

```

'バッファから構造体1つ分を抜き取る
Api_MoveMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

'プリンタ名を得る
PrinterName = String$(MAX_DEVICENAME, Chr$(0))
Api_MoveMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
PrinterName = kLeft$(PrinterName, KInStr(1, PrinterName, Chr$(0)) - 1)

'ポート名を得る
PortName = String$(MAX_DEVICENAME, Chr$(0))
Api_MoveMemory PortName, ByVal PI_5(CT).pPortName, Len(PortName)
PortName = kLeft$(PortName, KInStr(1, PortName, Chr$(0)) - 1)
Combo1.AddString Left$(PrinterName & space$(16), 16) & PortName
Next
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    PrinterName = Trim$(Left$(Combo1.GetText(Combo1.GetCursel), 16))
    PortName = Trim$(Mid$(Combo1.GetText(Combo1.GetCursel), 17))
    Text(5).SetWindowText PrinterName
    Text(6).SetWindowText PortName
    List1.ResetContent
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    on error goto *ER_Trap

    Var DeviceCapability As Long
    Var DeviceCopies As Long
    Var Need As Long
    Var CT As Long
    Var Ret As Long

'リストボックスを初期化
List1.ResetContent

'問い合わせる能力を指定
DeviceCapability = DC_BINS

'バッファに必要なサイズを取得
Need = Api_DeviceCapabilities(PrinterName, PortName, DeviceCapability, ByVal 0,
ByVal 0)

'バッファを確保
Var Bins(Need - 1) As Integer

'使用できる給紙方法を取得
Ret = Api_DeviceCapabilities(PrinterName, PortName, DeviceCapability, Bins(0), ByVal
0)

'使用できる給紙方法を列挙
For CT = 0 To Need - 1

    '給紙方法を表示
    Select Case Bins(CT)
        Case DMBIN_UPPER
            List1.AddString "UPPER"
        Case DMBIN_ONLYONE
            List1.AddString "ONLYONE"
        Case DMBIN_LOWER
            List1.AddString "LOWER"
        Case DMBIN_MIDDLE

```



```

        List1.AddString "MIDDLE"
    Case DMBIN_MANUAL
        List1.AddString "MANUAL"
    Case DMBIN_ENVELOPE
        List1.AddString "ENVELOPE"
    Case DMBIN_ENVMANUAL
        List1.AddString "ENVMANUAL"
    Case DMBIN_AUTO
        List1.AddString "AUTO"
    Case DMBIN_TRACTOR
        List1.AddString "TRACTOR"
    Case DMBIN_SMALLFMT
        List1.AddString "SMALLFMT"
    Case DMBIN_LARGEFORMAT
        List1.AddString "LARGEFORMAT"
    Case DMBIN_LARGEFORMATCAPACITY
        List1.AddString "LARGEFORMATCAPACITY"
    Case DMBIN_CASSETTE
        List1.AddString "CASSETTE"
    Case DMBIN_FORMSOURCE
        List1.AddString "FORMSOURCE"
    Case Else

        'ユーザー定義のときは
        If Bins(CT) >= DMBIN_USER Then
            List1.AddString "USER" & "(" & Str$(Bins(CT)) & ")"

        '未定義のときは
        Else
            List1.AddString "UNKNOWN" & "(" & Str$(Bins(CT)) & ")"
        End If
    End Select
Next CT

'問い合わせる能力を指定
DeviceCapability = DC_COPIES

'最大部数を取得
Ret = Api_DeviceCapabilities(PrinterName, PortName, DeviceCapability, ByVal 0, ByVal
0)
Text(7).SetWindowText Str$(Ret) & "枚"
Exit Sub

*ER_Trap
Resume Next
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの共有名を取得

プリンタの共有名を取得します。

- OpenPrinter** プリンタオブジェクトをオープン
- GetPrinter** プリンタの詳細な情報を取得
- MoveMemory** メモリの指定領域をコピー
- lstrcpy** 文字列をコピーする
- ClosePrinter** プリンタオブジェクトを閉じる



```
'=====
'= プリンタの共有名を取得
'= (GetPrinterShareName.bas)
'=====

#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDataType As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type PRINTER_INFO_2
    pServerName As Long
    pPrinterName As Long
    pShareName As Long
    pPortName As Long
    pDriverName As Long
    pComment As Long
    pLocation As Long
    pDevMode As Long
    pSepFile As Long
    pPrintProcessor As Long
    pDataType As Long
    pParameters As Long
    pSecurityDescriptor As Long
    Attributes As Long
    Priority As Long
    DefaultPriority As Long
    StartTime As Long
    UntilTime As Long
    Status As Long
    cJobs As Long
    AveragePPM As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数宣言
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す定数宣言
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す定数宣言
#define PRINTER_ALL_ACCESS (STANDARD_RIGHTS_REQUIRED Or PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE)

' プリントオブジェクトをオープン
Declare Function Api_OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得する関数の宣言
Declare Function Api_GetPrinter Lib "winspool.drv" Alias "GetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (dest As Any, Source As Any, ByVal Length&)

' 文字列をコピーする
Declare Function Api_lstrcpy Lib "Kernel32" Alias "lstrcpyA" (lpszString1 As Any, lpszString2 As Any)

' プリントオブジェクトを閉じる
Declare Function Api_ClosePrinter Lib "winspool.drv" Alias "ClosePrinter" (ByVal hPrinter&)
```

```

Var Shared Text(3) As Object
Var Shared Button1 As Object

For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var DeviceName As String
    Var pd As PRINTER_DEFAULTS
    Var hWnd As Long
    Var Level As Long
    Var Need As Long
    Var pi2 As PRINTER_INFO_2
    Var ShareName As String * 128
    Var Ret As Long

    'プリンタ名を指定
    DeviceName = "magicolor 2400W"

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(DeviceName, hWnd, pd)

    '構造体のレベルを指定
    Level = 2

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinter(hWnd, Level, ByVal 0, 0, Need)

    'バッファを確保
    Var pi2Buffer(Need - 1) As Byte

    '詳細なプリンタ情報を取得
    Ret = Api_GetPrinter(hWnd, Level, pi2Buffer(0), Need, Need)

    '取得した詳細なプリンタ情報を構造体へ移動
    MoveMemory pi2, pi2Buffer(0), Len(pi2)

    '共有名を複写
    Ret = Api_lstrcpy(ShareName, ByVal pi2.pShareName)

    '共有名取得
    Text(3).SetWindowText Left$(ShareName, InStr(ShareName, Chr$(0)) - 1)

    'プリンタオブジェクトをクローズ
    Ret = Api_ClosePrinter(hWnd)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの共有を設定・解除

指定したプリンタの共有を設定します。

OpenPrinter プリンタオブジェクトをオープン

GetPrinter プリンタの詳細な情報を取得

MoveMemory メモリの指定領域をコピー

SetPrinter プリンタの詳細な情報を設定

ClosePrinter プリンタオブジェクトを閉じる

例では、「指定しない」で実行をクリックした場合、「共有←→共有しない」を繰り返します。



```
'=====
'= プリンタの共有を設定
'= (PrinterShared.bas)
'=====
#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define PRINTER_ATTRIBUTE_SHARED &H8 '共有

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any, ByVal length&)

' プリンタの詳細な情報を設定
Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal hPrinter&)

Var Shared Text1 As Object
Var Shared Edit1 As Object
```

```

Var Shared Combo1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Combo1.AddString "指定しない"
    Combo1.AddString "指定する"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var PrinterDeviceName As String
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var Level As Long
    Var Needed As Long
    Var pi5 As PRINTER_INFO_5
    Var Ret As Long

    'プリンタ名を指定
    PrinterDeviceName = Edit1.GetWindowText

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(PrinterDeviceName, hPrinter, pd)

    '構造体のレベルを指定
    Level = 5

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Needed)

    'バッファを確保
    Var Buffer(Needed - 1)

    '詳細なプリンタ情報を取得
    Ret = Api_GetPrinter(hPrinter, Level, Buffer(0), Needed, Needed)

    '取得した詳細なプリンタ情報を構造体へ移動
    MoveMemory pi5, Buffer(0), Len(pi5)

    '共有を指定しない
    If Combo1.GetCursel = 0 Then

        '属性との排他的論理和を指定
        pi5.Attributes = pi5.Attributes Xor PRINTER_ATTRIBUTE_SHARED

    '共有を指定する
    Else

        '属性との論理和を指定
        pi5.Attributes = pi5.Attributes Or PRINTER_ATTRIBUTE_SHARED
    End If

    '詳細なプリンタ情報を設定
    Ret = Api_SetPrinter(hPrinter, Level, pi5, 0)

```

```

' プリントオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの切替「通常使うプリンタ」とプロパティ

接続されているプリンタを列挙し、選択したプリンタのプロパティダイアログを表示します。

EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙

RtlMoveMemory メモリブロックを別の領域に移動

OpenPrinter プリントオブジェクトをオープン

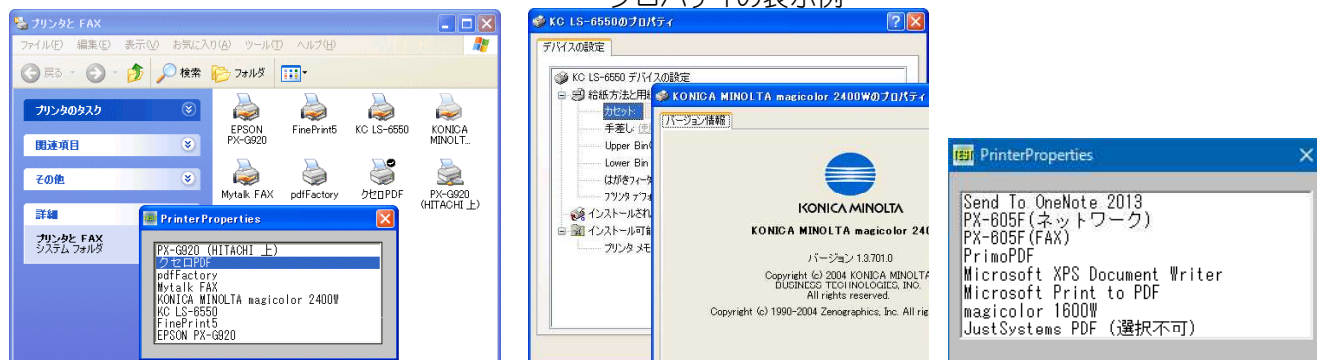
ClosePrinter プリントオブジェクトを閉じる

PrinterProperties プリンタ設定ダイアログ表示

SetDefaultPrinter 通常使うプリンタの設定

接続しているプリンタを列挙し、選択したプリンタを「通常使用するプリンタ」に設定するとともに、そのプリンタプロパティを表示します。

プロパティの表示例



```

' =====
' = プリンタの切替「通常使うプリンタ」とプロパティ
' = 通常使うプリンタの設定
' = (PrinterProperties.bas)
' =====
#include "Windows.bi"

```

```

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

```

' 使用可能なプリンタ・プリントサーバーなどを列挙する

```

Declare Function Api_EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags As Long, ByVal Name As String, ByVal Level As Long, pPrinterEnum As Any,
ByVal cbBuf As Long, pcbNeeded As Long, pcReturned As Long) As Long

```

' ある位置から別の位置にメモリブロックを移動

```

Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any,
Source As Any, ByVal Length As Long)

```

' プリントオブジェクトをオープン

```

Declare Function Api_OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName As String, phPrinter As Long, pDefault As Any) As Long

```

' プリントオブジェクトを閉じる

```
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal  
hPrinter&)
```

' プリント設定ダイアログ表示

```
Declare Function Api_PrinterProperties& Lib "winspool.drv" Alias "PrinterProperties"  
(ByVal hWnd&, ByVal hPrinter&)
```

' 通常使うプリンタの設定

```
Declare Function Api_SetDefaultPrinter& Lib "winspool.drv" Alias "SetDefaultPrinterA"  
(ByVal pszPrinter$)
```

```
#define PRINTER_ENUM_DEFAULT &H1  
#define PRINTER_ENUM_LOCAL &H2  
#define PRINTER_ENUM_NAME &H8  
#define PRINTER_ENUM_SHARED &H20  
#define MAX_DEVICENAME 64
```

' デフォルトのプリンタに関する情報を列挙

' Nameの設定を無視して、ローカルプリンタを列挙

' Nameで指定されたプリンタを列挙

' 共有属性を持つプリンタを列挙

```
Var Shared List1 As Object  
List1.Attach GetDlgItem("List1")  
List1.SetFontSize 14  
Var Shared PrinterName As String
```

```
' =====  
' =  
' =====
```

```
Declare Sub MainForm_Start edec1 ()  
Sub MainForm_Start()
```

```
    Var PrintServer As String  
    Var Needed As Long  
    Var Returned As Long  
    Var Level As Long  
    Var CT As Long  
    Var Ret As Long
```

' ローカルプリンタから検索

```
PrintServer = ""
```

' PRINTER_INFO_5 構造体を受け取る

```
Level = 5
```

' バッファに必要なバイト数を調べる

```
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,  
Returned)  
If Needed = 0 Then End
```

' 全プリンタ情報を得る

```
Var Buffer(Needed-1) As Byte  
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,  
Needed, Returned)
```

' 構造体リストの準備

```
Var PI_5(Returned - 1) As PRINTER_INFO_5
```

```
For CT = 0 To Returned - 1
```

' バッファから構造体1つ分を抜き取る

```
CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))
```

' プリンタ名を得る

```
PrinterName = String$(MAX_DEVICENAME, Chr$(0))  
CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)  
PrinterName = KLeft$(PrinterName, kInStr(1, PrinterName, Chr$(0)) - 1)
```

```
List1.AddString PrinterName
```

```
Next
```

```
End Sub
```

```

'=====
'=
'=====
Declare Sub List1_Change edec1 ()
Sub List1_Change ()
    Var hPrinter As Long
    Var Ret As Long

    PrinterName = List1.GetText(List1.GetCursel)

    Ret = Api_SetDefaultPrinter(PrinterName)
    Ret = Api_OpenPrinter(PrinterName, hPrinter, ByVal 0)
    Ret = Api_PrinterProperties(GethWnd, hPrinter)
    Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop

```

プリンタのコメントを取得

プリンタのコメントを取得します。

OpenPrinter プリンタオブジェクトをオープン

GetPrinter プリンタの詳細な情報を取得

MoveMemory メモリの指定領域をコピー

lstrcpy 文字列をコピーする

ClosePrinter プリンタオブジェクトを閉じる



```

'=====
'= プリンタのコメントを取得
'= (GetPrinterComment.bas)
'=====
#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type PRINTER_INFO_2
    pServerName As Long
    pPrinterName As Long
    pShareName As Long
    pPortName As Long
    pDriverName As Long
    pComment As Long
    pLocation As Long
    pDevMode As Long
    pSepFile As Long
    pPrintProcessor As Long

```



```

        pDatatype           As Long
        pParameters         As Long
        pSecurityDescriptor As Long
        Attributes          As Long
        Priority             As Long
        DefaultPriority      As Long
        StartTime           As Long
        UntilTime           As Long
        Status               As Long
        cJobs                As Long
        AveragePPM          As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4    'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8           'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (STANDARD_RIGHTS_REQUIRED Or PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE) '

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any, ByVal length&)

' 文字列をコピーする
Declare Function Api_lstrcpy& Lib "Kernel32" Alias "lstrcpyA" (lpszString1 As Any, lpszString2 As Any)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal hPrinter&)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Text1.SetWindowText "magicolor 2400W"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Devname As String
    Var pd As PRINTER_DEFAULTS
    Var pi2 As PRINTER_INFO_2
    Var PrtHandle As Long
    Var Level As Long
    Var Needed As Long
    Var Comment As String * 128
    Var Ret As Long

```

```

'プリンタ名を指定
Devname = Text1.GetWindowText

'プリンタアクセス権を指定
pd.DesiredAccess = PRINTER_ALL_ACCESS

'プリンタのオブジェクトハンドルを取得
Ret = Api_OpenPrinter(Devname, PrtHandle, pd)

'構造体のレベルを指定
Level = 2

'バッファに必要なサイズを取得
Ret = Api_GetPrinter(PrtHandle, Level, ByVal 0, 0, Needed)

'バッファを確保
Var Buffer(Needed - 1) As Byte

'プリンタ情報を取得
Ret = Api_GetPrinter(PrtHandle, Level, Buffer(0), Needed, Needed)

'プリンタ情報を構造体へ
MoveMemory pi2, Buffer(0), Len(pi2)

'コメントを表示
Ret = Api_lstrcpy(Comment, ByVal pi2.pComment)

'Null削除
Text2.SetWindowText Left$(Comment, InStr(Comment, Chr$(0)) - 1)

'プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(PrtHandle)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの固有情報を取得

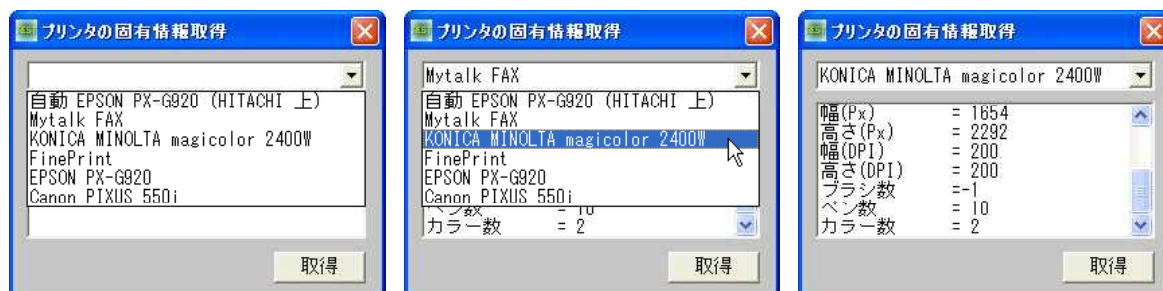
接続されているプリンタを列挙し、選択したプリンタの固有情報を取得表示します。

EnumPrinters 使用可能なプリンタ等を列挙

CopyMemory メモリブロックの移動

GetDeviceCaps デバイス固有の情報を取得

CreateDC デバイスコンテキストを指定された名前で作成



```

'=====
'= プリンタの固有情報を取得
'= (GetDeviceCaps3.bas)
'=====

```

```
#include "Windows.bi"
```

```
Type PRINTER_INFO_5
    pPrinterName           As Long
    pPortName              As Long
    Attributes              As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type
```

・ 使用可能なプリンタ・プリントサーバーなどを列挙する

```
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)
```

・ メモリブロックを別の領域に移動する

```
Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)
```

・ デバイス固有の情報を取得

```
Declare Function Api_GetDeviceCaps& Lib "gdi32" Alias "GetDeviceCaps" (ByVal hDC&, ByVal
nIndex&)
```

・ 指定されたデバイスのデバイスコンテキストを、指定された名前で作成

```
Declare Function Api_CreateDC& Lib "gdi32" Alias "CreateDCA" (ByVal lpDriverName$, ByVal
lpDevName$, ByVal lpOutput$, ByVal lpInitData&)
```

```
#define PRINTER_ENUM_NAME &H8           'Nameで指定されたプリンタを列挙
#define MAX_DEVICENAME 64

#define DT_BOTTOM &H8                    '長方形の下辺にテキストを揃える。DT_SINGLELINEと同時に指定
#define DT_CALCRECT &H400                'テキストを表示するために必要な長方形の大きさをlpRectパラメータに格納
#define DT_CENTER &H1                    'テキストを水平方向に中央揃えで表示します。
#define DT_CHARSTREAM 4                  'CharacterStream
#define DT_DISPFILE 6                    'DisplayFile
#define DT_ListCONTROL &H2000            '複数行エディットコントロールと同じ特性で描画
#define DT_EXPANDTABS &H40               '[TAB]をスペースに変換
#define DT_EXTERNALLEADING &H200         '行の高さに、行間として適当な高さを加える
#define DT_LEFT &H0                      'テキストを左揃え
#define DT_METAFILE 5                   'Metafile
#define DT_MODIFYSTRING &H10000          'DT_END_ELLIPSISフラグ、DT_PATH_ELLIPSISフラグによって変更された文字列をlpStringに格納
#define DT_NOCLIP &H100                  'クリッピングを行わない
#define DT_NOPREFIX &H800                 'プリフィックス文字(「&」を次の文字のアンダースコアに置き換える、「&&」を「&」として処理する)の処理を
#define DT_PATH_ELLIPSIS &H4000          '指定した長方形領域にテキストが収まらない場合、テキストの途中を(...)に置き換える
#define DT_PLOTTER 0                     'ベクタプロッタ
#define DT_RASCAMERA 3                   'ラスタカメラ
#define DT_RASDISPLAY 1                  'ラスタディスプレイ
#define DT_RASPRINTER 2                  'ラスタプリンタ
#define DT_RIGHT &H2                     'テキストを右揃え
#define DT_RTLREADING &H20000            '右から左に向かってテキストを表示。ヘブライ語、アラビア語などを表示する場合に利用する
#define DT_SINGLELINE &H20               'テキストを改行せず、一行で表示
#define DT_TABSTOP &H80                  'タブ文字の空白文字数を設定。このフラグを指定した場合はDRAWTEXTPARAMS構造体のiTabLengthメンバで空白文
#define DT_TOP &H0                       '上揃え。DT_SINGLELINEと同時に指定する必要がある
#define DT_VCENTER &H4                   'テキストを垂直方向の中央揃え。DT_SINGLELINEと同時に指定する必要がある
#define DT_WORDBREAK &H10                'テキストを複数行で表示。折り返しは自動的に行われる

#define DRIVERVERSION 0                  'デバイスドライバのバージョン番号
#define TECHNOLOGY 2                     'デバイス技術を指定する次の定数
#define HORZSIZE 4                       '物理画面の幅(ミリメートル単位)
#define VERTSIZE 6                       '物理画面の高さ(ミリメートル単位)
#define HORZRES 8                        '画面の幅(ピクセル単位)
#define VERTRES 10                       '画面の高さ(ピクセル単位)
```

```

#define NUMBRUSHES 16
#define Numpens 18
#define NUMCOLORS 24
#define LOGPIXELSX 88
#define LOGPIXELSY 90

'デバイス固有のブラシ数
'デバイス固有のペンの数
'デバイスのカラーテーブル内のエントリ数
'ディスプレイの幅のピクセル数 (Pixel/inch)
'ディスプレイの高さのピクセル数 (Pixel/inch)

Var Shared PrinterName As String

Var Shared List1 As Object
Var Shared Comb1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Need As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    'ローカルプリンタから検索(サーバー名 ¥¥xxx 指定可)
    PrintServer = ""

    'PRINTER_INFO_5構造体を受け取る
    Level = 5

    'バッファに必要なバイト数を調べる
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Need,
Returned)
    If Need = 0 Then End

    '全プリンタ情報を得る
    Var Buffer(Need-1) As byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Need, Need,
Returned)

    '構造体リストの準備
    Var PI_5(Returned-1) As PRINTER_INFO_5

    For CT = 0 To Returned - 1

        'バッファから構造体1つ分を抜き取る
        CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

        'プリンタ名を得る
        PrinterName = String$(MAX_DEVICENAME, Chr$(0))
        CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
        PrinterName = kLeft$(PrinterName$, KInStr(1, PrinterName, Chr$(0)) - 1)

        Comb1.AddString PrinterName
    Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var phDC As Long
    Var Ret As Long

    'プリンタ名から、そのデバイスコンテキストを作成
    phDC = Api_CreateDc("WinSpool", PrinterName, ByVal 0, 0)

```

```
List1.Resetcontent
List1.AddString PrinterName
List1.AddString "-----"
```

'デバイス固有の情報を取得

```
Ret = Api_GetDeviceCaps (phDc, TECHNOLOGY)
Select Case Ret
    Case DT_PLOTTER
        List1.AddString "デバイス技術 = プロッタ"
    Case DT_RASDISPLAY
        List1.AddString "デバイス技術 = ラスタディスプレイ"
    Case DT_RASPRINTER
        List1.AddString "デバイス技術 = ラスタプリンタ"
    Case DT_RASCAMERA
        List1.AddString "デバイス技術 = ラスタカメラ"
    Case DT_CHARSTREAM
        List1.AddString "デバイス技術 = 文字ストリーム"
    Case DT_METAFILE
        List1.AddString "デバイス技術 = メタファイル"
    Case DT_DISPFIL
        List1.AddString "デバイス技術 = ディスプレイファイル"
End Select

List1.AddString "ドライババージョン =" & Str$ (Api_GetDeviceCaps (phDc, DRIVERVERSION))
List1.AddString "幅 (mm) =" & Str$ (Api_GetDeviceCaps (phDc, HORZSIZE))
List1.AddString "高さ (mm) =" & Str$ (Api_GetDeviceCaps (phDc, VERTSIZE))
List1.AddString "幅 (Px) =" & Str$ (Api_GetDeviceCaps (phDc, HORZRES))
List1.AddString "高さ (Px) =" & Str$ (Api_GetDeviceCaps (phDc, VERTRES))
List1.AddString "幅 (DPI) =" & Str$ (Api_GetDeviceCaps (phDc, LOGPIXELSX))
List1.AddString "高さ (DPI) =" & Str$ (Api_GetDeviceCaps (phDc, LOGPIXELSY))
List1.AddString "ブラシ数 =" & Str$ (Api_GetDeviceCaps (phDc, NUMBRUSHES))
List1.AddString "ペン数 =" & Str$ (Api_GetDeviceCaps (phDc, NUMPENS))
List1.AddString "カラー数 =" & Str$ (Api_GetDeviceCaps (phDc, NUMCOLORS))
End Sub
```

```
'=====
'=
'=====
Declare Sub Comb1_Change edec1 ()
Sub Comb1_Change ()
    PrinterName = Comb1.GetWindowText
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

プリンタの詳細情報を取得

OpenPrinter プリンタオブジェクトをオープン
GetPrinter プリンタの詳細な情報を取得
ClosePrinter プリンタオブジェクトを閉じる
MoveMemory メモリの指定領域をコピー

指定されたプリンタのプロパティで設定されている情報を取得しています。
 ここでのプリンタ名 (PrtDevName) は、直接指定しています。接続されているプリンタ名の取得は、「[プリンタの固有情報を取得](#)」を参照してください。



```
'=====
'= プリンタの詳細情報を取得
'= (GetPrinter.bas)
'=====

#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDatatype           As Long
    pDevMode             As Long
    DesiredAccess        As Long
End Type

Type PRINTER_INFO_5
    pPrinterName         As Long
    pPortName            As Long
    Attributes           As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す定数宣言
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザ権限を示す定数宣言
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '

#define PRINTER_ATTRIBUTE_DIRECT &H2 '直接データを送る
#define PRINTER_ATTRIBUTE_ENABLE_BIDI &H800 '双方向通信
#define PRINTER_ATTRIBUTE_ENABLE_DEVQ &H80 '一致しないドキュメントの保留
#define PRINTER_ATTRIBUTE_KEEPPRINTEDJOBS &H100 '印刷後ドキュメントを残す
#define PRINTER_ATTRIBUTE_SHARED &H8 '共有

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter Lib "winspool.drv" Alias "GetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter Lib "winspool.drv" Alias "ClosePrinter" (ByVal hPrinter&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any, ByVal length&)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object
Var Shared Comb1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14

Var Shared PrtDevName As String
```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()

    'プリンタ名を指定
    PrtDevName = "KONICA MINOLTA magicolor 2400W" 'Printer.DeviceName
    Text1.SetWindowText PrtDevName

    Combo1.AddString "PRINTER_ATTRIBUTE_DIRECT"
    Combo1.AddString "PRINTER_ATTRIBUTE_ENABLE_BIDI"
    Combo1.AddString "PRINTER_ATTRIBUTE_ENABLE_DEVQ"
    Combo1.AddString "PRINTER_ATTRIBUTE_KEEPPRINTEDJOBS"
    Combo1.AddString "PRINTER_ATTRIBUTE_SHARED"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var Level5 As Long
    Var Need As Long
    Var pi5 As PRINTER_INFO_5
    Var Flg As Integer
    Var Ret As Long

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter (PrtDevName, hPrinter, pd)

    '構造体のレベルを指定
    Level5 = 5

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinter (hPrinter, Level5, ByVal 0, 0, Need)

    'バッファを確保
    Var Buffer (Need - 1) As Byte

    '詳細なプリンタ情報を取得
    Ret = Api_GetPrinter (hPrinter, Level5, Buffer(0), Need, Need)

    '取得した詳細なプリンタ情報を構造体へ移動
    MoveMemory pi5, Buffer(0), Len(pi5)

    Flg = Combo1.GetCursel

    Select Case Flg

        '直接データを送る
        Case 0
            If pi5.Attributes And PRINTER_ATTRIBUTE_DIRECT Then
                Text2.SetWindowText "「直接データを送る」が指定されています。"
            Else
                Text2.SetWindowText "「直接データを送る」が指定されていません。"
            End If

        '双方向通信
        Case 1
            If pi5.Attributes And PRINTER_ATTRIBUTE_ENABLE_BIDI Then
                Text2.SetWindowText "「双方向通信」が指定されています。"
            Else
                Text2.SetWindowText "「双方向通信」が指定されていません。"
            End If
    End Select
End Sub

```


'一致しないドキュメントの保留

Case 2

```
If pi5.Attributes And PRINTER_ATTRIBUTE_ENABLE_DEVQ Then
    Text2.SetWindowText "「ドキュメントの保留」が指定されています。"
Else
    Text2.SetWindowText "「ドキュメントの保留」が指定されていません。"
End If
```

'印刷後ドキュメントを残す

Case 3

```
If pi5.Attributes And PRINTER_ATTRIBUTE_KEEPPRINTEDJOBS Then
    Text2.SetWindowText "「印刷後ドキュメントを残す」が指定されています。"
Else
    Text2.SetWindowText "「印刷後ドキュメントを残す」が指定されていません。"
End If
```

'共有

Case 4

```
If pi5.Attributes And PRINTER_ATTRIBUTE_SHARED Then
    Text2.SetWindowText "「共有」が指定されています。"
Else
    Text2.SetWindowText "「共有」が指定されていません。"
End If
```

Case Else

```
Text2.SetWindowText "ComboBoxの項目を選択してください"
```

End Select

'プリンタオブジェクトをクローズ

```
Ret = Api_ClosePrinter(hPrinter)
```

End Sub

```
'=====
'=
'=====
```

While 1

```
    WaitEvent
```

Wend

Stop

End

プリンタの状態を設定

指定したプリンタの状態(一時停止・削除・再開)を設定します。

OpenPrinter プリンタのオブジェクトハンドルを取得

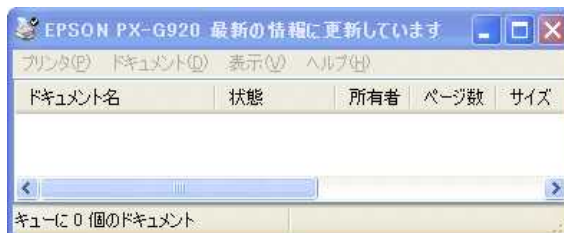
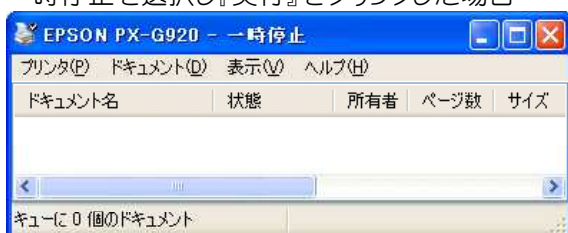
SetPrinter プリンタの詳細な情報を設定

ClosePrinter プリンタのオブジェクトハンドルを破棄

FindWindow クラス名とウィンドウ名が指定された文字列と一致するトップレベル ウィンドウのハンドルを取得



一時停止を選択し『実行』をクリックした場合



削除・再開を選択した場合、『最新の情報に更新しています』一瞬表示され、一時停止のチェックが外れる


```

'=====
'= プリンタの状態を設定
'= (SetPrinter.bas)
'=====
#include "Windows.bi"

' プリンタアクセス権を定義する構造体
Type PRINTER_DEFAULTS
    pDataType As Long
    pDevMod As Long
    DesiredAccess As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS &HF000C 'プリンタアクセス権すべての権限を示す (&HF0000
                                     Or &H4 Or &H8)
#define PRINTER_CONTROL_PAUSE 1 '一時停止を示す
#define PRINTER_CONTROL_RESUME 2 'すべての印刷ジョブ削除を示す
#define PRINTER_CONTROL_PURGE 3 '再開を示す

' プリンタのオブジェクトハンドルを取得
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を設定
Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

' 指定された文字列と一致するクラス名とウィンドウ名を持つトoplevelウィンドウ (親を持たないウィンドウ) のハン
ドルを返す
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,
ByVal lpWindowName$)

Var Shared CMD As String
Var Shared TargetName As String

Var Shared Text(4) As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

For i = 0 To 4
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    TargetName = "EPSON PX-G920"
    Text(3).SetWindowText TargetName
    Comb1.AddString "一時停止 PRINTER_CONTROL_PAUSE"
    Comb1.AddString "削除 PRINTER_CONTROL_RESUME"
    Comb1.AddString "再開 PRINTER_CONTROL_PURGE"
End Sub

'=====
'=
'=====
Declare Sub Comb1_Change edec1 ()
Sub Comb1_Change ()

```

```

        Combo1.SetWindowText Left$(Combo1.GetText(Combo1.GetCursel), 8)
        CMD = Mid$(Combo1.GetWindowText, 10)
End Sub

'=====
' = 通常使うプリンタの状態を設定
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var TargetHwnd As Long
    Var Defaults As PRINTER_DEFAULTS
    Var hWnd As Long
    Var InfoLevel As Long
    Var CtrlCmd As Long
    Var Ret As Long

    TargetHwnd = Api_FindWindow(ByVal 0, TargetName)

    'プリンタアクセス権を指定
    Defaults.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(TargetName, hWnd, Defaults)

    '構造体のレベルを指定
    InfoLevel = 0

    '新しいプリンタの状態を指定
    CtrlCmd = Combo1.GetCursel + 1

    Text(4).SetWindowText Mid$(Combo1.GetText(Combo1.GetCursel), 10)

    '新しいプリンタの状態を設定
    Ret = Api_SetPrinter(hWnd, InfoLevel, ByVal 0, CtrlCmd)

    'プリンタオブジェクトをクローズ
    Ret = Api_ClosePrinter(hWnd)
End Sub

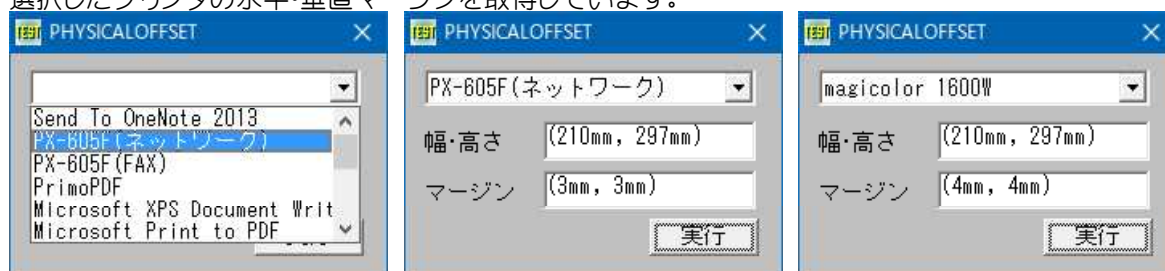
'=====
' =
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの水平・垂直マージンを取得

GetDeviceCaps デバイス固有の情報を取得
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
MoveMemory メモリブロックを別の領域に移動
CreateDC 指定されたデバイスのデバイスコンテキストを、指定された名前で作成
DeleteDC 指定されたデバイスコンテキストを削除

選択したプリンタの水平・垂直マージンを取得しています。



```

'=====
'= プリンタの水平・垂直マージンを取得
'= (PhysicalOffset.bas)
'=====
#include "Windows.bi"

Type PRINTER_INFO_5
    pPrinterName           As Long
    pPortName              As Long
    Attributes              As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

' デバイス固有の情報を取得
Declare Function Api_GetDeviceCaps& Lib "gdi32" Alias "GetDeviceCaps" (ByVal hDC&, ByVal nIndex&)

' 使用可能なプリンタ・プリントサーバーなどを列挙する
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&, pcReturned&)

' メモリブロックを別の領域に移動する
Declare Sub Api_MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length&)

' 指定されたデバイスのデバイスコンテキストを、指定された名前で作成
Declare Function Api_CreateDC& Lib "gdi32" Alias "CreateDCA" (ByVal lpDriverName$, lpDeviceName As Any, lpOutput As Any, ByVal lpInitData As Any)

' 指定されたデバイスコンテキストを削除
Declare Function Api_DeleteDC& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)

#define PRINTER_ENUM_NAME &H8
#define MAX_DEVICENAME 64
#define PHYSICALHEIGHT 111
#define PHYSICALOFFSETX 112
#define PHYSICALOFFSETY 113
#define PHYSICALWIDTH 110
#define LOGPIXELSX 88
#define LOGPIXELSY 90

'Nameで指定されたプリンタを列挙
'
'物理的高さ(単位ピクセル)
'実際に印刷可能なx方向のマージン
'実際に印刷可能なy方向のマージン
'物理的幅(単位ピクセル)
'画面(プリンタ)の幅のピクセル数(Pixel/inch)
'画面(プリンタ)の高さのピクセル数(Pixel/inch)

Var Shared Text(3) As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
Next i
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Need As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    'ローカルプリンタから検索
    PrintServer = ""

```

```

'PRINTER_INFO_5構造体を受け取る
Level = 5

'バッファに必要なバイト数を調べる
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Need,
Returned)
If Need = 0 Then End

'全プリンタ情報を得る
Var Buffer(Need - 1) As byte
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Need, Need,
Returned)

'構造体リストの準備
Var pi5(Returned-1) As PRINTER_INFO_5

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    Api_MoveMemory pi5(CT), Buffer(CT * Len(pi5(CT))), Len(pi5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    Api_MoveMemory PrinterName, ByVal pi5(CT).pPrinterName, Len(PrinterName)
    PrinterName = Left$(PrinterName, InStr(1, PrinterName, Chr$(0)) - 1)
    Comb1.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC As Long
    Var PhWidth As Long
    Var PhHeight As Long
    Var PhOffsetX As Long
    Var PhOffsetY As Long

    PrinterName = Comb1.GetWindowText
    hDC = Api_CreateDC(ByVal 0, PrinterName, 0, 0)

    '幅
    PhWidth = Api_GetDeviceCaps(hDC, PHYSICALWIDTH) / Api_GetDeviceCaps(hDC, LOGPIXELSX)
    * 25.4

    '高さ
    PhHeight = Api_GetDeviceCaps(hDC, PHYSICALHEIGHT) / Api_GetDeviceCaps(hDC,
LOGPIXELSY) * 25.4

    '水平マージン
    PhOffsetX = Api_GetDeviceCaps(hDC, PHYSICALOFFSETX) / Api_GetDeviceCaps(hDC,
LOGPIXELSX) * 25.4

    '垂直マージン
    PhOffsetY = Api_GetDeviceCaps(hDC, PHYSICALOFFSETY) / Api_GetDeviceCaps(hDC,
LOGPIXELSY) * 25.4

    '幅高さを表示
    Text(2).SetWindowText "(" & Trim$(Str$(CInt(PhWidth))) & "mm" & ", " &
Trim$(Str$(CInt(PhHeight))) & "mm" & ")"

    'マージンを表示
    Text(3).SetWindowText "(" & Trim$(Str$(CInt(PhOffsetX))) & "mm" & ", " &
Trim$(Str$(CInt(PhOffsetY))) & "mm" & ")"

    'デバイスコンテキストを削除
    Ret = Api_DeleteDC(hDC)
End Sub

```

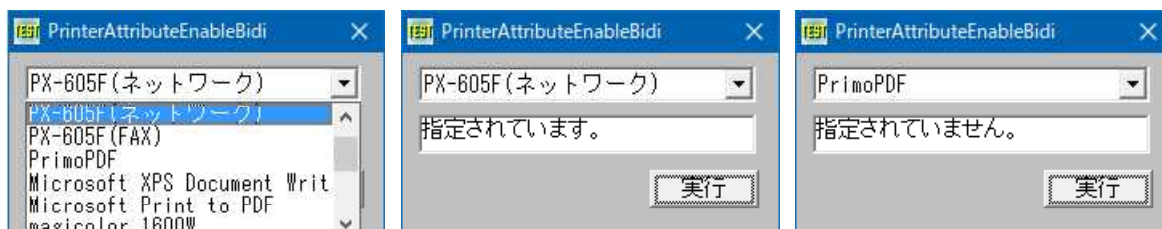
```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの双方向通信を判定する

OpenPrinter プリンタオブジェクトをオープン
GetPrinter プリンタの詳細な情報を取得
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
MoveMemory メモリの指定領域をコピー
ClosePrinter プリンタオブジェクトを閉じる
PRINTER_ATTRIBUTE_ENABLE_BIDI (&H800) 双方向通信



```

'=====
'= プリンタの双方向通信を判定する
'= (PrinterAttributeEnableBidi.bas)
'=====
#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define PRINTER_ATTRIBUTE_ENABLE_BIDI &H800 '双方向通信
#define PRINTER_ENUM_DEFAULT &H1 'デフォルトのプリンタに関する情報を列挙
#define PRINTER_ENUM_LOCAL &H2 'Nameの設定を無視して、ローカルプリンタを列挙
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙
#define PRINTER_ENUM_SHARED &H20 '共有属性を持つプリンタを列挙
#define MAX_DEVICENAME 64

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

```

' 使用可能なプリンタ・プリントサーバーなどを列挙

```
Declare Function Api_EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal  
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,  
pcReturned&)
```

' メモリの指定領域をコピー

```
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,  
ByVal Length&)
```

' プリンタオブジェクトを閉じる

```
Declare Function Api_ClosePrinter Lib "winspool.drv" Alias "ClosePrinter" (ByVal  
hPrinter&)
```

```
Var Shared Text1 As Object  
Var Shared Combo1 As Object  
Var Shared Button1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14  
Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
Var Shared PrinterName As String
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()  
    Var PrintServer As String  
    Var Needed As Long  
    Var Returned As Long  
    Var Level As Long  
    Var CT As Long  
    Var Ret As Long
```

' ローカルプリンタから検索

```
PrintServer = ""
```

' PRINTER_INFO_5 構造体を受け取る

```
Level = 5
```

' バッファに必要なバイト数を調べる

```
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,  
Returned)  
If Needed = 0 Then End
```

' 全プリンタ情報を得る

```
Var Buffer(Needed - 1) As Byte
```

```
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,  
Needed, Returned)
```

' 構造体リストの準備

```
Var pi5(Returned - 1) As PRINTER_INFO_5
```

```
For CT = 0 To Returned - 1
```

' バッファから構造体1つ分を抜き取る

```
MoveMemory pi5(CT), Buffer(CT * Len(pi5(CT))), Len(pi5(CT))
```

' プリンタ名を得る

```
PrinterName = String$(MAX_DEVICENAME, Chr$(0))  
MoveMemory PrinterName, ByVal pi5(CT).pPrinterName, Len(PrinterName)  
PrinterName = KLeft$(PrinterName, kInStr(1, PrinterName, Chr$(0)) - 1)
```

```
Combo1.AddString PrinterName
```

```
Next
```

```
End Sub
```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pd As PRINTER_DEFAULTS
    Var pi5 As PRINTER_INFO_5
    Var PrinterHandle As Long
    Var Level As Long
    Var Needed As Long
    Var Ret As Long

    Text1.SetWindowText ""

    'プリンタ名を指定
    PrinterName = Comb1.GetWindowText

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(PrinterName, PrinterHandle, pd)

    '構造体のレベルを指定
    Level = 5

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinter(PrinterHandle, Level, ByVal 0, 0, Needed)

    'バッファを確保
    Var Buffer(Needed - 1)

    '詳細なプリンタ情報を取得
    Ret = Api_GetPrinter(PrinterHandle, Level, Buffer(0), Needed, Needed)

    '取得した詳細なプリンタ情報を構造体へ移動
    MoveMemory pi5, Buffer(0), Len(pi5)

    '双方向通信を表示
    '指定されているとき
    If pi5.Attributes And PRINTER_ATTRIBUTE_ENABLE_BIDI Then
        Text1.SetWindowText "指定されています。"

    '指定されていないとき
    Else
        Text1.SetWindowText "指定されていません。"
    End If

    'プリンタオブジェクトをクローズ
    Ret = Api_ClosePrinter(PrinterHandle)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの速度を取得

DeviceCapabilities プリンタデバイスドライバの能力を取得
CopyMemory ある位置から別の位置にメモリブロックを移動
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
DC_PRINTRATE(26) 速度を取得することを示す

DC_PRINTRATEUNIT (27) 速度の単位を取得することを示す



```
'=====
'= プリンタの速度を取得
'= (DC_PRINTRATE.bas)
'=====

#include "Windows.bi"

#define PRINTER_ENUM_DEFAULT &H1
#define PRINTER_ENUM_LOCAL &H2
#define PRINTER_ENUM_NAME &H8
#define PRINTER_ENUM_SHARED &H20
#define MAX_DEVICENAME 64
#define DC_PRINTRATE 26
#define DC_PRINTRATEUNIT 27
#define PRINTRATEUNIT_PPM 1
#define PRINTRATEUNIT_CPS 2
#define PRINTRATEUNIT_LPM 3
#define PRINTRATEUNIT_IPM 4

' デフォルトのプリンタに関する情報を列挙
' Nameの設定を無視して、ローカルプリンタを列挙
' Nameで指定されたプリンタを列挙
' 共有属性を持つプリンタを列挙
'
' 速度を取得することを示す定数
' 速度の単位を取得することを示す定数
' PPM単位を示す定数
' CPS単位を示す定数
' LPM単位を示す定数
' IPM単位を示す定数

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

' プリンタデバイスドライバの能力を取得
Declare Function Api_DeviceCapabilities Lib "winspool.drv" Alias "DeviceCapabilitiesA"
    (ByVal lpDeviceName$, ByVal lpPort$, ByVal iIndex&, lpOutput As Any, lpDevMode As Any)

' ある位置から別の位置にメモリブロックを移動する関数の宣言
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
    As Any, ByVal Length&)

' 使用可能なプリンタ・プリントサーバーなどを列挙する
Declare Function Api_EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
    Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
    pcReturned&)

Var Shared Comb1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterName As String

'=====
'=
'=====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
```



```

Var Ret As Long

'ローカルプリンタから検索
PrintServer = ""

'PRINTER_INFO_5構造体を受け取る
Level = 5

'バッファに必要なバイト数を調べる
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
If Needed = 0 Then End

'全プリンタ情報を得る
Var Buffer(Needed - 1) As byte
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)

'構造体リストの準備
Var PI_5(Returned - 1) As PRINTER_INFO_5

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
    PrinterName = KLeft$(PrinterName, KInStr(1, PrinterName, Chr$(0)) - 1)

    Combo1.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var DeviceCapability As Long
    Var PortName As String
    Var PrintRate As Long
    Var PrintRateUnit As Long

    'デバイス名
    PrinterName = Combo1.GetText(Combo1.GetCursel)

    '問い合わせる能力を指定
    DeviceCapability = DC_PRINTRATE

    '速度を取得
    PrintRate = Api_DeviceCapabilities(PrinterName, PortName, DeviceCapability, ByVal 0,
ByVal 0)

    '問い合わせる能力を指定
    DeviceCapability = DC_PRINTRATEUNIT

    '速度の単位を取得
    PrintRateUnit = Api_DeviceCapabilities(PrinterName, PortName, DeviceCapability,
ByVal 0, ByVal 0)

    '取得できたときは
    If PrintRate > 0 Then

        '速度の単位で分岐
        Select Case PrintRateUnit

```

```

' PPMのときは
Case PRINTRATEUNIT_PPM
    Text1.SetWindowText Str$(PrintRate) & "ppm"

' CPSのときは
Case PRINTRATEUNIT_CPS
    Text1.SetWindowText Str$(PrintRate) & "cps"

' LPMのときは
Case PRINTRATEUNIT_LPM
    Text1.SetWindowText Str$(PrintRate) & "lpm"

' IPMのときは
Case PRINTRATEUNIT_IPM
    Text1.SetWindowText Str$(PrintRate) & "ipm"

Case Else
    Text1.SetWindowText Str$(PrintRate) & "(単位不明)"
End Select

' 取得できないとき
Else
    Text1.SetWindowText "取得できません！"
End If
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの対応する用紙を取得

接続されているプリンタを列挙し、選択したプリンタの使用できる用紙の種類を取得します。

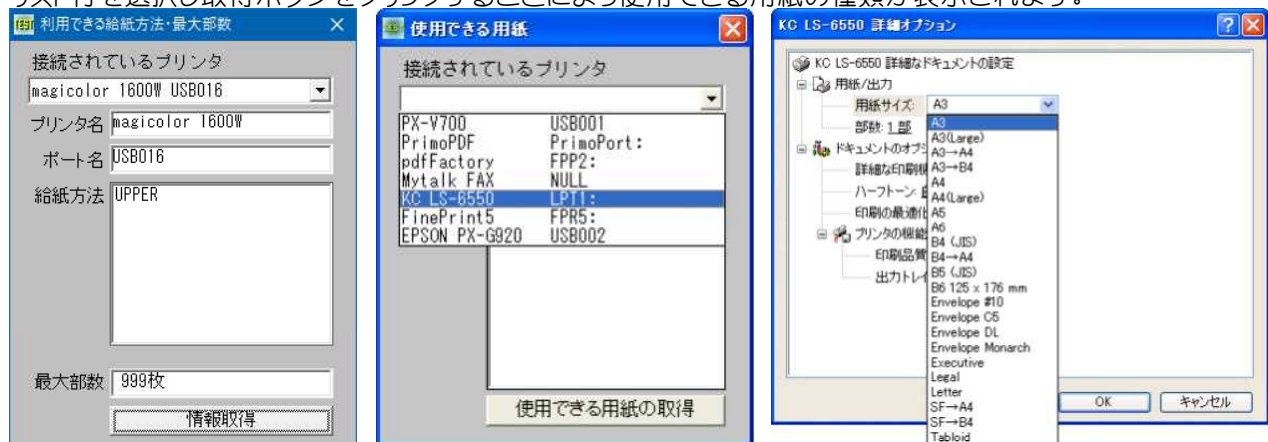
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙

RtlMoveMemory メモリブロックを別の領域に移動

DeviceCapabilities プリンタデバイスドライバの能力を取得

接続されているプリンタ名およびポート名がコンボボックスにリスト表示されます。

リスト行を選択し取得ボタンをクリックすることにより使用できる用紙の種類が表示されます。



```

' =====
' = 接続されているプリンタの使用できる用紙取得
' = (DeviceCapabilities.bas)
' =====
#include "Windows.bi"

```

```

Type PRINTER_INFO_5
    pPrinterName      As Long
    pPortName         As Long
    Attributes        As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

```

・ 使用可能なプリンタ・プリントサーバーなどを列挙する

```

Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags& , ByVal Name$ , ByVal Level& , pPrinterEnum As Any , ByVal cbBuf& , pcbNeeded& ,
pcReturned& )

```

```

#define PRINTER_ENUM_NAME &H8
#define MAX_DEVICENAME 64

```

・ メモリブロックを別の領域に移動する

```

Declare Sub Api_MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any,
Source As Any, ByVal Length&)

```

・ プリンタデバイスドライバの能力を取得する関数の宣言

```

Declare Function Api_DeviceCapabilities& Lib "winspool.drv" Alias "DeviceCapabilitiesA"
(ByVal lpDeviceName$, ByVal lpPort$, ByVal iIndex&, lpOutput As Any, lpDevMode As Any)

```

・ プリンタで利用できる用紙を取得することを示す定数の宣言

```

#define DC_PAPERS 2

```

・ 用紙を示す定数の宣言

```

#define DMPAPER_LETTER 1
#define DMPAPER_LETTERSMALL 2
#define DMPAPER_TABLOID 3
#define DMPAPER_LEDGER 4
#define DMPAPER_LEGAL 5
#define DMPAPER_STATEMENT 6
#define DMPAPER_EXECUTIVE 7
#define DMPAPER_A3 8
#define DMPAPER_A4 9
#define DMPAPER_A4SMALL 10
#define DMPAPER_A5 11
#define DMPAPER_B4 12
#define DMPAPER_B5 13
#define DMPAPER_FOLIO 14
#define DMPAPER_QUARTO 15
#define DMPAPER_10X14 16
#define DMPAPER_11X17 17
#define DMPAPER_NOTE 18
#define DMPAPER_ENV_9 19
#define DMPAPER_ENV_10 20
#define DMPAPER_ENV_11 21
#define DMPAPER_ENV_12 22
#define DMPAPER_ENV_14 23
#define DMPAPER_CSHEET 24
#define DMPAPER_DSHEET 25
#define DMPAPER_ESHEET 26
#define DMPAPER_ENV_DL 27
#define DMPAPER_ENV_C5 28
#define DMPAPER_ENV_C3 29
#define DMPAPER_ENV_C4 30
#define DMPAPER_ENV_C6 31
#define DMPAPER_ENV_C65 32
#define DMPAPER_ENV_B4 33
#define DMPAPER_ENV_B5 34
#define DMPAPER_ENV_B6 35
#define DMPAPER_ENV_ITALY 36
#define DMPAPER_ENV_MONARCH 37
#define DMPAPER_ENV_PERSONAL 38
#define DMPAPER_FANFOLD_US 39
#define DMPAPER_FANFOLD_STD_GERMAN 40
#define DMPAPER_FANFOLD_LGL_GERMAN 41
#define DMPAPER_ISO_B4 42

```

```

#define DMPAPER_JAPANESE_POSTCARD 43
#define DMPAPER_9X11 44
#define DMPAPER_10X11 45
#define DMPAPER_15X11 46
#define DMPAPER_ENV_INVITE 47
#define DMPAPER_RESERVED_48 48
#define DMPAPER_RESERVED_49 49
#define DMPAPER_LETTER_EXTRA 50
#define DMPAPER_LEGAL_EXTRA 51
#define DMPAPER_TABLOID_EXTRA 52
#define DMPAPER_A4_EXTRA 53
#define DMPAPER_LETTER_TRANSVERSE 54
#define DMPAPER_A4_TRANSVERSE 55
#define DMPAPER_LETTER_EXTRA_TRANSVERSE 56
#define DMPAPER_A_PLUS 57
#define DMPAPER_B_PLUS 58
#define DMPAPER_LETTER_PLUS 59
#define DMPAPER_A4_PLUS 60
#define DMPAPER_A5_TRANSVERSE 61
#define DMPAPER_B5_TRANSVERSE 62
#define DMPAPER_A3_EXTRA 63
#define DMPAPER_A5_EXTRA 64
#define DMPAPER_B5_EXTRA 65
#define DMPAPER_A2 66
#define DMPAPER_A3_TRANSVERSE 67
#define DMPAPER_A3_EXTRA_TRANSVERSE 68
#define DMPAPER_DBL_JAPANESE_POSTCARD 69
#define DMPAPER_A6 70
#define DMPAPER_JENV_KAKU2 71
#define DMPAPER_JENV_KAKU3 72
#define DMPAPER_JENV_CHOU3 73
#define DMPAPER_JENV_CHOU4 74
#define DMPAPER_LETTER_ROTATED 75
#define DMPAPER_A3_ROTATED 76
#define DMPAPER_A4_ROTATED 77
#define DMPAPER_A5_ROTATED 78
#define DMPAPER_B4_JIS_ROTATED 79
#define DMPAPER_B5_JIS_ROTATED 80
#define DMPAPER_JAPANESE_POSTCARD_ROTATED 81
#define DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED 82
#define DMPAPER_A6_ROTATED 83
#define DMPAPER_JENV_KAKU2_ROTATED 84
#define DMPAPER_JENV_KAKU3_ROTATED 85
#define DMPAPER_JENV_CHOU3_ROTATED 86
#define DMPAPER_JENV_CHOU4_ROTATED 87
#define DMPAPER_B6_JIS 88
#define DMPAPER_B6_JIS_ROTATED 89
#define DMPAPER_12X11 90
#define DMPAPER_JENV_YOU4 91
#define DMPAPER_JENV_YOU4_ROTATED 92
#define DMPAPER_P16K 93
#define DMPAPER_P32K 94
#define DMPAPER_P32KBIG 95
#define DMPAPER_PENV_1 96
#define DMPAPER_PENV_2 97
#define DMPAPER_PENV_3 98
#define DMPAPER_PENV_4 99
#define DMPAPER_PENV_5 100
#define DMPAPER_PENV_6 101
#define DMPAPER_PENV_7 102
#define DMPAPER_PENV_8 103
#define DMPAPER_PENV_9 104
#define DMPAPER_PENV_10 105
#define DMPAPER_P16K_ROTATED 106
#define DMPAPER_P32K_ROTATED 107
#define DMPAPER_P32KBIG_ROTATED 108
#define DMPAPER_PENV_1_ROTATED 109
#define DMPAPER_PENV_2_ROTATED 110
#define DMPAPER_PENV_3_ROTATED 111
#define DMPAPER_PENV_4_ROTATED 112

```

```

#define DMPAPER_PENV_5_ROTATED 113
#define DMPAPER_PENV_6_ROTATED 114
#define DMPAPER_PENV_7_ROTATED 115
#define DMPAPER_PENV_8_ROTATED 116
#define DMPAPER_PENV_9_ROTATED 117
#define DMPAPER_PENV_10_ROTATED 118
#define DMPAPER_USER 256

Var Shared List1 As Object
Var Shared Comb1 As Object
Var Shared Text(5) As Object

Comb1.Attach GetDlgItem("Comb1")
Comb1.SetFontSize 14
List1.Attach GetDlgItem("List1")
List1.SetFontSize 14
For i = 0 To 5
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i+1)))
    Text(i).SetFontSize 14
Next i

Var Shared PrinterName$ As String
Var Shared PortName$ As String

'=====
'=
'=====
Declare Sub Mainform_Start edec1 ()
Sub Mainform_Start()
    Var PrintServer$ As String
    Var Need As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    'ローカルプリンタから検索(サーバー名 ¥¥xxx 指定可)
    PrintServer$ = ""

    'PRINTER_INFO_5構造体を受け取る
    Level = 5

    'バッファに必要なバイト数を調べる
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer$, Level, Chr$(0), 0, Need,
Returned)
    If Need = 0 Then End

    '全プリンタ情報を得る
    Var Buffer(Need-1) As Byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer$, Level, Buffer(0), Need,
Need, Returned)

    '構造体リストの準備
    Var PI_5(Returned-1) As PRINTER_INFO_5

    For CT = 0 To Returned - 1

        'バッファから構造体1つ分を抜き取る
        Api_MoveMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

        'プリンタ名を得る
        PrinterName$ = String$(MAX_DEVICENAME, Chr$(0))
        Api_MoveMemory PrinterName$, ByVal PI_5(CT).pPrinterName, Len(PrinterName$)
        PrinterName$ = kLeft$(PrinterName$, kInStr(1, PrinterName$, Chr$(0)) - 1)

        'ポート名を得る
        PortName$ = String$(MAX_DEVICENAME, Chr$(0))
        Api_MoveMemory PortName$, ByVal PI_5(CT).pPortName, Len(PortName$)
        PortName$ = kLeft$(PortName$, kInStr(1, PortName$, Chr$(0)) - 1)
        Comb1.AddString Left$(PrinterName$ & Space$(16), 16) & PortName$
    
```

```

Next
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    PrinterName$ = Trim$(Left$(Combo1.GetText(Combo1.GetCursel), 16))
    PortName$ = Trim$(Mid$(Combo1.GetText(Combo1.GetCursel), 17))
    Text(4).SetWindowText PrinterName$
    Text(5).SetWindowText PortName$
    List1.ResetContent
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Device As Long
    Var Need As Long
    Var Ret As Long
    Var CT As Integer

    List1.ResetContent
    Device = DC_PAPERS

    'バッファに必要なサイズを取得
    Need = Api_DeviceCapabilities(PrinterName$, PortName$, Device, ByVal 0, ByVal 0)

    'バッファを確保
    Var Paper(Need-1) As Integer

    '使用できる用紙を取得
    Ret = Api_DeviceCapabilities(PrinterName$, PortName$, Device, Paper(0), ByVal 0)

    '使用できる給紙方法を列挙
    For CT = 0 To Need-1

        '用紙を表示
        Select Case Paper(CT)
            Case DMPAPER_LETTER
                List1.AddString "LETTER"
            Case DMPAPER_LETTERSMALL
                List1.AddString "LETTERSMALL"
            Case DMPAPER_TABLOID
                List1.AddString "TABLOID"
            Case DMPAPER_LEDGER
                List1.AddString "LEDGER"
            Case DMPAPER_LEGAL
                List1.AddString "LEGAL"
            Case DMPAPER_STATEMENT
                List1.AddString "STATEMENT"
            Case DMPAPER_EXECUTIVE
                List1.AddString "EXECUTIVE"
            Case DMPAPER_A3
                List1.AddString "A3"
            Case DMPAPER_A4
                List1.AddString "A4"
            Case DMPAPER_A4SMALL
                List1.AddString "A4SMALL"
            Case DMPAPER_A5
                List1.AddString "A5"
            Case DMPAPER_B4
                List1.AddString "B4"
            Case DMPAPER_B5
                List1.AddString "B5"
            Case DMPAPER_FOLIO
                List1.AddString "FOLIO"
        End Select
    Next
End Sub

```

```

Case DMPAPER_QUARTO
    List1.AddString "QUARTO"
Case DMPAPER_10X14
    List1.AddString "10X14"
Case DMPAPER_11X17
    List1.AddString "11X17"
Case DMPAPER_NOTE
    List1.AddString "NOTE"
Case DMPAPER_ENV_9
    List1.AddString "ENV_9"
Case DMPAPER_ENV_10
    List1.AddString "ENV_10"
Case DMPAPER_ENV_11
    List1.AddString "ENV_11"
Case DMPAPER_ENV_12
    List1.AddString "ENV_12"
Case DMPAPER_ENV_14
    List1.AddString "ENV_14"
Case DMPAPER_CSHEET
    List1.AddString "CSHEET"
Case DMPAPER_DSHEET
    List1.AddString "DSHEET"
Case DMPAPER_ESHEET
    List1.AddString "ESHEET"
Case DMPAPER_ENV_DL
    List1.AddString "ENV_DL"
Case DMPAPER_ENV_C5
    List1.AddString "ENV_C5"
Case DMPAPER_ENV_C3
    List1.AddString "ENV_C3"
Case DMPAPER_ENV_C4
    List1.AddString "ENV_C4"
Case DMPAPER_ENV_C6
    List1.AddString "ENV_C6"
Case DMPAPER_ENV_C65
    List1.AddString "ENV_C65"
Case DMPAPER_ENV_B4
    List1.AddString "ENV_B4"
Case DMPAPER_ENV_B5
    List1.AddString "ENV_B5"
Case DMPAPER_ENV_B6
    List1.AddString "ENV_B6"
Case DMPAPER_ENV_ITALY
    List1.AddString "ENV_ITALY"
Case DMPAPER_ENV_MONARCH
    List1.AddString "ENV_MONARCH"
Case DMPAPER_ENV_PERSONAL
    List1.AddString "ENV_PERSONAL"
Case DMPAPER_FANFOLD_US
    List1.AddString "FANFOLD_US"
Case DMPAPER_FANFOLD_STD_GERMAN
    List1.AddString "FANFOLD_STD_GERMAN"
Case DMPAPER_FANFOLD_LGL_GERMAN
    List1.AddString "FANFOLD_LGL_GERMAN"
Case DMPAPER_ISO_B4
    List1.AddString "ISO_B4"
Case DMPAPER_JAPANESE_POSTCARD
    List1.AddString "JAPANESE_POSTCARD"
Case DMPAPER_9X11
    List1.AddString "9X11"
Case DMPAPER_10X11
    List1.AddString "10X11"
Case DMPAPER_15X11
    List1.AddString "15X11"
Case DMPAPER_ENV_INVITE
    List1.AddString "ENV_INVITE"
Case DMPAPER_RESERVED_48
    List1.AddString "RESERVED_48"
Case DMPAPER_RESERVED_49
    List1.AddString "RESERVED_49"

```

```

Case DMPAPER_LETTER_EXTRA
    List1.AddString "LETTER_EXTRA"
Case DMPAPER_LEGAL_EXTRA
    List1.AddString "LEGAL_EXTRA"
Case DMPAPER_TABLOID_EXTRA
    List1.AddString "TABLOID_EXTRA"
Case DMPAPER_A4_EXTRA
    List1.AddString "A4_EXTRA"
Case DMPAPER_LETTER_TRANSVERSE
    List1.AddString "LETTER_TRANSVERSE"
Case DMPAPER_A4_TRANSVERSE
    List1.AddString "A4_TRANSVERSE"
Case DMPAPER_A_PLUS
    List1.AddString "A_PLUS"
Case DMPAPER_B_PLUS
    List1.AddString "B_PLUS"
Case DMPAPER_LETTER_PLUS
    List1.AddString "LETTER_PLUS"
Case DMPAPER_A4_PLUS
    List1.AddString "A4_PLUS"
Case DMPAPER_A5_TRANSVERSE
    List1.AddString "A5_TRANSVERSE"
Case DMPAPER_B5_TRANSVERSE
    List1.AddString "B5_TRANSVERSE"
Case DMPAPER_A3_EXTRA
    List1.AddString "A3_EXTRA"
Case DMPAPER_A5_EXTRA
    List1.AddString "A5_EXTRA"
Case DMPAPER_B5_EXTRA
    List1.AddString "B5_EXTRA"
Case DMPAPER_A2
    List1.AddString "A2"
Case DMPAPER_A3_TRANSVERSE
    List1.AddString "A3_TRANSVERSE"
Case DMPAPER_A3_EXTRA_TRANSVERSE
    List1.AddString "A3_EXTRA_TRANSVERSE"
Case DMPAPER_DBL_JAPANESE_POSTCARD
    List1.AddString "DBL_JAPANESE_POSTCARD"
Case DMPAPER_A6
    List1.AddString "A6"
Case DMPAPER_JENV_KAKU2
    List1.AddString "JENV_KAKU2"
Case DMPAPER_JENV_KAKU3
    List1.AddString "JENV_KAKU3"
Case DMPAPER_JENV_CHOU3
    List1.AddString "JENV_CHOU3"
Case DMPAPER_JENV_CHOU4
    List1.AddString "JENV_CHOU4"
Case DMPAPER_LETTER_ROTATED
    List1.AddString "LETTER_ROTATED"
Case DMPAPER_A3_ROTATED
    List1.AddString "A3_ROTATED"
Case DMPAPER_A4_ROTATED
    List1.AddString "A4_ROTATED"
Case DMPAPER_A5_ROTATED
    List1.AddString "A5_ROTATED"
Case DMPAPER_B4_JIS_ROTATED
    List1.AddString "B4_JIS_ROTATED"
Case DMPAPER_B5_JIS_ROTATED
    List1.AddString "B5_JIS_ROTATED"
Case DMPAPER_JAPANESE_POSTCARD_ROTATED
    List1.AddString "JAPANESE_POSTCARD_ROTATED"
Case DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED
    List1.AddString "DBL_JAPANESE_POSTCARD_ROTATED"
Case DMPAPER_A6_ROTATED
    List1.AddString "A6_ROTATED"
Case DMPAPER_JENV_KAKU2_ROTATED
    List1.AddString "JENV_KAKU2_ROTATED"
Case DMPAPER_JENV_KAKU3_ROTATED
    List1.AddString "JENV_KAKU3_ROTATED"

```



```

Case DMPAPER_JENV_CHOU3_ROTATED
    List1.AddString "JENV_CHOU3_ROTATED"
Case DMPAPER_JENV_CHOU4_ROTATED
    List1.AddString "JENV_CHOU4_ROTATED"
Case DMPAPER_B6_JIS
    List1.AddString "B6_JIS"
Case DMPAPER_B6_JIS_ROTATED
    List1.AddString "B6_JIS_ROTATED"
Case DMPAPER_12X11
    List1.AddString "12X11"
Case DMPAPER_JENV_YOU4
    List1.AddString "JENV_YOU4"
Case DMPAPER_JENV_YOU4_ROTATED
    List1.AddString "JENV_YOU4_ROTATED"
Case DMPAPER_P16K
    List1.AddString "P16K"
Case DMPAPER_P32K
    List1.AddString "P32K"
Case DMPAPER_P32KBIG
    List1.AddString "P32KBIG"
Case DMPAPER_PENV_1
    List1.AddString "PENV_1"
Case DMPAPER_PENV_2
    List1.AddString "PENV_2"
Case DMPAPER_PENV_3
    List1.AddString "PENV_3"
Case DMPAPER_PENV_4
    List1.AddString "PENV_4"
Case DMPAPER_PENV_5
    List1.AddString "PENV_5"
Case DMPAPER_PENV_6
    List1.AddString "PENV_6"
Case DMPAPER_PENV_7
    List1.AddString "PENV_7"
Case DMPAPER_PENV_8
    List1.AddString "PENV_8"
Case DMPAPER_PENV_9
    List1.AddString "PENV_9"
Case DMPAPER_PENV_10
    List1.AddString "PENV_10"
Case DMPAPER_P16K_ROTATED
    List1.AddString "P16K_ROTATED"
Case DMPAPER_P32K_ROTATED
    List1.AddString "P32K_ROTATED"
Case DMPAPER_P32KBIG_ROTATED
    List1.AddString "P32KBIG_ROTATED"
Case DMPAPER_PENV_1_ROTATED
    List1.AddString "PENV_1_ROTATED"
Case DMPAPER_PENV_2_ROTATED
    List1.AddString "PENV_2_ROTATED"
Case DMPAPER_PENV_3_ROTATED
    List1.AddString "PENV_3_ROTATED"
Case DMPAPER_PENV_4_ROTATED
    List1.AddString "PENV_4_ROTATED"
Case DMPAPER_PENV_5_ROTATED
    List1.AddString "PENV_5_ROTATED"
Case DMPAPER_PENV_6_ROTATED
    List1.AddString "PENV_6_ROTATED"
Case DMPAPER_PENV_7_ROTATED
    List1.AddString "PENV_7_ROTATED"
Case DMPAPER_PENV_8_ROTATED
    List1.AddString "PENV_8_ROTATED"
Case DMPAPER_PENV_9_ROTATED
    List1.AddString "PENV_9_ROTATED"
Case DMPAPER_PENV_10_ROTATED
    List1.AddString "PENV_10_ROTATED"
Case Else

```

'ユーザー定義のときは

```

If Paper (CT) >= DMPAPER_USER Then

```

```

        List1.AddString "USER" & "(" & Str$(Paper(CT)) & ")"

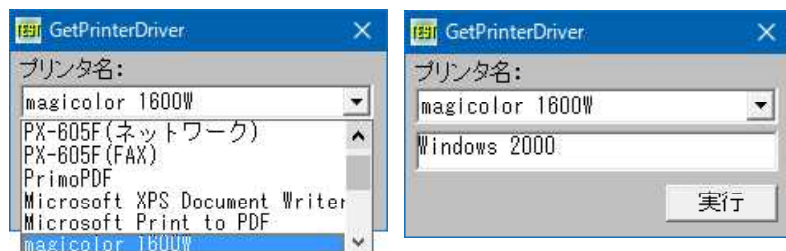
' 未定義のときは
Else
        List1.AddString "UNKNOWN" & "(" & Str$(Paper(CT)) & ")"
End If
End Select
Next CT
End Sub

'=====
'=
'=====
While 1
        WaitEvent
Wend
Stop
End

```

プリンタのドライバ対応osを取得

OpenPrinter プリンタオブジェクトをオープン
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙
GetPrinterDriver プリンタのドライバ情報を取得
ClosePrinter プリンタオブジェクトを閉じる
MoveMemory メモリの指定領域をコピー



```

'=====
'= プリンタのドライバ対応osを取得
'= (GetPrinterDriver2.bas)
'=====
#include "Windows.bi"

#define MAX_DEVICENAME 64
#define STANDARD_RIGHTS_REQUIRED &HF0000
#define PRINTER_ENUM_NAME &H8
#define PRINTER_ACCESS_ADMINISTER &H4
#define PRINTER_ACCESS_USE &H8
#define PRINTER_ALL_ACCESS (STANDARD_RIGHTS_REQUIRED Or PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE)

Type PRINTER_DEFAULTS
    pDatatype As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type DRIVER_INFO_2
    cVersion As Long
    pName As Long
    pEnvironment As Long
    pDriverPath As Long
    pDataFile As Long
    pConfigFile As Long
End Type

Type PRINTER_INFO_5
    pPrinterName As Long

```

```

    pPortName          As Long
    Attributes          As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

' プリントオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' 使用可能なプリンタ・プリントサーバーなどを列挙する
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)

' プリンタのドライバ情報を取得する関数の宣言
Declare Function Api_GetPrinterDriver& Lib "winspool.drv" Alias "GetPrinterDriverA"
(ByVal hPrinter&, ByVal pEnvironment$, ByVal Level&, pDriverInfo As Any, ByVal cdBuf&,
pcbNeeded&)

' プリントオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal length&)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterName As String
'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrintServer As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var CT As Long
    Var Ret As Long

    'ローカルプリンタから検索(サーバー名 ¥¥xxx 指定可)
    PrintServer = ""

    'PRINTER_INFO_5構造体を受け取る
    Level = 5

    'バッファに必要なバイト数を調べる
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
    If Needed = 0 Then End

    '全プリンタ情報を得る
    Var Buffer(Needed - 1) As byte

    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)

    '構造体リストの準備
    Var PI_5(Returned - 1) As PRINTER_INFO_5

```

```

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    MoveMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    MoveMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
    PrinterName = kLeft$(PrinterName$, KInStr(1, PrinterName, Chr$(0)) - 1)

    Combo1.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var PrinterDeviceName As String
    Var pd As PRINTER_DEFAULTS
    Var di2 As DRIVER_INFO_2
    Var hPrinter As Long
    Var Level As Long
    Var Needed As Long
    Var Ret As Long

    'プリンタ名を指定
    PrinterDeviceName = Combo1.GetText(Combo1.GetCursel)
    If PrinterDeviceName = "" Then Exit Sub

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(PrinterDeviceName, hPrinter, pd)

    '構造体のレベルを指定
    Level = 2

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinterDriver(hPrinter, ByVal 0, Level, ByVal 0, 0, Needed)

    'バッファを確保
    Var Buffer(Needed - 1)

    'プリンタのドライバ情報を取得
    Ret = Api_GetPrinterDriver(hPrinter, ByVal 0, Level, Buffer(0), Needed, Needed)

    '取得したプリンタのドライバ情報を構造体へ移動
    MoveMemory di2, Buffer(0), Len(di2)

    'プリンタのドライバ対応OSを表示
    Select Case di2.cVersion
        Case 0
            Text2.SetWindowText "Windows 95 or Windows 98"
        Case 1
            Text2.SetWindowText "Windows NT 3.51"
        Case 2
            Text2.SetWindowText "Windows NT 4.0"
        Case 3
            Text2.SetWindowText "Windows 2000"
        Case Else
            Text2.SetWindowText "判定できません。"
    End Select

    'プリンタオブジェクトをクローズ
    Ret = Api_ClosePrinter(hPrinter)
End Sub

```

```

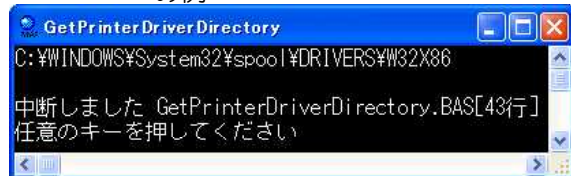
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

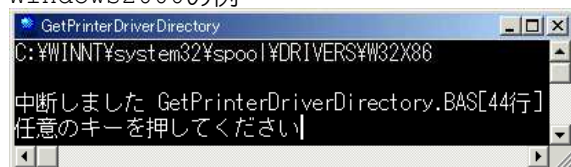
プリンタのドライバディレクトリを取得

GetPrinterDriverDirectory プリンタのドライバディレクトリを取得

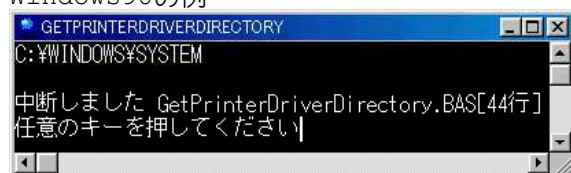
WindowsXPの例



Windows2000の例



Windows98の例



```

'=====
'= プリンタのドライバディレクトリを取得
'= (GetPrinterDriverDirectory.bas)
'=====

```

' プリンタのドライバディレクトリを取得

```

Declare Function Api_GetPrinterDriverDirectory Lib "winspool.drv" Alias
"GetPrinterDriverDirectoryA" (ByVal pName$, ByVal pEnvironment$, ByVal level&, ByVal
pDriverDirectory$, ByVal cbBuff&, pcbNeeded&)

```

```

'=====
'= Null[Chr$(0)]を取り除く
'=====

```

```

Declare Function TrimNull (item As String) As String
Function TrimNull(item As String) As String
    Var ePos As Integer

```

```

    ePos = InStr(item, Chr$(0))
    If ePos Then
        TrimNull = Left$(item, ePos - 1)
    Else
        TrimNull = item
    End If
End Function

```

```

'-----
Var Level As Long
Var cbBuff As Long
Var pcbNeeded As Long
Var pName As String
Var pEnvironment As String
Var pDriverDirectory As String

```

```

Level = 1
cbBuff = 0
pDriverDirectory = Chr$(0)
pName = Chr$(0)
pEnvironment = Chr$(0)

If Api_GetPrinterDriverDirectory(pName, pEnvironment, Level, pDriverDirectory, cbBuff,
pcbNeeded) = 0 Then
    pDriverDirectory = Space$(pcbNeeded)
    cbBuff = Len(pDriverDirectory)
    If Api_GetPrinterDriverDirectory(pName, pEnvironment, Level, pDriverDirectory,
cbBuff, pcbNeeded) = 1 Then
        Print TrimNull(pDriverDirectory)
    End If
End If

Stop
End

```

プリンタのドライバ名を取得

OpenPrinter プリンタオブジェクトをオープン
GetPrinterDriver プリンタのドライバ情報を取得
ClosePrinter プリンタオブジェクトを閉じる
MoveMemory メモリの指定領域をコピー
lstrcpy 文字列をコピーする



```

'=====
' プリンタのドライバ名を取得
' (GetPrinterDriver.bas)
'=====
#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDataType As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す定数の宣言
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す定数宣言
#define PRINTER_ALL_ACCESS (STANDARD_RIGHTS_REQUIRED Or PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE) '(&HF0000 or &H4 or &H8)

Type DRIVER_INFO_2
    cVersion As Long
    pName As Long
    pEnvironment As Long
    pDriverPath As Long
    pDataFile As Long
    pConfigFile As Long
End Type

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

```

' プリンタのドライバ情報を取得する関数の宣言

```
Declare Function Api_GetPrinterDriver& Lib "winspool.drv" Alias "GetPrinterDriverA"  
(ByVal hPrinter&, ByVal pEnvironment$, ByVal Level&, pDriverInfo As Any, ByVal cdBuf&,  
pcbNeeded&)
```

' プリンタオブジェクトを閉じる

```
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal  
hPrinter&)
```

' メモリの指定領域をコピー

```
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,  
ByVal length&)
```

' 文字列をコピーする

```
Declare Function Api_lstrcpy& Lib "Kernel32" Alias "lstrcpyA" (lpszString1 As Any,  
lpszString2 As Any)
```

```
Var Shared Text1 As Object  
Var Shared Text2 As Object  
Var Shared Edit1 As Object  
Var Shared Button1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14  
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14  
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
' =====  
' =  
' =====
```

```
Declare Sub Button1_on edec1 ()  
Sub Button1_on()  
    Var PrtDevName As String  
    Var pd As PRINTER_DEFAULTS  
    Var di2 As DRIVER_INFO_2  
    Var hPrinter As Long  
    Var Level As Long  
    Var Need As Long  
    Var DrvInfo2Env As String * 128  
    Var DrvInfo2Name As String * 128  
    Var Ret As Long
```

' プリンタ名を指定

```
PrtDevName = Edit1.GetWindowText
```

' プリンタアクセス権を指定

```
pd.DesiredAccess = PRINTER_ALL_ACCESS
```

' プリンタのオブジェクトハンドルを取得

```
Ret = Api_OpenPrinter(PrtDevName, hPrinter, pd)
```

' 構造体のレベルを指定

```
Level = 2
```

' バッファに必要なサイズを取得

```
Ret = Api_GetPrinterDriver(hPrinter, ByVal 0, Level, ByVal 0, 0, Need)
```

' バッファを確保

```
Var Buffer(Need - 1) As Byte
```

' プリンタのドライバ情報を取得

```
Ret = Api_GetPrinterDriver(hPrinter, ByVal 0, Level, Buffer(0), Need, Need)
```

' 取得したプリンタのドライバ情報を構造体へ

```
MoveMemory di2, Buffer(0), Len(di2)
```

```
Ret = Api_lstrcpy(DrvInfo2Name, ByVal di2.pName)
```

' 対応環境を切り出し

```
Edit1.SetWindowText PrtDevName
```

```

Text2.SetWindowText Left$(DrvInfo2Name, InStr(DrvInfo2Name, Chr$(0)) - 1)

' プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

' =====
' =
' =====
Declare Sub Edit1_SetFocus edec1 ()
Sub Edit1_SetFocus ()
    Text2.SetWindowText ""
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタのドライバを列挙

EnumPrinterDrivers プリンタのドライバを列挙
MoveMemory メモリの指定領域をコピー
lstrcpy 文字列をコピーする



```

' =====
' = プリンタのドライバを列挙
' = (EnumPrinterDrivers.bas)
' =====
#include "Windows.bi"

Type DRIVER_INFO_1
    pName As Long
End Type

' プリンタのドライバを列挙
Declare Function Api_EnumPrinterDrivers Lib "winspool.drv" Alias "EnumPrinterDriversA"
    (ByVal pName$, ByVal pEnvironment$, ByVal Level&, pDriverInfo As Any, ByVal cdBuf&,
    pcbNeeded&, pcReturned&)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
    ByVal Length&)

' 文字列をコピーする
Declare Function Api_lstrcpy Lib "Kernel32" Alias "lstrcpy" (lpszString1 As Any,
    lpszString2 As Any)

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

```



```

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Level As Long
    Var Needed As Long
    Var Returned As Long
    Var Count As Long
    Var DriverName As String * 128
    Var Ret As Long

    List1.Resetcontent

    '構造体のレベルを指定
    Level = 1

    'バッファに必要なサイズを取得
    Ret = Api_EnumPrinterDrivers (ByVal 0, ByVal 0, Level, ByVal 0, 0, Needed, Returned)

    'ドライバが取得できないとき
    If Needed <= 0 Then
        List1.AddString "取得できません！"
        Exit Sub
    End If

    'バッファを確保
    Var bytDriverInfo1Buffer (Needed - 1) As Byte

    'ドライバ情報を取得
    Ret = Api_EnumPrinterDrivers (ByVal 0, ByVal 0, Level, bytDriverInfo1Buffer(0),
    Needed, Needed, Returned)

    '取得したドライバ情報の構造体を確保
    Var udtDriverInfo1 (Returned - 1) As DRIVER_INFO_1

    '取得したドライバ情報を構造体へ移動
    MoveMemory udtDriverInfo1(0), bytDriverInfo1Buffer(0), Len(udtDriverInfo1(0)) *
Returned

    'バッファにドライバ情報がある間は
    For Count = 0 To Returned - 1

        'ドライバ名を複写
        Ret = Api_lstrcpy (DriverName, ByVal udtDriverInfo1 (Count).pName)

        'ドライバ名を表示
        List1.AddString Left$(DriverName, InStr(DriverName, Chr$(0)) - 1)
    Next Count
End Sub

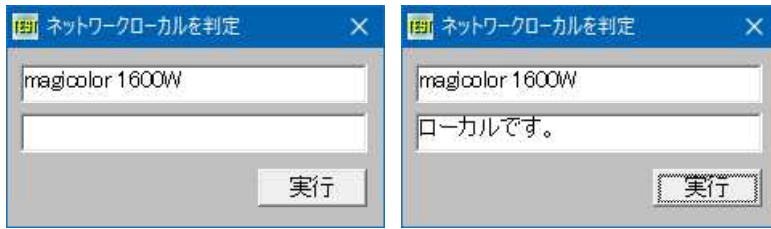
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタのネットワークローカルを判定

OpenPrinter プリンタオブジェクトをオープン
GetPrinter プリンタの詳細な情報を取得
ClosePrinter プリンタオブジェクトを閉じる

CopyMemory メモリの指定領域をコピー



```
'=====
'= プリンタのネットワークローカルを判定
'= (GetPrinter2.bas)
'=====
#include "Windows.bi"

Type PRINTER_DEFAULTS
    pDatatype           As Long
    pDevMode             As Long
    DesiredAccess        As Long
End Type

Type PRINTER_INFO_5
    pPrinterName         As Long
    pPortName            As Long
    Attributes           As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す定数の宣言
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す定数宣言
#define PRINTER_ALL_ACCESS &HF000C
#define PRINTER_ATTRIBUTE_NETWORK &H10 'ネットワーク
#define PRINTER_ATTRIBUTE_LOCAL &H40 'ローカル
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙
#define MAX_DEVICENAME 64

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' プリンタの詳細な情報を取得
Declare Function Api_GetPrinter Lib "winspool.drv" Alias "GetPrinterA" (ByVal
hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeeded&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

' メモリの指定領域をコピー
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal length&)

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared PrinterDevName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
```

```

Sub MainForm_Start()

    'プリンタ名を指定
    PrinterDevName = "KONICA MINOLTA magicolor 2400W"
    Text1.SetWindowText PrinterDevName
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var Level As Long
    Var Needed As Long
    Var pi5 As PRINTER_INFO_5
    Var Ret As Long

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(PrinterDevName, hPrinter, pd)

    '構造体のレベルを指定
    Level = 5

    'バッファに必要なサイズを取得
    Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Needed)

    'バッファを確保
    Var Buffer(Needed - 1) As Byte

    '詳細なプリンタ情報を取得
    Ret = Api_GetPrinter(hPrinter, Level, Buffer(0), Needed, Needed)

    '取得した詳細なプリンタ情報を構造体へ移動
    CopyMemory pi5, Buffer(0), Len(pi5)

    'ネットワークの場合
    If pi5.Attributes And PRINTER_ATTRIBUTE_NETWORK Then
        Text2.SetWindowText "ネットワークです。"
    End If

    'ローカルの場合
    Else If pi5.Attributes And PRINTER_ATTRIBUTE_LOCAL Then
        Text2.SetWindowText "ローカルです。"
    End If

    'その他の場合
    Else
        Text2.SetWindowText "判定できません。"
    End If

    'プリンタオブジェクトをクローズ
    Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタのプロパティダイアログを表示

接続されている 既知のプリンタ 名を明示的に指定してプロパティダイアログを表示します。

OpenPrinter プリンタオブジェクトをオープン

PrinterProperties プリンタ設定ダイアログ表示

ClosePrinter プリンタオブジェクトを閉じる



```
'=====
'= プリンタのプロパティダイアログを表示
'= プリンタ名を明示的に指定した場合
'= (PrinterProperties2.bas)
'=====
#include "Windows.bi"

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' プリンタ設定ダイアログ表示
Declare Function Api_PrinterProperties& Lib "winspool.drv" Alias "PrinterProperties"
(ByVal hWnd&, ByVal hPrinter&)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

Var Shared Edit1 As Object
Var SHared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'= DevName:既知のプリンタ名
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var DevName As String
    Var hPrinter As Long
    Var Ret As Long

    DevName = Edit1.GetWindowText

    'プリンタオブジェクトオープン
    Ret = Api_OpenPrinter(DevName, hPrinter, ByVal 0)

    If Ret <> 0 Then

        'プリンタ設定ダイアログ表示
        Ret = Api_PrinterProperties(GethWnd, hPrinter)

        'プリンタオブジェクトを閉じる
        Ret = Api_ClosePrinter(hPrinter)
    End If
End Sub

'=====
'=
'=====
```

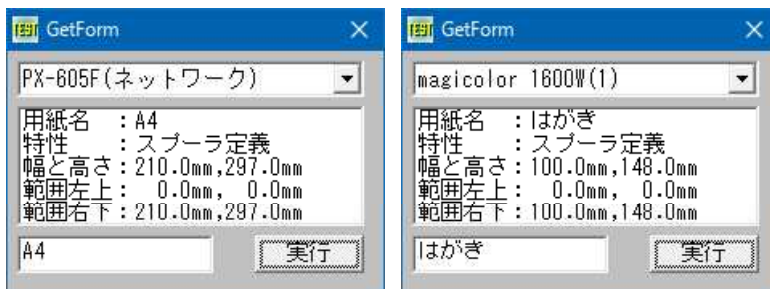
```

While 1
    WaitEvent
Wend
Stop
End

```

プリンタの用紙情報を取得

GetForm プリンタ用の指定されたフォームを記述するデータで FORM_INFO_1 構造体を初期化
EnumPrinters プリンタの詳細な情報を取得
OpenPrinter プリンタオブジェクトをオープン
MoveMemory メモリの指定領域をコピー
lstrcpy 文字列をコピーする
ClosePrinter プリンタオブジェクトを閉じる



```

'=====
'= プリンタの用紙情報を取得
'= (GetForm.bas)
'=====
#include "Windows.bi"
Type PRINTER_DEFAULTS
    pDataType As Long
    pDevMode As Long
    DesiredAccess As Long
End Type

Type SIZE
    cx As Long
    cy As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Type FORM_INFO_1
    Flags As Long
    pName As Long
    Size As SIZE
    ImageableArea As RECT
End Type

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す

```

```

#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define PRINTER_ENUM_NAME &H8 'Nameで指定されたプリンタを列挙
#define MAX_DEVICENAME 64 '
#define FORM_USER &H0 '
#define FORM_BUILTIN &H1 '
#define FORM_PRINTER &H2 '

' プリンタ用の指定されたフォームを記述するデータで FORM_INFO_1 構造体を初期化
Declare Function Api_GetForm& Lib "winspool.drv" Alias "GetFormA" (ByVal hPrinter&,
ByVal pFormName$, ByVal Level&, pForm As Any, ByVal cbBuf&, pcbNeeded&)

' プリンタの詳細な情報を取得
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&,
pcReturned&)

' プリンタオブジェクトをオープン
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)

' メモリの指定領域をコピー
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
ByVal Length&)

' 文字列をコピーする
Declare Function Api_lstrcpy& Lib "Kernel32" Alias "lstrcpy" (lpszString1 As Any,
lpszString2 As Any)

' プリンタオブジェクトを閉じる
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)

Var Shared Edit1 As Object
Var Shared List1 As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
List1.SetWindowSize 216, 74
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var PrinterServer As String
    Var PrinterName As String
    Var PortName As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var cnt As Long
    Var Ret As Long

    PrinterServer = ""
    Level = 5

    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrinterServer, Level, Chr$(0), 0, Needed,
Returned)
    If Needed = 0 Then End

    Var Buffer(Needed - 1) As Byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrinterServer, Level, Buffer(0), Needed,
Needed, Returned)

    Var pi5(Returned - 1) As PRINTER_INFO_5

```

```

For cnt = 0 To Returned - 1
    MoveMemory pi5(cnt), Buffer(cnt * Len(pi5(cnt))), Len(pi5(cnt))

    'プリンタ名取得
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    MoveMemory PrinterName, ByVal pi5(cnt).pPrinterName, Len(PrinterName)
    PrinterName = Left$(PrinterName, InStr(PrinterName, Chr$(0)) - 1)
    Comb1.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var DeviceName As String
    Var pd As PRINTER_DEFAULTS
    Var hPrinter As Long
    Var InfoName As String
    Var Level As Long
    Var Needed As Long
    Var fil As FORM_INFO_1
    Var strFormName As String * 64
    Var txt As String
    Var Ret As Long

    'リストボックスを初期化
    List1.Resetcontent

    'プリンタ名を指定
    DeviceName = Comb1.GetWindowText

    'プリンタアクセス権を指定
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    'プリンタのオブジェクトハンドルを取得
    Ret = Api_OpenPrinter(DeviceName, hPrinter, pd)

    '用紙名を指定
    InfoName = Edit1.GetWindowText

    '構造体のレベルを指定
    Level = 1

    'バッファに必要なサイズを取得
    Ret = Api_GetForm(hPrinter, InfoName, Level, ByVal 0, 0, Needed)

    'フォーム情報が取得できないときは
    If Needed <= 0 Then
        List1.AddString "取得できません。"

        '取得できないときの終了処理へ分岐
        GoTo *ClosePrt
    End If

    'バッファを確保
    Var Buffer(Needed - 1) As Byte

    '用紙情報を取得
    Ret = Api_GetForm(hPrinter, InfoName, Level, Buffer(0), Needed, Needed)

    '取得した用紙情報を構造体へ移動
    MoveMemory fil, Buffer(0), Len(fil)

    '用紙情報を列挙(用紙名を複写)
    Ret = Api_lstrcpy(strFormName, ByVal fil.pName)

    '用紙名を表示
    List1.AddString "用紙名  :" & Left$(strFormName, InStr(strFormName, Chr$(0)) - 1)

```

```

'特性を表示
Select Case fil.Flags

    'ユーザー定義のとき
    Case FORM_USER
        txt = "特性    :ユーザー定義 "

    'スプーラ定義のとき
    Case FORM_BUILTIN
        txt = "特性    :スプーラ定義 "

    'プリンタ定義のとき
    Case FORM_PRINTER
        txt = "特性    :プリンタ定義 "

    'その他のとき
    Case Else
        txt = "特性    :不明"
End Select
List1.AddString txt

'幅と高さを表示
List1.AddString "幅と高さ:" & Format$(fil.Size.cx / 1000, "###.##mm") & ", " &
Format$(fil.Size.cy / 1000, "###.##mm")

'範囲左上を表示
List1.AddString "範囲左上:" & Format$(fil.ImageableArea.left / 1000, "###.##mm") & ", " &
Format$(fil.ImageableArea.top / 1000, "###.##mm")

'範囲右下を表示
List1.AddString "範囲右下:" & Format$(fil.ImageableArea.right / 1000, "###.##mm") & ", " &
Format$(fil.ImageableArea.bottom / 1000, "###.##mm")

*ClosePrt

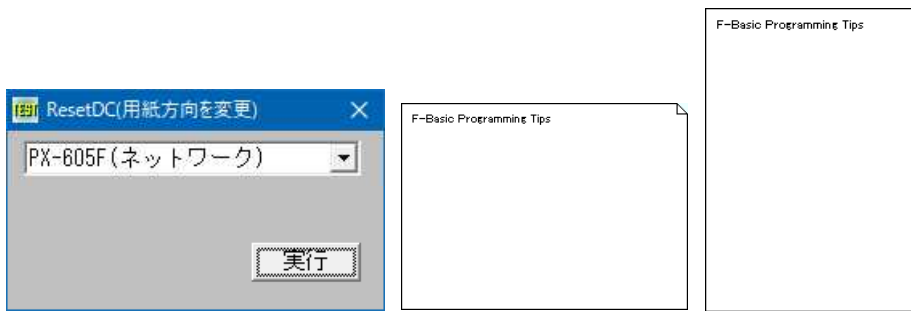
'プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタの用紙方向を変える

OpenPrinter プリンタオブジェクトをオープン
ClosePrinter プリンタオブジェクトを閉じる
DocumentProperties 特定のプリンタのプリンタ初期化情報を取得または変更
ResetDC 指定された情報を使って、指定されたプリンタやプロッタのデバイスコンテキストを更新
CreateDC 指定されたデバイスのデバイスコンテキストを、指定された名前で作成
DeleteDC 指定されたデバイスコンテキストを削除
TextOut 文字を描画
StartDoc 印刷ジョブを開始
StartPage プリンタドライバがデータを受け取る準備をさせる
EndDoc 印刷ジョブを終了
EndPage 1ページ書き込みの終了を通知
CopyMemory ある位置から別の位置にメモリブロックを移動
EnumPrinters 使用可能なプリンタ・プリントサーバーなどを列挙



```
'=====
'= プリンタの用紙方向を変える
'= (ResetDC.bas)
'=====

#include "Windows.bi"

#define CCHDEVICENAME 32
#define CCHFORMNAME 32
#define DM_MODIFY 8
#define DM_IN_BUFFER 8

#define DM_COPY 2
#define DM_OUT_BUFFER 2

#define DMORIENT_PORTRAIT 1
#define DMORIENT_LANDSCAPE 2

#define DMBIN_LOWER 2
#define DMBIN_UPPER 1

#define DM_ORIENTATION &H1
#define DM_DEFAULTSOURCE &H200

#define PRINTER_ENUM_DEFAULT &H1
#define PRINTER_ENUM_LOCAL &H2
#define PRINTER_ENUM_NAME &H8
#define PRINTER_ENUM_SHARED &H20
#define MAX_DEVICENAME 64

Type PRINTER_INFO_5
    pPrinterName As Long
    pPortName As Long
    Attributes As Long
    DeviceNotSelectedTimeOut As Long
    TransmissionRetryTimeOut As Long
End Type

Type DOCINFO
    cbSize As Long
    lpszDocName As Long
    lpszOutput As Long
End Type

Type DEVMODE
    dmDeviceName (CCHDEVICENAME - 1) As Byte
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
    dmOrientation As Integer

    dmPaperSize As Integer
    dmPaperLength As Integer

    dmPaperWidth As Integer
```

'デバイス名の長さを示す定数
'フォーム名の長さを示す定数

'pDevModeInputのdmFieldsで指定した設定値をダイアログの初期値に使う (DM_MODIFY)

'pDevModeOutputに現在のプリンタの設定値をコピー (DM_COPY)

'用紙を縦置
'用紙を横置

'下用紙トレイ
'上用紙フィーダ

'用紙の向きを指定

'デフォルトのプリンタに関する情報を列挙
'Nameの設定を無視して、ローカルプリンタを列挙
'Nameで指定されたプリンタを列挙
'共有属性を持つプリンタを列挙

'ドライバがサポートするデバイス名
'構造体の基準になった初期化データ仕様のバージョン番号
'プリンタドライバのバージョン番号
'この構造体のサイズ (バイト単位)
'この構造体に続くドライバ データのバイト数
'
'DMORIENT_PORTRAIT (縦置き)、
DMORIENT_LANDSCAPE (横置き)
'用紙サイズ
'dmPaperSizeメンバで指定した用紙の長さをオーバーライド
'dmPaperSizeメンバで指定した用紙の幅をオーバーライド

dmScale	As Integer	'印刷出力をスケールリングするときの、スケールリング係数
dmCopies	As Integer	'デバイスが複数の部数に対応する場合、印刷する部数
dmDefaultSource	As Integer	'予約済み(0)
dmPrintQuality	As Integer	'プリンタの解像度(ドット/インチ)
dmColor	As Integer	'カラープリンタの場合(DMCOLOR_COLOR・DMCOLOR_MONOCHROME)
dmDuplex	As Integer	'両面印刷が可能なプリンタ(DMDUP_SIMPLEX・DMDUP_HORIZONTAL・DMDUP_VERTICAL)
dmYResolution	As Integer	'プリンタのy方向の解像度(ドット/インチ)
dmTTOption	As Integer	'TrueTypeフォントの印刷方法
dmCollate	As Integer	'複数部数を印刷するときにページ順にそろえるかどうか
dmFormName(CCHFORMNAME - 1)	As Byte	'フォーム名を指定
dmUnusedPadding	As Integer	'使用しない
dmBitsPerPixel	As Integer	'ディスプレイ デバイスの解像度をピクセルあたりのビット数で指定
dmPelsWidth	As Long	'可視のデバイスの表面の幅をピクセル単位で指定
dmPelsHeight	As Long	'可視のデバイスの表面の高さをピクセル単位で指定
dmDisplayFlags	As Long	'デバイスのディスプレイ モードを指定
dmDisplayFrequency	As Long	'ディスプレイデバイスのリフレッシュレート(垂直同期周波数)を1秒当たりのサイクル数(Hz)で指定
dmICMMethod	As Long	'非ICMアプリケーションの場合に、ICMが使用可能かどうかを指定
dmICMIntent	As Long	'カラーマッチング方法のデフォルトを指定
dmMediaType	As Long	'印刷メディアのタイプを指定
dmDitherType	As Long	'ディザリング方法を指定
dmReserved1	As Long	'予約済み(0)
dmReserved2	As Long	'予約済み(0)
dmPanningWidth	As Long	'NT系(0)
dmPanningHeight	As Long	'NT系(0)

End Type

' プリンタオブジェクトをオープン

```
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, ByVal pDefault&)
```

' プリンタオブジェクトを閉じる

```
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal hPrinter&)
```

' 特定のプリンタのプリンタ初期化情報を取得または変更

```
Declare Function Api_DocumentProperties& Lib "winspool.drv" Alias "DocumentPropertiesA" (ByVal hWnd&, ByVal hPrinter&, ByVal pDeviceName$, dmOutput As Any, dmInput As Any, ByVal fMode&)
```

' 指定された情報を使って、指定されたプリンタやプロッタのデバイスコンテキストを更新

```
Declare Function Api_ResetDC& Lib "gdi32" Alias "ResetDCA" (ByVal hDC&, lpInitData As Any)
```

' 指定されたデバイスのデバイスコンテキストを、指定された名前で作成

```
Declare Function Api_CreateDC& Lib "gdi32" Alias "CreateDCA" (ByVal lpDriverName$, ByVal lpDeviceName$, ByVal lpOutput&, ByVal lpInitData&)
```

' 指定されたデバイスコンテキストを削除

```
Declare Function Api_DeleteDC& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)
```

' 文字を描画

```
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&, ByVal nYStart&, ByVal lpString$, ByVal cbString&)
```

' 印刷ジョブを開始

```
Declare Function Api_StartDoc& Lib "gdi32" Alias "StartDocA" (ByVal hDC&, lpdi As DOCINFO)
```

' プリントドライバがデータを受け取る準備をさせる

```
Declare Function Api_StartPage& Lib "gdi32" Alias "StartPage" (ByVal hDC&)
```

' 印刷ジョブを終了

```
Declare Function Api_EndDoc& Lib "gdi32" Alias "EndDoc" (ByVal hDC&)
```

' 1ページ書き込みの終了を通知

```
Declare Function Api_EndPage& Lib "gdi32" Alias "EndPage" (ByVal hDC&)
```

' ある位置から別の位置にメモリブロックを移動する関数の宣言

```
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length&)
```

' 使用可能なプリンタ・プリントサーバーなどを列挙する

```
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cbBuf&, pcbNeeded&, pcReturned&)
```

```
Var Shared Comb1 As Object
```

```
Var Shared Button1 As Object
```

```
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
Var Shared PrinterName As String
```

```
'=====
'=
'
```

```
Declare Function ResetPrinterDC(PrinterName As String, hPrtDc As Long, PaperSource As Long, PaperOrient As Long) As Integer
```

```
Function ResetPrinterDC(PrinterName As String, hPrtDc As Long, PaperSource As Long, PaperOrient As Long) As Integer
```

```
    Var nSize As Long
```

```
    Var dm As DEVMODE
```

```
    Var hPrinter As Long
```

```
    Var Ret As Long
```

```
    If Api_OpenPrinter(PrinterName, hPrinter, 0) Then
```

```
        nSize = Api_DocumentProperties(GethWnd, hPrinter, PrinterName, 0, 0, 0)
```

```
        Var aDevMode(nSize)
```

```
        nSize = Api_DocumentProperties(GethWnd, hPrinter, PrinterName, aDevMode(0), 0, DM_OUT_BUFFER)
```

```
        CopyMemory dm, aDevMode(0), Len(dm)
```

```
        dm.dmDefaultSource = PaperSource
```

```
        dm.dmOrientation = PaperOrient
```

```
        dm.dmFields = dm.dmFields Or DM_DEFAULTSOURCE Or DM_ORIENTATION
```

```
        CopyMemory aDevMode(0), dm, Len(dm)
```

```
        nSize = Api_DocumentProperties(GethWnd, hPrinter, PrinterName, aDevMode(0), aDevMode(0), DM_IN_BUFFER Or DM_OUT_BUFFER)
```

```
        nSize = Api_ResetDC(hPrtDc, aDevMode(0))
```

```
        Ret = Api_ClosePrinter(hPrinter)
```

```
        ResetPrinterDC = -1
```

```
    Else
```

```
        ResetPrinterDC = 0
```

```
    End If
```

```
End Function
```

```
'=====
'=
'
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var PrintServer As String
```

```
    Var Needed As Long
```

```
    Var Returned As Long
```

```
    Var Level As Long
```

```
    Var CT As Long
```

```
    Var Ret As Long
```

```

'ローカルプリンタから検索
PrintServer = ""

'PRINTER_INFO_5構造体を受け取る
Level = 5

'バッファに必要なバイト数を調べる
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Chr$(0), 0, Needed,
Returned)
If Needed = 0 Then End

'全プリンタ情報を得る
Var Buffer(Needed - 1) As byte
Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrintServer, Level, Buffer(0), Needed,
Needed, Returned)

'構造体リストの準備
Var PI_5(Returned - 1) As PRINTER_INFO_5

For CT = 0 To Returned - 1

    'バッファから構造体1つ分を抜き取る
    CopyMemory PI_5(CT), Buffer(CT * Len(PI_5(CT))), Len(PI_5(CT))

    'プリンタ名を得る
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    CopyMemory PrinterName, ByVal PI_5(CT).pPrinterName, Len(PrinterName)
    PrinterName = KLeft$(PrinterName, KInStr(1, PrinterName, Chr$(0)) - 1)

    Combol.AddString PrinterName
Next
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var di As DOCINFO
    Var hPrtDc As Long
    Var txt As String
    Var Ret As Long

    PrinterName = Combol.GetText(Combol.GetCursel)
    txt = "F-Basic Programming Tips"

    hPrtDc = Api_CreateDC("winspool", PrinterName, 0, 0)
    di.cbSize = Len(di)
    di.lpszDocName = StrAdr("F-Basic Test Document" & Chr$(0))

    '横置きで印刷
    Ret = ResetPrinterDC(PrinterName, hPrtDc, DMBIN_UPPER, DMORIENT_LANDSCAPE)
    Ret = Api_StartDoc(hPrtDc, di)
    Ret = Api_StartPage(hPrtDc)
    Ret = Api_TextOut(hPrtDc, 100, 100, txt, Len(txt))
    Ret = Api_EndPage(hPrtDc)

    '縦置きで印刷
    Ret = ResetPrinterDC(PrinterName, hPrtDc, DMBIN_UPPER, DMORIENT_PORTRAIT)
    Ret = Api_StartPage(hPrtDc)
    Ret = Api_TextOut(hPrtDc, 100, 100, txt, Len(txt))
    Ret = Api_EndPage(hPrtDc)
    Ret = Api_EndDoc(hPrtDc)

    Ret = Api_DeleteDC(hPrtDc)
End Sub

'=====
'=
'=====

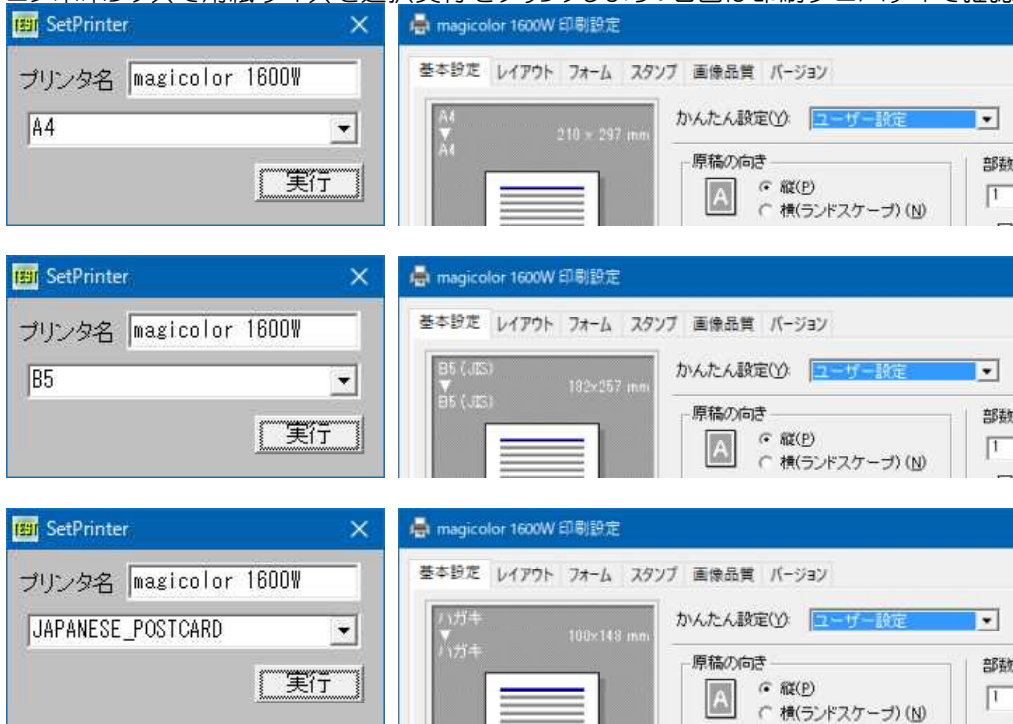
```

```
While 1
    WaitEvent
Wend
Stop
End
```

プリンタの用紙を設定

OpenPrinter プリンタオブジェクトをオープン
 GetPrinter プリンタの詳細な情報を取得
 MoveMemory メモリの指定領域をコピー
 SetPrinter プリンタの詳細な情報を設定
 ClosePrinter プリンタオブジェクトを閉じる

コンボボックスで用紙サイズを選択実行をクリックします。右図は印刷プロパティで確認しています。



```
'=====
'= プリンタの用紙を設定
'= (SetPrinter3.bas)
'=====

#include "Windows.bi"

#define STANDARD_RIGHTS_REQUIRED &HF0000 '標準的な権利を要求することを示す定数
#define PRINTER_ACCESS_ADMINISTER &H4 'プリンタアクセス権の管理者権限を示す
#define PRINTER_ACCESS_USE &H8 'プリンタアクセス権のユーザー権限を示す
#define PRINTER_ALL_ACCESS (&HF0000 or &H4 or &H8) '
#define CCHDEVICENAME 32 'デバイス名の長さを示す定数
#define CCHFORMNAME 32 'フォーム名の長さを示す定数
#define DM_PAPERSIZE &H2 '用紙が有効であることを示す

#define DMPAPER_LETTER 1 'レター、81/2x11in
#define DMPAPER_LETTERSMALL 2 'レターズモール、81/2x11in
#define DMPAPER_A3 8 'A3、297x420mm
#define DMPAPER_A4 9 'A4、210x297mm
#define DMPAPER_A4SMALL 10 'A4Small、210x297mm
#define DMPAPER_A5 11 'A5、148x210mm
#define DMPAPER_B4 12 'B4、250x354mm
#define DMPAPER_B5 13 'B5、182x257mm
#define DMPAPER_NOTE 18 'ノート、81/2x11in
#define DMPAPER_JAPANESE_POSTCARD 43 'はがき100x148
```

```
Type PRINTER_DEFAULTS
    pDatatype      As Long
    pDevMode        As Long
    DesiredAccess   As Long
End Type
```

```
Type DEVMODE
    dmDeviceName (CCHDEVICENAME - 1) As Byte
    dmSpecVersion      As Integer
    dmDriverVersion     As Integer
    dmSize              As Integer
    dmDriverExtra       As Integer
    dmFields            As Long
    dmOrientation       As Integer
    dmPaperSize         As Integer
    dmPaperLength       As Integer
    dmPaperWidth        As Integer
    dmScale             As Integer
    dmCopies            As Integer
    dmDefaultSource     As Integer
    dmPrintQuality      As Integer
    dmColor             As Integer
    dmDuplex            As Integer
    dmYResolution       As Integer
    dmTTOption          As Integer
    dmCollate           As Integer
    dmFormName (CCHFORMNAME - 1) As Byte
    dmLogPixels         As Integer
    dmBitsPerPel        As Long
    dmPelsWidth         As Long
    dmPelsHeight        As Long
    dmDisplayFlags      As Long
    dmDisplayFrequency  As Long
End Type
```

```
Type PRINTER_INFO_2
    pServerName      As Long
    pPrinterName     As Long
    pShareName       As Long
    pPortName        As Long
    pDriverName      As Long
    pComment         As Long
    pLocation        As Long
    pDevMode         As Long
    pSepFile         As Long
    pPrintProcessor  As Long
    pDatatype        As Long
    pParameters      As Long
    pSecurityDescriptor As Long
    Attributes       As Long
    Priority          As Long
    DefaultPriority   As Long
    StartTime        As Long
    UntilTime        As Long
    Status           As Long
    cJobs            As Long
    AveragePPM       As Long
End Type
```

・ プリントオブジェクトをオープン

```
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName$, phPrinter&, pDefault As Any)
```

・ プリンタの詳細な情報を取得

```
Declare Function Api_GetPrinter& Lib "winspool.drv" Alias "GetPrinterA" (ByVal hPrinter&, ByVal Level&, pPrinter As Any, ByVal cbBuf&, pcbNeed&)
```

・ メモリの指定領域をコピー

```
Declare Sub MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As Any,
```

```
ByVal Length&)
```

```
' プリンタの詳細な情報を設定
```

```
Declare Function Api_SetPrinter& Lib "winspool.drv" Alias "SetPrinterA" (ByVal  
hPrinter&, ByVal Level&, pPrinter As Any, ByVal Command&)
```

```
' プリンタオブジェクトを閉じる
```

```
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal  
hPrinter&)
```

```
Var Shared Text1 As Object  
Var Shared Edit1 As Object  
Var Shared Comb1 As Object  
Var Shared Button1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14  
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14  
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
Var Shared ps(9) As Long
```

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()  
Sub MainForm_Start()
```

```
'ここではプリンタ名を明示的に指定
```

```
Edit1.SetWindowText "magicolor 2400W"  
Comb1.AddString "LETTER" : ps(0) = DMPAPER_LETTER  
Comb1.AddString "LETTERSMAIL" : ps(1) = DMPAPER_LETTERSMAIL  
Comb1.AddString "A3" : ps(2) = DMPAPER_A3  
Comb1.AddString "A4" : ps(3) = DMPAPER_A4  
Comb1.AddString "A4SMALL" : ps(4) = DMPAPER_A4SMALL  
Comb1.AddString "A5" : ps(5) = DMPAPER_A5  
Comb1.AddString "B4" : ps(6) = DMPAPER_B4  
Comb1.AddString "B5" : ps(7) = DMPAPER_B5  
Comb1.AddString "NOTE" : ps(8) = DMPAPER_NOTE  
Comb1.AddString "JAPANESE_POSTCARD" : ps(9) = DMPAPER_JAPANESE_POSTCARD
```

```
End Sub
```

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub Button1_on edec1 ()  
Sub Button1_on()
```

```
Var DeviceName As String  
Var pd As PRINTER_DEFAULTS  
Var pi2 As PRINTER_INFO_2  
Var dm As DEVMODE  
Var hPrinter As Long  
Var Level As Long  
Var Needed As Long  
Var Ret As Long
```

```
DeviceName = Edit1.GetWindowText
```

```
'プリンタアクセス権を指定
```

```
pd.DesiredAccess = PRINTER_ALL_ACCESS
```

```
'プリンタのオブジェクトハンドルを取得
```

```
Ret = Api_OpenPrinter(DeviceName, hPrinter, pd)
```

```
'構造体のレベルを指定
```

```
Level = 2
```

```
'バッファに必要なサイズを取得
```

```
Ret = Api_GetPrinter(hPrinter, Level, ByVal 0, 0, Needed)
```



```

'バッファを確保
Var bytPrinterInfo2Buffer (Needed - 1) As Byte

'詳細なプリンタ情報を取得
Ret = Api_GetPrinter(hPrinter, Level, bytPrinterInfo2Buffer(0), Needed, Needed)

'取得した詳細なプリンタ情報を構造体へ移動
MoveMemory pi2, bytPrinterInfo2Buffer(0), Len(pi2)

'取得した詳細なプリンタデバイス情報を構造体へ移動
MoveMemory dm, ByVal pi2.pDevMode, Len(dm)

'用紙の指定可能
If dm.dmFields And DM_PAPERSIZE Then

    '用紙を指定
    dm.dmPaperSize = ps(Combo1.GetCursel)

    '構造体を取得した詳細なプリンタデバイス情報へ移動
    MoveMemory ByVal pi2.pDevMode, dm, Len(dm)

    '詳細なプリンタ情報を書き戻し
    Ret = Api_SetPrinter(hPrinter, Level, pi2, 0)
End If

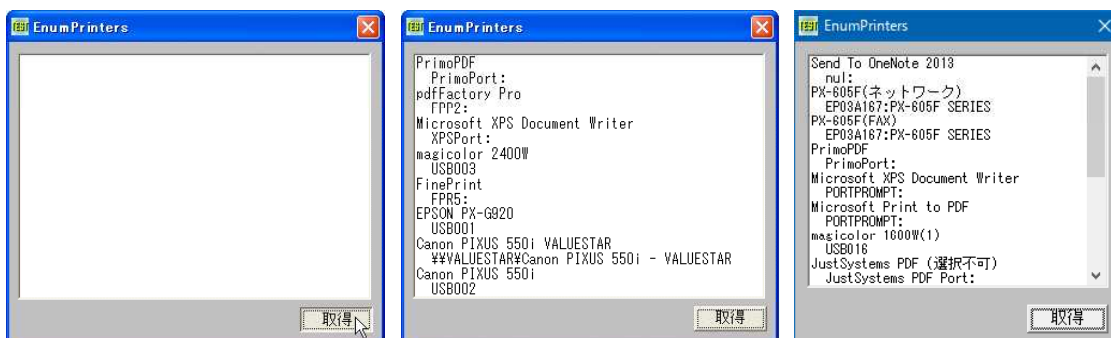
'プリンタオブジェクトをクローズ
Ret = Api_ClosePrinter(hPrinter)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プリンタ名とポート名を取得

プリンタ名とポート名を取得します。
EnumPrinters プリンタの詳細な情報を取得
CopyMemory ある位置から別の位置にメモリブロックを移動
PrinterProperties プリンタ設定ダイアログ表示
SetDefaultPrinter 通常使うプリンタの設定
OpenPrinter プリンタオブジェクトをオープン
ClosePrinter プリンタオブジェクトを閉じる



```

'=====
'= プリンタ名とポート名を取得
'= (EnumPrinters.bas)
'=====
#include "Windows.bi"

```


' 詳細なプリンタ情報を定義する構造体

```
Type PRINTER_INFO_5
    pPrinterName           As Long
    pPortName              As Long
    Attributes              As Long
    DeviceNotSelectedTimeout As Long
    TransmissionRetryTimeout As Long
End Type
```

```
#define PRINTER_ENUM_NAME &H8           'Nameで指定されたプリンタを列挙
#define MAX_DEVICENAME 64
```

' プリンタの詳細な情報を取得

```
Declare Function Api_EnumPrinters& Lib "winspool.drv" Alias "EnumPrintersA" (ByVal
Flags&, ByVal Name$, ByVal Level&, pPrinterEnum As Any, ByVal cdBuf&, pcbNeeded&,
pcReturned&)
```

' ある位置から別の位置にメモリブロックを移動

```
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)
```

' プリンタ設定ダイアログ表示

```
Declare Function Api_PrinterProperties& Lib "winspool.drv" Alias "PrinterProperties"
(ByVal hWnd&, ByVal hPrinter&)
```

' 通常使うプリンタの設定

```
Declare Function Api_SetDefaultPrinter& Lib "winspool.drv" Alias "SetDefaultPrinterA"
(ByVal pszPrinter$)
```

' プリンタオブジェクトをオープン

```
Declare Function Api_OpenPrinter& Lib "winspool.drv" Alias "OpenPrinterA" (ByVal
pPrinterName$, phPrinter&, pDefault As Any)
```

' プリンタオブジェクトを閉じる

```
Declare Function Api_ClosePrinter& Lib "winspool.drv" Alias "ClosePrinter" (ByVal
hPrinter&)
```

```
Var Shared List1 As Object
Var Shared Button1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
Sub Button1_on()
```

```
    Var PrinterServer As String
    Var PrinterName As String
    Var PortName As String
    Var Needed As Long
    Var Returned As Long
    Var Level As Long
    Var i As Long
    Var Ret As Long
```

```
    PrinterServer = ""
    Level = 5
```

```
    List1.ResetContent
```

```
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrinterServer, Level, Chr$(0), 0, Needed,
Returned)
    If Needed = 0 Then End
```

```
    Var Buffer(Needed - 1) As Byte
    Ret = Api_EnumPrinters(PRINTER_ENUM_NAME, PrinterServer, Level, Buffer(0), Needed,
Needed, Returned)
```

```

Var pi5 (Returned - 1) As PRINTER_INFO_5

For i = 0 To Returned - 1
    CopyMemory pi5(i), Buffer(i * Len(pi5(i))), Len(pi5(i))

    'プリンタ名取得
    PrinterName = String$(MAX_DEVICENAME, Chr$(0))
    CopyMemory PrinterName, ByVal pi5(i).pPrinterName, Len(PrinterName)
    PrinterName = Left$(PrinterName, InStr(PrinterName, Chr$(0)) - 1)
    List1.AddString PrinterName

    'ポート名取得
    PortName = String$(MAX_DEVICENAME, Chr$(0))
    CopyMemory PortName, ByVal pi5(i).pPortName, Len(PortName)
    PortName = Left$(PortName, InStr(PortName, Chr$(0)) - 1)

    List1.AddString " " & PortName
Next
End Sub

'=====
'=
'=====
Declare Sub List1_Change edec1 ()
Sub List1_Change ()
    Var hPrinter As Long
    Var Ret As Long

    PrinterName$ = List1.GetText (List1.GetCursel)

    Ret = Api_SetDefaultPrinter (PrinterName$)
    Ret = Api_OpenPrinter (PrinterName$, hPrinter, ByVal 0)
    Ret = Api_PrinterProperties (GethWnd, hPrinter)
    Ret = Api_ClosePrinter (hPrinter)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

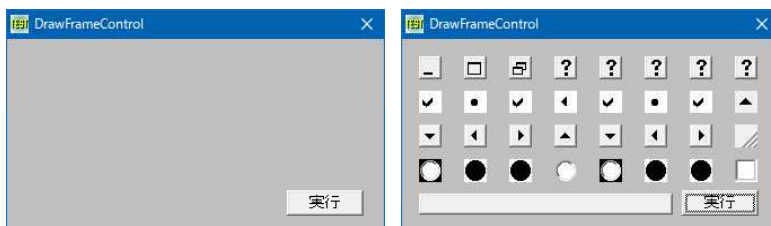
フレームコントロールを描画(1)

指定されたタイプとスタイルを備える、ボタンやスクロールバーなどのフレームコントロールを描画します。

DrawFrameControl フレームコントロールを描画

OffsetRect 矩形領域の補正

un1 (uType) と un2 (uState) の組み合わせでボタンやスクロールバーなどのフレームコントロールを描画します。



```

'=====
'= フレームコントロールを描画
'= (DrawFrameControl.bas)
'=====
#include "Windows.bi"

```

```

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

' 指定されたタイプとスタイルを備える、ボタンやスクロールバーなどのフレームコントロールを描画
Declare Function Api_DrawFrameControl& Lib "user32" Alias "DrawFrameControl" (ByVal
hDC&, lpRect As RECT, ByVal un1&, ByVal un2&)

' 矩形領域の補正
Declare Function Api_OffsetRect& Lib "user32" Alias "OffsetRect" (lpRect As RECT, ByVal
x&, ByVal y&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define DR_DRAWWIDTH 20
#define DR_DRAWHEIGHT 20

'uType
#define DFC_BUTTON 4                'ボタン
#define DFC_CAPTION 1              'タイトルバーのボタン
#define DFC_MENU 2                 'メニュー
#define DFC_POPUPMENU 5            'ポップアップメニュー
#define DFC_SCROLL 3               'スクロールバー

'uState
#define DFCS_ADJUSTRECT &H2000     '境界を調節してボタンの周囲の辺を除外
#define DFCS_BUTTON3STATE 8        '3ステートボタン
#define DFCS_BUTTONCHECK 0         'チェックボックス
#define DFCS_BUTTONPUSH &H10       'ボタン
#define DFCS_BUTTONRADIO 4         'ラジオボタン
#define DFCS_BUTTONRADIOIMAGE 1    'ラジオボタンのイメージ
#define DFCS_BUTTONRADIOMASK 2     'ラジオボタンのマスク
#define DFCS_CAPTIONCLOSE 0        '閉じるボタン
#define DFCS_CAPTIONHELP 4         'ヘルプボタン
#define DFCS_CAPTIONMAX 2          '最大化ボタン
#define DFCS_CAPTIONMIN 1          '最小化ボタン
#define DFCS_CAPTIONRESTORE 3      '元に戻すボタン
#define DFCS_CHECKED &H400         'チェックされた状態
#define DFCS_FLAT &H4000           '平らなデザイン
#define DFCS_HOT &H1000            '
#define DFCS_INACTIVE &H100        '使用不可
#define DFCS_MENUARROW 0           'サブメニュー矢印
#define DFCS_MENUARROWRIGHT 4      '矢印
#define DFCS_MENUBULLET 2          'オプションマーク
#define DFCS_MENUCHECK 1           'チェックマーク
#define DFCS_MONO &H8000           'モノクロの境界
#define DFCS_PUSHED &H200          '押された状態
#define DFCS_SCROLLCOMBOBOX 5      'コンボボックスタイプ
#define DFCS_SCROLLDOWN 1          '下
#define DFCS_SCROLLLEFT 2          '左
#define DFCS_SCROLLRIGHT 3         '右
#define DFCS_SCROLLSIZEGRIP 8      'サイズ変更グリップ
#define DFCS_SCROLLUP 0            '上
#define DFCS_TRANSPARENT &H800     '

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared hDC As Long

```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    hDC = Api_GetDC (GethWnd)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var rc As RECT
    Var i As Long
    Var j As Long
    Var Ret As Long

    For i = 1 To 4
        rc.Top = 30 * i - 15
        rc.Bottom = rc.Top + DR_DRAWHEIGHT
        For j = 1 To 8
            rc.Left = 40 * j - 25
            rc.Right = rc.Left + DR_DRAWWIDTH
            Ret = Api_DrawFrameControl (hDC, rc, i, j)
        Next
    Next

    rc.Left = 15
    rc.Right = 240
    Ret = Api_OffsetRect (rc, 0, 30)
    Ret = Api_DrawFrameControl (hDC, rc, 0, DFCS_BUTTON3STATE)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 (Cancel%, Mode%)
Sub MainForm_QueryClose (Cancel%, Mode%)
    Var Ret As Long

    If Cancel% = 0 Then
        Ret = Api_ReleaseDC (GethWnd, hDC)
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

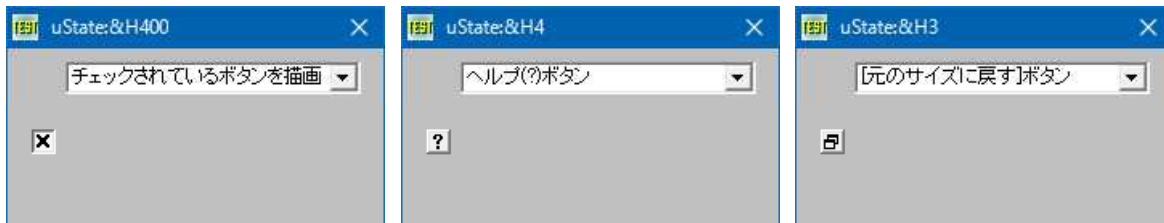
フレームコントロールを描画(II)

指定されたタイプとスタイルを備える、ボタンやスクロールバーなどのフレームコントロールを描画します。

DrawFrameControl フレームコントロールを描画

GetDC ディスプレイデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```
'=====
'= フレームコントロールを描画 (11)
'= (DrawFrameControl2.bas)
'=====
#include "Windows.bi"

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

' 指定されたタイプとスタイルを備える、ボタンやスクロールバーなどのフレームコントロールを描画
Declare Function Api_DrawFrameControl& Lib "user32" Alias "DrawFrameControl" (ByVal
hDC&, lpRect As RECT, ByVal un1&, ByVal un2&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define DFC_CAPTION 1
#define DFCS_CHECKED &H400
#define DFCS_FLAT &H4000
#define DFCS_INACTIVE &H100
#define DFCS_MONO &H8000
#define DFCS_PUSHED &H200
#define DFCS_CAPTIONCLOSE 0
#define DFCS_CAPTIONHELP 4
#define DFCS_CAPTIONMAX 2
#define DFCS_CAPTIONMIN 1
#define DFCS_CAPTIONRESTORE 3

' タイトルバーを描画
' チェックされているボタンを描画
' 平らなデザインのボタンを描画
' 使用不可
' モノクロの境界
' 押された状態
' [閉じる] ボタン
' ヘルプ (?) ボタン
' [最大化] ボタン
' [最小化] ボタン
' [元のサイズに戻す] ボタン

Var Shared Comb1 As Object
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 12

Var Shared VarrView(9) As String
Var Shared VarrValue(9) As Long
Var Shared vShowVal As Long
Var Shared hDC As Long

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var i As Integer

    hDC = Api_GetDC (GethWnd)

    VarrView(0) = "チェックされているボタンを描画"
    VarrView(1) = "平らなデザインのボタンを描画"
    VarrView(2) = "使用不可"
    VarrView(3) = "モノクロの境界"
    VarrView(4) = "押された状態"
    VarrView(5) = "[閉じる] ボタン"
    VarrView(6) = "[最小化] ボタン"
    VarrView(7) = "[最大化] ボタン"
    VarrView(8) = "[元のサイズに戻す] ボタン"
```

```

VarrView(9) = "ヘルプ(?)ボタン"

VarrValue(0) = DFCS_CHECKED
VarrValue(1) = DFCS_FLAT
VarrValue(2) = DFCS_INACTIVE
VarrValue(3) = DFCS_MONO
VarrValue(4) = DFCS_PUSHED
VarrValue(5) = DFCS_CAPTIONCLOSE
VarrValue(6) = DFCS_CAPTIONMIN
VarrValue(7) = DFCS_CAPTIONMAX
VarrValue(8) = DFCS_CAPTIONRESTORE
VarrValue(9) = DFCS_CAPTIONHELP

For i = 0 To 9
    Comb1.AddString VarrView(i)
Next
End Sub

'=====
'=
'=====
Declare Sub Comb1_Change edec1 ()
Sub Comb1_Change()
    Var rc As RECT

    vShowVal = VarrValue(Comb1.GetCursel)
    SetWindowText " uState:&H" & Hex$(vShowVal)

    rc.Left = 15
    rc.Right = 30
    rc.Top = 50
    rc.Bottom = 65

    Ret = Api_DrawFrameControl(hDC, rc, DFC_CAPTION, vShowVal)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

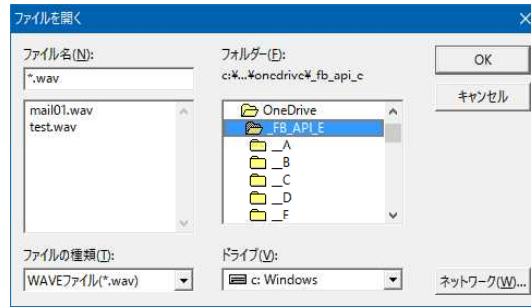
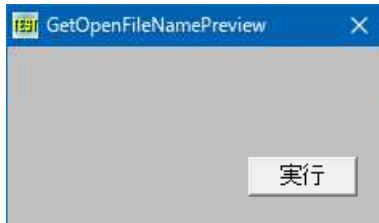
    Ret = Api_ReleaseDC(GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プレビュー機能付き「ファイルを開く」ダイアログを表示

[GetOpenFileNamePreview](#) プレビュー機能付き「ファイルを開く」ダイアログを表示
[GetSaveFileNamePreview](#) プレビュー機能付き「ファイルを保存」ダイアログを表示



```
'=====
'= プレビュー機能付き「ファイルを開く」ダイアログを表示
'= (GetOpenFileNamePreview.bas)
'=====
```

```
#include "Windows.bi"
```

```
#define OFN_EXPLORER &H80000
#define OFN_ALLOWMULTISELECT &H200
#define OFN_CREATEPROMPT &H2000
```

```
#define OFN_FILEMUSTEXIST &H1000
#define OFN_HIDEREADONLY &H4
#define OFN_NOCHANGEDIR &H8
#define OFN_NOREADONLYRETURN &H8000
```

```
#define OFN_NOVALIDATE &H100
#define OFN_OVERWRITEPROMPT &H2
#define OFN_PATHMUSTEXIST &H800
#define OFN_READONLY &H1
#define OFN_SHOWHELP &H10
```

'Explorer風ダイアログ
'複数ファイルを選択可能にする
'指定のファイル名が存在しない時にメッセージボックスを表示
'存在しないファイル名は入力不可
'読み取り専用のチェックボックスを非表示
'他のサブディレクトリから選択不可
'読み込み専用ファイルと書き込み禁止ディレクトリの選択不可
'ファイル名の有効性をチェックしない
'既存のファイル名を指定した時にメッセージを出す
'有効なパスだけをうけつける
'読み取り専用のチェックボックスをチェック
'ヘルプボタンを表示

```
Type OPENFILENAME
```

```
    lStructSize      As Long
    hwndOwner        As Long
```

```
    hInstance        As Long
    lpstrFilter        As Long
```

```
    lpstrCustomFilter As Long
```

```
    nMaxCustFilter    As Long
    nFilterIndex       As Long
    lpstrFile          As Long
    nMaxFile           As Long
    lpstrFileName      As Long
```

```
    nMaxFileName      As Long
    lpstrInitialDir    As Long
    lpstrTitle         As Long
```

```
    Flags             As Long
```

```
    nFileOffset        As Integer
```

```
    nFileExtension     As Integer
```

```
    lpstrDefExt        As Long
    lCustData          As Long
```

```
    lpfnHook           As Long
    lpTemplateName     As Long
```

'構造体のサイズをバイト単位で指定
'ダイアログボックスを所有するウィンドウへのハンドルを指定
'メモリオブジェクトへのハンドルを指定
'任意の数のNULL文字で終わる文字列のペアを保持するバッファへのポインタを指定
'ユーザーによって選択されたファイル フィルター文字列のペアが保存される
'lpstrCustomFilterメンバのバッファ サイズを指定
'フィルターの初期インデックスを指定
'文字列バッファを指定
'lpstrFileメンバのバッファ サイズを指定
'選択されたファイルの名前、および拡張子を保存するための文字列バッファを指定
'lpstrFileName メンバのバッファ サイズを指定
'初期ディレクトリのパスを指定
'ダイアログボックスのタイトルバーに表示する文字列を指定
'ダイアログボックスの初期化フラグを、「OFN_...」の定数を組み合わせて指定
'lpstrFileメンバが示すパス内の、ファイル名の位置がバッファの先頭から何バイト目であるかを示す
'lpstrFileメンバが示すパス内の、拡張子の位置がバッファの先頭から何バイト目であるかを示す
'デフォルト拡張子の文字列を示す
'lpfnHookメンバが示すフック プロシージャへ渡すデータを指定
'フック プロシージャへのポインタを指定
'hInstanceメンバが示すモジュール内にあるダイアログテンプレートを指定

```
End Type
```

```
' プレビュー機能付き「ファイルを開く」ダイアログを表示する
```

```
Declare Function Api_GetOpenFileNamePreview& Lib "msvfw32" Alias  
"GetOpenFileNamePreviewA" (ByRef lpofn As OPENFILENAME)
```

・プレビュー機能付き「ファイルを保存」ダイアログを表示する

```

Declare Function Api_GetSaveFileNamePreview Lib "msvfw32" Alias
"GetSaveFileNamePreviewA" (ByRef lpofn As OPENFILENAME)

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var ofn As OPENFILENAME
    Var Ret As Long

    ofn.lStructSize = Len(ofn)
    ofn.hInstance = GethInst
    ofn.hwndOwner = GethWnd
    ofn.lpstrTitle = StrAdr("ファイルを開く" & Chr$(0))
    ofn.lpstrFilter = StrAdr("WAVEファイル(*.wav)" & Chr$(0) & "*.wav" & Chr$(0) & "MIDIフ
イル(*.mid)" & Chr$(0) & "*.mid" & Chr$(0) & "AVIファイル(*.avi)" & Chr$(0) & "*.avi" &
Chr$(0) & "AVIファイル(*.avi)" & Chr$(0) & " *.*" & Chr$(0))
    ofn.lpstrFile = StrAdr(String$(250, Chr$(0)) & Chr$(0))
    ofn.nMaxFile = 250
    ofn.flags = OFN_PATHMUSTEXIST Or OFN_FILEMUSTEXIST Or OFN_HIDEREADONLY Or
OFN_OVERWRITEPROMPT

    Ret = Api_GetOpenFileNamePreview(ofn)

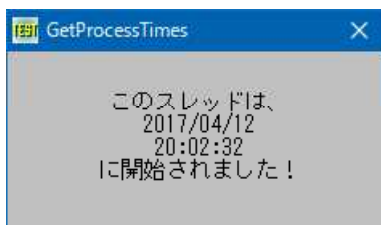
    ofn.lpstrTitle = StrAdr("名前を付けて保存" & Chr$(0))
    Ret = Api_GetSaveFileNamePreview(ofn)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プログラムの開始時間を取得

GetProcessTimes 指定されたプロセスのタイミング情報を取得
FileTimeToLocalFileTime 世界協定時刻形式ファイル時間をローカルファイル時間に変換
FileTimeToSystemTime ファイルタイムをシステムタイムに変換
GetCurrentProcess 現在のプロセスに対応する疑似ハンドルを取得



```

'=====
'= プログラムの開始時間を取得
'= WindowsNT3.1以降
'= (GetProcessTimes.bas)
'=====
#include "Windows.bi"

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

```



```

Type SYSTEMTIME
    wYear          As Integer
    wMonth         As Integer
    wDayOfWeek     As Integer
    wDay           As Integer
    wHour          As Integer
    wMinute        As Integer
    wSecond        As Integer
    wMilliseconds  As Integer
End Type

' 指定されたプロセスのタイミング情報を取得
Declare Function Api_GetProcessTimes& Lib "kernel32" Alias "GetProcessTimes" (ByVal
hProcess&, lpCreationTime As FILETIME, lpExitTime As FILETIME, lpKernelTime As FILETIME,
lpUserTime As FILETIME)

' 世界協定時刻形式ファイル時間をローカルファイル時間に変換
Declare Function Api_FileTimeToLocalFileTime& Lib "kernel32" Alias
"FileTimeToLocalFileTime" (lpFileTime As FILETIME, lpLocalFileTime As FILETIME)

' ファイルタイムをシステムファイルに変換
Declare Function Api_FileTimeToSystemTime& Lib "kernel32" Alias "FileTimeToSystemTime"
(lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME)

' 現在のプロセスに対応する疑似ハンドルを取得
Declare Function Api_GetCurrentProcess& Lib "Kernel32" Alias "GetCurrentProcess" ()

Var Shared Text1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var FT0 As FILETIME
    Var FT1 As FILETIME
    Var ST As SYSTEMTIME
    Var txt As String
    Var Ret As Long

    Ret = Api_GetProcessTimes(Api_GetCurrentProcess, FT1, FT0, FT0, FT0)
    Ret = Api_FileTimeToLocalFileTime(FT1, FT1)
    Ret = Api_FileTimeToSystemTime(FT1, ST)

    txt = txt & "このスレッドは、" & Chr$(13, 10)
    txt = txt & Trim$(Str$(ST.wYear)) & "/" & Right$(Str$(100 + ST.wMonth), 2) & "/" &
Right$(Str$(100 + ST.wDay), 2) & Chr$(13, 10)
    txt = txt & Right$(Str$(100 + ST.wHour), 2) & ":" & Right$(Str$(100 + ST.wMinute), 2) &
": " & Right$(Str$(100 + ST.wSecond), 2) & Chr$(13, 10)
    txt = txt & "に開始されました！"
    Text1.SetWindowText txt
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

プログレスバーの色を変える

プログレスバー の色、および背景色を変えてみます。

CreateWindowEx ウィンドウ(コントロール)を作成
 InitCommonControlsEx 特定のCOMMONCONTROLクラスを登録
 SendMessage ウィンドウにメッセージを送信

ColorRef = RGB(Red, Green, Blue) でプログレスバーの色を変えることができます。
 背景色は、それぞれ255までの数値を入れてください。



※どういう訳か、プログレスバー作成(スムーズ表示)の行で、&HBのところをPBS_SMOOTHにするとエラー[2]になります。

以前は、大丈夫だったはずなのだが???...

ちなみに、[プログレスバーの作成](#)では、大丈夫なのでF-BASICの不具合かなあ...

```
'=====
'= プログレスバーの色を変える
'= (ProgressBar2.bas)
'=====
#include "Windows.bi"
```

```
Type tagINITCOMMONCONTROLSEX
    dwSize      As Long
    dwICC       As Long
End Type
```

'構造体のバイト数
 'ロードするクラスを指定する

' ウィンドウ(コントロール)を作成

```
Declare Function Api_CreateWindowEx& Lib "user32" Alias "CreateWindowExA" (ByVal
ExStyle&, ByVal ClassName$, ByVal WinName$, ByVal Style&, ByVal x&, ByVal y&, ByVal
nWidth&, ByVal nHeight&, ByVal Parent&, ByVal Menu&, ByVal Instance&, ByVal Param&)
```

' コモンコントロールのダイナミックリンクライブラリ(DLL)に含まれている、特定のCOMMONCONTROLクラスを登録

```
Declare Function Api_InitCommonControlsEx& Lib "comctl32" Alias "InitCommonControlsEx"
(lpInitCtrls As tagINITCOMMONCONTROLSEX)
```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, ByVal lParam&)
```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessageP& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)
```

' ウィンドウスタイルの定数

```
#define WS_CHILD &H40000000
#define WS_VISIBLE &H10000000
#define WS_BORDER &H800000
```

'親ウィンドウを持つコントロール(子ウィンドウ)を作成
 'Visibleである
 'フォームの枠線がある

' プログレスバー関係

```
#define WM_USER &H400

#define PBM_SETRANGE &H401
#define PBM_SETPOS &H402
#define PBM_DELTAPOS &H403
#define PBM_SETSTEP &H404
#define PBS_SMOOTH &HB
#define ICC_PROGRESS_CLASS &H20
#define PROGRESS_CLASS "msctls_progress32"
```

'ユーザーが定義できるメッセージの使用領域を表すだけでこれ自体に意味はない
 '(WM_USER + 1) 範囲の設定
 '(WM_USER + 2) 位置の指定
 '(WM_USER + 3) デフォルトの位置
 '(WM_USER + 4) 増分の設定
 '滑らかなプログレスバー
 'プログレスバーコントロール
 'プログレスバーコントロールクラス

```
#define CCM_FIRST &H2000
#define CCM_SETBKCOLOR &H2001
#define PBM_SETBKCOLOR &H2001
```

'背景色の設定(IE3以降)

```

#define PBM_SETBARCOLOR &H409                                'バーの色設定 (IE4以降)

Var Shared Text1 As Object
Var Shared Check1 As Object
Var Shared Group1 As Object
Var Shared Edit(2) As Object
Var Shared Radio(2) As Object
Var Shared Timer1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
For i = 0 To 2
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1))) : Radio(i).SetFontSize 14
Next i
Check1.Attach GetDlgItem("Check1") : Check1.SetFontSize 14
Timer1.Attach GetDlgItem("Timer1")
Group1.Attach GetDlgItem("Group1") : Group1.setFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'プログレスバーのハンドル保存用
Var Shared pBarhWnd As Long
Var Shared Cnt As Long
Var Shared Flg As Byte
Var Shared ColorRef As Long

'=====
'=
'=====
Declare Function BkRGB (ByVal r As Byte, ByVal g As Byte, ByVal b As Byte) As Long
Function BkRGB (ByVal r As Byte, ByVal g As Byte, ByVal b As Byte) As Long

    'シフト演算した値の論理和
    BkRGB = CLng(b) * (2 ^ 16) Or CLng(g) * (2 ^ 8) Or CLng(r)
End Function

'=====
'=
'=====
Declare Function GetLong (ByVal UpperWord As Long, ByVal LowerWord As Long) As Long
Function GetLong (ByVal UpperWord As Long, ByVal LowerWord As Long) As Long

    '下位・上位を与えて長整数値 (32ビット値) を取得
    GetLong = UpperWord& + LowerWord& * &H10000
End Function

'=====
'=
'=====
Declare Function Index bdecl () As Integer
Function Index () As Integer
    Index = Val (Mid$(GetDlgRadioSelect("Radio1"), 6)) - 1
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edecl ()
Sub MainForm_Start ()

    'タイマーセット
    Timer1.SetInterval 5
    Timer1.Enable 0

    ShowWindow -1
End Sub

'=====
'=
'=====

```

```

Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var Ret As Long

    'プログレスバーのカウント
    Cnt = Cnt + 1
    Ret = Api_SendMessage(pBarhWnd, PBM_SETPOS, Cnt, 0)
    Text1.SetWindowText Str$(Cnt) & "%"

    If Cnt > 99 Then
        Cnt = 0
        Timer1.Enable 0
        Button1.EnableWindow -1
    End If
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var cc As tagINITCOMMONCONTROLSEX
    Var BkColor As Long
    Var bR As Byte
    Var bG As Byte
    Var bB As Byte
    Var Ret As Long

    cc.dwSize = Len(cc)
    cc.dwICC = ICC_PROGRESS_CLASS

    If Check1.GetCheck = 0 Then

        'プログレスバー作成(段階表示)
        pBarhWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD Or
WS_VISIBLE Or WS_BORDER, 14, 8, 238, 20, GethWnd, 0, GethInst, 0)
    Else

        'プログレスバー作成(スムーズ表示)
        pBarhWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD Or
WS_VISIBLE Or WS_BORDER Or &HB, 14, 8, 238, 20, GethWnd, 0, GethInst, 0)
    End If

    '範囲の設定(0~100)
    Ret = Api_SendMessage(pBarhWnd, PBM_SETRANGE, 0, GetLong(0, 100))

    'ステップの設定
    Ret = Api_SendMessage(pBarhWnd, PBM_SETSTEP, 1, 0)

    '背景色(RGB)を取得
    bR = Val(Edit(0).GetWindowText)
    bG = Val(Edit(1).GetWindowText)
    bB = Val(Edit(2).GetWindowText)

    BkColor = BkRGB(bR, bG, bB)

    'プログレスバーの背景色を設定
    Ret = Api_SendMessage(pBarhWnd, PBM_SETBKCOLOR, 0, ByVal BkColor)

    Select Case Index
        Case 0
            ColorRef = RGB(255, 0, 0)
        Case 1
            ColorRef = RGB(0, 255, 0)
        Case 2
            ColorRef = RGB(0, 0, 255)
    End Select

    'プログレスバーの色を設定
    Ret = Api_SendMessage(pBarhWnd, PBM_SETBARCOLOR, 0, ColorRef)

```

```

'プログレスバーの値を「0」にする
Ret = Api_SendMessage(pBarhWnd, PBM_SETPOS, 0, 0)
Cnt = 0

Timer1.Enable -1

'コマンドボタンの無効化
Button1.EnableWindow 0
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

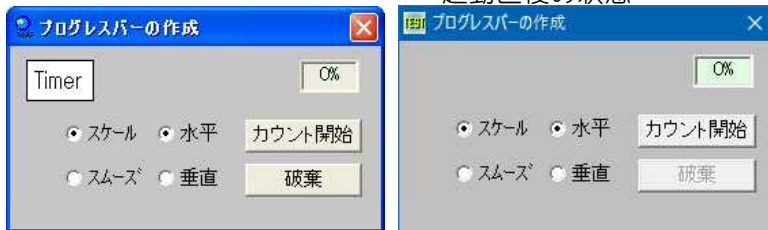
```

プログレスバーの作成

プログレスバーを作成します。
CreateWindowEx ウィンドウ(コントロール)の作成
SendMessage コントロールにメッセージを送る

タイマー、テキストボックス、ラジオボタン、ボタンを貼り付けます。 スケールとスムーズおよび水平と垂直は、それぞれグループ指定しています。

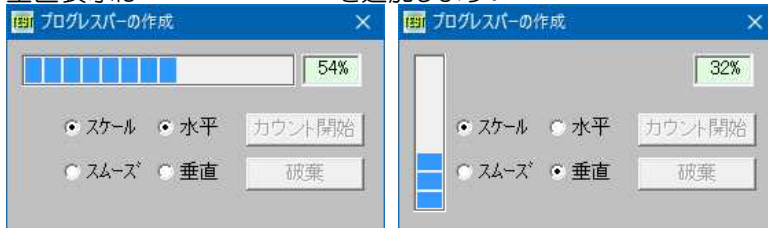
起動直後の状態



スケール表示

pBarhWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD or WS_VISIBLE or WS_BORDER, 10, 10, 250, 24, GetHwnd, 0, GetHinst, 0)

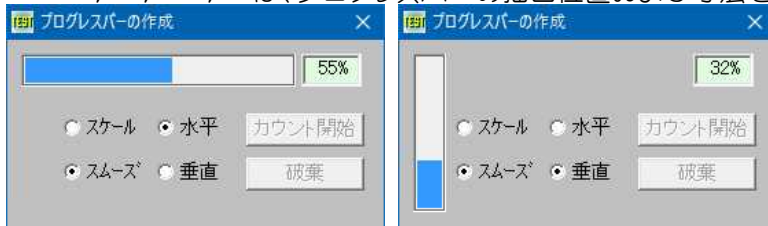
垂直表示は**PBS_VERTICAL**を追加します。



スムーズ表示

pBarhWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD or WS_VISIBLE or WS_BORDER or **PBS_SMOOTH**, 10, 10, 250, 24, GetHwnd, 0, GetHinst, 0)

10,10,250,24は、プログレスバーの描画位置および寸法を表しています。



```

'=====
'= プログレスバーの作成
'= (ProgressBar.bas)
'= Case5の部分は、1行の文字数制限(255)に注意
'=====
#include "Windows.bi"

```

```

' COMCTL32.dllからコモンコントロールクラスを登録する
Type tagINITCOMMONCONTROLSEX
    dwSize      As Long
    dwICC       As Long
End Type

' 構造体のバイト数
' ロードするクラスを指定する

' コモンコントロールのダイナミックリンクライブラリ(DLL)に含まれている、特定のコモンコントロールクラスを登録
Declare Function Api_InitCommonControlsEx& Lib "comctl32" Alias "InitCommonControlsEx"
(lpInitCtrls As tagINITCOMMONCONTROLSEX)

' ウィンドウ(コントロール)を作成
Declare Function Api_CreateWindowEx& Lib "user32" Alias "CreateWindowExA" (ByVal
ExStyle&, ByVal ClassName$, ByVal WinName$, ByVal Style&, ByVal x&, ByVal y&, ByVal
nWidth&, ByVal nHeight&, ByVal Parent&, ByVal Menu&, ByVal Instance&, ByVal Param&)

' CreateWindowExの解放
Declare Function Api_DestroyWindow& Lib "user32" Alias "DestroyWindow" (ByVal hWnd&)

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御
を返さない
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, ByVal lParam&)

' ウィンドウスタイルの定数
#define WS_CHILD &H40000000
#define WS_VISIBLE &H10000000
#define WS_BORDER &H800000
' 親ウィンドウを持つコントロール(子ウィンドウ)を作成する
' 表示する
' フォームの枠線がある

' プログレスバー関係
#define WM_USER &H400
' ユーザーが定義できるメッセージの使用領域を表すだけで
これ自体に意味はない
#define PBM_SETRANGE &H401
#define PBM_SETPOS &H402
#define PBM_DELTAPOS &H403
#define PBM_SETSTEP &H404
#define PBS_SMOOTH &HB
#define PBS_VERTICAL &H4
' (WM_USER + 1) 範囲の設定
' (WM_USER + 2) 位置の指定
' (WM_USER + 3) デフォルトの位置
' (WM_USER + 4) 増分の設定
' 滑らかなプログレスバー
' 垂直表示
#define ICC_PROGRESS_CLASS &H20
' プログレスバーコントロール
#define PROGRESS_CLASS "msctls_progress32" ' プログレスバーコントロールクラス

Var Shared Timer1 As Object
Var Shared Text1 As Object
Var Shared Radio1 As Object
Var Shared Radio2 As Object
Var Shared Radio3 As Object
Var Shared Radio4 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

Timer1.Attach GetDlgItem("Timer1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 14
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14
Radio3.Attach GetDlgItem("Radio3") : Radio3.SetFontSize 14
Radio4.Attach GetDlgItem("Radio4") : Radio4.SetFontSize 14

' プログレスバーのハンドル保存用
Var Shared hWnd As Long
Var Shared Cnt As Long
Var Shared Style(3) As byte

' =====
' =
' =====
Declare Function GetLong& (ByVal UpperWord&, ByVal LowerWord&)
Function GetLong& (ByVal UpperWord&, ByVal LowerWord&)

```

```

'下位・上位を与えて長整数値(32ビット値)を取得する
GetLong = UpperWord& + LowerWord& * &H10000
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()

    'タイマーセット
    Timer1.SetInterval 5
    Timer1.Enable 0

    Button2.EnableWindow 0
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var Ret As Long

    'プログレスバーのカウント
    Cnt = Cnt + 1
    Ret = Api_SendMessage(hWnd, PBM_SETPOS, Cnt, 0)
    Text1.SetWindowText Str$(Cnt) & "%"
    If Cnt > 99 Then
        Cnt = 0
        Timer1.Enable 0
        Button2.EnableWindow -1
    End If
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()

    'コモンコントロールを初期化する
    Var cc As tagINITCOMMONCONTROLSEX
    Var Ret As Long

    cc.dwSize = Len(cc)
    cc.dwICC = ICC_PROGRESS_CLASS

    If Radio1.GetCheck = 1 Then style(0) = 0 : style(1) = 0
    If Radio2.GetCheck = 1 Then style(0) = 0 : style(1) = 1
    If Radio3.GetCheck = 1 Then style(2) = 2 : style(3) = 0
    If Radio4.GetCheck = 1 Then style(2) = 0 : style(3) = 4

    Select Case style(0) + style(1) + style(2) + style(3)
    Case 2
        'プログレスバー作成(水平スケール表示)
        hWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD Or
WS_VISIBLE Or WS_BORDER, 10, 10, 200, 24, GethWnd, 0, GethInst, 0)
    Case 3
        'プログレスバー作成(水平スムーズ表示)
        hWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD Or
WS_VISIBLE Or WS_BORDER Or PBS_SMOOTH, 10, 10, 200, 24, GethWnd, 0, GethInst, 0)
    Case 4
        'プログレスバー作成(垂直スケール表示)
        hWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD Or
WS_VISIBLE Or WS_BORDER Or PBS_VERTICAL, 10, 10, 24, 116, GethWnd, 0, GethInst, 0)
    Case 5
        'プログレスバー作成(垂直スムーズ表示)
        hWnd = Api_CreateWindowEx(0, PROGRESS_CLASS, "Progress Bar", WS_CHILD Or
WS_VISIBLE Or WS_BORDER Or PBS_VERTICAL Or PBS_SMOOTH, 10, 10, 24, 116, GethWnd, 0,

```

```

GethInst, 0)
End Select

' 範囲の設定 (0~100)
Ret = Api_SendMessage (hWnd, PBM_SETRANGE, 0, GetLong (0, 100))

' ステップの設定
Ret = Api_SendMessage (hWnd, PBM_SETSTEP, 1, 0)

' プログレスバーの値を0にする
Ret = Api_SendMessage (hWnd, PBM_SETPOS, 0, 0)
Cnt = 0

Timer1.Enable -1

' コマンドボタンの無効化
Button1.EnableWindow 0
Button2.EnableWindow 0
End Sub

' =====
' =
' =====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Var Ret As Long

    Ret = Api_DestroyWindow (hWnd)
    Button1.EnableWindow -1
    Button2.EnableWindow 0
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

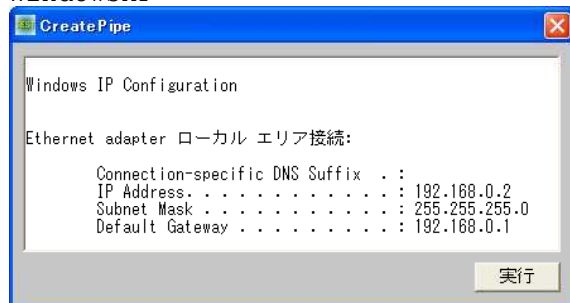
```

プロセス間の通信 (パイプ)

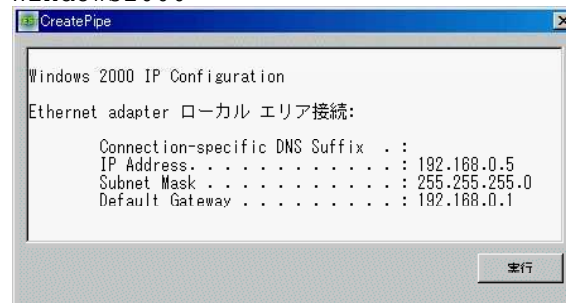
CreatePipe 匿名パイプの作成
ReadFile ファイルからデータを読み取る
CreateProcess プロセスの起動
CloseHandle オープンされているオブジェクトハンドルをクローズ
PeekNamedPipe パイプの内容を調べる
WaitForSingleObject 指定された時間が経過するまでスレッドをスリープ

例では、ipconfig.exeのデータを取得しています。

WindowsXP



Windows2000



コマンドプロンプトで確認

```
コマンド プロンプト

C:\>ipconfig

Windows IP Configuration

Ethernet adapter ローカル エリア接続:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.0.2
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1
```

```
'=====
'= プロセス間の通信 (パイプ)
'= (CreatePipu.bas)
'=====
```

```
#include "Windows.bi"
```

```
Type SECURITY_ATTRIBUTES
    nLength           As Long
    lpSecurityDescriptor As Long
    bInheritHandle     As Long
End Type
```

```
Type STARTUPINFO
    cb           As Long
    lpReserved   As Long
    lpDesktop    As Long
    lpTitle      As Long
    dwX          As Long
    dwY          As Long
    dwXSize      As Long
    dwYSize      As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags       As Long
    wShowWindow   As Integer
    cbReserved2   As Integer
    lpReserved2   As Long
    hStdInput     As Long
    hStdOutput    As Long
    hStdError     As Long
End Type
```

```
Type PROCESS_INFORMATION
    hProcess      As Long
    hThread       As Long
    dwProcessID   As Long
    dwThreadId    As Long
End Type
```

' 匿名パイプの作成

```
Declare Function Api_CreatePipe& Lib "kernel32" Alias "CreatePipe" (phReadPipe&,
phWritePipe&, lpPipeAttributes As Any, ByVal nSize&)
```

' ファイルからデータを読み取る

```
Declare Function Api_ReadFile& Lib "kernel32" Alias "ReadFile" (ByVal hFile&, ByVal
lpBuffer$, ByVal nNumberOfBytesToRead&, lpNumberOfBytesRead&, ByVal lpOverlapped As Any)
```

' プロセスの起動

```
Declare Function Api_CreateProcess& Lib "kernel32" Alias "CreateProcessA" (ByVal
AppName&, ByVal ComdLine$, ProcAttr As Any, ThreadAttr As Any, ByVal InheritHand&, ByVal
CreationFlags&, ByVal Env&, ByVal CurrentDir&, StartupInfo As Any, ProcInfo As Any)
```

' オープンされているオブジェクトハンドルをクローズ

```
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)
```

' パイプの内容を調べる

```
Declare Function Api_PeekNamedPipe Lib "kernel32" Alias "PeekNamedPipe" (ByVal  
hNamedPipe&, lpBuffer As Any, ByVal nBufferSize&, lpBytesRead&, lpTotalBytesAvail&,  
lpBytesLeftThisMessage&)
```

' 指定されたカーネルオブジェクトがシグナル状態になるか、指定された時間が経過するまでスレッドをスリープ

```
Declare Function Api_WaitForSingleObject Lib "Kernel32" Alias "WaitForSingleObject"  
(ByVal hHandle&, ByVal dwMilliseconds&)
```

```
#define NORMAL_PRIORITY_CLASS &H20          '通常クラス(一般的なプロセス)  
#define STARTF_USESTDHANDLES &H100
```

```
Var Shared Edit1 As Object  
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
```

```
'=====
```

```
Declare Function ExecCmd(cmdline$) As String
```

```
Function ExecCmd(cmdline$) As String
```

```
    Var pi As PROCESS_INFORMATION
```

```
    Var si As STARTUPINFO
```

```
    Var sa As SECURITY_ATTRIBUTES
```

```
    Var hReadPipe As Long
```

```
    Var hWritePipe As Long
```

```
    Var L As Long
```

```
    Var bSuccess As Long
```

```
    Var Buffer As String
```

```
    Var lPeekData As Long
```

```
    Var txt As String
```

```
    Var Ret As Long
```

```
    sa.nLength = Len(sa)
```

```
    sa.bInheritHandle = 1
```

```
    sa.lpSecurityDescriptor = 0
```

'匿名パイプの作成

```
Ret = Api_CreatePipe(hReadPipe, hWritePipe, sa, 0)
```

```
If Ret = 0 Then
```

```
    A% = MsgBox(GetWindowText, "CreatePipeエラー!", 0, 2)
```

```
    Exit Function
```

```
End If
```

```
si.cb = Len(si)
```

```
si.dwFlags = STARTF_USESTDHANDLES
```

```
si.hStdOutput = hWritePipe
```

'プロセスの起動(例では、ipconfig.exe)

```
Ret = Api_CreateProcess(0, cmdline$, sa, sa, 1, NORMAL_PRIORITY_CLASS, 0, 0, si, pi)
```

```
If Ret <> 0 Then
```

```
    Do
```

'パイプからデータを取得

```
Ret = Api_PeekNamedPipe(hReadPipe, ByVal 0, 0, ByVal 0, lPeekData, ByVal 0)
```

```
If lPeekData > 0 Then
```

```
    Buffer = Space$(lPeekData)
```

'データの読み取り

```
bSuccess = Api_ReadFile(hReadPipe, Buffer, Len(Buffer), L, 0)
```

```
If bSuccess = 1 Then
```

```
    txt = txt & Left$(Buffer, L)
```

```
Else
```

```
    A% = MsgBox(GetWindowText, "ReadFileに失敗!", 0, 2)
```

```
End If
```

```
Else
```

```
    bSuccess = Api_WaitForSingleObject(pi.hProcess, 0)
```

```

        If bSuccess = 0 Then
            Exit Do
        End If
    End If

    CallEvent
    Loop
Else
    A% = MsgBox(GetWindowText, "プロセスエラー！", 0, 2)
End If

' ハンドルの解放
Ret = Api_CloseHandle(pi.hProcess)
Ret = Api_CloseHandle(pi.hThread)
Ret = Api_CloseHandle(hReadPipe)
Ret = Api_CloseHandle(hWritePipe)

ExecCmd = txt
End Function

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Edit1.SetWindowText ExecCmd("ipconfig.exe")
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

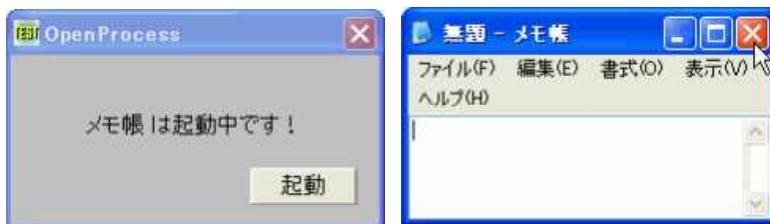
プロセス終了まで待機

CreateProcess プロセスの起動

CloseHandle オープンされているオブジェクトハンドルをクローズ

OpenProcess 既存のプロセスオブジェクトのハンドルを取得

例では、メモ帳を起動し終了を監視しています。



```

' =====
' = プロセス終了まで待機
' = (OpenProcess2.bas)
' =====
#include "Windows.bi"

Type PROCESS_INFORMATION
    hProcess      As Long
    hThread       As Long
    dwProcessId   As Long
    dwThreadId    As Long
End Type

Type STARTUPINFO

```

```

    cb                As Long
    lpReserved        As Long
    lpDesktop          As Long
    lpTitle            As Long
    dwX                As Long
    dwY                As Long
    dwXSize            As Long
    dwYSize            As Long
    dwXCountChars      As Long
    dwYCountChars      As Long
    dwFillAttribute    As Long
    dwFlags            As Long
    wShowWindow        As Integer
    cbReserved2        As Integer
    lpReserved2        As Long
    hStdInput          As Long
    hStdOutput          As Long
    hStdError          As Long
End Type

' プロセスの起動
Declare Function Api_CreateProcess& Lib "kernel32" Alias "CreateProcessA" (ByVal Name$,
ByVal Cmd$, pAtt As Any, tAtt As Any, ByVal Hand&, ByVal Flg&, Env As Any, ByVal Dir$, sInfo
As Any, pInfo As PROCESS_INFORMATION)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)

' 既存のプロセスオブジェクトのハンドルを取得
Declare Function Api_OpenProcess& Lib "kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess&, ByVal bInheritHandle&, ByVal dwProcessID&)

#define NORMAL_PRIORITY_CLASS &H20      ' 通常クラス (一般的なプロセス)
#define PROCESS_TERMINATE &H1           ' 取得したハンドルを TerminateProcess 関数で使用できる
                                         ' ように

Var Shared ProcessId As Long

Var Shared Timer1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Timer1.Attach GetDlgItem("Timer1")
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pi As PROCESS_INFORMATION
    Var si As STARTUPINFO
    Var Ret As Long

    si.cb = Len(si)
    Ret = Api_CreateProcess(ByVal 0, "notepad.exe", ByVal 0, ByVal 0, 0,
NORMAL_PRIORITY_CLASS, ByVal 0, "c:¥", si, pi)

    Ret = Api_CloseHandle(pi.hProcess)
    Ret = Api_CloseHandle(pi.hThread)

    ProcessId = pi.dwProcessId

    Timer1.SetInterval 100
    Timer1.Enable -1
End Sub

```

```

'=====
'= TimerによりhProcHandleの変化を監視
'=====
Declare Sub Timer1_Timer edecl ()
Sub Timer1_Timer()
    Var hProcHandle As Long
    Var Ret As Long

    'プロセスオブジェクトのハンドル取得
    hProcHandle = Api_OpenProcess(PROCESS_TERMINATE, 0, ProcessId)

    If hProcHandle = 0 Then
        Text1.SetWindowText "メモ帳は終了しました！"
        Timer1.Enable 0
        Exit Sub
    Else
        Text1.SetWindowText "メモ帳は起動中です！"
    End If

    Ret = Api_CloseHandle(hProcHandle)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

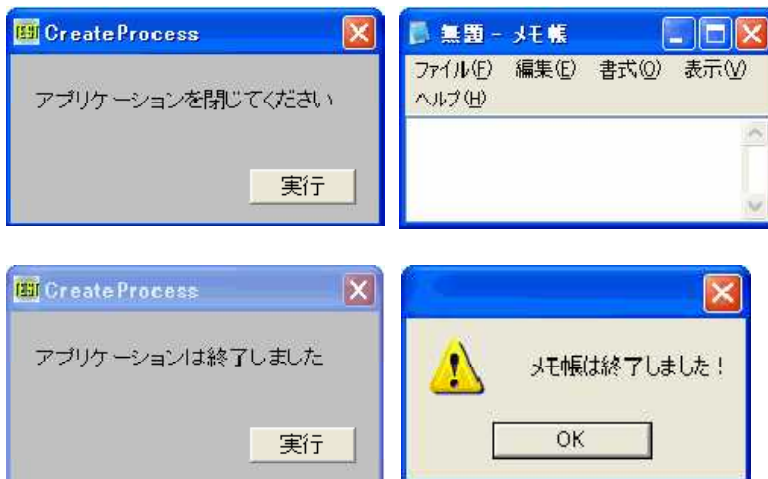
```

プロセスの起動と終了

CreateProcess プロセスの起動

WaitForSingleObject 指定されたカーネルオブジェクトがシグナル状態になるか、指定された時間が経過するまでスレッドをスリープ

CloseHandle オープンされているオブジェクトハンドルをクローズ



```

'=====
'= プロセスの起動と終了
'= (CreateProcess2.bas)
'=====
#include "Windows.bi"

```

```

Type STARTUPINFO
    cb                As Long
    lpReserved        As Long
    lpDesktop         As Long
    lpTitle           As Long

```

```

    dwX                As Long
    dwY                As Long
    dwXSize            As Long
    dwYSize            As Long
    dwXCountChars      As Long
    dwYCountChars      As Long
    dwFillAttribute    As Long
    dwFlags             As Long
    wShowWindow        As Integer
    cbReserved2        As Integer
    lpReserved2        As Long
    hStdInput           As Long
    hStdOutput          As Long
    hStdError           As Long
End Type

Type PROCESS_INFORMATION
    hProcess           As Long
    hThread             As Long
    dwProcessId        As Long
    dwThreadId         As Long
End Type

#define NORMAL_PRIORITY_CLASS &H20      '通常クラス(一般的なプロセス)
#define INFINITE &HFFFFFF               '無限に中断
#define WAIT_TIMEOUT &H102              'タイムアウト時間が経過したことを示す

' プロセスの起動
Declare Function Api_CreateProcess& Lib "Kernel32" Alias "CreateProcessA" (ByVal App&,
ByVal Cmd$, ByVal PAttr&, ByVal TAttr&, ByVal InhHandle&, ByVal Flags&, ByVal lpEnv&,
ByVal lpCurDir&, StartupInfo As STARTUPINFO, ProcessInfo As PROCESS_INFORMATION)

' 指定されたカーネルオブジェクトがシグナル状態になるか、指定された時間が経過するまでスレッドをスリープ
Declare Function Api_WaitForSingleObject& Lib "Kernel32" Alias "WaitForSingleObject"
(ByVal hHandle&, ByVal dwMilliseconds&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function ShellWait(cmdline As String, ByVal bShowApp As Integer) As Integer
Function ShellWait(cmdline As String, ByVal bShowApp As Integer) As Integer
    Var pi As PROCESS_INFORMATION
    Var si As STARTUPINFO
    Var Ret As Long

    si.cb = Len(si)
    si.wShowWindow = Abs(bShowApp)
    si.dwFlags = 1

    Text1.SetWindowText "メモ帳起動"
    Ret = Api_CreateProcess(0, cmdline, 0, 0, 1, NORMAL_PRIORITY_CLASS, 0, 0, si, pi)

    If Ret = 0 Then
        A% = MsgBox("", "起動に失敗しました！", 0, 2)

        ShellWait = False
        Exit Function
    End If

    Text1.SetWindowText "アプリケーションを閉じてください"

```

```

Do While Api_WaitForSingleObject(pi.hProcess, 10) = WAIT_TIMEOUT
    CallEvent
Loop

Text1.SetWindowText "アプリケーションは終了しました"

Ret = Api_CloseHandle(pi.hProcess)

ShellWait = (Ret <> 0)
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Integer

    Ret = ShellWait("notepad.exe", True)

    A% = MsgBox("", "メモ帳は終了しました！", 0, 2)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

プロセッサがサポートする機能

プロセッサがサポートする機能を取得します。

IsProcessorFeaturePresent プロセッサ機能がサポートされているかどうかを取得



```

'=====
'= プロセッサがサポートする機能
'= (IsProcessorFeaturePresent.bas)
'=====

#define PF_FLOATING_POINT_PRECISION_ERRATA 0      '浮動小数点精度エラー発生可能性
#define PF_FLOATING_POINT_EMULATED 1            '浮動小数点演算がソフトウェアエミュレータを使ってエミュレート
#define PF_COMPARE_EXCHANGE_DOUBLE 2            '比較置換二重演算に対応
#define PF_MMX_INSTRUCTIONS_AVAILABLE 3          'MMX命令セットに対応
#define PF_XMMI_INSTRUCTIONS_AVAILABLE 6         'XMMI命令セットに対応
#define PF_3DNOW_INSTRUCTIONS_AVAILABLE 7        '3D-Now命令セットに対応
#define PF_RDTSC_INSTRUCTION_AVAILABLE 8        'RDTSC命令セットに対応
#define PF_PAE_ENABLED 9                        'プロセッサはPAE-enabled

' プロセッサ機能がサポートされているかどうかを取得
Declare Function Api_IsProcessorFeaturePresent& Lib "kernel32" Alias
"IsProcessorFeaturePresent" (ByVal ProcessorFeature&)

```

```

'=====
'=
'=====
Declare Sub GetFeature (Index As Long, Msg As String)
Sub GetFeature (Index As Long, Msg As String)

    If Api_IsProcessorFeaturePresent (Index) = 0 Then
        Print Msg & ": false"
    Else
        Print Msg & ": true"
    End If
End Sub

GetFeature PF_FLOATING_POINT_PRECISION_ERRATA, "フローティングポイントエラー "
GetFeature PF_FLOATING_POINT_EMULATED, "浮動小数点演算エミュレート "
GetFeature PF_COMPARE_EXCHANGE_DOUBLE, "ダブルオペレーション "
GetFeature PF_MMX_INSTRUCTIONS_AVAILABLE, "MMX (MultiMedia eXtension) "
GetFeature PF_XMMI_INSTRUCTIONS_AVAILABLE, "XMMI "
GetFeature PF_3DNOW_INSTRUCTIONS_AVAILABLE, "3D-Now (AMD社の拡張命令セット) "
GetFeature PF_RDTSC_INSTRUCTION_AVAILABLE, "RDTSC (read-time stamp counter) "
GetFeature PF_PAE_ENABLED, "PAE-enabled (物理アドレス拡張) "

Stop
End

```

プロセッサの情報取得

GetSystemInfoでプロセッサの情報を取得、レジストリからCPUのスピードを取得します。

GetSystemInfo システム情報を取得

RegOpenKey 指定されたレジストリキーをオープン

RegQueryValueEx レジストリの値を取得

RegCloseKey レジストリのハンドルを解放

HP (WindowsXP)



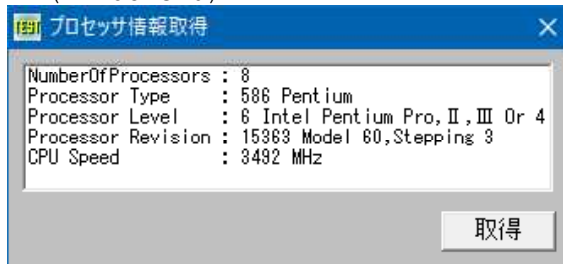
Dell (WindowsXP)



Valuestar (WindowsXP)



HP (Windows10)



```

'=====
'= プロセッサの情報取得
'= (GetProcessor.bas)
'=====
#include "Windows.bi"

#define PROCESSOR_ALPHA_21064 21064
#define PROCESSOR_ARCHITECTURE_ALPHA 2
#define PROCESSOR_ARCHITECTURE_ALPHA64 7
#define PROCESSOR_ARCHITECTURE_ARM 5
#define PROCESSOR_ARCHITECTURE_IA64 6
#define PROCESSOR_ARCHITECTURE_INTEL 0

```



```

#define PROCESSOR_ARCHITECTURE_MIPS 1      '
#define PROCESSOR_ARCHITECTURE_PPC 3      '
#define PROCESSOR_ARCHITECTURE_SHX 4      '
#define PROCESSOR_ARCHITECTURE_UNKNOWN &HFFF '
#define PROCESSOR_ARM_7TDMI 70001         'WindowsCE
#define PROCESSOR_ARM720 1824              'WindowsCE-0x720
#define PROCESSOR_ARM820 2080              'WindowsCE-0x820
#define PROCESSOR_ARM920 2336              'WindowsCE-0x920
#define PROCESSOR_HITACHI_SH3 10003        'WindowsCE
#define PROCESSOR_HITACHI_SH3E 10004       'WindowsCE
#define PROCESSOR_HITACHI_SH4 10005       'WindowsCE
#define PROCESSOR_INTEL_386 386            '
#define PROCESSOR_INTEL_486 486            '
#define PROCESSOR_INTEL_PENTIUM 586        '
#define PROCESSOR_LEVEL_80386 3            '
#define PROCESSOR_LEVEL_80486 4            '
#define PROCESSOR_LEVEL_PENTIUM 5          '
#define PROCESSOR_LEVEL_PENTIUMII 6        '
#define PROCESSOR_MIPS_R4000 4000          '
#define PROCESSOR_MOTOROLA_821 821         'WindowsCE
#define PROCESSOR_PPC_601 601              '
#define PROCESSOR_PPC_603 603              '
#define PROCESSOR_PPC_604 604              '
#define PROCESSOR_PPC_620 620              '
#define PROCESSOR_SHx_SH3 103              'WindowsCE
#define PROCESSOR_SHx_SH4 104              'WindowsCE
#define PROCESSOR_STRONGARM 2577           'WindowsCE-0xA11

#define sCPURegKey "HARDWARE\DESCRIPTION\System\CentralProcessor\0"
#define HKEY_LOCAL_MACHINE -2147483646     'PCを利用するユーザーに共通の設定情報

```

```

Type SYSTEM_INFO
    dwOemID                As Long
    dwPageSize              As Long
    lpMinimumApplicationAddress As Long
    lpMaximumApplicationAddress As Long
    dwActiveProcessorMask   As Long
    dwNumberOfProcessors    As Long
    dwProcessorType         As Long
    dwAllocationGranularity As Long
    wProcessorLevel         As Integer
    wProcessorRevision      As Integer
End Type

```

' システム情報を取得

```

Declare Sub Api_GetSystemInfo Lib "Kernel32" Alias "GetSystemInfo" (lpSystemInfo As SYSTEM_INFO)

```

' 指定されたレジストリキーをオープン

```

Declare Function Api_RegOpenKey& Lib "advapi32" Alias "RegOpenKeyA" (ByVal hKey&, ByVal lpSubKey$, phkResult&)

```

' レジストリの値を取得

```

Declare Function Api_RegQueryValueEx& Lib "advapi32" Alias "RegQueryValueExA" (ByVal hKey&, ByVal lpvName$, ByVal lpReserved&, ByVal lpType&, lpData As Any, lpcbData&)

```

' レジストリのハンドルを解放

```

Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)
Var Shared List1 As Object
Var Shared Button1 As Object

```

```

List1.Attach GetDlgItem("List1") : List1.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

```

'=====
'=
'=====

```

```

Declare Function GetCPUSpeed() As Long
Function GetCPUSpeed() As Long
    Var hKey As Long

```

```

    Var cpuSpeed As Long
    Var Ret As Long

'CPU key オープン
    Ret = Api_RegOpenKey(HKEY_LOCAL_MACHINE, sCPURegKey, hKey)

'値を取得
    Ret = Api_RegQueryValueEx(hKey, "~MHz", 0, 0, cpuSpeed, 4)
    Ret = Api_RegCloseKey(hKey)

    GetCPUSpeed = cpuSpeed
End Function

'=====
'=
'=====
Declare Function HiByte(ByVal wParam As Integer) As Byte
Function HiByte(ByVal wParam As Integer) As Byte
    HiByte = (wParam And &HFF00) ¥ (&H100)
End Function

'=====
'=
'=====
Declare Function LoByte(ByVal wParam As Integer) As Byte
Function LoByte(ByVal wParam As Integer) As Byte
    LoByte = wParam And &HFF
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var si As SYSTEM_INFO
    Var tmp As String
    Var Ret As Long

    Api_GetSystemInfo si

    List1.ResetContent
    List1.AddString "NumberOfProcessors :" & Str$(si.dwNumberOfProcessors)

    Select Case si.dwProcessorType
        Case PROCESSOR_INTEL_386
            tmp = " 386"
        Case PROCESSOR_INTEL_486
            tmp = " 486"
        Case PROCESSOR_INTEL_PENTIUM
            tmp = " Pentium"
        Case PROCESSOR_MIPS_R4000
            tmp = " MIPS 4000"
        Case PROCESSOR_ALPHA_21064
            tmp = " Alpha"
    End Select

    List1.AddString "Processor Type      :" & Str$(si.dwProcessorType) & tmp

    Select Case SI.wProcessorLevel
        Case PROCESSOR_LEVEL_80386
            tmp = " Intel 80386"
        Case PROCESSOR_LEVEL_80486
            tmp = " Intel 80486"
        Case PROCESSOR_LEVEL_PENTIUM
            tmp = " Intel Pentium"
        Case PROCESSOR_LEVEL_PENTIUMII
            tmp = " Intel Pentium Pro, II, III Or 4"
    End Select

    List1.AddString "Processor Level      :" & Str$(si.wProcessorLevel) & tmp

```

```

List1.AddString "Processor Revision :" & Str$(si.wProcessorRevision) & " Model" &
Str$(HiByte(si.wProcessorRevision)) & ",Stepping" &
Str$(LoByte(si.wProcessorRevision))
List1.AddString "CPU Speed          :" & Str$(GetCPUSpeed()) & " MHz"
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

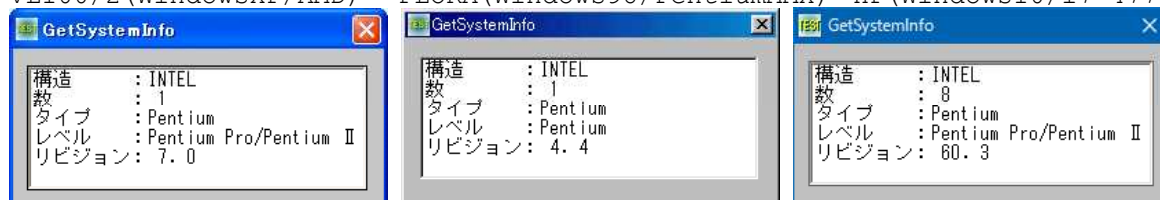
```

プロセッサのタイプを取得

システム情報 (プロセッサのタイプ) を取得します。

GetSystemInfo システム情報を取得する関数

VL100/2 (WindowsXP/AMD) FLORA (Windows98/PentiumMMX) HP (Windows10/i7-4770K)



```

'=====
'= プロセッサのタイプを取得
'= (GetSystemInfo.bas)
'=====
#include "Windows.bi"

Type SYSTEM_INFO
'    dwOemId                As Long
    wProcessorArchitecture  As Integer
    wReserved               As Integer
    dwPageSize              As Long
    lpMinimumApplicationAddress As Long
    lpMaximumApplicationAddress As Long
    dwActiveProcessorMask    As Long
    dwNumberOfProcessors     As Long
    dwProcessorType          As Long
    dwAllocationGranularity  As Long
    wProcessorLevel          As Integer
    wProcessorRevision       As Integer
End Type

#define PROCESSOR_ARCHITECTURE_INTEL 0
#define PROCESSOR_INTEL_386 386
#define PROCESSOR_INTEL_486 486
#define PROCESSOR_INTEL_PENTIUM 586

' システム情報を取得
Declare Sub Api_GetSystemInfo Lib "Kernel32" Alias "GetSystemInfo" (lpSystemInfo As SYSTEM_INFO)

Var Shared List1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

'=====
'=
'=====
Declare Sub Mainform_Start edec1 ()
Sub Mainform_Start ()

```

```

Var si As SYSTEM_INFO

'システム情報を取得
Api_GetSystemInfo si

List1.ResetContent

'プロセッサ情報を表示
Select Case si.wProcessorArchitecture
    Case PROCESSOR_ARCHITECTURE_INTEL
        List1.AddString "構造" & "INTEL"
    Case Else
        List1.AddString "構造" & "不明"
End Select

'数を表示
List1.AddString "数" & Str$(si.dwNumberOfProcessors)

'タイプを表示
Select Case si.dwProcessorType
    Case PROCESSOR_INTEL_386
        List1.AddString "タイプ" & "386"
    Case PROCESSOR_INTEL_486
        List1.AddString "タイプ" & "486"
    Case PROCESSOR_INTEL_PENTIUM
        List1.AddString "タイプ" & "Pentium"
    Case Else
        List1.AddString "タイプ" & "不明"
End Select

'レベルを表示
Select Case si.wProcessorLevel
    Case 3
        List1.AddString "レベル" & "80386"
    Case 4
        List1.AddString "レベル" & "80486"
    Case 5
        List1.AddString "レベル" & "Pentium"
    Case 6
        List1.AddString "レベル" & "Pentium Pro/Pentium II"
    Case Else
        List1.AddString "レベル" & "不明"
End Select

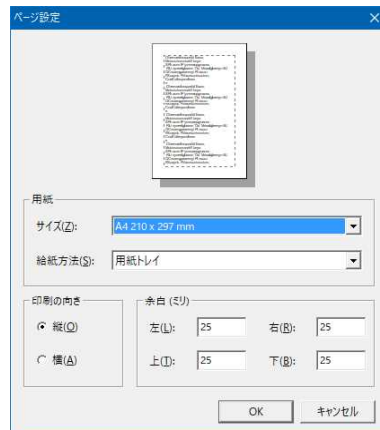
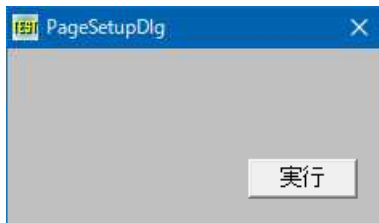
'リビジョンを表示
List1.AddString "リビジョン:" & Str$(((si.wProcessorRevision And &HFF00) \ &HFF)) & "." &
Str$((si.wProcessorRevision And &HFF))
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ページ設定ダイアグラム表示

ページ設定のダイアログボックスを呼び出します。
PageSetupDlg ページ設定のダイアログボックスを作成



```
'=====
'= ページ設定ダイアグラム表示
'= (PageSetupDlg.bas)
'=====
#include "Windows.bi"
```

```
Type POINTAPI
    x As Long
    y As Long
End Type
```

```
Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type
```

```
Type PAGESETUPDLG
    lStructSize As Long
    hwndOwner As Long
    hDevMode As Long
    hDevNames As Long
    flags As Long
    ptPaperSize As POINTAPI
    rtMinMargin As RECT
    rtMargin As RECT
    hInstance As Long
    lCustData As Long
    lpfnPageSetupHook As Long
    lpfnPagePaintHook As Long
    lpPageSetupTemplateName As Long
    hPageSetupTemplate As Long
End Type
```

' ページ設定のダイアログボックスを作成

```
Declare Function Api_PageSetupDlg Lib "comdlg32" Alias "PageSetupDlgA" (pPagesetupdlg
As PAGESETUPDLG)
```

```
'=====
'=
'=====
```

```
Declare Function ShowPageSetupDlg() As Long
Function ShowPageSetupDlg() As Long
    Var PSD As PAGESETUPDLG
    PSD.lStructSize = Len(PSD)
    PSD.hwndOwner = GethWnd
    PSD.hInstance = GethInst
    PSD.flags = 0
```

' 構造体サイズ
' オーナーウィンドウ設定
' インスタンス設定

```
    If Api_PageSetupDlg(PSD) Then
        ShowPageSetupDlg = 0
    Else
        ShowPageSetupDlg = -1
```

```

End If
End Function

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Ret As Long

    Ret = ShowPageSetupDlg ()
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

ベジエ曲線の描画 (1)

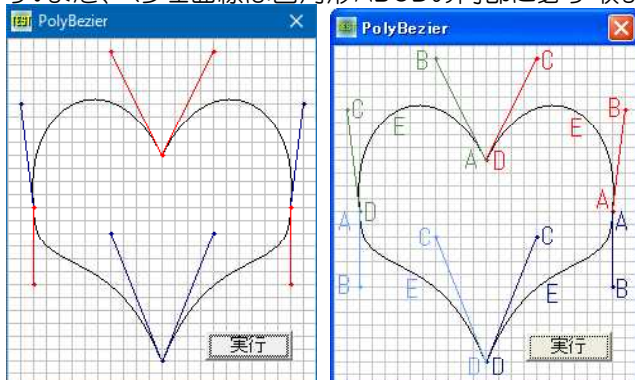
ベジエ曲線を描画します。例ではハートを描画しています。

PolyBezier ベジエ曲線を描画

ベジエ曲線は黒で描画されています。

ベジエ曲線を描くには4つの点(順にA、B、C、D)を指定し、その結果Eが描画されます。

ベジエ曲線は点A、点Dをそれぞれ始点、終点とし、この2点を必ず通り、線分AB、線分CDはベジエ曲線の接線となります。また、ベジエ曲線は四角形ABCDの内部に必ず収まります。



```

' =====
' = ベジエ曲線の描画
' = (PolyBezier.bas)
' =====
#include "Windows.bi"

```

```

Type POINTAPI
    x As Long
    y As Long
End Type

```

' ベジエ曲線を描画

```

Declare Function Api_PolyBezier& Lib "gdi32" Alias "PolyBezier" (ByVal hDC&, lppt As POINTAPI, ByVal cPoints&)

```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```

Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

```

' デバイスコンテキストを解放

```

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pts(3) As POINTAPI
    Var hDC As Long
    Var Ret As Long

    For x = 0 To GetWidth Step 10
        Line(x, 0) - (x, GetHeight), , 14
    Next
    For y = 0 To GetHeight Step 10
        Line(0, y) - (GetWidth, y), , 14
    Next

    hDC = Api_GetDC(GethWnd)

    pts(0).x = 120 : pts(0).y = 90
    pts(1).x = 160 : pts(1).y = 10
    pts(2).x = 230 : pts(2).y = 50
    pts(3).x = 220 : pts(3).y = 130
    GoSub *DrawBezier

    pts(0).x = 120 : pts(0).y = 90
    pts(1).x = 80 : pts(1).y = 10
    pts(2).x = 10 : pts(2).y = 50
    pts(3).x = 20 : pts(3).y = 130
    GoSub *DrawBezier

    pts(0).x = 220 : pts(0).y = 130
    pts(1).x = 220 : pts(1).y = 190
    pts(2).x = 160 : pts(2).y = 150
    pts(3).x = 120 : pts(3).y = 250
    GoSub *DrawBezier

    pts(0).x = 20 : pts(0).y = 130
    pts(1).x = 20 : pts(1).y = 190
    pts(2).x = 80 : pts(2).y = 150
    pts(3).x = 120 : pts(3).y = 250
    GoSub *DrawBezier

    Ret = Api_ReleaseDC(GethWnd, hDC)
    Exit Sub

    *DrawBezier
    Ret = Api_PolyBezier(hDC, pts(0), 4)
    Line(pts(0).x, pts(0).y) - (pts(1).x, pts(1).y), , 5
    Line(pts(2).x, pts(2).y) - (pts(3).x, pts(3).y), , 2

    SetDrawWidth 3

    For i = 0 To 3
        If i < 2 Then col = 5 else col = 2
        pset(pts(i).x, pts(i).y), col
    Next

    SetDrawWidth 0
    Return
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

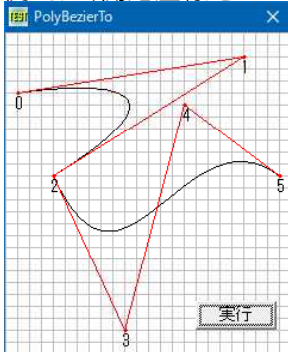
ベジェ曲線の描画 (II)

連続したベジェ曲線を描画します。

PolyBezierTo ベジェ曲線を描画。開始点はカレントポジションを使用し最後に描画したベジェ曲線の終了点にカレントポジションを設定

MoveToEx 現在位置を受け取るバッファを参照で指定

例では、開始座標をMoveToExで設定しています。数値は配列の座標を表しています。



```
'=====
'= ベジェ曲線の描画 (II)
'= (PolyBezierTo.bas)
'=====

#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type

' ベジェ曲線を描画。開始点はカレントポジションを使用し最後に描画したベジェ曲線の終了点にカレントポジションを設定
Declare Function Api_PolyBezierTo& Lib "gdi32" Alias "PolyBezierTo" (ByVal hDC&, lppt As POINTAPI, ByVal cCount&)

' 現在位置を受け取るバッファを参照で指定
Declare Function Api_MoveToEx& Lib "gdi32" Alias "MoveToEx" (ByVal hDC&, ByVal x&, ByVal y&, ByVal lpPoint As Any)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

'=====
'=
'=====

Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pts(5) As POINTAPI
    Var hDC As Long
    Var Ret As Long

    For x = 0 To GetWidth Step 10
        Line(x, 0) - (x, GetHeight), , 14
    Next
    For y = 0 To GetHeight Step 10
        Line(0, y) - (GetWidth, y), , 14
    Next

    hDC = Api_GetDC(GethWnd)

    pts(0).x = 10 : pts(0).y = 50
    pts(1).x = 200 : pts(1).y = 20
    pts(2).x = 40 : pts(2).y = 120
```



```

pts(3).x = 100 : pts(3).y = 250
pts(4).x = 150 : pts(4).y = 60
pts(5).x = 230 : pts(5).y = 120

Ret = Api_MoveToEx(hDC, 10, 50, ByVal 0)
Ret = Api_PolyBezierTo(hDC, pts(0), 6)

GoSub *DrawPointAndLine

Ret = Api_ReleaseDC(GetHwnd, hDC)
Exit Sub

*DrawPointAndLine
Pset(pts(0).x, pts(0).y)

For i = 1 To 5
    Connect(pts(i - 1).x, pts(i - 1).y) - (pts(i).x, pts(i).y), 5
Next

SetDrawWidth 3

For i = 0 To 5
    Pset(pts(i).x, pts(i).y), 5
    Symbol(pts(i).x - 1, pts(i).y + 2), Trim$(Str$(i)), 1, 1
Next

SetDrawWidth 0
Return
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ベジエ曲線の描画(III)

ベジエ曲線を描画します。

PolyBezier ベジエ曲線を描画

BeginPath デバイスコンテキストのパスを作成

EndPath BeginPathの終了

FillPath 選択されているブラシでパスの内部を塗りつぶす

StrokeAndFillPath パスを描画し、その内部を選択されているブラシで塗りつぶす

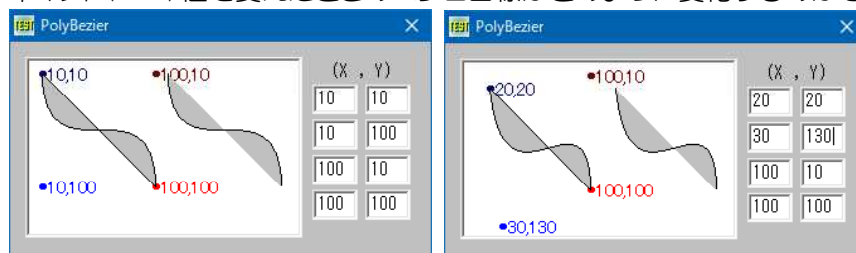
GetStokObject ストックオブジェクトのハンドルを取得

SelectObject 指定したデバイスコンテキストのオブジェクトを解放

GetDC デバイスコンテキストを取得

ReleaseDC デバイスコンテキストを解放

ポイント (Pts) 値を変えたときのベジエ曲線がどのように変化するのかを確認しています。



```

'=====
'= ベジエ曲線の描画(III)
'= (PolyBezier2.bas)
'=====

```

```

#include "Windows.bi"

Type POINTAPI
    X    As Long
    Y    As Long
End Type

' ベジエ曲線を描画
Declare Function Api_PolyBezier& Lib "gdi32" Alias "PolyBezier" (ByVal hDC&, lppt As POINTAPI, ByVal cPoints&)

' hDCで指定されたデバイスコンテキストのパスの作成
Declare Function Api_BeginPath& Lib "gdi32" Alias "BeginPath" (ByVal hDC&)

' BeginPathで開始したパスの作成を終了
Declare Function Api_EndPath& Lib "gdi32" Alias "EndPath" (ByVal hDC&)

' 選択されているブラシによってパスの内部を塗りつぶす
Declare Function Api_FillPath& Lib "gdi32" Alias "FillPath" (ByVal hDC&)

' パスを描画し、その内部を選択されているブラシによって塗りつぶす
Declare Function Api_StrokeAndFillPath& Lib "gdi32" Alias "StrokeAndFillPath" (ByVal hDC&)

' スtockオブジェクトのハンドルを取得
Declare Function Api_GetStockObject& Lib "gdi32" Alias "GetStockObject" (ByVal nIndex&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define BLACK_BRUSH 4
#define DKGRAY_BRUSH 3
#define GRAY_BRUSH 2
#define LTGRAY_BRUSH 1
#define WHITE_BRUSH 0
#define BLACK_PEN 7

' 黒
' 暗い灰色
' 灰色
' 明るい灰色
' 白

Var Shared Timer1 As Object
Var Shared Picture1 As Object
Var Shared Edit(7) As Object
Var Shared Text1 As Object

Timer1.Attach GetDlgItem("Timer1")
Picture1.Attach GetDlgItem("Picture1")
For i = 0 To 7
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
    Edit(i).SetFontSize 14
Next
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14

' =====
' =
' =====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Timer1.SetInterval 100
    Timer1.Enable -1
End Sub

' =====
' =
' =====

Declare Sub Timer1_Timer edec1 ()

```

```

Sub Timer1_Timer()
    Var hDC As Long
    Var Pts(3) As POINTAPI
    Var Cnt As Long
    Var OldPen As Long
    Var OldBrush As Long
    Var Ret As Long

    Picture1.cls
    hDC = Api_GetDC(Picture1.GethWnd)

    For Cnt = 0 To 3
        Pts(Cnt).X = Val(Edit(Cnt).GetWindowText)
        Pts(Cnt).Y = Val(Edit(Cnt + 4).GetWindowText)
        Picture1.SetDrawWidth 5
        Picture1.Pset(Pts(Cnt).X, Pts(Cnt).Y), Cnt + 2
        Picture1.SetDrawWidth 0
        Picture1.Symbol(Pts(Cnt).X + 5, Pts(Cnt).Y - 6), Trim$(Str$(Pts(Cnt).X)) & ", " &
Trim$(Str$(Pts(Cnt).Y)), 1, 1, Cnt + 2
    Next

    OldPen = Api_SelectObject(hDC, Api_GetStockObject(BLACK_PEN))
    OldBrush = Api_SelectObject(hDC, Api_GetStockObject(LTGRAY_BRUSH))

    Ret = Api_BeginPath(hDC)
    Ret = Api_PolyBezier(hDC, Pts(0), 4)
    Ret = Api_EndPath(hDC)

    Ret = Api_StrokeAndFillPath(hDC)

    ' オフセット (右側ベジェ曲線)
    For Cnt = 0 To 3
        Pts(Cnt).X = Pts(Cnt).X + 100
    Next

    Ret = Api_BeginPath(hDC)
    Ret = Api_PolyBezier(hDC, Pts(0), 4)
    Ret = Api_EndPath(hDC)

    Ret = Api_FillPath(hDC)
    Ret = Api_PolyBezier(hDC, Pts(0), 4)

    Ret = Api_SelectObject(hDC, OldPen)
    Ret = Api_SelectObject(hDC, OldBrush)
    Ret = Api_ReleaseDC(Picture1.GethWnd, hDC)
End Sub

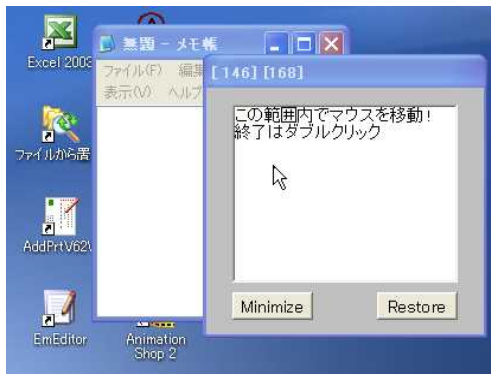
' =====
' =
' =====

While 1
    WaitEvent
Wend
Stop
End

```

別プロセスウィンドウの位置・サイズを変える

SetWindowPos ウィンドウのサイズ、位置、および z オーダーを設定
ShowWindow 指定されたウィンドウの表示状態を設定
GetWindowRect ウィンドウの座標をスクリーン座標系で取得
FindWindow クラス名またはキャプションを与えてウィンドウのハンドルを取得
GetDeviceCaps デバイス固有の情報を取得
SendMessage ウィンドウにメッセージを送信
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```
'=====
'= 別プロセスウィンドウの位置・サイズを変える
'= (SetWindowPos2.bas)
'=====
```

```
#include "Windows.bi"
```

```
Type RECT
```

```
    Left        As Long
```

```
    Top         As Long
```

```
    Right       As Long
```

```
    Bottom      As Long
```

```
End Type
```

```
' ウィンドウのサイズ、位置、および z オーダーを設定
```

```
Declare Function Api_SetWindowPos& Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal  
hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal CX&, ByVal CY&, ByVal uFlags&)
```

```
' 指定されたウィンドウの表示状態を設定
```

```
Declare Function Api_ShowWindow& Lib "user32" Alias "ShowWindow" (ByVal hWnd&, ByVal  
nCmdShow&)
```

```
' ウィンドウの座標をスクリーン座標系で取得
```

```
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,  
lpRect As RECT)
```

```
' クラス名またはキャプションを与えてウィンドウのハンドルを取得
```

```
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,  
ByVal lpWindowName$)
```

```
' デバイス固有の情報を取得
```

```
Declare Function Api_GetDeviceCaps& Lib "gdi32" Alias "GetDeviceCaps" (ByVal hDC&, ByVal  
nIndex&)
```

```
' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御  
を返さない
```

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal  
wMsg&, ByVal wParam&, lParam As Any)
```

```
' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
```

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

```
' デバイスコンテキストを解放
```

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
#define SWP_NOMOVE &H2
```

```
#define SWP_NOSIZE &H1
```

```
#define HWND_TOPMOST (-1)
```

```
#define SW_MAXIMIZE 3
```

```
#define SW_MINIMIZE 6
```

```
#define SW_RESTORE 9
```

```
#define WM_CLOSE &H10
```

```
#define HORZRES 8
```

```
#define VERTRES 10
```

```
'ウィンドウの現在位置を保持する
```

```
'ウィンドウの現在のサイズを保持する
```

```
'ウィンドウを常に最前面に配置
```

```
'最大化
```

```
'指定のウィンドウをアイコン化しタスクリスト内のトップレベル
```

```
ウィンドウをアクティブ化
```

```
'ウィンドウをアクティブ化し表示。ウィンドウがアイコン化ま  
たは最大化されているときは元の位置とサイズに
```

```
'ウィンドウ或いはアプリケーションをクローズされた
```

```
'画面の幅 (ピクセル単位)
```

```
'画面の高さ (ピクセル単位)
```

```

Var Shared MainForm As Object
Var Shared Picture1 As Object

MainForm.Attach GethWnd
Picture1.Attach GetDlgItem("Picture1")

Var Shared hWnd As Long
Var Shared hDC As Long

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var Ret As Long

    Picture1.Print "この範囲内でマウスを移動 !" & Chr$(13, 10) & "終了はダブルクリック"

    'メモ帳を起動
    Shell "notepad.exe", , 5

    Wait 20

    'メモ帳のハンドルを取得
    hWnd = Api_FindWindow(ByVal 0, "無題 - メモ帳")
    hDC = Api_GetDC(GethWnd)

    'フォームを最前面に設定
    Ret = Api_SetWindowPos(GethWnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE + SWP_NOSIZE)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Ret As Long

    'メモ帳をアイコン化
    Ret = Api_ShowWindow(hWnd, SW_MINIMIZE)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Var Ret As Long

    'メモ帳を元のサイズへ戻す
    Ret = Api_ShowWindow(hWnd, SW_RESTORE)
End Sub

'=====
'=
'=====
Declare Sub Picture1_MouseMove edec1 (ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
Sub Picture1_MouseMove (ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
    Var rct As RECT
    Var px As Long
    Var py As Long
    Var Ret As Long

    '位置の計算
    px = SX! * Api_GetDeviceCaps(hDC, HORZRES) / Picture1.GetWidth
    py = SY! * Api_GetDeviceCaps(hDC, VERTRES) / Picture1.GetHeight

    '新しい位置に設定
    Ret = Api_SetWindowPos(hWnd, 0, px * 0.5, py * 0.5, px * 0.5 + px * 0.25, py * 0.5 + py * 0.25, 0)

```

```

'新しい位置に表示
Ret = Api_GetWindowRect (hWnd, rct)

MainForm.SetWindowText "[ " & Trim$(Str$(rct.Left)) & " ] [ " & Trim$(Str$(rct.Top)) &
"] "
End Sub

'=====
'=
'=====
Declare Sub Picture1_DblClick edec1 ()
Sub Picture1_DblClick()
    Var Ret As Long

    Ret = Api_ReleaseDC (GethWnd, hDC)
    Ret = Api_SendMessage (hWnd, WM_CLOSE, 0, 0)
End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

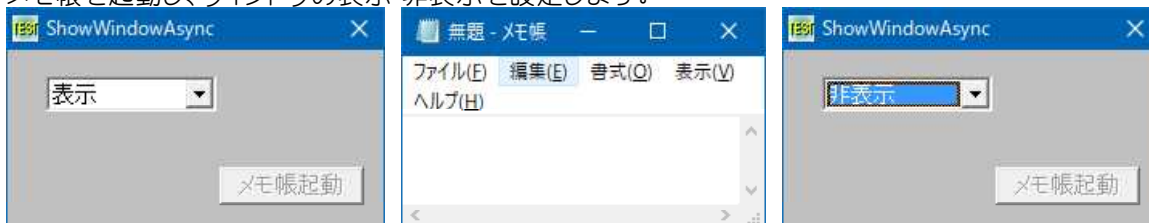
```

別プロセスのウィンドウを表示・非表示

FindWindow クラス名またはキャプションを与えてウィンドウのハンドルを取得

ShowWindowAsync 別のスレッドによって作成されたウィンドウの表示状態を設定

メモ帳を起動し、ウィンドウの表示・非表示を設定します。



```

'=====
'= 別プロセスのウィンドウを表示・非表示
'= (ShowWindowAsync.bas)
'=====
#include "Windows.bi"

' クラス名またはキャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,
ByVal lpWindowName$)

' 別のスレッドによって作成されたウィンドウの表示状態を設定
Declare Function Api_ShowWindowAsync& Lib "user32" Alias "ShowWindowAsync" (ByVal hWnd&,
ByVal nCmdShow&)

#define SW_HIDE 0
#define SW_SHOW 5

Var Shared Combo1 As Object
Var Shared Button1 As Object

Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

' 指定のウィンドウを非表示にし他のウィンドウをアクティブ化
' ウィンドウをアクティブ化し現在の位置とサイズで表示

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Combo1.AddString "表示"
    Combo1.AddString "非表示"
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change ()
    Var ClassName As String
    Var WinHandle As Long
    Var sw As Long
    Var Ret As Long

    'クラス名でウィンドウハンドルを取得
    ClassName = "Notepad"
    WinHandle = Api_FindWindow(ClassName, ByVal 0)

    'ウィンドウハンドルを取得できたときは
    If WinHandle <> 0 Then
        If Combo1.GetCursel <> 0 Then sw = SW_HIDE Else sw = SW_SHOW

        'ウィンドウの表示状態を設定
        Ret = Api_ShowWindowAsync(WinHandle, sw)
    End If
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()

    'コマンドボタンを無効に設定
    Button1.EnableWindow 0

    'メモ帳を起動
    Shell "Notepad.exe", , 5
End Sub

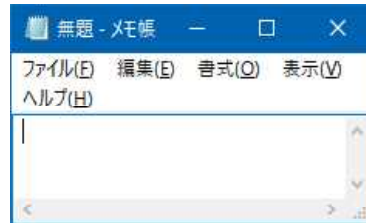
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

別プロセスのウィンドウを元に戻す

FindWindow クラス名またはキャプションを与えてウィンドウのハンドルを取得
ShowWindowAsync 別のスレッドによって作成されたウィンドウの表示状態を設定
SW_RESTORE (9) ウィンドウをアクティブ化し表示。ウィンドウがアイコン化または最大化されているときは元の位置とサイズに

例では、メモ帳を最小化または、最大化する直前の状態に戻します。



```
'=====
'= 別プロセスのウィンドウを元に戻す
'= (ShowWindowAsync2.bas)
'=====
#include "Windows.bi"

' クラス名またはキャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,
ByVal lpWindowName$)

' 別のスレッドによって作成されたウィンドウの表示状態を設定
Declare Function Api_ShowWindowAsync& Lib "user32" Alias "ShowWindowAsync" (ByVal hWnd&,
ByVal nCmdShow&)

#define SW_RESTORE 9

#define vbNullString ByVal 0

Var Shared Text1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()

    ' コマンドボタンを無効に設定
    Button1.EnableWindow 0

    ' メモ帳を起動
    Shell "Notepad.exe",,5
    Text1.ShowWindow -1
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var ClassName As String
    Var hWnd As Long
    Var Ret As Long

    ' クラス名でウィンドウハンドルを取得
    ClassName = "Notepad"
    hWnd = Api_FindWindow(ClassName, vbNullString)

    ' ウィンドウハンドルを取得できたときは
    If hWnd <> 0 Then

        ' ウィンドウの表示状態を設定
        Ret = Api_ShowWindowAsync(hWnd, SW_RESTORE)
    End If
End Sub
```



```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

方眼紙印刷

納品書・請求書などの書式を設定し印刷する場合、該当サイズの方眼紙を作成し寸法(mm)を決めています。適当な大きさのフォームにComboBox、PictureBox、TextBox、BmpButtonを貼り付けます。コンボボックス内の目的サイズの用紙を選択すると、そのサイズに見合った大きさのフォームが表示され、プリンタアイコンをクリックすると方眼紙が印刷されます。使用するプリンタの印字範囲を判断することができ、方眼紙上に線を引き書式を設定します。



A4用紙印刷されたGRID上で納品書の印刷書式を設定。点線1mm、実線10mm(図では点線2mm)
例:

```

ZL = 170 'left
ZR = 1930 'right
ZT = 310 'top
ZB = 770 'bottom
ZC = 20 'radius
PRT1.SETDRAWWIDTH 3
PRT1.line (ZL+ZC,ZT)-(ZR-ZC,ZT)
PRT1.circle(ZL+ZC,ZT+ZC),ZC,,,5,.75
PRT1.circle(ZR-ZC,ZT+ZC),ZC,,,75,0
PRT1.line (ZL,ZT+ZC)-(ZL,ZB-ZC)
PRT1.circle(ZL+ZC,ZB-ZC),ZC,,,25,.5
PRT1.line (ZL+ZC,ZB)-(ZR,ZB)
ZB = 850 : ZR = 1930
PRT1.line (ZR,ZT+ZC)-(ZR,ZB-ZC)
PRT1.circle(ZR-ZC,ZB-ZC),ZC,,,0,.25
ZL = 950
PRT1.line (ZL+ZC,ZB)-(ZR-ZC,ZB)
PRT1.circle(ZL+ZC,ZB-ZC),ZC,,,25,.5
ZT = 770
PRT1.line (ZL,ZT)-(ZL,ZB-ZC)
PRT1.SETDRAWWIDTH 0

```



台帳番号 〇〇〇〇 日付 14/02/01	納品書(領) 2002 年 2 月 1 日	No. 115 株式会社 北海〇〇販売 東京都中央区本町 1 丁目 1 番 1 号 電話 03-1-000-0000、FAX03-1-000-0000 担当 田中			
後藤機械産業株式会社 御中 001-0030 札幌市東区北七条東 5 丁目 2-2					
品名・数量・単価					
電機機	52-1235	1 台	25,800	71,400	
交換機	43-01796	1 台	36,000	344,000	
F.A.X	FAX-00000	10 台	26,800	26,800	40,200
電機機	62-1235	1 台	25,800	71,400	
合計			543,000	22,180	570,700

```

'=====
'= 方眼紙印刷
'= (Gridline.bas)
'=====

#include "Windows.bi"

Var Shared Picture1 As Object
Var Shared Comb1 As Object
Var Shared Text1 As Object
Var Shared BmpButton1 As Object
Var Shared Prt As Object

PrinterObject Prt
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Picture1.Attach GetDlgItem("Picture1")
BmpButton1.Attach GetDlgItem("BmpButton1")
SetFontSize 14

Var Shared pSize(14, 4) As String
Var Shared CrLf As String
Var Shared PaperNum As Integer
Var Shared FrmWidth As Integer
Var Shared FrmHeight As Integer
Var Shared DispWidth As Integer
Var Shared DispHeight As Integer
Var Shared Prm As PrintParam

'=====
'=
'=====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    If PrevInstance Then
        A% = MessageBox(GetWindowText, "このプログラムは実行中です.", 0, 0) : End
    End If

    FrmWidth = GetClientWidth
    FrmHeight = GetClientHeight
    DispWidth = GetDeviceCaps(8)
    DispHeight = GetDeviceCaps(10)
    MoveWindow(DispWidth - FrmWidth) / 2, (DispHeight - FrmHeight - 26) / 2
    ShowWindow -1
    CrLf = Chr$(13, 10)
    BmpButton1.EnableWindow 0

    '-----
    '各サイズ読込
    '-----
    '
    data "88", "1", "1285", "1820", "B6(128.5×182)縦方向"
    data "88", "2", "1820", "1285", "B6(182×128.5)横方向"
    data "13", "1", "1820", "2570", "B5(182×257)縦方向"
    data "13", "2", "2570", "1820", "B5(257×182)横方向"
    data "12", "1", "2570", "3640", "B4(257×364)縦方向"

```

```

data "12", "2", "3640", "2570", "B4 (364×257) 横方向"
data "70", "1", "1050", "1485", "A6 (105×148.5) 縦方向" 'ハガキ (1000×1480)
data "70", "2", "1485", "1050", "A6 (148.5×105) 横方向" 'ハガキ (1480×1000)
data "11", "1", "1485", "2100", "A5 (148.5×210) 縦方向"
data "11", "2", "2100", "1485", "A5 (210×148.5) 横方向"
data " 9", "1", "2100", "2970", "A4 (210×297) 縦方向"
data " 9", "2", "2970", "2100", "A4 (297×210) 横方向"
data " 8", "1", "2970", "4200", "A3 (297×420) 縦方向"
data " 8", "2", "4200", "2970", "A3 (420×297) 横方向"

For i% = 1 To 14
    For j% = 0 To 4
        Read pSize(i%, j%)
    Next
    Combol.AddString pSize(i%, 4)
Next
End Sub

'=====
'= 用紙描画
'=====
Declare Sub Draw edec1 ()
Sub Draw
    SetMapMode 1
    SetFontSize 7
    For x% = 4 To Val(pSize(PaperNum, 2)) / 5 - 5 Step 4
        If x% Mod 20 = 4 Then Col% = 1 Else Col% = 14
        Line(x%, 4)-(x%, Val(pSize(PaperNum, 3)) / 5 - 5),, Col%
    Next
    For y% = 4 To Val(pSize(PaperNum, 3)) / 5 - 5 Step 4
        If y% Mod 20 = 4 Then Col% = 1 Else Col% = 14
        Line(4, y%)-(Val(pSize(PaperNum, 2)) / 5 - 5, y%),, Col%
    Next
End Sub

'=====
'= 用紙選択
'=====
Declare Sub Combol_Change edec1 ()
Sub Combol_Change()
    Text1.ShowWindow 0
    BmpButton1.EnableWindow -1
    PaperNum = Combol.GetCursel + 1
    SetWindowSize Val(pSize(PaperNum, 2)) / 5 + 25, Val(pSize(PaperNum, 3)) / 5 + 70
    FrmWidth = GetClientWidth
    FrmHeight = GetClientHeight
    DispWidth = GetDeviceCaps(8)
    DispHeight = GetDeviceCaps(10)
    MoveWindow (DispWidth - FrmWidth) / 2, (DispHeight - FrmHeight - 26) / 2
    Picture1.ShowWindow -1
    Picture1.Cls
    Picture1.SetWindowSize Val(pSize(PaperNum, 2)) / 5, Val(pSize(PaperNum, 3)) / 5
    Picture1.Draw
End Sub

'=====
'= 印字
'=====
Declare Sub BmpButton1_ON edec1 ()
sub BmpButton1_ON()
    Prt.SetupPrinterMode "SetupPrinter:", Val(pSize(PaperNum, 0)), Val(pSize(PaperNum,
1))
    If Prt.PrintDlg(Prm) = 0 Then Exit Sub
    Prt.SetMapMode 2
    Prt.StartDoc "Sample"
    Prt.StartPage
    Prt.SetFontSize 7
    Prt.SetFontName "MS ゴシック"
    For x% = 0 To Val(pSize(PaperNum, 2)) Step 10
        If x% Mod 100 = 0 Then

```

```

        If x% Mod 100 = 0 Then Prt.Symbol(x% - 25, 10), Format$(x%, "####"), 1, 1
        Prt.Line(x%, 0)-(x%, Val(pSize(PaperNum, 3))),,,, Solid
    Else
        Prt.Line(x%, 0)-(x%, Val(pSize(PaperNum, 3))),,,, Dot
    End If
Next
For y% = 0 To Val(pSize(PaperNum, 3)) Step 10
    If y% Mod 100 = 0 Then
        If y% Mod 100 = 0 Then Prt.Symbol(0, y% - 13), Format$(y%, "####"), 1, 1
        Prt.Line(0, y%)-(Val(pSize(PaperNum, 2)), y%),,,, Solid
    Else
        Prt.Line(0, y%)-(Val(pSize(PaperNum, 2)), y%),,,, Dot
    End If
Next
Prt.EndPage
Prt.EndDoc
Prt.ClosePrinter
End Sub

'=====
'= 終了処理
'=====
Declare Sub MainForm_QueryClose edec1 (Cancel%, ByVal Mode%)
sub MainForm_QueryClose(Cancel%, ByVal Mode%)
    Cancel% = MessageBox(GetWindowText, "終了しますか", 1, 1 )
    If Cancel% = 0 Then End
End Sub

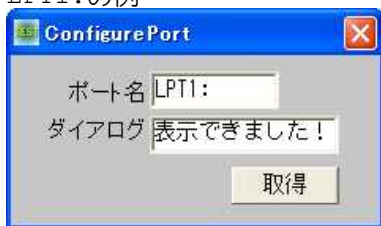
'=====
'=
'=====
While 1
    Waitevent
Wend
Stop
End

```

ポートの構成ダイアログを表示

LPT1:～、COM1:～のダイアログを取得表示します。
ConfigurePort ポートの設定ダイアログを表示

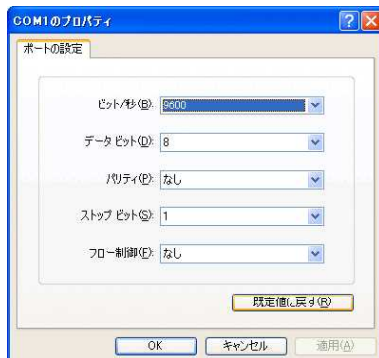
ポート名を入力し取得ボタンをクリックします。
 LPT1:の例



COM1:の例



COM2:の例



```

'=====
'= ポートの構成ダイアログを表示
'= (ConfigurePort.bas)
'=====
#include "Windows.bi"

' ポートの設定ダイアログを表示
Declare Function Api_ConfigurePort Lib "winspool.drv" Alias "ConfigurePortA" (ByVal
pName$, ByVal hWnd&, ByVal pPortName$)

Var Shared Edit1 As Object
Var Shared Text3 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text3.Attach GetDlgItem("Text3") : Text3.SetFontSize 14

'=====
'=
'=====
Declare Sub Mainform_Start edec1 ()
Sub Mainform_Start()
    Edit1.SetWindowText "LPT1:"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var PortName As String
    Var Ret As Long

    'ポート名を指定
    PortName = GetDlgItemText("Edit1")

    'ポートの構成ダイアログを表示
    Ret = Api_ConfigurePort(ByVal 0, GethWnd, PortName)

    '結果を表示
    If Ret <> 0 Then
        Text3.SetWindowText "表示できました！"
    Else
        Text3.SetWindowText "表示できません！"
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

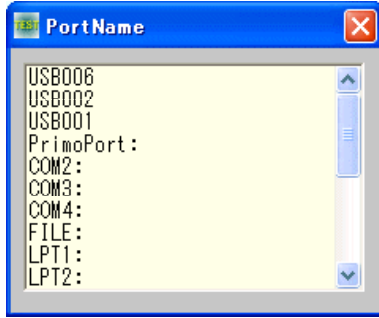
ポート名の列挙(1)

ポート名を列挙します。

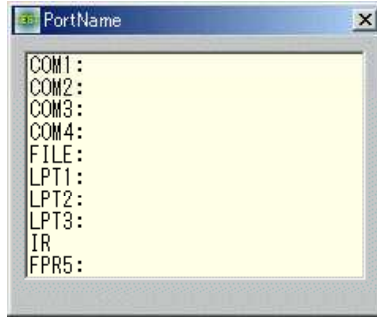
EnumPorts ポート名を列挙する関数

RtlMoveMemory 別の位置にメモリブロックを移動する関数

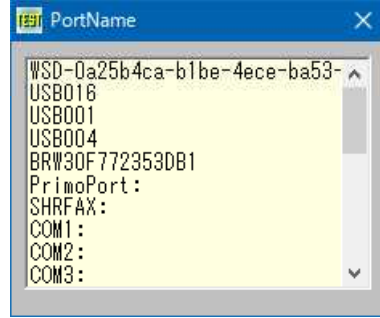
PC-VL100/2 (WindowsXP)



FLORA (Windows2K)



HP (Windows10)



```
'=====
'= ポート名取得
'= (PortName.bas)
'=====
#include "Windows.bi"

' ポート情報を受け取る構造体
Type PORT_INFO_1
    PNAME As Long
End Type

' ポート名を列挙する関数の宣言
Declare Function Api_EnumPorts Lib "winspool.drv" Alias "EnumPortsA" (ByVal pName$,
ByVal Level&, lpPorts As Any, ByVal cbBuf&, pcbNeeded&, pcReturned&)

' ある位置から別の位置にメモリブロックを移動
Declare Function MoveMemory Lib "Kernel32" Alias "RtlMoveMemory" (Dest As Any, Source As
Any, ByVal length&)

Var Shared List1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var Level As Long
    Var Needed As Long
    Var Returned As Long
    Var PortName As String * 64
    Var ct As Long
    Var Ret As Long

    ' 構造体のレベルを指定
    Level = 1

    ' バッファに必要なサイズを取得
    Ret = Api_EnumPorts (Chr$(0), Level, ByVal 0, 0, Needed, Returned)

    ' バッファを確保
    Var Buffer (Needed - 1) As byte

    ' ポート情報を取得
    Ret = Api_EnumPorts (Chr$(0), Level, Buffer(0), Needed, Needed, Returned)

    ' 取得したポート情報の構造体を確保
    Var pil (Returned - 1) As PORT_INFO_1

    For ct = 0 To Returned - 1

        ' データ列から構造体1つ分を抜き取る
        Ret = MoveMemory (pil(ct), Buffer(ct * Len(pil(ct))), Len(pil(ct)))

        ' ポインタから最大字数でポート名を得る
        PortName = String$(64, Chr$(0))
```

```

Ret = MoveMemory (PortName, ByVal pil (ct) .PNAME, Len (PortName))

'ポート名末尾の不要部分を捨てる
PortName = KLeft$(PortName, KInStr(PortName, Chr$(0)) - 1)
List1.AddString PortName
Next CT
End Sub

'=====
'=
'=====

While 1
    WaitEvent
Wend
Stop
End

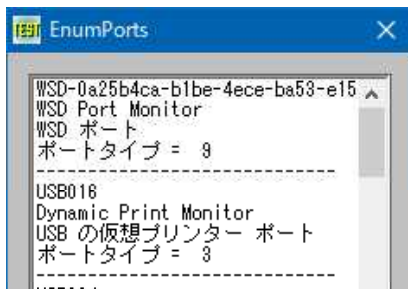
```

ポート名列挙 (II)

ポート名、モニタ名、ポートタイプ等を列挙します。

EnumPorts ローカルポートの列挙

RtlMoveMemory メモリコピー



上下2分割表示しています。

```

'=====
'= ポート名列挙II
'= (EnumPorts.bas)
'=====

#include "Windows.bi"

Type PORT_INFO_1
    pName As String * 256
End Type

Type PORT_INFO_2
    pPortName As Long
    pMonitorName As Long
    pDescription As Long
    fPortType As Long
    Reserved As Long
End Type

#define PORT_TYPE_NET_ATTACHED &H8
#define PORT_TYPE_READ &H2
#define PORT_TYPE_REDIRECTED &H4
#define PORT_TYPE_WRITE &H1
#define vbNullString byval 0
#define vbNullChar chr$(0)

' ローカルポートを列挙
Declare Function Api_EnumPorts Lib "winspool.drv" Alias "EnumPortsA" (ByVal pName$,
ByVal Level%, lpbPorts As byte, ByVal cbBuf%, pcbNeeded%, pcReturned%)

' メモリの指定領域をコピー
Declare Sub Api_RtlMoveMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any,
Source As Any, ByVal Length%)

```

' 文字列をコピーする

```
Declare Function Api_lstrcpy& Lib "kernel32" Alias "lstrcpyA" (ByVal lpString1$, ByVal lpString2&)
```

' 指定された文字列のバイトまたは文字の長さを返す

```
Declare Function Api_lstrlen& Lib "kernel32" Alias "lstrlenA" (ByVal lpString&)
```

```
Var Shared List1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 12
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var Needed As Long
    Var Returned As Long
    Var Buff As String
    Var i As Integer
    Var Ret As Long
```

```
    Ret = Api_EnumPorts(Chr$(0), 2, ByVal 0, 0, Needed, Returned)
```

```
    If Ret = 0 Then
```

```
        Var bytBuff(Needed - 1) As byte
```

' ローカルポートの情報を取得

```
    Ret = Api_EnumPorts(vbNullString, 2, bytBuff(0), Needed, Needed, Returned)
```

```
    If Ret Then
```

```
        If Returned > 0 Then
```

```
            Var pi2(Returned - 1) As PORT_INFO_2
```

```
            Api_RtlMoveMemory pi2(0).pPortName, bytBuff(0), Len(pi2(0)) * (Returned)
```

```
            For i = 0 To Returned - 1
```

' ポート名

```
                Ret = Api_lstrlen(pi2(i).pPortName)
```

```
                Buff = String$(Ret, Chr$(0))
```

```
                Ret = Api_lstrcpy(Buff, pi2(i).pPortName)
```

```
                List1.AddString Buff
```

' モニター名

```
                Ret = Api_lstrlen(pi2(i).pMonitorName)
```

```
                Buff = String$(Ret, Chr$(0))
```

```
                Ret = Api_lstrcpy(Buff, pi2(i).pMonitorName)
```

```
                List1.AddString Buff
```

' 種類

```
                Ret = Api_lstrlen(pi2(i).pDescription)
```

```
                Buff = String$(Ret, Chr$(0))
```

```
                Ret = Api_lstrcpy(Buff, pi2(i).pDescription)
```

```
                List1.AddString Buff
```

' ポートタイプ

```
                List1.AddString "ポートタイプ = " & Str$(pi2(i).fPortType)
```

```
                List1.AddString "-----"
```

```
            Next i
```

```
        End If
```

```
    End If
```

```
End Sub
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop : End
```

ホスト名からIPアドレスを取得

ホスト名からIPアドレスを取得します。

gethostbyname インターネットホスト名に対応するIPアドレスを取得

WSAStartup WinSockを初期化

WSACleanup WinSockのリソースを解放

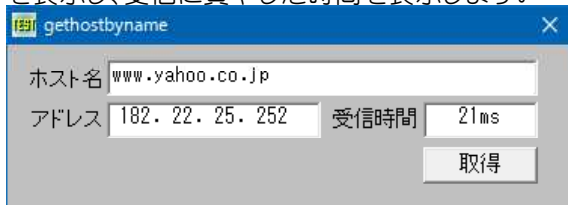
RtlMoveMemory ある位置から別の位置にメモリブロックを移動する関数

IcmpCreateFile ICMPファイルハンドルを開く

IcmpCloseHandle ICMPファイルハンドルを閉じる

IcmpSendEcho ICMPエコーを送信する

初期状態では、www.yahoo.co.jp が入力されています。ホスト名を入力し取得ボタンをクリックすると該当アドレスを表示し、受信に費やした時間を表示します。



※**ICMP** (Internet Control Message Protocol)

IPのエラーメッセージや制御メッセージを転送する**プロトコル**。TCP/IPで接続された**コンピュータ**やネットワーク機器間で、互いの状態を確認するために用いられる。

```
'=====
'= ホスト名からIPアドレスを取得
'=   (GetHostByName.bas)
'=====

#include "Windows.bi"

#define SOCKET_ERROR 0

Type WSADATA
    wVersion                As Integer
    wHighVersion            As Integer
    szDescription(255)      As byte
    szSystemStatus(128)     As byte
    iMaxSockets              As Integer
    iMaxUdpDg               As Integer
    lpVendorInfo             As Long
End Type

Type Hostent
    h_name                   As Long
    h_aliases                As Long
    h_addrTyp                As Integer
    h_length                 As Integer
    h_addr_list              As Long
End Type

Type IP_OPTION_INFORMATION
    TTL                     As byte
    Tos                     As byte
    Flags                   As byte
    OptionsSize             As Long
    OptionsData             As String * 128
End Type

Type IP_ECHO_REPLY
    Address(3)              As byte
    Status                  As Long
    RoundTripTime           As Long
    DataSize                As Integer
    Reserved                As Integer
    ndata                   As Long
    Options                 As IP_OPTION_INFORMATION
End Type
```

' インターネットホスト名に対応するIPアドレスを取得

```
Declare Function Api_gethostbyname& Lib "wsock32" Alias "gethostbyname" (ByVal  
HostName$)
```

' WinSockを初期化

```
Declare Function Api_WSASStartup& Lib "wsock32" Alias "WSASStartup" (ByVal VersionReq&,  
WSADATAReturn As WSADATA)
```

' WinSockのリソースを解放

```
Declare Function Api_WSACleanup& Lib "wsock32" Alias "WSACleanup" ()
```

' ある位置から別の位置にメモリブロックを移動する関数

```
Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (hpfDest As Any, hpfSource As  
Any, ByVal cbCopy&)
```

' ICMPファイルハンドルを開く

```
Declare Function Api_IcmpCreateFile& Lib "icmp" Alias "IcmpCreateFile" ()
```

' ICMPファイルハンドルを閉じる

```
Declare Function Api_IcmpCloseHandle& Lib "icmp" Alias "IcmpCloseHandle" (ByVal HANDLE&)
```

' ICMPエコーを送信する

```
Declare Function Api_IcmpSendEcho& Lib "icmp" Alias "IcmpSendEcho" (ByVal IcmpHandle&,  
ByVal DestAddress&, ByVal RequestData$, ByVal RequestSize&, RequestOptions As  
IP_OPTION_INFORMATION, ReplyBuffer As IP_ECHO_REPLY, ByVal ReplySize&, ByVal Timeout&)
```

```
Var Shared Text(4) As Object
```

```
Var Shared Edit1 As Object
```

```
Var Shared Button1 As Object
```

```
For i = 0 To 4
```

```
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i+1)))
```

```
    Text(i).SetFontSize 14
```

```
Next
```

```
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
' =====
```

```
' =
```

```
' =====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var HostName As String
```

```
    Var hFile As Long
```

```
    Var lpWSADATA As WSADATA
```

```
    Var hHostent As Hostent
```

```
    Var AddrList As Long
```

```
    Var Address As Long
```

```
    Var rIP As String
```

```
    Var OptInfo As IP_OPTION_INFORMATION
```

```
    Var EchoReply As IP_ECHO_REPLY
```

```
    Var Ret As Long
```

```
    HostName = GetDlgItemText("Edit1")
```

```
    Text(3).SetWindowText ""
```

```
    Text(4).SetWindowText ""
```

```
    Ret = Api_WSASStartup(&H101, lpWSADATA)
```

```
    If Api_GetHostByName(HostName & String$(64 - Len(HostName), 0)) <> SOCKET_ERROR Then
```

```
        CopyMemory hHostent.h_name, ByVal Api_GetHostByName(HostName & String$(64 -
```

```
Len(HostName), 0)), Len(hHostent)
```

```
        CopyMemory AddrList, ByVal hHostent.h_addr_list, 4
```

```
        CopyMemory Address, ByVal AddrList, 4
```

```
    End If
```

```
    hFile = Api_IcmpCreateFile()
```

```
    If hFile = 0 Then
```

```
        Text(3).SetWindowText "ファイルハンドルを作成できません！"
```

```

Exit Sub
End If

OptInfo.TTL = 255

If Api_IcmpSendEcho(hFile, Address, String$(32, "A"), 32, OptInfo, EchoReply,
Len(EchoReply) + 8, 2000) Then
    rIP = Str$(EchoReply.Address(0)) & "." & Str$(EchoReply.Address(1)) & "." &
Str$(EchoReply.Address(2)) & "." & Str$(EchoReply.Address(3))
Else
    Text(3).SetWindowText "タイムアウト！"
End If

If EchoReply.Status = 0 Then
    Text(3).SetWindowText rIP
    Text(4).SetWindowText Trim$(Str$(EchoReply.RoundTripTime)) & "ms"
Else
    Text(3).SetWindowText "失敗しました！"
End If

Ret = Api_IcmpCloseHandle(hFile)
Ret = Api_WSACleanup
End Sub

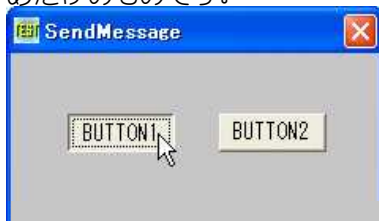
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ボタンクリックシミュレート

コマンドボタンのクリックをシミュレートします。
SendMessage 指定のウィンドウにメッセージを送る
SetFocus ウィンドウにフォーカスを設定
WM_KEYDOWN(&H100) キーボードが押された
WM_KEYUP(&H101) キーボードが解放された

WM_KEYDOWN と WM_KEYUPの間にWait 10 を入れていますが、Button2 押下シミュレート状態を見やすくするためだけのものです。



```

'=====
'= ボタンクリックシミュレート
'= (SendMessage3.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal wMsg&, ByVal wParam&, ByVal lParam&)

' ウィンドウにフォーカスを設定
Declare Function Api_SetFocus& Lib "user32" Alias "SetFocus" (ByVal hWnd&)

#define WM_KEYDOWN &H100 'キーボードのキーが押された
#define WM_KEYUP &H101 'キーボードのキーが離された
#define VK_SPACE &H20 '[Space]

Var Shared Button1 As Object
Var Shared Button2 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

Var Shared Flg As Byte

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Flg = 1
    Ret = Api_SetFocus(Button2.GethWnd)
    Ret = Api_SendMessage(Button2.GethWnd, WM_KEYDOWN, VK_SPACE, 0)
    Wait 10
    Ret = Api_SendMessage(Button2.GethWnd, WM_KEYUP, VK_SPACE, 0)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    If Flg = 1 Then
        A% = MessageBox(GetWindowText, "[Button2]クリックがシミュレートされました！", 0, 3)
    Else
        A% = MessageBox(GetWindowText, "[Button2]が直接クリックされました！", 0, 2)
    End If

    Flg = 0
End Sub

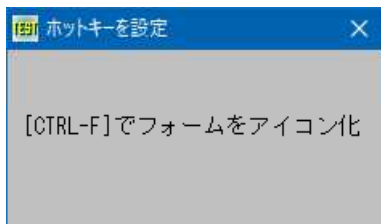
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ホットキーの設定

ホットキーを設定します。例では「Ctrl+F」でMainformを最小化します。
RegisterHotKey ホットキーを設定

UnregisterHotKey ホットキーを解除
 PeekMessage メッセージキューのメッセージを取得
 WaitMessage スレッドにメッセージが無い場合他のスレッドに制御を移す



```
'=====
'= ホットキーを設定
'= (RegisterHotkey.bas)
'=====
#include "Windows.bi"

Type POINTAPI
  x As Long
  y As Long
End Type

Type MSG
  hWnd As Long
  Message As Long
  wParam As Long
  lParam As Long
  ntime As Long
  pt As POINTAPI
End Type

' ホットキー (ショートカットキー) を登録
Declare Function Api_RegisterHotKey& Lib "user32" Alias "RegisterHotKey" (ByVal hWnd&,
ByVal id&, ByVal fsModifiers&, ByVal vk&)

' ホットキー (ショートカットキー) の登録を削除
Declare Function Api_UnregisterHotKey& Lib "user32" Alias "UnregisterHotKey" (ByVal
hWnd&, ByVal id&)

' メッセージキューのメッセージを取得
Declare Function Api_PeekMessage& Lib "user32" Alias "PeekMessageA" (lpMsg As Msg, ByVal
hWnd&, ByVal wParamFilterMin&, ByVal wParamFilterMax&, ByVal wRemoveMsg&)

' スレッドにメッセージがない場合はほかのスレッドに制御を移す
Declare Function Api_WaitMessage& Lib "user32" Alias "WaitMessage" ()

#define WM_HOTKEY &H312
#define MOD_ALT &H1
#define MOD_CONTROL &H2
#define MOD_SHIFT &H4
#define PM_REMOVE &H1
#define vbKeyF &H46

'RegisterHotKey関数で登録されたホットキーが押された
' [ALT]
' [Ctrl]
' [Shift]
'メッセージをキューから削除
' (70) Fキー

Var Shared hKey As Integer

'=====
'=
'=====
Declare Sub ProcessMessages ()
Sub ProcessMessages ()
  Var Message As MSG
  Var Ret As Long

  ' kKeyがTrueになるまでLoop
  Do While Not hKey
```

```

'メッセージを待つ
Ret = Api_WaitMessage

'ホットキーメッセージのチェック
If Api_PeekMessage(Message, GethWnd, WM_HOTKEY, WM_HOTKEY, PM_REMOVE) Then

    'フォームを最小化
    MinimizeWindow
End If

'イベント処理
CallEvent
Loop
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var Ret As Long

    hKey = False

    '「Ctrl + F」ホットキーの登録
    Ret = Api_RegisterHotKey(GethWnd, &HBFFF, MOD_CONTROL, vbKeyF)

    'ホットキーメッセージの処理
    ProcessMessages
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    hKey = True
    Ret = Api_UnregisterHotKey(GethWnd, &HBFFF)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ホットトラッキング

マウスの位置に対する反応を有効・無効に設定します。
SystemParametersInfo システム全体に関するパラメータを取得・設定
SPI_SETHOTTRACKING(&H100F) ホットトラッキング
SPIF_UPDATEINIFILE(&H1) ユーザープロファイルの更新を指定

「ホットトラッキング」とは、マウスカーソルをリストのアイテム上に持っていたとき、アイテムの外観に変化の機会を与える機能
 例では、マウスがメニュー項目上にあるかどうかの監視を有効・無効に設定しています。



```
'=====
'= ホットトラッキング
'= (HotTracking.bas)
'=====
#include "Windows.bi"

#define SPI_SETHOTTRACKING &H100F
#define SPIF_UPDATEINIFILE &H1
#define SPIF_SENDWININICHANGE &H2
#define SPIF_SENDCHANGE SPIF_SENDWININICHANGE

' ホットトラッキング
' ユーザープロファイルの更新を指定

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

Var Shared Radio1 As Object
Var Shared Radio2 As Object
Var Shared Button1 As Object

Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 14
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function Flg_bdecl () As Integer
Function Flg ()
    Flg = Val (Mid$ (GetDlgRadioSelect ("Radio1"), 6)) - 1
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var HotTracking As Long
    Var Ret As Long

    'メニューバーなどのホットトラッキングの有効無効を指定
    HotTracking = Flg

    'メニューバーなどのホットトラッキングの有効無効を設定
    Ret = Api_SystemParametersInfo (SPI_SETHOTTRACKING, 0, ByVal HotTracking,
    SPIF_UPDATEINIFILE Or SPIF_SENDCHANGE)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ポップアップウィンドウの表示・非表示

FindWindow クラス名またはキャプションを与えてウィンドウのハンドルを取得

ShowOwnedPopups 指定したウィンドウが所有するすべてのポップアップウィンドウを表示したり隠したりする

SendMessage ウィンドウにメッセージを送信

例では、メモ帳の持っている検索ポップアップウィンドウを表示・非表示させています。



```
'=====
'= ポップアップウィンドウの表示・非表示
'= (ShowOwnedPopups.bas)
'=====
#include "Windows.bi"
```

' クラス名またはキャプションを与えてウィンドウのハンドルを取得

```
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,
ByVal lpWindowName$)
```

' 指定したウィンドウが所有するすべてのポップアップウィンドウを表示したり隠したりする

```
Declare Function Api_ShowOwnedPopups& Lib "user32" Alias "ShowOwnedPopups" (ByVal hwnd&,
ByVal fShow&)
```

' ウィンドウにメッセージを送信

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)
```

```
#define WM_CLOSE &H10
```

'ウィンドウ或いはアプリケーションをクローズされた

```
Var Shared Comb1 As Object
Var Shared Button1 As Object
```

```
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
Var Shared hWnd As Long
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Comb1.AddString "表示"
    Comb1.AddString "非表示"
```



```

End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    'コマンドボタンの無効を設定
    Button1.EnableWindow False

    'メモ帳を起動
    Shell "Notepad.exe", , 5
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    Var ClassName As String
    Var Index As Long
    Var Ret As Long

    'クラス名でウィンドウハンドルを取得
    ClassName = "Notepad"
    hWnd = Api_FindWindow(ClassName, ByVal 0)

    'ウィンドウハンドルを取得できたときは
    If hWnd <> 0 Then
        Index = 1 - Combo1.GetCursel

        'ポップアップウィンドウの表示状態を設定
        Ret = Api_ShowOwnedPopups(hWnd, Index)
    End If
End Sub

End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

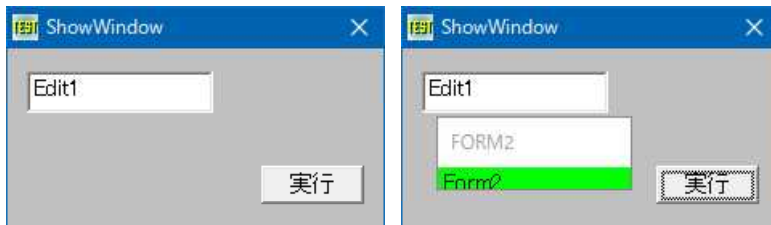
    'メモ帳を終了させる
    Ret = Api_SendMessage(hWnd, WM_CLOSE, 0, 0)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ポップアップフォームを置く

ClientToScreen 指定されたウィンドウ上の点の座標を、クライアント領域の座標からスクリーン座標に変換
ShowWindow 指定されたウィンドウの表示状態を設定
SetWindowPos フォームを常に表示



```
'=====
'= EditBoxの下にポップアップフォームを置く
'= (ShowWindow.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    X    As Long
    Y    As Long
End Type

' 指定されたウィンドウ上の点の座標を、クライアント領域の座標からスクリーン座標に変換
Declare Function Api_ClientToScreen& Lib "user32" Alias "ClientToScreen" (ByVal hWnd&,
lpPoint As POINTAPI)

' 指定されたウィンドウの表示状態を設定
Declare Function Api_ShowWindow& Lib "user32" Alias "ShowWindow" (ByVal hWnd&, ByVal
nCmndShow&)

' フォームを常に表示
Declare Function Api_SetWindowPos& Lib "user32" Alias "SetWindowPos" (ByVal hWnd&, ByVal
hWndInsertAfter&, ByVal X&, ByVal Y&, ByVal cx&, ByVal cy&, ByVal wFlags&)

#define SW_SHOWNOACTIVATE 4
#define HWND_TOPMOST -1
#define SWP_NOSIZE &H1
#define SWP_NOMOVE &H2

Var Shared Form2 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var Ret As Long

    Form2.CreateWindow "Form2", 0
    Form2.SetFontSize 14
    Ret = Api_SetWindowPos(Form2.GethWnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE Or
SWP_NOSIZE)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pt As POINTAPI
    Var Ret As Long

    pt.X = 0
    pt.Y = Edit1.GetHeight
    Ret = Api_ClientToScreen(Edit1.GethWnd, pt)

```

```

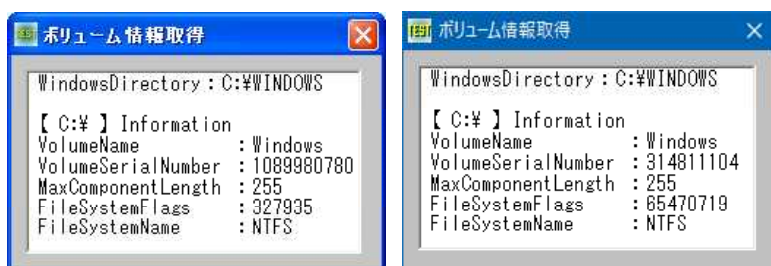
Form2.MoveWindow pt.X, pt.Y
Form2.SetWindowSize Edit1.GetWidth, Edit1.GetHeight * 2
Form2.Symbol(3, 2), "Form2", 1, 1
Ret = Api_ShowWindow(Form2.GethWnd, SW_SHOWNOACTIVATE)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ボリューム情報の取得

Cドライブのボリューム情報を取得します。
 アプリケーションをインストールするPCの識別に使えるかな..
 GetWindowsDirectory Windowsディレクトリを取得
 GetVolumeInformation ボリューム情報の取得



```

'=====
'= ボリューム情報取得
'= (GetVolumeInformation2.bas)
'=====
#include "Windows.bi"

' Windowsディレクトリのパス名を取得
Declare Function Api_GetWindowsDirectory& Lib "Kernel32" Alias "GetWindowsDirectoryA"
    (ByVal lpBuffer$, ByVal nSize&)

' ルート ディレクトリが呼び出しで指定されたファイル システムとボリュームについての情報を返す
Declare Function Api_GetVolumeInformation& Lib "Kernel32" Alias "GetVolumeInformationA"
    (ByVal RootPathName$, ByVal VolNameBuff$, ByVal VolNameSize&, VolSerialNum&,
    MaxComponentLen&, FileSysFlag&, ByVal FileSysNameBuff$, ByVal FileSysNameSize&)

#define MAX_PATH 260

Var Shared Text1 As Object
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var WinDir As String
    Var WkDir As String
    Var VolName As String
    Var VolSerNum As Long
    Var MaxCompoLen As Long
    Var FileSysFlags As Long
    Var FileSysNameBuff As String
    Var txt As String
    Var Ret As Long

```

```

'Windowsがインストールされているドライブを取得
WinDir = space$(255)
Ret = Api_GetWindowsDirectory(WinDir, Len(WinDir))
WinDir = Left$(WinDir, InStr(WinDir, Chr$(0)) - 1)

'ボリューム情報の取得にはルートパスにする
WkDir = Left$(WinDir, 3)

'ボリューム情報取得
VolName = Space$(MAX_PATH)
FileSysNameBuff = Space$(MAX_PATH)
Ret = Api_GetVolumeInformation(WkDir, VolName, MAX_PATH, VolSerNum, MaxCompoLen,
FileSysFlags, FileSysNameBuff, MAX_PATH)

'文字列にはNullが入ってるのでNullをカット
VolName = Left$(VolName, InStr(VolName, Chr$(0)) - 1)
FileSysNameBuff = Left$(FileSysNameBuff, InStr(FileSysNameBuff, Chr$(0)) - 1)

'TextBoxにボリューム情報をセット
txt = ""
txt = txt & " WindowsDirectory:" & WinDir & Chr$(13, 10)
txt = txt & Chr$(13, 10)
txt = txt & "[ " & WkDir & " ]Information" & Chr$(13, 10)
txt = txt & " VolumeName          :" & VolName & Chr$(13, 10)
txt = txt & " VolumeSerialNumber :" & Trim$(Str$(VolSerNum)) & Chr$(13, 10)
txt = txt & " MaxComponentLength :" & Trim$(Str$(MaxCompoLen)) & Chr$(13, 10)
txt = txt & " FileSystemFlags      :" & Trim$(Str$(FileSysFlags)) & Chr$(13, 10)
txt = txt & " FileSystemName       :" & FileSysNameBuff & Chr$(13, 10)

Text1.SetWindowText txt
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

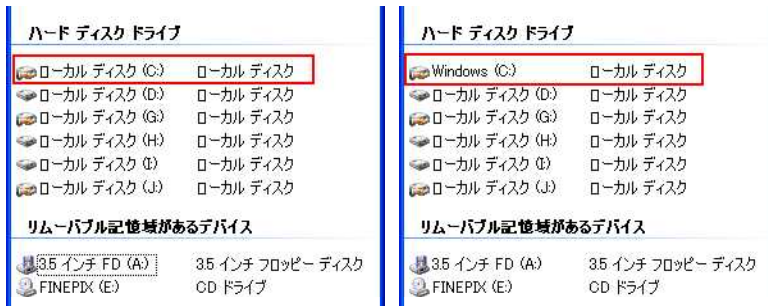
ボリューム名の取得と設定

GetVolumeInformation ファイルシステムの情報を取得する関数
SetVolumeLabel ボリューム名の設定

例ではcドライブのボリューム名(ボリュームラベル)を初期状態のローカル ディスクからWindowsに変更しています。
エディットボックスにボリューム名を入力し『設定』ボタンをクリックします。



Windowsに変更して『設定』ボタンクリック 変更された状態の確認



```
'=====
'= ボリューム名の取得と設定
'= (SetVolumeLabel.bas)
'=====
#include "Windows.bi"

' 利用可能ディスクドライブ取得
Declare Function Api_GetLogicalDrives& Lib "kernel32" Alias "GetLogicalDrives" ()

' ファイルシステムの情報を取得する関数
Declare Function Api_GetVolumeInformation& Lib "kernel32" Alias "GetVolumeInformationA"
(ByVal RootPathName$, ByVal VolNameBuff$, ByVal VolNameSize&, VolSerialNum&,
MaxComponentLen&, FileSysFlag&, ByVal FileSysNameBuff$, ByVal FileSysNameSize&)

' 指定されたボリュームに、ボリュームラベルを設定
Declare Function Api_SetVolumeLabel& Lib "kernel32" Alias "SetVolumeLabelA" (ByVal
lpRootPathName$, ByVal lpVolumeName$)

#define MAX_PATH 260

Var Shared Edit1 As Object
Var Shared Comb1 As Object
Var Shared Text(3) As Object

For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next i
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 14

Var Shared RootPathName As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Drives = Api_GetLogicalDrives()
    If Drives = 0 Then Exit Sub

    For i = 0 To 25
        If (Drives And 1) = 1 Then
            RootPathName = Chr$(65 + i)
            RootPathName = RootPathName & ":¥"
            Comb1.AddString RootPathName
        End If
        Drives = Drives ¥ 2
    Next i
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var VolumeName As String * MAX_PATH
    Var SerialNumber As Long
```

'利用可能なディスクドライブ取得
'関数が失敗したとき

'A～Zドライブを検索する

'ドライブ名(A～Z)に変換

'ドライブ検索

```

Var MaxCompoLength As Long
Var FileSystemFlags As Long
Var FileSystemName As String * 32
Var Ret As Long

'対象のルートパス名を指定
RootPathName = Left$(Combo1.GetText(Combo1.GetCursel), 3)

'ボリュームに関する情報を取得
Ret = Api_GetVolumeInformation(RootPathName, VolumeName, Len(VolumeName),
SerialNumber, MaxCompoLength, FileSystemFlags, FileSystemName, Len(FileSystemName))

'ボリュームに関する情報を表示
Text(3).SetWindowText Left$(VolumeName, InStr(VolumeName, Chr$(0)) - 1)
Edit1.SetWindowText ""
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var VolumeName As String
    Var Ret As Long

    'ボリューム名を指定
    VolumeName = GetDlgItemText("Edit1")

    'ボリューム名を設定
    Ret = Api_SetVolumeLabel(RootPathName, VolumeName)
End Sub

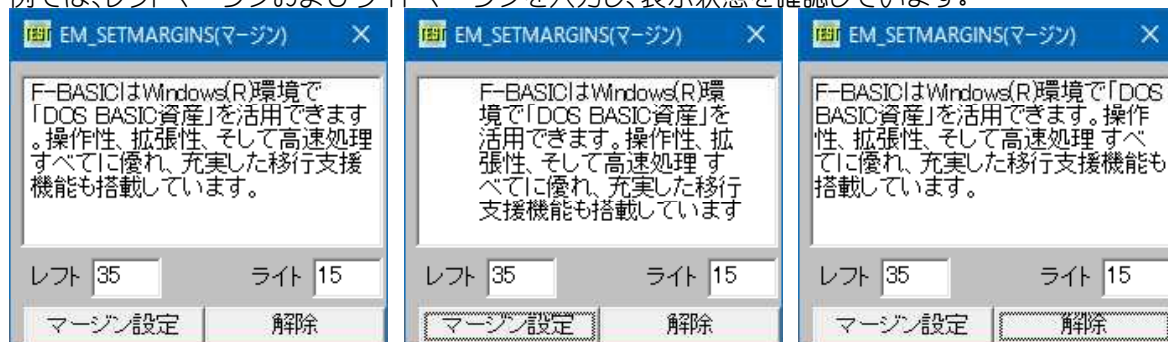
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マージンの設定と解除

エディットボックス内の文字列をマージンを設定して表示します。
SendMessage ウィンドウにメッセージを送信

例では、レフトマージンおよびライトマージンを入力し、表示状態を確認しています。



```

'=====
'= マージンの設定と解除
'= (EM_SETMARGINS.bas)
'=====
#include "Windows.bi"

```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal  
wMsg&, ByVal wParam&, ByVal lParam&)
```

```
#define EM_SETMARGINS &HD3
```

'左右マージンの設定

```
#define EC_LEFTMARGIN 1
```

'左マージンの設定

```
#define EC_RIGHTMARGIN 2
```

'右マージンの設定

```
Var Shared Edit(2) As Object
```

```
Var Shared Text(1) As Object
```

```
Var Shared Button(1) As Object
```

```
For i = 0 To 2
```

```
    If i < 2 Then
```

```
        Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
```

```
        Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1))) :
```

```
Button(i).SetFontSize 14
```

```
    End If
```

```
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
```

```
Next
```

```
Var Shared txt As String
```

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    txt = "F-BASICはWindows(R)環境で「DOS BASIC資産」を活用できます。操作性、拡張性、そして高速処理  
すべてに優れ、充実した移行支援機能も搭載しています。"
```

```
    Edit(0).SetWindowText txt
```

```
End Sub
```

```
'=====
```

```
'= マージン設定
```

```
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var LeftMargin As Integer
```

```
    Var RightMargin As Integer
```

```
    Var Margin As Long
```

```
    Var Ret As Long
```

```
    LeftMargin = Val(Edit(1).GetWindowText)
```

```
    RightMargin = Val(Edit(2).GetWindowText)
```

```
    Margin = RightMargin * 16 ^ 4 + LeftMargin 'Margin = (RightMargin * &H10000) Or
```

```
LeftMargin
```

```
    Ret = Api_SendMessage(Edit(0).GethWnd, EM_SETMARGINS, EC_LEFTMARGIN Or
```

```
EC_RIGHTMARGIN, Margin)
```

```
    Edit(0).SetWindowText txt
```

```
End Sub
```

```
'=====
```

```
'= マージン解除
```

```
'=====
```

```
Declare Sub Button2_on edec1 ()
```

```
Sub Button2_on()
```

```
    Var Ret As Long
```

```
    Ret = Api_SendMessage(Edit(0).GethWnd, EM_SETMARGINS, EC_LEFTMARGIN Or
```

```
EC_RIGHTMARGIN, 0)
```

```
    Edit(0).SetWindowText txt
```

```
End Sub
```

```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

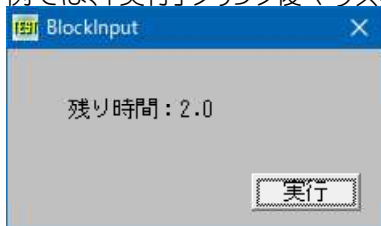
```

マウス・キーボードをブロック

マウス・キーボードをブロックします。

BlockInput マウス・キーボードをブロック

例では、「実行」クリック後マウス、キーボードが一定時間ブロックされ、0.0で解除されます。



```

'=====
'= マウス・キーボードをブロック
'= (BlockInput.bas)
'=====
#include "Windows.bi"

' マウス・キーボードをブロック
Declare Function Api_BlockInput& Lib "user32" Alias "BlockInput" (ByVal fBlock&)

#define API_FALSE 0
#define API_TRUE 1

Var Shared Timer1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Timer1.Attach GetDlgItem("Timer1")
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.SetInterval 50
    Timer1.Enable 0
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Timer1.Enable -1
    Ret = Api_BlockInput(API_TRUE)
End Sub

'=====
'=
'=====

```



```

Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Static Cnt As Long
    Var Ret As Long

    Cnt = Cnt + 1
    If Cnt > 10 Then
        Cnt = 0
        Timer1.Enable 0
        Ret = Api_BlockInput(API_FALSE)
        Text1.SetWindowText ""
    Else
        Text1.SetWindowText "残り時間:" & Format$((10 - Cnt) / 2, "#.##")
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マウスイベントのシミュレート

マウスボタンのイベントをシミュレートします。

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

FindWindow 指定された文字列と一致するクラス名とウィンドウ名を持つトップレベルウィンドウのハンドルを返す

ウィンドウ(フォーム)の位置はクラス名からFindWindowでハンドルを取得しGetWindowRectで左上の座標を読み取っています。

フォーム上のボタン位置は既知の値であり、そのほぼ中点にカーソルを移動しクリック操作をシミュレートしています。MOUSEEVENTF_ABSOLUTEは、絶対座標(スクリーン座標 Left=0・Top=0・Right=65535・Bottom=65535)を要求します。



```

'=====
'= マウスイベントのシミュレート
'= (mouse_event.bas)
'=====
#include "Windows.bi"

```

```

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

```

' ウィンドウの座標をスクリーン座標系で取得

```

Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&, lpRect As RECT)

```

' 指定された文字列と一致するクラス名とウィンドウ名を持つトップレベルウィンドウのハンドルを返す

```

Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$, ByVal lpWindowName$)

```

マウスイベントの生成

```
Declare Sub Api_mouse_event Lib "user32" Alias "mouse_event" (ByVal dwFlags&, ByVal dx&,
ByVal dy&, ByVal cButtons&, ByVal dwExtraInfo&)

#define MOUSEEVENTF_ABSOLUTE &H8000          '絶対座標(スクリーン座標 Left=0・Top=0・Right=35535
                                              '・Bottom=65535)
#define MOUSEEVENTF_LEFTDOWN &H2            'マウス左ボタンDOWN
#define MOUSEEVENTF_LEFTUP &H4              'マウス左ボタンUP
#define MOUSEEVENTF_MIDDLEDOWN &H20         'マウス中ボタンDOWN
#define MOUSEEVENTF_MIDDLEUP &H40           'マウス中ボタンUP
#define MOUSEEVENTF_MOVE &H1               'マウス移動
#define MOUSEEVENTF_RIGHTDOWN &H8          'マウス右ボタンDOWN
#define MOUSEEVENTF_RIGHTUP &H10           'マウス右ボタンUP

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var rct As RECT
    Var lpClassName As String
    Var hWnd As Long

    lpClassName = "FlA0WCMAINFORM"

    hWnd = Api_FindWindow(lpClassName, ByVal 0)
    Ret = Api_GetWindowRect(hWnd, rct)

    x = (rct.Left + GetWidth - 30) / GetDeviceCaps(8) * 65535    '絶対座標(x)
    y = (rct.Top + GetHeight - 20) / GetDeviceCaps(10) * 65535   '絶対座標(y)

    Wait 100
    Api_mouse_event MOUSEEVENTF_MOVE Or MOUSEEVENTF_ABSOLUTE, x, y, 0, 0
    Api_mouse_event MOUSEEVENTF_LEFTDOWN Or MOUSEEVENTF_LEFTUP, 0, 0, 0, 0
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    A% = MsgBox(GetWindowText, "ボタンが押されました！" & Chr$(13) & Chr$(13) & "終了しま
す！", 0, 2)
    End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

マウスカーソル位置(スクリーン座標)の取得と制限

マウスカーソルのスクリーン座標を一定時間毎に取得し表示します。ついでにマウスの移動範囲をフォーム上に限定してみます。

GetCursorPos スクリーン座標取得関数

ClipCursor マウスの移動範囲を限定



TimerとButtonを貼り付けま
初期状態および「ScreenPos」をクリックした場合
マウスカーソルがフォーム外にある場合は、スクリーン座標のみ表示
「ClipPos」をクリックした場合マウスの移動範囲はフォーム上に限定されます。
マウスカーソルがフォーム内にある場合(フォーム内の座標/スクリーン座標)



```
'=====
'= スクリーン上のマウスカーソルの位置を取得
'= (GetCursorPos.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    X    As Long
    Y    As Long
End Type

Type RECT
    Left    As Long
    Top     As Long
    Right   As Long
    Bottom  As Long
End Type

' マウスカーソル(マウスポインタ)の現在の位置に相当するスクリーン座標を取得
Declare Function Api_GetCursorPos& Lib "user32" Alias "GetCursorPos" (pa As POINTAPI)

' マウスの移動範囲を制限
Declare Function Api_ClipCursor& Lib "user32" Alias "ClipCursor" (lpRect As Any)

Var Shared Text(3) As Object
Var Shared Button(1) As Object
Var Shared Timer1 As Object
Var Shared Group1 As Object

For i = 0 To 3
    If i < 2 Then
        Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1)))
        Button(i).SetFontSize 14
    End If
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next i

Timer1.Attach GetDlgItem("Timer1")
Group1.Attach GetDlgItem("Group1") : Group1.SetFontSize 14

Var Shared rc As RECT
```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.SetInterval 10
    Timer1.Enable -1

    ShowWindow -1
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var pa As POINTAPI
    Var Ret As Long

    rc.Left = GetLeft
    rc.Top = GetTop
    rc.Right = rc.Left + GetWidth
    rc.Bottom = rc.Top + GetHeight

    Ret = Api_GetCursorPos(pa)

    If pa.X - rc.Left < 0 Or pa.Y - rc.Top < 0 Or pa.X > rc.Right Or pa.Y > rc.Bottom Then
        Text(0).SetWindowText Format$(pa.X, "#,###")
        Text(1).SetWindowText Format$(pa.Y, "#,###")
    Else
        Text(0).SetWindowText Format$(pa.X - rc.Left, "#,###") & " /" & format$(pa.X,
"#,###")
        Text(1).SetWindowText Format$(pa.Y - rc.Top, "#,###") & " /" & format$(pa.Y,
"#,###")
    End If
End Sub

'=====
'= スクリーン座標
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_ClipCursor(ByVal 0)
    SetWindowText "GetCursorPos"
End Sub

'=====
'= フォーム上の座標に限定
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    Ret = Api_ClipCursor(rc)
    SetWindowText "ClipCursor"
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

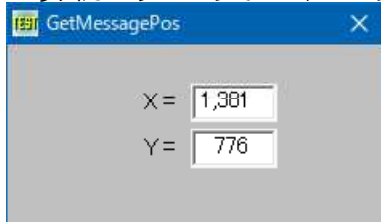
```

マウスカーソルの位置を取得

マウスカーソル座標を取得します。

GetMessagePos メッセージが送られたときのカーソル位置座標を取得

マウスカーソルがフォーム上にある時のスクリーン座標を取得し表示します。



```
'=====
'= マウスカーソル位置を取得
'= (GetMessagePos.bas)
'=====
#include "Windows.bi"

' メッセージが送られたときのカーソル位置座標を取得
Declare Function Api_GetMessagePos& Lib "user32" Alias "GetMessagePos" ()

Type POINTS
    X As Integer
    Y As Integer
End Type

Var Shared Text(3) As Object
For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i+1)))
    Text(i).SetFontSize 14
Next

'=====
'=
'=====
Declare Sub MainForm_MouseMove edec1 ()
Sub MainForm_MouseMove()
    Var pts As POINTS
    Var Ret As Long

    Ret = Api_GetMessagePos()
    pts.X = cint(Ret and &H7FFF)
    pts.Y = cint((Ret and &H7FFF0000) ¥ &H10000)

    Text(2).SetWindowText Format$(pts.X, " #,###")
    Text(3).SetWindowText Format$(pts.Y, " #,###")
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

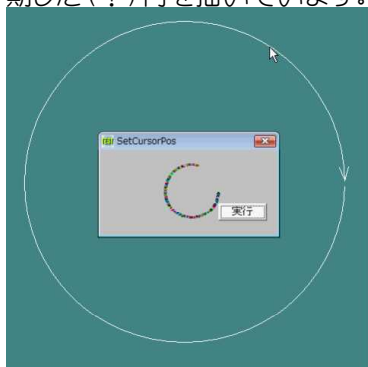
マウスカーソルの位置を変更する

マウスカーソルの位置を変更します。

SetCursorPos マウスカーソルの位置を設定

Sleep 指定時間処理を停止

マウスカーソルを白い円に沿って移動させています。背景によりカーソルが見えにくいことがあるのでフォーム上に同期した(?)円を描いています。



```
'=====
'= マウスカーソルの位置を変更する
'= (SetCursorPos.bas)
'=====
#include "Windows.bi"

' マウスカーソルの位置を変更する
Declare Function Api_SetCursorPos& Lib "user32" Alias "SetCursorPos" (ByVal x&, ByVal y&)

' 指定した時間、処理を停止
Declare Sub Api_Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds&)

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

ShowWindow -1
Cls

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var MiddleX(1) As Long
    Var MiddleY(1) As Long
    Var Radius(1) As Long
    Var TX(1) As Long
    Var TY(1) As Long
    Var Grad As Long
    Var Ret As Long

    Cls

    Do
        MiddleX(0) = GetDeviceCaps(8) / 2
        MiddleY(0) = GetDeviceCaps(10) / 2
        MiddleX(1) = GetWidth / 2
        MiddleY(1) = GetHeight / 2
        Radius(0) = MiddleY(0) / 2
        Radius(1) = MiddleY(1) / 2

        Grad = Grad + 1
        TX(0) = MiddleX(0) + Cos((Grad / 360) * 2 * 3.141) * Radius(0)
        TY(0) = MiddleY(0) + Sin((Grad / 360) * 2 * 3.141) * Radius(0)
        TX(1) = MiddleX(1) + Cos((Grad / 360) * 2 * 3.141) * Radius(1)
        TY(1) = MiddleY(1) + Sin((Grad / 360) * 2 * 3.141) * Radius(1)

        Api_Sleep 5
        CallEvent

        Ret = Api_SetCursorPos(TX(0), TY(0))
        SetDrawWidth 3
        Pset(TX(1), TY(1) - 16), Int(Rnd(1) * 15)
```

```

Loop Until Grad > 360
End Sub

```

```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マウスカーソルの移動領域を取得

マウスカーソルの移動領域取得と制限をします。

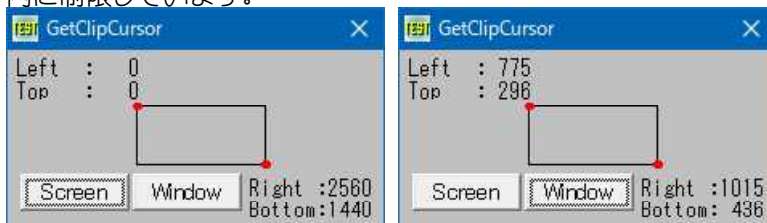
GetClipCursor マウスカーソルの移動領域を取得

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

ClipCursor マウスの移動範囲を制限

GetDesktopWindow Windowsのデスクトップ ウィンドウを識別

左:デスクトップの移動領域取得 右:ウインドウ(フォーム)の移動領域取得/ClipCursorでマウス移動はフォーム内に制限しています。



```

'=====
'= マウスカーソルの移動領域を取得
'= (GetClipCursor.bas)
'=====
#include "Windows.bi"

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

```

' マウスカーソルの移動領域を取得

```
Declare Function Api_GetClipCursor& Lib "user32" Alias "GetClipCursor" (lpRect As RECT)
```

' ウィンドウの座標をスクリーン座標系で取得

```
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&, lpRect As RECT)
```

' マウスの移動範囲を制限

```
Declare Function Api_ClipCursor& Lib "user32" Alias "ClipCursor" (lpRect As RECT)
```

' Windowsのデスクトップ ウィンドウを識別。返されるポインタは、一時的なポインタ。後で使用するために保存しておくことはできない

```
Declare Function Api_GetDesktopWindow& Lib "user32" Alias "GetDesktopWindow" ()
```

```
Var Shared Text1 As Object
```

```
Var Shared Text2 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
```

```

'=====
'=
'=====

```

```

Declare Sub ShowCurrentRect ()
Sub ShowCurrentRect ()
    Var rct As RECT
    Var Ret As Long

    Ret = Api_GetClipCursor(rct)
    Text1.SetWindowText Format$(rct.Left, "Left :####") & Chr$(13) & Format$(rct.Top
, "Top :####")
    Text2.SetWindowText Format$(rct.Right, "Right :####") & Chr$(13) &
Format$(rct.Bottom, "Bottom:####")
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var Ret As Long

    ShowCurrentRect

    Line(80, 34) - (160, 70), , , b
    SetDrawWidth 5
    Pset(80, 34), 5
    Pset(160, 70), 5
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var rct As RECT
    Var Ret As Long

    Ret = Api_GetWindowRect(Api_GetDesktopWindow(), rct)
    Ret = Api_ClipCursor(rct)
    ShowCurrentRect
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Var rct As RECT
    Var Ret As Long

    Ret = Api_GetWindowRect(GethWnd, rct)
    Ret = Api_ClipCursor(rct)
    ShowCurrentRect
End Sub

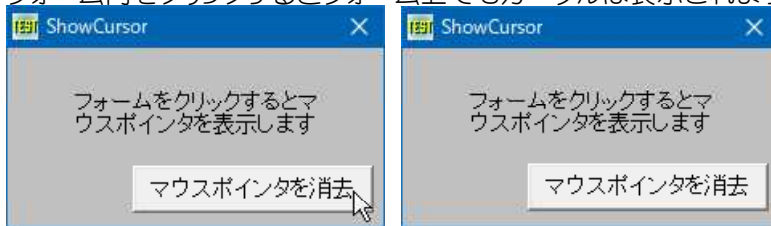
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マウスカーソルの表示・非表示

フォーム上のマウスカーソルの表示および非表示を設定します。
ShowCursor マウスポインターの表示/非表示

ボタンをクリックすると、マウスが非表示になります。フォームを外れた場合、マウスカーソルは表示されます。フォーム内をクリックするとフォーム上でもカーソルは表示されます。



```
'=====
'= マウスカーソルの表示/非表示
'= (ShowCursor.bas)
'=====
#include "Windows.bi"

' マウスカーソルの表示・非表示
Declare Function Api_ShowCursor& Lib "user32" Alias "ShowCursor" (ByVal bShow&)

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var Ret As Long
    Ret = Api_ShowCursor(False)
End Sub

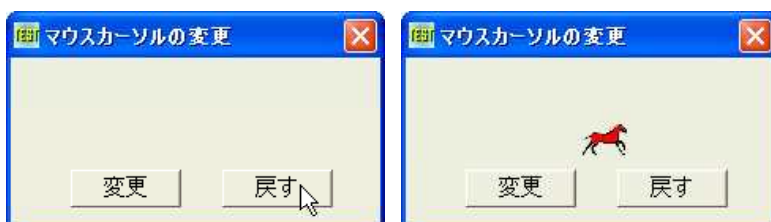
'=====
'=
'=====
Declare Sub MainForm_Click edec1 ()
Sub MainForm_Click ()
    Var Ret As Long
    Ret = Api_ShowCursor(True)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

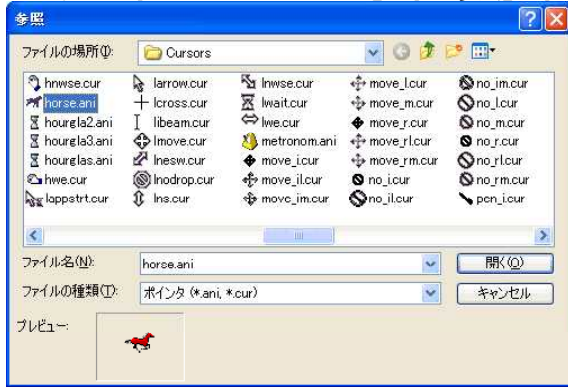
マウスカーソルの変更 (1)

ここでは、マウスカーソルをアニメーション形式に変更してみます。

GetWindowsDirectory Winodws フォルダのパス名を取得
CopyIcon カーソルのコピーを作成
LoadCursorFromFile カーソルをファイルから取得
SetSystemCursor システムのマウスカーソルを変更
GetCursor 現在使用しているマウスカーソルのハンドルを取得



カーソルは『コントロールパネル』→『マウス』→『ポインタ』→『参照』で見ることができます。



```
'=====
'= マウスカーソルの変更
'= (SetCursor.bas)
'=====
#include "Windows.bi"

' Windowsディレクトリのパス名を取得
Declare Function Api_GetWindowsDirectory& Lib "Kernel32" Alias "GetWindowsDirectoryA"
(ByVal lpBuffer$, ByVal nSize&)

' カーソルのコピーを作成
Declare Function Api_CopyIcon& Lib "user32" Alias "CopyIcon" (ByVal hIcon&)

' カーソルをファイルから取得
Declare Function Api_LoadCursorFromFile& Lib "user32" Alias "LoadCursorFromFileA" (ByVal
lpFileName$)

' システムのマウスカーソルを変更
Declare Function Api_SetSystemCursor& Lib "user32" Alias "SetSystemCursor" (ByVal hcur&,
ByVal id&)

' 現在使用しているマウスカーソルのハンドルを取得
Declare Function Api_GetCursor& Lib "user32" Alias "GetCursor" ()

#define OCR_NORMAL 32512                                'Windows組込ビットマップ

Var Shared CurrenthCurs As Long
Var Shared TempCurs As Long
Var Shared NewhCurs As Long

Var Shared Button1 As Object
Var Shared Button2 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var myDir As String
    Var lDir As Long
    Var Ret As Long

    myDir = Space$(255)

    '現在のカーソルを取得しコピーを作成
    CurrenthCurs = Api_GetCursor()
    TempCurs = Api_CopyIcon(CurrenthCurs)

    'Windowsディレクトリのパス名を取得
    lDir = Api_GetWindowsDirectory(myDir, 255)
```

'Horse.aniまでのパスを取得(Windows10では無くなったようです)

```
myDir = Left$(myDir, lDir) & "%cursors%horse.ani"
```

'カーソルをファイルから取得

```
NewhCurs = Api_LoadCursorFromFile(myDir)
```

'システムのマウスカーソルを変更

```
Ret = Api_SetSystemCursor(NewhCurs, OCR_NORMAL)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button2_on edec1 ()
```

```
Sub Button2_on()
```

```
Var Ret As Long
```

'システムカーソルを元に戻す

```
Ret = Api_SetSystemCursor(TempCurs, OCR_NORMAL)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

マウスカーソルの変更(II)

ここでは、マウスカーソルをアニメーション形式に変更してみます。

GetWindowsDirectory Winodws フォルダのパス名を取得

CopyIcon カーソルのコピーを作成

LoadCursorFromFile カーソルをファイルから取得

SetCursor マウスカーソルの形状を設定

SetSystemCursor システムのマウスカーソルを変更

GetCursor 現在使用しているマウスカーソルのハンドルを取得

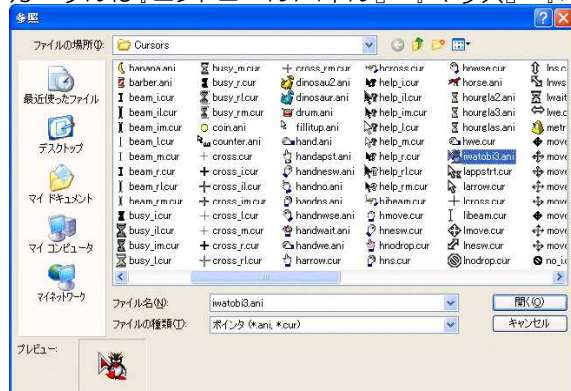
変更した場合



戻した場合



カーソルは『コントロールパネル』→『マウス』→『ポインタ』→『参照』で見ることができます。



```

'=====
'= マウスカーソルの変更
'= (SetCursor.bas)
'=====
#include "Windows.bi"

' Windowsディレクトリのパス名を取得
Declare Function Api_GetWindowsDirectory& Lib "Kernel32" Alias "GetWindowsDirectoryA"
(ByVal lpBuffer$, ByVal nSize&)

' カーソルのコピーを作成
Declare Function Api_CopyIcon& Lib "user32" Alias "CopyIcon" (ByVal hIcon&)

' カーソルをファイルから取得
Declare Function Api_LoadCursorFromFile& Lib "user32" Alias "LoadCursorFromFileA" (ByVal
lpFileName$)

' マウスカーソルの形状を設定
Declare Function Api_SetCursor& Lib "user32" Alias "SetCursor" (ByVal hCursor&)

' システムのマウスカーソルを変更
Declare Function Api_SetSystemCursor& Lib "user32" Alias "SetSystemCursor" (ByVal hcur&,
ByVal id&)

' 現在使用しているマウスカーソルのハンドルを取得
Declare Function Api_GetCursor& Lib "user32" Alias "GetCursor" ()

#define OCR_NORMAL 32512                                'Windows組込ビットマップ

Var Shared CurrenthCurs As Long
Var Shared TempCurs As Long
Var Shared NewhCurs As Long

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var myDir As String
    Var lDir As Long
    Var Ret As Long

    myDir = Space$(255)
    CurrenthCurs = Api_GetCursor()
    TempCurs = Api_CopyIcon(CurrenthCurs)
    lDir = Api_GetWindowsDirectory(myDir, 255)
    myDir = Left$(myDir, lDir) & "¥cursors¥iwatobi3.ani" ' (Windows10では無くなったようです)
    NewhCurs = Api_LoadCursorFromFile(myDir)
    Ret = Api_SetSystemCursor(NewhCurs, OCR_NORMAL)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    Ret = Api_SetSystemCursor(TempCurs, OCR_NORMAL)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

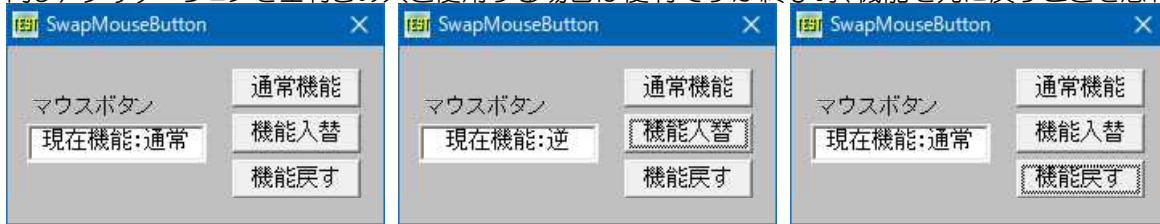
マウスボタンの左右機能入替

マウスボタン左右の機能を入れ替えます。

SwapMouseButton マウスの左と右の機能を入れ替える

左:通常のボタン機能・中:機能を入れ替えた状態(これ以降は当然右ボタンが機能しています)・右:機能を元の戻す(戻す前は右ボタンでクリック)

同じアプリケーションを左利きの人と使用する場合は便利ですが終了時、機能を元に戻すことを忘れてはなりません！



SwapMouseButton(1) 機能を入れ替える
SwapMouseButton(0) 元に戻す

```
'=====
'= マウスボタンの左右機能入替
'= (SwapMouseButton.bas)
'=====
#include "Windows.bi"

' マウスの左と右の機能を入れ替える
Declare Function Api_SwapMouseButton Lib "user32" Alias "SwapMouseButton" (ByVal
bSwap&)

Var Shared Text2 As Object
Text2.Attach GetDlgItem("Text2")

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()

    Text2.SetWindowText "現在機能:通常"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()

    Text2.SetWindowText "現在機能:通常"
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Var Ret As Long

    Ret = Api_SwapMouseButton(1)
    Text2.SetWindowText "現在機能:逆"
End Sub

'=====
'=
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on ()
    Var Ret As Long

    Ret = Api_SwapMouseButton(0)
```

```

Text2.SetWindowText "現在機能:通常"
End Sub

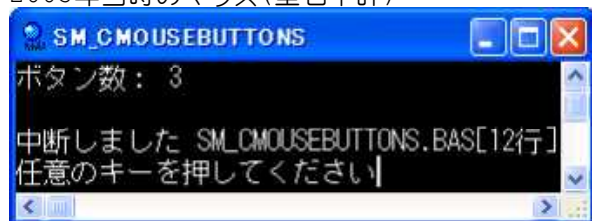
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

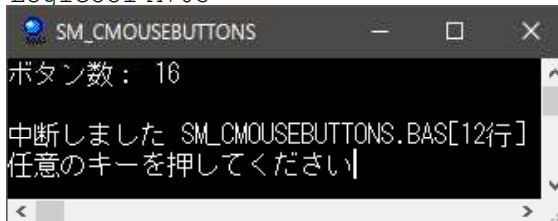
マウスのボタン数取得

GetSystemMetrics システムの現在の構成を取得
SM_CMOUSEBUTTONS (43) マウスボタンの数

2005年当時のマウス(型名不詳)



Logicool M705



```

'=====
'= マウスのボタン数取得
'= (SM_CMOUSEBUTTONS.bas)
'=====

```

' さまざまなシステムメトリックの値(表示要素の幅と高さ)とシステムの現在の構成を取得

```

Declare Function Api_GetSystemMetrics Lib "user32" Alias "GetSystemMetrics" (ByVal nIndex As Integer) As Integer

```

```

#define SM_CMOUSEBUTTONS 43

```

' マウスボタンの数(マウスがないとき0)

```

Print "ボタン数:" & Str$(Api_GetSystemMetrics(SM_CMOUSEBUTTONS))

```

```

Stop
End

```

マウスホイールのスクロール量を設定

マウスホイールのスクロール量を取得・設定します。

SystemParametersInfo システム全体に関するパラメータを取得・設定

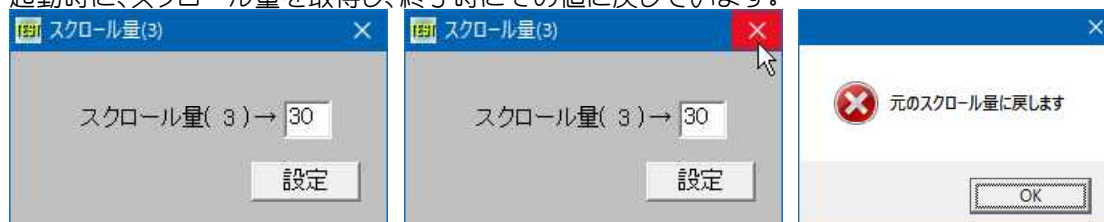
SPI_GETWHEELSCROLLLINES (104) マウスホイールの回転によってスクロールされる行数の値を取得

SPI_SETWHEELSCROLLLINES (105) マウスホイールの回転によってスクロールされる行数の値を設定

SPIF_SENDWININICHANGE (&H2) 全てのアプリケーションに通知して更新

SPIF_UPDATEINIFILE (&H1) ユーザープロファイルの更新を指定

起動時に、スクロール量を取得し、終了時にその値に戻しています。



```

'=====
'= マウスホイールのスクロール量を設定
'= (SetWheelScrollLines.bas)
'=====
#include "Windows.bi"

#define SPI_GETWHEELSCROLLLINES 104      'マウスホイールの回転によってスクロールされる行数の値
                                           'を取得 (WindowsNT以降)
#define SPI_SETWHEELSCROLLLINES 105      'マウスホイールの回転によってスクロールされる行数の値
                                           'を設定 (WindowsNT以降)
#define SPIF_SENDWININICHANGE &H2        '全てのアプリケーションに通知して更新
#define SPIF_UPDATEINIFILE &H1          'ユーザープロファイルの更新を指定

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared ScrollValueOrg As Long
Var Shared ScrollValue As Long

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var Ret As Long

    Ret = Api_SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, ScrollValueOrg, 0)
    Text1.SetWindowText "スクロール量 (" & Str$(ScrollValueOrg) & ") →"

    SetWindowText "スクロール量 (" & Trim$(Str$(ScrollValueOrg)) & ") "
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, ScrollValue, 0)
    Text1.SetWindowText "スクロール量 (" & Str$(ScrollValue) & ") →"

    'ホイールの1目盛りでスクロールする量を指定
    ScrollValue = Val(Edit1.GetWindowText)

    'ホイールの1目盛りでスクロールする量を設定
    Ret = Api_SystemParametersInfo(SPI_SETWHEELSCROLLLINES, ScrollValue, ByVal CLng(0),
    SPIF_UPDATEINIFILE Or SPIF_SENDCHANGE)

    SetWindowText "スクロール量 (" & Trim$(Str$(ScrollValue)) & ") "
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

    Ret = Api_SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, ScrollValue, 0)
    Ret = Api_SystemParametersInfo(SPI_SETWHEELSCROLLLINES, ScrollValueOrg, ByVal CLng

```

```

(0), SPIF_UPDATEINIFILE Or SPIF_SENDCHANGE)

Text1.SetWindowText "スクロール量 (" & Str$(ScrollValue) & " ) →"
Edit1.SetWindowText Trim$(Str$(ScrollValueOrg))
A% = MsgBox("", "元のスクロール量に戻します", 0, 0)
End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マウスポインタの移動速度取得と設定

マウスポインタの移動速度取得と設定を実行します。

SystemParametersInfo システム全体に関するパラメータを取得・設定

SPI_GETMOUSESPEED (112) マウスポインタのスピードを取得

SPI_SETMOUSESPEED (113) マウスポインタのスピードを設定

SPIF_SENDWININICHANGE (&H2) 全てのアプリケーションに通知して更新

SPIF_UPDATEINIFILE (&H1) ユーザープロファイルの更新を指定



```

'=====
'= マウスポインタの移動速度取得と設定
'= (MouseSpeed.bas)
'=====
#include "Windows.bi"

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

#define SPI_GETMOUSESPEED 112
#define SPI_SETMOUSESPEED 113
#define SPIF_SENDWININICHANGE &H2
#define SPIF_UPDATEINIFILE &H1

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Speed As Long
    Var Ret As Long

```



```

'ポインタの移動速度を取得
Ret = Api_SystemParametersInfo (SPI_GETMOUSESPEED, 0, Speed, 0)

'ポインタの移動速度を表示
Text1.SetWindowText Str$(Speed)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Speed As Long
    Var Ret As Long

    Speed = Val (Edit1.GetWindowText)
    If Speed < 1 Or Speed > 20 Then
        A% = MsgBox("", "設定範囲:1~20", 0, 2)
        Exit Sub
    End If
End If

'ポインタの移動速度を設定
Ret = Api_SystemParametersInfo (SPI_SETMOUSESPEED, 0, ByVal Speed, SPIF_UPDATEINIFILE
Or SPIF_SENDWININICHANGE)
End Sub

'=====
'=
'=====
Declare Sub Edit1_Change edec1 ()
Sub Edit1_Change()
    Text1.SetWindowText ""
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

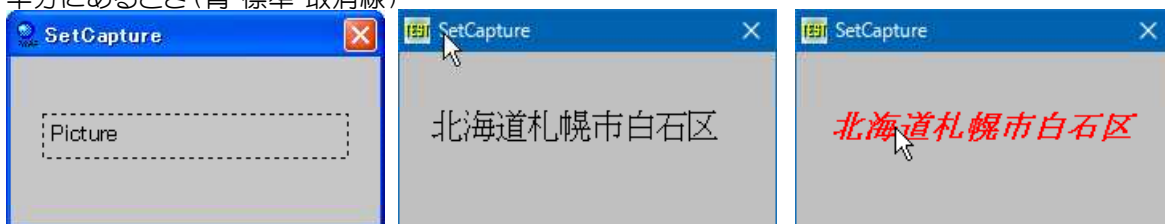
```

マウスホーバーとマウスリープ(1)

マウスホーバー (コントロール上にマウスがある状態) と、マウスリープ (マウスがコントロール上から離れた状態) のテスト

GetCapture マウス入力を受け持っているハンドルを取得
SetCapture 指定のウィンドウにマウスキャプチャを設定
ReleaseCapture マウスのキャプチャを解放
SetFocus ウィンドウにフォーカスを設定

左: フォームに PictureBox を貼り付ける 右: マウスがコントロールの左半分上にあるとき (赤・斜体・ボールド) ・右半分にあるとき (青・標準・取消線)



```

'=====
'= マウスホーバーとマウスリープ
'= (SetCapture.bas)
'=====

```

```

#include "Windows.bi"

' マウス入力を受け持っているハンドルを取得
Declare Function Api_GetCapture& Lib "user32" Alias "GetCapture" ()

' 指定のウィンドウにマウスキャプチャを設定
Declare Function Api_SetCapture& Lib "user32" Alias "SetCapture" (ByVal hWnd&)

' マウスのキャプチャを解放
Declare Function Api_ReleaseCapture& Lib "user32" Alias "ReleaseCapture" ()

' ウィンドウにフォーカスを設定
Declare Function Api_SetFocus& Lib "user32" Alias "SetFocus" (ByVal hWnd&)

Var Shared Picture1 As Object
Picture1.Attach GetDlgItem("Picture1")

Var Shared myStr As String

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    myStr = "北海道札幌市白石区"
    Picture1.SetFontName "MS ゴシック"
    Picture1.SetFontSize 20
    Picture1.Symbol(0, 0), myStr, 1, 1
End Sub

' =====
' =
' =====
Declare Sub Picture1_MouseMove edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal x As Single, ByVal y As Single)
Sub Picture1_MouseMove (ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single,
ByVal y As Single)
    Var Ret As Long

    'ピクチャボックスの左半分にマウスがある場合 (赤・ボールド)
    If X >= 0 And Y >= 0 And X <= Picture1.GetWidth / 2 And Y <= Picture1.GetHeight Then
        Ret = Api_SetCapture (Picture1.Gethwnd)
        Picture1.Cls
        Picture1.SetFontBold -1
        Picture1.SetFontItalic -1
        Picture1.SetFontStrikeOut 0
        Picture1.Symbol(0, 0), myStr, 1, 1, 5

    'ピクチャボックスの右半分にマウスがある場合 (青・取り消し線)
    Else If X > Picture1.GetWidth / 2 And Y >= 0 And X <= Picture1.GetWidth And Y <=
Picture1.GetHeight Then
        Ret = Api_SetCapture (Picture1.Gethwnd)
        Picture1.Cls
        Picture1.SetFontBold 0
        Picture1.SetFontItalic 0
        Picture1.SetFontStrikeOut -1
        Picture1.Symbol(0, 0), myStr, 1, 1, 2

    '通常 (黒・標準)
    Else
        Ret = Api_ReleaseCapture
        Picture1.Cls
        Picture1.SetFontBold 0
        Picture1.SetFontItalic 0
        Picture1.SetFontStrikeOut 0
        Picture1.Symbol(0, 0), myStr, 1, 1

    End If
End Sub

```

```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マウスホバーとマウスリープ (II)

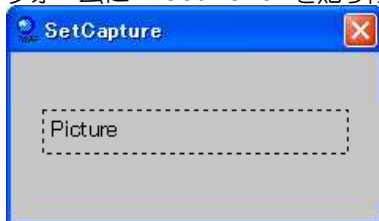
マウスホバー (コントロール上にマウスがある状態) と、マウスリープ (マウスがコントロール上から離れた状態) のテスト

GetCapture マウス入力を受け持っているハンドルを取得

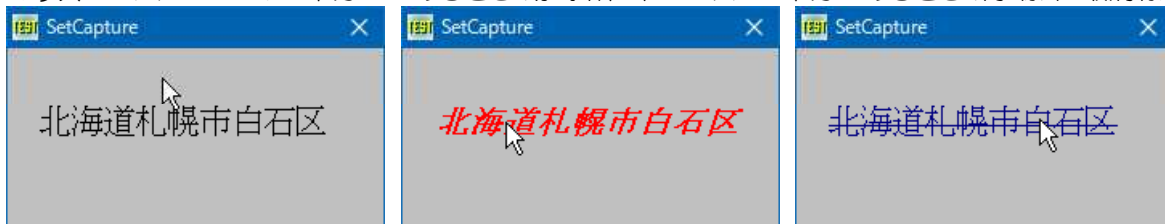
SetCapture 指定のウィンドウにマウスキャプチャを設定

ReleaseCapture マウスのキャプチャを解放

フォームにPictureBoxを貼り付ける



マウスがコントロールの左半分上にあるとき (赤・斜体・ボールド) ・右半分にあるとき (青・標準・取消線)



```

'=====
'= マウスホバーとマウスリープ (II)
'= (SetCapture2.bas)
'=====
#include "Windows.bi"

' マウス入力を受け持っているハンドルを取得
Declare Function Api_GetCapture& Lib "user32" Alias "GetCapture" ()

' 指定のウィンドウにマウスキャプチャを設定
Declare Function Api_SetCapture& Lib "user32" Alias "SetCapture" (ByVal hWnd&)

' マウスのキャプチャを解放
Declare Function Api_ReleaseCapture& Lib "user32" Alias "ReleaseCapture" ()

Var Shared Picture1 As Object

Picture1.Attach GetDlgItem("Picture1")

'=====
'=
'=====
Declare Sub Picture1_MouseMove edec1 (ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)

Sub Picture1_MouseMove (ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
    Var Ret As Long

    If Api_GetCapture() = Picture1.GetHwnd Then
        If (X < 0) Or (X > Picture1.GetWidth) Or (Y < 0) Or (Y > Picture1.GetHeight) Then

```

```

        Ret = Api_ReleaseCapture
        Picture1.SetBackColor RGB(255, 255, 255)
        Picture1.Cls
    End If
Else
    Picture1.SetBackColor RGB(255, 255, 0)
    Ret = Api_SetCapture(Picture1.GethWnd)
    Picture1.Cls
End If
End Sub

'=====
'=
'=====
Declare Sub Picture1_MouseUp edec1 (ByVal Button As Integer, ByVal Shift As Integer, ByVal
X As Single, ByVal Y As Single)

Sub Picture1_MouseUp(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single,
ByVal Y As Single)
    Var Ret As Long

    If Api_GetCapture() <> Picture1.GethWnd Then Ret = Api_SetCapture(Picture1.GethWnd)
End Sub

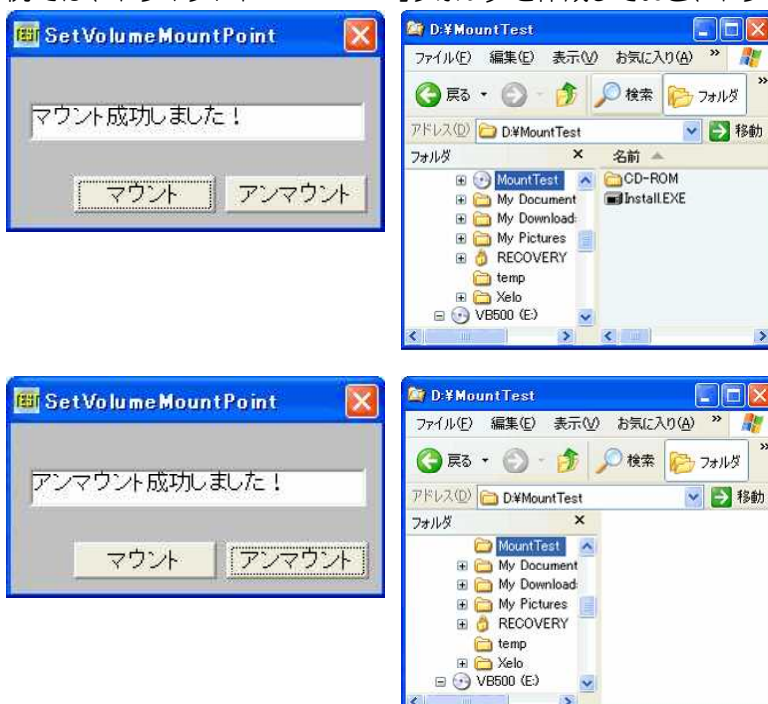
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

マウントポイントにボリュームをマウントする

SetVolumeMountPoint 指定したマウントポイントにボリュームをマウントする
DeleteVolumeMountPoint 指定したマウントポイントからボリュームをアンマウントする
GetVolumeNameForVolumeMountPoint 一意のボリューム名を返す
DefineDosDevice MS-DOS デバイス名の定義、再定義、削除
QueryDosDevice MS-DOS デバイス名に対する現在のマッピングを取得

例では、Dドライブに「MountTest」フォルダを作成しておき、Eドライブ情報をマウント、アンマウントしています。



```

'=====
'= マウントポイントにボリュームをマウントする
'= (SetVolumeMountPoint.bas)
'=====
#include "Windows.bi"

' 指定したマウントポイントにボリュームをマウントする
Declare Function Api_SetVolumeMountPoint& Lib "kernel32" Alias "SetVolumeMountPointA"
(ByVal lpszVolumeMountPoint$, ByVal lpszVolumeName$)

' 指定したマウントポイントからボリュームをアンマウントする
Declare Function Api_DeleteVolumeMountPoint& Lib "kernel32" Alias
"DeleteVolumeMountPointA" (ByVal lpszVolumeMountPoint$)

' 一意のボリューム名を返す
Declare Function Api_GetVolumeNameForVolumeMountPoint& Lib "kernel32" Alias
"GetVolumeNameForVolumeMountPointA" (ByVal lpszVolumeMountPoint$, ByVal
lpszVolumeName$, ByVal cchBufferLength&)

' MS-DOS デバイス名の定義、再定義、削除
Declare Function Api_DefineDosDevice& Lib "kernel32" Alias "DefineDosDeviceA" (ByVal
dwFlags&, ByVal lpDeviceName$, ByVal lpTargetPath$)

' MS-DOS デバイス名に対する現在のマッピングを取得
Declare Function Api_QueryDosDevice& Lib "kernel32" Alias "QueryDosDeviceA" (ByVal
lpDeviceName$, ByVal lpTargetPath$, ByVal ucchMax&)

#define DDD_EXACT_MATCH_ON_REMOVE &H4 'DDD REMOVE DEFINITIONと共に指定すると、完全に一
致するマッピングから、削除するマッピングを判断
#define DDD_RAW_TARGET_PATH &H1 'lpTargetPathに指定されている文字列は、ドライブパス
#define DDD_REMOVE_DEFINITION &H2 '指定のデバイスの指定された定義を削除

Var Shared Text1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'= CDドライブ『E:』を『D:¥MountTest』にマウントする
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var MountPoint As String
    Var VolumeName As String
    Var DeviceName As String
    Var TargetPath As String
    Var DriveLetter As String
    Var NTDevice As String
    Var Ret As Long

    'MS-DOS デバイス名に対する現在のマッピングを取得
    DeviceName = "E:" 'CDドライブ
    TargetPath = String$(255, Chr(0))
    Ret = Api_QueryDosDevice(DeviceName, TargetPath, Len(TargetPath))
    TargetPath = Left$(TargetPath, InStr(TargetPath, Chr$(0)) - 1)

    '一時的に仮想ドライブに関連付ける
    DriveLetter = "I:" '空いてるドライブ指定
    NTDevice = TargetPath
    Ret = Api_DefineDosDevice(DDD_RAW_TARGET_PATH, DriveLetter, NTDevice)

    If Ret Then

        'ボリューム名の取得
        MountPoint = "I:¥"
        VolumeName = String$(255, Chr(0))
        Ret = Api_GetVolumeNameForVolumeMountPoint(MountPoint, VolumeName,

```

```

Len (VolumeName) )

' 関連付けの削除
Ret = Api_DefineDosDevice (DDD_RAW_TARGET_PATH Or DDD_REMOVE_DEFINITION Or
DDD_EXACT_MATCH_ON_REMOVE, DriveLetter, NTDevice)
VolumeName = Left$ (VolumeName, InStr (VolumeName, Chr$ (0)) - 1)

' ボリュームのマウント
MountPoint = "D:¥MountTest¥"
Ret = Api_SetVolumeMountPoint (MountPoint, VolumeName)

If Ret <> 0 Then Text1.SetWindowText "マウント成功しました！" Else Text1.SetWindowText
"マウント失敗しました！"
End If
End Sub

' =====
' =
' =====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Var MountPoint As String          ' マウントポイント
    Var Ret As Long

    MountPoint = "D:¥MountTest¥"

    Ret = Api_DeleteVolumeMountPoint (MountPoint)

    If Ret <> 0 Then Text1.SetWindowText "アンマウント成功しました！" Else Text1.SetWindowText
"アンマウント失敗しました！"
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

マッピングモードの設定

マッピングモードを設定し、フォームサイズに合わせて図形および文字列を拡大縮小描画します。

CreateFontIndirect 論理フォントを作成

TextOut 文字を描画

SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

GetMapMode 現在のマッピングモードを取得

SetMapMode 指定のデバイスコンテキストのマッピングモードを設定

GetWindowExtEx 指定のデバイスコンテキストのエクステントを取得

SetWindowExtEx 指定のデバイスコンテキストのエクステントを設定

GetViewportExtEx 指定のデバイスコンテキストのビューポートのエクステントを取得

SetViewportExtEx 指定のデバイスコンテキストのビューポートのエクステントを設定

Polyline 複数の線分からなる連続した線を描画 (実行後はペンの現在位置が変更される)

GetClientRect ウィンドウのクライアント領域の座標を取得

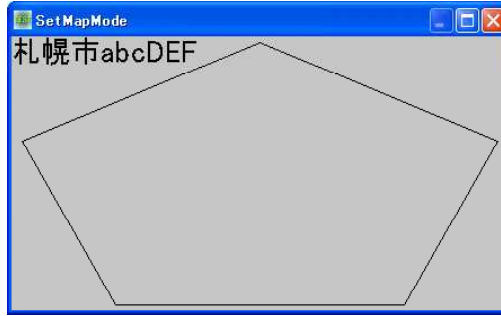
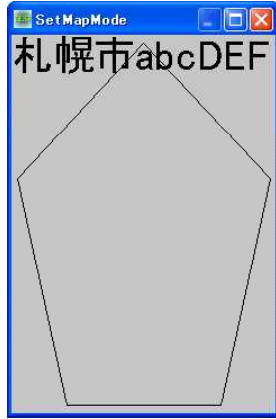
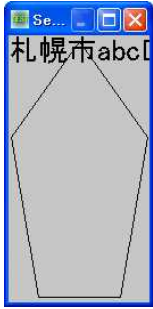
GetPixel 指定された座標のピクセルのRGB値を取得

SetBkColor デバイスコンテキストの背景色を設定

GetDC デバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放

フォームサイズを変更しても同じ比率で図形を拡大縮小します。
文字サイズは、フォームの縦サイズに比例させています。



```
'=====
'= マッピングモードの設定
'= (SetmapMode.bas)
'=====
#include "Windows.bi"

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

Type POINTAPI
    x As Long
    y As Long
End Type

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight      As Long
    lfWidth       As Long
    lfEscapement  As Long
    lfOrientation As Long
    lfWeight      As Long
    lfItalic      As Byte
    lfUnderline   As Byte
    lfStrikeOut   As Byte
    lfCharSet     As Byte
    lfOutPrecision As Byte
    lfClipPrecision As Byte
    lfQuality     As Byte
    lfPitchAndFamily As Byte
    lfFaceName(LF_FACESIZE) As Byte
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 現在のマッピングモードを取得
Declare Function Api_GetMapMode& Lib "gdi32" Alias "GetMapMode" (ByVal hDC&)

' 指定のデバイスコンテキストのマッピングモードを設定
Declare Function Api_SetMapMode& Lib "gdi32" Alias "SetMapMode" (ByVal hDC&, ByVal
```

```
nMapMode&)
```

' 指定のデバイスコンテキストのエクステントを取得

```
Declare Function Api_GetWindowExtEx& Lib "gdi32" Alias "GetWindowExtEx" (ByVal hDC&, lpSize As POINTAPI)
```

' 指定のデバイスコンテキストのエクステントを設定

```
Declare Function Api_SetWindowExtEx& Lib "gdi32" Alias "SetWindowExtEx" (ByVal hDC&, ByVal nX&, ByVal nY&, lpSize As Any)
```

' 指定のデバイスコンテキストのビューポートのエクステントを取得

```
Declare Function Api_GetViewportExtEx& Lib "gdi32" Alias "GetViewportExtEx" (ByVal hDC&, lpSize As POINTAPI)
```

' 指定のデバイスコンテキストのビューポートのエクステントを設定

```
Declare Function Api_SetViewportExtEx& Lib "gdi32" Alias "SetViewportExtEx" (ByVal hDC&, ByVal nX&, ByVal nY&, lpSize As Any)
```

' 複数の線分からなる連続した線を描画 (実行後はペンの現在位置が変更される)

```
Declare Function Api_Polyline& Lib "gdi32" Alias "Polyline" (ByVal hDC&, lpPoint As POINTAPI, ByVal nCount&)
```

' ウィンドウのクライアント領域の座標を取得

```
Declare Function Api_GetClientRect& Lib "user32" Alias "GetClientRect" (ByVal hWnd&, lpRect As RECT)
```

' 指定された座標のピクセルのRGB値を取得

```
Declare Function Api_GetPixel& Lib "gdi32" Alias "GetPixel" (ByVal hDC&, ByVal X&, ByVal Y&)
```

' デバイスコンテキストの背景色を設定

```
Declare Function Api_SetBkColor& Lib "gdi32" Alias "SetBkColor" (ByVal hDC&, ByVal crColor&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
#define MM_ANISOTROPIC 8
```

' 論理単位は、任意にスケーリングされた軸上の任意の単位にマップされる

```
#define MM_HIENGLISH 5
```

' 各論理単位は、0.001インチにマップされる。xの値は右に、yの値は上に向かって増加

```
#define MM_HIMETRIC 3
```

' 各論理単位は、0.01ミリメートルにマップされる。xの値は右に、yの値は上に向かって増加

```
#define MM_ISOTROPIC 7
```

' 論理単位は、等しくスケーリングされた軸上の任意の単位にマップされる。すなわち、x 軸方向の1単位は、y軸方向の1単位と同じになる

```
#define MM_LOENGLISH 4
```

' 各論理単位は、0.01インチにマップされる。xの値は右に、yの値は上に向かって増加

```
#define MM_LOMETRIC 2
```

' 各論理単位は、0.1 ミリにマップされる。xの値は右に、yの値は上に向かって増加

```
#define MM_TEXT 1
```

' 各論理単位は、1デバイスピクセルにマップされる。xの値は右に、yの値は下に向かって増加

```
#define MM_TWIPS 6
```

' 各論理単位は、ポイント数の20分の1にマップされる。xの値は右に、yの値は上に向かって増加

```
Var Shared OldMapMode As Long
Var Shared OldWindowExt As POINTAPI
Var Shared OldViewport As POINTAPI
Var Shared PT(5) As POINTAPI
Var Shared ClientR As RECT
Var Shared hDC As Long
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
```



```

Var Txt As String
Var length As Long
Var LF As LOGFONT
Var rFont As Long
Var Size As Integer
Var BkCol As Long
Var Ret As Long

'5角形要素
PT(0).x = 50
PT(0).y = 2
PT(1).x = 98
PT(1).y = 35
PT(2).x = 79
PT(2).y = 90
PT(3).x = 21
PT(3).y = 90
PT(4).x = 2
PT(4).y = 35
PT(5).x = 50
PT(5).y = 2

Txt = "札幌市abcDEF"
length = Len(Txt)
Size = 10
LF.lfHeight = Size
LF.lfCharSet = 128                                '日本語(全角)対応

Cls

hDC = Api_GetDC(GethWnd)
BkCol = Api_GetPixel(hDC, 0, 0)                    'フォームの背景色取得
Ret = Api_SetBkColor(hDC, BkCol)                  '文字の背景色を設定

OldMapMode = Api_GetMapMode(hDC)                  '現在のMapMode取得
Ret = Api_GetViewportExtEx(hDC, OldViewport)
Ret = Api_GetWindowExtEx(hDC, OldWindowExt)
Ret = Api_GetClientRect(GethWnd, ClientR)

Ret = Api_SetMapMode(hDC, MM_ANISOTROPIC) '拡大縮小を設定できるMapModeにする
Ret = Api_SetWindowExtEx(hDC, 100, 92, ByVal 0) 'Formサイズに対する5角形の比率(100:92)
Ret = Api_SetViewportExtEx(hDC, ClientR.Right, ClientR.Bottom, ByVal 0) '拡大縮小率を
                                         設定する

rFont = Api_CreateFontIndirect(LF)                '論理フォントを作成
Ret = Api_SelectObject(hDC, rFont)                'オブジェクトを選択
Ret = Api_TextOut(hDC, 0, 0, Txt, length) '位置(0,0)文字描画

Ret = Api_Polyline(hDC, PT(0), 6)                  '連続した線を描画
End Sub

'=====
'=
'=====
Declare Sub MainForm_Resize edec1 ()
Sub MainForm_Resize()
    MainForm_Start
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

    Ret = Api_SetMapMode(hDC, OldMapMode)
    Ret = Api_SetViewportExtEx(hDC, OldViewport.x, OldViewport.y, ByVal 0)
    Ret = Api_SetWindowExtEx(hDC, OldWindowExt.x, OldWindowExt.y, ByVal 0)
    Ret = Api_ReleaseDC(GethWnd, hDC)

```

```
End Sub
```

```
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

メッセージキュー内のメッセージの有無を判定

GetInputState 呼び出し側スレッドのメッセージキューの中に、メッセージが存在するかどうかを判断



```
'=====
'= メッセージキュー内のメッセージの有無を判定
'= (GetInputState.bas)
'=====
#include "Windows.bi"
```

' 呼び出し側スレッドのメッセージキューの中に、マウスボタンメッセージまたはキーボードメッセージが存在するかどうかを調べる

```
Declare Function Api_GetInputState& Lib "user32" Alias "GetInputState" ()
```

```
Var Shared bCancel As Integer
```

```
Var Shared Text1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var n As Long
```

' 初期状態

```
bCancel = False
```

' Loop開始

```
Do
```

```
    n = n + 1
```

' マウスおよびキーボードからのメッセージをチェック

```
    If Api_GetInputState() = False Then CallEvent
```

```
        Text1.SetWindowText Str$(n)
```

```
    Loop Until bCancel = True
```

```
    A% = MessageBox(GetWindowText, "メッセージを確認しました！", 0, 2)
```

```
End Sub
```

```

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    bCancel = True
End Sub

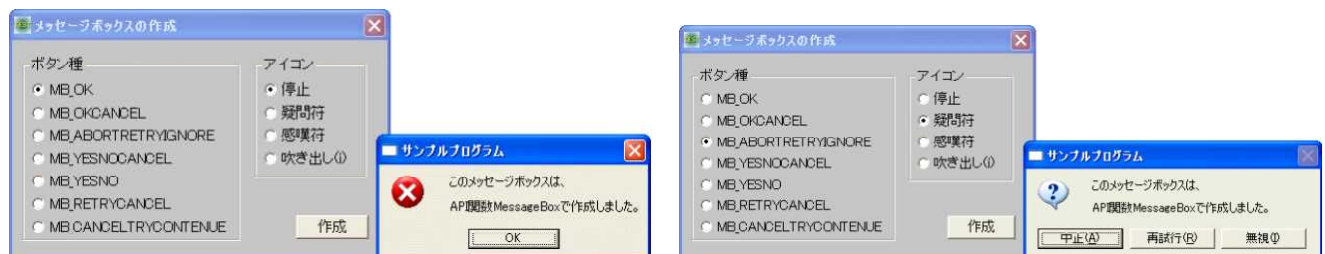
'=====
'=
'=====

While 1
    WaitEvent
Wend
Stop
End

```

メッセージボックスの作成 (I)

APIでメッセージボックスを作成します。ボタン種・アイコンをチェックして作成ボタンをクリックします。
MessageBox メッセージボックスの作成



Windows2000・WindowsXP (Windows9x、WindowsMeでは対応していません！エラー処理をしていないのでCtrl+Alt+Delで停止してください)



```

'=====
'= メッセージボックスの作成
'= (MessageBox.bas)
'=====
#include "Windows.bi"

```

メッセージボックスの作成、表示、操作を行う

Declare Function Api_MessageBox& Lib "user32" Alias "MessageBoxA" (ByVal hWnd&, ByVal lpText\$, ByVal lpCaption\$, ByVal wType&)

第3引数

```

#define MB_OK &H0
#define MB_OKCANCEL &H1
#define MB_ABORTRETRYIGNORE &H2
#define MB_YESNOCANCEL &H3
#define MB_YESNO &H4
#define MB_RETRYCANCEL &H5
#define MB_CANCELTRYCONTINUE &H6

```

```

#define MB_ICONSTOP &H10
#define MB_ICONQUESTION &H20
#define MB_ICONEXCLAMATION &H30
#define MB_ICONINFORMATION &H40

```

```

' [OK]
' [OK] [キャンセル]
' [中止] [再試行] [無視]
' [[はい] [いいえ] [キャンセル]
' [[はい] [いいえ]
' [再試行] [キャンセル]
' [キャンセル] [再実行] [続行] ※Windows2000、
WindowsXP

```

```

' 停止アイコン
' 疑問符アイコン
' 感嘆符アイコン
' 吹き出しに「i」のアイコン

```

```

#define MB_SYSTEMMODAL &H1000

Var Shared Group(1) As Object
Var Shared Radio(10) As Object
Var Shared button1 As Object

For i = 0 To 10
    If i < 2 Then
        Group(i).Attach GetDlgItem("Group" & Trim$(Str$(i + 1))) : Group(i).SetFontSize
14
    End If
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1))) : Radio(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function ButtonNo bdecl () As Long
Function ButtonNo()
    ButtonNo = Val(Mid$(GetDlgRadioSelect("Radio1"), 6)) - 1
End Function

'=====
'=
'=====
Declare Function IconNo bdecl () As Long
Function IconNo()
    IconNo = ((Val(Mid$(GetDlgRadioSelect("Radio8"), 6)) - 8) + 1) * 16
End Function

'=====
'=
'=====
Declare Sub Button1_on edecl ()
Sub Button1_on()
    Var hWnd As Long
    Var wStyle As Long
    Var Text As String
    Var Caption As String
    Var Ret As Long

    hWnd = GethWnd
    Text = "このメッセージボックスは、" & Chr$(13,10) & Chr$(13,10) & "API関数「MessageBox」で作成
しました。"
    Caption = "サンプルプログラム"
    wStyle = ButtonNo Or IconNo Or MB_SYSTEMMODAL

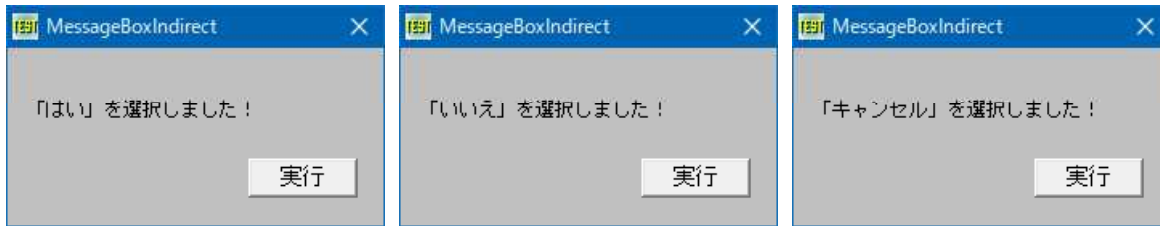
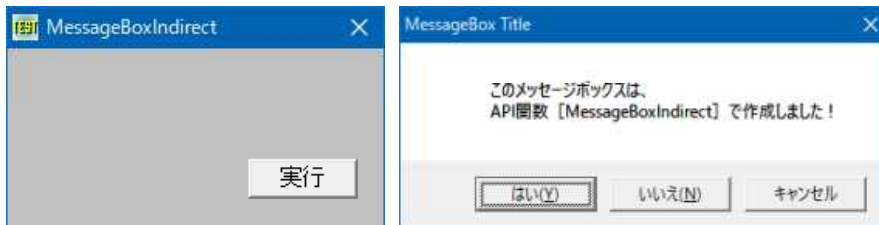
    Ret = Api_MessageBox(hWnd, Text, Caption, wStyle)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メッセージボックスの作成 (II)

APIでメッセージボックスを作成します。ボタン種・アイコンをチェックして作成ボタンをクリックします。
MessageBoxIndirect メッセージボックスの作成



※lpszIconは、今のところ設定できていません(_ _ ;)

```
'=====
'= メッセージボックスの作成 (2)
'= (MessageBoxIndirect.bas)
'=====
#include "Windows.bi"
```

```
Type MSGBOXPARAMS
    cbSize           As Long
    hwndOwner        As Long
    hInstance         As Long
    lpszText          As Long
    lpszCaption       As Long
    dwStyle           As Long
    lpszIcon          As Long
    dwContextHelpId  As Long
    lpfnMsgBoxCallback As Long
    dwLanguageId      As Long
End Type
```

' メッセージボックスの作成、表示、操作を行う

```
Declare Function Api_MessageBoxIndirect& Lib "user32" Alias "MessageBoxIndirectA"
(lpMsgBoxParams As MSGBOXPARAMS)
```

' ウェーブフォームサウンドを再生

```
Declare Function Api_MessageBeep& Lib "user32" Alias "MessageBeep" (ByVal wType&)
```

```
#define MB_ABORTRETRYIGNORE &H2
#define MB_APPLMODAL &H0

#define MB_CANCELTRYCONTINUE &H6
#define MB_COMPOSED &H2

#define MB_DEFAULT_DESKTOP_ONLY &H20000
#define MB_DEFBUTTON1 &H0
#define MB_DEFBUTTON2 &H100
#define MB_DEFBUTTON3 &H200
#define MB_DEFBUTTON4 &H300
#define MB_ERR_INVALID_CHARS &H8
#define MB_HELP &H4000
#define MB_ICONASTERISK &H40
#define MB_ICONEXCLAMATION &H30
#define MB_ICONHAND &H10
#define MB_ICONINFORMATION &H40
#define MB_ICONQUESTION &H20
#define MB_ICONSTOP &H10
#define MB_OK &H0
#define MB_OKCANCEL &H1
#define MB_PRECOMPOSED &H1

#define MB_RETRYCANCEL &H5
#define MB_RIGHT &H80000

' [中止] [再試行] [無視]
' メッセージボックスの親ウィンドウ以外のウィンドウの操作は可能
' [キャンセル] [再実行] [続行]
' 合成文字を使う (MB_PRECOMPOSITEとは組み合わせない)
' (WinNT/2000)
' 最初のボタンをデフォルトプッシュボタンに (デフォルト)
' 2番目のボタンをデフォルトプッシュボタンに
' 3番目のボタンをデフォルトプッシュボタンに
' 4番目のボタンをデフォルトプッシュボタンに
' 不明な文字があったときにエラーを発生する
' ヘルプボタンを追加
' メッセージ (情報)
' 感嘆符アイコン
' システムエラー
' 吹き出しに「i」のアイコン
' 疑問符アイコン
' 停止アイコン
' [OK]
' [OK] [キャンセル]
' 作成済み文字を使う (MB_COMPOSITEとは組み合わせない)
' [再試行] [キャンセル]
' テキストを右寄せ
```

```

#define MB_RTLREADING &H100000 'テキストを右から左へ表示(アラビア語)
#define MB_SETFOREGROUND &H10000 'メッセージボックスをフォアグラウンドウィンドウに
#define MB_SYSTEMMODAL &H1000 'メッセージボックスがWS_EX_TOPMOSTスタイルを持つ以外は、MB_APPLMODALと同じ
                                     '他のアプリケーションを中断せず、他のウィンドウへの入力を禁止

#define MB_TASKMODAL &H2000 'メッセージボックスを最前面ウィンドウ(WS_EX_TOPMOST)で作成

#define MB_USEGLYPHCHARS &H4 '制御文字の代わりにグリフ文字を使う
#define MB_USERICON &H80 '独自のアイコン
#define MB_YESNO &H4 '「はい」「いいえ」
#define MB_YESNOCANCEL &H3 '「はい」「いいえ」「キャンセル」

#define IDABORT 3 '「中止」ボタンを選択
#define IDCANCEL 2 '「キャンセル」ボタンを選択
#define IDCLOSE 8 '「閉じる」ボタンを選択
#define IDCONTINUE 11 '「続行」ボタンを選択
#define IDHELP 9 '「ヘルプ」ボタンを選択
#define IDIGNORE 5 '「無視」ボタンを選択
#define IDNO 7 '「いいえ」ボタンを選択
#define IDOK 1 '「OK」ボタンを選択
#define IDRETRY 4 '「再試行」ボタンを選択
#define IDTRYAGAIN 10 '「再実行」ボタンを選択
#define IDYES 6 '「はい」ボタンを選択
#define LANG_JAPANESE &H11 '日本語

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var mbp As MSGBOXPARAMS
    Var msg As String
    Var Ret As Long

    msg = "このメッセージボックスは、" & Chr$(13, 10) & "API関数[MessageBoxIndirect]で作成しました！"

    mbp.cbSize = Len(mbp)
    mbp.hwndOwner = GethWnd
    mbp.hInstance = GethInst
    mbp.lpszText = StrAdr(msg & Chr$(0))
    mbp.lpszCaption = StrAdr("MessageBox Title" & Chr$(0))
    mbp.dwStyle = MB_YESNOCANCEL Or MB_TOPMOST Or MB_USERICON
    mbp.lpszIcon = 101
    mbp.lpfncallback = 0
    mbp.dwLanguageId = LANG_JAPANESE

    Ret = Api_MessageBeep(MB_ICONINFORMATION)
    Ret = Api_MessageBoxIndirect(mbp)

    If Ret = IDYES Then
        Text1.SetWindowText "「はい」を選択しました！"
    Else If Ret = IDNO Then
        Text1.SetWindowText "「いいえ」を選択しました！"
    Else If Ret = IDCANCEL Then
        Text1.SetWindowText "「キャンセル」を選択しました！"
    End If
End Sub

'=====
'=
'=====
While 1

```

```

WaitEvent
Wend
Stop
End

```

メッセージボックスの作成 (Ⅲ)

APIでメッセージボックスを作成します。ボタン種・アイコンをチェックして作成ボタンをクリックします。

GetLogicalDrives 有効なドライブの情報を取得

MessageBoxEx メッセージボックスの作成、表示、操作を行う

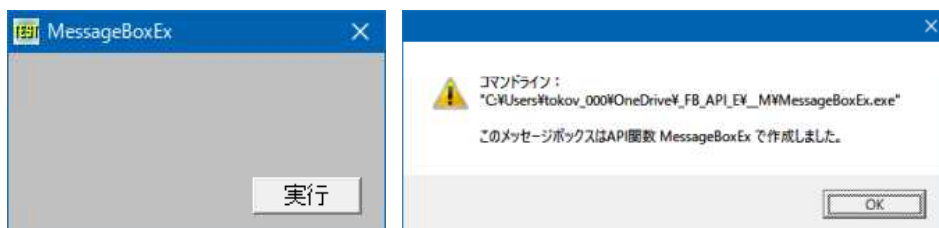
MessageBoxIndirect メッセージボックスの作成、表示、操作を行う

PostQuitMessage アプリケーションを停止するキューをWindowsに送る

GetCommandLine 現在のプロセスのコマンドライン文字列へのポインタを取得

lstrlen 指定された文字列のバイトまたは文字の長さを返す

CopyMemory ある位置から別の位置にメモリブロックを移動



```

'=====
' = メッセージボックスの作成 (Ⅲ)
' =   (MessageBoxEx.bas)
'=====

```

```
#include "Windows.bi"
```

```
#define MB_ICONASTERISK &H40
```

```
#define MB_ICONEXCLAMATION &H30
```

```
#define MAX_PATH = 260
```

'メッセージ (情報)

'感嘆符アイコン

```
Type MSGBOXPARAMS
```

```
    cbSize As Long
```

```
    hwndOwner As Long
```

```
    hInstance As Long
```

```
    lpstrText As Long
```

```
    lpstrCaption As Long
```

```
    dwStyle As Long
```

```
    lpstrIcon As Long
```

```
    dwContextHelpId As Long
```

```
    lpfnMsgBoxCallback As Long
```

```
    dwLanguageId As Long
```

```
End Type
```

' 有効なドライブの情報を取得。ディスクの挿入の有無やネットワークに接続されているかなどは関係なく、現在割り当てられている全てのドライブ一覧を返す

```
Declare Function Api_GetLogicalDrives& Lib "Kernel32" Alias "GetLogicalDrives" ()
```

' メッセージボックスの作成、表示、操作を行う

```
Declare Function Api_MessageBoxEx& Lib "user32" Alias "MessageBoxExA" (ByVal hwnd&,
ByVal lpText$, ByVal lpCaption$, ByVal uType&, ByVal wLanguageId&)
```

' メッセージボックスの作成、表示、操作を行う

```
Declare Function Api_MessageBoxIndirect& Lib "user32" Alias "MessageBoxIndirectA"
```

```

(lpMsgBoxParams As MSGBOXPARAMS)

' アプリケーションを停止するキューをWindowsに送る
Declare Sub Api_PostQuitMessage Lib "user32" Alias "PostQuitMessage" (ByVal nExitCode&)

' 現在のプロセスのコマンドライン文字列へのポインタを取得
Declare Function Api_GetCommandLine& Lib "kernel32" Alias "GetCommandLineA" ()

' 指定された文字列のバイトまたは文字の長さを返す
Declare Function Api_lstrlen& Lib "Kernel32" Alias "lstrlenA" (ByVal lpString&)

' ある位置から別の位置にメモリブロックを移動
Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (Destination As Any, Source
As Any, ByVal Length&)

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Function GetCommLine() As String
Function GetCommLine() As String
    Var SLen As Long
    Var RetStr As Long
    Var Buffer As String

    ' コマンドラインを含むStringのポインタを取得
    RetStr = Api_GetCommandLine

    ' 文字列の長さを取得
    SLen = Api_lstrlen(RetStr)

    If SLen > 0 Then

        ' バッファを初期化
        Buffer = Space$(SLen)

        ' バッファにコピー
        CopyMemory Buffer, ByVal RetStr, SLen

        GetCommLine = Buffer
    End If
End Function

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var mbp As MSGBOXPARAMS
    Var LDs As Long
    Var Cnt As Long
    Var sDrives As String
    Var Ret As Long

    ' 利用可能なドライブを取得
    LDs = Api_GetLogicalDrives

    sDrives = "利用可能なドライブ:"
    For Cnt = 0 To 25
        If (LDs And 2 ^ Cnt) <> 0 Then
            sDrives = sDrives & " " & Chr$(65 + Cnt)
        End If
    Next Cnt

    ' コマンドラインを表示
    Ret = Api_MessageBoxEx(GethWnd, "コマンドライン:" & GetCommLine & Chr$(13, 10) & Chr$(13,
10) & "このメッセージボックスはAPI関数 MessageBoxEx で作成しました。", "コマンドライン",

```



```
MB_ICONEXCLAMATION, 0)
```

```
'構造体サイズを設定
```

```
mbp.cbSize = Len(mbp)
```

```
'アイコンスタイルを設定
```

```
mbp.dwStyle = MB_ICONASTERISK
```

```
'オーナーウィンドウを設定
```

```
mbp.hwndOwner = GethWnd
```

```
'テキストを設定
```

```
mbp.lpszText = StrAdr(sDrives & Chr$(13, 10) & Chr$(13, 10) & "このメッセージボックスはAPI  
関数 MessageBoxIndirect で作成しました。" & Chr$(0))
```

```
'キャプションを設定
```

```
mbp.lpszCaption = StrAdr("利用可能なドライブ" & Chr$(0))
```

```
'メッセージボックスを表示
```

```
Ret = Api_MessageBoxIndirect(mbp)
```

```
'アプリケーションの終了
```

```
' Ret = Api_PostQuitMessage(0)
```

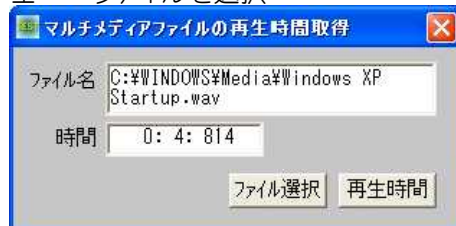
```
End Sub
```

```
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

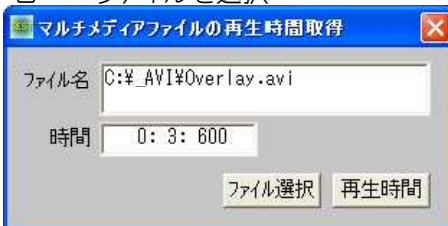
メディアファイルの再生時間取得

メディアファイル(WAV、AVI、MP3、MIDI)を指定し、その再生時間を取得します。
mciSendString 文字列を MCI に送信

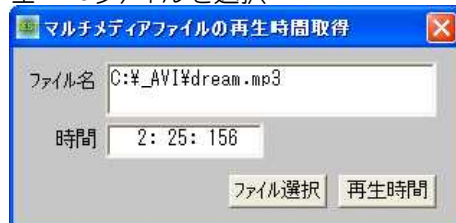
左:WAVファイルを選択



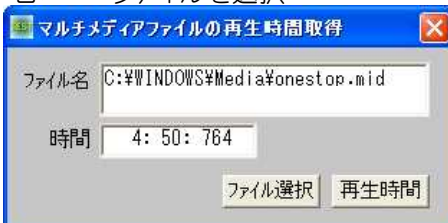
右:AVIファイルを選択



左:MP3ファイルを選択



右:MIDIファイルを選択



※FileNameに空白があることも想定し、chr\$(34) & FileName & chr\$(34) のようにファイルパスを「"」で囲みます。

```
'=====
'= メディアファイルの再生時間取得
'= (GetMediaLength.bas)
'=====
#include "Windows.bi"
```

' 文字列を MCI に送信

```
Declare Function Api_mciSendString& Lib "winmm" Alias "mciSendStringA" (ByVal  
lpstrCommand$, ByVal lpstrReturnString$, ByVal uReturnLength&, ByVal hwndCallback&)  
Var Shared Text(2) As Object  
Var Shared Edit1 As Object  
Var Shared Button(1) As Object
```

```
For i = 0 To 2  
    If i < 2 Then  
        Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1))) :  
    Button(i).SetFontSize 14  
    End If  
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14  
Next  
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
```

```
Var Shared FileName As String
```

```
ShowWindow -1
```

```
'=====
```

```
Declare Function GetMediaLength(FileName As String)
```

```
Function GetMediaLength(FileName As String)
```

```
    Var MediaLength As Long
```

```
    Var RetString As String * 256
```

```
    Var CommandString As String
```

```
    Var Ret As Long
```

```
    CommandString = "Open " & FileName & " Alias MediaFile"
```

```
    Ret = Api_mciSendString(CommandString, ByVal 0, 0, 0)
```

```
    CommandString = "Set MediaFile time format milliseconds"
```

```
    Ret = Api_mciSendString(CommandString, ByVal 0, 0, 0)
```

```
    CommandString = "Status MediaFile length"
```

```
    Ret = Api_mciSendString(CommandString, RetString, Len(RetString), 0)
```

```
    GetMediaLength = CLng(Val(RetString))
```

```
    CommandString = "Close MediaFile"
```

```
    Ret = Api_mciSendString(CommandString, ByVal 0, 0, 0)
```

```
End Function
```

```
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Text(2).SetWindowText ""
```

```
    FileName = WinOpenDlg("ファイルのオープン", "*.wav;*.avi;*.mp3;*.mid", "WAVファイル  
(*.wav);AVIファイル(*.avi);MP3ファイル(*.mp3);MIDIファイル(*.mid);全てのファイル(*.*)", 0)
```

```
    If FileName <> Chr$(&H1B) Then
```

```
        Edit1.SetWindowText FileName
```

```
    End If
```

```
    FileName = Chr$(34) & FileName & Chr$(34)
```

```
End Sub
```

```
'=====
```

```
Declare Sub Button2_on edec1 ()
```

```
Sub Button2_on()
```

```
    If Edit1.GetWindowText() = "" Then Exit Sub
```

```
    Var Seconds As Integer
```

```
    Var Minutes As Integer
```

```
    Var MilliSeconds As Long
```

```

Var TotalTime As String

Milliseconds = GetMediaLength(FileName)
Seconds = Int(Milliseconds / 1000) Mod 60
Minutes = Int(Milliseconds / 60000)
Milliseconds = Milliseconds Mod 1000

TotalTime = Str$(Minutes) & ":" & Str$(Seconds) & ":" & Str$(Milliseconds)
Text(2).SetWindowText TotalTime
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メニュー項目を挿入

CreatePopupMenu ポップアップメニューを作成
DestroyMenu 指定されたメニューを破棄
InsertMenuItem メニュー内の指定された位置に、新しいメニュー項目を挿入
TrackPopupMenu 指定の位置にポップアップメニューを表示
GetWindowRect ウィンドウの座標をスクリーン座標系で取得



```

'=====
'= メニュー項目を挿入
'= (InsertMenuItem.bas)
'=====
#include "Windows.bi"

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

Type MENUITEMINFO
    cbSize      As Long
    fMask       As Long
    fType       As Long
    fState      As Long
    wID         As Long
    hSubMenu    As Long
    hbmpChecked As Long
    hbmpUnChecked As Long
    dwItemData  As Long
    dwTypeData  As Long

    cch        As Long
End Type

```

' 構造体のバイト数
 ' 取得する情報を指定する定数の組み合わせ
 ' メニュー項目のタイプを指定する定数の組み合わせ
 ' メニューの状態を指定する定数の組み合わせ
 ' ユーザー定義のメニュー項目のID
 ' 指定のメニュー項目と関連するサブメニューのハンドル
 ' チェックマーク用のビットマップのハンドル
 ' 未チェック時のときのビットマップハンドル
 ' メニュー項目と関連するユーザー定義の値
 ' メニュー項目のタイプ (fMaskにMIIM_Typeを指定したときのみ有効)
 ' メニュー項目のテキストのバイト数

・ポップアップメニューを作成

```
Declare Function Api_CreatePopupMenu& Lib "user32" Alias "CreatePopupMenu" ()
```

・指定されたメニューを破棄

```
Declare Function Api_DestroyMenu& Lib "user32" Alias "DestroyMenu" (ByVal hMenu&)
```

・メニュー内の指定された位置に、新しいメニュー項目を挿入

```
Declare Function Api_InsertMenuItem& Lib "user32" Alias "InsertMenuItemA" (ByVal hMenu&,  
ByVal uItem&, ByVal fByPosition&, lpmi As MENUITEMINFO)
```

・指定の位置にポップアップメニューを表示

```
Declare Function Api_TrackPopupMenu& Lib "user32" Alias "TrackPopupMenu" (ByVal hMenu&,  
ByVal uFlags&, ByVal x&, ByVal y&, ByVal nReserved&, ByVal hWnd&, ByVal prcRect&)
```

・ウィンドウの座標をスクリーン座標系で取得

```
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,  
lpRect As RECT)
```

```
#define MIIM_CHECKMARKS &H8
```

```
#define MIIM_DATA &H20
```

```
#define MIIM_ID &H2
```

```
#define MIIM_STATE &H1
```

```
#define MIIM_SUBMENU &H4
```

```
#define MIIM_TYPE &H10
```

'hbmChecked、hbmUncheckedメンバをセット

'dwItemDataメンバをセット

'wIDメンバをセット

'fStateメンバをセット

'hSubMenuメンバをセット

'fTypeメンバをセット

```
#define MFT_BITMAP &H4
```

```
#define MFT_MENUBARBREAK &H20
```

'ビットマップを使ってメニュー項目を表示

'メニュー項目を新しい列に置く(ドロップダウンメニュー、サブメニュー、ショートカットメニューの時は区切・

'メニュー項目を新しい列に置く(区切り線は入らない)

'オーナードロウをする時指定

'hbmCheckedメンバがNULLの時ラジオボタンを使ってチェック

```
#define MFT_RIGHTJUSTIFY &H4000
```

```
#define MFT_RIGHTORDER &H2000
```

'メニュー項目が右揃え

'アラビア語とかヘブライ語の時指定 (Win9x・2000)

```
#define MFT_SEPARATOR &H800
```

'区切り線(dwTypeDataとcchメンバは無視)

```
#define MFT_STRING &H0
```

'メニュー項目に文字列を使う

```
#define MFS_CHECKED &H8
```

```
#define MFS_DEFAULT &H1000
```

```
#define MFS_DISABLED &H2
```

```
#define MFS_ENABLED &H0
```

```
#define MFS_GRAYED &H1
```

```
#define MFS_HILITE &H80
```

```
#define MFS_UNCHECKED &H0
```

```
#define MFS_UNHILITE &H0
```

'メニュー項目にチェックをつける

'メニュー項目がデフォルトであることを指定

'メニュー項目を選択不可にする

'メニュー項目を選択可能にする

'メニュー項目を灰色にして選択不可にする

'メニュー項目をハイライトにする

'メニュー項目からチェックをはずす

'メニュー項目のハイライトをやめる

```
#define TPM_BOTTOMALIGN &H20
```

'ショートカットメニューの下端を、yパラメータが指定する座標に合わせる

```
#define TPM_CENTERALIGN &H4
```

'ショートカットメニューの中心を、xパラメータが指定する座標に合わせる

```
#define TPM_HORNEGANIMATION &H800
```

```
#define TPM_HORPOSANIMATION &H400
```

'左から右へ向かってメニューのアニメーション表示

'右から左へ向かってメニューのアニメーション表示

```
#define TPM_LEFTALIGN &H0
```

'ショートカットメニューの左端を、xパラメータが指定する座標に合わせる

```
#define TPM_LEFTBUTTON &H0
```

'マウスの左ボタンでポップアップメニューからの選択が行えるようにする

```
#define TPM_NONOTIFY &H80
```

'ユーザーがメニュー項目をクリックしたとき、通知メッセージを送らず

```
#define TPM_RETURNCMD &H100
```

'関数の戻り値として、ユーザーが選択したメニュー項目のIDを返す

```
#define TPM_RIGHTALIGN &H8
```

'ショートカットメニューの右端を、xパラメータが指定する座標に合わせる

```
#define TPM_RIGHTBUTTON &H2
```

'マウスの右ボタンでポップアップメニューからの選択が行えるようにする

```
#define TPM_TOPALIGN &H0
```

'ショートカットメニューの上端を、yパラメータが指定する座標に合わせる

```
#define TPM_VCENTERALIGN &H10
```

'ショートカットメニューの中心を、yパラメータが指定する座標に合わせる

```
#define TPM_VNEGANIMATION &H2000
```

'下から上へ向かってメニューのアニメーション表示

```
#define TPM_VERPOSANIMATION &H1000
```

'上から下へ向かってメニューのアニメーション表示

```

Var Shared hPopupMenu As Long
Var Shared mii(1) As MENUITEMINFO
Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var Ret As Long

    hPopupMenu = Api_CreatePopupMenu()

    'メニュー1の内容
    mii(0).cbSize = Len(mii(0))
    mii(0).dwTypeData = StrAdr("&File" & Chr$(0))
    mii(0).cch = Len(mii(0).dwTypeData)
    mii(0).fMask = MIIM_TYPE Or MIIM_ID
    mii(0).fState = MFS_DEFAULT Or MFS_HILITE
    mii(0).fType = MFT_STRING
    mii(0).wID = 1

    Ret = Api_InsertMenuItem(hPopupMenu, 0, 0, mii(0))
    If Ret = 0 Then
        A% = MsgBox("", "[MenuFile]は挿入できません！", 0, 2)
    End If

    'メニュー2の内容
    mii(1).cbSize = Len(mii(1))
    mii(1).dwTypeData = StrAdr("&Edit" & Chr$(0))
    mii(1).cch = Len(mii(1).dwTypeData)
    mii(1).fMask = MIIM_TYPE Or MIIM_ID
    mii(1).fState = MFS_DEFAULT Or MFS_HILITE
    mii(1).fType = MFT_STRING
    mii(1).wID = 2

    Ret = Api_InsertMenuItem(hPopupMenu, 0, 0, mii(1))
    If Ret = 0 Then
        A% = MsgBox("", "[MenuEdit]は挿入できません！", 0, 2)
    End If
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var rct As RECT
    Var Flag As Long
    Var Ret As Long

    'Button1のスクリーン座標
    Ret = Api_GetWindowRect(Button1.GethWnd, rct)

    Flag = TPM_LEFTBUTTON Or TPM_TOPALIGN Or TPM_NONOTIFY Or TPM_RETURNCMD

    '指定の位置にポップアップメニューを表示
    Ret = Api_TrackPopupMenu(hPopupMenu, Flag, rct.Right, rct.Bottom, 0, GethWnd, 0)

    Select Case Ret
        Case 0
            A% = MsgBox("", "キャンセルされました！", 0, 2)
        Case 1
            A% = MsgBox("", "File が選択されました！", 0, 2)
        Case 2
            A% = MsgBox("", "Edit が選択されました！", 0, 2)
    End Select
End Sub

```

```

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    Ret = Api_DestroyMenu (hPopupMenu)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メニューの矩形サイズを取得

指定したメニューの矩形サイズを取得し、そのメニューのクリックをシミュレートします。

GetMenu メニューのハンドルを取得

GetMenuItemRect 指定したメニューの矩形サイズを取得

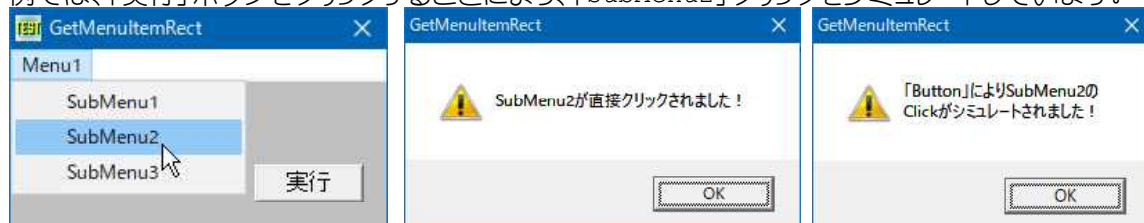
GetSubMenu ポップアップメニューのハンドルを取得

mouse_event マウスイベントの生成

GetMessageExtraInfo 現在のスレッドに関するメッセージの拡張情報を取得

GetSystemMetrics システムメトリックの値とシステムの現在の構成を取得

例では、「実行」ボタンをクリックすることにより、「SubMenu2」をクリックをシミュレートしています。



```

'=====
'= メニューの矩形サイズを取得
'= (GetMenuItemRect.bas)
'=====
#include "Windows.bi"

```

```

Type POINTAPI
    x As Long
    y As Long
End Type

```

```

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

```

' メニューのハンドルを取得

```
Declare Function Api_GetMenu& Lib "user32" Alias "GetMenu" (ByVal hWnd&)
```

' 指定のメニューの矩形サイズを取得

```
Declare Function Api_GetMenuItemRect& Lib "user32" Alias "GetMenuItemRect" (ByVal hWnd&,
ByVal hMenu&, ByVal uItem&, lprcItem As RECT)
```

' ポップアップメニューのハンドルを取得

```
Declare Function Api_GetSubMenu& Lib "user32" Alias "GetSubMenu" (ByVal hMenu&, ByVal
nPos&)
```

マウスイベントの生成

```
Declare Sub Api_mouse_event Lib "user32" Alias "mouse_event" (ByVal dwFlags&, ByVal dx&, ByVal dy&, ByVal cButtons&, ByVal dwExtraInfo&)
```

現在のスレッドに関するメッセージの拡張情報を取得

```
Declare Function Api_GetMessageExtraInfo& Lib "user32" Alias "GetMessageExtraInfo" ()
```

さまざまなシステムメトリックの値とシステムの現在の構成を取得

```
Declare Function Api_GetSystemMetrics& Lib "user32" Alias "GetSystemMetrics" (ByVal nIndex&)
```

```
#define MOUSEEVENTF_ABSOLUTE &H8000          '絶対座標(スクリーン座標Left=0・Top=0・Right=35535・Bottom=65535)
#define MOUSEEVENTF_LEFTDOWN &H2             'マウス左ボタンDOWN
#define MOUSEEVENTF_LEFTUP &H4               'マウス左ボタンUP
#define MOUSEEVENTF_MIDDLEDOWN &H20          'マウス中ボタンDOWN
#define MOUSEEVENTF_MIDDLEUP &H40            'マウス中ボタンUP
#define MOUSEEVENTF_MOVE &H1                'マウス移動
#define MOUSEEVENTF_RIGHTDOWN &H8           'マウス右ボタンDOWN
#define MOUSEEVENTF_RIGHTUP &H10            'マウス右ボタンUP
#define SM_CXSCREEN 0                        'ディスプレイの幅
#define SM_CYSCREEN 1                       'ディスプレイ高さ
#define vbCrLf (Chr$(13) & Chr$(10))        'キャリッジリターンとラインフィード(¥r¥n)
```

```
Var Shared Flg As Integer
```

```
'=====
'=
'=====
```

```
Declare Sub ScreenToAbsolute(pa As POINTAPI)
Sub ScreenToAbsolute(pa As POINTAPI)
    pa.x = pa.x * (&HFFFF / Api_GetSystemMetrics(SM_CXSCREEN))
    pa.y = pa.y * (&HFFFF / Api_GetSystemMetrics(SM_CYSCREEN))
End Sub
```

```
'=====
'= マウスクリックシミュレート
'=====
```

```
Declare Sub MouseClick(pa As POINTAPI)
Sub MouseClick(pa As POINTAPI)
    Api_mouse_event MOUSEEVENTF_ABSOLUTE Or MOUSEEVENTF_MOVE, pa.x, pa.y, 0,
    Api_GetMessageExtraInfo()

    'Mouse Down
    Api_mouse_event MOUSEEVENTF_LEFTDOWN, 0, 0, 0, Api_GetMessageExtraInfo()

    'Mouse Up
    Api_mouse_event MOUSEEVENTF_LEFTUP, 0, 0, 0, Api_GetMessageExtraInfo()
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub ClickMenuItem(ByVal mWnd As Long, ByVal hMenu As Long, ByVal hPos As Long)
Sub ClickMenuItem(ByVal mWnd As Long, ByVal hMenu As Long, ByVal hPos As Long)
    Var rct As RECT
    Var pa As POINTAPI
    Var Ret As Long

    Ret = Api_GetMenuItemRect(mWnd, hMenu, hPos, rct)
    If Ret = 0 Then Exit Sub

    pa.x = (rct.Left + rct.Right) / 2
    pa.y = (rct.Top + rct.Bottom) / 2

    ScreenToAbsolute pa

    MouseClick pa
    Flg = 0
End Sub
```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on
    Var mWnd As Long
    Var hMenu As Long
    Var hSubMenu As Long

    mWnd = GethWnd

    hMenu = Api_GetMenu(mWnd)
    ClickMenuItem mWnd, hMenu, 0
    'メニューのハンドルを取得
    'Menu1をクリック

    hSubMenu = Api_GetSubMenu(hMenu, 0)
    ClickMenuItem mWnd, hSubMenu, 1
    'サブメニューのハンドルを取得
    'SubMenu2をクリック

    Flg = 1
End Sub

'=====
'=
'=====
Declare Sub SubMenu2_on edec1 ()
Sub SubMenu2_on()
    Var txt As String

    If Flg = 1 Then
        txt = "[Button]によりSubMenu2の" & vbCrLf & "Clickがシミュレートされました！"
    Else
        txt = "SubMenu2が直接クリックされました！"
    End If

    A% = MsgBox(GetWindowText, txt, 0, 2)
    Flg = 0
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var mWnd As Long
    Var pa As POINTAPI
    Var hMenu As Long
    Var hSubMenu As Long

    mWnd = GethWnd

    hMenu = Api_GetMenu(mWnd)
    ClickMenuItem mWnd, hMenu, 0
    'メニューのハンドルを取得
    '

    hSubMenu = Api_GetSubMenu(hMenu, 0)
    ClickMenuItem mWnd, hSubMenu, 1
    'サブメニューのハンドルを取得
    'サブメニューの1番目
    pa.x = &HFFFF / 2
    pa.y = &HFFFF / 2

    MouseClick pa
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

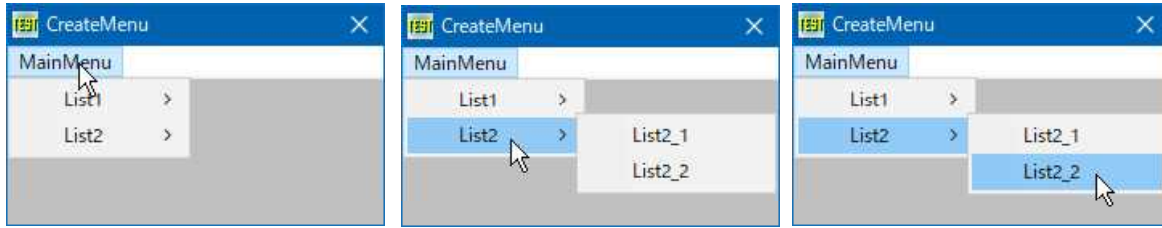

メニューの作成

CreateMenu メニューを作成

CreatePopupMenu ドロップダウンメニュー、サブメニュー、ショートカットメニューのいずれかを作成

AppendMenu メニューの末尾に新しいメニュー項目を追加 (InsertMenuItemを推奨)

SetMenu 指定されたウィンドウに、新しいメニューを割り当てる



```
'=====
'= メニューの作成
'= (CreateMenu.bas)
'=====
#include "Windows.bi"

' メニューを作成。メニューは初期状態では空であるが、InsertMenuItem、InsertMenuの各関数を使ってメニュー項目を追加または挿入できる
Declare Function Api_CreateMenu& Lib "user32" Alias "CreateMenu" ( )

' ドロップダウンメニュー、サブメニュー、ショートカットメニューのいずれかを作成
Declare Function Api_CreatePopupMenu& Lib "user32" Alias "CreatePopupMenu" ( )

' メニューの末尾に新しいメニュー項目を追加 ( InsertMenuItemを推奨)
Declare Function Api_AppendMenu& Lib "user32" Alias "AppendMenuA" (ByVal hMenu&, ByVal wFlags&, ByVal wIDNewItem&, ByVal lpNewItem As Any)

' 指定されたウィンドウに、新しいメニューを割り当てる
Declare Function Api_SetMenu& Lib "user32" Alias "SetMenu" (ByVal hWnd&, ByVal hMenu&)

#define MF_STRING &H0                                '文字列
#define MF_POPUP &H10                                'ドロップダウンメニュー・サブメニューを開くアイテム追加

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ( )
Sub MainForm_Start ( )
    Var hMenu As Long
    Var hSubMenu As Long
    Var hLevel2L1SubMenu As Long
    Var hLevel2L2SubMenu As Long
    Var MenuID As Long
    Var hID As Long
    Var Ret As Long

    hMenu = Api_CreateMenu ( )
    hSubMenu = Api_CreatePopupMenu ( )
    hLevel2L1SubMenu = Api_CreatePopupMenu ( )
    hLevel2L2SubMenu = Api_CreatePopupMenu ( )

    Ret = Api_AppendMenu (hLevel2L1SubMenu, MF_STRING, 211, "List1_1")
    Ret = Api_AppendMenu (hLevel2L1SubMenu, MF_STRING, 212, "List1_2")

    Ret = Api_AppendMenu (hLevel2L2SubMenu, MF_STRING, 221, "List2_1")
    Ret = Api_AppendMenu (hLevel2L2SubMenu, MF_STRING, 222, "List2_2")

    Ret = Api_AppendMenu (hSubMenu, MF_POPUP, hLevel2L1SubMenu, "List1")
    Ret = Api_AppendMenu (hSubMenu, MF_POPUP, hLevel2L2SubMenu, "List2")

    Ret = Api_AppendMenu (hMenu, MF_POPUP, hSubMenu, "MainMenu")
    MenuID = Api_SetMenu (GethWnd, hMenu)
End Sub
```

```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メニューのチェックマークをビットマップで

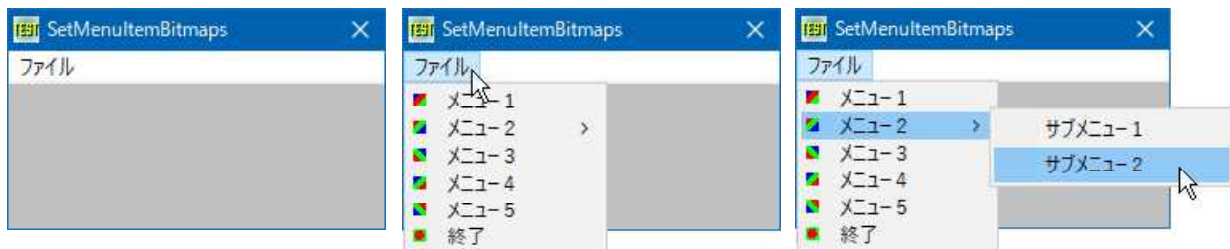
メニューのチェックマークを、作成したBitmapに変更してみます。

LoadImage 画像ファイルの読み込み

GetMenu メニューのハンドルを取得

GetSubMenu ポップアップメニューのハンドルを取得

SetMenuItemBitmaps メニューのチェックマークをビットマップに変更



```

'=====
'= メニューのチェックマークをビットマップで
'= (SetMenuItemBitmaps.bas)
'=====
#include "Windows.bi"

```

```

#define MF_BYPOSITION &H400
#define IMAGE_BITMAP 0
#define LR_LOADFROMFILE &H10

```

'nPositionはメニュー項目のインデックス
'ビットマップ
'外部ファイルからロードする

' 画像ファイルの読み込み

```

Declare Function Api_LoadImage& Lib "user32" Alias "LoadImageA" (ByVal hInst&, ByVal
lpzName$, ByVal uType&, ByVal cxDesired&, ByVal cyDesired&, ByVal fuLoad&)

```

' メニューのハンドルを取得

```

Declare Function Api_GetMenu& Lib "user32" Alias "GetMenu" (ByVal hWnd&)

```

' ポップアップメニューのハンドルを取得

```

Declare Function Api_GetSubMenu& Lib "user32" Alias "GetSubMenu" (ByVal hMenu&, ByVal
nPos&)

```

' メニューのチェックマークをビットマップに変更

```

Declare Function Api_SetMenuItemBitmaps& Lib "user32" Alias "SetMenuItemBitmaps" (ByVal
hMenu&, ByVal nPosition&, ByVal wFlags&, ByVal hBitmapUnchecked&, ByVal hBitmapChecked&)

```

```

'=====
'=
'=====

```

```

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var hMenu As Long
    Var hSubMenu As Long
    Var hBmp(5) As Long
    Var Ret As Long

```

'メニューのハンドルを取得

```

hMenu = Api_GetMenu(GetHwnd)

```

'メニューをチェック

```

If hMenu = 0 Then

```

```

    A% = MessageBox(GetWindowText, "このフォームに「Menu」はありません！", 0, 2)

```

```

Exit Sub
End If

'サブメニューのハンドルを取得
hSubMenu = Api_GetSubMenu(hMenu, 0)

'サブメニューをチェック
If hSubMenu = 0 Then
    A% = MsgBox(GetWindowText, "このフォームに「SubMenu」はありません！", 0, 2)
    Exit Sub
End If

For i% = 0 To 5

    'Bitmapの読み込み
    hBmp(i%) = Api_LoadImage(0, "Menu" & Trim$(Str$(i%)) & ".bmp", IMAGE_BITMAP, 8, 8,
    LR_LOADFROMFILE)

    'メニューにビットマップを設定
    Ret = Api_SetMenuItemBitmaps(hSubMenu, i%, MF_BYPOSITION, hBmp(i%), hBmp(i%))
Next
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メニューのフェード効果を設定

メニューのフェード効果をON/OFFします。
SystemParametersInfo システム全体に関するパラメータを取得・設定
SPI_SETSELECTIONFADE (&H1015) フェード効果の設定
SPIF_UPDATEINIFILE (&H1) ユーザープロファイルの更新を指定
SPIF_SENDWININICHANGE (&H2) 全てのアプリケーションに通知して更新

メニューをクリック後のフェード効果を確認してください。短い時間ですのでお見逃し無く(^^;)



```

'=====
'= メニューのフェード効果を設定
'= (SelectFade.bas)
'=====
#include "Windows.bi"

#define SPI_SETSELECTIONFADE &H1015      'フェード効果の設定
#define SPIF_UPDATEINIFILE &H1          'ユーザープロファイルの更新を指定
#define SPIF_SENDWININICHANGE &H2      '全てのアプリケーションに通知して更新

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo& Lib "user32" Alias "SystemParametersInfoA"
(ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

```

```

Var Shared Radio1 As Object
Var Shared Radio2 As Object
Var Shared Button1 As Object

Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 14
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function Fade bdecl () As Integer
Function Fade ()
    Fade = Val (Mid$(GetDlgRadioSelect("Radio1"), 6)) - 1
End Function

'=====
'=
'=====
Declare Sub Button1_on edecl ()
Sub Button1_on ()
    Var Ret As Long

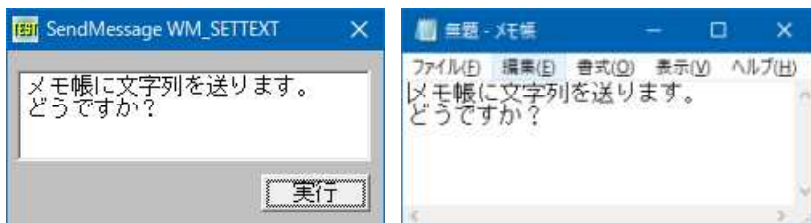
    ' メニューのフェード効果の有効無効を設定
    Ret = Api_SystemParametersInfo (SPI_SETSELECTIONFADE, 0, ByVal Fade,
SPIF_UPDATEINIFILE Or SPIF_SENDCHANGE)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メモ帳のEditに文字列を送る

FindWindow クラス名またはキャプションを与えてウィンドウのハンドルを取得
FindWindowEx クラス名、または キャプションを与えてウィンドウのハンドルを取得
SendMessage ウィンドウにメッセージを送信
WM_SETTEXT(&HC) ウィンドウ(コントロール)のタイトル・テキストを変更
WM_CLOSE(&H10) ウィンドウ或いはアプリケーションをクローズされた



```

'=====
'= メモ帳のEditに文字列を送る
'= (SendMessage9.bas)
'=====
#include "Windows.bi"

' クラス名またはキャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindow& Lib "user32" Alias "FindWindowA" (ByVal lpClassName$,
ByVal lpWindowName$)

' クラス名、または キャプションを与えてウィンドウのハンドルを取得
Declare Function Api_FindWindowEx& Lib "user32" Alias "FindWindowExA" (ByVal
hWndParent&, ByVal hWndChildAfter&, ByVal lpszClass$, ByVal lpszWindow$)

```

' ウィンドウにメッセージを送信

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal  
wMsg&, ByVal wParam&, lParam As Any)
```

#define WM_SETTEXT &HC

#define WM_CLOSE &H10

'ウィンドウ(コントロール)のタイトル・テキストを変更

'ウィンドウ或いはアプリケーションをクローズされた

Var Shared Edit1 As Object

Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared hWnd As Long

```
'=====
'=
'=====
```

Declare Sub MainForm_Start edec1 ()

Sub MainForm_Start()

Shell "notepad", , 5

End Sub

```
'=====
'=
'=====
```

Declare Sub Button1_on edec1 ()

Sub Button1_on()

Var ClassName As String

Var hChildWindow As Long

Var txt As String

Var Ret As Long

txt = Edit1.GetWindowText

'クラス名でウィンドウハンドルを取得

ClassName = "Notepad"

hWnd = Api_FindWindow(ClassName, ByVal 0)

'ウィンドウハンドルを取得できたときは

If hWnd <> 0 Then

 'クラス名で子ウィンドウハンドルを取得

 ClassName = "Edit"

 hChildWindow = Api_FindWindowEx(hWnd, 0, ClassName, ByVal 0)

 'コントロールのテキストを設定

 Ret = Api_SendMessage(hChildWindow, WM_SETTEXT, 0, txt)

End If

End Sub

```
'=====
'=
'=====
```

Declare Sub MainForm_QueryClose edec1 ()

Sub MainForm_QueryClose()

Var Ret As Long

'メモ帳を終了させる

Ret = Api_SendMessage(hWnd, WM_CLOSE, 0, 0)

End Sub

```
'=====
'=
'=====
```

While 1

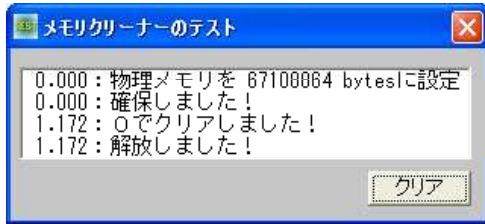
 WaitEvent

Wend

Stop : End

メモリクリーナーのテスト

VirtualAlloc 仮想アドレス空間に領域を確保
VirtualFree 仮想空間に確保した領域を解放
FillMemory メモリブロックを指定の値で初期化
timeGetTime システムが起動してからの経過時間を取得



```
'=====
'= メモリクリーナー
'= (VirtualAlloc.bas)
'=====
#include "Windows.bi"

' 仮想アドレス空間に領域を確保
Declare Function Api_VirtualAlloc& Lib "kernel32" Alias "VirtualAlloc" (lpAddress As Any,
ByVal dwSize&, ByVal flAllocationType&, ByVal flProtect&)

' 仮想アドレス空間に確保した領域を解放
Declare Function Api_VirtualFree& Lib "kernel32" Alias "VirtualFree" (lpAddress As Any,
ByVal dwSize&, ByVal dwFreeType&)

' メモリブロックを指定の値で初期化
Declare Sub FillMemory Lib "Kernel32" Alias "RtlFillMemory" (Destination As Any, ByVal
Length&, ByVal Fill As Byte)

' システムが起動してからの起動時間を取得
Declare Function Api_timeGetTime& Lib "winmm" Alias "timeGetTime" ()

#define MEM_COMMIT &H1000
#define MEM_DECOMMIT &H4000
#define PAGE_READWRITE &H4
#define PAGE_NOCACHE &H200

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var AllocPtr As Long
    Var cSize As Long
    Var nStart As Long
    Var nEnd As Long
    Var Ret As Long

    List1.ResetContent

    nStart = Api_TimeGetTime()

    ' 掃除するメモリーのサイズを設定
    cSize = 64
    cSize = cSize * 1024 * 1024
    nEnd = Api_TimeGetTime()

    'メモリ内またはディスクのページングファイル内に、指定さ
    れたページ領域に相当する物理的な記憶域を割り当
    '物理メモリへのコミット解除
    'コミット済みのページ領域に対する読み取りアクセスと書き
    込みアクセスの両方を許可
    'コミット済みのページ領域に対するキャッシュを禁止

    'アドレス保存用変数
    '掃除するサイズ

    '未指定で64メガバイトとする
    'メガバイトをバイトに変換
```

```

List1.AddString Format$((Cdbl(nEnd) - Cdbl(nStart)) / 1000, "##.###") & ":物理メモリを"
& Str$(cSize) & " bytesに設定！"

'メモリーの掃除
AllocPtr = Api_VirtualAlloc(ByVal 0, cSize, MEM_COMMIT, PAGE_READWRITE Or
PAGE_NOCACHE)
nEnd = Api_TimeGetTime()
List1.AddString Format$((Cdbl(nEnd) - Cdbl(nStart)) / 1000, "##.###") & ":確保しました！"

"

FillMemory ByVal AllocPtr, cSize, 0
nEnd = Api_TimeGetTime()
List1.AddString Format$((Cdbl(nEnd) - Cdbl(nStart)) / 1000, "##.###") & ":0でクリアしまし
た！"

Ret = Api_VirtualFree(AllocPtr, cSize, MEM_DECOMMIT)
nEnd = Api_TimeGetTime()
List1.AddString Format$((Cdbl(nEnd) - Cdbl(nStart)) / 1000, "##.###") & ":解放しました！"

"
End Sub

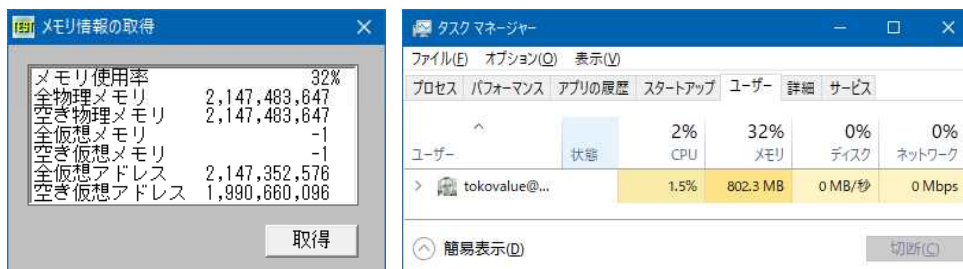
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メモリ情報の取得(1)

メモリ情報を取得します。

GlobalMemoryStatus メモリ情報取得



```

'=====
'= メモリ情報の取得
'= (GlobalMemoryStatus.bas)
'=====
#include "Windows.bi"

Type MEMORYSTATUS
    dwLength          As Long
    dwMemoryLoad       As Long
    dwTotalPhys        As Long
    dwAvailPhys        As Long
    dwTotalPageFile    As Long
    dwAvailPageFile    As Long
    dwTotalVirtual     As Long
    dwAvailVirtual     As Long
End Type

'メモリ使用率
'全物理メモリ
'空き物理メモリ
'全仮想メモリ
'空き仮想メモリ
'全仮想アドレス
'空き仮想アドレス

'メモリ情報を取得
Declare Sub Api_GlobalMemoryStatus Lib "Kernel32" Alias "GlobalMemoryStatus" (lpBuffer
As MEMORYSTATUS)

```

```

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14 : List1.SetWindowSize 242, 108
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

ShowWindow -1
Cls

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var ms As MEMORYSTATUS

    'メモリ情報を取得する
    Api_GlobalMemoryStatus ms

    '取得したメモリ情報を表示する
    List1.ResetContent
    List1.AddString "メモリ使用率" & Format$(ms.dwMemoryLoad , "#####")
    List1.AddString "全物理メモリ" & Format$(ms.dwTotalPhys , "###,###,###,###")
    List1.AddString "空き物理メモリ" & Format$(ms.dwAvailPhys , "###,###,###,###")
    List1.AddString "全仮想メモリ" & Format$(ms.dwTotalPageFile, "###,###,###,###")
    List1.AddString "空き仮想メモリ" & Format$(ms.dwAvailPageFile, "###,###,###,###")
    List1.AddString "全仮想アドレス" & Format$(ms.dwTotalVirtual , "###,###,###,###")
    List1.AddString "空き仮想アドレス" & Format$(ms.dwAvailVirtual , "###,###,###,###")
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

メモリ情報の取得 (II)

メモリ情報を取得します。

GlobalMemoryStatus メモリ情報取得

Timerを使って空き物理メモリ情報を継続して取得してみます。

フリーソフト「MemoryPlus」と同時起動させ確認しています。

Timer を使って継続取得...



```

'=====
'= メモリ情報の取得 (II)
'= (GlobalMemoryStatus2.bas)
'=====
#include "Windows.bi"

Type MEMORYSTATUS
    dwLength      As Long
    dwMemoryLoad  As Long
    dwTotalPhys   As Long
    dwAvailPhys   As Long
'メモリ使用率
'全物理メモリ
'空き物理メモリ

```



```

        dwTotalPageFile    As Long
        dwAvailPageFile    As Long
        dwTotalVirtual     As Long
        dwAvailVirtual     As Long
End Type

'メモリ情報を取得
Declare Sub Api_GlobalMemoryStatus Lib "Kernel32" Alias "GlobalMemoryStatus" (lpBuffer As MEMORYSTATUS)

Var Shared Text(3) As Object
Var Shared Timer1 As Object

For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 12
Next
Timer1.Attach GetDlgItem("Timer1")

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.SetInterval 10
    Timer1.Enable -1
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var ms As MEMORYSTATUS

    'メモリ情報を取得する
    Api_GlobalMemoryStatus ms

    '取得したメモリ情報を表示する
    Text(2).SetWindowText Format$(ms.dwTotalPhys, "##,###,###,### Byte")
    Text(3).SetWindowText Format$(ms.dwAvailPhys, "##,###,###,### Byte")
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

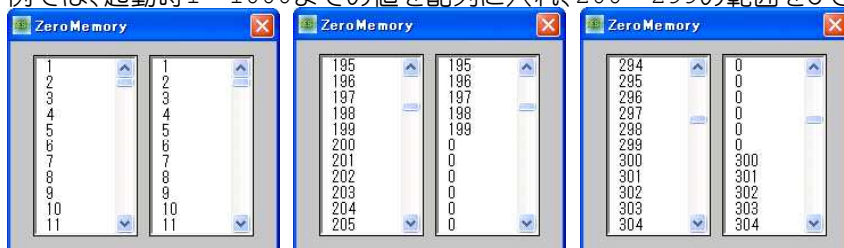
```

メモリブロックを0で初期化

指定した範囲を0で埋めます。

RtlZeroMemory (ZeroMemory) メモリブロックを0で初期化

例では、起動時1~1000までの値を配列に入れ、200~299の範囲を0で埋めています。



左:一見同じようですが
中:200~
右:~299を確認

```

'=====
'= メモリブロックを0で初期化
'= (ZeroMemory.bas)
'=====
#include "Windows.bi"

' メモリブロックを0で初期化
Declare Sub ZeroMemory Lib "kernel32" Alias "RtlZeroMemory" (Destination As Any, ByVal
Length&)

Var Shared List1 As Object
Var Shared List2 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
List2.Attach GetDlgItem("List2") : List2.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var dt(1000) As Long

    '配列にデータを入れる
    For i = 1 To 1000
        dt(i) = i
        List1.AddString Str$(dt(i))
    Next

    '200番目の配列から100項目、0で埋める
    ZeroMemory dt(200), 100 * Len(dt(1))

    '確認
    For i = 1 To 1000
        List2.AddString Str$(dt(i))
    Next
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

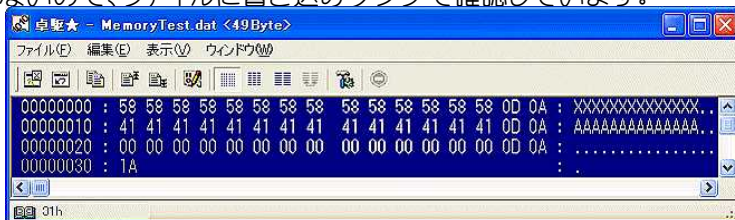
メモリブロックを0または指定の値で初期化

メモリブロックを0または指定の値で初期化します。

RtlFillMemory メモリブロックを指定の値で初期化

RtlZeroMemory メモリブロックを0で初期化

例では、ZeroMemoryの結果をPrintでは確認できないので、ファイルに書き込みダンプで確認しています。



```

'=====
'= メモリブロックを0または指定の値で初期化
'= (Memory.bas)
'=====

```

メモリブロックを指定の値で初期化

```
Declare Sub FillMemory Lib "kernel32" Alias "RtlFillMemory" (Destination As Any, ByVal Length&, ByVal Fill As Byte)
```

メモリブロックを0で初期化

```
Declare Sub ZeroMemory Lib "kernel32" Alias "RtlZeroMemory" (Destination As Any, ByVal Length&)
```

```
var TestString As String
var ff As Integer

ff = FreeFile
Open "MemoryTest.dat" For Create As ff

TestString = string$(14, "X")
Print "TestString: " & TestString
Print #ff, TestString

FillMemory TestString, Len(TestString), Asc("A")
Print "FillMemory: " & TestString
Print #ff, TestString

ZeroMemory TestString, Len(TestString)
Print "ZeroMemory: " & TestString
Print #ff, TestString

Close
Stop
End
```

メモリマップドファイルをファイルに書き込む

CreateFileMapping 指定されたファイルに対するファイルマッピングオブジェクトを作成またはオープン
MapViewOfFile ファイルビューをマッピング
UnmapViewOfFile ファイルのマッピングビューをアンマップ
CreateFile 指定したファイルをオープンし、デバイスハンドルを返す
CloseHandle オープンされているオブジェクトハンドルをクローズ
GetFileSize ファイルのサイズをバイト単位で取得
OpenFileMapping 名前付きファイルマッピングオブジェクトをオープン
FlushViewOfFile マップドファイルのビュー内にある、指定された範囲のデータをディスクに書き込む
SetFilePointer 開いているファイルのポインタを移動
WriteFile データをファイルに書き出す
FillMemory メモリブロックを指定の値で初期化



書き込まれたファイルの中をバイナリダンプで見える。

```
'=====
'= メモリマップドファイルをファイルに書き込む
'= (FlushViewOfFile.bas)
'=====
```

```
#include "Windows.bi"
```

' 指定されたファイルに対するファイルマッピングオブジェクトを作成またはオープン

```
Declare Function Api_CreateFileMapping& Lib "Kernel32" Alias "CreateFileMappingA" (ByVal hFile&, ByVal lpFileMappingAttributes&, ByVal flProtect&, ByVal dwMaximumSizeHigh&, ByVal dwMaximumSizeLow&, ByVal lpName$)
```

' ファイルビューをマッピング

```
Declare Function Api_MapViewOfFile& Lib "Kernel32" Alias "MapViewOfFile" (ByVal hFileMappingObject&, ByVal dwDesiredAccess&, ByVal dwFileOffsetHigh&, ByVal dwFileOffsetLow&, ByVal dwNumberOfBytesToMap&)
```

' ファイルのマッピングビューをアンマップ

```
Declare Function Api_UnmapViewOfFile& Lib "Kernel32" Alias "UnmapViewOfFile" (ByVal lpBaseAddress&)
```

' 指定したファイルをオープンし、デバイスハンドルを返す

```
Declare Function Api_CreateFile& Lib "Kernel32" Alias "CreateFileA" (ByVal lpFileName$, ByVal dwDesiredAccess&, ByVal dwShareMode&, lpSecurityAttributes As Any, ByVal dwCreationDisposition&, ByVal dwFlagsAndAttributes&, ByVal hTemplateFile&)
```

' オープンされているオブジェクトハンドルをクローズ

```
Declare Function Api_CloseHandle& Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)
```

' ファイルのサイズをバイト単位で取得

```
Declare Function Api_GetFileSize& Lib "Kernel32" Alias "GetFileSize" (ByVal hFile&, lpFileSizeHigh&)
```

' 名前付きファイルマッピングオブジェクトをオープン

```
Declare Function Api_OpenFileMapping& Lib "Kernel32" Alias "OpenFileMappingA" (ByVal dwDesiredAccess&, ByVal bInheritHandle&, ByVal lpName$)
```

' マップトファイルのビュー内にある、指定された範囲のデータをディスクに書き込む

```
Declare Function Api_FlushViewOfFile& Lib "kernel32" Alias "FlushViewOfFile" (lpBaseAddress As Any, ByVal dwNumberOfBytesToFlush&)
```

' 開いているファイルのポインタを移動

```
Declare Function Api_SetFilePointer& Lib "Kernel32" Alias "SetFilePointer" (ByVal hFile&, ByVal lDistanceToMove&, lpDistanceToMoveHigh&, ByVal dwMoveMethod&)
```

' データをファイルに書き出す

```
Declare Function Api_WriteFile& Lib "Kernel32" Alias "WriteFile" (ByVal hFile&, lpBuffer As Any, ByVal nNumberOfBytesToWrite&, lpNumberOfBytesWritten&, lpOverlapped As Any)
```

' メモリブロックを指定の値で初期化

```
Declare Sub FillMemory Lib "Kernel32" Alias "RtlFillMemory" (Destination As Any, ByVal Length&, ByVal Fill As byte)
```

```
#define PAGE_READONLY &H2  
#define PAGE_READWRITE &H4
```

```
#define FILE_MAP_READ &H4  
#define FILE_MAP_WRITE &H2  
#define FILE_MAP_COPY &H1  
#define GENERIC_READ -2147483648  
#define GENERIC_WRITE &H40000000  
#define OPEN_ALWAYS 4  
#define OPEN_EXISTING 3  
#define CREATE_ALWAYS 2  
#define FILE_ATTRIBUTE_NORMAL &H80  
#define FILE_BEGIN 0  
#define FILE_CURRENT 1  
#define FILE_END 2  
#define mapname "maptest"  
#define strFilename "maptest.dat"
```

'コミット済みのページ領域に対する読み取りアクセスを許可
'コミット済みのページ領域に対する読み取りアクセスと書き込みアクセスの両方を許可
'読み込み専用モード
'読み書きモード
'上書きモード
'読み込みモード(&H80000000)
'書き込みモード
'ファイルをオープンする(存在しない場合作成)
'ファイルをオープンする(存在しない場合失敗)
'新しいファイルを作る(存在する場合には上書)
'他のファイル属性を持たない
'ファイルポインタを先頭に
'現在のファイルポインタの値が開始点
'現在ファイルの終端の位置が開始点

```
Var Shared Button1 As Object
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```

Var Shared Addr As Long
Var Shared lMap As Long

'=====
'=
'=====
Declare Sub makefile ()
Sub makefile ()
    Var File As Long
    Var LowSize As Long
    Var HighSize As Long
    Var Dat As Long
    Var WriteSize As Long
    Var Ret As Long

    File = Api_CreateFile(strFilename, GENERIC_WRITE, 0, ByVal 0, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0)

    If File = 0 Then
        A% = MsgBox("", "ファイル作成に失敗", 0, 2)
        Exit Sub
    End If

    '10MBのファイルを作成
    LowSize = 1024 ^ 2 * 10
    HighSize = 0

    Ret = Api_SetFilePointer(File, LowSize, HighSize, FILE_BEGIN)
    Ret = Api_WriteFile(File, Dat, Len(Dat), WriteSize, ByVal 0)
    Ret = Api_CloseHandle(File)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var File As Long
    Var Size As Long

    '10MBのファイルを作成
    makefile

    File = Api_CreateFile(strFilename, GENERIC_READ Or GENERIC_WRITE, 0, ByVal 0,
OPEN_EXISTING, 0, 0)

    If File = -1 Then
        A% = MsgBox("", "CreateFile で失敗", 0, 2)
        Exit Sub
    End If

    Size = Api_GetFileSize(File, 0)

    lMap = Api_CreateFileMapping(File, ByVal 0, PAGE_READWRITE, 0, Size, mapname)
    Ret = Api_CloseHandle(File)
    If lMap = 0 Then
        A% = MsgBox("", "CreateFileMapping で失敗", 0, 2)
        Exit Sub
    End If

    Addr = Api_MapViewOfFile(lMap, FILE_MAP_WRITE, 0, 0, 0)

    If Addr = 0 Then
        A% = MsgBox("", "MapViewOfFile で失敗", 0, 2)
        Exit Sub
    End If

    '一気に "x" で初期化する
    FillMemory ByVal Addr, Size, Asc("x")

```

```

'ファイルに書き込み
Ret = Api_FlushViewOfFile(Addr, 0)

Ret = Api_UnmapViewOfFile(Addr)
Ret = Api_CloseHandle(lMap)

A% = MsgBox("", "[x]で初期化し、" & Chr$(13, 10) & "ファイルに書き込みました。", 0, 2)
End Sub

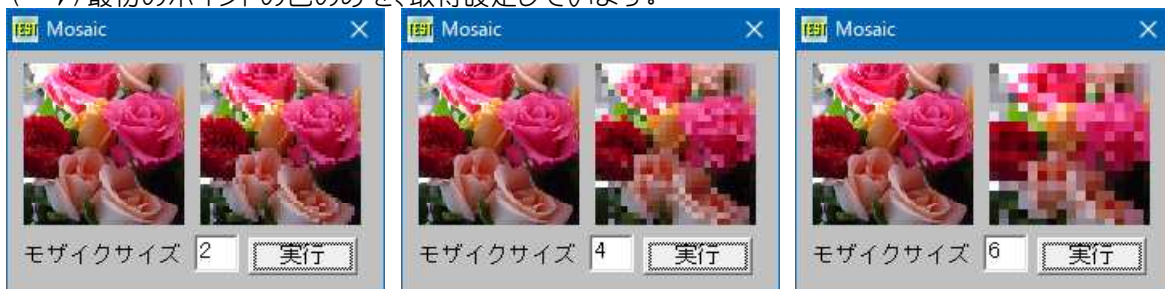
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モザイク(1)

CreateBrushIndirect LOGBRUSH構造体を定義して論理ブラシを作成
SetRect RECT構造体の値を設定
FillRect ブラシで矩形領域を塗りつぶす
DeleteObject オブジェクトを削除
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

本来は、モザイクサイズの2乗に当たる各ポイントの色を取得しそれらの平均値を設定するのですが、面倒なので(^.^;) 最初のポイントの色のみを、取得設定しています。



```

'=====
'= モザイク(1)
'= (Mosaic.bas)
'=====
#include "Windows.bi"

Type LOGBRUSH
    lbStyle As Long
    lbColor As Long
    lbHatch As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Type sColor
    r As Byte
    g As Byte
    b As Byte
End Type

' LOGBRUSH構造体を定義して論理ブラシを作成
Declare Function Api_CreateBrushIndirect Lib "gdi32" Alias "CreateBrushIndirect"

```

```

(lpLogBrush As LOGBRUSH)

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' ブラシで矩形領域を塗りつぶす
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hDC&, ByRef r As RECT,
ByVal hBrush&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define BS_SOLID 0                                'ソリッドブラシ

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object
Var Shared Picture1 As Object
Var Shared Picture2 As Object
Var Shared Bitmap As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Picture1.Attach GetDlgItem("Picture1")
Picture2.Attach GetDlgItem("Picture2")
BitmapObject Bitmap

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Bitmap.LoadFile "flower.bmp"
    Picture1.DrawBitmap Bitmap, 0, 0
    Bitmap.DeleteObject
End Sub

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC As Long
    Var lb As LOGBRUSH
    Var rct As RECT
    Var col As sColor
    Var hBrush As Long
    Var Size As Long
    Var x As Long
    Var y As Long
    Var Ret As Long

    ' デバイスコンテキストのハンドルを取得
    hDC = Api_GetDC(Picture2.GethWnd)

    Size = Val(Edit1.GetWindowText)

    Picture2.Cls

    ' 変換する画像の位置の色を取得
    For x = 0 To Picture1.GetWidth - 1 Step Size

```



```

For y = 0 To Picture1.GetHeight - 1 Step Size
    col.r = (Picture1.GetPixel(x, y) And &HFF)
    col.g = ((Picture1.GetPixel(x, y) ¥ &H100) And &HFF)
    col.b = ((Picture1.GetPixel(x, y) ¥ &H10000) And &HFF)

    '取得した色を設定
    lb.lbStyle = BS_SOLID
    lb.lbColor = RGB(col.r, col.g, col.b)

    'ブラシを作成
    hBrush = Api_CreateBrushIndirect(lb)

    'RECT構造体に値を設定
    Ret = Api_SetRect(rct, x, y, x + Size, y + Size)

    '矩形領域をブラシで塗りつぶす
    Ret = Api_FillRect(hDC, rct, hBrush)

    'オブジェクトを削除
    Ret = Api_DeleteObject(hBrush)
    CallEvent

Next
Next

'デバイスコンテキストを解放
Ret = Api_ReleaseDC(Picture2.GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モザイク(II)

CreateBrushIndirect LOGBRUSH構造体を定義して論理ブラシを作成
SetRect RECT構造体の値を設定
FillRect ブラシで矩形領域を塗りつぶす
DeleteObject オブジェクトを削除
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

モザイクサイズの2乗に当たる各ポイントの色を取得しそれらの平均値を設定しています。
アニメーションは、モザイク(I)との比較です。




```

'=====
'= モザイク(11)
'= (Mosaic2.bas)
'=====
#include "Windows.bi"

Type LOGBRUSH
    lbStyle As Long
    lbColor As Long
    lbHatch As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Type sColor
    r As Byte
    g As Byte
    b As Byte
End Type

' LOGBRUSH構造体を定義して論理ブラシを作成
Declare Function Api_CreateBrushIndirect& Lib "gdi32" Alias "CreateBrushIndirect"
(lpLogBrush As LOGBRUSH)

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' ブラシで矩形領域を塗りつぶす
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hDC&, ByRef r As RECT,
ByVal hBrush&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define BS_SOLID 0                                'ソリッドブラシ

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object
Var Shared Picture1 As Object
Var Shared Picture2 As Object
Var Shared Bitmap As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Picture1.Attach GetDlgItem("Picture1")
Picture2.Attach GetDlgItem("Picture2")
BitmapObject Bitmap

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Bitmap.LoadFile "flower.bmp"
    Picture1.DrawBitmap Bitmap, 0, 0

```

```

        Bitmap.DeleteObject
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC As Long
    Var lb As LOGBRUSH
    Var rct As RECT
    Var col As sColor
    Var hBrush As Long
    Var Size As Long
    Var z As Long
    Var x As Long
    Var y As Long
    Var rr As Long
    Var gg As Long
    Var bb As Long
    Var Ret As Long

    'デバイスコンテキストのハンドルを取得
    hDC = Api_GetDC(Picture2.GethWnd)

    Size = Val(Edit1.GetWindowText)

    Picture2.Cls

    '変換する画像の位置の色を取得
    For x = 0 To Picture1.GetWidth Step Size
        For y = 0 To Picture1.GetHeight Step Size

            'モザイクサイズ(2乗)
            z = Size ^ 2

            'モザイクサイズ内の各ポイントの色を取得
            For i = 1 To Size
                For j = 1 To Size
                    If Picture1.GetPixel(x + i, y + j) = -1 Then

                        '色を取得できなかった場合
                        z = z - 1
                    Else

                        '取得した色を加算
                        rr = rr + (Picture1.GetPixel(x + i, y + j) And &HFF)
                        gg = gg + ((Picture1.GetPixel(x + i, y + j) ¥ &H100) And &HFF)
                        bb = bb + ((Picture1.GetPixel(x + i, y + j) ¥ &H10000) And &HFF)
                    End If
                Next
            Next

            '色を平均値に戻す
            If z <> 0 Then
                col.r = rr / z
                col.g = gg / z
                col.b = bb / z
            End If

            '取得した色を設定
            lb.lbStyle = BS_SOLID
            lb.lbColor = RGB(col.r, col.g, col.b)

            'ブラシを作成
            hBrush = Api_CreateBrushIndirect(lb)

            'RECT構造体に値を設定
            Ret = Api_SetRect(rct, x, y, x + Size, y + Size)

```

```

'矩形領域をブラシで塗りつぶす
Ret = Api_FillRect(hDC, rct, hBrush)

rr = 0
gg = 0
bb = 0

'オブジェクトを削除
Ret = Api_DeleteObject(hBrush)

CallEvent
Next
Next

'デバイスコンテキストを解放
Ret = Api_ReleaseDC(Picture2.GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モザイク(Ⅲ)

CreateBrushIndirect LOGBRUSH構造体を定義して論理ブラシを作成
SetRect RECT構造体の値を設定
FillRect ブラシで矩形領域を塗りつぶす
DeleteObject オブジェクトを削除
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

モザイクサイズの2乗に当たる各ポイントの色を取得しそれらの平均値を設定しています。



```

'=====
'= モザイク(Ⅲ)
'= (Mosaic3.bas)
'=====
#include "Windows.bi"

Type LOGBRUSH
    lbStyle As Long
    lbColor As Long
    lbHatch As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

```

```

Type sColor
    r            As Byte
    g            As Byte
    b            As Byte
End Type

' LOGBRUSH構造体を定義して論理ブラシを作成
Declare Function Api_CreateBrushIndirect& Lib "gdi32" Alias "CreateBrushIndirect"
(lpLogBrush As LOGBRUSH)

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' ブラシで矩形領域を塗りつぶす
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hDC&, ByRef r As RECT,
ByVal hBrush&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define BS_SOLID 0                                'ソリッドブラシ

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Button1 As Object
Var Shared Picture1 As Object
Var Shared Picture2 As Object
Var Shared Bitmap As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Picture1.Attach GetDlgItem("Picture1")
Picture2.Attach GetDlgItem("Picture2")
BitmapObject Bitmap

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC As Long
    Var lb As LOGBRUSH
    Var rct As RECT
    Var col As sColor
    Var hBrush As Long
    Var Size As Long
    Var z As Long
    Var x As Long
    Var y As Long
    Var rr As Long
    Var gg As Long
    Var bb As Long
    Var Ret As Long

    ' デバイスコンテキストのハンドルを取得
    hDC = Api_GetDC(Picture2.GethWnd)

    Size = Val(Edit1.GetWindowText)

```

Picture2.Cls

'変換する画像の位置の色を取得

For x = 0 To Picture1.GetWidth Step Size

For y = 0 To Picture1.GetHeight Step Size

'モザイクサイズ (2乗)

Z = Size ^ 2

'モザイクサイズ内の各ポイントの色を取得

For i = 1 To Size

For j = 1 To Size

If Picture1.GetPixel(x + i, y + j) = -1 Then

'色を取得できなかった場合

z = z - 1

Else

'取得した色を加算

rr = rr + (Picture1.GetPixel(x + i, y + j) And &HFF)

gg = gg + ((Picture1.GetPixel(x + i, y + j) ¥ &H100) And &HFF)

bb = bb + ((Picture1.GetPixel(x + i, y + j) ¥ &H10000) And &HFF)

End If

Next

Next

'色を平均値に戻す

If z <> 0 Then

col.r = rr / z

col.g = gg / z

col.b = bb / z

End If

'取得した色を設定

lb.lbStyle = BS_SOLID

lb.lbColor = RGB(col.r, col.g, col.b)

'ブラシを作成

hBrush = Api_CreateBrushIndirect(lb)

'RECT構造体に値を設定

Ret = Api_SetRect(rct, x, y, x + Size, y + Size)

'矩形領域をブラシで塗りつぶす

Ret = Api_FillRect(hDC, rct, hBrush)

rr = 0

gg = 0

bb = 0

'オブジェクトを削除

Ret = Api_DeleteObject(hBrush)

CallEvent

Next

Next

'デバイスコンテキストを解放

Ret = Api_ReleaseDC(Picture2.GethWnd, hDC)

End Sub

'=====

'= シェルドロップされたファイル名を取得

'=====

Declare Sub Edit2_DropFiles edec1 (ByVal DF As Long)

Sub Edit2_DropFiles (ByVal DF As Long)

Var FileName As String

Var CN As Long

Picture1.Cls

```

CN = GetDropFileCount (DF)
FileName = GetDropFileName (DF, 0)
Edit2.SetWindowText FileName

Bitmap.LoadFile FileName
Picture1.DrawBitmap Bitmap, 0, 0
Bitmap.DeleteObject
End Sub

'=====
' =
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モザイク (IV) 矩形範囲指定

CreateBrushIndirect LOGBRUSH構造体を定義して論理ブラシを作成
SetRect RECT構造体の値を設定
FillRect ブラシで矩形領域を塗りつぶす
DeleteObject オブジェクトを削除
DrawFocusRect 指定の矩形の周囲にフォーカスを示す点線を描画
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放

モザイクサイズの2乗に当たる各ポイントの色を取得しそれらの平均値を設定しています。
 いろいろな画像をテストするため、ファイルをドラッグ & ドロップするようにしました。
 更に、矩形範囲を指定できるようにしてみました。Picture1 (左側) は、比較のための画像で特別必要ではありません。



DrawFocusRectは、もう一度実行すると消去されます。
 矩形領域設定時、領域を連続して表示するには、
 Var Shared oy1 As Single
 Var Shared oy2 As Single
 を加え

```

Declare Sub Picture2_MouseMove edec1 (ByVal Button%, ByVal Shift%, ByVal sx!, ByVal sy!)
Sub Picture2_MouseMove (ByVal Button%, ByVal Shift%, ByVal sx!, ByVal sy!)
    Var rct As RECT
    Var Ret As Long

    rct.Left = x1
    rct.Top = y1

    If Button% = 1 Then
        x2 = sx
        y2 = sy

        ' マウス移動前の矩形領域を設定し、消去
        rct.Right = ox2
        rct.Bottom = oy2
        Ret = Api_DrawFocusRect (hDC, rct)
    End If
End Sub

```

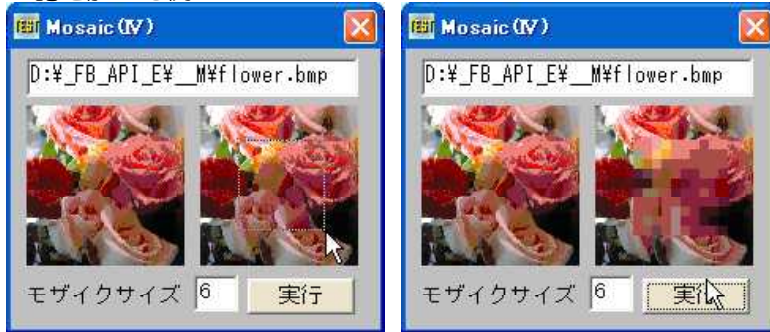
```

'現在の矩形領域を設定し、表示
rct.Right = x2
rct.Bottom = y2
Ret = Api_DrawFocusRect(hDC, rct)

'現在の矩形領域情報を保持
ox2 = x2
oy2 = y2
End If
End Sub

```

上記を加えた例



```

'=====
'= モザイク(IV)範囲指定
'= (Mosaic4.bas)
'=====
#include "Windows.bi"

Type LOGBRUSH
    lbStyle As Long
    lbColor As Long
    lbHatch As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Type sColor
    r As Byte
    g As Byte
    b As Byte
End Type

' LOGBRUSH構造体を定義して論理ブラシを作成
Declare Function Api_CreateBrushIndirect& Lib "gdi32" Alias "CreateBrushIndirect"
(lpLogBrush As LOGBRUSH)

' RECT構造体の値を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' ブラシで矩形領域を塗りつぶす
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hDC&, ByRef r As RECT,
ByVal hBrush&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定の矩形の周囲にフォーカスを示す点線を描画する
Declare Function Api_DrawFocusRect& Lib "user32" Alias "DrawFocusRect" (ByVal hDC&,
lpRect As RECT)

```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define BS_SOLID 0

'ソリッドブラシ

Var Shared Text1 As Object
Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Button1 As Object
Var Shared Picture1 As Object
Var Shared Picture2 As Object
Var Shared Bitmap As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Picture1.Attach GetDlgItem("Picture1")
Picture2.Attach GetDlgItem("Picture2")
BitmapObject Bitmap

Var Shared hDC As Long
Var Shared x1 As Single
Var Shared x2 As Single
Var Shared y1 As Single
Var Shared y2 As Single
Var Shared Capture As Integer

'=====

'=

'=====

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()

' デバイスコンテキストのハンドルを取得

hDC = Api_GetDC(Picture2.GethWnd)

End Sub

'=====

'=

'=====

Declare Sub Button1_on edec1 ()
Sub Button1_on()

Var lb As LOGBRUSH
Var rct As RECT
Var col As sColor
Var hBrush As Long
Var Size As Long
Var z As Long
Var x As Long
Var y As Long
Var rr As Long
Var gg As Long
Var bb As Long
Var Ret As Long

Size = Val(Edit1.GetWindowText)

If x1 + x2 + y + y2 = 0 Then

x1 = 0

x2 = 100

y1 = 0

y2 = 100

End If

' 変換する画像の位置の色を取得

For x = x1 To x2 Step Size


```

For y = y1 To y2 Step Size

    'モザイクサイズ(2乗)
    z = Size ^ 2

    'モザイクサイズ内の各ポイントの色を取得
    For i = 1 To Size
        For j = 1 To Size
            If Picture1.GetPixel(x + i, y + j) = -1 Then

                '色を取得できなかった場合
                z = z - 1
            Else

                '取得した色を加算
                rr = rr + (Picture1.GetPixel(x + i, y + j) And &HFF)
                gg = gg + ((Picture1.GetPixel(x + i, y + j) ¥ &H100) And &HFF)
                bb = bb + ((Picture1.GetPixel(x + i, y + j) ¥ &H10000) And &HFF)
            End If
        Next
    Next

    '色を平均値に戻す
    If z <> 0 Then
        col.r = rr / z
        col.g = gg / z
        col.b = bb / z
    End If

    '取得した色を設定
    lb.lbStyle = BS_SOLID
    lb.lbColor = RGB(col.r, col.g, col.b)

    'ブラシを作成
    hBrush = Api_CreateBrushIndirect(lb)

    'RECT構造体に値を設定
    Ret = Api_SetRect(rct, x, y, x + Size, y + Size)

    '矩形領域をブラシで塗りつぶす
    Ret = Api_FillRect(hDC, rct, hBrush)

    rr = 0
    gg = 0
    bb = 0

    'オブジェクトを削除
    Ret = Api_DeleteObject(hBrush)

    CallEvent
Next
Next
End Sub

'=====
'= シェルドロップされたファイル名を取得
'=====
Declare Sub Edit2_DropFiles edec1 (ByVal DF As Long)
Sub Edit2_DropFiles (ByVal DF As Long)
    Var FileName As String
    Var CN As Long

    Picture1.Cls
    Picture2.Cls

    CN = GetDropFileCount(DF)
    FileName = GetDropFileName(DF, 0)

    If Right$(FileName, 4) <> ".bmp" Then
        A% = MsgBox("", "BMPファイルをドラッグ&ドロップしてください!", 0, 2)
    End If
End Sub

```

```

        Exit Sub
    End If
    Edit2.SetWindowText FileName

    Bitmap.LoadFile FileName
    Picture1.StretchBitmap Bitmap, 0, 0, 100, 100
    Picture2.StretchBitmap Bitmap, 0, 0, 100, 100
    Bitmap.DeleteObject
End Sub

'=====
'=
'=====
Declare Sub Picture2_MouseDown edec1 (ByVal Button%, ByVal Shift%, ByVal sx!, ByVal sy!)
Sub Picture2_MouseDown (ByVal Button%, ByVal Shift%, ByVal sx!, ByVal sy!)

    '矩形領域の左上を設定
    x1 = sx!
    y1 = sy!
End Sub

'=====
'=
'=====
Declare Sub Picture2_MouseUp edec1 (ByVal Button%, ByVal Shift%, ByVal sx!, ByVal sy!)
Sub Picture2_MouseUp (ByVal Button%, ByVal Shift%, ByVal sx!, ByVal sy!)
    Var rct As RECT
    Var Ret As Long

    '矩形領域の右下を設定
    x2 = sx!
    y2 = sy!

    rct.Left = x1
    rct.Top = y1
    rct.Right = x2
    rct.Bottom = y2

    '選択矩形領域を表示
    Ret = Api_DrawFocusRect (hDC, rct)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()

    'デバイスコンテキストを解放
    Ret = Api_ReleaseDC (Picture2.GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

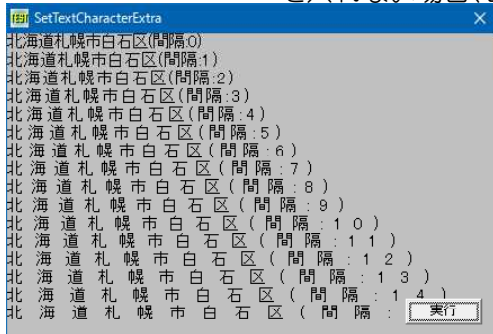
```

文字間隔の調整

文字間隔をピクセル単位で調整します。
CreateFontIndirect 論理フォントを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
GetTextCharacterExtra 文字間隔を取得

SetTextCharacterExtra 文字間隔を設定
 TextOut 文字を描画
 SetBkMode バックグラウンドの塗りつぶしモード設定
 GetDC デバイスコンテキストのハンドルを取得
 ReleaseDC デバイスコンテキストを解放
 TRANSPARENT(1) 背景色を設定しない

文字間隔を0～15ピクセルまで可変表示させています。
 LF.lfCharSet = 128を入れない場合、日本語表示で文字化けをおこします。



文字間隔の取得は

Ret = Api_GetTextCharacterExtra(hDC) で Ret に間隔値が入ります。

```

' =====
' = 文字間隔の調整
' = (SetTextCharacterExtra.bas)
' =====
#include "Windows.bi"

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight As Long
    lfWidth As Long
    lfEscapement As Long
    lfOrientation As Long
    lfWeight As Long
    lfItalic As byte
    lfUnderline As byte
    lfStrikeOut As byte
    lfCharSet As byte
    lfOutPrecision As byte
    lfClipPrecision As byte
    lfQuality As byte
    lfPitchAndFamily As byte
    lfFaceName(LF_FACESIZE) As byte
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 文字間隔を取得
Declare Function Api_GetTextCharacterExtra& Lib "gdi32" Alias "GetTextCharacterExtra"
(ByVal hDC&)

' 文字間隔を設定
Declare Function Api_SetTextCharacterExtra& Lib "gdi32" Alias "SetTextCharacterExtra"
(ByVal hDC&, ByVal nCharExtra&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)
  
```

' バックグラウンドの塗りつぶしモード設定

```
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal iBkMode&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
#define TRANSPARENT 1 '背景色を設定しない
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var hDC As Long
```

```
    Var txt As String
```

```
    Var lf As LOGFONT
```

```
    Var rFont As Long
```

```
    Var Ret As Long
```

```
    hDC = Api_GetDC (GethWnd)
```

```
    Ret = Api_SetBkMode (hDC, TRANSPARENT)
```

```
    lf.lfHeight = 14
```

```
    lf.lfCharSet = 128 '日本語
```

```
    rFont = Api_CreateFontIndirect (lf)
```

```
    Ret = Api_SelectObject (hDC, rFont)
```

```
    For i = 0 To 15
```

```
        txt = "北海道札幌市白石区(間隔:" & Trim$(Str$(i)) & ")"
```

```
        Ret = Api_SetTextCharacterExtra (hDC, i)
```

```
        Ret = Api_TextOut (hDC, 0, i * 16, txt, Len(txt))
```

```
    Next
```

```
    Ret = Api_ReleaseDC (GethWnd, hDC)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

文字サイズのポイントからピクセル変換

文字サイズのポイントからピクセルに変換します。

MulDiv 2つの符号付き32ビット整数を乗算し、その結果(64ビット)を1つの符号付き32ビット整数で除算

```
Declare Function Api_MulDiv& Lib "kernel32" Alias "MulDiv" (ByVal nNumber&, ByVal
```

```
nNumerator&, ByVal nDenominator&)
```

```
    nNumber          被乗数を指定
```

```
    nNumerator       乗数を指定
```

```
    nDenominator      除数を指定
```

(nNumber * nNumerator) を、nDenominator で割ることを意味します。

例では、10Pointの文字を印刷する場合の高さを求めています。整数同士の計算ですので結果を10で除算します。



```
'=====
'= 文字サイズのポイントからピクセルに変換
'= (MulDiv.bas)
'=====
#include "Windows.bi"

' 2つの符号付き32ビット整数を乗算(64ビット)し、その結果を1つの符号付き32ビット整数で除算
Declare Function Api_MulDiv& Lib "kernel32" Alias "MulDiv" (ByVal nNumber&, ByVal
nNumerator&, ByVal nDenominator&)

Var Shared Text(4) As Object
Var Shared Edit(3) As Object
Var Shared Button1 As Object

For i = 0 To 4
    If i < 4 Then
        Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
        Edit(i).SetFontSize 14
    End If
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var A As Long
    Var B As Long
    Var C As Long
    Var D As Long

    A = Val(Edit(0).GetWindowText)
    B = Val(Edit(1).GetWindowText)
    C = Val(Edit(2).GetWindowText)

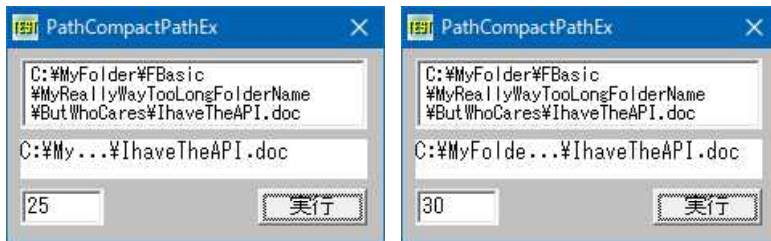
    D = Api_MulDiv(A, B, C)

    Edit(3).SetWindowText Str$(D)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

[文字数に収まるようにパスの一部を省略](#)

[PathCompactPathEx](#) 文字数に収まるようにパスの一部を省略



```
'=====
'= 文字数に収まるようにパスの一部を省略
'= (PathCompactPathEx.bas)
'=====
#include "Windows.bi"

' 文字数に収まるようにパスの一部を省略
Declare Function Api_PathCompactPathEx& Lib "shlwapi" Alias "PathCompactPathExA" (ByVal
pszOut$, ByVal pszSrc$, ByVal cchMax&, ByVal dwFlags&)

Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'= Chr$(0)を取り除く
'=====
Declare Function TrimNull (item As String) As String
Function TrimNull (item As String) As String
    Var ePos As Integer

    ePos = Instr(item, Chr$(0))
    If ePos Then
        TrimNull = Left$(item, ePos - 1)
    Else
        TrimNull = item
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Edit1.SetWindowText
"C:\MyFolder\FBasic\MyReallyWayTooLongFolderName\ButWhoCares\IHaveTheAPI.doc"
    Edit2.SetWindowText "25"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var nTxt As Integer
    Var sSave As String
    Var Buff As String
    Var Ret As Long

    Buff = String$(100, 0)
    sSave = Edit1.GetWindowText
    nTxt = Val (Edit2.GetWindowText)

    Ret = Api_PathCompactPathEx (Buff, sSave, nTxt, 0)
```

```

Text1.SetWindowText TrimNull (Buff)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字の幅を論理単位で取得

文字列を論理単位で取得します。
GetCharWidth32 文字の幅を論理単位で取得
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```

'=====
'= 文字の幅を論理単位で取得
'= (GetCharWidth32.bas)
'=====
#include "Windows.bi"

' 文字の幅を論理単位で取得 (昇順に連続した文字 [AからEまで、SからTまで、など] の幅を一括して取得)
Declare Function Api_GetCharWidth32& Lib "gdi32" Alias "GetCharWidthA" (ByVal hDC&,
ByVal iFirstChar&, ByVal iLastChar&, lpBuffer As Any)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function sGetTextWidth(strText As String) As Long
Function sGetTextWidth(strText As String) As Long
    Var tWidth As Long
    Var TotalWidth As Long
    Var i As Integer
    Var hDC As Long
    Var Ret As Long

    hDC = Api_GetDC(Edit1.GethWnd)

    For i = 1 To Len(strText)
        Ret = Api_GetCharWidth32(hDC, Asc(Mid$(strText, i, 1)), Asc(Mid$(strText, i, 1)),

```

```

tWidth)
    TotalWidth = TotalWidth + tWidth
Next i
sGetTextWidth = TotalWidth

Ret = Api_ReleaseDC (Edit1.GethWnd, hDC)
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Text1.SetWindowText Str$(sGetTextWidth(Edit1.GetWindowText))
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モジュール名の列挙

モジュール名を検索列挙します。

GetCurrentProcessId 自分自身のプロセスIDを取得

CreateToolhelp32Snapshot 最初のモジュールに関する情報を取得

Module32First 最初のモジュールに関する情報を取得

Module32Next 2番目以降のモジュールに関する情報を取得



```

'=====
'= モジュール名の列挙
'= (Module32.bas)
'=====
#include "Windows.bi"

```

```
Type MODULEENTRY32
```

```

    dwSize           As Long
    th32ModuleID     As Long
    th32ProcessID    As Long
    GblcntUsage      As Long
    ProccntUsage     As Long
    modBaseAddr      As Long
    modBaseSize      As Long
    hModule          As Long
    szModule         As String * 256
    szExePath        As String * 260
End Type

```

```
' 自分自身のプロセスIDを取得
```

```
Declare Function Api_GetCurrentProcessId& Lib "kernel32" Alias "GetCurrentProcessId" ()
```


' スナップショットを取得

```
Declare Function Api_CreateToolhelp32Snapshot& Lib "kernel32" Alias  
"CreateToolhelp32Snapshot" (ByVal lFlags&, ByVal lProcessID&)
```

' 最初のモジュールに関する情報を取得

```
Declare Function Api_Module32First& Lib "kernel32" Alias "Module32First" (ByVal  
hSnapshot&, uProcess As MODULEENTRY32)
```

' 2番目以降のモジュールに関する情報を取得

```
Declare Function Api_Module32Next& Lib "kernel32" Alias "Module32Next" (ByVal  
hSnapshot&, uProcess As MODULEENTRY32)
```

```
Var Shared Text1 As Object  
Var Shared List1 As Object  
Var Shared Button1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14  
List1.SetWindowSize 248, 132  
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
```

```
'=  
'=====
```

```
Declare Sub Button1_on edec1 ()  
Sub Button1_on()  
    Var uProcess As MODULEENTRY32  
    Var lProcessID As Long  
    Var hSnapshot As Long  
    Var Ret As Long  
  
    List1.ResetContent  
  
    lProcessID = Api_GetCurrentProcessId  
    hSnapshot = Api_CreateToolhelp32Snapshot(8, 0)  
    uProcess.dwSize = Len(uProcess)  
    Ret = Api_Module32First(hSnapshot, uProcess)  
  
    Do While Ret  
        List1.AddString Left$(uProcess.szModule, InStr(uProcess.szModule, Chr$(0)) - 1)  
        Ret = Api_Module32Next(hSnapshot, uProcess)  
    Loop  
End Sub
```

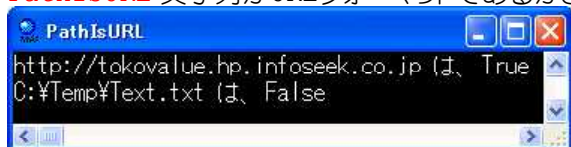
```
'=====
```

```
'=  
'=====
```

```
While 1  
    WaitEvent  
Wend  
Stop  
End
```

文字列がURLフォーマットであるか判断

PathIsURL 文字列がURLフォーマットであるかどうかのみ判断。そのURLの存在は判断しません。



```
'=====
```

```
'= 文字列がURLフォーマットであるか判断  
'= (PathIsURL.bas/P)  
'=====
```

' 文字列がURLフォーマットであるか判断

```
Declare Function Api_PathIsURL Lib "shlwapi" Alias "PathIsURLA" (ByVal pszPath$)

Var MyStr As String
Var Ret As Long

MyStr = "http://tokovalue.web.infoseek.co.jp"
GoSub *judge

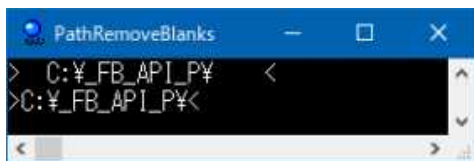
MyStr = "C:¥Temp¥Text.txt"
GoSub *judge

Stop
End

*judge
If Api_PathIsURL(MyStr) Then
    Print MyStr & " は、True"
Else
    Print MyStr & " は、False"
End If
Return
```

文字列前後のスペースを削除

PathRemoveBlanks 文字列の前後にある全てのスペース文字を削除
F-BasicでのTrim\$と同様な働きをします。



```
'=====
'= 文字列前後のスペースを削除
'= (PathRemoveBlanks.bas/P)
'=====
```

' 文字列の前後にある全てのスペース文字を削除

```
Declare Sub Api_PathRemoveBlanks Lib "shlwapi" Alias "PathRemoveBlanksA" (ByVal
pszPath$)

Var Path As String

Path = " C:¥_FB_API_P¥ "
Print ">" & Path & "<"

Api_PathRemoveBlanks Path
Print ">" & Left$(Path, Instr(Path, Chr$(0)) - 1) & "<"
Stop
End
```

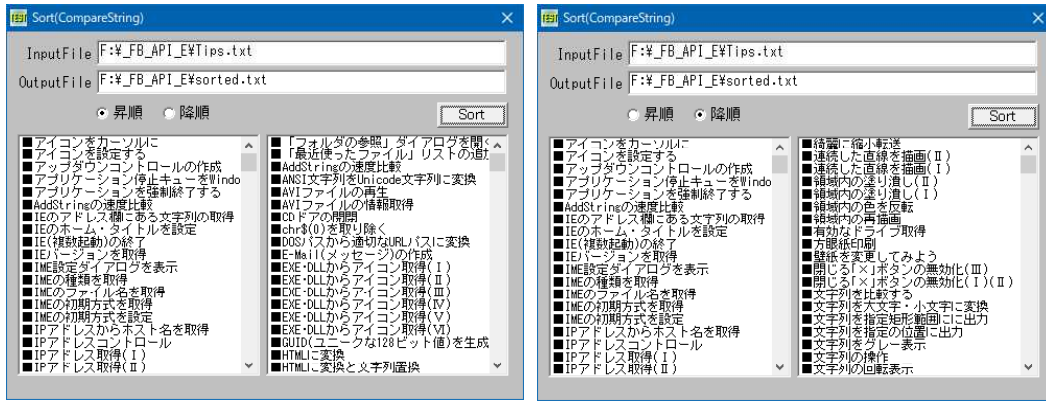
文字列(テキスト)の並び替え

文字列(テキスト)の並び替え

GetThreadLocale スレッドのロケールIDを取得

CompareString 比較オプションに対応した文字列比較関数

例では、当HPの索引(発音順:個人的な^^;)を読み込み昇順または降順に並び替えを行っています。
VisualBasicでのStrCmpの代わりにAPIのCompareStringを使っています。



```
'=====
'= 文字列(テキスト)の並び替え
'= (Sort.bas)
'=====
#include "Windows.bi"

' スレッドのロケールIDを取得
Declare Function Api_GetThreadLocale& Lib "Kernel32" Alias "GetThreadLocale" (

' 比較オプションに対応した文字列比較関数
Declare Function Api_CompareString& Lib "kernel32" Alias "CompareStringA" (ByVal
Locale&, ByVal dwCmpFlags&, ByVal lpString1$, ByVal cchCount1&, ByVal lpString2$, ByVal
cchCount2&)

#define NORM_IGNORECASE &H1
#define CSTR_LESS_THAN 1
#define CSTR_EQUAL 2
#define CSTR_GREATER_THAN 3

'大文字・小文字を区別しない
'文字列1 < 文字列2
'文字列1 = 文字列2
'文字列1 > 文字列2

Var Shared Text(1) As Object
Var Shared Edit(1) As Object
Var Shared Radio(1) As Object
Var Shared List(1) As Object
Var Shared Button1 As Object

For i = 0 To 1
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1))) : Radio(i).SetFontSize 14
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
    List(i).Attach GetDlgItem("List" & Trim$(Str$(i + 1))) : List(i).SetFontSize 12
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub SelectionSort (Dat() As String, ByVal Min As Integer, ByVal Max As Integer)
Sub SelectionSort (Dat() As String, ByVal Min As Integer, ByVal Max As Integer)
    Var hTL As Long
    Var i As Integer
    Var j As Integer
    Var bj As Integer
    Var bs As String
    Var temp As String

    ' カレントロケール取得
    hTL = Api_GetThreadLocale()

    For i = Min To Max - 1
        bj = i
        bs = Dat(i)
        For j = i + 1 To Max
            If Api_CompareString(hTL, NORM_IGNORECASE, Dat(j), Len(Dat(j)), bs, Len(bs))
= CSTR_LESS_THAN Then
                bs = Dat(j)
            End If
        Next j
    Next i
End Sub
```

```

        bj = j
    End If
Next j
Dat(bj) = Dat(i)
Dat(i) = bs
Next i
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var FilePath As String

    FilePath = "C:\¥_FB_API_E¥"
    If Right$(FilePath, 1) <> "¥" Then FilePath = FilePath & "¥"

    Edit(0).SetWindowText FilePath & "Tips.txt"
    Edit(1).SetWindowText FilePath & "sorted.txt"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Lines(500) As String
    Var NewLine As String
    Var MaxLine As Integer
    Var FF As Integer
    Var i As Integer

    List(0).ResetContent
    List(1).ResetContent

    SetMousePointer 2
    CallEvent

    'ソート前テキスト読み込み
    FF = FreeFile
    Open Edit(0).GetWindowText For Input As FF
    Do While Not EOF(FF)
        Line Input #FF, NewLine
        NewLine = Trim$(NewLine)

        MaxLine = MaxLine + 1
        Lines(MaxLine) = NewLine
        List(0).AddString NewLine
    Loop
    Close FF

    'ソート実行
    SelectionSort Lines(), 1, MaxLine

    On Error Goto *Er_Trap

    'ソート後テキスト書き込み
    Open Edit(1).GetWindowText For Output As FF
    If Radio(0).GetCheck = 1 Then

        '昇順
        For i = 1 To MaxLine
            Print #FF, Lines(i)
            List(1).AddString Lines(i)
        Next i
    Else

        '降順
        For i = MaxLine To 1 Step -1

```

```

        Print #FF, Lines(i)
        List(1).AddString Lines(i)
    Next i
End If
Close FF

SetMousePointer 0
' A% = MessageBox(GetWindowText, "Sorted " & Format$(MaxLine) & " Lines", 0, 2)
Exit Sub

*Er_Trap
If Err = 64 Then
    Close #FF
    Kill Edit(1).GetWindowText
    Open Edit(1).GetWindowText For Output As FF
    Resume Next
End If
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

文字列にスペース文字がある場合、文字列全体を""で囲む

PathQuoteSpaces 文字列にスペース文字がある場合、文字列全体を""で囲む



```

' =====
' = 文字列にスペース文字がある場合、文字列全体を""で囲む
' = (PathQuoteSpaces.bas)
' =====
#include "Windows.bi"

' パスを表す文字列にスペース文字がある場合は、文字列全体をダブルクォーテーションマークで囲む
Declare Sub Api_PathQuoteSpaces Lib "shlwapi" Alias "PathQuoteSpacesA" (ByVal lpsz$)

Var Shared Edit1 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' = Chr$(0)を取り除く
' =====
Declare Function TrimNull (item As String) As String
Function TrimNull(item As String) As String
    Var ePos As Integer

    ePos = Instr(item, Chr$(0))
    If ePos Then

```

```

        TrimNull = Left$(item, ePos - 1)
    Else
        TrimNull = item
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "C:¥THE DIR¥MYFILE¥"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sSave As String

    sSave = Edit1.GetWindowText & String$(100, 0)

    Api_PathQuoteSpaces sSave

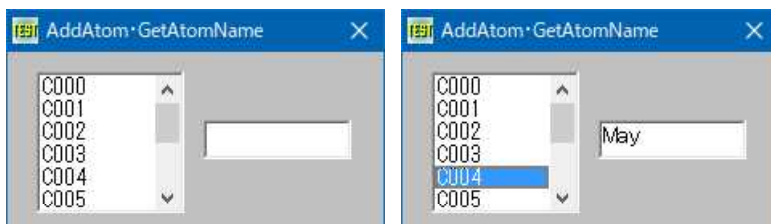
    Text1.SetWindowText TrimNull(sSave)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列のアトムテーブル追加と取得

AddAtom 文字列をローカルアトムテーブルに追加し、その文字列を識別する一意の値(アトム)を返す
GetAtomName 指定されたローカルアトムに関連付けられている文字列のコピーを取得



```

'=====
'= 文字列のアトムテーブル追加と取得
'= (AddAtom.bas)
'=====
#include "Windows.bi"

' 文字列をローカルアトムテーブルに追加し、その文字列を識別する一意の値(アトム)を返す
Declare Function Api_AddAtom% Lib "kernel32" Alias "AddAtomA" (ByVal lpString$)

' 指定されたローカルアトムに関連付けられている文字列のコピーを取得
Declare Function Api_GetAtomName& Lib "kernel32" Alias "GetAtomNameA" (ByVal nAtom%,
ByVal lpBuffer$, ByVal nSize&)

#define MAX_ATOM 255

```

```

Var Shared Text1 As Object
var shared List1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

Var Shared iAtom(11) As Integer

'=====
'=
'=====
Declare Sub MainForm_Start edecl ()
Sub MainForm_Start()
    Var i As Integer
    Var mm(11) As String
    List1.ResetContent

    For i = 0 To 11
        Read mm(i)
        iAtom(i) = Api_AddAtom(mm(i) & Chr$(0))
        List1.AddString Right$(Hex$(iAtom(i)), 4)
    Next i

    data "January"
    data "February"
    data "March"
    data "April"
    data "May"
    data "June"
    data "July"
    data "August"
    data "September"
    data "October"
    data "November"
    data "December"
End Sub

'=====
'=
'=====
Declare Sub List1_Click edecl ()
Sub List1_Click()
    Var i As Integer
    Var txt As String
    Var Ret As Long

    txt = Space$(MAX_ATOM)
    i = List1.GetCursel

    Ret = Api_GetAtomName(iAtom(i), txt, Len(txt))
    Text1.SetWindowText Left$(txt, Ret)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列の回転表示 (I)

文字を回転表示させます。
GetDeviceCaps デバイス固有の情報を取得
BitBlt ビットブロック転送

CreateCompatibleBitmap デバイスコンテキストと互換性のあるビットマップを作成
 CreateCompatibleDC 指定されたDCに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成
 CreateEllipticRgn 楕円形のリージョンを作成
 SetWindowRgn 指定の領域をウィンドウ領域として設定
 CreateFont 指定の属性を持つ論理フォントを作成
 SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
 TextOut 文字を描画
 MulDiv 2つの符号付き32ビット整数を乗算(64ビット)し、その結果を1つの符号付き32ビット整数で除算
 SetBkMode バックグラウンドの塗りつぶしモード設定
 FillRect ブラシで矩形領域を塗りつぶす
 SetRect RECT構造体の値を設定
 GetSysColorBrush システムカラーのインデックスに対応した論理ブラシのハンドルを取得
 GetDC デバイスコンテキストのハンドルを取得
 DeleteDC 指定されたデバイスコンテキストを削除
 DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放
 ReleaseDC デバイスコンテキストを解放



????

統合環境で下記のようにREM[']を入れる位置により

```
#define FW_NORMAL 400
```

CreateMyFont = Api CreateFont(-Api_MulDiv(nSize, Api_GetDeviceCaps(Api_GetDC(0).....行
が文法エラーになります。

私の環境では、

```
#define FW_NORMAL 400 '半角空白が5個まではOK、6個以上はNGでした。(Windows 7でも同様)
```

```
'=====
'= 文字の回転表示
'= (MulDiv2.bas)
'=====
#include "Windows.bi"
```

```

#define FW_BLACK 900 'LOGFONT(lfWeight)・FW_HEAVY
#define FW_BOLD 700 'LOGFONT(lfWeight)
#define FW_DEMIBOLD 600 'LOGFONT(lfWeight)・FW_SEMIBOLD
#define FW_DONTCARE 0 'LOGFONT(lfWeight)
#define FW_EXTRABOLD 800 'LOGFONT(lfWeight)
#define FW_EXTRALIGHT 200 'LOGFONT(lfWeight)
#define FW_HEAVY 900 'LOGFONT(lfWeight)
#define FW_LIGHT 300 'LOGFONT(lfWeight)
#define FW_MEDIUM 500 'LOGFONT(lfWeight)
#define FW_NORMAL 400
#define FW_REGULAR 400 'LOGFONT(lfWeight)
#define FW_SEMIBOLD 600 'LOGFONT(lfWeight)
#define FW_THIN 100 'LOGFONT(lfWeight)
#define FW_ULTRABOLD 800 'LOGFONT(lfWeight)・FW_EXTRABOLD
#define FW_ULTRALIGHT 200 'LOGFONT(lfWeight)・FW_EXTRALIGHT

#define ANSI_CHARSET 0 'Windows標準文字セット
#define DEFAULT_CHARSET 1 'デフォルト(指定なし)
#define SYMBOL_CHARSET 2 'シンボル文字セット
#define SHIFTJIS_CHARSET 128 'シフトJIS文字セット
#define HANGEUL_CHARSET 129 'ハングル文字セット
#define CHINESEBIG5_CHARSET 136 '中国語文字セット
#define OEM_CHARSET 255 'OEM文字セット
```



```

#define OUT_CHARACTER_PRECIS 2
#define OUT_DEFAULT_PRECIS 0
#define OUT_DEVICE_PRECIS 5

#define OUT_OF_MEMORY_OR_RESOURCES 0
#define OUT_OUTLINE_PRECIS 8

#define OUT_RASTER_PRECIS 6
#define OUT_STROKE_PRECIS 3

#define OUT_TT_ONLY_PRECIS 7

#define OUT_TT_PRECIS 4

#define CLIP_CHARACTER_PRECIS 1
#define CLIP_DEFAULT_PRECIS 0
#define CLIP_STROKE_PRECIS 2
#define DEFAULT_PITCH 0
#define DEFAULT_QUALITY 0
#define DRAFT_QUALITY 1

#define FIXED_PITCH 1
#define PROOF_QUALITY 2

#define VARIABLE_PITCH 2

#define OPAQUE 2
#define TRANSPARENT 1
#define LOGPIXELSX 88
#define LOGPIXELSY 90
#define COLOR_WINDOW 5
#define SRCCOPY &HCC0020
#define MSG " 札幌市白石区"

```

'使用されない
 'デフォルトの動作に任せる
 '同じ名前のフォントが複数あった場合、デバイスフォントを選択
 'メモリまたはリソースが足りない
 'WindowsNT2K・XPではTrueTypeフォントやその他のアウトラインベースのフォントを選択。Windows9xでは使用せ・
 '同じ名前のフォントが複数あった場合、ラスタフォントを選択
 'WindowsNT・2K・XPでは使用しない。Windows9x・Meではベクトルフォントを選択
 'TrueTypeフォントだけを選択。システムにTrueTypeフォントが組み込まれていないときは、デフォルトの動作
 '同じ名前のフォントが複数あった場合、TrueTypeフォントを選択
 '使用されない
 'デフォルトの動作に任せる
 '使用されない
 '文字ピッチデフォルト
 'フォントの文字品質は重視されない
 'フォントの文字品質は、PROOF_QUALITYを使用したときほどは重視されない
 '文字ピッチ固定
 'フォントの文字品質が、論理フォントの属性を正確に一致させることよりも重視
 '文字ピッチ可変
 '背景色を設定する
 '背景色を設定しない
 'スクリーン幅において1論理インチあたりのピクセル数
 'スクリーン高さにおいて1論理インチあたりのピクセル数
 'ウィンドウの背景色
 'そのまま転送

```

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

```

・ デバイス固有の情報を取得

```

Declare Function Api_GetDeviceCaps& Lib "gdi32" Alias "GetDeviceCaps" (ByVal hDC&, ByVal nIndex&)

```

・ ビットブロック転送を行う。コピー元からコピー先のデバイスコンテキストへ、指定された長方形内の各ピクセルの色データをコピー

```

Declare Function Api_BitBlt& Lib "gdi32" Alias "BitBlt" (ByVal hDestDC&, ByVal X&, ByVal Y&, ByVal nWidth&, ByVal nHeight&, ByVal hSrcDC&, ByVal xSrc&, ByVal ySrc&, ByVal dwRop&)

```

・ デバイスコンテキストと互換性のあるビットマップを作成

```

Declare Function Api_CreateCompatibleBitmap& Lib "gdi32" Alias "CreateCompatibleBitmap" (ByVal hDC&, ByVal nWidth&, ByVal nHeight&)

```

・ 指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成

```

Declare Function Api_CreateCompatibleDC& Lib "gdi32" Alias "CreateCompatibleDC" (ByVal hDC&)

```

・ 楕円形のリージョンを作成

```

Declare Function Api_CreateEllipticRgn& Lib "gdi32" Alias "CreateEllipticRgn" (ByVal nLeftRect&, ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&)

```

・ 指定の領域をウィンドウ領域として設定

```

Declare Function Api_SetWindowRgn& Lib "user32" Alias "SetWindowRgn" (ByVal hWnd&, ByVal hRgn&, ByVal bRedraw&)

```

・ 指定の属性を持つ論理フォントを作成

```

Declare Function Api_CreateFont& Lib "gdi32" Alias "CreateFontA" (ByVal Hei&, ByVal Wid&, ByVal Esc&, ByVal Ori&, ByVal Wei&, ByVal Ita&, ByVal Und&, ByVal Stk&, ByVal Chr&, ByVal Out&, ByVal Clp&, ByVal Qua&, ByVal Pit&, ByVal Face$)

```

' 指定されたデバイスコンテキストのオブジェクトを選択

```
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal hObject&)
```

' 文字を描画

```
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&, ByVal nYStart&, ByVal lpString$, ByVal cbString&)
```

' 2つの符号付き32ビット整数を乗算(64ビット)し、その結果を1つの符号付き32ビット整数で除算

```
Declare Function Api_MulDiv& Lib "Kernel32" Alias "MulDiv" (ByVal nNumber&, ByVal nNumerator&, ByVal nDenominator&)
```

' バックグラウンドの塗りつぶしモード設定

```
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal iBkMode&)
```

' ブラシで矩形領域を塗りつぶす

```
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hDC&, ByRef r As RECT, ByVal hBrush&)
```

' RECT構造体の値を設定

```
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&, ByVal Y1&, ByVal X2&, ByVal Y2&)
```

' システムカラーのインデックスに対応した論理ブラシのハンドルを取得

```
Declare Function Api_GetSysColorBrush& Lib "user32" Alias "GetSysColorBrush" (ByVal nIndex&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' 指定されたデバイスコンテキストを削除

```
Declare Function Api_DeleteDC& Lib "gdi32" Alias "DeleteDC" (ByVal hDC&)
```

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放

```
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
Var Shared hDC As Long  
Var Shared mDC As Long  
Var Shared mBitmap As Long
```

```
'=====
```

```
Declare Function CreateMyFont(nSize As Integer, nDegrees As Long) As Long  
Function CreateMyFont(nSize As Integer, nDegrees As Long) As Long  
    CreateMyFont = Api_CreateFont(-Api_MulDiv(nSize, Api_GetDeviceCaps(Api_GetDC(0), LOGPIXELSY), 72), 0, nDegrees * 10, 0, FW_NORMAL, 0, 0, 0, DEFAULT_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, PROOF_QUALITY, DEFAULT_PITCH, "MS ゴシック")  
End Function
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()  
    Var mRGN As Long  
    Var Cnt As Long  
    Var mBrush As Long  
    Var R As RECT  
    Var Ret As Long
```

```
    hDC = Api_GetDC(GethWnd)  
    mDC = Api_CreateCompatibleDC(Api_GetDC(0))  
    mBitmap = Api_CreateCompatibleBitmap(Api_GetDC(0), GetWidth, GetHeight)
```

```

Ret = Api_SelectObject(mDC, mBitmap)
Ret = Api_SetBkMode(mDC, TRANSPARENT)
Ret = Api_SetRect(R, 0, 0, GetWidth, GetHeight)
Ret = Api_FillRect(mDC, R, Api_GetSysColorBrush(COLOR_WINDOW))

For Cnt = 0 To 330 Step 30
    Ret = Api_DeleteObject(Api_SelectObject(mDC, CreateMyFont(14, Cnt)))
    Ret = Api_TextOut(mDC, GetWidth / 2, GetHeight / 2 - 18, MSG, Len(MSG))
Next Cnt

mRGN = Api_CreateEllipticRgn(0, 0, GetWidth, GetHeight)
Ret = Api_SetWindowRgn(GethWnd, mRGN, True)

Ret = Api_DeleteObject(mRGN)
End Sub

' =====
' =
' =====
Declare Sub MainForm_MouseMove edec1 ()
Sub MainForm_MouseMove()
    Var Ret As Long

    Ret = Api_BitBlt(hDC, 0, 0, GetWidth, GetHeight, mDC, 0, 0, SRCCOPY)
End Sub

' =====
' =
' =====
Declare Sub MainForm_Click edec1 ()
Sub MainForm_Click()
    Var Ret As Long

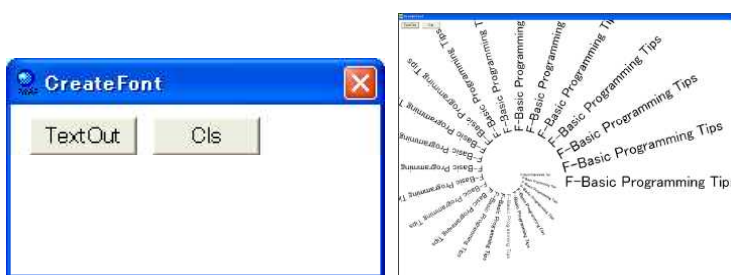
    Ret = Api_ReleaseDC(GethWnd, hDC)
    Ret = Api_DeleteDC(mDC)
    Ret = Api_DeleteObject(mBitmap)
End
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

文字列の回転表示 (II)

CreateFont 指定の属性を持つ論理フォントを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放
TextOut 文字を描画
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```

'=====
'= 文字列の回転表示(11)
'= (CreateFont2.bas)
'=====
#include "Windows.bi"

' 指定の属性を持つ論理フォントを作成
Declare Function Api_CreateFont& Lib "gdi32" Alias "CreateFontA" (ByVal Hei&, ByVal Wid&,
ByVal Esc&, ByVal Ori&, ByVal Wei&, ByVal Ita&, ByVal Und&, ByVal Stk&, ByVal Chr&, ByVal
Out&, ByVal Clp&, ByVal Qua&, ByVal Pit&, ByVal Face$)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Button1 As Object
Var Shared Button2 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14

'=====
'=
'=====
Declare Sub tOut (x As Long, y As Long, Angle As Long, Size As Integer, Txt As String)
Sub tOut (x As Long, y As Long, Angle As Long, Size As Integer, Txt As String)
    Var hDC As Long
    Var hFont As Long
    Var FontMem As Long
    Var Bold As Long
    Var Ret As Long

    hDC = Api_GetDC (GethWnd)

    x = GetWidth / 3
    y = GetHeight / 1.8

    hFont = Api_CreateFont (Size, 0, Angle * 10, 0, 0, 0, 0, 0, 1, 4, &H10, 2, 4, "")

    FontMem = Api_SelectObject (hDC, hFont)
    Ret = Api_TextOut (hDC, x, y, Txt, Len (Txt))
    Ret = Api_SelectObject (hDC, FontMem)
    Ret = Api_DeleteObject (hFont)
    Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    MaximizeWindow
End Sub

'=====
'=
'=====

```

```

Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var x As Long
    Var y As Long
    Var z As Long
    Var n As Integer

    n = 60

    For z = 0 To 360 Step 15
        If n > 10 Then
            n = n - 2
        Else
            If n > 2 Then n = n - 1
        End If

        tOut x, y, z, n, "          F-Basic Programming Tips"
        'Wait 20
    Next z
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Cls
    Refresh
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列の回転表示 (III)

CreateFontIndirect 論理フォントを作成

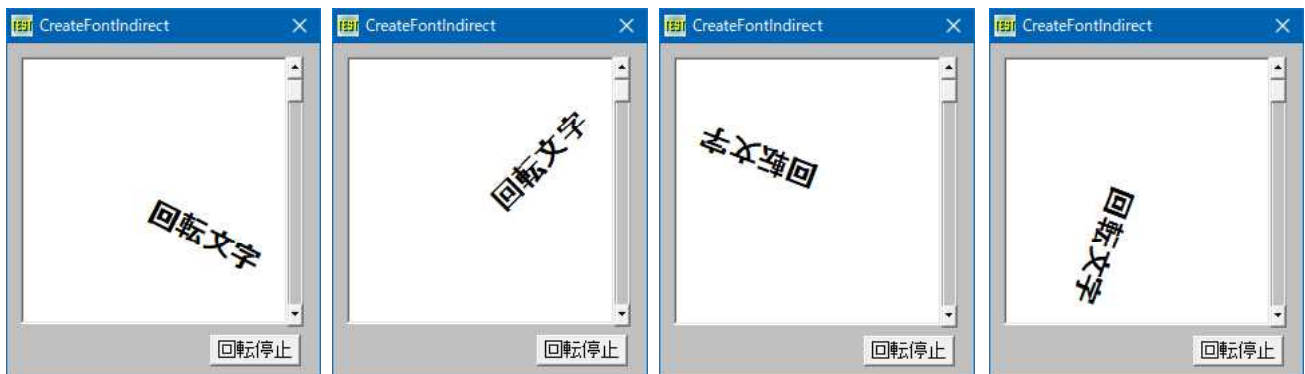
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放

TextOut 文字を描画

GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```

'=====
'= 文字列の回転表示 (III)
'= (CreateFontIndirect3.bas)
'=====
#include "Windows.bi"

```

```

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight           As Long           '文字セルまたは文字の高さ
    lfWidth            As Long           '平均文字幅
    lfEscapement       As Long           '文字送りの方向とx軸との角度
    lfOrientation      As Long           'ベースラインとx軸との角度
    lfWeight           As Long           'フォントの太さ
    lfItalic           As Byte           'イタリック体指定
    lfUnderline        As Byte           '下線付き指定
    lfStrikeOut        As Byte           '打ち消し線付き指定
    lfCharSet          As Byte           'キャラクタセット
    lfOutPrecision     As Byte           '出力精度
    lfClipPrecision    As Byte           'クリッピングの精度
    lfQuality          As Byte           '出力品質
    lfPitchAndFamily   As Byte           'ピッチとファミリ
    lfFaceName         As String * LF_FACESIZE 'フォント名
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Picture1 As Object
Var Shared VScroll1 As Object
Var Shared Timer1 As Object
Var Shared Button1 As Object

Picture1.Attach GetDlgItem("Picture1")
VScroll1.Attach GetDlgItem("VScroll1")
Timer1.Attach GetDlgItem("Timer1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared hDC As Long
Var Shared txt As String
Var Shared i As Integer

'=====
' =
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    txt = "回転文字"
    hDC = Api_GetDC(Picture1.GethWnd)
    VScroll1.SetScrollRange 0, 3600
    VScroll1.SetScrollStep 360, 36
    VScroll1.SetScrollPos 0
End Sub

'=====
' =
'=====
Declare Sub Button1_on edec1 ()

```

```

Sub Button1_on()
    If Timer1.IsEnabled = False Then
        Timer1.Enable True
        Button1.SetWindowText "回転停止"
        VScroll1.EnableWindow False
    Else
        Timer1.Enable False
        Button1.SetWindowText "文字回転"
        VScroll1.EnableWindow True
    End If
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edecl ()
Sub Timer1_Timer()
    Var lf As LOGFONT
    Var prevFont As Long
    Var hFont As Long
    Var Ret As Long

    i = i + 30

    If i >= 1800 * 2 Then i = 0

    Static FONTSIZE As Long
    FONTSIZE = 22

    Picture1.Cls
    lf.lfEscapement = i
    lf.lfFaceName = "ARP丸ゴシック体M" & Chr$(0)
    lf.lfHeight = FONTSIZE
    lf.lfCharSet = 128
    lf.lfWeight = 800

    hFont = Api_CreateFontIndirect(lf)
    prevFont = Api_SelectObject(hDC, hFont)
    Ret = Api_DeleteObject(hFont)

    '文字を描画
    Ret = Api_TextOut(hDC, 100, 100, txt, Len(txt))
End Sub

'=====
'=
'=====
Declare Sub VScroll1_Change edecl ()
Sub VScroll1_Change()
    Var lf As LOGFONT
    Var prevFont As Long
    Var hFont As Long
    Var Ret As Long

    Static FONTSIZE As Long
    FONTSIZE = 22

    Picture1.Cls

    lf.lfEscapement = VScroll1.GetScrollPos
    lf.lfFaceName = "MS 明朝" & Chr$(0)
    lf.lfHeight = FONTSIZE
    lf.lfCharSet = 128

    hFont = Api_CreateFontIndirect(lf)
    prevFont = Api_SelectObject(hDC, hFont)
    Ret = Api_DeleteObject(hFont)

    '文字を描画
    Ret = Api_TextOut(hDC, 100, 100, txt, Len(txt))

```

```

End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose cdecl ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    Ret = Api_ReleaseDC (Picture1.GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列の加工 (LOGFONT構造体)

LOGFONT構造体に値を入れ変化を確認します。

CreateFontIndirect 論理フォントを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
CreateSolidBrush ソリッドカラーで論理ブラシを作成
BeginPath 指定されたデバイスコンテキストのパスを作成
TextOut 文字を描画
EndPath パスブラケットのクローズ
StrokeAndFillPath パスの輪郭を描画
FillPath パスを塗り潰す
DeleteObject システムリソースの解放
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストの解放



```

'=====
'= 文字列の加工
'= (Paths.bas)
'=====

#include "Windows.bi"

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight           As Long
    lfWidth            As Long
    lfEscapement       As Long
    lfOrientation      As Long
    lfWeight           As Long
    lfItalic           As Byte
    lfUnderline        As Byte
    lfStrikeOut        As Byte
    lfCharSet          As Byte
    lfOutPrecision     As Byte
    lfClipPrecision    As Byte
    '文字セルまたは文字の高さ
    '平均文字幅
    '文字送りの方向とx軸との角度
    'ベースラインとx軸との角度
    'フォントの太さ
    'イタリック体指定
    '下線付き指定
    '打ち消し線付き指定
    'キャラクタセット
    '出力精度
    'クリッピングの精度

```



```

    lfQuality As Byte '出力品質
    lfPitchAndFamily As Byte 'ピッチとファミリ
    lfFaceName(LF_FACESIZE) As Byte 'フォント名
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' ソリッドカラーで論理ブラシを作成
Declare Function Api_CreateSolidBrush& Lib "gdi32" Alias "CreateSolidBrush" (ByVal
crColor&)

' 指定されたデバイスコンテキストのパスの作成
Declare Function Api_BeginPath& Lib "gdi32" Alias "BeginPath" (ByVal hDC&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' パスブラケットのクローズ
Declare Function Api_EndPath& Lib "gdi32" Alias "EndPath" (ByVal hDC&)

' パスの輪郭を描画
Declare Function Api_StrokePath& Lib "gdi32" Alias "StrokePath" (ByVal hDC&)

' パスの描画と塗りつぶし
Declare Function Api_StrokeAndFillPath& Lib "gdi32" Alias "StrokeAndFillPath" (ByVal
hDC&)

' パスを塗りつぶす
Declare Function Api_FillPath& Lib "gdi32" Alias "FillPath" (ByVal hDC&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得。
その後、GDI 関数を使って、返されたデバイスコンテキスト内で描画を行える
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define FW_DONTCARE 0 'LOGFONT(lfWeight)
#define FW_BOLD 700 'LOGFONT(lfWeight)
Var Shared Picture1 As Object
Var Shared Text(1) As Object
Var Shared Edit(1) As Object
Var Shared Check(3) As Object
Var Shared Button1 As Object

Picture1.Attach GetDlgItem("Picture1")
For i = 0 To 3
    If i < 2 Then
        Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
        Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
    End If
    Check(i).Attach GetDlgItem("Check" & Trim$(Str$(i + 1)))
    Check(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====

```

```

Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    ShowWindow -1
    Edit(0).SetWindowText "50"
    Edit(1).SetWindowText "0"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var LF As LOGFONT
    Var rFont As Long
    Var hBrush As Long
    Var oldBrush As Long
    Var crColor As Long
    Var hDC As Long
    Var sText As String
    Var size As Integer
    Var angle As Integer
    Var Ret As Long

    hDC = Api_GetDC(Picture1.GethWnd)
    Picture1.Cls

    sText = "北海道札幌市"
    size = Val(Edit(0).GetWindowText)
    angle = Val(Edit(1).GetWindowText)
    crColor = rgb(255, 255, 0)

    LF.lfCharSet = 128                                '日本語 (SHIFTJIS_CHARSET)
    LF.lfEscapement = angle * 10                       '角度設定
    LF.lfHeight = size                                 '文字高さ設定
    If Check(0).GetCheck = 1 Then LF.lfWeight = FW_BOLD Else LF.lfWeight = FW_DONTCARE
    LF.lfItalic = Check(1).GetCheck                    'イタリック
    LF.lfUnderline = Check(2).GetCheck                 '下線
    LF.lfStrikeOut = Check(3).GetCheck                '打ち消し線

    rFont = Api_CreateFontIndirect(LF)                '論理フォントの作成
    Ret = Api_SelectObject(hDC, rFont)                '指定されたデバイスコンテキストのオブジェクトを選択
    Ret = Api_TextOut(hDC, 0, 0, sText, Len(sText)) 'フォームの指定位置に指定角度・大きさで文字列
                                                    を描画
    hBrush = Api_CreateSolidBrush(crColor) '塗り潰し色
    oldBrush = Api_SelectObject(hDC, hBrush) '現在の塗り潰し色を設定

    Ret = Api_BeginPath(hDC)
    Ret = Api_TextOut(hDC, 0, 0, sText, Len(sText))
    Ret = Api_EndPath(hDC)

    Ret = Api_StrokePath(hDC)                        'パスの輪郭を描画
    Ret = Api_BeginPath(hDC)
    Ret = Api_TextOut(hDC, 0, LF.lfHeight + 1, sText, Len(sText))
    Ret = Api_EndPath(hDC)

    Ret = Api_FillPath(hDC)                          'パスを塗りつぶす
    Ret = Api_BeginPath(hDC)
    Ret = Api_TextOut(hDC, 0, LF.lfHeight * 2 + 1, sText, Len(sText))
    Ret = Api_EndPath(hDC)

    Ret = Api_StrokeAndFillPath(hDC)                 'パスの描画と塗りつぶし
    Ret = Api_SelectObject(hDC, oldBrush)            'ブラシを元に戻す

    Ret = Api_DeleteObject(hBrush)
    Ret = Api_ReleaseDC(Picture1.GethWnd, hDC)
End Sub

'=====
'=
'=====

```

```
While 1
    WaitEvent
Wend
Stop
End
```

文字列の均等割付

CreateFontIndirect 論理フォントを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
TextOut 文字を描画
SetTextJustification システムがテキスト文字列内のブレイク文字に追加する間隔の量を指定
GetTextExtentPoint32 文字列全体の幅と高さを取得
GetClientRect ウィンドウのクライアント領域の座標を取得
lstrlen 指定された文字列のバイトまたは文字の長さを返す
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```
'=====
'= 文字列の均等割付
'= (SetTextJustification.bas)
'=====
#include "Windows.bi"

Type SIZE
    cx As Long
    cy As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight As Long
    lfWidth As Long
    lfEscapement As Long
    lfOrientation As Long
    lfWeight As Long
    lfItalic As byte
    lfUnderline As byte
    lfStrikeOut As byte
    lfCharSet As byte
    lfOutPrecision As byte
    lfClipPrecision As byte
    lfQuality As byte
    lfPitchAndFamily As byte
    lfFaceName As String * LF_FACESIZE
End Type
```

'幅を指定
 '高さを指定

'文字セルまたは文字の高さ
 '平均文字幅
 '文字送りの方向とx軸との角度
 'ベースラインとx軸との角度
 'フォントの太さ
 'イタリック体指定
 '下線付き指定
 '打ち消し線付き指定
 'キャラクタセット
 '出力精度
 'クリッピングの精度
 '出力品質
 'ピッチとファミリ
 'フォント名

```

Type TEXTMETRIC
    tmHeight          As Long          'フォントの高さ
    tmAscent          As Long          'ベースラインから一番上までの高さ
    tmDescent         As Long          'ベースラインから一番下までの高さ
    tmInternalLeading  As Long          'tmHeightメンバが示す高さに含まれる、上部スペースの高さが格納
    tmExternalLeading  As Long          'フォントを描画する際の、行同士の幅
    tmAveCharWidth    As Long          'フォントの平均の幅
    tmMaxCharWidth    As Long          'フォントを描画する際もっとも広い幅
    tmWeight          As Long          'フォントの太さ
    tmOverhang        As Long          '太字やイタリック体などのフォントに付加される幅
    tmDigitizedAspectX As Long          'デバイス コンテキストの水平アスペクト値
    tmDigitizedAspectY As Long          'デバイス コンテキストの垂直アスペクト値
    tmFirstChar       As Byte          'フォントに含まれる文字中の、先頭の文字コード
    tmLastChar        As Byte          'フォントに含まれる文字中の、最後の文字コード
    tmDefaultChar     As Byte          'フォントに含まれない文字を描画するための文字コード
    tmBreakChar       As Byte          '単語と単語の間を示す文字コード
    tmItalic          As Byte          'フォントがイタリック体のときは0以外の値が、そうでないと
    tmUnderlined      As Byte          '下線付きフォントのときは0以外の値が、そうでないときは0
    tmStruckOut       As Byte          '打ち消しフォントのときは0以外の値が、そうでないときは0
    tmPitchAndFamily  As Byte          '下位 4 ビットにフォントのピッチおよび属性
    tmCharSet         As Byte          'フォントの文字セット
End Type

```

' 論理フォントを作成

```

Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

```

' 指定されたデバイスコンテキストのオブジェクトを選択

```

Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

```

' 文字を描画

```

Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

```

' システムがテキスト文字列内のブレイク文字に追加する間隔の量を指定

```

Declare Function Api_SetTextJustification& Lib "gdi32" Alias "SetTextJustification"
(ByVal hDC&, ByVal nBreakExtra&, ByVal nBreakCount&)

```

' 文字列全体の幅と高さを取得

```

Declare Function Api_GetTextExtentPoint32& Lib "gdi32" Alias "GetTextExtentPoint32A"
(ByVal hDC&, ByVal lpsz$, ByVal cbString&, lpSize As SIZE)

```

' ウィンドウのクライアント領域の座標を取得

```

Declare Function Api_GetClientRect& Lib "user32" Alias "GetClientRect" (ByVal hWnd&,
lpRect As RECT)

```

' 指定された文字列のバイトまたは文字の長さを返す

```

Declare Function Api_lstrlen& Lib "Kernel32" Alias "lstrlenA" (ByVal lpString$)

```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```

Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

```

' デバイスコンテキストを解放

```

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

```

```

Var Shared Edit(1) As Object
Var Shared Radio(2) As Object
Var Shared Picture1 As Object
Var Shared Button1 As Object

```

```

Picture1.Attach GetDlgItem("Picture1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
For i = 0 To 2
    If i < 2 Then
        Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
        Edit(i).SetFontSize 14
    End If
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1)))

```

```

        Radio(i).SetFontSize 14
Next

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hDC As Long
    Var sz As SIZE
    Var rc As RECT
    Var lf As LOGFONT
    Var rFont As Long
    Var bCount As Long
    Var txt As String
    Var txt2 As String
    Var temp As String
    Var bc As String
    Var i As Integer
    Var Ret As Long

    Picture1.Cls

    lf.lfCharSet = 128                                '日本語 (SHIFTJIS_CHARSET)
    lf.lfHeight = 16                                  '文字高さ設定
    lf.lfFaceName = Edit(1).GetWindowText & Chr$(0) 'デフォルト指定フォント名
    If Radio(1).GetCheck Then lf.lfWeight = 700      '太字
    lf.lfItalic = Radio(2).GetCheck                  '斜体

    txt = Edit(0).GetWindowText                      '均等割り付けする文字列

    '空白の有無によりブレイク文字の数を取得
    If InStr(txt, " ") = 0 Then
        For i = 1 To KLen(txt) - 1
            temp = KMid$(txt, i, 1)
            txt2 = txt2 & temp
            txt2 = txt2 & " "
            bCount = bCount + 1
        Next
        txt2 = txt2 & KRight$(txt, 1)
        txt = txt2
    Else
        For i = 1 To Len(txt)
            Temp = KMid$(txt, i, 1)
            If temp = " " then bCount = bCount + 1
        Next
    End If

    'デバイスコンテキストのハンドルを取得
    hDC = Api_GetDC(Picture1.GethWnd)

    '論理フォントの作成
    rFont = Api_CreateFontIndirect(lf)

    '指定されたデバイスコンテキストのオブジェクトを選択
    Ret = Api_SelectObject(hDC, rFont)

    'クライアント領域の座標を取得
    Ret = Api_GetClientRect(Picture1.GethWnd, rc)

    '文字列の論理単位を取得
    Ret = Api_GetTextExtentPoint32(hDC, txt, Api_lstrlen(txt), sz)

    'ブレイク文字に追加する間隔の量を取得
    Ret = Api_SetTextJustification(hDC, rc.Right - sz.cx, bCount)

    If Ret <> 0 Then

        'デバイスコンテキストに文字列を描画
        Ret = Api_TextOut(hDC, 0, 3, txt, Api_lstrlen(txt))

```

```

End If

' デバイスコンテキストの解放
Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

文字列のスクロール

エディットボックスの文字列表示位置をスクロールします。

SendMessage ウィンドウにメッセージを送信

SB_LINERIGHT (1) 右矢印ボタンクリック

SB_LINELEFT (0) 左矢印ボタンクリック

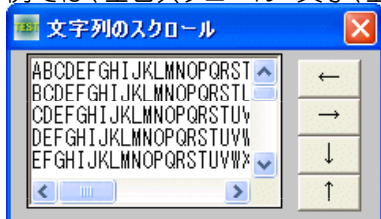
SB_LINEUP (0) 上向きの矢印ボタンクリック

SB_LINEDOWN (1) 下向きの矢印ボタンクリック

WM_HSCROLL (&H114) 水平スクロールバーを調整している

WM_VSCROLL (&H115) ユーザーが垂直スクロールバーを調整した

例では、左右スクロール5文字、上下スクロール1行に設定しています。



```

' =====
' = 文字列のスクロール
' = (TextScroll.bas)
' =====
#include "Windows.bi"

```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```

Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, ByVal lParam&)

```

```

#define SB_LINERIGHT 1

```

```

#define SB_LINELEFT 0

```

```

#define SB_LINEUP 0

```

```

#define SB_LINEDOWN 1

```

```

#define WM_HSCROLL &H114

```

```

#define WM_VSCROLL &H115

```

' 右矢印ボタンクリック

' 左矢印ボタンクリック

' 上向きの矢印ボタンクリック

' 下向きの矢印ボタンクリック

' 水平スクロールバーを調整している

' ユーザーが垂直スクロールバーを調整した

```

Var Shared Edit1 As Object

```

```

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14

```

```

' =====
' =
' =====

```

```

Declare Function EditHScroll(hWnd As Long, bScrollRight As Integer, LenTxt As Long) As
Long

```

```

Function EditHScroll(hWnd As Long, bScrollRight As Integer, LenTxt As Long) As Long

```

```

    Var Txt As Long

```

```

    If bScrollRight Then

```

```

        For Txt = 1 To LenTxt

```

```

        EditHScroll = Api_SendMessage(hWnd, WM_HSCROLL, SB_LINERIGHT, ByVal 0)
    Next
Else
    For Txt = 1 To LenTxt
        EditHScroll = Api_SendMessage(hWnd, WM_HSCROLL, SB_LINELEFT, ByVal 0)
    Next
End If
End Function

'=====
'=
'=====
Declare Function EditVScroll(hWnd As Long, bScrollUp As Integer, LenTxt As Long) As Long
Function EditVScroll(hWnd As Long, bScrollUp As Integer, LenTxt As Long) As Long
    Var Txt As Long

    If bScrollUp Then
        For Txt = 1 To LenTxt
            EditVScroll = Api_SendMessage(hWnd, WM_VSCROLL, SB_LINEUP, ByVal 0)
        Next
    Else
        For Txt = 1 To LenTxt
            EditVScroll = Api_SendMessage(hWnd, WM_VSCROLL, SB_LINEDOWN, ByVal 0)
        Next
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var txt As String

    For i = 0 To 19
        For j = 65+i To 120+i
            txt = txt & Chr$(j)
        Next
        txt = txt & Chr$(13, 10)
    Next
    Edit1.SetWindowText txt
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long
    Ret = EditHScroll(Edit1.GethWnd, False, 5)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long
    Ret = EditHScroll(Edit1.GethWnd, True, 5)
End Sub

'=====
'=
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on()
    Var Ret As Long
    Ret = EditVScroll(Edit1.GethWnd, False, 1)
End Sub

```

```

'=====
'=
'=====
Declare Sub Button4_on edec1 ()
Sub Button4_on ()
    Var Ret As Long
    Ret = EditVScroll(Edit1.GethWnd, True, 1)
End Sub

'=====
'=
'=====

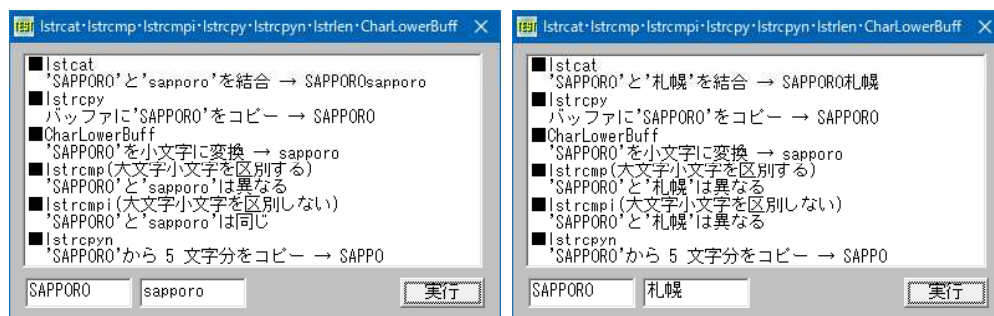
While 1
    WaitEvent
Wend
Stop
End

```

文字列操作

文字列を操作する関数をテストします

lstrcat ある文字列の末尾に別の文字列を結合
lstrcmp 2つの文字列を比較 (大文字と小文字を区別する)
lstrcmpi 2つの文字列を比較 (大文字と小文字を区別しない)
lstrcpy 文字列をコピーする
lstrcpyn バッファにソース文字列から指定した数の文字をコピーする
lstrlen 指定された文字列のバイトまたは文字の長さを返す
CharLowerBuff バッファの大文字を小文字に変換する



```

'=====
'= 文字列操作
'= (StringFunctions.bas)
'=====

#include "Windows.bi"

```

' ある文字列の末尾に別の文字列を結合

```

Declare Function Api_lstrcat& Lib "kernel32" Alias "lstrcatA" (ByVal lpString1$, ByVal lpString2$)

```

' 2つの文字列を比較 (敏感なケース)

```

Declare Function Api_lstrcmp& Lib "kernel32" Alias "lstrcmpA" (ByVal lpString1$, ByVal lpString2$)

```

' 2つの文字列を比較 (敏感ではないケース)

```

Declare Function Api_lstrcmpi& Lib "kernel32" Alias "lstrcmpiA" (ByVal lpString1$, ByVal lpString2$)

```

' 文字列をコピーする

```

Declare Function Api_lstrcpy& Lib "Kernel32" Alias "lstrcpyW" (lpszString1 As Any, lpszString2 As Any)

```

' バッファにソース文字列から指定した数の文字をコピーする

```

Declare Function Api_lstrcpyn& Lib "kernel32" Alias "lstrcpynA" (ByVal lpString1$, ByVal lpString2$, ByVal iMaxLength&)

```


' 指定された文字列のバイトまたは文字の長さを返す

```
Declare Function Api_lstrlen& Lib "kernel32" Alias "lstrlenA" (ByVal lpString$)
```

' バッファの大文字を小文字に変換する

```
Declare Function Api_CharLowerBuff& Lib "user32" Alias "CharLowerBuffA" (ByVal lpsz$,  
ByVal cchLength&)
```

```
Var Shared List1 As Object  
Var Shared Edit1 As Object  
Var Shared Edit2 As Object  
Var Shared Button1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14  
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14  
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
' =====  
' =  
' =====
```

```
Declare Sub Button1_on edec1 ()  
Sub Button1_on()
```

```
    Var Txt1 As String  
    Var Txt2 As String  
    Var Buff As String  
    Var Ret As Long
```

```
    Txt1 = Edit1.GetWindowText  
    Txt2 = Edit2.GetWindowText
```

```
    List1.ResetContent
```

' 2つの文字列を結合

```
List1.AddString "■lstrcat"  
Buff = String$(Api_lstrlen(Txt1) & Api_lstrlen(Txt2), 0)  
Ret = Api_lstrcat(Buff, Txt1) ' バッファにTxt1を結合  
Ret = Api_lstrcat(Buff, Txt2) ' 更にTxt2を結合  
List1.AddString " '" & Txt1 & "'と'" & Txt2 & "'を結合 → " & Buff
```

' 文字列をバッファにコピー

```
List1.AddString "■lstrcpy"  
Buff = String$(Api_lstrlen(Txt1), 0) ' バッファ  
Ret = Api_lstrcpy(Buff, Txt1) ' バッファにTempをコピー  
List1.AddString " バッファに'" & Txt1 & "'をコピー → " & Buff
```

' 文字列を小文字に変換

```
List1.AddString "■CharLowerBuff"  
Buff = Txt1  
Ret = Api_CharLowerBuff(Buff, Api_lstrlen(Buff))  
List1.AddString " '" & Txt1 & "'を小文字に変換 → " & Buff
```

' 文字列が同じであるかをチェック:大文字小文字を区別する

```
List1.AddString "■lstrcmp(大文字小文字を区別する)"  
If Api_lstrcmp(Txt1, Txt2) = 0 Then  
    List1.AddString " '" & Txt1 & "'と'" & Txt2 & "'は同じ"  
Else  
    List1.AddString " '" & Txt1 & "'と'" & Txt2 & "'は異なる"  
End If
```

' 文字列が同じであるかをチェック:大文字小文字を区別しない

```
List1.AddString "■lstrcmpi(大文字小文字を区別しない)"  
If Api_lstrcmpi(Txt1, Txt2) = 0 Then  
    List1.AddString " '" & Txt1 & "'と'" & Txt2 & "'は同じ"  
Else  
    List1.AddString " '" & Txt1 & "'と'" & Txt2 & "'は異なる"  
End If
```

' バッファにソース文字列から指定した数の文字をコピー

```
List1.AddString "■lstrcpyn"  
Ret = Api_lstrcpyn(Buff, Txt1, 5 + 1) ' 5文字 + 1 (Nullの分を含める)
```

```
List1.AddString " '" & Txt1 & "'から 5 文字分をコピー → " & Buff
End Sub

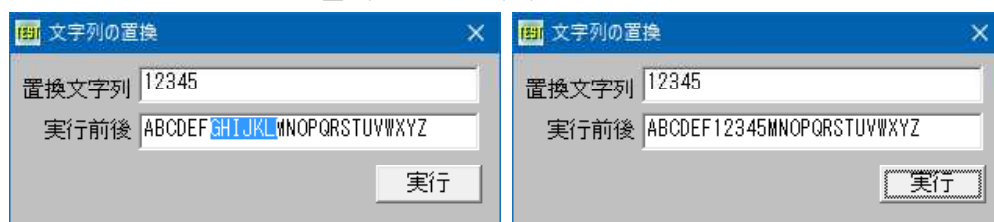
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

文字列の置換 (1)

SendMessage ウィンドウにメッセージを送信

EM_REPLACESEL (&HC2) エディットコントロール内の現在の選択項目を置き換える

例では、選択した文字列が置換文字列に変換されたことを表しています。



```
'=====
'= 文字列の置換
'= (EM_REPLACESEL.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

#define EM_REPLACESEL &HC2                                ' エディットコントロール内の現在の選択項目を置き換える

Var Shared Text(1) As Object
Var Shared Edit(1) As Object
Var Shared Button1 As Object

For i = 0 To 1
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sReplaceSel As String
    Var Ret As Long

    ' 置換文字列を指定
    sReplaceSel = Edit(0).GetWindowText

    ' 選択文字列を置換
    Ret = Api_SendMessage(Edit(1).GethWnd, EM_REPLACESEL, 0, sReplaceSel)
End Sub

'=====
'=
'=====
```

```
While 1
    WaitEvent
Wend
Stop
End
```

文字列の置換 (II)

文字列置換関数

Replace (元の文字列, 修正する文字列, 置換文字列)

[declare function]を「Declare Function」に置換する例を示しています。



```
'=====
'= 文字列の置換 (II) (置換関数)
'= (Replace.bas)
'=====
#include "Windows.bi"

Var Shared Text(4) As Object
Var Shared Edit(2) As Object

For i = 0 To 4
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    If i < 2 Then
        Text(i).SetFontSize 14
    Else
        Text(i).SetFontSize 12
    End If
Next
For i = 0 To 2
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1)))
    Edit(i).SetFontSize 14
Next

Var Shared FL As String
Var Shared Buffer As String

'=====
'= 置換関数
'= sOriginal:原本 sCorrection:修正文字 sReplace:置換文字
'=====
Declare Function Replace(strOriginal As String, strCororrection As String,
strRepeplace As String) As String
Function Replace(strOriginal As String, strCororrection As String, strRepeplace As
String) As String
    Var txt As String
    Var iPos As Integer

    txt = ""
    Do
        iPos = InStr(strOriginal, strCororrection)
        If iPos = 0 Then Exit do

        txt = txt & Left$(strOriginal, iPos - 1) & strRepeplace
        strOriginal = Mid$(strOriginal, iPos + Len(strCororrection))
    Loop
```

```

    txt = txt & strOriginal
    Replace = txt
End Function

'=====
'= 置換実行
'=====
Declare Sub BmpButton2_on edec1 ()
Sub BmpButton2_on ()
    If Edit(0).GetWindowText = "" Then
        A% = MsgBox("Attention!", "検索文字列を指定してください！", 0, 2)
        Exit Sub
    End If

    Var strOrg As String
    Var strCor As String
    Var strRep As String

    strOrg = Edit(2).GetWindowText
    strCor = Edit(0).GetWindowText
    strRep = Edit(1).GetWindowText

    '-----
    Buffer = Replace(strOrg, strCor, strRep)
    '-----

    Edit(2).SetWindowText Buffer
    SetFocus
End Sub

'=====
'= ファイルを開く
'=====
Declare Sub BmpButton1_on edec1 ()
Sub BmpButton1_on ()
    Var hFile As Integer

    For i = 0 To 2
        Edit(i).SetWindowText ""
    Next

    FL = WinOpenDlg("ファイルのオープン", "*.txt;*.bas;*.htm", "テキスト(*.txt);basファイル(*.bas);htmファイル(*.htm)", 0)

    If FL <> Chr$(&H1B) Then
        hFile = FreeFile
        Open FL For BinInp As hFile
        Buffer = Space$(Lof(hFile))
        FRead hFile, Buffer
        Buffer = JConv$(Buffer, 0, 2)

        Edit(2).SetWindowText Buffer
        Close hFile
    Else
        FL = ""
    End If
End Sub

'=====
'= 名前を付けて保存
'=====
Declare Sub BmpButton3_on edec1 ()
Sub BmpButton3_on ()
    Var hFile As Integer

    FL = WinSaveDlg("名前を付けて保存", FL, "テキスト(*.txt);basファイル(*.bas);htmファイル(*.htm)", 0)

    If FL <> Chr$(&H1B) Then
        hFile = FreeFile
        Open FL For BinIO As hFile

```

```

        If Lof(hFile) <> 0 Then
            Close hFile
            Kill FL
            Open FL For BinIO As hFile
        End If

        fWrite hFile, Buffer
        Close hFile
    End If
End Sub

'=====
'= リサイズ
'=====
Declare Sub MainForm_Resize edec1 ()
Sub MainForm_Resize()
    Line( 0, 1) - (GetWidth, 1), , 1 : Line( 0, 2) - (GetWidth, 2), , 15
    Line( 0, 33) - (GetWidth, 33), , 1 : Line( 0, 34) - (GetWidth, 34), , 15

    If GetWidth < 435 Or GetHeight < 280 Then
        SetWindowSize 435, 280
    End If

    Edit(0).SetWindowSize GetWidth - 134, 24
    Edit(1).SetWindowSize GetWidth - 134, 24
    Edit(2).SetWindowSize GetWidth - 32, GetHeight - 148
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

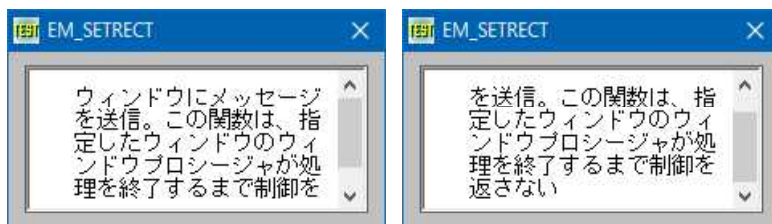
```

文字列の表示領域を設定

エディットボックス内の文字列表示領域を設定します。

SendMessage ウィンドウにメッセージを送信

EM_SETRECT (&HB3) テキスト表示領域を設定



```

'=====
'= 文字列の表示領域を設定
'= (EM_SETRECT.bas)
'=====
#include "Windows.bi"

Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type

```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal  
wMsg&, ByVal wParam&, lParam As Any)
```

```
#define EM_SETRECT &HB3
```

'テキスト表示領域を設定

```
Var Shared Edit1 As Object
```

```
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edecl ()
```

```
Sub MainForm_Start()
```

```
    Var rct As RECT
```

```
    Var x As Long
```

```
    Var y As Long
```

```
    Var Ret As Long
```

'表示する領域

```
    x = Edit1.GetWidth / 8
```

```
    y = Edit1.GetHeight / 10
```

```
    rct.Left = x
```

```
    rct.Top = y
```

```
    rct.Right = Edit1.GetWidth - x
```

```
    rct.Bottom = Edit1.GetHeight - y
```

```
    Ret = Api_SendMessage(Edit1.GetHwnd, EM_SETRECT, ByVal 0, rct)
```

```
    Edit1.SetFocus
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

文字列のリバース

入力した文字列を逆に表示します。
「どういう使い方があるの？」って言うのは無しで..



```
'=====
'= 文字列のリバース
'= (ReverseString.bas)
'=====
```

```
#include "Windows.bi"
```

```
Var Shared Text1 As Object
```

```
Var Shared Edit1 As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
```

```
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
```

```

'=====
'=
'=====
Declare Function RevStr(sString As String) As String
Function RevStr(sString As String) As String
    Var i As Long
    Var nLen As Long
    Var sBuffer As String

    nLen = Len(sString)
    For i = nLen To 1 step -1
        sBuffer = sBuffer & KMid$(sString, i, 1)
    Next i

    RevStr = sBuffer
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var MyStr As String

    MyStr = Edit1.GetWindowText
    Text1.SetWindowText RevStr(MyStr)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を大文字・小文字に変換 (文字数指定)

先頭からの文字数を指定し、大文字は小文字に、小文字は大文字に変換します。

CharLowerBuff 文字数を指定し、大文字を小文字に変換

CharUpperBuff 文字数を指定し、小文字を大文字に変換



```

'=====
'= 文字列を大文字・小文字に変換 (文字数指定)
'= (CharLowerBuff2.bas)
'=====
#include "Windows.bi"

' バッファの大文字を小文字に変換する (処理する文字数の指定が可能)
Declare Function Api_CharLowerBuff& Lib "user32" Alias "CharLowerBuffA" (ByVal lpsz$,
ByVal cchLength&)

' バッファの小文字を大文字に変換する (処理する文字数の指定が可能)
Declare Function Api_CharUpperBuff& Lib "user32" Alias "CharUpperBuffA" (ByVal lpsz$,
ByVal cchLength&)

```

```

Var Shared Edit1 As Object
Var Shared Edit2 As Object
var shared Button1 As Object
var Shared Radio1 As Object
var Shared Radio2 As Object

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Radio1.Attach GetDlgItem("Radio1") : Radio1.SetFontSize 14
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14

'=====
'=
'=====
Declare Function Index bdecl () As Integer
Function Index()
    Index = Val (Mid$ (GetDlgRadioSelect ("Radio1"), 6)) -1
End Function

'=====
'=
'=====
Declare Sub Button1_on edecl ()
Sub Button1_on()
    Var txt As String
    Var Ret As Long

    txt = Edit1.GetWindowText

    If Index = 0 Then

        '先頭から指定文字目まで大文字に変換
        Ret = Api_CharUpperBuff (txt, Val (Edit2.GetWindowText))
    Else

        '先頭から指定文字目まで小文字に変換
        Ret = Api_CharLowerBuff (txt, Val (Edit2.GetWindowText))
    End If
    Edit1.SetWindowtext txt
End Sub
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を大文字・小文字に変換

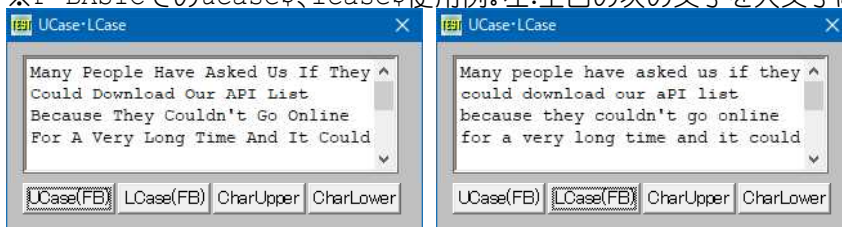
入力されている文字を大文字または小文字に変換します。

[CharUpper](#)は半角英数文字列を大文字に、[CharLower](#)は小文字に変換します。F-Basicでのucase\$、およびlcase\$と同じです。

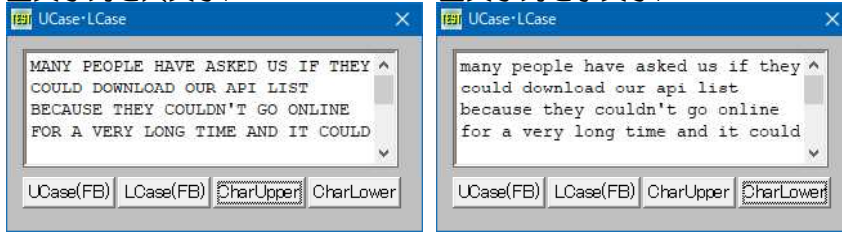
[CharUpper](#) 文字列を大文字に変換

[CharLower](#) 文字列を小文字に変換

※F-BASICでのucase\$、lcase\$使用例。左:空白の次の文字を大文字に。右:空白の次の文字を小文字に



※ApiでのCharUpper、CharLower使用例
 全文字列を大文字に 全文字列を小文字に



```
'=====
'= 大文字・小文字に変換
'= (Ucase.bas)
'=====
#include "Windows.bi"

' 文字列を小文字に変換
Declare Function Api_CharLower& Lib "user32" Alias "CharLowerA" (ByVal lpsz$)

' 文字列を大文字に変換
Declare Function Api_CharUpper& Lib "user32" Alias "CharUpperA" (ByVal lpsz$)

Var Shared Str As String
Var Shared Upper As String
Var Shared Lower As String

Var Shared Edit1 As Object
Var Shared Button(3) As Object

Edit1.attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
For i = 0 To 3
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1))) : Button(i).SetFontSize 14
Next i

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Str = GetDlgItemText("Edit1")
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var i As Integer

    If Str <> "" Then
        Mid$(Str, 1, 1) = Ucase$(Mid$(Str, 1, 1))
        For i = 1 To Len(Str) - 2
            If Mid$(Str, i, 2) = Chr$(13) & Chr$(10) Then
                Mid$(Str, i + 2, 1) = Ucase$(Mid$(Str, i + 2, 1))
            End If
            If Mid$(Str, i, 1) = " " Then
                Mid$(Str, i + 1, 1) = Ucase$(Mid$(Str, i + 1, 1))
            End If
        Next
        Edit1.SetWindowText Str
    End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var i As Integer
```

```

If Str <> "" Then
    Mid$(Str, 1, 1) = Ucase$(Mid$(Str, 1, 1))
    For i = 1 To Len(Str) - 2
        If Mid$(Str, i, 2) = Chr$(13) & Chr$(10) Then
            Mid$(Str, i + 2, 1) = Lcase$(Mid$(Str, i + 2, 1))
        End If
        If Mid$(Str, i, 1) = " " Then
            Mid$(Str, i + 1, 1) = Lcase$(Mid$(Str, i + 1, 1))
        End If
    Next
    Edit1.SetWindowText Str
End If
End Sub

'=====
'=
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on()
    Var Ret As Long

    Ret = Api_CharUpper(Str)
    Edit1.SetWindowText Str
End Sub

'=====
'=
'=====
Declare Sub Button4_on edec1 ()
Sub Button4_on()
    Var Ret As Long

    Ret = Api_CharLower(Str)
    Edit1.SetWindowText Str
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列をグレー表示(1)

文字列をグレー表示します。

GetSysColor システムの背景色を取得

SetBkColor デバイスコンテキストの背景色を設定

GetPixel 指定された座標のピクセルのRGB値を取得

CreateFontIndirect 論理フォントを作成

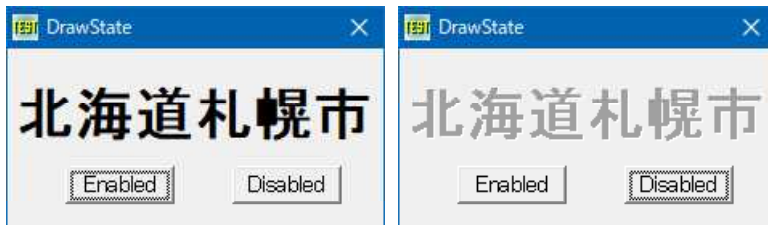
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

DrawState イメージを加工して表示



左:フォーム設定 右:Buttonのシステムカラーを取得し、その色をフォーム背景色に。DrawStateのみの場合文字背景色が白になる

左:Enableボタンを選択した場合 右:Disableボタンを選択した状態



```
'=====
'= 文字列をグレー表示
'= (DrawState2.bas)
'=====
#include "Windows.bi"

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight          As Long
    lfWidth           As Long
    lfEscapement       As Long
    lfOrientation      As Long
    lfWeight           As Long
    lfItalic           As byte
    lfUnderline        As byte
    lfStrikeOut        As byte
    lfCharSet          As byte
    lfOutPrecision     As byte
    lfClipPrecision    As byte
    lfQuality          As byte
    lfPitchAndFamily   As byte
    lfFaceName (LF_FACESIZE) As byte
End Type

' システムの背景色を取得
Declare Function Api_GetSysColor& Lib "user32" Alias "GetSysColor" (ByVal nIndex&)

' デバイスコンテキストの背景色を設定
Declare Function Api_SetBkColor& Lib "gdi32" Alias "SetBkColor" (ByVal hDC&, ByVal crColor&)

' 指定された座標のピクセルのRGB値を取得
Declare Function Api_GetPixel& Lib "gdi32" Alias "GetPixel" (ByVal hDC&, ByVal X&, ByVal Y&)

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal hObject&)

' イメージを加工して表示
Declare Function Api_DrawState& Lib "user32" Alias "DrawStateA" (ByVal hDC&, ByVal hBrush&, ByVal lpDrawStateProc&, ByVal lData$, ByVal wData&, ByVal X&, ByVal Y&, ByVal cx&, ByVal cy&, ByVal fFlags&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define TXT_DISABLED &H20
#define TXT_PREFIXTEXT &H2
#define TXT_TEXT &H1
#define COLOR_BTNFACE 15

' コマンドボタンの表面色

Var Shared XPos As Long
```

```

Var Shared YPos As Long
Var Shared hDC As Long
Var Shared rgbColor As Long

'=====
'=
'=====
Declare Sub sDrawTextEnabledDisabled(Index As Integer)
Sub sDrawTextEnabledDisabled(Index As Integer)
    Var LF As LOGFONT
    Var rFont As Long
    Var Size As Integer
    Var lDrawStateFlag As Long
    Var Txt As String
    Var BkCol As Long

    Txt = "北海道札幌市"
    Size = 40
    XPos = 5
    YPos = 20
    LF.lfHeight = Size
    LF.lfWeight = 700
    rFont = Api_CreateFontIndirect(LF)

    hDC = Api_GetDC(GethWnd)
    If Index = 1 Then lDrawStateFlag = TXT_DISABLED

    cls

    BkCol = Api_GetPixel(hDC, 0, 0)
    Ret = Api_SetBkColor(hDC, BkCol)

    Ret = Api_SelectObject(hDC, rFont)
    Ret = Api_DrawState(hDC, 0, 0, Txt, Len(Txt), XPos, YPos, 0, 0, TXT_TEXT Or
lDrawStateFlag)
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    rgbColor = Api_GetSysColor(COLOR_BTNFACE)
    SetBackColor rgbColor
    ShowWindow -1

    Index = 0
    sDrawTextEnabledDisabled 0
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Index = 0
    sDrawTextEnabledDisabled 0
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Index = 1
    sDrawTextEnabledDisabled 1
End Sub

```

```

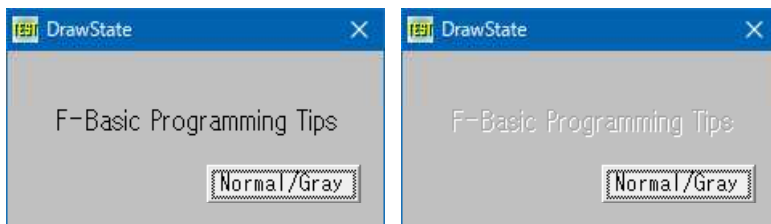
'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 (Cancel As Integer, Mode As Integer)
Sub MainForm_QueryClose (Cancel As Integer, Mode As Integer)
    If Cancel = 0 Then
        Ret = Api_ReleaseDC (GethWnd, hDC)
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列のグレー表示 (II)

DrawState イメージを加工して表示
GetPixel 指定された座標のピクセルのRGB値を取得
SetBkColor デバイスコンテキストの背景色を設定
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```

'=====
'= 文字列のグレー表示 (II)
'= (DrawState3.bas)
'=====
#include "Windows.bi"

' イメージを加工して表示
Declare Function Api_DrawState& Lib "user32" Alias "DrawStateA" (ByVal hDC&, ByVal
hBrush&, ByVal lpOutputFunc&, ByVal lData$, ByVal wData&, ByVal X&, ByVal Y&, ByVal cx&,
ByVal cy&, ByVal fuFlags&)

' システムの背景色を取得
Declare Function Api_GetSysColor& Lib "user32" Alias "GetSysColor" (ByVal nIndex&)

' 指定された座標のピクセルのRGB値を取得
Declare Function Api_GetPixel& Lib "gdi32" Alias "GetPixel" (ByVal hDC&, ByVal X&, ByVal
Y&)

' デバイスコンテキストの背景色を設定
Declare Function Api_SetBkColor& Lib "gdi32" Alias "SetBkColor" (ByVal hDC&, ByVal
crColor&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define TXT_DISABLED &H20
#define TXT_PREFIXTEXT &H2
#define TXT_TEXT &H1

```

```

#define COLOR_BTNFACE 15                                'コマンドボタンの表面色

Var SHared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared flg As Integer
Var Shared rgbColor As Long
Var Shared hDC As Long
Var Shared txt As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var BkCol As Long

    hDC = Api_GetDC(GethWnd)
    txt = "F-Basic Programming Tips"

    rgbColor = RGB(192, 192, 192)

    'フォームの背景色取得
    BkCol = Api_GetPixel(hDC, 0, 0)

    '文字の背景色を設定
    Ret = Api_SetBkColor(hDC, BkCol)

    Cls

    'Mainformを表示
    ShowWindow -1
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var lDrawStateFlag As Long
    Var Ret As Long

    '交互に切り替え
    flg = Not flg

    If flg = 0 Then lDrawStateFlag = TXT_DISABLED

    Cls

    Ret = Api_DrawState(hDC, 0, 0, txt, Len(txt), 30, 34, 0, 0, TXT_TEXT Or lDrawStateFlag)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

    Ret = Api_ReleaseDC(GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop : End

```

文字列を指定の矩形範囲内に出力

文字列を指定した矩形範囲に出力します。TextOutは文字列の描画位置のみの指定でした。

WindowFromPoint 指定の座標位置にあるウィンドウハンドルを取得

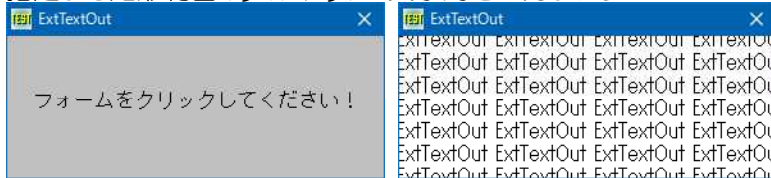
GetCursorPos マウスカーソル(マウスポインタ)の現在の位置に相当するスクリーン座標を取得

ExtTextOut 指定の矩形の範囲内でテキストを表示

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

GetTextExtentPoint32 文字列全体の幅と高さを取得

指定した矩形範囲(ウィンドウ)に文字列を出力します。



```
'=====
'= 文字列を指定の矩形範囲内に出力
'=      (GetTextExtentPoint32.bas)
'=====
#include "Windows.bi"
```

```
Type POINTAPI
    X    As Long
    Y    As Long
End Type
```

```
Type RECT
    Left      As Long
    Top       As Long
    Right     As Long
    Bottom    As Long
End Type
```

' 指定の座標位置にあるウィンドウハンドルを取得

```
Declare Function Api_WindowFromPoint& Lib "user32" Alias "WindowFromPoint" (ByVal
xPoint&, ByVal yPoint&)
```

' マウスカーソル(マウスポインタ)の現在の位置に相当するスクリーン座標を取得

```
Declare Function Api_GetCursorPos& Lib "user32" Alias "GetCursorPos" (lpPoint As
POINTAPI)
```

' 指定の矩形の範囲内でテキストを表示

```
Declare Function Api_ExtTextOut& Lib "gdi32" Alias "ExtTextOutA" (ByVal hDC&, ByVal X&,
ByVal Y&, ByVal wOptions&, ByVal lpRect As Any, ByVal lpString$, ByVal nCount&, lpDx&)
```

' ウィンドウの座標をスクリーン座標系で取得

```
Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,
lpRect As RECT)
```

' 文字列全体の幅と高さを取得

```
Declare Function Api_GetTextExtentPoint32& Lib "gdi32" Alias "GetTextExtentPoint32A"
(ByVal hDC&, ByVal lpstr$, ByVal cbString&, lpSize As POINTAPI)
```

' ウィンドウ全体のデバイスコンテキストを取得

```
Declare Function Api_GetWindowDC& Lib "user32" Alias "GetWindowDC" (ByVal hWnd&)
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_MouseUp edec1 (ByVal Button As Integer, ByVal Shift As Integer, ByVal
SX As Single, ByVal SY As Single)
Sub MainForm_MouseUp (ByVal Button As Integer, ByVal Shift As Integer, ByVal SX As Single,
ByVal SY As Single)
    Var pt As POINTAPI
    Var mWnd As Long
    Var rc As RECT
```

```

Var nDC As Long
Var TextSize As POINTAPI
Var CX As Long
Var CY As Long
Var txt As String
Var Ret As Long

txt = "ExtTextOut "

'カーソル位置取得
Ret = Api_GetCursorPos (pt)

'カーソル下のウィンドウ位置取得
mWnd = Api_WindowFromPoint (pt.X, pt.Y)

'ウィンドウの矩形を取得
Ret = Api_GetWindowRect (mWnd, rc)

'ウィンドウのデバイスコンテキスト取得
nDC = Api_GetWindowDC (mWnd)

'テキストのサイズを取得
Ret = Api_GetTextExtentPoint32 (nDC, txt, Len(txt), TextSize)

For CX = 1 To rc.Right - rc.Left Step TextSize.X
  For CY = 1 To rc.Bottom - rc.Top Step TextSize.Y

    'テキストをウィンドウに描画
    Ret = Api_ExtTextOut (nDC, CX, CY, 0, ByVal 0, txt, Len(txt), ByVal 0)
  Next
Next
End Sub

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
  Symbol (20, 40), "フォームをクリックしてください！", 1, 1
End Sub

'=====
'=
'=====
While 1
  WaitEvent
Wend
Stop
End

```

文字列を指定の位置に出力

スクリーン座標、フォーム内の指定位置に文字列を描画します。

GetActiveWindow アクティブウィンドウのハンドル取得
GetWindowDC ウィンドウ全体のデバイスコンテキストのハンドルを取得
Ellipse 楕円の描画
TextOut 文字列を指定の位置に出力
GetCursorPos マウスカーソルの座標を取得

フォームにタイマーを2個、コマンドボタンを1個貼り付けます。直径5ピクセルの円をカーソル移動位置に描画しています。



左:フォームの内(MainForm)外(スクリーン)に描画している
 右:『テキスト描画』ボタンクリックでスクリーン座標にテキストを描画しています。



```
'=====
'= 文字列を指定の位置に出力
'= (TextOut2.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type

' アクティブウィンドウのハンドル取得
Declare Function Api_GetActiveWindow& Lib "user32" Alias "GetActiveWindow" ()

' ウィンドウ全体のデバイスコンテキストのハンドルを取得
Declare Function Api_GetWindowDC& Lib "user32" Alias "GetWindowDC" (ByVal hWnd&)

' 楕円の描画
Declare Function Api_Ellipse& Lib "gdi32" Alias "Ellipse" (ByVal hDC&, ByVal X1&, ByVal Y1&, ByVal X2&, ByVal Y2&)

' 文字列を指定の位置に出力
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal x&, ByVal y&, ByVal lpString$, ByVal nCount&)

' マウスカースルの座標を取得
Declare Function Api_GetCursorPos& Lib "user32" Alias "GetCursorPos" (lpPoint As POINTAPI)

Var Shared Timer1 As Object
Var Shared Timer2 As Object

Timer1.Attach GetDlgItem("Timer1")
Timer2.Attach GetDlgItem("Timer2")

'=
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.SetInterval 10
    Timer1.Enable -1

    Timer2.SetInterval 10
    Timer2.Enable -1
End Sub
```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var i As Integer
    Var Str As String
    Var Ret As Long

    Str = "スクリーンにテキストを描画！"

    For i = 0 To 30

        'スクリーンにテキストを描画
        Ret = Api_TextOut(Api_GetWindowDC(0), i * 20, i * 20, Str, Len(Str))
    Next i
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var pa As POINTAPI
    Var Ret As Long

    'カーソル位置取得
    Ret = Api_GetCursorPos(pa)

    'スクリーンDCに楕円(円)を描画
    Ret = Api_Ellipse(Api_GetWindowDC(0), pa.x - 5, pa.y - 5, pa.x + 5, pa.y + 5)
End Sub

'=====
'=
'=====
Declare Sub Timer2_Timer edec1 ()
Sub Timer2_Timer()
    Var Ret As Long

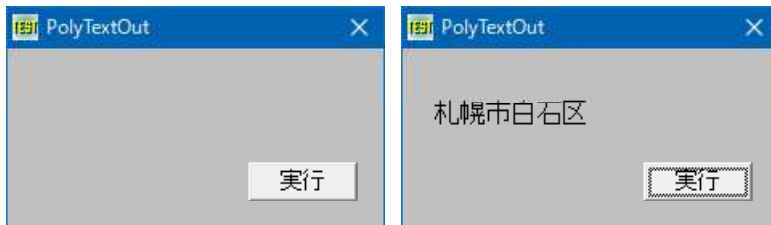
    'テキストをアクティブウインドウに描画
    Ret = Api_TextOut(Api_GetWindowDC(Api_GetActiveWindow), 50, 50, "ここはMainFormです！
", 20)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を指定のデバイスコンテキストに描画

PolyTextOut 指定されたDCで現在選択されているフォントおよびテキストの色を使って、文字列を描画
SetBkMode バックグラウンドの塗りつぶしモード設定
GetDC 指定されたウインドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストハンドル取得
ReleaseDC デバイスコンテキストを解放



```
'=====
'= 文字列を指定のデバイスコンテキストに描画
'= (PolyTextOut.bas)
'=====
#include "Windows.bi"

Type RECT
    Left        As Long
    Top         As Long
    Right       As Long
    Bottom      As Long
End Type

Type POLYTEXT
    x           As Long
    y           As Long
    n           As Long
    lpStr       As Long
    uiFlags     As Long
    rcl         As RECT
    pdx         As Long
End Type

' 指定されたデバイスコンテキストで現在選択されているフォントおよびテキストの色を使って、文字列を描画
Declare Function Api_PolyTextOut& Lib "gdi32" Alias "PolyTextOutA" (ByVal hDC&, pptxt As POLYTEXT, ByVal cStrings&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal iBkMode&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define TRANSPARENT 1                                ' 背景色を設定しない

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var pt As POLYTEXT
    Var hDC As Long
    Var mStr As String
    Var Ret As Long

    ' デバイスコンテキストのハンドルを取得
    hDC = Api_GetDC(GetHwnd)

    mStr = "札幌市白石区"

    ' POLYTEXT構造体を設定
    pt.n = Len(mStr)
    pt.lpStr = StrAdr(mStr & Chr$(0))
```

```

pt.x = 20
pt.y = 30

'sDCに文字列を描写
Ret = Api_SetBkMode(hDC, TRANSPARENT)
Ret = Api_PolyTextOut(hDC, pt, 1)

'hDCの解放
Ret = Api_ReleaseDC(GetHwnd, hDC)
End Sub

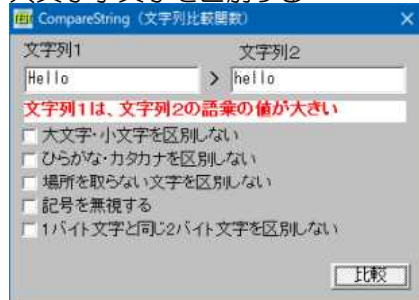
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

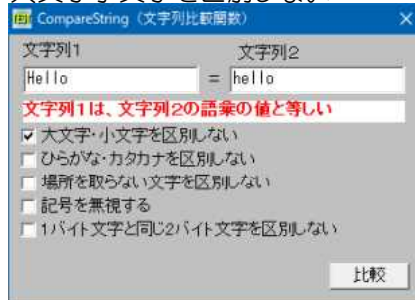
文字列を比較する

文字列1と文字列2の語彙の値を比較します。
[GetThreadLocale](#) スレッドのロケールIDを取得
[CompareString](#) 比較オプションに対応した文字列比較関数

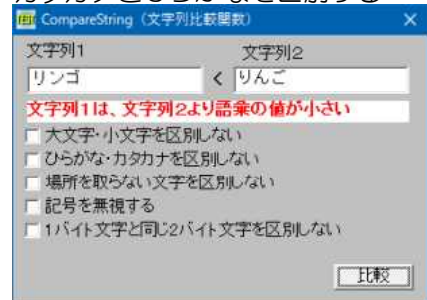
大文字小文字を区別する



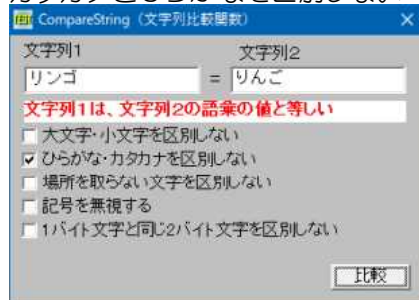
大文字小文字を区別しない



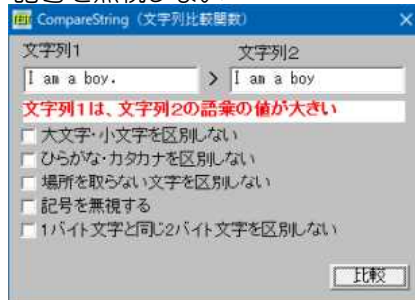
カタカナとひらがなを区別する



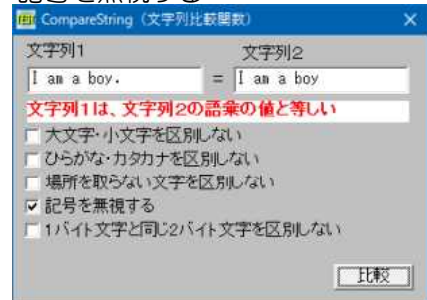
カタカナとひらがなを区別しない



記号を無視しない



記号を無視する



参考

ロケールとは、ソフトウェアに内蔵される、言語や国・地域ごとに異なる単位、記号、日付、通貨などの表記規則の集合。
 そういえば、Microsoft Office 配下などに1041などのフォルダがありましたネ！

アイスランド語	1039
インドネシア語	1057
イタリア語	1040
英語 (オーストラリア)	3081
英語 (カナダ)	4105
英語 (U.K.)	2057
英語 (U.S.)	1033
日本語	1041
韓国語	1042
ラテン語	1142

```

'=====
'= 文字列を比較する
'= (CompareString.bas)
'=====
#include "Windows.bi"

' スレッドのロケールIDを取得
Declare Function Api_GetThreadLocale& Lib "kernel32" Alias "GetThreadLocale" ()

' 比較オプションに対応した文字列比較関数
Declare Function Api_CompareString& Lib "kernel32" Alias "CompareStringA" (ByVal
Locale&, ByVal dwCmpFlags&, ByVal lpString1$, ByVal cchCount1&, ByVal lpString2$, ByVal
cchCount2&)

#define CSTR_EQUAL 2                                '文字列1=文字列2
#define CSTR_GREATER_THAN 3                          '文字列1>文字列2
#define CSTR_LESS_THAN 1                             '文字列1<文字列2

#define NORM_IGNORECASE &H1                          '大文字・小文字を区別しない
#define NORM_IGNOREKANATYPE &H10000                 'ひらがな・カタカナを区別しない
#define NORM_IGNORENONSPACE &H2                     '場所を取らない文字を区別しない
#define NORM_IGNORESYMBOLS &H4                       '記号を無視する
#define NORM_IGNOREWIDTH &H20000                     '1バイト文字とそれと同じ2バイト文字を区別しない

#define SORT_STRINGSORT &H1000
#define LOCALE_SYSTEM_DEFAULT &H400                  'システムのデフォルトロケール
#define LOCALE_USER_DEFAULT &H800                    '現在のユーザのデフォルトロケール

Var Shared Edit1 As Object : Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Var Shared Edit2 As Object : Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Var Shared Text3 As Object : Text3.Attach GetDlgItem("Text3") : Text3.SetFontSize 14
Var Shared Text4 As Object : Text4.Attach GetDlgItem("Text4") : Text4.SetFontSize 14
Var Shared Check(4) As Object

For i = 0 To 4
    Check(i).Attach GetDlgItem("Check" & Trim$(Str$(i + 1)))
    Check(i).SetFontSize 14
Next

Var Shared Flags(4) As Long

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hTL As Long
    Var Str1 As String
    Var Str2 As String

    'スレッドのロケールIDを取得
    hTL = Api_GetThreadLocale()

    Str1 = GetDlgItemText("Edit1")
    Str2 = GetDlgItemText("Edit2")

    If Check(0).GetCheck = 1 Then Flags(0) = NORM_IGNORECASE Else Flags(0) = 0
    If Check(1).GetCheck = 1 Then Flags(1) = NORM_IGNOREKANATYPE Else Flags(1) = 0
    If Check(2).GetCheck = 1 Then Flags(2) = NORM_IGNORENONSPACE Else Flags(2) = 0
    If Check(3).GetCheck = 1 Then Flags(3) = NORM_IGNORESYMBOLS Else Flags(3) = 0
    If Check(4).GetCheck = 1 Then Flags(4) = NORM_IGNOREWIDTH Else Flags(4) = 0

    Select Case Api_CompareString(hTL, Flags(0) + Flags(1) + Flags(2) + Flags(3) +
Flags(4), Str1, Len(Str1), Str2, Len(Str2))
        Case CSTR_LESS_THAN
            Text3.SetWindowText "<"
            Text4.SetWindowText "文字列1は、文字列2より語彙の値が小さい"
        Case CSTR_EQUAL
            Text3.SetWindowText "="
            Text4.SetWindowText "文字列1は、文字列2の語彙の値と等しい"
    End Select
End Sub

```

```

        Case CSTR_GREATER_THAN
            Text3.SetWindowText ">"
            Text4.SetWindowText "文字列1は、文字列2の語彙の値が大きい"
        End Select
    End Sub
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (1)

CreateFont 指定の属性を持つ論理フォントを作成

TextOut 文字を描画

SelectObject 指定されたデバイスコンテキストのオブジェクトを選択

DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放

GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```

'=====
'= 文字列を描画
'= (CreateFont.bas)
'=====
#include "Windows.bi"

```

' 指定の属性を持つ論理フォントを作成

```

Declare Function Api_CreateFont& Lib "gdi32" Alias "CreateFontA" (ByVal Hei&, ByVal Wid&,
ByVal Esc&, ByVal Ori&, ByVal Wei&, ByVal Ita&, ByVal Und&, ByVal Stk&, ByVal Chr&, ByVal
Out&, ByVal Clp&, ByVal Qua&, ByVal Pit&, ByVal Face$)

```

' 文字を描画

```

Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

```

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得

```

Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

```

' デバイスコンテキストを解放

```

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

```

' 指定されたデバイスコンテキストのオブジェクトを選択

```

Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

```

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放

```

Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

```

```

#define DEFAULT_CHARSET 1
#define FW_BOLD 700
#define FW_DONTCARE 0

```

```

'デフォルトの文字セット
'フォントの線幅
'デフォルト

```

```

Var Shared Picture1 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

Picture1.Attach GetDlgItem("Picture1")
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub sFontPreview(fSize As Integer, FontName As String, fDig As Integer, fBold As Integer, Top As Single, Left As Single, MyText As String)
Sub sFontPreview(fSize As Integer, FontName As String, fDig As Integer, fBold As Integer, Top As Single, Left As Single, MyText As String)
    Var hDC As Long
    Var Bold As Long
    Var ObjhDC1 As Long
    Var ObjhDC2 As Long
    Var Ret As Long

    If fBold = True Then
        Bold = FW_BOLD
    Else
        Bold = FW_DONTCARE
    End If

    fDig = 450

    'Picture1 のハンドル取得
    hDC = Api_GetDC(Picture1.GethWnd)

    ObjhDC1 = Api_CreateFont(fSize, 10, fDig, fDig, Bold, 0, 0, 0, DEFAULT_CHARSET, 0, 0, 0, 0, FontName)
    ObjhDC2 = Api_SelectObject(hDC, ObjhDC1)

    '表示位置の設定
    Ret = Api_TextOut(hDC, Top, Left, MyText$, Len(MyText$))

    '元のフォントに戻す
    Ret = Api_SelectObject(hDC, ObjhDC2)

    '取得したハンドルを開放(システムリソースを開放する)
    Ret = Api_DeleteObject(ObjhDC1)
    Ret = Api_DeleteObject(ObjhDC2)
    Ret = Api_ReleaseDC(Picture1.GethWnd, hDC)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var FontName As String

    FontName = Edit1.GetWindowText
    sFontPreview 60, FontName, 0, False, 10, 80, "札幌市白石区"
End Sub

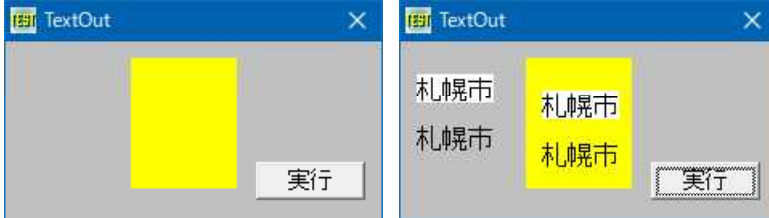
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (II)

SetBkMode バックグラウンドの塗りつぶしモード設定
TextOut 文字を描画
lstrlen 指定された文字列のバイトまたは文字の長さを返す
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放
TRANSPARENT (1) 背景色を設定しない

TextOutのみの場合と、SetBkModeでTRANSPARENTを指定した場合を表しています。



```
'=====
'= 文字列を描画 (II)
'= (TextOut3.bas)
'=====
#include "Windows.bi"

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal iBkMode&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&, ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' 指定された文字列のバイトまたは文字の長さを返す
Declare Function Api_lstrlen& Lib "Kernel32" Alias "lstrlenA" (ByVal lpString$)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define TRANSPARENT 1                                '背景色を設定しない

Var Shared Button1 As Object
Var Shared Picture1 As Object

Picture1.Attach GetDlgItem("Picture1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var hDC As Long
    Var phDC As Long
    Var Str As String
    Var Ret As Long

    Str = "札幌市"

    hDC = Api_GetDC (GethWnd)
    phDC = Api_GetDC (Picture1.GethWnd)

    Ret = Api_TextOut (hDC, 10, 20, Str, Api_lstrlen (Str))
    Ret = Api_TextOut (phDC, 10, 20, Str, Api_lstrlen (Str))

    Ret = Api_SetBkMode (hDC, TRANSPARENT)
```



```

Ret = Api_TextOut(hDC, 10, 50, Str, Api_lstrlen(Str))

Ret = Api_SetBkMode(phDC, TRANSPARENT)
Ret = Api_TextOut(phDC, 10, 50, Str, Api_lstrlen(Str))

Ret = Api_ReleaseDC(GethWnd, hDC)
Ret = Api_ReleaseDC(Picture1.GethWnd, phDC)
End Sub

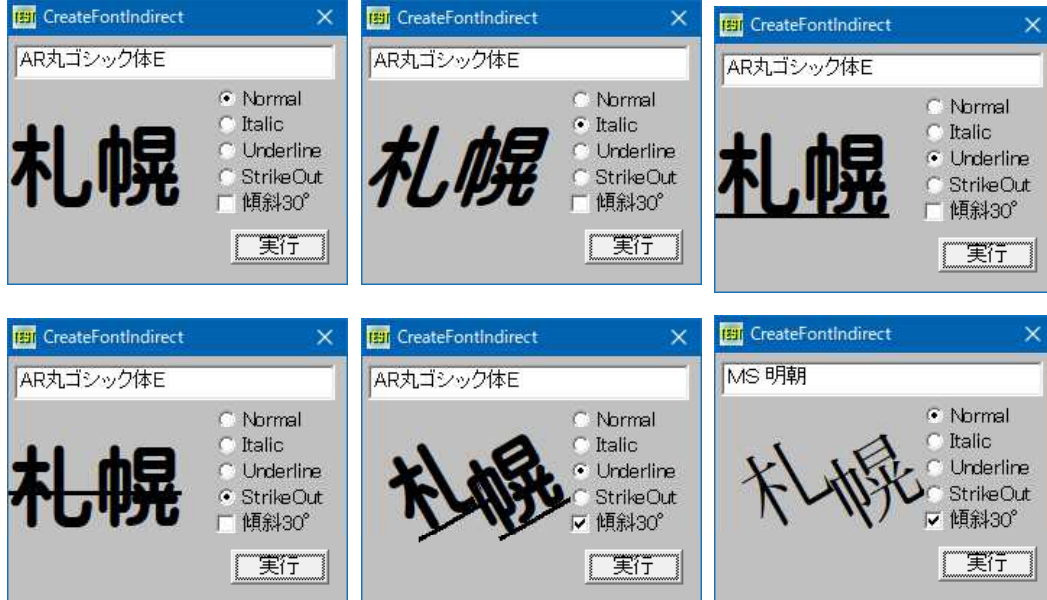
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画(Ⅲ)

CreateFontIndirect 論理フォントを作成
SetGraphicsMode 指定されたデバイスコンテキストのグラフィックスモードを設定
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放
CreateSolidBrush 純色で論理ブラシを作成
SetBkMode バックグラウンドの塗りつぶしモード設定
SetBkColor デバイスコンテキストの背景色を設定
TextOut 文字を描画

例では、EditBoxにフォント名を入れラジオボタンで文字修飾を選択します。



```

'=====
'= 文字列を描画(Ⅲ)
'= (CreateFontIndirect.bas)
'=====
#include "Windows.bi"

#define LF_FACESIZE 32
#define GM_ADVANCED &H2
#define OPAQUE 2

Type LOGFONT
    lfHeight          As Long
    lfWidth           As Long

```

'WinNT・2K・XPグラフィックモード
 '背景色を設定する
 '文字セルまたは文字の高さ
 '平均文字幅

```

    lfEscapement      As Long      '文字送りの方向とx軸との角度
    lfOrientation     As Long      'ベースラインとx軸との角度
    lfWeight          As Long      'フォントの太さ
    lfItalic          As Byte      'イタリック体指定
    lfUnderline       As Byte      '下線付き指定
    lfStrikeOut       As Byte      '打ち消し線付き指定
    lfCharSet         As Byte      'キャラクタセット
    lfOutPrecision     As Byte      '出力精度
    lfClipPrecision   As Byte      'クリッピングの精度
    lfQuality         As Byte      '出力品質
    lfPitchAndFamily  As Byte      'ピッチとファミリ
    lfFaceName        As String * 32 ' (LF_FACESIZE) As Byte'フォント名
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのグラフィックスモードを設定
Declare Function Api_SetGraphicsMode& Lib "gdi32" Alias "SetGraphicsMode" (ByVal hDC&,
ByVal iMode&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

' 純色で論理ブラシを作成
Declare Function Api_CreateSolidBrush& Lib "gdi32" Alias "CreateSolidBrush" (ByVal
crColor&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)

' デバイスコンテキストの背景色を設定
Declare Function Api_SetBkColor& Lib "gdi32" Alias "SetBkColor" (ByVal hDC&, ByVal
crColor&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

Var Shared Button1 As Object
Var Shared Radio(3) As Object
Var Shared Check1 As Object
Var Shared Edit1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Check1.Attach GetDlgItem("Check1") : Check1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
For i = 0 To 3
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1)))
    Radio(i).SetFontSize 14
Next

'=====
'=
'=====

Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var lf As LOGFONT

```

```

Var rFont As Long
Var hDC As Long
Var OldGM As Long
Var rgbColor As Long
Var txt As String
Var Ret As Long

txt = "札幌"

'Formの色
rgbColor = RGB(192, 192, 192)

'30度傾斜をつける
If Check1.GetCheck Then lf.lfOrientation = 300

lf.lfHeight = 60
lf.lfItalic = Radio(1).GetCheck
lf.lfUnderline = Radio(2).GetCheck
lf.lfStrikeOut = Radio(3).GetCheck

'Shift JIS に設定 (設定しない場合全角が文字化けする)
lf.lfCharSet = 128

'フォント名の最後尾にChr$(0) は必須
lf.lfFaceName = Edit1.GetWindowText & Chr$(0)

'Edit1の背景色のモードを設定
Ret = Api_SetBkMode(hDC, OPAQUE)

'DC取得
hDC = Api_GetDC(GethWnd)

'描画色を設定
colBrush = Api_CreateSolidBrush(rgbColor)

'グラフィックモードを設定
OldGM = Api_SetGraphicsMode(hDC, GM_ADVANCED)

'背景色を設定
Ret = Api_SetBkColor(hDC, rgbColor)

Cls

'論理フォントを作成
rFont = Api_CreateFontIndirect(lf)

'オブジェクトを選択
Ret = Api_SelectObject(hDC, rFont)

'文字を描画
Ret = Api_TextOut(hDC, 0, 60, txt, Len(txt))

'グラフィックモードを元に戻す
Ret = Api_SetGraphicsMode(hDC, OldGM)

Ret = Api_ReleaseDC(GethWnd, hDC)
Ret = Api_DeleteObject(rFont)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (IV)

CreateFontIndirect 論理フォントを作成
CreateSolidBrush ソリッドカラーで論理ブラシを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
BeginPath hDCで指定されたデバイスコンテキストのパスの作成
EndPath BeginPathで開始したパスの作成を終了
TextOut 文字を描画
StrokePath 選択されているパスを選択されているペンを使用して描画
FillPath 選択されているブラシによってパスの内部を塗りつぶす
DeleteObject 関連付けられていたすべてのシステムリソースを解放
StrokeAndFillPath パスを描画し、その内部を選択されているブラシによって塗りつぶす
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放
SetBkMode バックグラウンドの塗りつぶしモード設定



```
'=====
'= 文字列を描画
'= (EndPath.bas)
'=====
#include "windows.bi"

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight           As Long
    lfWidth            As Long
    lfEscapement       As Long
    lfOrientation      As Long
    lfWeight           As Long
    lfItalic           As Byte
    lfUnderline        As Byte
    lfStrikeOut        As Byte
    lfCharSet          As Byte
    lfOutPrecision     As Byte
    lfClipPrecision    As Byte
    lfQuality          As Byte
    lfPitchAndFamily   As Byte
    lfFaceName         As String * LF_FACESIZE
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
    (lpLogFont As LOGFONT) ' ソリッドカラーで論理ブラシを作成
Declare Function Api_CreateSolidBrush& Lib "gdi32" Alias "CreateSolidBrush" (ByVal
    crColor&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
    hObject&)

' hDCで指定されたデバイスコンテキストのパスの作成
Declare Function Api_BeginPath& Lib "gdi32" Alias "BeginPath" (ByVal hDC&)

' BeginPathで開始したパスの作成を終了
Declare Function Api_EndPath& Lib "gdi32" Alias "EndPath" (ByVal hDC&)
```

' 文字を描画

```
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)
```

' 選択されているパスを選択されているペンを使用して描画する

```
Declare Function Api_StrokePath& Lib "gdi32" Alias "StrokePath" (ByVal hDC&)
```

' 選択されているブラシによってパスの内部を塗りつぶす

```
Declare Function Api_FillPath& Lib "gdi32" Alias "FillPath" (ByVal hDC&)
```

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる

```
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

' パスを描画し、その内部を選択されているブラシによって塗りつぶす

```
Declare Function Api_StrokeAndFillPath& Lib "gdi32" Alias "StrokeAndFillPath" (ByVal
hDC&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

' バックグラウンドの塗りつぶしモード設定

```
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)
```

```
#define TRANSPARENT 1
```

' 背景色を設定しない

```
Var Shared Button1 As Object
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
Var hDC As Long
```

```
Var sText As String
```

```
Var lf As LOGFONT
```

```
Var rFont As Long
```

```
Var Size As Integer
```

```
Var Ret As Long
```

```
sText = "札幌市"
```

```
hDC = Api_GetDC(GethWnd)
```

```
Size = 48
```

```
lf.lfHeight = Size
```

```
lf.lfCharSet = 128
```

```
lf.lfFaceName = "DF太楷書体" & Chr$(0)
```

```
lf.lfWeight = 700
```

```
rFont = Api_CreateFontIndirect(lf)
```

```
Ret = Api_SelectObject(hDC, rFont)
```

```
Ret = Api_SetBkMode(hDC, TRANSPARENT)
```

```
Ret = Api_BeginPath(hDC)
```

```
Ret = Api_TextOut(hDC, 10, 4, sText, Len(sText))
```

```
Ret = Api_EndPath(hDC)
```

```
Ret = Api_StrokePath(hDC)
```

```
Ret = Api_BeginPath(hDC)
```

```
Ret = Api_TextOut(hDC, 10, 52, sText, Len(sText))
```

```
Ret = Api_EndPath(hDC)
```

```
Ret = Api_FillPath(hDC)
```

```
Ret = Api_BeginPath(hDC)
```

```
Ret = Api_TextOut(hDC, 10, 100, sText, Len(sText))
```

```

Ret = Api_EndPath (hDC)

Ret = Api_StrokeAndFillPath (hDC)
Ret = Api_SelectObject (hDC, oldBrush)
Ret = Api_DeleteObject (hBrush)

Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (V)

CreateFont 指定の属性を持つ論理フォントを作成
TextOut 文字を描画
SetBkMode バックグラウンドの塗りつぶしモード設定
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
DeleteObject 論理オブジェクトを削除し、そのオブジェクトに関連付けられていた全てのシステムリソースを解放
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```

'=====
'= 文字列の回転
'= (CreateFont3.bas)
'=====
#include "Windows.bi"

' 指定の属性を持つ論理フォントを作成
Declare Function Api_CreateFont& Lib "gdi32" Alias "CreateFontA" (ByVal Hei&, ByVal Wid&,
ByVal Esc&, ByVal Ori&, ByVal Wei&, ByVal Ita&, ByVal Und&, ByVal Stk&, ByVal Chr&, ByVal
Out&, ByVal Clp&, ByVal Qua&, ByVal Pit&, ByVal Face$)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

```

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得

Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define CLIP_LH_ANGLES 16

' Needed for tilted fonts.

#define PI 3.14159625

#define PI_180 (PI / 180#)

#define TRANSPARENT 1

' 背景色を設定しない

#define DEFAULT_CHARSET 1

' デフォルトの文字セット

#define FW_NORMAL 800

Var Shared Timer1 As Object

Timer1.Attach GetDlgItem("Timer1")

Var Shared hDC As Long

'=====

'=

'=====

Declare Sub DrawRotatedText(txt\$, ByVal X!, ByVal Y!, font_name\$, ByVal size&, ByVal weight&, ByVal escapement&, ByVal use_italic%, ByVal use_underline%, ByVal use_strikethrough%)

Sub DrawRotatedText(txt\$, ByVal X!, ByVal Y!, font_name\$, ByVal size&, ByVal weight&, ByVal escapement&, ByVal use_italic%, ByVal use_underline%, ByVal use_strikethrough%)

Var newfont As Long

Var oldfont As Long

Var Ret As Long

newfont = Api_CreateFont(size&, 0, escapement&, escapement&, weight&, use_italic%, use_underline%, use_strikethrough%, DEFAULT_CHARSET, 0, CLIP_LH_ANGLES, 0, 0, font_name\$)

' 新規フォント選択

oldfont = Api_SelectObject(hDC, newfont)

Ret = Api_SetBkMode(hDC, TRANSPARENT)

Ret = Api_TextOut(hDC, X, Y, txt\$, Len(txt\$))

' 元のフォントに戻す

newfont = Api_SelectObject(hDC, oldfont)

' リソース解放

Ret = Api_DeleteObject(newfont)

End Sub

'=====

'=

'=====

Declare Sub MainForm_Start edec1 ()

Sub MainForm_Start()

hDC = Api_GetDC(GetHwnd)

Timer1.SetInterval 5

Timer1.Enable -1

End Sub

'=====

'=

'=====

Declare Sub Timer1_Timer edec1 ()

Sub Timer1_Timer()

Static angle As Long

' 角度

Cls

DrawRotatedText "札幌", 115, 100, "AR丸ゴシック体E", 50, FW_NORMAL, angle * 10, False, False, False

```

    angle = angle + 10
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose cdecl ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (VI)

CreateFontIndirect 論理フォントを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
BeginPath hDCで指定されたデバイスコンテキストのパスの作成
EndPath BeginPathで開始したパスの作成を終了
TextOut 文字を描画
BitBlt ビットブロック転送
SelectClipPath 指定したデバイスコンテキストのクリッピング領域とパスを結合
SetBkMode バックグラウンドの塗りつぶしモード設定
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストのハンドルを解放



```

'=====
'= 文字列を描画 (VI)
'= (SelectClipPath4.bas)
'=====
#include "Windows.bi"

#define LF_FACESIZE 32

Type LOGFONT
    lfHeight      As Long
    lfWidth       As Long
    lfEscapement  As Long
    lfOrientation As Long
    lfWeight      As Long
    lfItalic      As Byte
    lfUnderline   As Byte
    lfStrikeOut   As Byte
    lfCharSet     As Byte
    lfOutPrecision As Byte
    lfClipPrecision As Byte
    lfQuality     As Byte

```



```

    lfPitchAndFamily As Byte
    lfFaceName       As String * LF_FACESIZE
End Type

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' hDCで指定されたデバイスコンテキストのパスの作成
Declare Function Api_BeginPath& Lib "gdi32" Alias "BeginPath" (ByVal hDC&)

' BeginPathで開始したパスの作成を終了
Declare Function Api_EndPath& Lib "gdi32" Alias "EndPath" (ByVal hDC&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' ビットブロック転送を行う。コピー元からコピー先のデバイスコンテキストへ、指定された長方形内の各ピクセルの色
データをコピー
Declare Function Api_BitBlt& Lib "gdi32" Alias "BitBlt" (ByVal hDestDC&, ByVal X&, ByVal
Y&, ByVal nWidth&, ByVal nHeight&, ByVal hSrcDC&, ByVal xSrc&, ByVal ySrc&, ByVal dwRop&)

' 指定したデバイスコンテキストのクリッピング領域とパスを結合
Declare Function Api_SelectClipPath& Lib "gdi32" Alias "SelectClipPath" (ByVal hDC&,
ByVal iMode&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define SRCCOPY &HCC0020
#define RGN_COPY 5
#define TRANSPARENT 1

'そのまま転送
'HRGN_SRC1のコピーを作成
'背景色を設定しない

Var Shared Bitmap As Object
Var Shared Picture1 As Object
Var Shared Picture2 As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

BitmapObject Bitmap
Picture1.Attach GetDlgItem("Picture1")
Picture2.Attach GetDlgItem("Picture2")
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared hDC1 As Long
Var Shared hDC2 As Long

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var Ret As Long

    hDC1 = Api_GetDC(Picture1.GethWnd)
    hDC2 = Api_GetDC(Picture2.GethWnd)

```

```

        Bitmap.LoadFile "Pic.bmp"
        Picture2.DrawBitmap Bitmap, 0, 0
        Bitmap.DeleteObject
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var lf As LOGFONT
    Var rFont As Long
    Var txt As String
    Var Ret As Long

    txt = "札幌市白石区"

    lf.lfHeight = 60
    lf.lfCharSet = 128
    lf.lfFaceName = Edit1.GetWindowText & Chr$(0)
    lf.lfWeight = 700

    Picture1.Cls

    rFont = Api_CreateFontIndirect(lf)
    Ret = Api_SelectObject(hdc1, rFont)
    Ret = Api_SetBkMode(hdc1, TRANSPARENT)
    Ret = Api_BeginPath(hdc1)
    Ret = Api_TextOut(hdc1, 5, 5, txt, Len(txt))
    Ret = Api_EndPath(hdc1)
    Ret = Api_SelectClipPath(hdc1, RGN_COPY)
    Ret = Api_BitBlt(hdc1, 5, 5, Picture2.GetWidth, Picture2.GetHeight, hdc2, 0, 0,
SRCCOPY)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

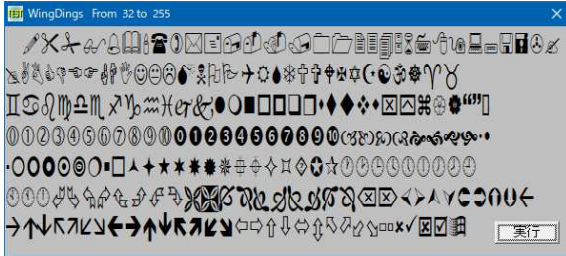
    'デバイスコンテキストを解放
    Ret = Api_ReleaseDC(Picture1.GethWnd, hdc1)
    Ret = Api_ReleaseDC(Picture2.GethWnd, hdc2)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (VII)

TextOut 文字を描画
DeleteObject システムリソースを解放
CreateFontIndirect 論理フォントを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
SetBkMode バックグラウンドの塗りつぶしモード設定
GetDC 指定されたウィンドウのデバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



```
'=====
'= 文字列を描画 (VII)
'= (TextOut4.bas)
'=====
#include "Windows.bi"

Type LOGFONT
    lfHeight          As Long
    lfWidth           As Long
    lfEscapement       As Long
    lfOrientation      As Long
    lfWeight           As Long
    lfItalic           As Byte
    lfUnderline        As Byte
    lfStrikeOut        As Byte
    lfCharSet          As Byte
    lfOutPrecision     As Byte
    lfClipPrecision    As Byte
    lfQuality          As Byte
    lfPitchAndFamily   As Byte
    lfFaceName         As String * 32
End Type

#define SYMBOL_CHARSET 2
#define COLOR_BTNFACE 15
#define TRANSPARENT 1

' シンボル文字セット
' 3Dオブジェクトの表面色
' 背景色を設定しない

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, lpString As Any, ByVal cbString&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 論理フォントを作成
Declare Function Api_CreateFontIndirect& Lib "gdi32" Alias "CreateFontIndirectA"
(lpLogFont As LOGFONT)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
```

```

Sub Button1_on()
    Var hDC As Long
    Var lf As LOGFONT
    Var chars(257) As Byte
    Var DispChar As Integer
    Var NumDispchar As Integer
    Var hFont As Long
    Var StartASCII As Integer
    Var StopASCII As Integer
    Var StartRow As Integer
    Var NumOfChars As Integer
    Var i As Integer
    Var j As Integer
    Var EndRow As Integer
    Var NewStart As Integer
    Var tmp As Integer
    Var xd As Long
    Var xt As Long
    Var xs As Long
    Var fCaption As String
    Var Ret As Long

    lf.lfCharSet = SYMBOL_CHARSET
    lf.lfFaceName = "WingDings" & Chr$(0)
    lf.lfClipPrecision = 64
    lf.lfOutPrecision = 0
    lf.lfEscapement = 0
    lf.lfItalic = 0
    lf.lfWidth = 16
    lf.lfHeight = 32
    lf.lfOrientation = 0

    hDC = Api_GetDC(GethWnd)

    fCaption = Left$(lf.lfFaceName, InStr(lf.lfFaceName, Chr$(0)) - 1)

    hFont = Api_CreateFontIndirect(lf)
    xs = Api_SelectObject(hDC, hFont)

    DispChar = 32
    StartASCII = 32
    StopASCII = 255
    StartRow = 1

    '一行に表示するキャラクタ数
    '最初のキャラクタのアスキー値
    '最後のキャラクタのアスキー値

    If StartASCII < 0 Then StartASCII = 32
    If StopASCII > 255 Then StopASCII = 255

    If StartASCII > StopASCII Then
        tmp = StartASCII
        StartASCII = StopASCII
        StopASCII = tmp
    End If

    NumOfChars = StopASCII - StartASCII + 1

    If Int(NumOfChars / DispChar) = (NumOfChars) / DispChar Then
        EndRow = Int(NumOfChars / DispChar)
    Else
        EndRow = Int(NumOfChars / DispChar) + 1
    End If

    For i = StartASCII To StopASCII
        chars(i) = i
    Next i

    Ret = Api_SetBkMode(hDC, TRANSPARENT)
    Cls

    For j = StartRow To EndRow
        NewStart = StartASCII + (j - 1) * DispChar

```

```

    If NumOfChars < DispChar And NumOfChars > 0 Then
        NumDispchar = NumOfChars
    Else
        NumDispchar = DispChar
    End If

    xt = Api_TextOut(hDC, 0, (j * lf.lfHeight + 2) - 32, chars(NewStart), NumDispchar)
    NumOfChars = NumOfChars - NumDispchar
Next j

xd = Api_DeleteObject(hFont)

fCaption = fCaption & "   From " & Str$(StartASCII) & " to " & Str$(StopASCII)
SetWindowText fCaption

Ret = Api_ReleaseDC(GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字列を描画 (VIII)

CreateFont 指定の属性を持つ論理フォントを作成
CreateSolidBrush ソリッドカラーで論理ブラシを作成
SelectObject 指定されたデバイスコンテキストのオブジェクトを選択
BeginPath hDCで指定されたデバイスコンテキストのパスの作成
SetBkMode バックグラウンドの塗りつぶしモード設定
TextOut 文字を描画
EndPath BeginPathで開始したパスの作成を終了
StrokePath 選択されているパスを選択されているペンを使用して描画
FillPath 選択されているブラシによってパスの内部を塗りつぶす
DeleteObject 関連付けられていたすべてのシステムリソースを解放
StrokeAndFillPath パスを描画し、その内部を選択されているブラシによって塗りつぶす
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



LOGFONT構造体を使用せず、CreateFontでフォント名を指定しています。

```

'=====
'= 文字列を描画 (VIII)
'= (StrokePath2.bas)
'=====
#include "Windows.bi"

' 指定の属性を持つ論理フォントを作成
Declare Function Api_CreateFont& Lib "gdi32" Alias "CreateFontA" (ByVal Hei&, ByVal Wid&,
ByVal Esc&, ByVal Ori&, ByVal Wei&, ByVal Ita&, ByVal Und&, ByVal Stk&, ByVal Chr&, ByVal
Out&, ByVal Clp&, ByVal Qua&, ByVal Pit&, ByVal Face$)

```

```

' ソリッドカラーで論理ブラシを作成
Declare Function Api_CreateSolidBrush& Lib "gdi32" Alias "CreateSolidBrush" (ByVal
crColor&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' hDCで指定されたデバイスコンテキストのパスの作成
Declare Function Api_BeginPath& Lib "gdi32" Alias "BeginPath" (ByVal hDC&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' BeginPathで開始したパスの作成を終了
Declare Function Api_EndPath& Lib "gdi32" Alias "EndPath" (ByVal hDC&)

' 選択されているパスを選択されているペンを使用して描画する
Declare Function Api_StrokePath& Lib "gdi32" Alias "StrokePath" (ByVal hDC&)

' 選択されているブラシによってパスの内部を塗りつぶす
Declare Function Api_FillPath& Lib "gdi32" Alias "FillPath" (ByVal hDC&)

' 論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' パスを描画し、その内部を選択されているブラシによって塗りつぶす
Declare Function Api_StrokeAndFillPath& Lib "gdi32" Alias "StrokeAndFillPath" (ByVal
hDC&)

' 指定されたウィンドウのデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define TRANSPARENT 1
#define FW_BOLD 700
#define vbWhite &HFFFFFF

' 背景色を設定しない
' LOGFONT(lfWeight)
' 白のカラーコード

Var Shared Button1 As Object

Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var txt As String
    Var hDC As Long
    Var Size As Long
    Var rFont As Long
    Var hBrush As Long
    Var oldBrush As Long
    Var Ret As Long

    txt = "札幌市"
    Size = 50

    hDC = Api_GetDC(GethWnd)

    rFont = Api_CreateFont(Size, 0, 0, 0, 700, 0, 0, 0, 1, 0, 0, 0, 0, "DCGクリスタルW5")
    Ret = Api_SelectObject(hDC, rFont)

```

```

hBrush = Api_CreateSolidBrush( RGB(255, 255, 0) )      'ブラシを作成
oldBrush = Api_SelectObject( hDC, hBrush )             'オブジェクトを選択
Ret = Api_SetBkMode( hDC, TRANSPARENT )               '背景の塗り潰しモードを設定

Ret = Api_BeginPath( hDC )                             'パスブラケットオープン
Ret = Api_TextOut( hDC, 10, 0, txt, Len(txt) )         'テキスト描画
Ret = Api_EndPath( hDC )                               'パスブラケットクローズ
Ret = Api_StrokePath( hDC )                           '輪郭を描画

Ret = Api_BeginPath( hDC )                             '次のパスブラケットオープン
Ret = Api_TextOut( hDC, 10, Size, txt, Len(txt) )      'テキスト描画
Ret = Api_EndPath( hDC )                               'パスブラケットクローズ
Ret = Api_FillPath( hDC )                             '指定のブラシで塗りつぶす

hBrush = Api_CreateSolidBrush( RGB(255, 0, 255) )     'ブラシを作成
oldBrush = Api_SelectObject( hDC, hBrush )            'オブジェクトを選択

Ret = Api_BeginPath( hDC )                             '次のパスブラケットオープン
Ret = Api_TextOut( hDC, 10, Size * 2, txt, Len(txt) ) 'テキスト描画
Ret = Api_EndPath( hDC )                               'パスブラケットクローズ
Ret = Api_StrokeAndFillPath( hDC )                    '指定のブラシで塗りつぶすとともに、輪郭を描画
Ret = Api_SelectObject( hDC, oldBrush )              'ブラシをオリジナルのものに選択
Ret = Api_DeleteObject( hBrush )                     'ブラシを削除
Ret = Api_ReleaseDC( GethWnd, hDC )                  'デバイスコンテキストを解放
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

文字を右揃え・左揃えで描画

文字を左揃え・右揃えで描画します。
SetTextAlign 文字を描画する時の書式を設定
TextOut 文字を描画



```

' =====
' = TextOut・SetTextAlign
' = (TextOut.bas)
' =====
#include "Windows.bi"

' 文字を描画する時の書式を設定(右揃え・左揃え)
Declare Function Api_SetTextAlign& Lib "gdi32" Alias "SetTextAlign" (ByVal hDC&, ByVal wFlags&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&, ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

```

・ デバイスコンテキストを解放

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

```
#define TA_BASELINE 24
#define TA_BOTTOM 8
#define TA_CENTER 6
#define TA_LEFT 0
#define TA_MASK (TA_BASELINE Or TA_CENTRE Or TA_UPDATECP)
#define TA_NOUPDATECP 0

#define TA_RIGHT 2
#define TA_TOP 0
#define TA_UPDATECP 1
```

'参照点と、テキストのベースラインを揃える
'参照点と、境界長方形の下端を揃える
'参照点と、境界長方形の水平方向の中心を揃える
'参照点と、境界長方形の左端を揃える (デフォルト)
'各テキスト出力関数が呼び出されても、カレントポジションを更新しない (デフォルト)
'参照点と、境界長方形の右端を揃える
'参照点と、境界長方形の上端を揃える (デフォルト)
'各テキスト出力関数が呼び出されると、カレントポジションを更新

```
#define lStr "AlignLeft"
#define cStr "AlignCenter"
#define rStr "AlignRight"
```

```
'=====
'=
'=====
```

Declare Sub Mainform_Start edec1 ()

Sub Mainform_Start()

Var Ret As Long

Var hDC As Long

hDC = Api_GetDC (GethWnd)

Line (120, 0) - (120, 140), , 14

・ 左揃え

Ret = Api_SetTextAlign (hDC, TA_LEFT Or TA_TOP Or TA_NOUPDATECP)

Ret = Api_TextOut (hDC, 120, 10, lStr, Len (lStr))

SetDrawWidth 3

Pset (120, 10), 5

SetDrawWidth 0

・ 中央揃え

Ret = Api_SetTextAlign (hDC, TA_CENTER Or TA_TOP Or TA_NOUPDATECP)

Ret = Api_TextOut (hDC, 120, 35, cStr, Len (cStr))

SetDrawWidth 3

Pset (120, 35), 5

SetDrawWidth 0

・ 右揃え

Ret = Api_SetTextAlign (hDC, TA_RIGHT Or TA_TOP Or TA_NOUPDATECP)

Ret = Api_TextOut (hDC, 120, 60, rStr, Len (rStr))

SetDrawWidth 3

Pset (120, 60), 5

SetDrawWidth 0

End Sub

```
'=====
'=
'=====
```

While 1

WaitEvent

Wend

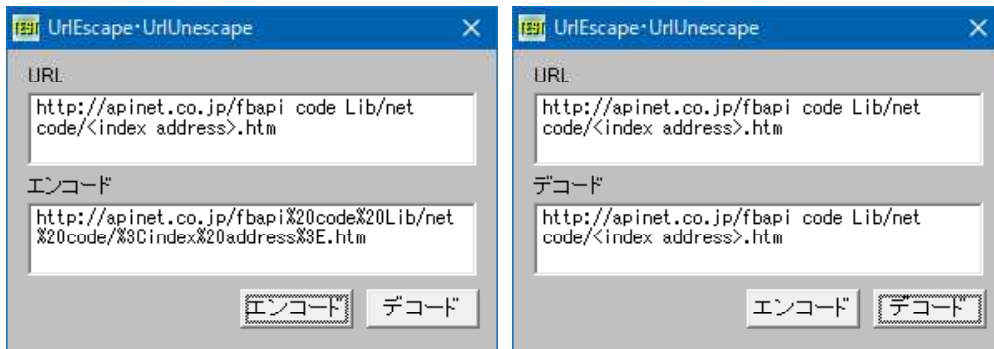
Stop

End

文字をエスケープシーケンスに変換 (I)

UrlEscape %ASCIIに変換可能な安全ではない文字をエスケープシーケンスに変換

UrlUnescape エスケープシーケンスを普通の文字に変換



キャラクタとエスケープシーケンス

^	%5E
&	%26
`	%60
{	%7B
}	%7D
	%7C
]	%5D
[	%5B
"	%22
<	%3C
>	%3E
¥	%5C

```
'=====
'= %ASCIIに変換可能な安全ではない文字をエスケープシーケンスに変換
'= (UrlEscape.bas)
'=====
#include "Windows.bi"

#define MAX_PATH 260
#define ERROR_SUCCESS 0

#define URL_ESCAPE_SEGMENT_ONLY &H2000
#define URL_ESCAPE_PERCENT &H1000
#define URL_UNESCAPE_INPLACE &H100000

#define URL_INTERNAL_PATH &H800000
#define URL_DONT_ESCAPE_EXTRA_INFO &H2000000
#define URL_ESCAPE_SPACES_ONLY &H4000000
#define URL_DONT_SIMPLIFY &H8000000

' %ASCIIに変換可能な安全ではない文字をエスケープシーケンスに変換
Declare Function Api_UrlEscape& Lib "shlwapi" Alias "UrlEscapeA" (ByVal pszURL$, ByVal
pszEscaped$, pcchEscaped&, ByVal dwFlags&)

' エスケープシーケンスを普通の文字に変換
Declare Function Api_UrlUnescape& Lib "shlwapi" Alias "UrlUnescapeA" (ByVal pszURL$,
ByVal pszUnescaped$, pcchUnescaped&, ByVal dwFlags&)

Var Shared Text(1) As Object
Var Shared Edit(1) As Object
Var Shared Button(1) As Object

For i = 0 To 1
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 12
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 12
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1))) : Button(i).SetFontSize 14
Next

'=====
'=
'=====
Declare Function EncodeUrl(sUrl As String) As String
Function EncodeUrl(sUrl As String) As String
    Var buff As String
    Var dwSize As Long
```

```

    Var dwFlags As Long
    If Len(sUrl) > 0 Then
        buff = Space$(MAX_PATH)
        dwSize = Len(buff)
        dwFlags = URL_DONT_SIMPLIFY
        If Api_UrlEscape(sUrl, buff, dwSize, dwFlags) = ERROR_SUCCESS Then
            EncodeUrl = Left$(buff, dwSize)
        End If
    End If
End Function

'=====
'=
'=====
Declare Function DecodeUrl(sUrl As String) As String
Function DecodeUrl(sUrl As String) As String
    Var buff As String
    Var dwSize As Long
    Var dwFlags As Long

    If Len(sUrl) > 0 Then
        buff = Space$(MAX_PATH)
        dwSize = Len(buff)
        dwFlags = URL_DONT_SIMPLIFY
        If Api_UrlUnescape(sUrl, buff, dwSize, dwFlags) = ERROR_SUCCESS Then
            DecodeUrl = Left$(buff, dwSize)
        End If
    End If
End Function

'UrlUnescape
'Len(sUrl)

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit(0).SetWindowText "http://apinet.co.jp/fbapi code Lib/net code/<index
address>.htm"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sUrl As String
    Var buff As String

    sUrl = Edit(0).GetWindowText()
    buff = EncodeUrl(sUrl)

    Text(1).SetWindowText "エソコード"
    Edit(1).SetWindowText buff
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var sUrl As String
    Var buff As String

    sUrl = Edit(1).GetWindowText()
    buff = DecodeUrl(sUrl)

    Text(1).SetWindowText "デコード"
    Edit(1).SetWindowText buff
End Sub

```

```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

文字をエスケープシーケンスに変換(II)

URL文字列をエスケープシーケンスに変換(エンコード)・戻す(デコード)



```

'=====
'= 文字をエスケープシーケンスに変換(II)
'= (URLEncodeDecode.bas)
'=====
#include "Windows.bi"

Var Shared Text(2) As Object
Var Shared Edit1 As Object

For i = 0 To 2
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 12
Next
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 12

'=====
'=
'=====

Declare Function ChangeToHex(Number As Long, Numlen As Integer) As String
Function ChangeToHex(Number As Long, Numlen As Integer) As String
    Var cHex As String

    cHex = hex$(Number)

    If Len(cHex) < Numlen Then
        cHex = String$(Numlen - Len(cHex), "0") & cHex
    End If

```

```

        If Not UpperCase Then
            ChangeToHex = lcase$(cHex)
        Else
            ChangeToHex = cHex
        End If
    End Function

End Function

'=====
'= URLエンコード
'=====
Declare Function URLEncode(sInput As String) As String
Function URLEncode(sInput As String) As String
    Var sRet As String
    Var lChr As Long

    Do While Len(sInput) > 0
        lChr = Asc(sInput)
        sInput = Right$(sInput, Len(sInput) - 1)
        If (lChr >= Asc("a") And lChr <= Asc("z")) Or (lChr >= Asc("A") And lChr <=
asc("Z")) Or (lChr >= Asc("0") And lChr <= Asc("9")) Then
            sRet = sRet & Chr$(lChr)
        Else If lChr = Asc(" ") Then
            sRet = sRet & "+"
        Else
            sRet = sRet & "%" & ChangeToHex(lChr, 2)
        End If
    Loop

    URlencode = sRet
End Function

'=====
'= URLデコード
'=====
Declare Function URLDecode(sInput As String) As String
Function URLDecode(sInput As String) As String
    Var sRet As String
    Var lChr As Long
    Var sChr As String
    Var i As Long

    For i = 1 To Len(sInput)
        sChr = Mid$(sInput, i, 1)
        If sChr = "+" Then
            sRet = sRet & " "
        Else If sChr = "%" Then
            sRet = sRet & Chr$(Val("&H" & Mid$(sInput, i + 1, 1) & Mid$(sInput, i + 2, 1)))
            i = i + 2
        Else
            sRet = sRet & sChr
        End If
    Next

    URLDecode = sRet
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Text(1).SetWindowText "エンコード"
    Text(2).SetWindowText URLEncode(Edit1.GetWindowText)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()

```

```

Sub Button2_on()
    Text(1).SetWindowText "デコード"
    Text(2).SetWindowText URLDecode(Edit1.GetWindowText)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モニターのON/OFF

SendMessage ウィンドウにメッセージを送信

MONITOR_OFF (2)

MONITOR_ON (-1)

SC_MONITORPOWER (&HF170) ディスプレイの電源管理状態を設定

WM_SYSCOMMAND (&H112) システムメニューが操作された



```

'=====
'= モニターのON/OFF
'= (MONITOR_OFF.bas)
'=====
#include "Windows.bi"

#define MONITOR_OFF 2
#define MONITOR_ON -1
#define MONITOR_STANDBY 1
#define SC_MONITORPOWER &HF170
#define WM_SYSCOMMAND &H112

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

Var Shared Timer1 As Object
Var Shared Button1 As Object

Timer1.Attach GetDlgItem("Timer1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_SendMessage(GethWnd, WM_SYSCOMMAND, SC_MONITORPOWER, ByVal MONITOR_OFF)

    Timer1.SetInterval 500
    Timer1.Enable True
End Sub

```

```

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var Ret As Long

    Timer1.Enable False

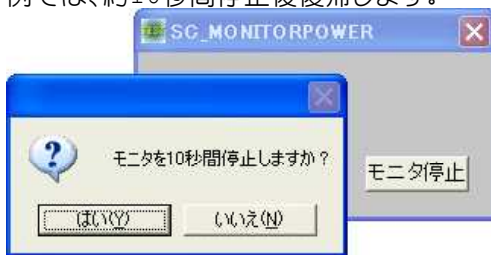
    Ret = Api_SendMessage (GethWnd, WM_SYSCOMMAND, SC_MONITORPOWER, ByVal MONITOR_ON)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

モニタを指定時間停止

モニタの電源を指定時間停止させます。
SendMessage ウィンドウにメッセージを送信
SC_MONITORPOWER (&HF170) ディスプレイの電源管理状態を設定
 例では、約10秒間停止後復帰します。



```

'=====
'= モニタを指定時間停止
'= (SC_MONITORPOWER.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信
Declare Function Api_SendMessageA Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

#define SC_MONITORPOWER &HF170
#define WM_SYSCOMMAND &H112

Var Shared Timer1 As Object
Timer1.Attach GetDlgItem("Timer1")

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Timer1.Enable 0
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()

```

'ディスプレイの電源管理状態を設定
 'システムメニューが操作された

```

Var Ret As Long

A% = MsgBox("", "モニタを10秒間停止しますか?", 4, 1)
If A% = 5 Then
    Timer1.SetInterval 1000
    Timer1.Enable -1
    Ret = Api_SendMessage(GetHwnd, WM_SYSCOMMAND, SC_MONITORPOWER, ByVal 2)
End If
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var Ret As Long

    Timer1.Enable 0
    Ret = Api_SendMessage(GetHwnd, WM_SYSCOMMAND, SC_MONITORPOWER, ByVal -1)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

有効なドライブ名の取得

GetLogicalDriveStrings 有効なドライブ名の取得
 GetDriveType ドライブタイプを取得

ComboBoxでドライブを指定し『取得』をクリックします。有効なドライブが表示されます。



```

'=====
'= 有効なドライブ名の取得
'= (GetLogicalDriveStrings.bas)
'=====
#include "Windows.bi"

' 有効なドライブ名の取得
Declare Function Api_GetLogicalDriveStrings& Lib "Kernel32" Alias
"GetLogicalDriveStringsA" (ByVal nBufferLength&, ByVal lpBuffer$)

' ドライブのタイプを取得
Declare Function Api_GetDriveType& Lib "Kernel32" Alias "GetDriveTypeA" (ByVal nDrive$)

#define DRIVE_NOEXIST_ROOTDIRECTORY 1
#define DRIVE_REMOVABLE 2
#define DRIVE_FIXED 3
#define DRIVE_REMOTE 4

'ドライブが不明
'ルートディレクトリなし
'取り外し可能タイプーリムーバブルディスク(FD,MO)
'固定タイプー主にハードディスク
'ネットワーク

```

```

#define DRIVE_CDROM 5
#define DRIVE_RAMDISK 6

'CD-ROMドライブ
'RAMドライブ

Var Shared drvNo As Integer
Var Shared Drv$(20) As String
Var Shared drvNam$(4) As String

Var Shared List1 As Object
Var Shared Combo1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    For i = 0 To 4
        Read drvNam$(i)
        Combo1.AddString drvNam$(i)
    Next i

    data DRIVE_REMOVABLE, DRIVE_FIXED, DRIVE_REMOTE, DRIVE_CDROM, DRIVE_RAMDISK
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Buf$ As String
    Var Ret As Long
    Var ct As Integer
    Var i As Integer

    ct = 0
    Buf$ = String$(255, Chr$(0))
    Ret = Api_GetLogicalDriveStrings(255, Buf$)
    Do until Left$(Buf$, 1) = Chr$(0)
        If Api_GetDriveType(Left$(Buf$, InStr(1, Buf$, Chr$(0)) - 1)) = drvNo Then
            ct = ct + 1
            Drv$(ct) = Left$(Buf$, 3)
        End If
        Buf$ = Right$(Buf$, Len(Buf$) - 4)
    Loop
    If ct = 0 Then Exit Sub
    For i = 1 To ct
        List1.AddString "Found : " & Str$(i) & " = " & Drv$(i)
    Next
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    drvNo = Combo1.GetCursel + 2
    List1.ResetContent
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop : End

```


ユーザー指定キーのサブキーを列挙

SHRegOpenUSKey ユーザー指定キーをオープン
SHRegEnumUSKey ユーザー指定キーのサブキーを列挙
SHRegCloseUSKey ユーザー固有のレジストリキーをクローズ



```
'=====
'= ユーザー指定キーのサブキーを列挙
'= (SHRegOpenUSKey.bas)
'=====
#include "Windows.bi"

' ユーザー指定キーをオープン
Declare Function Api_SHRegOpenUSKey& Lib "Shlwapi" Alias "SHRegOpenUSKeyA" (ByVal
pszPath$, ByVal samDesired&, ByVal hRelativeUSKey&, phNewUSKey&, ByVal fIgnoreHKCU&)

' ユーザー指定キーのサブキーを列挙
Declare Function Api_SHRegEnumUSKey& Lib "Shlwapi" Alias "SHRegEnumUSKeyA" (ByVal
hUSKey&, ByVal dwIndex&, ByVal pszName$, pcchName&, ByVal enumRegFlags&)

' ユーザー固有のレジストリキーをクローズ
Declare Function Api_SHRegCloseUSKey& Lib "Shlwapi" Alias "SHRegCloseUSKey" (ByVal
hUSKey&)

#define KEY_QUERY_VALUE &H1
#define ERROR_SUCCESS &H0

#define SHREGENUM_DEFAULT &H0
#define SHREGENUM_HKCU &H1
#define SHREGENUM_HKLM &H10
#define SHREGENUM_BOTH &H11

#define REGSTR_MAX_VALUE_LENGTH 256
#define ERROR_NO_MORE_ITEMS 259

Var Shared List1 As Object
Var Shared Radio(2) As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14 : List1.SetWindowSize 216, 78
For i = 0 To 2
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1))) : Radio(i).SetFontSize 14
Next i
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function Index bdecl () As Integer
Function Index()
    Index = Val(Mid$(GetDlgItemRadioSelect("Radio1"), 6)) - 1
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
```

```

Sub MainForm_Start()
    ShowWindow -1
    Cls
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hRegSubKey As Long
    Var sRegSubKey As String
    Var sRegEnumSubKey As String
    Var SubKeyIndex As Long
    Var SubKeyNameLength As Long
    Var RegFlags As Long
    Var Ret As Long

    'リストボックスをクリア
    List1.ResetContent

    'サブキーを列挙するキーのパスを設定
    sRegSubKey = ""

    'ユーザー固有のレジストリを問い合わせモードでオープン
    Ret = Api_SHRegOpenUSKey(sRegSubKey, KEY_QUERY_VALUE, 0, hRegSubKey, CLng(1))

    'オープンエラーのときは
    If Ret <> ERROR_SUCCESS Then

        'エラーを表示して終了
        List1.AddString "キーのオープンに失敗しました"
        Exit Sub
    End If

    Select Case Index
        Case 0
            RegFlags = SHREGENUM_DEFAULT
        Case 1
            RegFlags = SHREGENUM_HKCU
        Case 2
            RegFlags = SHREGENUM_HKLM
    End Select

    'サブキーのインデックスを初期化
    SubKeyIndex = 0

    'サブキーの列挙が完了するまで処理を反復
    Do
        '列挙するサブキー名の格納バッファを初期化
        sRegEnumSubKey = Space$(REGSTR_MAX_VALUE_LENGTH)

        'バッファサイズを初期化
        SubKeyNameLength = Len(sRegEnumSubKey)

        'インデックスで指定したサブキーを取得
        Ret = Api_SHRegEnumUSKey(hRegSubKey, SubKeyIndex, sRegEnumSubKey,
SubKeyNameLength, RegFlags)

        '列挙するサブキーがないとき
        If Ret = ERROR_NO_MORE_ITEMS Then

            '反復処理終了
            Exit Do
        End If

        'リストボックスに取得したサブキーを追加
        List1.AddString Left$(sRegEnumSubKey, InStr(sRegEnumSubKey, Chr$(0)) - 1)

```

```

'サブキーのインデックスを加算
SubKeyIndex = SubKeyIndex + 1
Loop

'オープンしたレジストリをクローズ
Ret = Api_SHRegCloseUSKey(hRegSubKey)
End Sub

'=====
' =
'=====

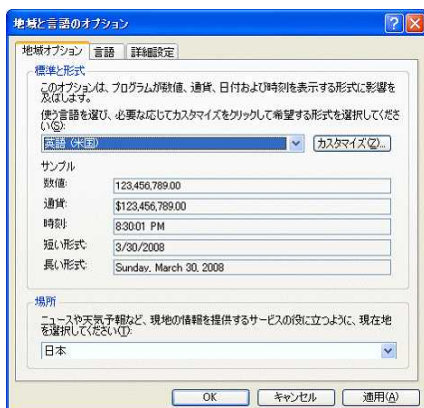
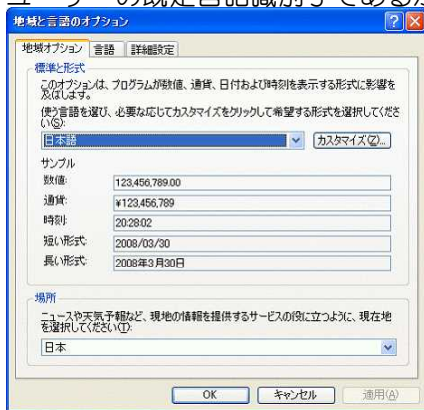
While 1
    WaitEvent
Wend
Stop
End

```

ユーザーの既定言語識別子を取得

GetUserDefaultLangID ユーザーの既定言語識別子を取得

ユーザーの既定言語識別子であるから、下図のように英語(米国)に変更すると識別子は1033(米国)になる。



```

'=====
' = ユーザーの既定言語識別子を取得
' = (GetUserDefaultLangID.bas)
'=====

#include "Windows.bi"

' ユーザーの既定言語識別子を取得
Declare Function Api_GetUserDefaultLangID% Lib "Kernel32" Alias "GetUserDefaultLangID"
()

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14

```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_GetUserDefaultLangID()

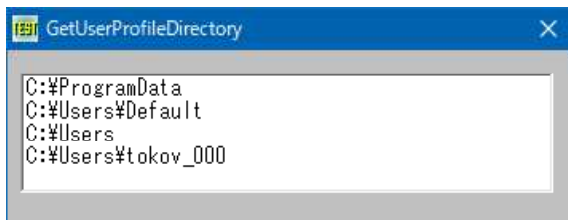
    Text1.SetWindowText "言語識別子:" & Str$(Ret)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ユーザプロフィールディレクトリの取得

ユーザプロフィールディレクトリを取得します。

GetAllUsersProfileDirectory 全ユーザー (All Users) プロファイルのルートのパスを取得
GetDefaultUserProfileDirectory 既定ユーザー (Default User) プロファイルのルートのパスを取得
GetProfilesDirectory ユーザプロフィールが格納されているルートディレクトリのパスを取得
GetCurrentUserProfileDirectory 現在ログインしているユーザのプロファイルディレクトリを取得
GetCurrentProcess 現在のプロセスに対応する疑似ハンドルを取得
OpenProcessToken アクセストークンをオープン



```
'=====
'= プロファイルディレクトリの取得
'= (GetUserProfileDirectory.bas)
'=====
#include "Windows.bi"

#define TOKEN_QUERY &H8                                'アクセストークンをクエリする

' 全ユーザプロフィールのルートのパスを取得
Declare Function Api_GetAllUsersProfileDirectory& Lib "userenv" Alias
"GetAllUsersProfileDirectoryA" (ByVal lpProfileDir$, lpCchSize&)

' 現在のプロセスの疑似ハンドルを取得
Declare Function Api_GetDefaultUserProfileDirectory& Lib "userenv" Alias
"GetDefaultUserProfileDirectoryA" (ByVal lpProfileDir$, lpCchSize&)

' 全てのユーザプロフィールが蓄えられているルートディレクトリパスを取得
Declare Function Api_GetProfilesDirectory& Lib "userenv" Alias "GetProfilesDirectoryA"
(ByVal lpProfileDir$, lpCchSize&)

' 全ユーザー (All Users) プロファイルのルートのパスを取得
Declare Function Api_GetUserProfileDirectory& Lib "userenv" Alias
"GetUserProfileDirectoryA" (ByVal hToken&, ByVal lpProfileDir$, lpCchSize&)
```

```

' 現在のプロセスに対応する疑似ハンドルを取得
Declare Function Api_GetCurrentProcess& Lib "Kernel32" Alias "GetCurrentProcess" ()

' プロセスと結び付けられたアクセストークンを開く
Declare Function Api_OpenProcessToken& Lib "advapi32" Alias "OpenProcessToken" (ByVal
ProcessHandle&, ByVal DesiredAccess&, TokenHandle&)

Var Shared List1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

'=====
'=
'=====
Declare Function StripTerminator(sInput As String) As String
Function StripTerminator(sInput As String) As String
    Var zPos As Long

    zPos = InStr(1, sInput, Chr$(0))
    If zPos > 0 Then
        StripTerminator = Left$(sInput, zPos - 1)
    Else
        StripTerminator = sInput
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var sBuffer As String
    Var Ret As Long
    Var hToken As Long

    List1.ResetContent

    '全ユーザー (All Users) プロファイルのルートのパスを取得
    sBuffer = String$(255, 0)
    Ret = Api_GetAllUsersProfileDirectory(sBuffer, 255)
    List1.AddString StripTerminator(sBuffer)

    '既定ユーザー (Default User) プロファイルのルートのパスを取得
    sBuffer = String$(255, 0)
    Ret = Api_GetDefaultUserProfileDirectory(sBuffer, 255)
    List1.AddString StripTerminator(sBuffer)

    'ユーザープロファイルが格納されているルートディレクトリのパスを取得
    sBuffer = String$(255, 0)
    Ret = Api_GetProfilesDirectory(sBuffer, 255)
    List1.AddString StripTerminator(sBuffer)

    '指定したユーザーのプロファイルのルートディレクトリのパスを取得
    sBuffer = String$(255, 0)
    Ret = Api_OpenProcessToken((Api_GetCurrentProcess), TOKEN_QUERY, hToken)

    '現在ログインしているユーザのプロファイルディレクトリを取得
    Ret = Api_GetUserProfileDirectory(hToken, sBuffer, 255)
    List1.AddString StripTerminator(sBuffer)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ユーザ名を取得 (I)

ネットワーク接続を確立するために使われるユーザ名を取得します。

WNetGetUser ユーザ名を取得



```
'=====
'= ユーザ名を取得
'= (WNetGetUser.bas)
'=====
#include "Windows.bi"

' ユーザ名を取得 (ネットワーク接続を確立するために使われるユーザ名)
Declare Function Api_WNetGetUser& Lib "Mpr" Alias "WNetGetUserA" (ByVal lpName$, ByVal
lpUserName$, lpnLength&)

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var buff As String
    Var nSize As Long
    Var nUserName As String
    Var Ret As Long

    buff = String$(255, Chr$(0))
    nSize = Len(buff)

    Ret = Api_WNetGetUser (ByVal 0, buff, nSize)

    nUserName = Left$(buff, InStr(buff, Chr$(0)) - 1)
    Text1.SetWindowText nUserName
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

ユーザ名を取得 (II)

GetUserNameEx 呼び出し側スレッドと結び付いているユーザーなどのセキュリティプリンスパルの名前を取得



```
'=====
'= ユーザー名を取得
'= (GetUserNameEx.bas)
'=====
#include "Windows.bi"

#define NameSamCompatible 2

' 呼び出し側スレッドと結び付いているユーザーなどのセキュリティプリンスパルの名前を取得
Declare Function Api_GetUserNameEx Lib "secur32" Alias "GetUserNameExA" (ByVal
NameFormat&, ByVal lpNameBuffer$, ByRef nSize&)

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Buffer As String
    Var Length As Long

    Buffer = String$(256, 0)
    Length = Len(Buffer)

    If Api_GetUserNameEx(NameSamCompatible, Buffer, Length) <> 0 Then
        Text1.SetWindowText "Username: " & Left$(Buffer, Length)
    Else
        Text1.SetWindowText "ユーザー名を取得できません！"
    End If
End Sub

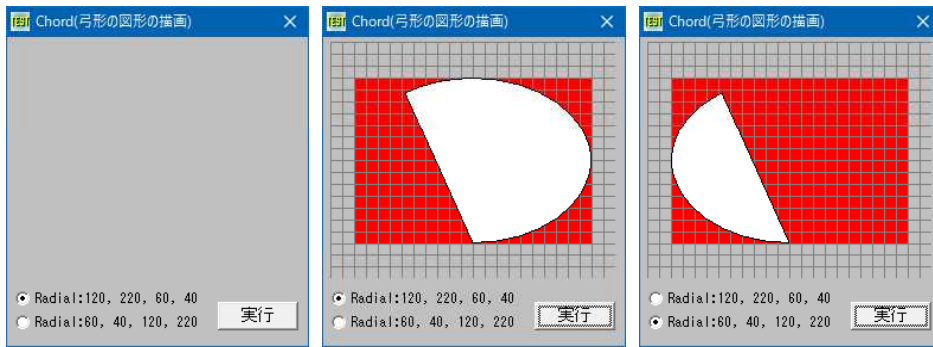
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

弓形の図形の描画

弓形の図形を描画します。

Chord 円周の一部を切り取り、切り取った部分を直線で結んだ形の図形を描画

円を包含する矩形(赤で示しています)を指定して、その中に開始点と終点を指定することで円・円弧を描画します。
 (Left, Top)、(Right, Bottom)を対角とする矩形領域に内接する楕円を用意し、円周とその中心から
 (XRadial1, YRadial1)への直線との交点を開始点、円周とその中心から(XRadial2, YRadial2)への直線との
 交点を終了点とした時、開始点から終了点への直線、開始点から終了点への円弧を描画します。
 位置関係を把握するため、10ピクセル毎に線を引いています。
 ラジオボタンで描画開始および終了の値を変え描画される図形を確認しています。



```
'=====
'= 弓形の図形の描画
'=      (Chord.bas)
'=====

#include "Windows.bi"

' 円周の一部を切り取り、切り取った部分を直線で結んだ形の図形を描画
Declare Function Api_Chord& Lib "gdi32" Alias "Chord" (ByVal hDC&, ByVal Left&, ByVal Top&,
ByVal Right&, ByVal Bottom&, ByVal XRadial1&, ByVal YRadial1&, ByVal XRadial2&,
ByVal YRadial2&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得。
その後、GDI 関数を使って、返されたデバイスコンテキスト内で描画を行える
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Type RECT
    x As Long
    y As Long
    dx As Long
    dy As Long
End Type

Var Shared Picture1 As Object
Var Shared Radiol As Object
Var Shared Radio2 As Object
Var Button1 As Object

Picture1.Attach GetDlgItem("Picture1")
Radiol.Attach GetDlgItem("Radiol") : Radiol.SetFontSize 12
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function Index bdecl () As Integer
Function Index()
    Index = val (Mid$(GetDlgRadioSelect("Radiol"), 6)) -1
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var rct As RECT
    Var hWnd As Long
    Var hDC As Long
    Var Ret As Long

    hWnd = Picture1.GethWnd
    hDC = Api_GetDC(hWnd)

    rct.x = 20
```



```

rct.y = 30
rct.dx = 220
rct.dy = 170

Picture1.line(rct.x, rct.y)-(rct.x+rct.dx, rct.y+rct.dy),,5,bf
For x = 0 To Picture1.GetWidth Step 10
    Picture1.line(x, 0)-(x, Picture1.GetHeight),, 1
Next
For y = 0 To Picture1.GetHeight Step 10
    Picture1.line(0, y)-(Picture1.GetWidth, y),, 1
Next

If Index = 0 Then
    Ret = Api_Chord(hDC, rct.x, rct.y, rct.dx, rct.dy, 120, 220, 60, 40)
Else
    Ret = Api_Chord(hDC, rct.x, rct.y, rct.dx, rct.dy, 60, 40, 120, 220)
End If

Ret = Api_ReleaseDC(hWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

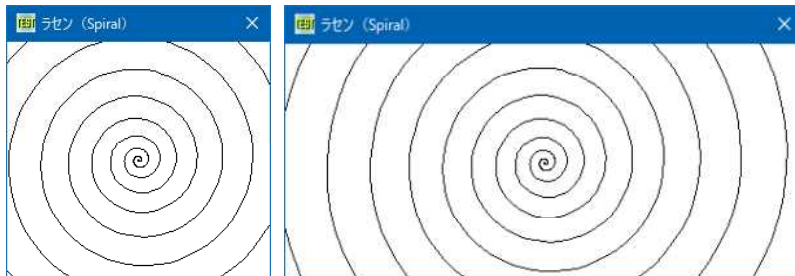
```

ラセン(Spiral)を描画

フォーム上にラセンを描画します。フォームのリサイズに連動させています。

PolylineTo 折れ線の描画

MoveToEx 現在位置を受け取るバッファを参照で指定



' 現在位置を受け取るバッファを参照で指定

```

Declare Function Api_MoveToEx Lib "gdi32" Alias "MoveToEx" (ByVal hDC&, ByVal x&, ByVal y&, ByVal lpPoint As Any)

```

' 現在位置を受け取るバッファをアドレス値で指定

```

Declare Function Api_MoveToEx Lib "gdi32" Alias "MoveToEx" (ByVal hDC&, ByVal x&, ByVal y&, ByVal lpPoint&)

```

```

'=====
'= 螺旋 (Spiral) を描画
'= (PolylineTo.bas)
'=====
#include "Windows.bi"

```

```

Type POINTAPI
    x As Long
    y As Long
End Type

```

' 折れ線の描画

```

Declare Function Api_PolylineTo Lib "gdi32" Alias "PolylineTo" (ByVal hDC&, lppt As

```

```
POINTAPI, ByVal cCount&)
```

```
' 現在位置を受け取るバッファを参照で指定
```

```
Declare Function Api_MoveToEx& Lib "gdi32" Alias "MoveToEx" (ByVal hDC&, ByVal x&, ByVal y&, ByVal lpPoint As Any)
```

```
' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
```

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

```
' デバイスコンテキストを解放
```

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
'=====
```

```
'=
```

```
'=====
```

```
Declare Sub FormPaint edec1 ()
```

```
Sub FormPaint()
```

```
    Var Pt(1000) As POINTAPI
```

```
    Var Ang As Single
```

```
    Var Rad As Single
```

```
    Var Num As Integer
```

```
    Var XMid As Long
```

```
    Var YMid As Long
```

```
    Var hDC As Long
```

```
    Var Ret As Long
```

```
'角度
```

```
'半径
```

```
'線の数
```

```
'フォームの横中心
```

```
'フォームの縦中心
```

```
'フォームのデバイスコンテキスト
```

```
'デバイスコンテキストの取得
```

```
hDC = Api_GetDC(GethWnd)
```

```
XMid = GetClientWidth / 2
```

```
YMid = GetClientHeight / 2
```

```
'Pt(0~1000)位置計算
```

```
For Num = 1 To 1000
```

```
    Ang = Num * 0.1
```

```
    Rad = Rad + Ang * 0.01
```

```
    Pt(Num).x = XMid + Cos(Ang) * Rad
```

```
    Pt(Num).y = YMid - Sin(Ang) * Rad
```

```
Next
```

```
'最初の位置を決定
```

```
Ret = Api_MoveToEx(hDC, XMid, YMid, ByVal 0&)
```

```
'Pt(1) ~ Pt(1000)まで連続した線を引く
```

```
Ret = Api_PolylineTo(hDC, Pt(1), 1000)
```

```
'デバイスコンテキストの解放
```

```
Ret = Api_ReleaseDC(GethWnd, hDC)
```

```
End Sub
```

```
'=====
```

```
'= フォームリサイズ
```

```
'=====
```

```
Declare Sub MainForm_Resize edec1 ()
```

```
Sub MainForm_Resize()
```

```
    Cls
```

```
    FormPaint
```

```
End Sub
```

```
'=====
```

```
'=
```

```
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

リージョンデモ

各種リージョンのデモンストレーションを実行します。

CreateRoundRectRgn 角の丸い長方形のリージョンを作成

CreateEllipticRgnIndirect 指定の領域をウィンドウ領域として設定

SetWindowRgn 指定の領域をウィンドウ領域として設定

DeleteObject ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除

FillRgn 指定のブラシで領域を塗りつぶす

CreatePolygonRgn 多角形のリージョンを作成

CreatePolyPolygonRgn 複数の多角形を組み合わせて、一つの多角形領域を作成

GetWindowRect ウィンドウの座標をスクリーン座標系で取得

CreateRectRgnIndirect 矩形領域をRECT構造体に基づいて作成

InflateRect RECT構造体の座標値を拡大・縮小

CopyRect RECT構造体の座標値を別のRECT構造体にコピー

EqualRgn 指定した2つの領域が等しいかどうか判定

SetRect RECT構造体の値を設定

SetRectRgn 既存の領域を矩形の領域に変更

InvertRgn 指定された領域の色を反転

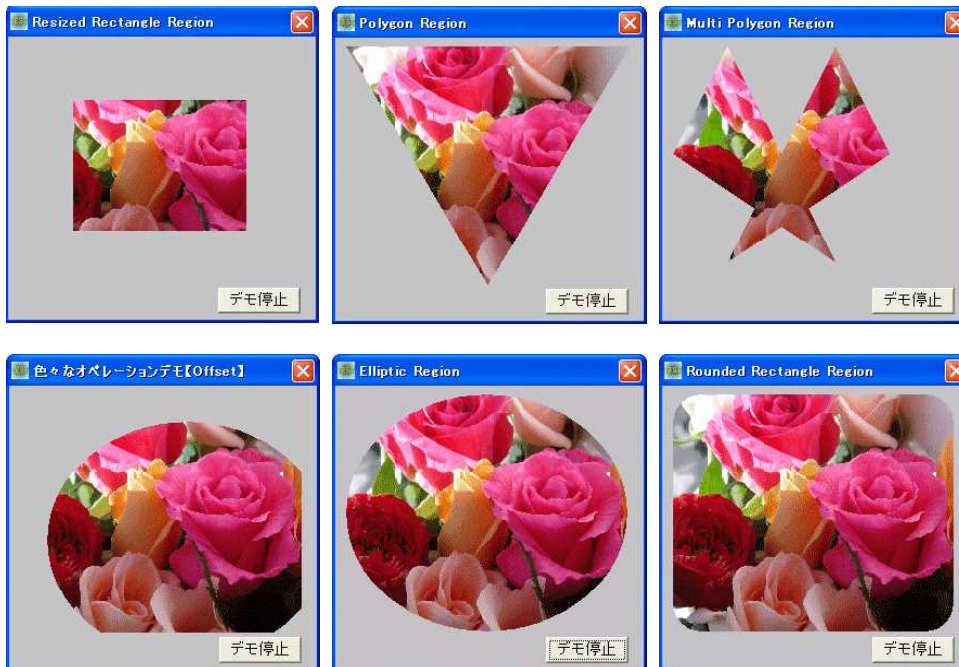
FrameRgn 指定の領域の周囲に指定のブラシで境界線を描く

CreateHatchBrush ハッチパターンの論理ブラシを作成

OffsetRgn 領域を移動

GetDC デバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```
'=====
'= リージョンデモ
'= (Region.bas)
'=====
```

```
#include "Windows.bi"
```

```
Type POINTAPI
```

```
    x As Long
```

```
    y As Long
```

```
End Type
```

```
Type RECT
```

```
    left As Long
```

```
    top As Long
```

```
    right As Long
```

```
    bottom As Long
```

```
End Type
```

```
Type RGNDATAHEADER
```

```
    dwSize As Long
```

```
    iType As Long
```

```

        nCount      As Long
        nRgnSize    As Long
        rcBound     As RECT
End Type

```

```

Type RGNDATA
    rdh          As RGNDATAHEADER
    Buffer        As Byte
End Type

```

・ 角の丸い長方形のリージョンを作成

```

Declare Function Api_CreateRoundRectRgn& Lib "gdi32" Alias "CreateRoundRectRgn" (ByVal
nLeftRect&, ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&, ByVal
nWidthEllipse&, ByVal nHeightEllipse&)

```

・ 指定の領域をウィンドウ領域として設定

```

Declare Function Api_CreateEllipticRgnIndirect& Lib "gdi32" Alias
"CreateEllipticRgnIndirect" (lpRect As RECT)

```

・ 指定の領域をウィンドウ領域として設定

```

Declare Function Api_SetWindowRgn& Lib "user32" Alias "SetWindowRgn" (ByVal hWnd&, ByVal
hRgn&, ByVal bRedraw&)

```

・ ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる

```

Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

```

・ 指定のブラシで領域を塗りつぶす

```

Declare Function Api_FillRgn& Lib "gdi32" Alias "FillRgn" (ByVal hDC&, ByVal hRgn&, ByVal
hBrush&)

```

・ 多角形のリージョンを作成

```

Declare Function Api_CreatePolygonRgn& Lib "gdi32" Alias "CreatePolygonRgn" (lppt As
POINTAPI, ByVal nCount&, ByVal nPolyFillMode&)

```

・ 複数の多角形を組み合わせて、一つの多角形領域を作成する

```

Declare Function Api_CreatePolyPolygonRgn& Lib "gdi32" Alias "CreatePolyPolygonRgn"
(ByRef lpPoint As POINTAPI, ByRef lpPolyCounts&, ByVal nCount&, ByVal nPolyFillMode&)

```

・ ウィンドウの座標をスクリーン座標系で取得

```

Declare Function Api_GetWindowRect& Lib "user32" Alias "GetWindowRect" (ByVal hWnd&,
lpRect As RECT)

```

・ 矩形領域をRECT構造体に基づいて作成

```

Declare Function Api_CreateRectRgnIndirect& Lib "gdi32" Alias "CreateRectRgnIndirect"
(lpRect As RECT)

```

・ RECT構造体の座標値を拡大・縮小

```

Declare Function Api_InflateRect& Lib "user32" Alias "InflateRect" (lpRect As RECT, ByVal
x&, ByVal y&)

```

・ RECT構造体の座標値を別のRECT構造体にコピー

```

Declare Function Api_CopyRect& Lib "user32" Alias "CopyRect" (lpDestRect As RECT,
lpSourceRect As RECT)

```

・ 指定した2つの領域が等しいかどうか判定

```

Declare Function Api_EqualRgn& Lib "gdi32" Alias "EqualRgn" (ByVal hSrcRgn1&, ByVal
hSrcRgn2&)

```

・ RECT構造体の値を設定

```

Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

```

・ 既存の領域を矩形の領域に変更

```

Declare Function Api_SetRectRgn& Lib "gdi32" Alias "SetRectRgn" (ByVal hRgn&, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

```

・ 指定された領域の色を反転

```

Declare Function Api_InvertRgn& Lib "gdi32" Alias "InvertRgn" (ByVal hDC&, ByVal hRgn&)

```

' 指定の領域の周囲に指定のブラシで境界線を描く

```
Declare Function Api_FrameRgn& Lib "gdi32" Alias "FrameRgn" (ByVal hDC&, ByVal hRgn&,
ByVal hBrush&, ByVal nWidth&, ByVal nHeight&)
```

' ハッチパターンの論理ブラシを作成

```
Declare Function Api_CreateHatchBrush& Lib "gdi32" Alias "CreateHatchBrush" (ByVal
nIndex&, ByVal crColor&)
```

' 領域を移動

```
Declare Function Api_OffsetRgn& Lib "gdi32" Alias "OffsetRgn" (ByVal hRgn&, ByVal x&,
ByVal y&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
#define HS_BDIAGONAL 3           '斜線 (左上-右下)
#define HS_CROSS 4              '水平と垂直クロスハッチ
#define HS_DIAGCROSS 5         '45度のクロスハッチ
#define HS_FDIAGONAL 2         '45度下向きのハッチ (左から右へ)
#define HS_HORIZONTAL 0        '水平ハッチ
#define HS_VERTICAL 1          '垂直ハッチ
```

```
#define ALTERNATE 1
#define WINDING 2
```

```
#define ERROR 0
#define NULLREGION 1
#define SIMPLEREGION 2
#define COMPLEXREGION 3
```

```
Var Shared Timer1 As Object
Var Shared Picture1 As Object
Var Shared Button1 As Object
Var Shared Bitmap As Object
```

```
Timer1.Attach GetDlgItem("Timer1")
Picture1.Attach GetDlgItem("Picture1")
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
BitmapObject Bitmap
```

```
Var Shared NextOp As Long
Var Shared hRgn As Long
Var Shared Ret As Long
```

```
Var Shared WndRect As RECT
Var Shared TmpRect As RECT
Var Shared FLG As Integer
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Button1.SetWindowText "デモ開始"
```

```
    Bitmap.LoadFile "FlowerZ.bmp"
    Picture1.StretchBitmap Bitmap, 0, 0, Picture1.GetWidth, Picture1.GetHeight
    Bitmap.DeleteObject
```

```
    Timer1.SetInterval 200
    Timer1.Enable 0
```

```
    'Get Rectangle coordinates of picturebox and set Rectangle
```

```
    Ret = Api_GetWindowRect(Picture1.GethWnd, WndRect)
```

```
    Ret = Api_SetRect(WndRect, 0, 0, WndRect.right - WndRect.left, WndRect.bottom -
WndRect.top)
End Sub
```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    FLG = 1 - FLG

    If FLG = 0 Then
        Timer1.Enable 0
        Button1.SetWindowText "デモ開始"
    Else
        Timer1.Enable -1
        Button1.SetWindowText "デモ停止"
    End If
End Sub

'=====
'=
'=====
Declare Sub RectFromAnyRgnDemo edec1 ()
Sub RectFromAnyRgnDemo()

    'デモのための領域指定
    hRgn = Api_CreateEllipticRgnIndirect(WndRect)
    SetWindowText "いろいろな領域から矩形領域に変換"

    '(0, 0)-(100, 100)の領域を設定
    Ret = Api_SetRectRgn(hRgn, 0, 0, 100, 100)
End Sub

'=====
'= Elliptic region
'=====
Declare Sub ElliRgnDemo edec1 ()
Sub ElliRgnDemo()
    hRgn = Api_CreateEllipticRgnIndirect(WndRect)

    If DoInvert Then
        SetWindowText "Elliptic Region【With Invert】"
    Else
        SetWindowText "Elliptic Region"
    End If
End Sub

'=====
'= Polygonal region
'=====
Declare Sub PolyRgnDemo edec1 ()
Sub PolyRgnDemo()
    Var PolyPoints(2) As POINTAPI

    PolyPoints(0).x = 0
    PolyPoints(0).y = 0

    PolyPoints(1).x = WndRect.right
    PolyPoints(1).y = 0

    PolyPoints(2).x = WndRect.right / 2
    PolyPoints(2).y = WndRect.bottom
    hRgn = Api_CreatePolygonRgn(PolyPoints(0), 3, WINDING)

    SetWindowText "Polygon Region"
End Sub

'=====
'= Multiple Polygonal (PolyPoly) region
'=====
Declare Sub MultiPolyRgnDemo edec1 ()
Sub MultiPolyRgnDemo()
    Var PolyPoints(5) As POINTAPI

```

```

Var FigurePtCnts(1) As Long
PolyPoints(0).x = 0
PolyPoints(0).y = 100

PolyPoints(1).x = 50
PolyPoints(1).y = 0

PolyPoints(2).x = 150
PolyPoints(2).y = 200

FigurePtCnts(0) = 3

PolyPoints(3).x = 150
PolyPoints(3).y = 0

PolyPoints(4).x = 200
PolyPoints(4).y = 100

PolyPoints(5).x = 50
PolyPoints(5).y = 200

FigurePtCnts(1) = 3

hRgn = Api_CreatePolyPolygonRgn(PolyPoints(0), FigurePtCnts(0), 2, WINDING)

SetWindowText "Multi Polygon Region"
End Sub

'=====
'= Multiple Polygonal (PolyPoly) region
'=====
Declare Sub RoundRectRgnDemo edec1 ()
Sub RoundRectRgnDemo ()
    hRgn = Api_CreateRoundRectRgn(WndRect.left, WndRect.top, WndRect.right,
    WndRect.bottom, 50, 50)
    SetWindowText "Rounded Rectangle Region"
End Sub

'=====
'= Inflate (Resize) the Rect and create Rectangle region
'=====
Declare Sub ResizeRectRgnDemo edec1 ()
Sub ResizeRectRgnDemo ()
    Ret = Api_CopyRect(TmpRect, WndRect)
    Ret = Api_InflateRect(TmpRect, -50, -50) 'Reduce rect by 50 pix from each side

    hRgn = Api_CreateRectRgnIndirect(TmpRect)
    SetWindowText "Resized Rectangle Region"
End Sub

'=====
'= Rectangle region
'=====
Declare Sub RectRgnDemo edec1 ()
Sub RectRgnDemo ()
    hRgn = Api_CreateRectRgnIndirect(WndRect)
    SetWindowText "Rectangle Region"
End Sub

'=====
'= Fill and Invert Region Demo(1=Fill, 2=Invert, 3=Frame, 4=offset)
'=====
Declare Sub MiscRgnOperationDemo (OpCode As Integer)
Sub MiscRgnOperationDemo (OpCode As Integer)
    Var hBrush As Long
    Var hDC As Long

    hDC = Api_GetDC(Picture1.GethWnd)

```

'デモのための領域設定

```
hRgn = Api_CreateEllipticRgnIndirect(WndRect)
SetWindowText "色々なオペレーションデモ"
hBrush = Api_CreateHatchBrush(HS_FDIAGONAL, RGB(0, 255, 0))

'Fill
If OpCode = 1 Then
    Ret = Api_FillRgn(hDC, hRgn, hBrush)
    SetWindowText GetWindowText & "[Fill]"
End If

'Invert
If OpCode = 2 Then
    Ret = Api_InvertRgn(hDC, hRgn)
    SetWindowText GetWindowText & "[Invert]"
End If

'Frame
If OpCode = 3 Then
    Ret = Api_FrameRgn(hDC, hRgn, hBrush, 5, 5) '10x10 brush
    SetWindowText GetWindowText & "[Frame]"
End If

If OpCode = 4 Then
    Ret = Api_OffsetRgn(hRgn, 25, 25) 'Move region by x=50 y=50 pix
    SetWindowText GetWindowText & "[Offset]"
End If
```

```
Ret = Api_DeleteObject(hBrush)
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    NextOp = NextOp + 1
```

```
    Select Case NextOp
        Case 1
            ElliRgnDemo
        Case 2
            ElliRgnDemo
            MiscRgnOperationDemo(1)
        Case 3
            ElliRgnDemo
            MiscRgnOperationDemo(1)
            MiscRgnOperationDemo(2)
            MiscRgnOperationDemo(4)
        Case 4
            ElliRgnDemo
            MiscRgnOperationDemo(3)
        Case 5
            PolyRgnDemo
        Case 6
            RoundRectRgnDemo
        Case 7
            RectRgnDemo
        Case 8
            ResizeRectRgnDemo
        Case 9
            RectFromAnyRgnDemo
        Case 10
            MultiPolyRgnDemo
        Case 10
            MultiPolyRgnDemo
        Case 11
            Ret = Api_SetWindowRgn(Picture1.GethWnd, 0, True) 'Reset region
            SetWindowText "領域設定無し"
            NextOp = 0
```



```

End Select

'領域をピクチャボックスに設定
Ret = Api_SetWindowRgn(Picture1.GethWnd, hRgn, True)

'作成した領域を削除
If hRgn <> 0 Then
    Ret = Api_DeleteObject(hRgn)
End If
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    If hRgn <> 0 Then
        Ret = Api_DeleteObject(hRgn)
        Ret = Api_ReleaseDC(Picture1.GethWnd, hDC)
    End If
End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックス・コンボボックスの行間隔変更

リストボックス・コンボボックスの項目高さ(行間隔)を12ドット(ピクセル)～28ドットまで可変します。

SendMessage ウィンドウにメッセージを送信

リストボックスに対して

LB_GETITEMHEIGHT(&H1A1) 高さを取得

LB_SETITEMHEIGHT(&H1A0) 高さを設定

コンボボックスに対して

CB_GETITEMHEIGHT(&H154) 高さを取得

CB_SETITEMHEIGHT(&H153) 高さを設定

例では、項目の高さを取得(初期高さ:12ピクセル)し(大)をクリックする毎に1.1倍の高さを設定、最大行間隔28ピクセルまで変更させています。

また、(小)をクリックする毎に1.1で除算し高さを設定しています。



```

'=====
'= リストボックス・コンボボックスの行間隔変更
'= (ListHeight.bas)
'=====
#include "Windows.bi"

```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal  
wMsg&, ByVal wParam&, lParam As Any)
```

```
#define LB_GETITEMHEIGHT &H1A1
```

'リストボックス項目の高さを取得する

```
#define LB_SETITEMHEIGHT &H1A0
```

'リストボックス項目の高さを設定する

```
#define CB_GETITEMHEIGHT &H154
```

'コンボボックス内の項目の高さを取得する

```
#define CB_SETITEMHEIGHT &H153
```

'コンボボックス内の項目の高さを設定する

```
#define CB_SHOWDROPDOWN &H14F
```

'コンボボックスのリストボックスの表示または非表示を切り替える

```
Var Shared List1 As Object
```

```
Var Shared Comb1 As Object
```

```
Var Shared Text1 As Object
```

```
Var Shared Button1 As Object
```

```
Var Shared Button2 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 12
```

```
Comb1.Attach GetDlgItem("Comb1") : Comb1.SetFontSize 12
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 12
```

```
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
```

```
Var Shared ItemHeight As Long
```

```
'=====
'=
'=====
```

```
Declare Sub NumSet ()
```

```
Sub NumSet ()
```

```
    Var Num As String
```

```
    List1.SetWindowSize 88,160
```

```
    List1.ResetContent
```

```
    Comb1.ResetContent
```

'文字列用意

```
For Count = 1 To 12
```

```
Num = Format$(Rnd(Count) * 100000000, "###,###,###")
```

```
    List1.AddString Num
```

```
    Comb1.AddString Num
```

```
Next Count
```

'リストボックス項目の高さを取得

```
ItemHeight = Api_SendMessage(List1.GethWnd, LB_GETITEMHEIGHT, 0, ByVal 0)
```

'コンボボックス項目の高さを取得(ListBoxと同じ値なので不要)

```
' ItemHeight = Api_SendMessage(Comb1.GethWnd, CB_GETITEMHEIGHT, 0, ByVal 0)
```

'マウス砂時計解除

```
SetMousePointer 0
```

```
Ret = Api_SendMessage(Comb1.GethWnd, CB_SHOWDROPDOWN, 1, ByVal 0)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start ()
```

```
    NumSet
```

'リストボックス項目の高さを表示

```
Text1.SetWindowText "高さ(Pixel)" & Chr$(13,10) & "改行間隔:" & Str$(ItemHeight)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```

Sub Button1_on()
    NumSet

    '新しいリストボックス(コンボボックス)項目の高さを指定
    ItemHeight = ItemHeight * 1.1

    If ItemHeight > 28 Then ItemHeight = 28

    'ストボックス項目の高さを設定
    Ret = Api_SendMessage(List1.GethWnd, LB_SETITEMHEIGHT, 0, ByVal ItemHeight)
    Ret = Api_SendMessage(Combo1.GethWnd, CB_SETITEMHEIGHT, 0, ByVal ItemHeight)

    'リストボックス項目の高さを表示
    Text1.SetWindowText "高さ(Pixel)" & Chr$(13,10) & "改行間隔:" & Str$(ItemHeight)
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    NumSet

    '新しいリストボックス項目の高さを指定
    ItemHeight = ItemHeight / 1.1

    If ItemHeight < 12 Then ItemHeight = 12

    'リストボックス項目の高さを設定
    Ret = Api_SendMessage(List1.GethWnd, LB_SETITEMHEIGHT, 0, ByVal ItemHeight)
    Ret = Api_SendMessage(Combo1.GethWnd, CB_SETITEMHEIGHT, 0, ByVal ItemHeight)

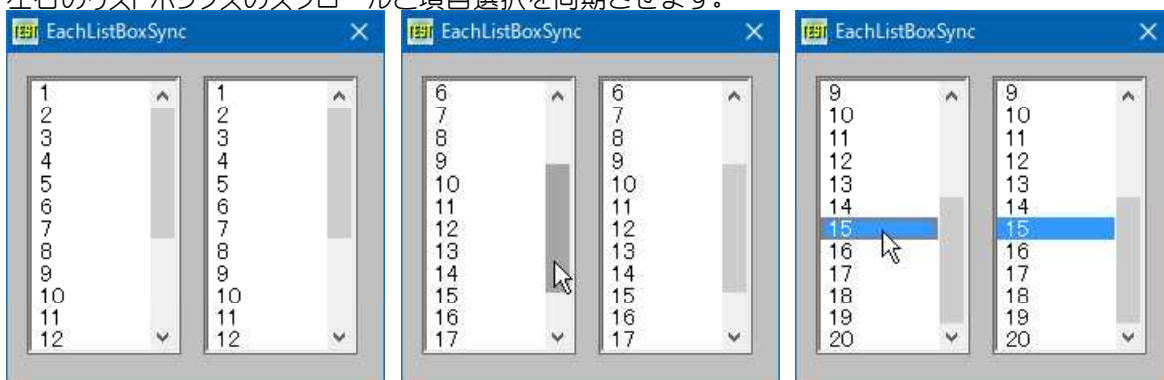
    'リストボックス項目の高さを表示
    Text1.SetWindowText "高さ(Pixel)" & Chr$(13,10) & "改行間隔:" & Str$(ItemHeight)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックス同士の同期

左右のリストボックスのスクロールと項目選択を同期させます。



```

'=====
'= リストボックス同士の同期
'= (ListBoxSync.bas)
'=====

```

```

#include "Windows.bi"

Var Shared List1 As Object
Var Shared List2 As Object
Var Shared Timer1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
List2.Attach GetDlgItem("List2") : List2.SetFontSize 14
Timer1.Attach GetDlgItem("Timer1")

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    For i = 1 To 20
        List1.AddString Str$(i)
        List2.AddString Str$(i)
    Next

    Timer1.SetInterval 5
    Timer1.Enable -1
End Sub

'=====
'=
'=====
Declare Sub Timer1_Timer edec1 ()
Sub Timer1_Timer()
    Var List1TopIndex As Integer
    Var List2TopIndex As Integer

    '先頭に表示される項目を揃える
    List1TopIndex = List1.GetTopIndex
    If List1TopIndex <> List2TopIndex Then
        List2.SetTopIndex List1TopIndex
        List2TopIndex = List1TopIndex
    End If

    '選択項目を揃える
    If List1.GetCursel <> List2.GetCursel Then
        List2.SetCursel List1.GetCursel
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックス内の項目を全選択・解除

リストボックス内の項目を全選択、解除を実行します。

SendMessage ウィンドウにメッセージを送信

LB_SETSEL (&H185) 複数選択リストボックス内の文字列を選択する



```
'=====
'= リストボックス内の項目を全選択・解除
'= (SelectingAllItems.bas)
'=====
#include "Windows.bi"
```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)
```

```
#define LB_SETSEL &H185 '複数選択リストボックス内の文字列を選択する
```

```
Var Shared List1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14 : List1.SetWindowSize 208, 123
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    For i = 1 To 20
        List1.AddString format$(rnd(i) * 10000000, "###,###,###.##")
    Next
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_SendMessage(List1.GethWnd, LB_SETSEL, True, ByVal -1)
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var Ret As Long

    Ret = Api_SendMessage(List1.GethWnd, LB_SETSEL, False, ByVal -1)
End Sub
```

```
'=====
'=
'=====
```

```
While 1
    WaitEvent
Wend
Stop
End
```

リストボックス内のリストを取り出す

リストボックス内のリストを取り出します。

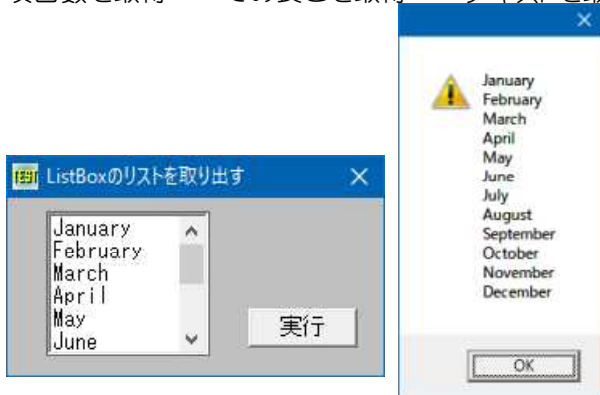
SendMessage 指定のウィンドウにメッセージを送る

LB_GETCOUNT (&H18B) リストボックスの項目数を取得する

LB_GETTEXT (&H189) リストボックスから文字列を取得する

LB_GETTEXTLEN (&H18A) リストボックス内の文字列の長さを取得する

項目数を取得 → その長さを取得 → テキストを取得



ListBoxのプロパティで**複数選択**、または**拡張選択**どちらか一方を**あり**に設定しておきます。

```
'=====
'= リストボックスのリストを取り出す
'= (SendMessage5.bas)
'=====
#include "Windows.bi"
```

ウィンドウにメッセージを送信

Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

```
#define LB_GETCOUNT &H18B
#define LB_GETTEXT &H189
#define LB_GETTEXTLEN &H18A
```

'リストボックスの項目数を取得する
'リストボックスから文字列を取得する
'リストボックス内の文字列の長さを取得する

```
#define vbCrLf (Chr$(13) & Chr$(10))
```

'キャリッジリターンとラインフィード(¥r¥n)

```
Var Shared List1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
```

```
List1.AddString "January"
List1.AddString "February"
List1.AddString "March"
List1.AddString "April"
List1.AddString "May"
List1.AddString "June"
List1.AddString "July"
List1.AddString "August"
List1.AddString "September"
List1.AddString "October"
List1.AddString "November"
List1.AddString "December"
```

```
End Sub
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
    Var num As Long
    Var i As Integer
    Var txt As String
```

```

Var Buffer As String
Var length As Long

'コンボボックスの項目数を取得
num = Api_SendMessage(List1.GethWnd, LB_GETCOUNT, 0, 0)

'リストを調べる
For i = 0 To num - 1

    '項目の長さを取得
    length = Api_SendMessage(List1.GethWnd, LB_GETTEXTLEN, i, 0)

    'バッファの確保
    Buffer = Space$(length + 1)

    'テキストを取得
    length = Api_SendMessage(List1.GethWnd, LB_GETTEXT, i, Buffer)
    txt = txt & Left$(Buffer, length) & vbCrLf
Next i

A% = MessageBox("", txt, 0, 2)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックスに水平スクロールバーを

Listboxに水平スクロールバーを設定します。
SendMessage ウィンドウにメッセージを送信
ShowScrollBar スクロールバーを表示

List1は、垂直スクロール**あり**に設定します。VBではSendMessageでLB_SETHORIZONTALEXTENTを送るだけでいいのですが、F-BASICではShowScrollBarが必要でした。



※スクロールに支障はありませんが、水平スクロールバーのスライダーがスグに初期状態に戻ってしまいます。固定する方法を探しています。(^^;

ついでですが、F-BasicではListBox内でのTab「Chr\$(9)」は効きません。

```

'=====
'= リストボックスに水平スクロールバーを
'= (HScrollBarInListBox.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

' スクロールバーを表示
Declare Function Api_ShowScrollBar& Lib "user32" Alias "ShowScrollBar" (ByVal hWnd&,

```

```
ByVal wBar&, ByVal bShow&)
```

```
#define LB_SETHORIZONTALTEXT &H194  
#define SB_HORZ 0
```

'リストボックスの水平範囲を設定する
'標準スクロールバーの水平

```
Var Shared List1 As Object  
Var Shared Button1 As Object  
Var Shared Button2 As Object  
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14  
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
```

```
Var Shared n As Long
```

```
'=====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var txt As String
```

```
    Var i As Integer
```

```
    n = 0
```

```
    For i = 1 To 10
```

```
        Read txt
```

```
        List1.AddString txt
```

'一番長い文字列の文字数を取得

```
    If n < Len(txt) Then
```

```
        n = Len(txt)
```

```
    End If
```

```
Next i
```

```
Data "123456789012345"
```

```
Data "123456789012345678"
```

```
Data "123456789012345678901234567"
```

```
Data "123456789012345678901234567890"
```

```
Data "123"
```

```
Data "123456"
```

```
Data "123456789"
```

```
Data "123456789012"
```

```
Data "123456789012345678901"
```

```
Data "123456789012345678901234"
```

```
End Sub
```

```
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var ListWidth As Long
```

```
    Var Ret As Long
```

'水平スクロールバーを表示(フォントサイズ×最大文字数)

```
ListWidth = List1.GetFontSize * n / 2 + 5
```

'水平スクロールバーを表示

```
Ret = Api_SendMessage(List1.GethWnd, LB_SETHORIZONTALTEXT, ListWidth, ByVal 0)
```

```
Ret = Api_ShowScrollBar(List1.GethWnd, SB_HORZ, True)
```

```
End Sub
```

```
'=====
```

```
Declare Sub Button2_on edec1 ()
```

```
Sub Button2_on()
```

```
    Var ListWidth As Long
```

```
    Var Ret As Long
```



```

'幅を指定
ListWidth = 0

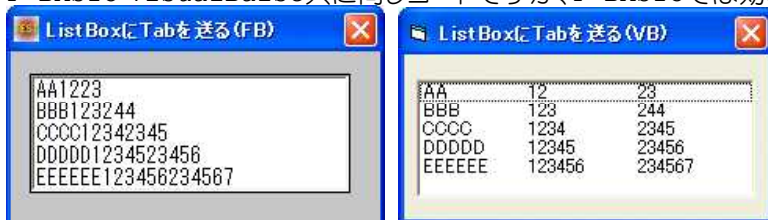
'水平にスクロールできる幅を設定
Ret = Api_SendMessage(List1.GethWnd, LB_SETHORIZONTALEXTENT, ListWidth, ByVal 0)
Ret = Api_ShowScrollBar(List1.GethWnd, SB_HORZ, False)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックスにタブストップを設定

F-BASIC・VisualBaisc共に同じコードですが、F-BASICでは効き目がないようです。残念！！^^;)



```

'=====
'= リストボックスにタブストップを設定
'=====
#include "Windows.bi"

' メッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

#define LB_SETTABSTOPS &H192
#define vbTab Chr$(9)

'リストボックスにタブストップを設定
'TAB

Var Shared List1 As Object
List1.Attach GetDlgItem("List1")
List1.SetFontSize 14

'=====
'=
'=====
Declare Sub Mainform_Start edec1 ()
Sub Mainform_Start ()
    Var Ret As Long
    Var lTabCol As Long
    Var lTabS(2) As Long

    'タブストップの配列
    lTabCol = 3
    lTabS(0) = 100
    lTabS(1) = 150
    lTabS(2) = 200

    Ret = Api_SendMessage(List1.GethWnd, LB_SETTABSTOPS, lTabCol, lTabS(0))
    List1.AddString "AA" & vbTab & "12" & vbTab & "23"
    List1.AddString "BBB" & vbTab & "123" & vbTab & "244"
    List1.AddString "CCCC" & vbTab & "1234" & vbTab & "2345"
    List1.AddString "DDDDD" & vbTab & "12345" & vbTab & "23456"
    List1.AddString "EEEEEE" & vbTab & "123456" & vbTab & "234567"

```

```
End Sub
```

```
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

リストボックスの項目を重複させずに追加

前方一致文字列を検索し、重複させずに追加します。

SendMessage ウィンドウにメッセージを送信

LB_FINDSTRING (&H18F) 前方部分検索

LB_ADDSTRING (&H180) リストボックスに文字列を追加



```
'=====
'= リストボックスの項目を重複させずに追加
'= (LB_FINDSTRING.bas)
'=====
#include "Windows.bi"
```

' ウィンドウにメッセージを送信。この関数は、指定したウィンドウのウィンドウプロシージャが処理を終了するまで制御を返さない

```
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal  
wMsg&, ByVal wParam&, lParam As Any)
```

```
#define LB_FINDSTRING &H18F  
#define LB_ADDSTRING &H180  
#define LB_ERR (-1)
```

' 前方部分検索
' リストボックスに文字列を追加する
' エラー

```
Var Shared Edit1 As Object  
Var Shared Text1 As Object  
Var Shared List1 As Object  
Var Shared Button1 As Object
```

```
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14  
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14  
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14  
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()  
Sub MainForm_Start()  
    For i = 1 To 20  
        List1.AddString Trim$(Str$(i * 1000))  
    Next  
End Sub
```

```
'=====
'=
'=====
Declare Sub Button1_on edec1 ()  
Sub Button1_on()
```

```

Var Start As Long
Var MyStr As String
Var Cnt As Long
Var Ret As Long

'検索条件を全範囲に設定
Start = -1

'検索する文字列を指定
MyStr = Edit1.GetWindowText
If MyStr = "" Then Exit Sub

'指定した文字列を検索
Ret = Api_SendMessage(List1.GethWnd, LB_FINDSTRING, Start, MyStr)

'検索できなかったとき
If Ret = LB_ERR Then

    'リストボックスに項目を追加
    Cnt = Api_SendMessage(List1.GethWnd, LB_ADDSTRING, 0, MyStr)

    '追加した項目を選択
    List1.SetCursel Cnt

    '処理結果を表示
    Text1.SetWindowText Str$(Cnt + 1) & "番目に追加"

'検索できたとき
Else

    '検索した項目を選択
    List1.SetCursel Ret

    '項目のインデックスを表示
    Text1.SetWindowText "追加できません"
End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックスのスムーズスクロールを有効無効に

SystemParametersInfo システム全体に関するパラメータを取得・設定
SPI_SETLISTBOXSMOOTHSCROLLING (&H1007) リストボックスのスムーズスクロールを有効・無効に
SPIF_SENDWININICHANGE (&H2) 全てのアプリケーションに通知して更新する
SPIF_UPDATEINIFILE (&H1) ユーザープロファイルの更新を指定する



有効・無効を選択し「実行」ボタンをクリック後、スクロールバーの「へ」「v」をクリックしスクロールを確認してください。

```

'=====
'= リストボックスのスムーズスクロールを有効無効に設定
'= (SPI_SETLISTBOXSMOOTHSCROLLING.bas)
'=====

```

```

#include "Windows.bi"

#define SPI_SETLISTBOXSMOOTHSCROLLING &H1007 'リストボックスのスムーズスクロールを有効・無効に
#define SPIF_SENDWININICHANGE &H2 '全てのアプリケーションに通知して更新する
#define SPIF_UPDATEINIFILE &H1 'ユーザープロファイルの更新を指定する定数の宣言
#define SPIF_SENDCHANGE SPIF_SENDWININICHANGE

' システム全体に関するパラメータを取得・設定
Declare Function Api_SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA"
    (ByVal uiAction&, ByVal uiParam&, pvParam As Any, ByVal fWinIni&)

Var Shared List1 As Object
Var Shared Radiol As Object
Var Shared Radio2 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Radiol.Attach GetDlgItem("Radiol") : Radiol.SetFontSize 14
Radio2.Attach GetDlgItem("Radio2") : Radio2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

' =====
' =
' =====
Declare Function Index bdecl () As Integer
Function Index()
    Index = Val (Mid$(GetDlgRadioSelect("Radiol"), 6)) -1
End Function

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    For i = 0 To 20
        List1.AddString Str$(i)
    Next
End Sub

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    ' スmoothスクロール効果の有効無効を設定
    Ret = Api_SystemParametersInfo(SPI_SETLISTBOXSMOOTHSCROLLING, 0, ByVal Index,
    SPIF_UPDATEINIFILE Or SPIF_SENDCHANGE)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックスの選択項目に挿入ガイドを

DrawInsert インデックスで示された項目の前に挿入ガイドを表示



```
'=====
'= リストボックスの選択項目に挿入ガイドを
'= (DrawInsert.bas)
'=====
#include "Windows.bi"

' インデックスで示された項目の前に挿入ガイドを表示
Declare Sub Api_DrawInsert Lib "comctl32" Alias "DrawInsert" (ByVal handParent&, ByVal
hLB&, ByVal nItem&)

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    For i = 1 To 20
        List1.AddString Str$(i * 1000)
    Next
End Sub

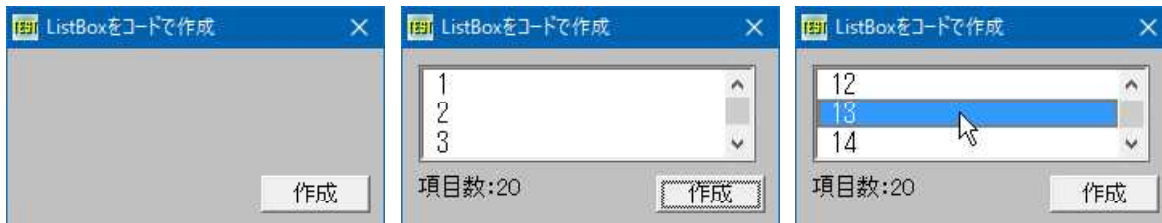
'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Api_DrawInsert GethWnd, List1.GethWnd, List1.GetCursel
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Api_DrawInsert GethWnd, List1.GethWnd, -1
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End
```

リストボックスをコードで作成

CreateWindowEX 新しいウインドウ(コントロール)を作成
DestroyWindow ウインドウ(コントロール)の解放
SendMessage ウインドウにメッセージを送信



```
'=====
'= リストボックスをコードで作成
'= (CreateWindowEx3.bas)
'=====
#include "Windows.bi"

' ウィンドウ (コントロール) を作成
Declare Function Api_CreateWindowEx& Lib "user32" Alias "CreateWindowExA" (ByVal
ExStyle&, ByVal ClassName$, ByVal WinName$, ByVal Style&, ByVal x&, ByVal y&, ByVal
nWidth&, ByVal nHeight&, ByVal Parent&, ByVal Menu&, ByVal Instance&, ByVal Param&)

' CreateWindowExの解放
Declare Function Api_DestroyWindow& Lib "user32" Alias "DestroyWindow" (ByVal hWnd&)

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

#define WS_BORDER &H8000000
#define WS_CHILD &H40000000
#define WS_CLIPSIBLINGS &H40000000
#define WS_EX_WINDOWEDGE &H100
#define WS_EX_OVERLAPPEDWINDOW &H300

#define WS_VISIBLE &H100000000
#define WS_HSCROLL &H100000
#define WS_VSCROLL &H200000
#define WS_EX_CLIENTEDGE &H200
#define WS_THICKFRAME &H40000

#define BS_PUSHBUTTON &H0
#define BS_AUTOCHECKBOX &H3
#define BS_AUTORADIOBUTTON &H9
#define BS_GROUPBOX &H7

#define ES_AUTOHSCROLL &H80
#define ES_AUTOVSCROLL &H40
#define ES_CENTER 1
#define ES_LEFT 0
#define ES_LOWERCASE &H10
#define ES_MULTILINE 4
#define ES_NOHIDESEL &H100
#define ES_NUMBER &H2000
#define ES_OEMCONVERT &H400
#define ES_PASSWORD &H20
#define ES_READONLY &H800
#define ES_RIGHT 2
#define ES_UPPERCASE 8
#define ES_WANTRETURN &H1000

#define CBS_SIMPLE &H1
#define CBS_DROPDOWN &H2

#define LB_ADDFILE &H196
#define LB_ADDSTRING &H180
#define LB_DELETETESTRING &H182
#define LB_DIR &H18D
#define LB_ERR (-1)
#define LB_FINDSTRING &H18F
#define LB_FINDSTRINGEXACT &H1A2
#define LB_GETANCHORINDEX &H9D
```

' フォームの枠線がある
' 親ウィンドウを持つコントロール (子ウィンドウ) を作成
' 兄弟関係にある子ウィンドウをクリップする
' ウィンドウが盛り上がった縁の境界線を持つように指定
' WS_EX_WINDOWEDGEとWS_EX_CLIENTEDGEの組み合わせ
' 可視状態のウィンドウを作成する
' 水平スクロールバーを持つウィンドウを作成する
' 垂直スクロールバーを持つウィンドウを作成する
' 縁が沈んで見える境界線を持つウィンドウを指定
' サイズ変更境界を持つウィンドウを作成する

' プッシュボタン
' チェックボックス
' ラジオボタン
' グループボックス

' 入力範囲を越えたら自動的に水平スクロールする
' 最後の行で自動的に垂直スクロールをする
' テキストを水平方向で中央に表示する
' テキストを左揃えする (デフォルト)
' 入力文字を小文字にする
' 複数行エディットを作成する
' フォーカスを失っても選択範囲を表示する
' 数値入力専用にする
' 入力文字をOEM文字セットに変換する
' 入力文字をデフォルトでは*にして表示する
' 読みとり専用にする (選択してコピーはできる)
' テキストを右揃えする
' 入力文字を大文字にする
' 複数行エディットでEnterキーで改行する

' 単純なコンボボックス。リストは常にドロップダウン
' CBS_SIMPLEで、リストはドロップダウンアイコンで表示

' リストボックス内のディレクトリリストにファイルを追加
' リストボックスに文字列を追加する
' リストボックス中の文字列を削除する
' リストボックスにファイル名を追加する
' エラー
' 前方部分検索
' リストボックスでプリフィックス文字を探す
' リストボックスのアンカー項目を取得する

```

#define LB_GETCARETINDEX &H19E
#define LB_GETCOUNT &H18B
#define LB_GETCURSEL &H188
#define LB_GETHORIZONTALEXTENT &H193
#define LB_GETITEMDATA &H199
#define LB_GETITEMHEIGHT &H1A1
#define LB_GETITEMRECT &H198
#define LB_GETLOCALE &H1A6
#define LB_GETSEL &H187
#define LB_GETSELCOUNT &H190
#define LB_GETSELITEMS &H191
#define LB_GETTEXT &H189
#define LB_GETTEXTLEN &H18A
#define LB_GETTOPINDEX &H18E

#define LB_INSERTSTRING &H181
#define LB_ITEMFROMPOINT &H1A9
#define LB_RESETCONTENT &H184
#define LB_SELECTSTRING &H18C
#define LB_SELITEMRANGE &H19B
#define LB_SETANCHORINDEX &H19C
#define LB_SETCARETINDEX &H19E
#define LB_SETCOLUMNWIDTH &H195
#define LB_SETCOUNT &H1A7
#define LB_SETCURSEL &H186
#define LB_SETHORIZONTALEXTENT &H194
#define LB_SETITEMDATA &H19A
#define LB_SETITEMHEIGHT &H1A0
#define LB_SETLOCALE &H1A5
#define LB_SETSEL &H185
#define LB_SETTABSTOPS &H192
#define LB_SETTOPINDEX &H197
#define LBS_DISABLENOSCROLL &H1000

#define LBS_EXTENDEDSEL &H800

#define LBS_HASSTRINGS &H40

#define LBS_MULTICOLUMN &H200

#define LBS_MULTIPLESEL &H8

#define LBS_NODATA &H2000

#define LBS_NOINTEGRALHEIGHT &H100

#define LBS_NOREDRAW &H4

#define LBS_NOSEL &H4000L

#define LBS_NOTIFY &H1

#define LBS_OWNERDRAWFIXED &H10

#define LBS_OWNERDRAWVARIABLE &H20

#define LBS_SORT &H2
#define LBS_STANDARD &HA00003

#define LBS_USETABSTOPS &H80

#define LBS_WANTKEYBOARDINPUT &H400

Var Shared Text1 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

'長方形を持つリストボックス項目のインデックスを取得
 'リストボックスの項目数を取得する
 'リストボックス内で選択された項目のインデックスを取得
 'リストボックスの水平方向範囲を取得する
 'リストボックスの項目に関連付けられている値を取得する
 'リストボックス内の項目の高さを取得する
 '項目を囲む境界長方形を取得する
 '現在のリストボックスのロケールを取得する
 '項目の選択状態を取得する
 'リストボックスで選択されている項目の数を取得する
 '選択されたリストボックス項目の項目番号をリストする
 'リストボックスから文字列を取得する
 'リストボックス内の文字列の長さを取得する
 'リストボックスの最初に表示されている(可視属性の)項目のインデックスを取得する
 '文字列をリストボックスに挿入する
 '特定位置の再近接項目の取得
 'リストボックスからすべての項目を除去する
 'リストボックス内の文字列を選択する
 'リストボックス内の連続する項目を選択する
 'リストボックスにアンカー項目を設定する
 'リストボックス内にフォーカス長方形を設定する
 'リストボックス内の列の幅を設定する
 'リストボックスがデータを持たないときの項目数を設定
 'リストボックス内の文字列を選択する
 'リストボックスの水平範囲を設定する
 'リストボックスの項目に値に関連付ける
 'リストボックス項目の高さを設定
 '現在のリストボックスのロケールを設定する
 '複数選択リストボックス内の文字列を選択する
 'リストボックス内のタブストップを設定する
 'リストボックスの項目が必ず表示状態になるようにする
 'スクロールするほどの項目数がない場合、リストボックスの垂直方向のスクロールバーが無効状態で表示
 '[Shift]キーとマウス、または特殊なキーの組み合わせで、複数のアイテムを選択できるようにする
 'オーナードローリストボックスのアイテムが文字列であることを指定
 '水平にスクロールする複数列のリストボックスを指定します。LB_SETCOLUMNWIDTH メッセージで列の幅を設定
 '項目をクリックまたはダブルクリックして、複数の項目を選択できるようにする
 'データを持たないリストボックスを指定。リストボックス内の項目数が1000を超えると、このスタイルを指定
 'リストボックスのサイズを指定します。デフォルトでは、アイテムが途中で切れないようにリストボックスのサイズ
 'アイテムが変更されてもリストボックスの外観を更新しないよう指定
 '表示するだけで選択できないアイテムを持つリストボックスを指定
 'リストボックス内の文字列をクリック(またはダブルクリック)するたびに、親ウィンドウに入力メッセージを通知
 '各アイテムの高さがすべて同じ、オーナードローリストボックスを作成
 '各アイテムの高さがすべて可変の、オーナードローリストボックスを作成
 'リストボックス内の文字列を自動的にソート
 '(LBS_NOTIFY Or LBS_SORT Or WS_VSCROLL Or WS_BORDER) スタイルの組み合わせ
 'リストボックスが文字列を描画するときにタブ文字を認識し表示できるようにする
 'リストボックスが入力フォーカスを持っているとき、ユーザーがキーを押すと、リストボックスのオーナーがWM


```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hLB As Long
    Var i As Integer
    Var Ret As Long

    'リストボックスを作成 (ソートなし)
    hLB = Api_CreateWindowEx(WS_EX_CLIENTEDGE, "ListBox", "", WS_CHILD Or WS_VISIBLE Or
    LBS_NOTIFY Or WS_VSCROLL Or WS_BORDER, 10, 10, 210, 70, GethWnd, 0, 0, 0)

    'リストボックスを作成 (ソートあり)
    'hLB = Api_CreateWindowEx(WS_EX_CLIENTEDGE, "ListBox", "", WS_CHILD Or WS_VISIBLE Or
    LBS_STANDARD, 10, 10, 210, 70, GethWnd, 0, 0, 0)

    '項目を設定
    For i = 1 To 20
        Ret = Api_SendMessage(hLB, LB_ADDSTRING, 0, Str$(i))
    Next

    '項目数を取得
    Ret = Api_SendMessage(hLB, LB_GETCOUNT, 0, 0)
    Text1.SetWindowText "項目数:" & Trim$(Str$(Ret))
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックスを範囲指定して選択する

SendMessage ウィンドウにメッセージを送信

LB_SELITEMRANGEEX (&H183) リストボックス内の連続する項目を選択する

LB_SETTOPINDEX (&H197) リストボックスの項目が必ず表示状態になるようにする

LB_ERR (-1) エラー



```

'=====
'= リストボックスを範囲指定して選択する
'= (LB_SELITEMRANGEEX.bas)
'=====
#include "Windows.bi"

' ウィンドウにメッセージを送信
Declare Function Api_SendMessage& Lib "user32" Alias "SendMessageA" (ByVal hWnd&, ByVal
wMsg&, ByVal wParam&, lParam As Any)

#define LB_SELITEMRANGEEX &H183
#define LB_SETTOPINDEX &H197

```

'リストボックス内の連続する項目を選択する
'リストボックスの項目が必ず表示状態になるようにする


```

#define LB_ERR (-1)                                'エラー

Var Shared Text(1) As Object
Var Shared Edit(1) As Object
Var Shared List1 As Object
Var Shared Button1 As Object

For i = 0 To 1
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1))) : Text(i).SetFontSize 14
    Edit(i).Attach GetDlgItem("Edit" & Trim$(Str$(i + 1))) : Edit(i).SetFontSize 14
Next
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14 : List1.SetWindowSize 122, 118
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var Count As Long
    Var AddItem As String

    SetMousePointer 2

    'コントロールを非表示
    List1.ShowWindow 0

    'リストボックスに項目を追加
    For Count = 0 To (40000 - 1)
        AddItem = "Item " & Str$(Count)
        List1.AddString AddItem
    Next Count

    'コントロールを表示
    List1.ShowWindow -1
    SetMousePointer 0
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var StartIndex As Long
    Var EndIndex As Long
    Var Ret As Long

    '選択範囲を指定
    StartIndex = Val(Edit(0).GetWindowText)
    EndIndex = Val(Edit(1).GetWindowText)

    'リストボックスの指定範囲を選択状態に設定
    Ret = Api_SendMessage(List1.GethWnd, LB_SELITEMRANGEEX, StartIndex, ByVal EndIndex)

    '選択できたときは
    If Ret <> LB_ERR Then

        '選択開始項目を先頭に表示
        Ret = Api_SendMessage(List1.GethWnd, LB_SETTOPINDEX, StartIndex, ByVal 0)
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リストボックスをフローティング表示

リストボックスをマウスで移動させることができます。

SetWindowLong ウィンドウのスタイルなどの情報を設定

GetWindowLong ウィンドウのスタイルなどの情報を取得

SetWindowText ウィンドウのタイトルを変更



```
'=====
'= リストボックスをフローティング表示
'= (FloatListBox.bas)
'=====
#include "Windows.bi"

'ウィンドウのスタイルなどの情報を設定
Declare Function Api_SetWindowLong& Lib "user32" Alias "SetWindowLongA" (ByVal hWnd&,
ByVal nIndex&, ByVal dwNewLong&)

'ウィンドウのスタイルなどの情報を取得
Declare Function Api_GetWindowLong& Lib "user32" Alias "GetWindowLongA" (ByVal hWnd&,
ByVal nIndex&)

'ウィンドウのタイトルを変更
Declare Function Api_SetWindowText& Lib "user32" Alias "SetWindowTextA" (ByVal hWnd&,
ByVal lpstring$)

#define WS_CAPTION &HC00000 'WS_BORDER Or WS_DLGFRAME
#define WS_THICKFRAME &H40000 'サイズ変更境界を持つウィンドウを作成する
#define WS_EX_TOOLWINDOW &H80 'ツールウィンドウを作成
#define GWL_STYLE (-16) 'ウィンドウスタイルを取得
Var Shared List1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var Ret As Long
    Var LB_STYLE As Long

    LB_STYLE = Api_GetWindowLong(List1.GethWnd, GWL_STYLE)
    LB_STYLE = LB_STYLE Or WS_EX_TOOLWINDOW
    Ret = Api_SetWindowLong(List1.GethWnd, GWL_STYLE, LB_STYLE)

    LB_STYLE = Api_GetWindowLong(List1.GethWnd, GWL_STYLE)
    LB_STYLE = LB_STYLE Or WS_CAPTION Or WS_THICKFRAME
    Ret = Api_SetWindowLong(List1.GethWnd, GWL_STYLE, LB_STYLE)

    Ret = Api_SetWindowText(List1.GethWnd, "拡張ListBox")
    List1.ResetContent
    For i = 0 To 19
        List1.AddString Trim$(Str$(i))
    Next
End Sub
```

```

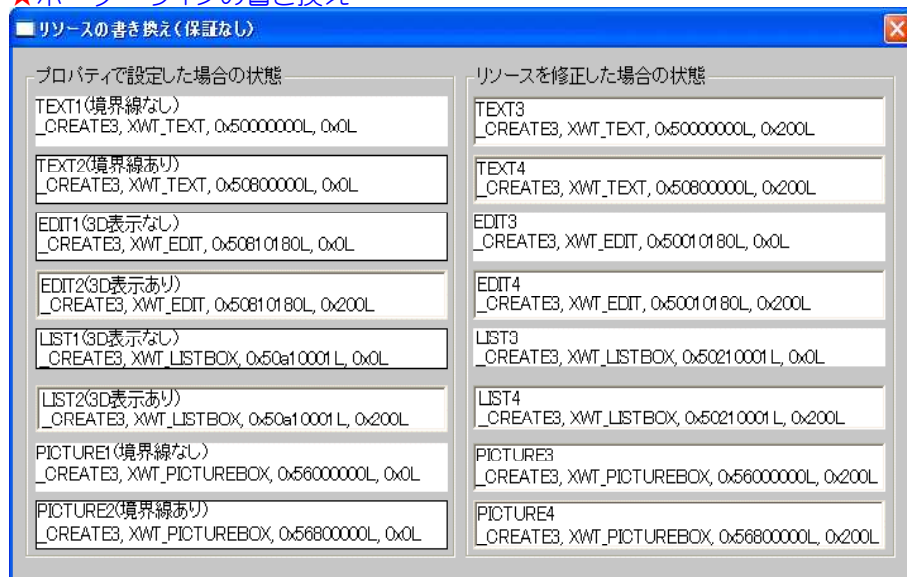
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

リソースの直接編集

プロパティで設定できないところは、リソースを直接編集します。(保証はいたしません！自己責任で..)

★ボーダーラインの書き換え



3D表示時、ボーダーラインの差異
コントロールにより、プロパティに境界線有無の設定項目がある場合と無い場合では、見た目でのサイズとx, y座標位置が異なるので注意が必要です。



★コントロールのテキスト色 (BGRの順であることに注意)

```

__CTRLTEXTCOLOR, 0x0L 黒
__CTRLTEXTCOLOR, 0xfffffl 白

```

★コントロールの背景色 (BGRの順であることに注意)

```

__CTRLTEXTCOLOR, 0x0L 黒
__CTRLTEXTCOLOR, 0xfffffl 白

```

★バージョン情報の設定

下記のとおり// File version info ===と// 'MAINFORM' form data ===の間に茶色の部分を挿入します(赤色が個別に入力する部分です)。
2行目のFILEVERSIONはWindows2K、WindowsXPでの表示用、17行目の"FileVersion"は、Windows9xにおける表示用

```

#include "fla0form.h"

FB$VersionInfo RCDATA
BEGIN
    // RCDATAITEMHEADER
    "FB", 0

```

```

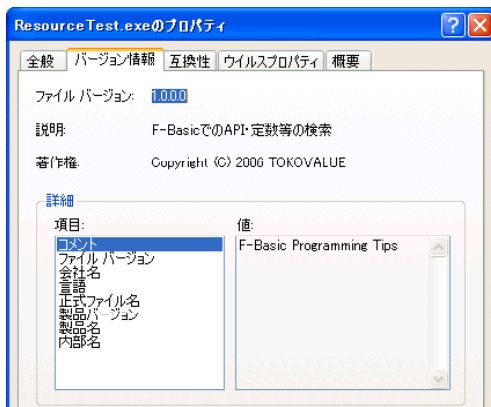
        0x6, 0x10
End

// File version info ===

1 VERSIONINFO
FILEVERSION      1,0,0,0
PRODUCTVERSION  1,0,0,0
FILEFLAGSMASK    0x3fL
FILEFLAGS        0x0L
FILEOS           0x40004L
FILETYPE         0x1L
FILESUBTYPE      0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "041103a4"
        BEGIN
            VALUE "Comments",          "F-Basic Programming Tips¥0"
            VALUE "CompanyName",       "xxxxxxx Software Inc.¥0"
            VALUE "FileDescription",   "F-BasicでのAPI・定数等の検索¥0"
            VALUE "FileVersion",       "1, 0, 0, 0¥0"
            VALUE "InternalName",      "Tokosvalue¥0"
            VALUE "LegalCopyright",    "Copyright (C) 2006 TOKOVALUE¥0"
            VALUE "OriginalFilename",  "Tokosvalue.exe¥0"
            VALUE "ProductName",       "Tokosvalue.exe Version 1.0.0.0¥0"
            VALUE "ProductVersion",    "1, 0, 0, 0¥0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x411, 932
    END
END

// 'MAINFORM' form data ===

```



領域内の色を反転する

領域内の色を反転します。

CreateEllipticRgn 楕円形のリージョンを作成

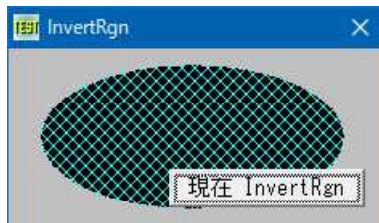
InvertRgn 指定されたリージョン領域の色を反転

CreateHatchBrush ハッチパターンの論理ブラシを作成

FillRgn 指定のブラシで領域を塗りつぶす

DeleteObject 論理オブジェクトを削除

例では、起動時ハッチブラシで塗り潰した楕円形を表示させ、ボタンクリックでその領域の色を反転します。



```
'=====
'= 領域内の色を反転する
'= (InvertRgn.bas)
'=====
#include "Windows.bi"

' 楕円形のリージョンを作成
Declare Function Api_CreateEllipticRgn& Lib "gdi32" Alias "CreateEllipticRgn" (ByVal
nLeftRect&, ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&)

' 指定されたリージョン領域の色を反転
Declare Function Api_InvertRgn& Lib "gdi32" Alias "InvertRgn" (ByVal hDC&, ByVal hRgn&)

' ハッチパターンの論理ブラシを作成
Declare Function Api_CreateHatchBrush& Lib "gdi32" Alias "CreateHatchBrush" (ByVal
nIndex&, ByVal crColor&)

' 指定のブラシで領域を塗りつぶす
Declare Function Api_FillRgn& Lib "gdi32" Alias "FillRgn" (ByVal hDC&, ByVal hRgn&, ByVal
hBrush&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

#define HS_DIAGCROSS 5                                '45度のクロスハッチ

Var Shared Button1 As Object
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared hDC As Long
Var Shared FG As Integer

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var hRgn As Long
    Var hBr As Long
    Var Ret As Long

    FG = 0
    Button1.SetWindowText "現在 Normal"

    hDC = Api_GetDC(GetHwnd)
    hRgn = Api_CreateEllipticRgn(20, 10, 210, 100)
    hBr = Api_CreateHatchBrush(HS_DIAGCROSS, RGB(255, 0, 0))
    Ret = Api_FillRgn(hDC, hRgn, hBr)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
```

```

Sub Button1_on()
    Var hRgn As Long
    Var Ret As Long

    FG = not FG
    If FG = 0 Then
        Button1.SetWindowText "現在 Normal"
    Else
        Button1.SetWindowText "現在 InvertRgn"
    End If

    hRgn = Api_CreateEllipticRgn(20, 10, 210, 100)
    Ret = Api_InvertRgn(hDC, hRgn)
    Ret = Api_DeleteObject(hRgn)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

    Ret = Api_ReleaseDC(GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

領域内の再描画

指定領域内を再描画します。

CreateBrushIndirect LOGBRUSH構造体を定義して論理ブラシを作成

CreateRectRgn 矩形の領域を作成

IsWindow ウィンドウハンドルが有効か調べる

SetRectEmpty 空の長方形を作成

SetRect 長方形を設定

IsRectEmpty 矩形の高さ・幅が0かどうかを判定

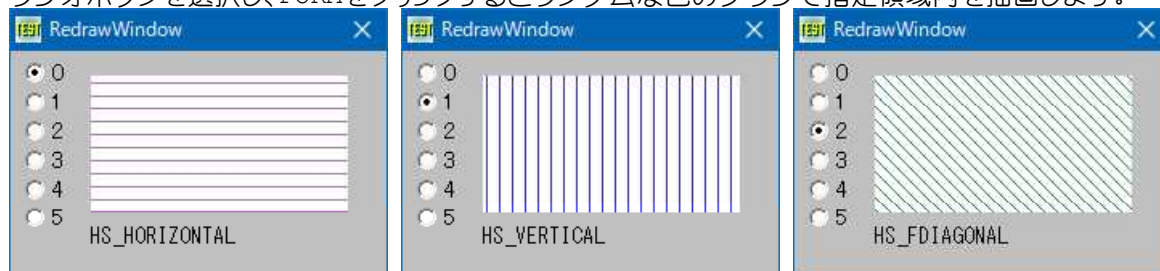
IntersectRect 2つの長方形の交差長方形の座標を取得

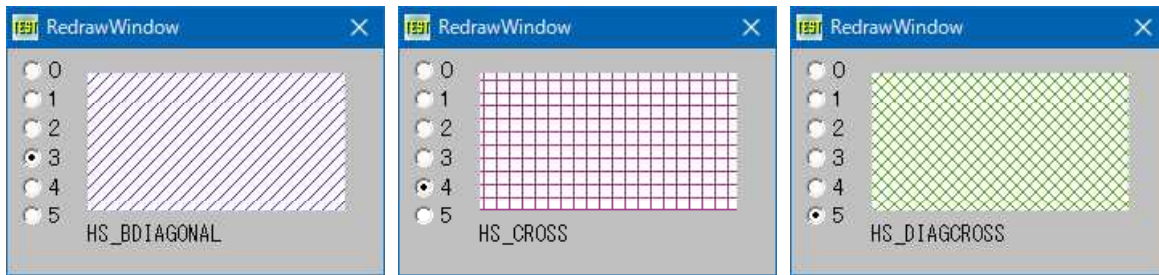
RedrawWindow 指定の条件でクライアント領域内の指定の範囲を再描画

GetRgnBox リージョンの境界長方形の取得

FillRect グラフィックスオブジェクトを削除し、システムリソースを解放

ラジオボタンを選択し、FORMをクリックするとランダムな色のブラシで指定領域内を描画します。





```
'=====
'= 領域内の再描画
'= (RedrawWindow.bas)
'=====

#include "Windows.bi"

#define RDW_INVALIDATE &H1                                '再描画領域を無効化

#define BS_SOLID 0                                         'ソリッドブラシ
#define BS_HOLLOW 1                                       '塗りつぶさない
#define BS_NULL 1                                         'BS_HOLLOW
#define BS_HATCHED 2                                       'ハッチング (スタイルはlbHatchで指定)
#define BS_PATHTERN 3                                     'ビットマップパターン (lpHatchにビットマップのハンドルを指定)

#define BS_DIBPATTERN 5                                    'DIBパターン (lpHatchにDIBのハンドルを指定)
#define BS_DIBPATTERNPT 6                                 'DIBパターン (lpHatchにDIBのハンドルを指定)
#define BS_PATTERN8X8 7                                   'BS_PATTERN
#define BS_DIBPATTERN8X8 8                               'BS_DIBPATTERN

#define HS_HORIZONTAL 0                                    '横線
#define HS_VERTICAL 1                                    '縦線
#define HS_FDIAGONAL 2                                    '斜線 (左下-右上)
#define HS_BDIAGONAL 3                                    '斜線 (左上-右下)
#define HS_CROSS 4                                        '縦横線のクロス
#define HS_DIAGCROSS 5                                    '斜めクロス

Type LOGBRUSH
    lbStyle As Long
    lbColor As Long
    lbHatch As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

' LOGBRUSH構造体を定義して論理ブラシを作成
Declare Function Api_CreateBrushIndirect& Lib "gdi32" Alias "CreateBrushIndirect"
(lpLogBrush As LOGBRUSH)

' 矩形の領域を作成
Declare Function Api_CreateRectRgn& Lib "gdi32" Alias "CreateRectRgn" (ByVal X1&, ByVal
Y1&, ByVal X2&, ByVal Y2&)

' ウィンドウハンドルが有効か調べる
Declare Function Api_IsWindow& Lib "user32" Alias "IsWindow" (ByVal hWnd&)

' 空の長方形を作成
Declare Function Api_SetRectEmpty& Lib "user32" Alias "SetRectEmpty" (lpRect As RECT)

' 長方形を設定
Declare Function Api_SetRect& Lib "user32" Alias "SetRect" (lpRect As RECT, ByVal X1&,
ByVal Y1&, ByVal X2&, ByVal Y2&)

' 矩形の高さ・幅が0かどうかを判定
Declare Function Api_IsRectEmpty& Lib "user32" Alias "IsRectEmpty" (lpRect As RECT)
```

' 2つの長方形の交差長方形の座標を取得

```
Declare Function Api_IntersectRect& Lib "user32" Alias "IntersectRect" (lpDestRect As RECT, lpSrc1Rect As RECT, lpSrc2Rect As RECT)
```

' 指定の条件でクライアント領域内の指定の範囲を再描画

```
Declare Function Api_RedrawWindow& Lib "user32" Alias "RedrawWindow" (ByVal hWnd&, lpRectUpdate As Any, ByVal hrgnUpdate&, ByVal fuRedraw&)
```

' リージョンの境界長方形の取得

```
Declare Function Api_GetRgnBox& Lib "gdi32" Alias "GetRgnBox" (ByVal hRgn&, lpRect As RECT)
```

' グラフィックスオブジェクトを削除し、システムリソースを解放

```
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

' 長方形の内部を再描画

```
Declare Function Api_FillRect& Lib "user32" Alias "FillRect" (ByVal hdc&, lpRect As RECT, ByVal hBrush&)
```

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hdc&)
```

```
Var Shared Text1 As Object
```

```
Var Shared Radio(5) As Object
```

```
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
```

```
For i = 0 To 5
```

```
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1)))
```

```
    Radio(i).SetFontSize 14
```

```
Next
```

```
' =====
```

```
' =
```

```
' =====
```

```
Declare Function HS bdecl () As Integer
```

```
Function HS()
```

```
    HS = Val(Mid$(GetDlgItemRadioSelect("Radio1"), 6)) - 1
```

```
End Function
```

```
' =====
```

```
' =
```

```
' =====
```

```
Declare Sub Mainform_Click edec1 ()
```

```
Sub Mainform_Click()
```

```
    Var lb As LOGBRUSH
```

```
    Var rc As RECT
```

```
    Var rrc As RECT
```

```
    Var Rgn As Long
```

```
    Var hBrush As Long
```

```
    Var hdc As Long
```

```
    Var strHS As String
```

```
    Var Ret As Long
```

' 再描画

```
Ret = Api_RedrawWindow(GethWnd, ByVal 0, ByVal 0, RDW_INVALIDATE)
```

```
Cls
```

```
Randomize Time
```

```
lb.lbColor = RGB(Int(Rnd(1) * 255), Int(Rnd(1) * 255), Int(Rnd(1) * 255))
```

```
lb.lbStyle = BS_HATCHED
```

```
lb.lbHatch = HS
```

```
hdc = Api_GetDC(GethWnd)
```

' ブラシ設定

```
hBrush = Api_CreateBrushIndirect(lb)
```

' 長方形を設定

```
Ret = Api_SetRect(rc, 0, 0, 240, 140)
```



```

'長方形領域を設定
Rgn = Api_CreateRectRgn(50, 15, 210, 100)

'領域ボックス取得
Ret = Api_GetRgnBox(Rgn, rrc)

'2つの長方形の交差を計算
Ret = Api_IntersectRect(rc, rrc, rc)

'長方形を空に
Ret = Api_SetRectEmpty(rrc)

'長方形にブラシを設定
Ret = Api_FillRect(hDC, rc, hBrush)

'ブラシを削除
Ret = Api_DeleteObject(hBrush)

'長方形が空であることをチェック
If Api_IsRectEmpty(rrc) <> 0 Then
    Ret = Api_SetRectEmpty(rrc)
End If

Select Case HS
    Case 0
        strHS = "HS_HORIZONTAL"
    Case 1
        strHS = "HS_VERTICAL"
    Case 2
        strHS = "HS_FDIAGONAL"
    Case 3
        strHS = "HS_BDIAGONAL"
    Case 4
        strHS = "HS_CROSS"
    Case 5
        strHS = "HS_DIAGCROSS"
End Select

Text1.SetWindowText strHS

Ret = Api_ReleaseDC(GethWnd, hDC)
End Sub
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

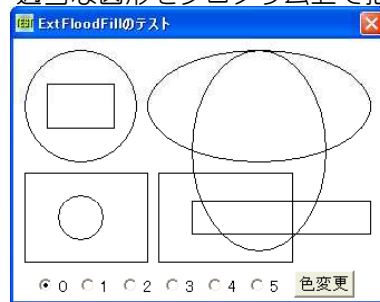
領域の塗り潰し(1)

指定領域内を塗り潰すテスト
ExtFloodFill 塗り潰しの指定
FloodFill 塗り潰しの指定 (Win32アプリケーションでは、ExtFloodFill関数を使う)
CreateSolidBrush 塗り潰し指定
CreateHatchBrush ハッチ指定
SelectObject グラフィックオブジェクトをデバイスコンテキストと関連付ける
DeleteObject オブジェクトの削除
GetDC フォームのデバイスコンテキスト取得
ReleaseDC フォームのデバイスコンテキスト解放

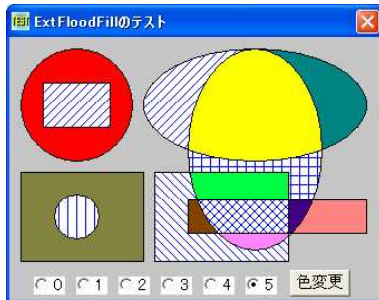
フォームにRadioButtonを6個、Buttonを1個貼り付けます。Radio1～Radio5のイベントは全てRadioOp1_ONとします。



適当な図形をプログラム上で描画します。



マウス左ボタンをクリックすると、色設定ダイアログボックスで指定した色で塗り潰し、右ボタンをクリックするとラジオボタンで選択した種類のハッチ(青)で塗り潰しを実行します。



```
'=====
'= 塗り潰しのテスト
'= (ExtFloodFill.bas)
'=====
#include "Windows.bi"
```

' 現在選択されているブラシで一定の範囲内を塗りつぶす

```
Declare Function Api_ExtFloodFill Lib "gdi32" Alias "ExtFloodFill" (ByVal hdc&, ByVal
X&, ByVal Y&, ByVal crColor&, ByVal wFillType&)
```

' 画面の領域を現在のブラシで塗りつぶす (Win32 アプリケーションでは、ExtFloodFill 関数を使う)

```
Declare Function Api_FloodFill Lib "gdi32" Alias "FloodFill" (ByVal hdc&, ByVal x&,
ByVal y&, ByVal crColor&)
```

' ソリッドカラーで論理ブラシを作成

```
Declare Function Api_CreateSolidBrush Lib "gdi32" Alias "CreateSolidBrush" (ByVal
crColor&)
```

' ハッチパターンの論理ブラシを作成

```
Declare Function Api_CreateHatchBrush Lib "gdi32" Alias "CreateHatchBrush" (ByVal
nIndex&, ByVal crColor&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hdc&)
```

' 指定されたデバイスコンテキストのオブジェクトを選択

```
Declare Function Api_SelectObject Lib "gdi32" Alias "SelectObject" (ByVal hdc&, ByVal
hObject&)
```

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる

```
Declare Function Api_DeleteObject Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

```
#define FLOODFILLBORDER 0
```

'塗りつぶし領域がcrColorで指定された色に囲まれている範囲

```
#define FLOODFILLSURFACE 1
```

'crColorで指定された色の領域(指定色がある限り塗りつぶし)

```
#define HS_HORIZONTAL 0
```

'水平ハッチ

```

#define HS_VERTICAL 1
#define HS_FDIAGONAL 2
#define HS_BDIAGONAL 3
#define HS_CROSS 4
#define HS_DIAGCROSS 5

'垂直ハッチ
'45度下向きのハッチ (左から右へ)
'45度上向きのハッチ (左から右へ)
'水平と垂直クロスハッチ
'45度のクロスハッチ

Var Shared Flag As byte
Var Shared Hatch As byte
Var Shared rgbColor As Long

'=====
'=   とりあえず描画
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Circle(60, 60), 50, 0
    Circle(220, 60), 100, 0, .5
    Circle(220, 100), 60, 0, 1.5
    Circle(60, 160), 20, 0
    Line(30, 40)-(90, 80), Pset, 0, b
    Line(10, 120)-(120, 200), Pset, 0, b
    Line(130, 120)-(250, 200), Pset, 0, b
    Line(160, 145)-(320, 175), Pset, 0, b
    Hatch = HS_BDIAGONAL
    rgbColor = RGB(255, 0, 0)
End Sub

'=====
'=   ハッチ種類取得
'=====
Declare Sub RadioOpl_on edec1 ()
Sub RadioOpl_on()
    Var Radio As Object

    Radio.Attach GetFocus
    Select Case Radio.GetWindowText
        Case "0"
            Hatch = HS_HORIZONTAL
        Case "1"
            Hatch = HS_VERTICAL
        Case "2"
            Hatch = HS_FDIAGONAL
        Case "3"
            Hatch = HS_BDIAGONAL
        Case "4"
            Hatch = HS_CROSS
        Case "5"
            Hatch = HS_DIAGCROSS
    End Select
End Sub

'=====
'=   塗り潰し
'=====
Declare Sub MainForm_MouseDown edec1 (ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
Sub MainForm_MouseDown(ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
    Var Ret As Long
    Var hBr As Long
    Var hDC As Long

    hDC = Api_GetDC(GethWnd)
    'MainFormのデバイスコンテキスト取得

    If Button% = 1 Then
        hBr = Api_CreateSolidBrush(rgbColor) '塗りつぶし指定
    Else
        hBr = Api_CreateHatchBrush(Hatch, RGB(0, 0, 255)) 'ハッチ指定
    End If

    Ret = Api_SelectObject(hDC, hBr)
    Ret = Api_ExtFloodFill(hDC, SX, SY, 0, FLOODFILLBORDER) '塗りつぶし

```

```

' Ret = Api_FloodFill(hDC, SX, SY, FLOODFILLBORDER) '塗りつぶし

Ret = Api_DeleteObject(hBr) 'ブラシの解放
Ret = Api_ReleaseDC(hBr, hDC) 'デバイスコンテキストの解放
End Sub

'=====
'= 塗り潰し色変更
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()

    If ChooseColor(rgbColor) Then
        SetFillColor rgbColor
    End If
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

領域内の塗り潰し(II)

RoundRectで指定した領域内を塗り潰します。

CreateSolidBrush 純色(ソリッドカラー)で論理ブラシを作成

DeleteObject グラフィックスオブジェクトを削除

ExtFloodFill 選択されているブラシで一定の範囲内を塗りつぶす

SelectObject デバイスコンテキストにオブジェクトを選択

RoundRect 角の丸い矩形を描画

GetSystemMetrics 表示要素の寸法とシステム構成の設定を取得

フォームにはサイズグリップを表示させています。

フォームをリサイズした場合、同じ比率で枠を作成しランダムな色で塗り潰します。

塗り潰し領域をフォームの中央にくるよう、メニュー・フレーム・ボーダー等の幅を取得計算しています。



```

'=====
'= 領域内の塗り潰し(II)
'= (CreateSolidBrush.bas)
'=====
#include "Windows.bi"

' 純色(ソリッドカラー)で論理ブラシを作成
Declare Function Api_CreateSolidBrush Lib "gdi32" Alias "CreateSolidBrush" (ByVal
crColor&)

```

' グラフィックスオブジェクトを削除

```
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)
```

' 選択されているブラシで一定の範囲内を塗りつぶす

```
Declare Function Api_ExtFloodFill& Lib "gdi32" Alias "ExtFloodFill" (ByVal hDC&, ByVal nXStart&, ByVal nYStart&, ByVal crColor&, ByVal fuFillType&)
```

' デバイスコンテキストにオブジェクトを選択

```
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal hObject&)
```

' 角の丸い矩形を描画

```
Declare Function Api_RoundRect& Lib "gdi32" Alias "RoundRect" (ByVal hDC&, ByVal nLeftRect&, ByVal nTopRect&, ByVal nRightRect&, ByVal nBottomRect&, ByVal nWidth&, ByVal nHeight&)
```

' 表示要素の寸法とシステム構成の設定を取得

```
Declare Function Api_GetSystemMetrics& Lib "user32" Alias "GetSystemMetrics" (ByVal nIndex&)
```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)
```

' デバイスコンテキストを解放

```
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)
```

```
#define FLOODFILLBORDER 0  
#define FLOODFILLSURFACE 1
```

```
#define SM_CXFRAME 32  
#define SM_CYFRAME 33  
#define SM_CYSIZE 31  
#define SM_CYBORDER 6
```

' サイズ可変ウィンドウの境界線の、x方向の幅
' " Y方向の幅
' タイトルバー内のビットマップの高さ
' サイズ固定ウィンドウの境界線のy方向の幅

```
Var Shared Text1 As Object
```

```
Var Shared zX As Integer
```

```
Var Shared zY As Integer
```

```
' =====  
' =  
' =====
```

```
Declare Sub MainForm_Start edec1 ()
```

```
Sub MainForm_Start()
```

```
    Var fs As Integer
```

```
        fs = 14
```

```
        Text1.Attach GetDlgItem("Text1")
```

```
        Text1.SetFontSize fs
```

```
        Text1.SetWindowSize fs, fs
```

```
        Text1.SetFontName "Marlett"
```

```
        Text1.SetWindowText "o"
```

```
        ShowWindow -1
```

```
        zX = Api_GetSystemMetrics(SM_CXFRAME) * 2
```

```
        zY = Api_GetSystemMetrics(SM_CYFRAME) * 2 + Api_GetSystemMetrics(SM_CYBORDER) +
```

```
        Api_GetSystemMetrics(SM_CYSIZE)
```

```
End Sub
```

```
' =====  
' =  
' =====
```

```
Declare Sub MainForm_Resize edec1 ()
```

```
Sub MainForm_Resize()
```

```
    Var hBrush As Long
```

```
    Var hOldBrush As Long
```

```
    Var RndCol As Long
```

```
    Var hDC As Long
```

```
    Var Ret As Long
```

```

on error goto *Er_Trap

' サイズグリッド表示
Text1.MoveWindow (GetWidth - Text1.GetWidth) - zX, GetHeight - Text1.GetHeight - zY

cls

hDC = Api_GetDC (GethWnd)
' Left Top Right Bottom Corner
Ret = Api_RoundRect (hDC, (GetWidth - zX) / 5, (GetHeight - zY) / 5, (GetWidth - zX) / 5 *
4, (GetHeight - zY) / 5 * 4, 20, 20)

' ランダムなブラシ作成
Randomize time
RndCol = rgb (int (rnd (1) * 255 + 1), int (rnd (1) * 255 + 1), int (rnd (1) * 255 + 1))
hBrush = Api_CreateSolidBrush (RndCol)

hOldBrush = Api_SelectObject (hDC, hBrush)

' RoundRect内を塗り潰し
Ret = Api_ExtFloodFill (hDC, (GetWidth - zX) / 2, (GetHeight - zY) / 2, rgb (0, 0, 0),
FLOODFILLBORDER)

Ret = Api_SelectObject (hDC, hOldBrush)

Ret = Api_DeleteObject (hBrush)
Ret = Api_ReleaseDC (GethWnd, hDC)
Exit Sub

*Er_Trap
resume Next
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

領域内の塗り潰し (Ⅲ)

指定領域内を塗り潰すテスト

FloodFill 塗り潰しの指定 (Win32アプリケーションでは、ExtFloodFill関数を使う)

CreateSolidBrush 塗り潰し指定

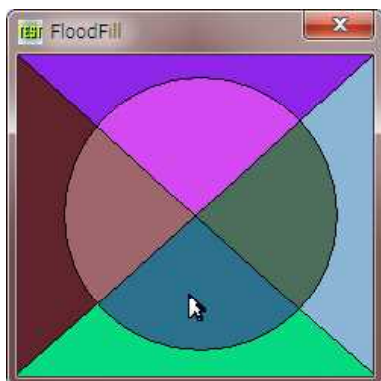
CreateHatchBrush ハッチ指定

SelectObject グラフィックオブジェクトをデバイスコンテキストと関連付ける

DeleteObject オブジェクトの削除

GetDC フォームのデバイスコンテキスト取得

ReleaseDC フォームのデバイスコンテキスト解放



```

'=====
'= 領域の塗り潰し
'= (FloodFill.bas)
'= ExtFloodFill.bas参照
'=====
#include "Windows.bi"

#define FLOODFILLBORDER 0 '塗りつぶし領域がcrColorで指定された色に囲まれている範囲
#define FLOODFILLSURFACE 1 'crColorで指定された色の領域(指定色がある限り塗りつぶし)
' 画面の領域を現在のブラシで塗りつぶす (Win32 アプリケーションでは、ExtFloodFill 関数を使う)
Declare Function Api_FloodFill& Lib "gdi32" Alias "FloodFill" (ByVal hDC&, ByVal x&,
ByVal y&, ByVal crColor&)

' ソリッドカラーで論理ブラシを作成
Declare Function Api_CreateSolidBrush& Lib "gdi32" Alias "CreateSolidBrush" (ByVal
crColor&)

' 指定されたデバイスコンテキストのオブジェクトを選択
Declare Function Api_SelectObject& Lib "gdi32" Alias "SelectObject" (ByVal hDC&, ByVal
hObject&)

' ペン、ブラシ、フォント、ビットマップ、リージョン、パレットのいずれかの論理オブジェクトを削除し、そのオブジェクトに
関連付けられていたすべてのシステムリソースを解放。オブジェクトを削除した後は、指定されたハンドルは無効になる
Declare Function Api_DeleteObject& Lib "gdi32" Alias "DeleteObject" (ByVal hObject&)

' 楕円の描画
Declare Function Api_Ellipse& Lib "gdi32" Alias "Ellipse" (ByVal hDC&, ByVal X1&, ByVal
Y1&, ByVal X2&, ByVal Y2&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Circle(120, 104), 90, 0
    Line(0, GetHeight - 31)-(GetWidth - 7, 0),, 0
    Line(0, 0)-(GetWidth - 7, GetHeight - 31),, 0
End Sub

'=====
'= 塗り潰し
'=====
Declare Sub MainForm_MouseDown edec1 (ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
Sub MainForm_MouseDown(ByVal Button%, ByVal Shift%, ByVal SX!, ByVal SY!)
    Var hDC As Long
    Var hBr As Long
    Var rgbColor As Long
    Var Ret As Long

    Randomize (Time)

    rgbColor = RGB (Rnd(1) * 255, Rnd(1) * 255, Rnd(1) * 255)

    hDC = Api_GetDC (GethWnd)

    hBr = Api_CreateSolidBrush (rgbColor)

    Ret = Api_SelectObject (hDC, hBr)
    Ret = Api_FloodFill (hDC, SX!, SY!, FLOODFILLBORDER)

    ' ブラシの解放
    Ret = Api_DeleteObject (hBr)

```

```

' デバイスコンテキストの解放
Ret = Api_ReleaseDC (hBr, hDC)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

ルートパスかどうかを判断

PathIsRoot ルートパスかどうかを判断



```

' =====
' = ルートパスかどうかを判断
' = (PathIsRoot.bas/P)
' =====

' ルートパスかどうかを判断
Declare Function Api_PathIsRoot & Lib "shlwapi" Alias "PathIsRootA" (ByVal pszPath$)

Var MyPath As String
Var Ret As Long

MyPath = "C:¥"

Ret = Api_PathIsRoot (MyPath)

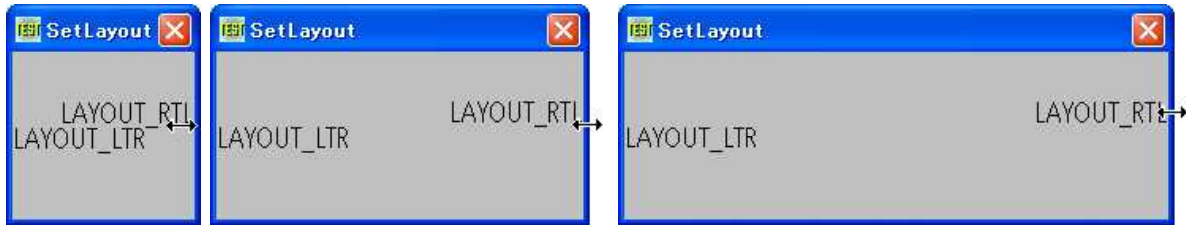
If Ret <> 0 Then
    Print MyPath & "はルートパスです！"
Else
    Print MyPath & "はルートパスではない！"
End If

Stop
End

```

レイアウトを変更

SetLayout デバイスコンテキスト (DC) のレイアウトを変更
GetLayout デバイスコンテキスト (DC) のレイアウトを返す
TextOut 文字を描画
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放
SetBkMode バックグラウンドの塗りつぶしモード設定



```
'=====
'= レイアウトを変更
'= (SetLayout.bas)
'=====
#include "Windows.bi"

' デバイスコンテキスト (DC) のレイアウトを変更
Declare Function Api_SetLayout& Lib "gdi32" Alias "SetLayout" (ByVal hDC&, ByVal
dwLayout&)

' デバイスコンテキスト (DC) のレイアウトを返す
Declare Function Api_GetLayout& Lib "gdi32" Alias "GetLayout" (ByVal hDC&)

' 文字を描画
Declare Function Api_TextOut& Lib "gdi32" Alias "TextOutA" (ByVal hDC&, ByVal nXStart&,
ByVal nYStart&, ByVal lpString$, ByVal cbString&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

' バックグラウンドの塗りつぶしモード設定
Declare Function Api_SetBkMode& Lib "gdi32" Alias "SetBkMode" (ByVal hDC&, ByVal
iBkMode&)

#define TRANSPARENT 1
#define LAYOUT_LTR &H2
#define LAYOUT_RTL &H1

'=====
'=
'=====
Declare Sub MainForm_Resize edec1 ()
Sub MainForm_Resize ()
    Var hDC As Long
    Var txt As String
    Var Ret As Long

    hDC = Api_GetDC (GethWnd)

    Ret = Api_SetBkMode (hDC, TRANSPARENT)

    txt = "LAYOUT_RTL"
    Ret = Api_SetLayout (hDC, LAYOUT_RTL)
    Ret = Api_TextOut (hDC, 0, 30, txt, Len(txt))

    txt = "LAYOUT_LTR"
    Ret = Api_SetLayout (hDC, LAYOUT_LTR)
    Ret = Api_TextOut (hDC, 0, 46, txt, Len(txt))

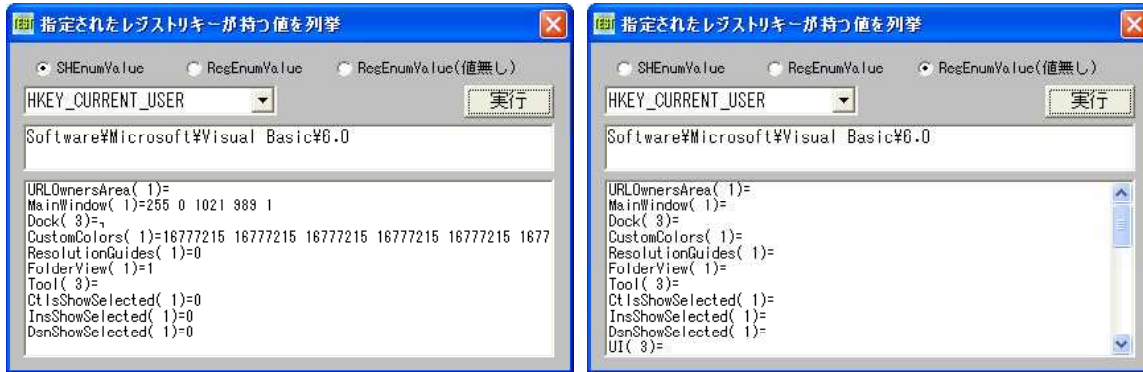
    Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
```

Stop
End

レジストリキーが持つ値を列挙

SHEnumValue 指定されたレジストリキーが持つ値を列挙
RegEnumValue 指定されたレジストリキーが持つ値を列挙
RegOpenKeyEx レジストリのキーのハンドルを確保
RegCloseKey レジストリのハンドルを解放



```
'=====
'= レジストリキーが持つ値を列挙
'= (SHEnumValue.bas)
'=====
#include "Windows.bi"
```

' 指定されたレジストリキーが持つ値を列挙

```
Declare Function Api_SHEnumValue& Lib "shlwapi" Alias "SHEnumValueA" (ByVal hKey&, ByVal dwIndex&, ByVal pszsValueName$, ByRef pcchValueName&, ByRef pdwType&, ByRef pvData As Any, ByRef pcdData&)
```

' 指定されたレジストリキーが持つ値を列挙

```
Declare Function Api_RegEnumValue& Lib "advapi32" Alias "RegEnumValueA" (ByVal hKey&, ByVal dwIndex&, ByVal lpsValueName$, ByRef lpcbValueName&, ByVal lpReserved&, ByRef lpType&, ByRef lpData As Any, ByRef lpcbData&)
```

' レジストリのキーのハンドルを確保

```
Declare Function Api_RegOpenKeyEx& Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&, ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)
```

' レジストリのハンドルを解放

```
Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)
```

```
#define HKEY_CLASSES_ROOT -2147483648
#define HKEY_CURRENT_CONFIG -2147483643
#define HKEY_CURRENT_USER -2147483647
#define HKEY_DYN_DATA -2147483642
#define HKEY_LOCAL_MACHINE -2147483646
#define HKEY_USERS -2147483645
```

```
#define REG_BINARY 3
#define REG_DWORD 4
#define REG_DWORD_BIG_ENDIAN 5
#define REG_DWORD_LITTLE_ENDIAN 4
```

```
#define REG_EXPAND_SZ 2
#define REG_LINK 6
#define REG_MULTI_SZ 7
#define REG_NONE 0
#define REG_QWORD 11
#define REG_QWORD_LITTLE_ENDIAN 11
```

' 拡張子に関する情報や、それらとアプリケーションとの関連づけに関する情報

' Display解像度/Font等の画面表示に関するカレントユーザーの情報、プリンターに関する情報

' 現在Windowsにログインしているユーザーの情報

' 動的に変化する様々な情報

' PCを利用するユーザーに共通の設定情報

' Windowsを利用するユーザー個別の情報

' バイナリデータ

' 32ビットの数値

' ビッグエンディアン形式の32ビット数値

' リトルエンディアン形式の32ビット数値 (REG_DWORDと同等)

' 展開前の環境変数への参照が入ったヌル終端文字列

' Unicodeシンボリックリンク

' ヌル終端文字列の配列

' 定義されていない型

' 64ビット数値

' リトルエンディアン形式の64ビット数値 (REG_QWORDと同等)

```

#define REG_RESOURCE_LIST 8
#define REG_SZ 1
#define ERROR_SUCCESS &H0

#define ERROR_NO_MORE_ITEMS 259

#define SYNCHRONIZE &H100000
#define STANDARD_RIGHTS_READ &H20000

#define KEY_QUERY_VALUE &H1
#define KEY_SET_VALUE &H2
#define KEY_CREATE_SUB_KEY &H4
#define KEY_ENUMERATE_SUB_KEYS &H8
#define KEY_NOTIFY &H10
#define KEY_CREATE_LINK &H20
#define KEY_READ ((STANDARD_RIGHTS_READ Or KEY_QUERY_VALUE Or KEY_ENUMERATE_SUB_KEYS Or
KEY_NOTIFY) And (Not SYNCHRONIZE))
#define KEY_ALL_ACCESS &H3F

Var Shared Radio(2) As Object
Var Shared List1 As Object
Var Shared Combol As Object
Var Shared Edit1 As Object
Var Shared Button1 As Object

For i = 0 To 2
    Radio(i).Attach GetDlgItem("Radio" & Trim$(Str$(i + 1)))
    Radio(i).SetFontSize 12
Next
List1.Attach GetDlgItem("List1") : List1.SetFontSize 12
Combol.Attach GetDlgItem("Combol") : Combol.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Function Index bdecl () As Integer
Function Index()
    Index = Val(Mid$(GetDlgRadioSelect("Radiol"), 6)) - 1
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edecl ()
Sub MainForm_Start()
    Combol.AddString "HKEY_CLASSES_ROOT"
    Combol.AddString "HKEY_CURRENT_USER"
    Combol.AddString "HKEY_LOCAL_MACHINE"
    Combol.AddString "HKEY_USERS"
    Combol.AddString "HKEY_CURRENT_CONFIG"

    Edit1.SetWindowText "Software¥Microsoft¥Visual Basic¥6.0"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edecl ()
Sub Button1_on()
    Var lKey As Long
    Var hKey As Long
    Var sValueName As String
    Var lValueName As Long
    Var lType As Long
    Var sData As String
    Var lData As Long
    Var Subkey As String

```

等)
'デバイスドライバのリソースリスト
'ヌル終端文字列
'正常終了の戻り値を示す
'
'
'(READ_CONTROL) オブジェクトのセキュリティ記述子の読み取り許可
'サブキーデータを問い合わせるためのアクセス権

```

Var Item As Long
Var i As Long
Var Ret As Long

Item = Combo1.GetCursel
If Item < 0 Then
    A% = MsgBox("", "[HKEY_]項目を選択してください!", 0, 2)
    Exit Sub
End If

Select Case Item
    Case 0
        hKey = HKEY_CLASSES_ROOT
    Case 1
        hKey = HKEY_CURRENT_USER
    Case 2
        hKey = HKEY_LOCAL_MACHINE
    Case 3
        hKey = HKEY_USERS
    Case 4
        hKey = HKEY_CURRENT_CONFIG
End Select

List1.Resetcontent

Subkey = Edit1.GetWindowText
Ret = Api_RegOpenKeyEx(hKey, Subkey, 0, KEY_ALL_ACCESS, lKey)
If Ret <> ERROR_SUCCESS Then Exit Sub

i = 0
Do
    lValueName = 1000
    sValueName = String$(lValueName, 0)
    lData = 2000
    sData = String$(lData, 0)
    lType = REG_SZ

    Select Case Index
        Case 0
            Ret = Api_SHEnumValue(lKey, i, sValueName, lValueName, lType, sData,
lData)
        Case 1
            Ret = Api_RegEnumValue(lKey, i, sValueName, lValueName, 0, lType, sData,
lData)
        Case 2
            Ret = Api_RegEnumValue(lKey, i, sValueName, lValueName, 0, lType, ByVal 0,
0)
    End Select

    If Ret <> ERROR_SUCCESS Then Exit Do
    sValueName = Left$(sValueName, lValueName)
    sData = Left$(sData, lData)
    List1.AddString sValueName & "(" & Str$(lType) & ")" & "=" & sData
    i = i + 1
Loop

Ret = Api_RegCloseKey(lKey)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリキーに関する情報を取得 (1)

RegOpenKeyEx レジストリのキーのハンドルを確保

RegQueryInfoKey 指定されたレジストリキーに関する情報を取得

RegCloseKey レジストリのハンドルを解放

FileTimeToSystemTime ファイルタイムをシステムタイムに変換



```
'=====
'= レジストリキーに関する情報を取得
'= (RegQueryInfoKey.bas)
'=====
#include "Windows.bi"

Type FILETIME
    dwLowDateTime    As Long
    dwHighDateTime   As Long
End Type

Type SYSTEMTIME
    wYear            As Integer
    wMonth           As Integer
    wDayOfWeek       As Integer
    wDay             As Integer
    wHour            As Integer
    wMinute          As Integer
    wSecond          As Integer
    wMilliseconds    As Integer
End Type

#define HKEY_LOCAL_MACHINE -2147483646 'PCを利用するユーザーに共通の設定情報
#define KEY_QUERY_VALUE &H1 'サブキーデータを問い合わせるためのアクセス権
#define ERROR_SUCCESS &H0 '正常終了の戻り値を示す

' レジストリのキーのハンドルを確保
Declare Function Api_RegOpenKeyEx& Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&,
ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)

' 指定されたレジストリキーに関する情報を取得
Declare Function Api_RegQueryInfoKey& Lib "advapi32" Alias "RegQueryInfoKeyA" (ByVal
hKey&, ByVal lpClass$, lpcbClass&, ByVal Reserved&, SubKeys&, SubKeyLen&, ClassLen&,
Values&, ValueNameLen&, ValueLen&, SecurityDescriptor&, LastWriteTime As FILETIME)

' レジストリのハンドルを解放
Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)

' ファイルタイムをシステムタイムに変換
Declare Function Api_FileTimeToSystemTime& Lib "kernel32" Alias "FileTimeToSystemTime"
(lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME)

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 12
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
```

```

Sub Button1_on()
    Var ft As FILETIME
    Var st As SYSTEMTIME
    Var RegSubKey As String
    Var RegSubKeyHandle As Long
    Var CbClass As Long
    Var SubKeys As Long
    Var SubKeyLength As Long
    Var ClassLength As Long
    Var Values As Long
    Var ValueNameLength As Long
    Var ValueLength As Long
    Var sFileTime As String
    Var Ret As Long

    'リストボックスを初期化
    List1.Resetcontent

    'サブキーを設定
    RegSubKey = ""

    'レジストリを問い合わせモードでオープン
    Ret = Api_RegOpenKeyEx(HKEY_LOCAL_MACHINE, RegSubKey, 0, KEY_QUERY_VALUE,
    RegSubKeyHandle)

    'オープンできたとき
    If Ret = ERROR_SUCCESS Then

        '指定したキーのレジストリ情報を取得
        Ret = Api_RegQueryInfoKey(RegSubKeyHandle, ByVal 0, CbClass, 0, SubKeys,
        SubKeyLength, ClassLength, Values, ValueNameLength, ValueLength, ByVal 0, ft)

        'レジストリ情報が取得できたとき
        If Ret = ERROR_SUCCESS Then

            'ファイル時間をシステム時間に変換
            Ret = Api_FileTimeToSystemTime(ft, st)

            'レジストリ情報を表示
            List1.AddString "SubKey数" & " " & Str$(SubKeys)
            List1.AddString "SubKey最大文字数" & " " & Str$(SubKeyLength)
            List1.AddString "値の数" & " " & Str$(Values)
            List1.AddString "値の最大文字数" & " " & Str$(ValueNameLength)

            'ファイルの最終書き込み日時を編集
            sFileTime = Trim$(Str$(st.wYear)) & "/" & Trim$(Str$(st.wMonth)) & "/" & Trim$(
            Str$(st.wDay)) & "(" & KMid$("日月火水木金土", st.wDayOfWeek + 1, 1) & ")"
            List1.AddString "最終書き込み日" & " " & sFileTime
            sFileTime = Trim$(Str$(st.wHour)) & ":" & Trim$(Str$(st.wMinute)) & ":" &
            Trim$(Str$(st.wSecond))
            List1.AddString "最終書き込み時間" & " " & sFileTime

            'レジストリ情報が取得できなかったとき
        Else
            List1.AddString "取得できません！"
        End If

        'ハンドルをクローズ
        Ret = Api_RegCloseKey(RegSubKeyHandle)

        'オープンできなかったとき
    Else
        List1.AddString "オープンできません！"
    End If
End Sub

'=====
'=
'=====
While 1

```

```

WaitEvent
Wend
Stop
End

```

レジストリキーに関する情報を取得 (II)

RegOpenKeyEx レジストリのキーのハンドルを確保
 SHQueryInfoKey レジストリキーに関する情報を取得
 RegCloseKey レジストリのハンドルを解放



```

'=====
'= レジストリキーに関する情報を取得
'=   (SHQueryInfoKey.bas)
'=====

#include "Windows.bi"

#define HKEY_USERS -2147483645      'Windowsを利用するユーザー個別の情報
#define HKEY_LOCAL_MACHINE -2147483646 'PCを利用するユーザーに共通の設定情報
#define HKEY_CURRENT_USER -2147483647 '現在Windowsにログインしているユーザーの情報
#define HKEY_CLASSES_ROOT -2147483648 '拡張子に関する情報や、それらとアプリケーションとの関連
                                     'づけに関する情報
#define KEY_QUERY_VALUE &H1        'サブキーデータを問い合わせるためのアクセス権
#define ERROR_SUCCESS &H0         '正常終了の戻り値を示す

' レジストリのキーのハンドルを確保
Declare Function Api_RegOpenKeyEx& Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&,
ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)

' レジストリキーに関する情報を取得
Declare Function Api_SHQueryInfoKey& Lib "Shlwapi" Alias "SHQueryInfoKeyA" (ByVal hkey&,
pcSubKeys&, pcchMaxSubKeyLen&, pcValues&, pcchMaxValueNameLen&)

' レジストリのハンドルを解放
Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)

Var Shared List1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
List1.SetWindowSize 202, 62
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====

Declare Sub Button1_on edec1 ()
Sub Button1_on ()
  Var SubKey As String
  Var hSubKey As Long
  Var SubKeys As Long
  Var SubKeyLength As Long
  Var Values As Long
  Var ValueNameLength As Long
  Var Ret As Long

  'リストビューを初期化
  List1.ResetContent

```

```

'サブキーを設定
SubKey = ""

'レジストリを読み取りモードでオープン
Ret = Api_RegOpenKeyEx(HKEY_LOCAL_MACHINE, SubKey, 0, KEY_QUERY_VALUE, hSubKey)

'オープンできたときは
If Ret = ERROR_SUCCESS Then

    '指定したキーのレジストリ情報を取得
    Ret = Api_SHQueryInfoKey(hSubKey, SubKeys, SubKeyLength, Values,
ValueNameLength)

    'レジストリ情報が取得できたとき
    If Ret = ERROR_SUCCESS Then

        'レジストリ情報を表示
        List1.AddString "サブキーの数" & Str$(SubKeys)
        List1.AddString "サブキーの最大文字数" & Str$(SubKeyLength)
        List1.AddString "値の数" & Str$(Values)
        List1.AddString "値の最大文字数" & Str$(ValueNameLength)

    'レジストリ情報が取得できなかったとき
    Else

        'エラーメッセージを表示
        List1.AddString "取得できませんでした。"
    End If

    'ハンドルをクローズ
    Ret = Api_RegCloseKey(hSubKey)

'オープンできなかったとき
Else

    'エラーメッセージを表示
    List1.AddString "オープンできませんでした。"
End If
End Sub

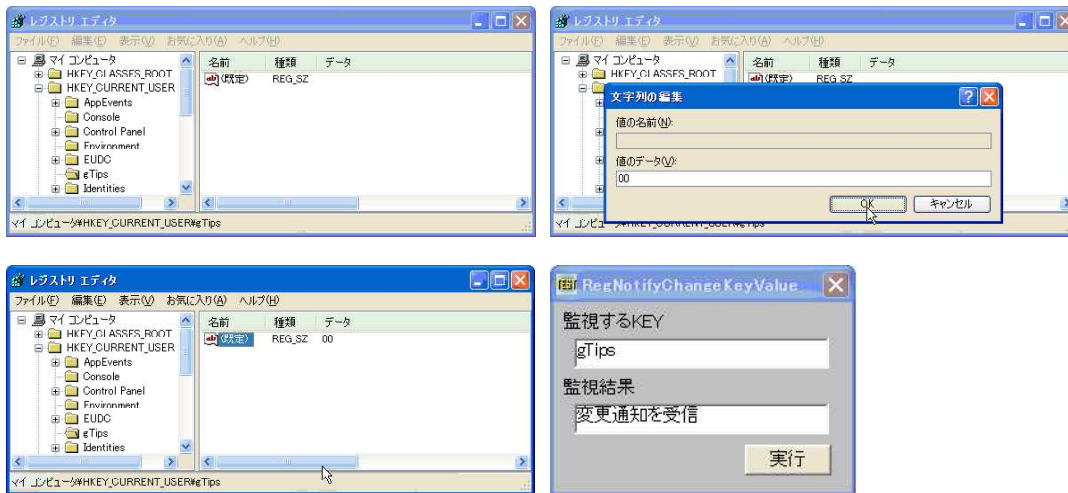
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリキーの変更を監視

RegOpenKeyEx レジストリのキーのハンドルを確保
RegNotifyChangeKeyValue レジストリキーの変更を通知
CreateEvent イベントオブジェクトを作成
WaitForSingleObject 指定された時間が経過するまでスレッドをスリープ
CloseHandle オープンされているオブジェクトハンドルをクローズ
RegCloseKey レジストリのハンドルを解放

例では、10秒間のタイムアウト時間を設けその間にレジストリに変化があれば即通知、無ければタイムアウトと表示されます。



```

'=====
' = レジストリキーの変更を監視
' = (RegNotifyChangeKeyValue.bas)
'=====

#include "Windows.bi"

Type SECURITY_ATTRIBUTES
    nLength           As Long
    lpSecurityDescriptor As Long
    bInheritHandle     As Long
End Type

#define HKEY_CURRENT_USER -2147483647
#define REG_NOTIFY_CHANGE_LAST_SET &H4

' 現在Windowsにログインしているユーザーの情報
' キー内のレジストリエントリの変更が発生したとき、呼び出し
' への通知

#define KEY_READ &H20019      ' (( &H20000 Or &H1 Or &H8 Or &H10) And (Not &H100000))
#define KEY_WRITE &H20006     ' (( &H20000 Or &H2 Or &H4) And (Not &H100000))
#define KEY_ALL_ACCESS &HF003F

#define WAIT_ABANDONED &H80
' 放棄されたためにミューテックスオブジェクトがシグナル状態
' になったことを示す
#define WAIT_FAILED &HFFFFFFF
' エラーが発生したことを示す
#define WAIT_OBJECT_0 &H0
' オブジェクトがシグナル状態になったことを示す
#define WAIT_TIMEOUT &H102
' タイムアウト時間が経過したことを示す
#define vbNullString byval 0
' 値0の文字列。値0を持つ文字列。空文字列ではない

' レジストリのキーのハンドルを確保
Declare Function Api_RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&,
ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)

' レジストリキーの変更を通知
Declare Function Api_RegNotifyChangeKeyValue Lib "advapi32" Alias
"RegNotifyChangeKeyValue" (ByVal hKey&, ByVal bWatchSubtree&, ByVal dwNotifyFilter&,
ByVal hEvent&, ByVal fAsynchronous&)

' イベントオブジェクトを作成
Declare Function Api_CreateEvent Lib "Kernel32" Alias "CreateEventA" (lpEventAttributes
As SECURITY_ATTRIBUTES, ByVal bManualReset&, bInitialState&, ByVal lpName$)

' 指定されたカーネルオブジェクトがシグナル状態になるか、指定された時間が経過するまでスレッドをスリープ
Declare Function Api_WaitForSingleObject Lib "Kernel32" Alias "WaitForSingleObject"
(ByVal hHandle&, ByVal dwMilliseconds&)

' オープンされているオブジェクトハンドルをクローズ
Declare Function Api_CloseHandle Lib "Kernel32" Alias "CloseHandle" (ByVal hObject&)

' レジストリのハンドルを解放
Declare Function Api_RegCloseKey Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)

Var Shared text(3) As Object
Var Shared Button1 As Object

```

```

For i = 0 To 3
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var KeyName As String
    Var hRegistryKey As Long
    Var EventHandle As Long
    Var sa As SECURITY_ATTRIBUTES
    Var Timeout As Long
    Var Ret As Long

    'ラベルをクリア
    Text(3).SetWindowText ""

    'レジストリキーの名前を指定
    KeyName = "gTips"
    Text(2).SetWindowText KeyName

    'レジストリキーをオープン
    Ret = Api_RegOpenKeyEx(HKEY_CURRENT_USER, KeyName, 0, KEY_READ, hRegistryKey)

    'イベントオブジェクトを作成
    EventHandle = Api_CreateEvent(sa, 1, 0, vbNullString)

    'レジストリキーの変更を監視
    Ret = Api_RegNotifyChangeKeyValue(hRegistryKey, 0, REG_NOTIFY_CHANGE_LAST_SET,
    EventHandle, 1)

    'タイムアウト時間を指定
    Timeout = 10000

    'オブジェクトのシグナル状態を監視
    Ret = Api_WaitForSingleObject(EventHandle, Timeout)

    '制御を戻す原因となったイベントを表示
    Select Case Ret
        Case WAIT_ABANDONED
            Text(3).SetWindowText "未開放ミューテックスオブジェクト"
        Case WAIT_FAILED
            Text(3).SetWindowText "待機失敗"
        Case WAIT_OBJECT_0
            Text(3).SetWindowText "変更通知を受信"
        Case WAIT_TIMEOUT
            Text(3).SetWindowText "タイムアウト"
    End Select

    'イベントオブジェクトを閉じる
    Ret = Api_CloseHandle(EventHandle)

    'レジストリキーをクローズ
    Ret = Api_RegCloseKey(hRegistryKey)
End Sub

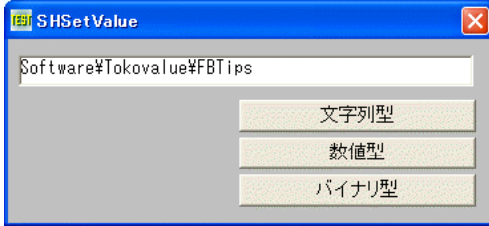
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリの値を設定

SHSetValue レジストリ値を設定

HKEY_CURRENT_USER¥Tokovalue¥FBTipsを作成し、文字列型、数値型、バイナリ型の値を書き込んでいます。



レジストリエディタで確認をしています。



```
'=====
'= レジストリの値を設定
'= (SHSetValue.bas)
'=====
#include "Windows.bi"

' レジストリ値を設定
Declare Function Api_SHSetValue& Lib "Shlwapi" Alias "SHSetValueA" (ByVal hkey&, ByVal
pszSubKey$, ByVal pszValue$, ByVal dwType&, pvData As Any, ByVal cbData&)

#define HKEY_CURRENT_USER -2147483647 '現在Windowsにログインしているユーザーの情報
#define REG_BINARY 3 'バイナリデータ
#define REG_DWORD 4 '32ビットの数値
#define REG_SZ 1 'ヌル終端文字列

Var Shared Text1 As Object
Var Shared Button1 As Object
Var Shared Button2 As Object
Var Shared Button3 As Object
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
Button2.Attach GetDlgItem("Button2") : Button2.SetFontSize 14
Button3.Attach GetDlgItem("Button3") : Button3.SetFontSize 14

Var Shared SubKey As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    SubKey = "Software¥Tokovalue¥FBTips"
    Text1.SetWindowText SubKey
End Sub
```

```

'=====
'= 文字列型を設定
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sData As String
    Var Ret As Long
    sData = "Test Value"
    Ret = Api_SHSetValue(HKEY_CURRENT_USER, SubKey, "Test", REG_SZ, sData, Len(sData))
End Sub

'=====
'= 数値型を設定
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var lData As Long
    Var Ret As Long

    lData = 10
    Ret = Api_SHSetValue(HKEY_CURRENT_USER, SubKey, "Test Value", REG_DWORD, lData,
Len(lData))
End Sub

'=====
'= バイナリ型を設定
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on()
    Var bData(10) As Byte
    Var Ret As Long

    bData(0) = 10
    bData(1) = 20
    Ret = Api_SHSetValue(HKEY_CURRENT_USER, SubKey, "Test Binary", REG_BINARY, ByVal
VarAdr(bData(0)), bData(1))
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

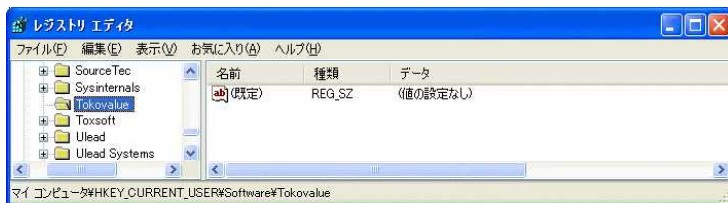
レジストリのキーを削除 (1)

RegDeleteKey 1つのサブキーを削除
FormatMessage メッセージの文字列を指定の書式で取得

図は、「FBTips」が存在し削除した場合と、存在しない場合の状態を表しています。



削除前と削除後



```
'=====
'= レジストリのキーを削除 (1)
'= Windows 95:指定したキーと全てのサブキーを削除
'= Windows NT:サブキーを持つキーは削除できない
'= (RegDeleteKey.bas)
'=====

#include "Windows.bi"

' 1つのサブキーを削除
Declare Function Api_RegDeleteKey& Lib "advapi32" Alias "RegDeleteKeyA" (ByVal hkey&,
ByVal pszSubKey$)

' メッセージの文字列を指定の書式で取得
Declare Function Api_FormatMessage& Lib "Kernel32" Alias "FormatMessageA" (ByVal
dwFlags&, lpSource As Any, ByVal dwMessageId&, ByVal dwLanguageId&, ByVal lpBuffer$,
ByVal nSize&, Arguments&)

#define FORMAT_MESSAGE_FROM_SYSTEM &H1000 'メッセージ定義として、システムメッセージテーブルリソースを使用するよう要求
#define HKEY_CURRENT_USER -2147483647 '現在Windowsにログインしているユーザーの情報

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared SubKey As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Subkey = "Software¥Tokovalue¥FBTips"
    Text1.SetWindowText SubKey
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on ()
    Var ErrCode As Long
    Var Buffer As String
    Var Ret As Long

    ErrCode = Api_RegDeleteKey(HKEY_CURRENT_USER, Subkey)

    Buffer = Space$(256)
    Ret = Api_FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0, ErrCode, 0, Buffer,
```

```

Len(Buffer), ByVal 0)

Text2.SetWindowText Left$(Buffer, Ret)
End Sub

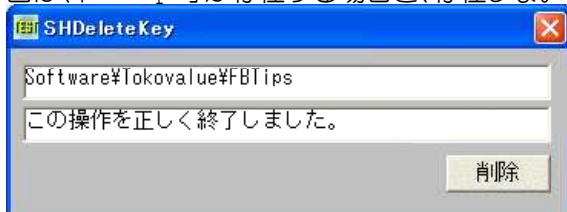
'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリのキーを削除(II)

SHDeleteKey レジストリツリーを削除
FormatMessage メッセージの文字列を指定の書式で取得

図は、「FBTips」が存在する場合と、存在しない場合の状態を表しています。



削除前と削除後



```

'=====
'= レジストリのキーを削除(II)
'= 指定したキーが持つサブキーを含めて削除
'= (SHDeleteKey.bas)
'=====
#include "Windows.bi"

```

' レジストリツリーを削除する。

```

Declare Function Api_SHDeleteKey& Lib "shlwapi" Alias "SHDeleteKeyA" (ByVal hkey&, ByVal
pszSubKey$)

```

' メッセージの文字列を指定の書式で取得

```

Declare Function Api_FormatMessage& Lib "Kernel32" Alias "FormatMessageA" (ByVal
dwFlags&, lpSource As Any, ByVal dwMessageId&, ByVal dwLanguageId&, ByVal lpBuffer$,
ByVal nSize&, Arguments&)

```

```

#define FORMAT_MESSAGE_FROM_SYSTEM &H1000 'メッセージ定義として、システムメッセージテーブルリソース
                                           'を使用するよう要求
#define HKEY_CURRENT_USER -2147483647 '現在Windowsにログインしているユーザーの情報

```

```

Var Shared Text1 As Object
Var Shared Text2 As Object

```

```

Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared SubKey As String

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Subkey = "Software¥Tokovalue¥FBTips"
    Text1.SetWindowText SubKey
End Sub

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var ErrCode As Long
    Var Buffer As String
    Var Ret As Long

    ErrCode = Api_SHDeleteKey(HKEY_CURRENT_USER, Subkey)

    Buffer = Space$(256)
    Ret = Api_FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0, ErrCode, 0, Buffer,
Len(Buffer), ByVal 0)

    Text2.SetWindowText Left$(Buffer, Ret)
End Sub

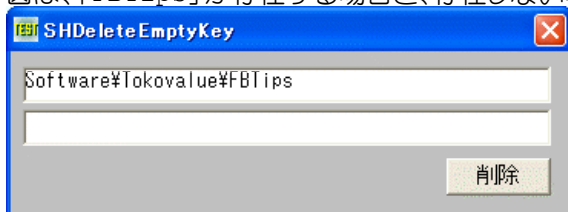
' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリのキーを削除(III)

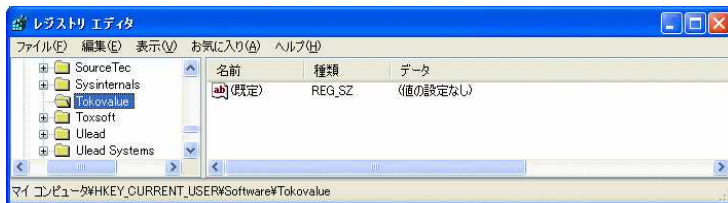
SHDeleteEmptyKey 空のレジストリツリーを削除
FormatMessage メッセージの文字列を指定の書式で取得

図は、「FBTips」が存在する場合と、存在しない場合の状態を表しています。



削除前と削除後





```
'=====
'= レジストリのキーを削除 (III)
'= 指定したキーがサブキーを持つ場合削除しない
'= (SHDeleteEmptyKey.bas)
'=====

#include "Windows.bi"

' 空のレジストリツリーを削除
Declare Function Api_SHDeleteEmptyKey& Lib "shlwapi" Alias "SHDeleteEmptyKeyA" (ByVal
hkey&, ByVal pszSubKey$)

' メッセージの文字列を指定の書式で取得
Declare Function Api_FormatMessage& Lib "Kernel32" Alias "FormatMessageA" (ByVal
dwFlags&, lpSource As Any, ByVal dwMessageId&, ByVal dwLanguageId&, ByVal lpBuffer$,
ByVal nSize&, Arguments&)

#define FORMAT_MESSAGE_FROM_SYSTEM &H1000 'メッセージ定義として、システムメッセージテーブルリソース
                                           'を使用するよう要求
#define HKEY_CURRENT_USER -2147483647     '現在Windowsにログインしているユーザーの情報

Var Shared Text1 As Object
Var Shared Text2 As Object
Var Shared Button1 As Object

Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

Var Shared SubKey As String

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Subkey = "Software¥Tokovalue¥FBTips"
    Text1.SetWindowText SubKey
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var ErrCode As Long
    Var Buffer As String
    Var Ret As Long

    ErrCode = Api_SHDeleteEmptyKey(HKEY_CURRENT_USER, Subkey)

    Buffer = Space$(256)
    Ret = Api_FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0, ErrCode, 0, Buffer,
Len(Buffer), ByVal 0)

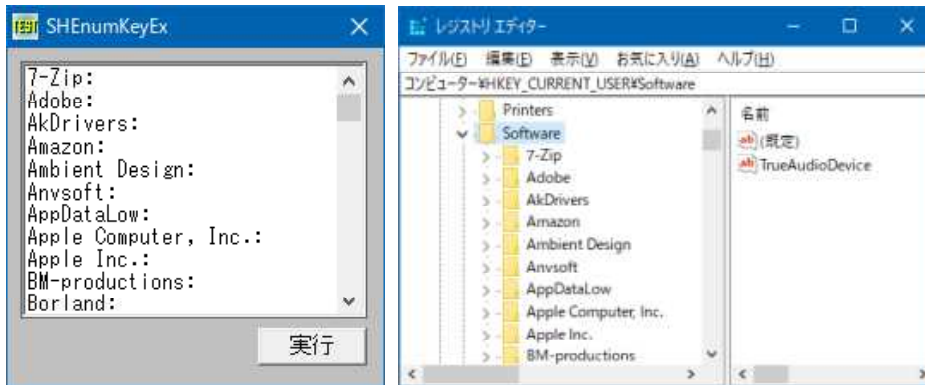
    Text2.SetWindowText Left$(Buffer, Ret)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
```


Wend
Stop
End

レジストリのキーを列挙(1)

SHEnumKeyEx レジストリのサブキーを列挙
RegOpenKeyEx レジストリのキーのハンドルを確保
RegCloseKey レジストリのハンドルを解放



```
'=====
'= レジストリのキーを列挙(1)
'= (SHEnumKeyEx.bas)
'=====

#include "Windows.bi"

#define HKEY_CURRENT_USER -2147483647 '現在Windowsにログインしているユーザーの情報
#define ERROR_SUCCESS &H0 '正常終了の戻り値を示す
#define KEY_ALL_ACCESS &H3F

' レジストリのサブキーを列挙
Declare Function Api_SHEnumKeyEx& Lib "Shlwapi" Alias "SHEnumKeyExA" (ByVal hKey&, ByVal dwIndex&, ByVal pszName$, ByRef pcchName&)

' レジストリのキーのハンドルを確保
Declare Function Api_RegOpenKeyEx& Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&, ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)

' レジストリのハンドルを解放
Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)

Var Shared List1 As Object
Var Shared Button1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====

Declare Sub Button1_on_eidecl ()
Sub Button1_on()
    Var lKey As Long
    Var sSubkey As String
    Var sName As String
    Var lName As Long
    Var sClass As String
    Var lClass As Long
    Var i As Long
    Var Ret As Long

    sSubkey = "Software"
    Ret = Api_RegOpenKeyEx(HKEY_CURRENT_USER, sSubkey, 0, KEY_ALL_ACCESS, lKey)
```

```

If Ret <> ERROR_SUCCESS Then Exit Sub

List1.Resetcontent
i = 0
Do
    lName = 1000
    lClass = 1000
    sName = String$(lName, 0)
    sClass = String$(lClass, 0)

    Ret = Api_SHEnumKeyEx(lKey, i, sName, lName)

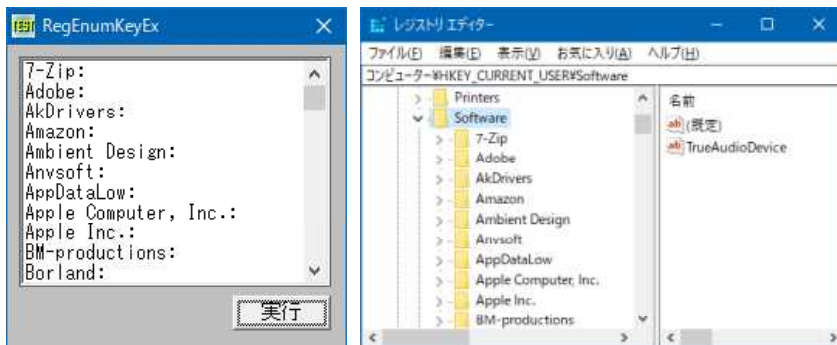
    If Ret <> ERROR_SUCCESS Then Exit Do
    sName = Left$(sName, lName)
    sClass = Left$(sClass, lClass)
    List1.AddString sName & ":" & sClass
    i = i + 1
Loop
Ret = Api_RegCloseKey(lKey)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリのキーを列挙(II)

RegEnumKeyEx 指定された開いているレジストリーキーのサブキーを列挙
RegOpenKeyEx レジストリのキーのハンドルを確保
RegCloseKey レジストリのハンドルを解放



SHEnumKeyEx との違いは、FILETIME構造体より最終書き込み時間が取得できることです。

```

' =====
' = レジストリのキーを列挙 (II)
' = (RegEnumKeyEx.bas)
' =====

#include "Windows.bi"

#define HKEY_CURRENT_USER -2147483647
#define ERROR_SUCCESS &H0
#define KEY_ALL_ACCESS &H3F

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

' 指定された開いているレジストリーキーのサブキーを列挙
Declare Function Api_RegEnumKeyEx& Lib "advapi32" Alias "RegEnumKeyExA" (ByVal hKey&,

```

```
ByVal dwIndex&, ByVal lpName$, ByRef lpcbName&, ByVal lpReserved&, ByVal lpClass$, ByRef lpcbClass&, ByRef lpftLastWriteTime As FILETIME)
```

' レジストリのキーのハンドルを確保

```
Declare Function Api_RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&, ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)
```

' レジストリのハンドルを解放

```
Declare Function Api_RegCloseKey Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)
```

```
Var Shared List1 As Object
Var Shared Button1 As Object
```

```
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

```
'=====
'=
'=====
```

```
Declare Sub Button1_on edec1 ()
```

```
Sub Button1_on()
```

```
    Var ft As FILETIME
    Var lKey As Long
    Var sSubkey As String
    Var sName As String
    Var lName As Long
    Var sClass As String
    Var lClass As Long
    Var i As Long
    Var Ret As Long
```

```
    sSubkey = "Software"
```

```
    Ret = Api_RegOpenKeyEx(HKEY_CURRENT_USER, sSubkey, 0, KEY_ALL_ACCESS, lKey)
```

```
    If Ret <> ERROR_SUCCESS Then Exit Sub
```

```
    List1.Resetcontent
```

```
    i = 0
```

```
    Do
```

```
        lName = 1000
        lClass = 1000
        sName = String$(lName, 0)
        sClass = String$(lClass, 0)
```

```
        Ret = Api_RegEnumKeyEx(lKey, i, sName, lName, 0, sClass, lClass, ft)
```

```
        If Ret <> ERROR_SUCCESS Then Exit Do
```

```
        sName = Left$(sName, lName)
```

```
        sClass = Left$(sClass, lClass)
```

```
        List1.AddString sName & ":" & sClass
```

```
        i = i + 1
```

```
    Loop
```

```
    Ret = Api_RegCloseKey(lKey)
```

```
End Sub
```

```
'=====
'=
'=====
```

```
While 1
```

```
    WaitEvent
```

```
Wend
```

```
Stop
```

```
End
```

レジストリの最終書込時間を取得

RegEnumKeyEx 指定された開いているレジストリキーのサブキーを列挙

RegOpenKeyEx レジストリのキーのハンドルを確保

RegCloseKey レジストリのハンドルを解放
 FileTimeToSystemTime ファイルタイムをシステムタイムに変換



```
'=====
'= レジストリの最終書込時間を取得
'= (RegOpenKeyEx.bas)
'=====

#include "Windows.bi"

#define HKEY_CLASSES_ROOT -2147483648
#define HKEY_CURRENT_CONFIG -2147483643
#define HKEY_CURRENT_USER -2147483647
#define HKEY_DYN_DATA -2147483642
#define HKEY_LOCAL_MACHINE -2147483646
#define HKEY_USERS -2147483645
#define KEY_ENUMERATE_SUB_KEYS &H8
#define ERROR_NO_MORE_ITEMS 259
#define REGSTR_MAX_VALUE_LENGTH 256

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type

' レジストリのキーのハンドルを確保
Declare Function Api_RegOpenKeyEx& Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&,
ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)

' 指定された開いているレジストリーキーのサブキーを列挙
Declare Function Api_RegEnumKeyEx& Lib "advapi32" Alias "RegEnumKeyExA" (ByVal hKey&,
ByVal dwIndex&, ByVal lpName$, ByRef lpcbName&, ByVal lpReserved&, ByVal lpClass$, ByRef
lpcbClass&, ByRef lpftLastWriteTime As FILETIME)

' ファイルタイムをシステムタイムに変換
Declare Function Api_FileTimeToSystemTime& Lib "kernel32" Alias "FileTimeToSystemTime"
(lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME)

' レジストリのハンドルを解放
Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)

Var Shared List1 As Object
Var Shared Button1 As Object
List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14
```

' 拡張子に関する情報や、それらとアプリケーションとの関連
 づけに関する情報
 ' Display解像度/Font等の画面表示に関するカレントユー
 ザーの情報、プリンターに関する情報
 ' 現在Windowsにログインしているユーザーの情報
 ' 動的に変化する様々な情報
 ' PCを利用するユーザーに共通の設定情報
 ' Windowsを利用するユーザー個別の情報
 ' サブキーの列挙を許可
 ' ,
 ' ,
 ' 下位32ビット値
 ' 上位32ビット値
 ' 年
 ' 月(1:1月 2:2月 ...)
 ' 曜(0:日曜 1:月曜 ...)
 ' 日(1:1日 2:2日 ...)
 ' 時
 ' 分
 ' 秒
 ' ミリ秒

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hSubKey As Long
    Var RegSubKey As String
    Var RegEnumSubKey As String
    Var SubKeyNameLen As Long
    Var ft As FILETIME
    Var st As SYSTEMTIME
    Var FileTime As String
    Var i As Long
    Var Ret As Long

    'リストを初期化
    List1.Resetcontent

    'サブキーを列挙するキーのパスを設定
    RegSubKey = ""

    'レジストリを読み取りモードでオープン
    Ret = Api_RegOpenKeyEx(HKEY_LOCAL_MACHINE, RegSubKey, 0, KEY_ENUMERATE_SUB_KEYS,
hSubKey)

    'オープンエラーのときは
    If Ret <> ERROR_SUCCESS Then

        'エラーを表示して終了
        List1.AddString "オープンできませんでした。"
        Exit Sub
    End If

    'サブキーのインデックスを初期化
    SubKeyIndex = 0

    'サブキーの列挙が完了するまで処理を反復
    Do
        '列挙するサブキー名の格納バッファを初期化
        RegEnumSubKey = Space$(REGSTR_MAX_VALUE_LENGTH)

        'バッファサイズを初期化
        SubKeyNameLen = Len(RegEnumSubKey)

        'インデックスで指定したサブキーを取得
        Ret = Api_RegEnumKeyEx(hSubKey, i, RegEnumSubKey, SubKeyNameLen, 0, ByVal 0, 0,
ft)

        '列挙するサブキーがないときは
        If Ret = ERROR_NO_MORE_ITEMS Then

            '反復処理終了
            Exit Do
        End If

        'ファイル時間をシステム時間に変換
        Ret = Api_FileTimeToSystemTime(ft, st)

        'レジストリ情報を表示
        RegEnumSubKey = Left$(RegEnumSubKey, InStr(RegEnumSubKey, Chr$(0)) - 1)

        'ファイルの最終書き込み日時を編集
        FileTime = Trim$(Str$(st.wYear)) & "/" & Trim$(Str$(st.wMonth)) & "/" &
Trim$(Str$(st.wDay)) & "(" & KMid$("日月火水木金土", st.wDayOfWeek + 1, 1) & ")" &
Trim$(Str$(st.wHour)) & ":" & Trim$(Str$(st.wMinute)) & ":" & Trim$(Str$(st.wSecond))
        List1.AddString Format$(RegEnumSubKey, "&          & ") & FileTime

        'サブキーのインデックスを加算
        i = i + 1
    Loop

```

```

' オープンしたレジストリをクローズ
Ret = Api_RegCloseKey (hSubKey)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリのサブキーを列挙

レジストリのサブキーを列挙します。
RegOpenKeyEx レジストリキーのハンドルを取得
RegCloseKey レジストリのハンドルを解放
SHEnumKeyEx レジストリのサブキーを列挙



```

' =====
' = レジストリのサブキーを列挙
' = (RegSubKeyList.bas)
' =====
#include "Windows.bi"

' レジストリのキーのハンドルを確保
Declare Function Api_RegOpenKeyEx& Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&,
ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)

' レジストリのハンドルを解放
Declare Function Api_RegCloseKey& Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)

' レジストリのサブキーを列挙
Declare Function Api_SHEnumKeyEx& Lib "Shlwapi" Alias "SHEnumKeyExA" (ByVal hKey&, ByVal
dwIndex&, ByVal pszName$, pcchName&)

#define HKEY_CLASSES_ROOT -2147483648
#define HKEY_CURRENT_USER -2147483647
#define HKEY_LOCAL_MACHINE -2147483646
#define HKEY_USERS -2147483645
#define HKEY_CURRENT_CONFIG -2147483643

#define HKEY_DYN_DATA -2147483642
#define KEY_QUERY_VALUE &H1
#define REGSTR_MAX_VALUE_LENGTH 256
#define ERROR_NO_MORE_ITEMS 259
#define ERROR_SUCCESS &H0

Var Shared List1 As Object
Var Shared Comb1 As Object
Var Shared Button1 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14

```

' 拡張子に関する情報や、それらとアプリケーションとの関連
 づけに関する情報
 ' 現在Windowsにログインしているユーザーの情報
 ' PCを利用するユーザーに共通の設定情報
 ' Windowsを利用するユーザー個別の情報
 ' Display解像度/Font等の画面表示に関するカレントユー
 ザーの情報、プリンターに関する情報

' 動的に変化する様々な情報
 ' サブキーデータを問い合わせるためのアクセス権

' 正常終了の戻り値を示す

```

Combol.Attach GetDlgItem("Combol") : Combol.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Combol.AddString "HKEY_CLASSES_ROOT"
    Combol.AddString "HKEY_CURRENT_USER"
    Combol.AddString "HKEY_LOCAL_MACHINE"
    Combol.AddString "HKEY_USERS"
    Combol.AddString "HKEY_CURRENT_CONFIG"
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hKeys As Long
    Var Item As Long
    Var hRegSubKey As Long
    Var strRegSubKey As String
    Var strRegEnumSubKey As String
    Var SubKeyIndex As Long
    Var SubKeyNameLength As Long
    Var Ret As Long

    Item = Combol.GetCursel
    If Item < 0 Then
        A% = MsgBox("", "項目を選択してください！", 0, 2)
        Exit Sub
    End If

    Select Case Item
        Case 0
            hKeys = HKEY_CLASSES_ROOT
        Case 1
            hKeys = HKEY_CURRENT_USER
        Case 2
            hKeys = HKEY_LOCAL_MACHINE
        Case 3
            hKeys = HKEY_USERS
        Case 4
            hKeys = HKEY_CURRENT_CONFIG
    End Select

    'リストボックスをクリア
    List1.ResetContent

    'サブキーを列挙するキーのパスを設定
    strRegSubKey = ""

    'レジストリを読み取りモードでオープン
    Ret = Api_RegOpenKeyEx(hKeys, strRegSubKey, 0, KEY_QUERY_VALUE, hRegSubKey)

    'オープンエラーのとき
    If Ret <> ERROR_SUCCESS Then
        'エラーを表示して終了
        List1.AddString "キーのオープンに失敗しました"
        Exit Sub
    End If

    'サブキーのインデックスを初期化
    SubKeyIndex = 0

    'サブキーの列挙が完了するまで処理を反復
    Do

```

```

' 列挙するサブキー名の格納バッファを初期化
strRegEnumSubKey = Space$(REGSTR_MAX_VALUE_LENGTH)

' バッファサイズを初期化
SubKeyNameLength = Len(strRegEnumSubKey)

' インデックスで指定したサブキーを取得
Ret = Api_SHEnumKeyEx(hRegSubKey, SubKeyIndex, strRegEnumSubKey,
SubKeyNameLength)

' 列挙するサブキーがないとき
If Ret = ERROR_NO_MORE_ITEMS Then

    ' 反復処理終了
    Exit Do
End If

' リストボックスに取得したサブキーを追加
List1.AddString Left$(strRegEnumSubKey, InStr(strRegEnumSubKey, Chr$(0)) - 1)

' サブキーのインデックスを加算
SubKeyIndex = SubKeyIndex + 1
Loop

' レジストリをクローズ
Ret = Api_RegCloseKey(hRegSubKey)
End Sub

' =====
' =
' =====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    List1.Resetcontent
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリを読む (32Bit機)

レジストリから**Internet Explorer**のバージョンを取得する例です。

RegQueryValueEx レジストリの値を取得する

RegOpenKeyEx レジストリのキーを開ける

RegCloseKey レジストリのハンドルを解放する



バージョンは下記レジストリにあります。

RootKey:HKEY_LOCAL_MACHINE

SubKey:"SOFTWARE\Microsoft\Internet Explorer"

名前:"Version"

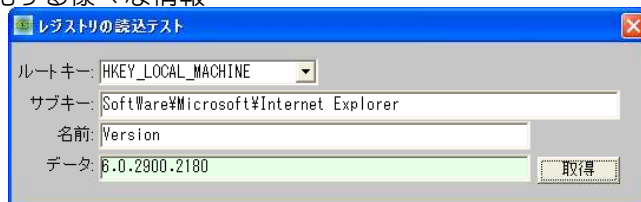
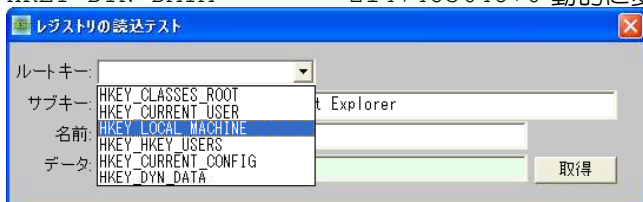
データ:6.0.2900.2180

※RootKey

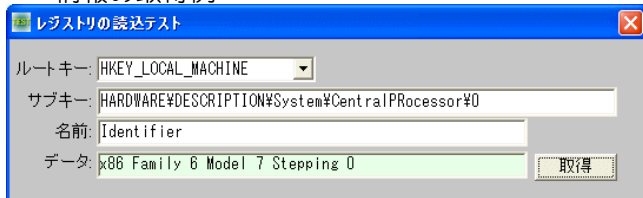
HKEY_CLASSES_ROOT -2147483648 拡張子に関する情報や、それらとアプリケーションとの関連づけに関する情報

HKEY_CURRENT_USER -2147483648+1 現在Windowsにログインしているユーザーの情報

HKEY_LOCAL_MACHINE -2147483648+2 PCを利用するユーザーに共通の設定情報
HKEY_USERS -2147483648+3 Windowsを利用するユーザー個別の情報
HKEY_CURRENT_CONFIG -2147483648+5 Display解像度/Font等の画面表示に関するカレントユーザーの情報、プリンターに関する情報
HKEY_DYN_DATA -2147483648+6 動的に変化する様々な情報



CPU情報の取得例



Windowsのバージョンによっては、異なる場合があります。

取得項目	RootKey	SubKey	名前	データ
IE Version	HKEY_LOCAL_MACHINE	Software\Microsoft\Internet Explorer	Version	6.0.2900.2180
使用者名	HKEY_LOCAL_MACHINE	Software\Microsoft\Windows\CurrentVersion	RegisteredOwner	
会社名	HKEY_LOCAL_MACHINE	Software\Microsoft\Windows\CurrentVersion	RegisteredOrganization	
Mail Client	HKEY_LOCAL_MACHINE	Software\Clients\Mail		Outlook Express
プロダクトID	HKEY_LOCAL_MACHINE	Software\Microsoft\Windows\CurrentVersion	ProductID	55380-OEM-00xxxxx-xxxxxx
		Software\Microsoft\Windows NT\CurrentVersion		
システムフォント	HKEY_CURRENT_CONFIG	Software\Fonts	FIXEDFON.FON	jvqafix.fon
			FONT.S.FON	jvqasys.fon
			OEMFONT.FON	vga932.fon
ホームページアドレス	HKEY_CURRENT_USER	Software\Internet Explorer\Main	Start Page	http://www.yahoo.co.jp/

```
'=====
'= レジストリの読み
'= (ReadReg.bas)
'=====
#include "Windows.bi"
```

'レジストリの値を取得する

```
Declare Function Api_RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" (ByVal hKey&, ByVal lpvName$, ByVal lpReserved&, ByVal lpType&, ByVal lpData$, lpcbData&)
```

'レジストリのキーを開ける (ハンドルの確保)

```
Declare Function Api_RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&, ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)
```

'レジストリのハンドルを解放する

```
Declare Function Api_RegCloseKey Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)
```

```
#define HKEY_CLASSES_ROOT -2147483648
#define HKEY_CURRENT_CONFIG -2147483643
#define HKEY_CURRENT_USER -2147483647
#define HKEY_DYN_DATA -2147483642
#define HKEY_LOCAL_MACHINE -2147483646
#define HKEY_USERS -2147483645
#define KEY_QUERY_VALUE &H1
#define ERROR_SUCCESS &H0
```

'拡張子に関する情報や、それらとアプリケーションとの関連づけに関する情報
'Display解像度/Font等の画面表示に関するカレントユーザーの情報、プリンターに関する情報
'現在Windowsにログインしているユーザーの情報
'動的に変化する様々な情報
'PCを利用するユーザーに共通の設定情報
'Windowsを利用するユーザー個別の情報
'サブキーデータを問い合わせるためのアクセス権
'正常終了の戻り値を示す

```
Var Shared Comb1 As Object
```

```

Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Text(4) As Object
Var Shared Button1 As Object

Combol.Attach GetDlgItem("Combol") : Combol.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

For i = 0 To 4
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next

Var Shared cNo As Integer

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Combol.AddString "HKEY_CLASSES_ROOT"
    Combol.AddString "HKEY_CURRENT_USER"
    Combol.AddString "HKEY_LOCAL_MACHINE"
    Combol.AddString "HKEY_HKEY_USERS"
    Combol.AddString "HKEY_CURRENT_CONFIG"
    Combol.AddString "HKEY_DYN_DATA"

    Combol.SetWindowText "HKEY_LOCAL_MACHINE"
    cNo = 2
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hKey As Long
    Var Length As Long
    Var RootKey As Long
    Var vName$ As String
    Var lpData$ As String
    Var SubKey$ As String
    Var Ret As Long

    Select Case cNo
        Case 0
            RootKey = HKEY_CLASSES_ROOT
        Case 1
            RootKey = HKEY_CURRENT_USER
        Case 2
            RootKey = HKEY_LOCAL_MACHINE
        Case 3
            RootKey = HKEY_HKEY_USERS
        Case 4
            RootKey = HKEY_CURRENT_CONFIG
        Case 5
            RootKey = HKEY_DYN_DATA
    End Select

    SubKey$ = Edit1.GetWindowText    '"SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion"

    'キーをオープンしてハンドルを得る
    Ret = Api_RegOpenKeyEx(RootKey, SubKey$, 0, KEY_QUERY_VALUE, hKey)

    'vNameに値を取得したい名前を入れる
    vName$ = Edit2.GetWindowText    '"ProductID"

```

```

'バッファを確保する
lpData$ = String$(250, Chr$(0))
Length = Len(lpData$)

'hKeyはRegOpenKeyExで開いたキーのハンドル
Ret = Api_RegQueryValueEx(hKey, vName$, 0, 0, lpData$, Length)

'テキストに出力
Text(4).SetWindowText Left$(lpData$, InStr(lpData$, Chr$(0)) - 1)

'ハンドルを閉じる
Ret = Api_RegCloseKey(hKey)
End Sub

'=====
'=
'=====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change ()
    cNo = Combo1.GetCursel
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

レジストリを読む(64Bit機)

レジストリから**Internet Explorer**のバージョンを取得する例です。

RegQueryValueEx レジストリの値を取得する
RegOpenKeyEx レジストリのキーを開ける
RegCloseKey レジストリのハンドルを解放する

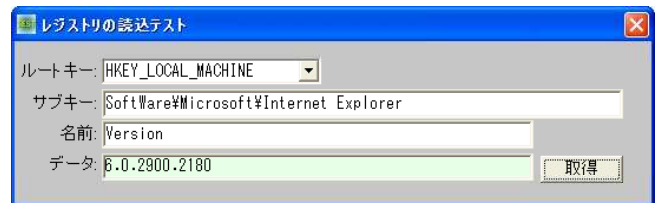
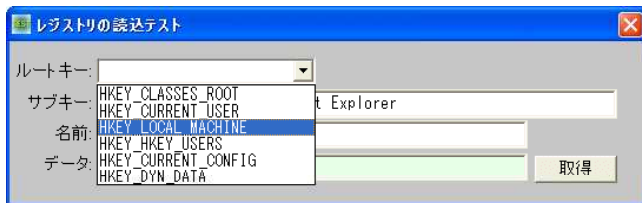
バージョンは下記レジストリにあります。

RootKey:HKEY_LOCAL_MACHINE
 SubKey:"SOFTWARE\Microsoft\Internet Explorer"
 名前:"Version"
 データ:6.0.2900.2180

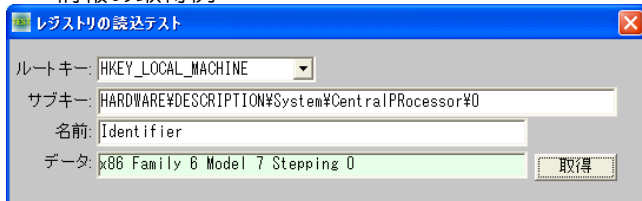


※RootKey

HKEY_CLASSES_ROOT	-2147483648	拡張子に関する情報や、それらとアプリケーションとの関連づけに関する情報
HKEY_CURRENT_USER	-2147483648+1	現在Windowsにログインしているユーザーの情報
HKEY_LOCAL_MACHINE	-2147483648+2	PCを利用するユーザーに共通の設定情報
HKEY_USERS	-2147483648+3	Windowsを利用するユーザー個別の情報
HKEY_CURRENT_CONFIG	-2147483648+5	Display解像度/Font等の画面表示に関するカレントユーザーの情報、プリンターに関する情報
HKEY_DYN_DATA	-2147483648+6	動的に変化する様々な情報



CPU情報の取得例



Windowsのバージョンによっては、異なる場合があります。

取得項目		RootKey	SubKey	名前	データ
IE Version		HKEY_LOCAL_MACHINE	Software\Microsoft\Internet Explorer	Version	6.0.2900.2180
使用者名		HKEY_LOCAL_MACHINE	Software\Microsoft\Windows\CurrentVersion	RegisteredOwner	
会社名		HKEY_LOCAL_MACHINE	Software\Microsoft\Windows\CurrentVersion	RegisteredOrganization	
Mail Client		HKEY_LOCAL_MACHINE	Software\Clients\Mail		Outlook Express
プロダクトID	95、98	HKEY_LOCAL_MACHINE	Software\Microsoft\Windows\CurrentVersion	ProductID	55380-OEM-00xxxxx-xxxxxx
	NT、2K、XP		Software\Microsoft\Windows NT\CurrentVersion		
システムフォント		HKEY_CURRENT_CONFIG	Software\Fonts	FIXEDFON.FON	jvqafix.fon
				FONTS.FON	jvqasys.fon
				OEMFONT.FON	vga932.fon
ホームページアドレス		HKEY_CURRENT_USER	Software\Internet Explorer\Main	Start Page	http://www.yahoo.co.jp/

```
'=====
'= レジストリの読み
'= (ReadReg2.bas) Windows 7 64Bit版対応
'=====
#include "Windows.bi"
```

'レジストリの値を取得する

```
Declare Function Api_RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" (ByVal hKey&, ByVal lpvName$, ByVal lpReserved&, ByVal lpType&, ByVal lpData$, lpcbData&)
```

'レジストリのキーを開ける (ハンドルの確保)

```
Declare Function Api_RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey&, ByVal lpSubKey$, ByVal ulOptions&, ByVal samDesired&, phkResult&)
```

'レジストリのハンドルを解放する

```
Declare Function Api_RegCloseKey Lib "advapi32" Alias "RegCloseKey" (ByVal hKey&)
```

```
#define HKEY_CLASSES_ROOT -2147483648
#define HKEY_CURRENT_CONFIG -2147483643
#define HKEY_CURRENT_USER -2147483647
#define HKEY_DYN_DATA -2147483642
#define HKEY_LOCAL_MACHINE -2147483646
#define HKEY_USERS -2147483645
#define KEY_QUERY_VALUE &H1
#define KEY_WOW64_32KEY &H200
#define KEY_WOW64_64KEY &H100
#define ERROR_SUCCESS &H0
```

```
Var Shared Comb1 As Object
Var Shared Edit1 As Object
```

'拡張子に関する情報や、それらとアプリケーションとの関連づけに関する情報

'Display解像度/Font等の画面表示に関するカレントユーザーの情報、プリンターに関する情報

'現在Windowsにログインしているユーザーの情報

'動的に変化する様々な情報

'PCを利用するユーザーに共通の設定情報

'Windowsを利用するユーザー個別の情報

'サブキーデータを問い合わせるためのアクセス権

'64bitまたは32bitアプリケーションが、32bitキーをオープンできるようにする

'64bitまたは32bitアプリケーションが、64bitキーをオープンできるようにする

'正常終了の戻り値を示す

```

Var Shared Edit2 As Object
Var Shared Text(4) As Object
Var Shared Button1 As Object

Combo1.Attach GetDlgItem("Combo1") : Combo1.SetFontSize 14
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

For i = 0 To 4
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next

Var Shared cNo As Integer

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Combo1.AddString "HKEY_CLASSES_ROOT"
    Combo1.AddString "HKEY_CURRENT_USER"
    Combo1.AddString "HKEY_LOCAL_MACHINE"
    Combo1.AddString "HKEY_HKEY_USERS"
    Combo1.AddString "HKEY_CURRENT_CONFIG"
    Combo1.AddString "HKEY_DYN_DATA"

    Combo1.SetWindowText "HKEY_LOCAL_MACHINE"
    cNo = 2
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var hKey As Long
    Var Length As Long
    Var RootKey As Long
    Var vName$ As String
    Var lpData$ As String
    Var SubKey$ As String
    Var Ret As Long

    Select Case cNo
        Case 0
            RootKey = HKEY_CLASSES_ROOT
        Case 1
            RootKey = HKEY_CURRENT_USER
        Case 2
            RootKey = HKEY_LOCAL_MACHINE
        Case 3
            RootKey = HKEY_HKEY_USERS
        Case 4
            RootKey = HKEY_CURRENT_CONFIG
        Case 5
            RootKey = HKEY_DYN_DATA
    End Select

    SubKey$ = Edit1.GetWindowText    '"SOFTWARE\Microsoft\Windows NT\CurrentVersion"

    'キーをオープンしてハンドルを得る
    Ret = Api_RegOpenKeyEx(RootKey, SubKey$, 0, KEY_QUERY_VALUE Or KEY_WOW64_64KEY, hKey)

    'vNameに値を取得したい名前を入れる
    vName$ = Edit2.GetWindowText    '"ProductID"

    'バッファを確保する
    lpData$ = String$(250, Chr$(0))

```

```

Length = Len(lpData$)

' hKeyはRegOpenKeyExで開いたキーのハンドル
Ret = Api_RegQueryValueEx(hKey, vName$, 0, 0, lpData$, Length)

' テキストに出力
Text(4).SetWindowText Left$(lpData$, InStr(lpData$, Chr$(0)) - 1)

' ハンドルを閉じる
Ret = Api_RegCloseKey(hKey)
End Sub

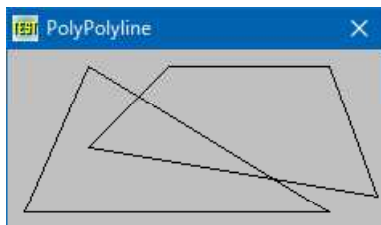
' =====
' =
' =====
Declare Sub Combo1_Change edec1 ()
Sub Combo1_Change()
    cNo = Combo1.GetCursel
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

連続した線分を一度に複数個描画

PolyPolyline 連続した線分を一度に複数個描画
GetDC デバイスコンテキストのハンドルを取得
ReleaseDC デバイスコンテキストを解放



例では、四角形と三角形を一度に描画しています。

```

' =====
' = 連続した線分を一度に複数個描画
' = (PolyPolyline.bas)
' =====
#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type

' 連続した線分を一度に複数個描画
Declare Function Api_PolyPolyline& Lib "gdi32" Alias "PolyPolyline" (ByVal hDC&, lppt As POINTAPI, lpdwPolypts&, ByVal cCount&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start ()
    Var hDC As Long
    Var pts(8) As POINTAPI
    Var numpts(1) As Long
    Var Ret As Long

    'デバイスコンテキスト取得
    hDC = Api_GetDC (GethWnd)

    '△のポイント
    pts(0).x = 50 : pts(0).y = 10
    pts(1).x = 200 : pts(1).y = 100
    pts(2).x = 10 : pts(2).y = 100
    pts(3).x = 50 : pts(3).y = 10
    numpts(0) = 4

    '□のポイント
    pts(4).x = 100 : pts(4).y = 10
    pts(5).x = 200 : pts(5).y = 10
    pts(6).x = 230 : pts(6).y = 90
    pts(7).x = 50 : pts(7).y = 60
    pts(8).x = 100 : pts(8).y = 10
    numpts(1) = 5

    '△と□の線分を一度に描画
    Ret = Api_PolyPolyline (hDC, pts(0), numpts(0), 2)

    'デバイスコンテキスト解放
    Ret = Api_ReleaseDC (GethWnd, hDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

連続した直線を描画(1)

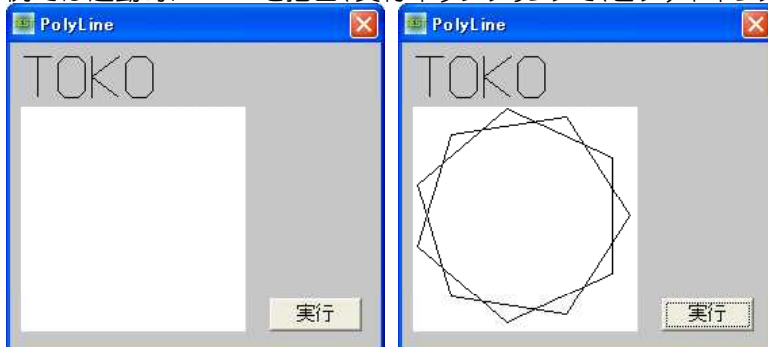
連続した直線を引きます。

Polyline 連続直線を引く

GetDC ディスプレイデバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放

例では起動時にTOKOを描画、実行ボタンクリックで、ピクチャボックスに11角形を描画しています。



```

'=====
'= 連続した直線を引く
'= (Polyline.bas)
'=====
#include "Windows.bi"

Type POINTAPI
    X    As Long
    Y    As Long
End Type

' 連続直線を引く
Declare Function Api_Polyline& Lib "gdi32" Alias "Polyline" (ByVal hDC&, lpPoint As
POINTAPI, ByVal nCount&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得。
その後、GDI 関数を使って、返されたデバイスコンテキスト内で描画を行える
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Picture1 As Object
Picture1.Attach GetDlgItem("Picture1")

Var Shared Pts(11) As POINTAPI

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Var hDC As Long
    Var Ret As Long

    hDC = Api_GetDC(GetHwnd)

    Pts(0).X = 10: Pts(0).Y = 10
    Pts(1).X = 30: Pts(1).Y = 10
    Pts(2).X = 20: Pts(2).Y = 10
    Pts(3).X = 20: Pts(3).Y = 40
    Ret = Api_Polyline(hDC, Pts(0), 4)

    Pts(0).X = 40: Pts(0).Y = 10
    Pts(1).X = 35: Pts(1).Y = 15
    Pts(2).X = 35: Pts(2).Y = 35
    Pts(3).X = 40: Pts(3).Y = 40
    Pts(4).X = 50: Pts(4).Y = 40
    Pts(5).X = 55: Pts(5).Y = 35
    Pts(6).X = 55: Pts(6).Y = 15
    Pts(7).X = 50: Pts(7).Y = 10
    Pts(8).X = 40: Pts(8).Y = 10
    Ret = Api_Polyline(hDC, Pts(0), 9)

    Pts(0).X = 60: Pts(0).Y = 10
    Pts(1).X = 60: Pts(1).Y = 40
    Pts(2).X = 60: Pts(2).Y = 25
    Pts(3).X = 80: Pts(3).Y = 10
    Pts(4).X = 60: Pts(4).Y = 25
    Pts(5).X = 80: Pts(5).Y = 40
    Ret = Api_Polyline(hDC, Pts(0), 6)

    Pts(0).X = 90: Pts(0).Y = 10
    Pts(1).X = 85: Pts(1).Y = 15
    Pts(2).X = 85: Pts(2).Y = 35
    Pts(3).X = 90: Pts(3).Y = 40
    Pts(4).X = 100: Pts(4).Y = 40
    Pts(5).X = 105: Pts(5).Y = 35
    Pts(6).X = 105: Pts(6).Y = 15
    Pts(7).X = 100: Pts(7).Y = 10

```



```

    Pts(8).X = 90: Pts(8).Y = 10
    Ret = Api_Polyline(hDC, Pts(0), 9)

    Ret = Api_ReleaseDC(GethWnd, hDC)
End Sub

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var PI As single
    Var i As Long
    Var PointNum As Long
    Var phDC As Long
    Var Ret As Long

    phDC = Api_GetDC(Picture1.GethWnd)
    PI = 3.14159
    PointNum = 11

    For i = 0 To PointNum
        Pts(i).X = Cos(2 * PI * i * 2 / PointNum) * 80 + 80
        Pts(i).Y = Sin(2 * PI * i * 2 / PointNum) * 80 + 80
    Next

    Ret = Api_Polyline(phDC, Pts(0), PointNum + 1)

    Ret = Api_ReleaseDC(Picture1.GethWnd, phDC)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

連続した直線を描画 (II)

連続した直線を描画します。

- CreatePolygonRgn 多角形のリージョンを作成
- SetWindowRgn 指定の領域をウィンドウ領域として設定
- Polyline 複数の線分からなる連続した線を描画
- GetDC デバイスコンテキストのハンドルを取得
- ReleaseDC デバイスコンテキストを解放

例では逆三角形を描画し、「Region」でその領域をウィンドウ領域に設定しています。



```

'=====
'= 連続した直線を描画
'= (Polyline2.bas)
'=====
#include "Windows.bi"

```

```

Type POINTAPI
    X    As Long
    Y    As Long
End Type

' 多角形のリージョンを作成
Declare Function Api_CreatePolygonRgn& Lib "gdi32" Alias "CreatePolygonRgn" (lppt As
POINTAPI, ByVal nCount&, ByVal nPolyFillMode&)

' 指定の領域をウィンドウ領域として設定
Declare Function Api_SetWindowRgn& Lib "user32" Alias "SetWindowRgn" (ByVal hWnd&, ByVal
hRgn&, ByVal bRedraw&)

' 複数の線分からなる連続した線を描画
Declare Function Api_Polyline& Lib "gdi32" Alias "Polyline" (ByVal hWnd&, lpPoint As
POINTAPI, ByVal nCount&)

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得
Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

' デバイスコンテキストを解放
Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

Var Shared Points(4) As POINTAPI
Var Shared hDC As Long

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    hDC = Api_GetDC(GethWnd)
    Points(1).X = 150
    Points(1).Y = 150
    Points(2).X = 0
    Points(2).Y = 0
    Points(3).X = 300
    Points(3).Y = 0
    Points(4).X = 150
    Points(4).Y = 150
End Sub

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var Ret As Long

    Ret = Api_Polyline(hDC, Points(1), 4)
End Sub

' =====
' =
' =====
Declare Sub Button2_on edec1 ()
Sub Button2_on()
    Var hRgn As Long

    hRgn = Api_CreatePolygonRgn(Points(1), 4, 1)
    Ret = Api_SetWindowRgn(GethWnd, hRgn, True)
End Sub

' =====
' =
' =====
Declare Sub Button3_on edec1 ()
Sub Button3_on()
    Var Ret As Long

```

```

Ret = Api_SetWindowRgn (GethWnd, 0, True)
End Sub

' =====
' =
' =====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose ()
    Var Ret As Long

    Ret = Api_ReleaseDC (hWnd, hDC)
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

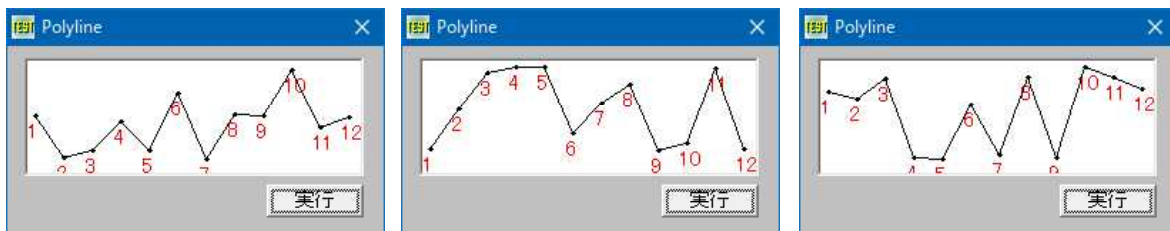
連続した直線を描画 (III)

連続した直線を描画します。例では1～12月の折れ線グラフのつもりです。

Polyline 複数の線分からなる連続した線を描画

GetDC デバイスコンテキストのハンドルを取得

ReleaseDC デバイスコンテキストを解放



```

' =====
' = 連続した直線を描画 (III)
' = (Polyline3.bas)
' =====
#include "Windows.bi"

Type POINTAPI
    x As Long
    y As Long
End Type

```

' 複数の線分からなる連続した線を描画 (実行後はペンの現在位置が変更されない)

```

Declare Function Api_Polyline& Lib "gdi32" Alias "Polyline" (ByVal hDC&, lpPoint As POINTAPI, ByVal nCount&)

```

' 指定されたウィンドウのクライアント領域または画面全体を表すディスプレイデバイスコンテキストのハンドルを取得

```

Declare Function Api_GetDC& Lib "user32" Alias "GetDC" (ByVal hWnd&)

```

' デバイスコンテキストを解放

```

Declare Function Api_ReleaseDC& Lib "user32" Alias "ReleaseDC" (ByVal hWnd&, ByVal hDC&)

```

```

Var Shared pa(11) As POINTAPI

```

```

Var Shared Picture1 As Object
Picture1.Attach GetDlgItem("Picture1")

```

```

' =====
' =
' =====

```

```

Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var phDC As Long
    Var mm As Integer
    Var Corner As Long
    Var Ret As Long

    Randomize Time

    'Picture1のデバイスコンテキスト
    phDC = Api_GetDC(Picture1.GethWnd)

    '連続直線の点の数
    Corner = 12

    '連続線分の座標
    Picture1.Cls
    Picture1.SetDrawWidth 3
    For mm = 0 To 11
        pa(mm).x = (mm + 1) * 20 - 15
        pa(mm).y = Rnd(1) * 70

        Picture1.Pset(pa(mm).x, pa(mm).y), 0
        Picture1.Symbol(pa(mm).x - 4, pa(mm).y + 4), Trim$(Str$(mm + 1)), 1, 1, 5
    Next
    Picture1.SetDrawWidth 0

    '連続した線分を描画
    Ret = Api_Polyline(phDC, pa(0), Corner)
End Sub

'=====
'=
'=====
Declare Sub MainForm_QueryClose edec1 ()
Sub MainForm_QueryClose()
    Var Ret As Long

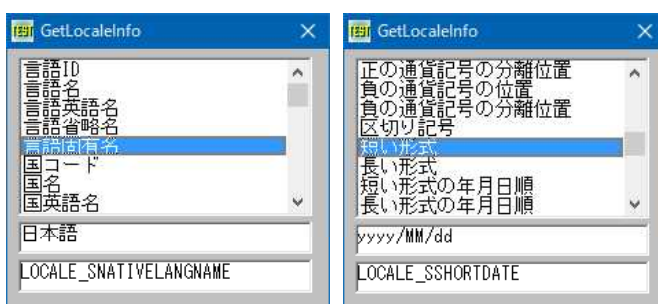
    Ret = Api_ReleaseDC(Picture1.GethWnd, phDC)
    End
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ロケール情報の取得

GetLocaleInfo ロケール情報を取得



GetLocaleInfo()関数は、第2引数のLCTypeに定数を指定することによって、いろいろなロケール情報を取得できます。

コントロールパネル → 地域と言語のオプション



```
'=====
'=   ロケール情報の取得
'=   (GetLocaleInfo.bas)
'=====

#include "Windows.bi"

'   ロケール情報を取得
Declare Function Api_GetLocaleInfo& Lib "kernel32" Alias "GetLocaleInfoA" (ByVal
Locale&, ByVal LCType&, ByVal lpLCData$, ByVal cchData&)

Var Shared List1 As Object
Var Shared Text1 As Object
Var Shared Text2 As Object

List1.Attach GetDlgItem("List1") : List1.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Text2.Attach GetDlgItem("Text2") : Text2.SetFontSize 14

Var Shared Locale(79, 2) As String

'=====
'=
'=====
Declare Function GetInfo(lInfo As Long) As String
Function GetInfo(lInfo As Long) As String
    Var Buffer As String
    Var Ret As Long

    Buffer = String$(256, 0)
    Ret = Api_GetLocaleInfo(LOCALE_USER_DEFAULT, ByVal lInfo, Buffer, Len(Buffer))
    If Ret > 0 Then
        GetInfo = Left$(Buffer, Ret - 1)
    Else
        GetInfo = ""
    End If
End Function

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    For i = 0 To 79
        For j = 0 To 2
            read Locale(i, j)
        Next
        List1.AddString Locale(i, 2)
    Next

    data LOCALE_ILANGUAGE           , &H1           , 言語ID
    data LOCALE_SLANGUAGE          , &H2           , 言語名
```

data LOCALE_	SENGLANGUAGE	, &H1001	, 言語英語名
data LOCALE_	SABBEVLANGNAME	, &H3	, 言語省略名
data LOCALE_	SNATIVELANGNAME	, &H4	, 言語固有名
data LOCALE_	ICOUNTRY	, &H5	, 国コード
data LOCALE_	SCOUNTRY	, &H6	, 国名
data LOCALE_	SENGCOUNTRY	, &H1002	, 国英語名
data LOCALE_	SABBEVCTRYNAME	, &H7	, 国省略名
data LOCALE_	SNATIVECTRYNAME	, &H8	, 国固有名
data LOCALE_	IDEFAULTLANGUAGE	, &H9	, デフォルト言語ID
data LOCALE_	IDEFAULTCOUNTRY	, &HA	, デフォルト国コード
data LOCALE_	IDEFAULTCODEPAGE	, &HB	, デフォルトOEMコードページ
data LOCALE_	IDEFAULTANSICODEPAGE	, &H1004	, デフォルトANSIコードページ
data LOCALE_	IDEFAULTMACCODEPAGE	, &H1011	, デフォルトMACコードページ
data LOCALE_	FONTSIGNATURE	, &H58	, フォント署名
data LOCALE_	SISO639LANGNAME	, &H59	, ISO言語省略名
data LOCALE_	SISO3166CTRYNAME	, &H5A	, ISO国省略名
data LOCALE_	SLIST	, &HC	, 区切り記号
data LOCALE_	IMEASURE	, &HD	, 単位
data LOCALE_	SDECIMAL	, &HE	, 小数点の記号
data LOCALE_	STHOUSAND	, &HF	, 桁区切り記号
data LOCALE_	SGROUPING	, &H10	, 区切る桁数
data LOCALE_	IDIGITS	, &H11	, 小数点以下の桁数
data LOCALE_	ILZERO	, &H12	, 少数前ゼロの桁数
data LOCALE_	INEGNUMBER	, &H1010	, 負の値の形式
data LOCALE_	SNATIVEDIGITS	, &H13	, 0から9の表記
data LOCALE_	SPOSITIVESIGN	, &H50	, 正の記号
data LOCALE_	SNEGATIVESIGN	, &H51	, 負の記号
data LOCALE_	IPOSSIGNPOSN	, &H52	, 正の記号の位置
data LOCALE_	INEGSIGNPOSN	, &H53	, 負の記号の位置
data LOCALE_	SCURRENCY	, &H14	, 通貨記号
data LOCALE_	SINTLSYMBOL	, &H15	, 国際通貨記号
data LOCALE_	SMONDECIMALSEP	, &H16	, 小数点の記号
data LOCALE_	SMONTHOUSANDSEP	, &H17	, 桁区切り記号
data LOCALE_	SMONGROUPING	, &H18	, 区切る桁数
data LOCALE_	ICURRDIGITS	, &H19	, 小数点以下の桁数
data LOCALE_	IINTLCURRDIGITS	, &H1A	, 小数点以下の桁数
data LOCALE_	ICURRENCY	, &H1B	, 正の値の形式
data LOCALE_	INEGCURR	, &H1C	, 負の値の形式
data LOCALE_	IPOSSYMPRECEDES	, &H54	, 正の通貨記号の位置
data LOCALE_	IPOSSEPBYSPACE	, &H55	, 正の通貨記号の分離位置
data LOCALE_	INEGSYMPRECEDES	, &H56	, 負の通貨記号の位置
data LOCALE_	INEGSEPBYSPACE	, &H57	, 負の通貨記号の分離位置
data LOCALE_	SDATE	, &H1D	, 区切り記号
data LOCALE_	SSHORTDATE	, &H1F	, 短い形式
data LOCALE_	SLONGDATE	, &H20	, 長い形式
data LOCALE_	IDATE	, &H21	, 短い形式の年月日順
data LOCALE_	ILDATE	, &H22	, 長い形式の年月日順
data LOCALE_	ICENTURY	, &H24	, 年の桁数
data LOCALE_	IDAYLZERO	, &H26	, 日前ゼロの有無
data LOCALE_	IMONLZERO	, &H27	, 月前ゼロの有無
data LOCALE_	ICALENDARType	, &H1009	, カレンダの種類
data LOCALE_	IOPTIONALCALENDAR	, &H100B	, 追加カレンダーの種類
data LOCALE_	IFIRSTDAYOFWEEK	, &H100C	, 週の先頭日
data LOCALE_	IFIRSTWEEKOFYEAR	, &H100D	, 年の先頭月
data LOCALE_	SDAYNAME1	, &H2A	, 週の第1日名
data LOCALE_	SDAYNAME2	, &H2B	, 週の第2日名
data LOCALE_	SDAYNAME3	, &H2C	, 週の第3日名
data LOCALE_	SDAYNAME7	, &H30	, 週の第7日名
data LOCALE_	SABBEVDAYNAME1	, &H31	, 週の第1日省略名
data LOCALE_	SABBEVDAYNAME2	, &H32	, 週の第2日省略名
data LOCALE_	SABBEVDAYNAME3	, &H33	, 週の第3日省略名
data LOCALE_	SABBEVDAYNAME7	, &H37	, 週の第7日省略名
data LOCALE_	SMONTHNAME1	, &H38	, 年の第1月名
data LOCALE_	SMONTHNAME2	, &H39	, 年の第2月名
data LOCALE_	SMONTHNAME3	, &H3A	, 年の第3月名
data LOCALE_	SMONTHNAME12	, &H43	, 年の第12月名
data LOCALE_	SMONTHNAME13	, &H100E	, 年の第13月名

```

data LOCALE_ SABBREVMONTHNAME1 , &H44 , 年の第1月省略名
data LOCALE_ SABBREVMONTHNAME2 , &H45 , 年の第2月省略名
data LOCALE_ SABBREVMONTHNAME3 , &H46 , 年の第3月省略名
data LOCALE_ SABBREVMONTHNAME12 , &H4F , 年の第12月省略名

data LOCALE_ STIME , &H1E , 区切り記号
data LOCALE_ STIMEFORMAT , &H1003 , 時間の形式
data LOCALE_ ITIME , &H23 , 時間制 (12/24)
data LOCALE_ ITIMEMARKPOSN , &H1005 , 午前午後記号の位置
data LOCALE_ ITLZERO , &H25 , 時刻前ゼロの有無
data LOCALE_ S1159 , &H28 , 午前の記号
data LOCALE_ S2359 , &H29 , 午後の記号
End Sub

' =====
' =
' =====
Declare Sub List1_Change edec1 ()
Sub List1_Change ()

    No = List1.GetCursel
    Text1.SetWindowText GetInfo (val (Locale (No, 1)))
    Text2.SetWindowText Locale (No, 0)
End Sub

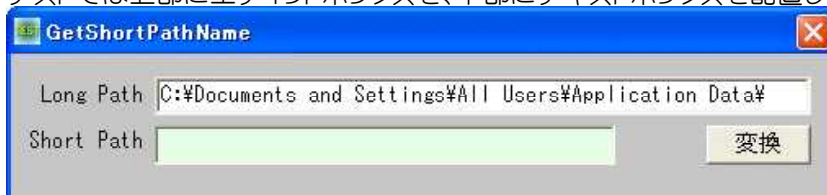
' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End

```

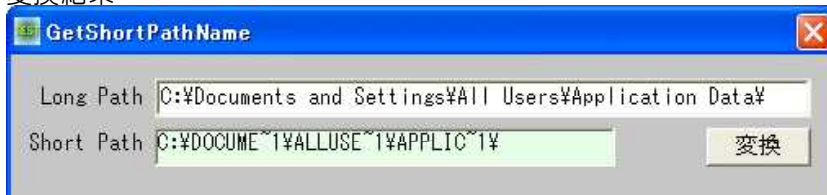
ロングパス名からショートパス名へ変換

ロングパス名をショートパス名に変換します。
GetShortPathName ファイルの短い形式のパス名を取得

LongPathで与えられたパス名を短い形式のパス名ShortPathに変換します。LongPathに含まれるすべてのディレクトリ名も短い形式に変換されます。
LongPathで与えるパスは、長い形式のパス名でなくても良くパス名中の短い形式のディレクトリ名やファイル名の部分は、そのままShortPathに出力されます。
与えるパス名は、実際にディスク上に存在していなければなりません。
テストでは上部にエディットボックスを、下部にテキストボックスを配置しています。



変換結果



```

' =====
' = ロングパス名からショートパス名へ変換
' = (GetShortPathName.bas)
' =====
#include "Windows.bi"

```

「ファイルの短い形式のパス名を取得

```
Declare Function Api_GetShortPathName& Lib "kernel32" Alias "GetShortPathNameA" (ByVal lpszLongPath$, ByVal lpszShortPath$, ByVal lBuffer&)

Var Shared Edit1 As Object
Var Shared Text(2) As Object
Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
For i = 0 To 2
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
Next

' =====
' =
' =====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()

    Edit1.SetWindowText "C:¥Documents and Settings¥All Users¥Application Data¥"
End Sub

' =====
' =
' =====
Declare Function GetShortPath edec1 (strFileName As String) As String
Function GetShortPath(strFileName As String) As String
    Var Ret As Long
    Var strPath As String

    strPath = String$(165, 0)
    Ret = Api_GetShortPathName(strFileName, strPath, 164)
    GetShortPath = Left$(strPath, Ret)
End Function

' =====
' =
' =====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var oPath As String
    Var sPath As String

    oPath = RTrim$(Edit1.GetWindowText)
    sPath = GetShortPath(oPath)

    Text(2).SetWindowText sPath
End Sub

' =====
' =
' =====
While 1
    WaitEvent
Wend
Stop
End
```

ロングパス名・ショートパス名の相互変換

ロングパス名からショートパス名へ、ショートパス名からロングパス名へ相互変換します。

GetShortPathName ファイルの短い形式のパス名を取得

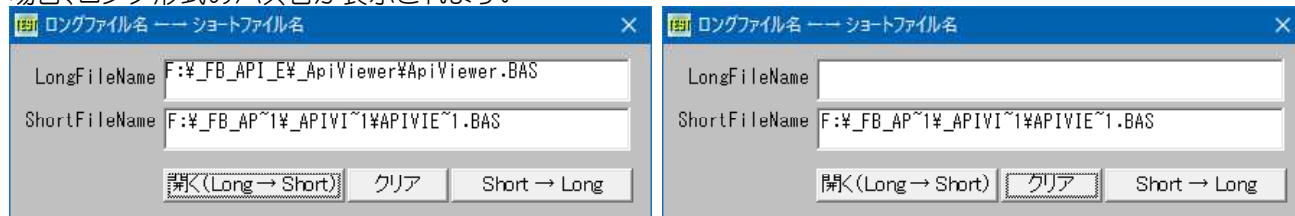
GetLongPathName ロングパス名を取得

「開く(Long → Short)」でファイルを選択すると、ショート形式で表示されます。

念のため「クリア」でロングファイル名を消去し、「Short → Long」をクリックするとロング形式のパス名が表示されます。

ShortFileName欄に適当なショート形式パス名を入力後、「Short → Long」をクリックするとそのパスが存在した

場合、ロング形式のパス名が表示されます。



```
'=====
'= ロングパス名・ショートパス名相互変換
'= (LongShortFileName.bas)
'=====
#include "Windows.bi"

' ファイルの短い形式のパス名を取得
Declare Function Api_GetShortPathName& Lib "Kernel32" Alias "GetShortPathNameA" (ByVal
lpszLongPath$, ByVal lpszShortPath$, ByVal lBuffer&)

' ロングパス名を取得
Declare Function Api_GetLongPathName& Lib "Kernel32" Alias "GetLongPathNameA" (ByVal
lpszShortPath$, ByVal lpszLongPath$, ByVal cchBuffer&)

#define MAX_PATH 260

Var Shared Edit1 As Object
Var Shared Text(2) As Object
Var Shared Button(2) As Object

Edit1.Attach getDlgItem("Edit1") : Edit1.SetFontSize 14
For i = 0 To 2
    Text(i).Attach GetDlgItem("Text" & Trim$(Str$(i + 1)))
    Text(i).SetFontSize 14
    Button(i).Attach GetDlgItem("Button" & Trim$(Str$(i + 1)))
    Button(i).SetFontSize 14
Next

'=====
'=
'=====
Declare Function GetShortName(sLongFileName As String) As String
Function GetShortName(sLongFileName As String) As String
    Var sShortPathName As String
    Var iLen As Integer
    Var Ret As Long

    'バッファの設定
    sShortPathName = Space$(260)
    iLen = Len(sShortPathName)

    'ファンクションコール
    Ret = Api_GetShortPathName(sLongFileName, sShortPathName, iLen)

    '必要な長さを切り取る
    GetShortName = Left$(sShortPathName, Ret)
End Function

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var FileName As String
    Var sFile As String

    FileName = WinOpenDlg("ファイルのオープン", "*.*", "全てのファイル(*.*)", 0)
    If FileName <> Chr$(&H1B) Then
        Text(2).SetWindowText FileName
        Edit1.SetWindowText GetShortName(FileName)
```

```

End If
End Sub

'=====
'=
'=====
Declare Sub Button2_on edec1 ()
Sub Button2_on ()
    Text(2).SetWindowText ""
End Sub

'=====
'=
'=====
Declare Sub Button3_on edec1 ()
Sub Button3_on ()
    Var sLongPathName As String
    Var iLen As Integer
    Var Ret As Long

    'バッファの設定
    sLongPathName = Space$(260)
    iLen = Len(sLongPathName)

    'ファンクションコール
    Ret = Api_GetLongPathName(Edit1.GetWindowText, sLongPathName, iLen)

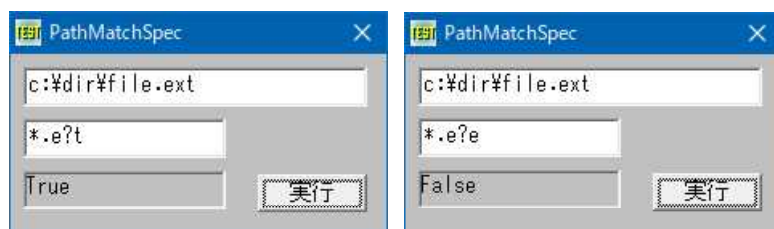
    '必要な長さを切り取る
    Text(2).SetWindowText Left$(sLongPathName, Ret)
End Sub

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```

ワイルドカードの形式と一致する文字列を検索

PathMatchSpec ワイルドカードの形式と一致する文字列を検索



```

'=====
'= ワイルドカードの形式と一致する文字列を検索
'= (PathMatchSpec2.bas)
'=====
#include "Windows.bi"

' ワイルドカードの形式と一致する文字列を検索
Declare Function Api_PathMatchSpec& Lib "shlwapi" Alias "PathMatchSpecA" (ByVal
pszFile$, ByVal pszSpec$)

Var Shared Edit1 As Object
Var Shared Edit2 As Object
Var Shared Text1 As Object
Var Shared Button1 As Object

```

```

Edit1.Attach GetDlgItem("Edit1") : Edit1.SetFontSize 14
Edit2.Attach GetDlgItem("Edit2") : Edit2.SetFontSize 14
Text1.Attach GetDlgItem("Text1") : Text1.SetFontSize 14
Button1.Attach GetDlgItem("Button1") : Button1.SetFontSize 14

```

```

'=====
'=
'=====
Declare Sub MainForm_Start edec1 ()
Sub MainForm_Start()
    Edit1.SetWindowText "c:\%dir%file.ext"
    Edit2.SetWindowText "*.e?t"
End Sub

```

```

'=====
'=
'=====
Declare Sub Button1_on edec1 ()
Sub Button1_on()
    Var sSave As String
    Var sCard As String
    Var Ret As Long

    sSave = Edit1.GetWindowText & String$(100, 0)
    sCard = Edit2.GetWindowText

    Ret = Api_PathMatchSpec(sSave, sCard)

    If Ret = 0 Then
        Text1.SetWindowText "False"
    Else
        Text1.SetWindowText "True"
    End If
End Sub

```

```

'=====
'=
'=====
While 1
    WaitEvent
Wend
Stop
End

```