ウィンドウの情報

コントロールのサイズ

.NET アプリケーションのウィンドウは、主に Form クラスを継承したオブジェクトでしょう
しかし、この過程には様々なクラスが継承されているので、とても複雑です

.NET のウィンドウの制御を知るには、全てのコントロールの
基底となる **System.Windows.Forms.Control** クラスを知る必要があります
このクラスは、インターフェイスなどを実装した Component クラスを継承していています

```
Object
   MarshalByRefObject
      Component
         Control

public class Control : Component, ISynchronizeInvoke, IWin32Window
```

このクラスは全てのウィンドウ、コントロールの基本となるクラスです
基底クラスの Component は、System.Windows.Forms 名前空間の全ての要素が持つ
簡単なインターフェイスなどを実装しただけの、ルートとなるクラスです
この Component クラスについては、今は特に説明することはありません

全てのコントロールは Control クラスを継承しているため
サイズや位置、有効や向こうなどの基本情報や制御は、このクラスのメンバで行えます
例えば、コントロールの長方形は **Control.Left**、**Control.Top**、
Control.Right、**Control.Bottom** プロパティなどで制御できます

public int Left {get; set;}
public int Top {get; set;}
public int Right {get; set;}
public int Bottom {get; set;}

Left はコントロールの左上隅の X 座標、Top は Y 座標
Right はコントロールの右下隅の X 座標、Bottom は Y 座標を指定します
子コントロールであれば、座標は親ウィンドウのクライアント座標となりますが
メインウィンドウの時のみ、スクリーン座標系で計算されます

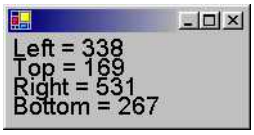
```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }

 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;

  g.DrawString("Left = " + Left ,
   new Font("MS Serif" , 14) , Brushes.Black , 0 , 0);
  g.DrawString("Top = " + Top ,
   new Font("MS Serif" , 14) , Brushes.Black , 0 , 15);
  g.DrawString("Right = " + Right ,
   new Font("MS Serif" , 14) , Brushes.Black , 0 , 30);
  g.DrawString("Bottom = " + Bottom ,
   new Font("MS Serif" , 14) , Brushes.Black , 0 , 45);

 }
}
```



このプログラムは、ウィンドウの左上の座標と右下の座標を表示するものです
ただし、ウィンドウを移動させても、移動に対する再描画イベントを設定していないので
ウィンドウを最小化するなどして、無効矩形を作らなければ OnPaint() が呼び出されません

同様に、コントロールの長方形を **Control.Bounds** プロパティからも得られます
この場合は、単純な数値ではなく Rectangle 型として入出力できるので便利です

public Rectangle Bounds {get; set;}

また、コントロールの座標ではなく単純な幅と高さをで制御したいのであれば
Control.Width と **Control.Height** プロパティがあります

public int Width {get; set;}
public int Height {get; set;}

Width はコントロールの幅を、Height はコントロールの高さを表しています
これらの値を変更することによって、コントロールのサイズを変更することができます

これまでの値はウィンドウ全体の幅や高さ、座標でしたが
メインウィンドウのようなものの場合、クライアント領域のサイズを知りたい場合もあるでしょう

作業領域のサイズは Control.ClientRectangle プロパティで取得できます

public Rectangle ClientRectangle {get;}

このプロパティを使えば、クライアント領域のサイズを取得することができるため
ウィンドウサイズに相対的な描画処理を記述することなどができるようになるでしょう

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }

 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  g.FillRectangle(Brushes.Red , ClientRectangle);
 }
}
```

このプログラムは、単純にクライアント領域全体を
FillRectangle() メソッドを用いて赤色で塗りつぶすというものです

デフォルト値

Control クラスは、静的なプロパティで
コントロールが定義するデフォルトの値をいくつか提供しています

デフォルトの背景色は Control.DefaultBackColor プロパティで
フォントは Control.DefaultFont プロパティ、
前景色は Control.DefaultForeColor プロパティで得られます

public static Color DefaultBackColor {get;}
public static Color DefaultFont {get;}
public static Color DefaultForeColor {get;}

描画処理などを行う時に、デフォルトの値を使用したい場合は
このプロパティから取得することによって、論理的にデフォルトであることを保証できます

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }

 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Brush brush = new SolidBrush(DefaultForeColor);
  g.DrawString("Kitty on your lap" , DefaultFont , brush , 0 , 0);
 }
}
```

このプログラムは、デフォルトの前景色とフォントを使って
ウィンドウのクライアント領域に文字列を描画するというものです

コントロールの状態

複雑なプログラムになると、コントロールの状態も重要な情報になります
後ほど説明する「マルチスレッド」と実現すると、プログラムは並列に実行されるため
必要なコントロールが作られる前に、別のコードが実行されるという問題も出てきます

こういった事態を防ぐには、コントロールの状態をチェックする必要が出てくるでしょう
コントロールが作られているかどうかは Control.Created プロパティを使います
また、Control.Disposing プロパティを使えば、破棄されているかを調べられます

public bool Created {get;}
public bool Disposing {get;}

Created は、コントロールが作成されていれば true を、そうでなければ false を返します
Disposing は、コントロールが破棄されるプロセスの中にあれば true
そうでなければ false が返ります

この他にも、コントロールの有効/無効を示す Control.Enabled や
入力フォーカスの有無をしめす Control.Focused プロパティもあります
有効/無効というのは、例えばチェックボックスコントロールなどで重要になります

public bool Enabled {get; set;}
public bool Focused {get; set;}

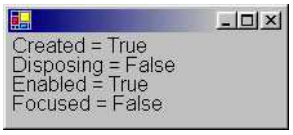
Enabled はコントロールが有効であれば true、そうでなければ false を
Focused は入力フォーカスを保有していれば true、そうでなければ false を返します

```
using System.Windows.Forms;
using System.Drawing;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }

 override protected void OnPaint(PaintEventArgs e) {
  Graphics g = e.Graphics;
  Font ft = new Font("MS Serif" , 12);

  g.DrawString("Created = " + Created , ft , Brushes.Black , 0 , 0);
  g.DrawString("Disposing = " + Disposing , ft , Brushes.Black , 0 , 15);
  g.DrawString("Enabled = " + Enabled , ft , Brushes.Black , 0 , 30);
  g.DrawString("Focused = " + Focused , ft , Brushes.Black , 0 , 45);
 }
}
```



このプログラムは、各プロパティの状態をウィンドウに表示します
イベントに対応していないので、リアルタイムの表示ではありませんが
ウィンドウを隠すなどして再描画すれば、最新の情報を取得することができます

前のページへ　戻る　次のページへ