1 Form Events

This section will show you the procedures to create a drawing area.

In the C++ Builder project environment, select the Form window with the form itself is selected. i.e. None of the component (if there exists) is selected.

Go to the Object Inspector window, click on the Events tab. Double click the value column of OnPaint. The value of OnPaint will be filled “FormPaint” automatically and Code window will now be focused as shown in Figure 1.
The OnPaint event is called every time the application is restored from Minimize, Resize, and when it receives WM_PAINT message from the Windows OS. There are other events for Form object, such as OnResize and OnClick, while the behaviour varies according to the event natures. For simplicity, we will do the rest of the coding with the OnPaint event.

2. Drawing on Canvas

Canvas is where we draw the graphics. There are number of helper classes in Canvas class, which helps you do drawing with ease. They are:

**Pen:** It is used to define the colour, thickness mode, and style (such as solid, dotted, and dashed).

**Brush:** It defines the colour to fill in an object.

The following example illustrates how to draw a rectangle with canvas and its helper classes.

```cpp
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    Canvas->Brush->Color=clWhite;
    Canvas->Pen->Color=clBlack;
    Canvas->Rectangle(10,20,100,150); // Rectangle(x1,y1,x2,y2)
}
```

Compile and execute the project. The following result will be obtained.
To set the colour of Pen and Brush, we can use predefined system colour such as \texttt{clWhite}, \texttt{clBlack}, etc. An alternative way to set color is to use \texttt{RGB} function. Sample code in line drawing with \texttt{RGB} function is shown below:

```cpp
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    Canvas->Pen->Color=RGB(255,0,0); // RGB(rrr,ggg,bbb)
    Canvas->PenPos = TPoint(10,10); // start point of a line
    Canvas->LineTo(80,100); // end point of the line
}
```

Result is shown below: (a red line is drawn)

3. **Working with Image**

In this section, I will show you how to manipulate image data. First of all, add 2 image frames, detailed procedures please refer to supplementary notes on *Borland C++ Programming 1 – Media Component*. Now you should see something similar as shown below:
I am going to build an application as an example, in which I will show you the procedures in coding. This example loads an image file into Image1, the image is processed, and then display the result in Image2.

The following code segment loads an image file. In this sample code, `bm1` and `bm2` are the pointers to bitmaps of Image1 and Image2 respectively.

Note that in Borland C++ version 6, the executable binary sends a WM_PAINT message to the OS once an image is loaded or updated. This will result in undesirable effect. One simple way to get around the problem is to put the code in the OnActivate event instead of OnPaint. Please refer to step 1 for detailed procedures.

```cpp
void __fastcall TForm1::FormActivate(TObject *Sender) {
    Graphics::TBitmap *bm1 = Image1->Picture->Bitmap;
    Graphics::TBitmap *bm2 = Image2->Picture->Bitmap;
    bm2->PixelFormat = pf24bit;
    bm1->LoadFromFile("C:\mt888\lena.bmp");
}
```
As we are NOT going to load an image file into Image2, Image2 will not contain any information. Therefore, we need to set the `PixelFormat` in the code.

Now set the size of `bm2` to match with `bm1`. There are 2 parameters: `bm->Width / bm->Height` and `Image->Width / Image->Height`. The `bm->Width` and `bm->Height` information defines the bitmap area for which the actual bitmap image data is held. The `Image->Width` and `Image->Height` information defines the display size in the `Form`.

```cpp
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    Graphics::TBitmap *bm1 = Image1->Picture->Bitmap;
    Graphics::TBitmap *bm2 = Image2->Picture->Bitmap;
    bm2->PixelFormat = pf24bit;
    bm1->LoadFromFile("C:\mt888\lena.bmp");
    // Set bm2 size to match with bm1
    bm2->Width = bm1->Width;
    bm2->Height = bm1->Height;
    // Update display size of Image2
    Image2->Width = Image1->Width;
    Image2->Height = Image1->Height;
}
```

In this example, the application inverts the color information of each pixel in the source image.

```cpp
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    Graphics::TBitmap *bm1 = Image1->Picture->Bitmap;
    Graphics::TBitmap *bm2 = Image2->Picture->Bitmap;
    bm2->PixelFormat = pf24bit;
    bm1->LoadFromFile("C:\mt888\lena.bmp");
    bm2->Width = bm1->Width;
    bm2->Height = bm1->Height;
    Image2->Width = Image1->Width;
    Image2->Height = Image1->Height;
    // Image Processing Starts Here
    BYTE* LinePtr1; // Line Pointer of Image1
    BYTE* LinePtr2; // Line Pointer of Image2
    for (int y = 0; y < bm1->Height; y++) {
        // Read a line of source image
        LinePtr1 = (Byte *)bm1->ScanLine[y];
        // Line to write the result image
        LinePtr2 = (Byte *)bm2->ScanLine[y];
        for (int x = 0; x < bm1->Width; x++) {
            // Invert Color for each pixel
```
```cpp
LinePtr2[x*3] = 255-LinePtr1[x*3]; //blue
LinePtr2[x*3+1] = 255-LinePtr1[x*3+1]; //green
LinePtr2[x*3+2] = 255-LinePtr1[x*3+2]; //red
}
}
```