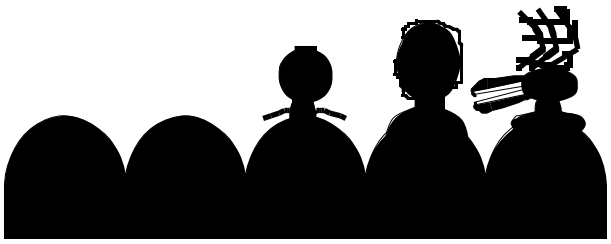

C++ Application Framework Use

Kent Sandvik/DTS

A Pattern approach into how to use the Application framework.

Kent Sandvik/DTS
May 20, 2025

INTRODUCTION

This document decribes case studies how to use the DTS supplied simple C++ shell for quickly writing applications.

CASE 1 - START AN APPLICATION AND LET IT RUN

This is the case where all the end user wants is to kick alive a simple application with a window that shows something by default:

```
#ifndef _APPLICATION_
#include "Application.h"
#endif


// M A I N    F U N C T I O N
void main(void)
{
      TMyApplication * myApp = new TMyApplication;
      myApp->Start();
}
```

Now, this does not do much. We want to add in behavior -- in other words our own code -- in order to achieve something using the framework. However this example shows how simple it is to create an instance of a `TApplication` class. This will also open up a default window with nothing inside.

CASE 2 - WRITE QUICKDRAW STATEMENTS INSIDE WINDOW

Now we want to draw something inside he default window:

```
#ifndef _APPLICATION_
#include "Application.h"
#endif

const short H = 10;
const short V = 20;
const short DELTA = 12;

// This is a simple test, draw something in the default window port.
class TMyApplication : public TGUIApplication
{
public:
      virtual void Draw();
};
```

```
#pragma segment Application
void TMyApplication::Draw()
{
      // Draw something into the default window.

      PenNormal();      // Set font and font size
      TextFont(geneva);
      TextSize(9);
      TextFace(bold);

      MoveTo(H, V);
      DrawString("\pTEST1: If you want to terminate");
      MoveTo(H, V + DELTA);
      DrawString("\pthe program you need to click");
      MoveTo(H, V + 2 * DELTA);
      DrawString("\pin the window close box!");

      PenNormal();      // restore pen
}

// M A I N    F U N C T I O N
void main(void)
{
      TMyApplication * myApp = new TMyApplication;
      myApp->Start();
}
```

As seen this is done by overriding `TApplication.Draw` so we could define what is happening inside the `Draw` member function. Note that the member function will draw when we switch between contexts, as well as when we move windows over the drawing area.

CASE 3 - DEFINE OUR OWN WINDOW HANDLING

In this case we want to control how the window is behaving, and we also want to control how the framework creates windows or documents (we assume that the earlier header information is used):

```
// In this test we will override the CreateNewDocument class and use our //
own special TWindow.


class TMyApplication : public TGUIApplication
{
public:
      virtual void DoCreateDocument();
};


class TMyWindow : public TWindow
{
public:
```

```
      virtual void Draw();
};


void TMyApplication::DoCreateDocument()
{
      // Create this time my TMyWindow document, and add it to the
TApplication list.
      TMyWindow * aWindow = new TMyWindow;
      this->AddDocument(aWindow);
}


void TMyWindow::Draw()
{
      // Draw something into my special window.

      PenNormal();      // Set font and font size
      TextFont(geneva);
      TextSize(9);
      TextFace(bold);

      MoveTo(H, V);
      DrawString("\pTEST2: If you want to terminate");
      MoveTo(H, V + DELTA);
      DrawString("\pthe program you need to click");
      MoveTo(H, V + 2 * DELTA);
      DrawString("\pin the window close box!");

      PenNormal();      // restore pen
}


// M A I N   F U N C T I O N
void main(void)
{
      TMyApplication * myApp = new TMyApplication;
      myApp->Start();
}
```

The things to note are:
• We override `TApplication.DoCreateDocument` so we are able to control that the new `TMyWindow` class is created and added to the framework that keeps track of windows.
•We create a new `TWindow` and override `TWindow.Draw` with our own drawing information.

CASE 4- GRAPHICSENVIRONMENTS

The next components are used for defining the printing environment, color, pen size and fonts. They could be used to pre-define styles, and apply them when using QuickDraw. Here's an example concerning color:

```
TColorEnvironment myMetalGray(36000,
                              40500,
                              37500);
TColorEnvironment myMetalBlue(26312,
                              14340,
                              47359);

#pragma segment Window
void TMyWindow::Draw()
{
      // Paint background in metal color
      myMetalGray.SetForeground();  // set gray foreground
      Rect windowRect = this->GetExtent();
      // get window extent

      ::PaintRect(&windowRect);     // paint background with gray

      // Set real background & foreground colors
      myMetalBlue.SetForeground();
      // set blue paint foreground
      myMetalGray.SetBackground();
      // set gray background
```

If you want to go back to the original values (black&white), calle the Reset member function.

Here's an example where Fontenvironments are used:

```
TFontEnvironment myGenevaBold(srcOr,
                              geneva,
                              9,
                              bold);
TFontEnvironment myGenevaInvBold(notSrcCopy,
                                 geneva,
                                 10,
                                 bold);

#pragma segment Window
void TMyWindow::Draw()
{
      // Draw something into my special window.
      ::MoveTo(H, V);
      myGenevaBold.Set();
      ::DrawString("\pThis is a color and Quickdraw test.");

      ::MoveTo(H, V + DELTA);
      myGenevaInvBold.Set();
      ::DrawString("\pI will use various classes, do");

      ::MoveTo(H, V + 2 * DELTA);
      myGenevaBold.Reset();   // reset font values to default state
      ::DrawString("\pthey they work or not?");
}
```

Finally we have a case where a pen line environment is defined and used:

```
TPenEnvironment myThickLine(srcOr,
                            qd. dkGray,
                            5,
                            5);

#pragma segment Window
void TMyWindow::Draw()
{

      ::MoveTo(H, V + 3 * DELTA);
      myThickLine.Set();      // draw thick line
      ::Line(200, 0);
      myThickLine.Reset();    // back to thin lines
      ::MoveTo(H, V + 4 * DELTA);
      ::Line(200, 0);
      ::MoveTo(H, V + 5 * DELTA);
      ::Line(200, 0);

      ::PenNormal();          // restore pen
}
```