Display mode with alpha channel on Symbian devices

Overview

Symbian C++ applications should be ready to handle a display mode (`TDisplayMode`) with the alpha channel defined (`EColor16MA` and `EColor16MAP`).

Detailed description

To implement low-level graphics operations, such as conversion from one format (colour depth) to another or drawing directly to the data address of an offscreen bitmap, one must be aware of the display mode (colour depth) and colour format used by the device. Possible display modes are listed in the `TDisplayMode` enumeration.

For current and upcoming Symbian devices with true colour displays, the following display modes are used:

- `EColor16MU` True colour display mode (32 bpp, but top byte is unused and unspecified)
- `EColor16MA` Display mode with alpha (32 bpp - 24bpp colour plus 8bpp alpha)
- `EColor16MAP` Display mode with alpha (32 bpp - 24bpp colour plus 8bpp alpha) - colour channels pre-multiplied with alpha value

Applications that are designed to run in `EColor16MU` display mode typically require no modifications to accommodate `EColor16MA` or `EColor16MAP` as the default mode used by the screen device. This is because the colour format and bits-per-pixel length are identical in all modes. The only difference is that the top 8 bits unused in `EColor16MU` are used for the alpha channel in the two other modes. However, this should be taken into account when handling `TDisplayMode` values in the code.

Solution

To ensure that the application works on devices with the `EColor16MU` default display mode as well as on devices with the `EColor16MA` display mode, **avoid** using the following type of code when handling `TDisplayMode` values:

```
  switch( iWindow.DisplayMode() )
    {
    case( EColor4K ) :  { bitsPerPixel = 12; break; }
    case( EColor64K ):  { bitsPerPixel = 16; break; }
    case( EColor16M ):  { bitsPerPixel = 24; break; }
    case( EColor16MU ): { bitsPerPixel = 32; break; }
    default: { Panic( EPanicUnsupportedDisplayMode ); } // Error: Panics if display mode
is EColor16MA(P)!
    }
```

Instead, the code should be written as follows:

```
  switch( iWindow.DisplayMode() )
    {
    case( EColor4K ) : { bitsPerPixel = 12; break; }
    case( EColor64K ): { bitsPerPixel = 16; break; }
    case( EColor16M ): { bitsPerPixel = 24; break; }
    // Valid for EColor16MU, EColor16MA, EColor16MAP*
    default: bitsPerPixel = 32;
    }
```

Similarly, for code that uses display mode value in managing bitmaps (`CFbsBitmap`) and/or graphics contexts (`CFbsBitGc`), display modes with alpha channels can be treated as `EColor16MU` if there's no need for transparency:

```
iDisplayMode = iCoeEnv->ScreenDevice()->DisplayMode();
if ( iDisplayMode == EColor16MA || iDisplayMode == EColor16MAP )
  {
  iDisplayMode = EColor16MU;
  }
...
```

* Note however that to get correct transparency results when directly manipulating pixel data in bitmaps with `EColor16MAP` mode, each colour component must be premultiplied with the alpha value.