How to display splash screen in a non GUI thread in Symbian C++

There are lot of times when before the application can start and displaying the view, we need to do a lot of background processing/initializations from the main UI thread itself. This slows down/delays loading of the main view of the application and the user more often then not ends up believing that the program is has either hanged or is about to crash/cause a malfunction. Either of the scenarios are not good from a usability perspective as we always want to keep the user informed as to what is happening at that point in time.

Symbian signing test case UNI-01(Installation, Normal and Stressed Usage) also stresses that the application should start up normally and in case it is taking more time then normal the user should be notified.

For more details check : How to conform with Symbian Signed criteria#Application startup times .28UNI-01.29 Symbian Sign article on wiki and Symbian Signed Test Criteria

In some cases it is not possible to proceed further without initializing/loading the other components of the application, and the same has to be done in the main UI thread itself. In those cases we need to implement a splash screen of sorts in a separate thread which is displayed till the time the application is ready to load the main application page.

The example below explains how to create a splash screen in a non GUI thread which is spawned from the main GUI thread and is destroyed once the main thread is ready to show the application view.

The .h file for the appui, the appui is the class from where we will spawn the thread to create the splash screen. Right now no activity really happens in the appui other then creating the thread however in a real time scenario, you can create the splash thread and keep it running till all the background loading/initialization in the main thread is complete and you are ready to show the main view to the user.

```
#ifndef MULTITHREADED_SPLASHSCREENAPPUI_H
#define MULTITHREADED_SPLASHSCREENAPPUI_H

#include <aknviewappui.h>
#include "splashactive.h"

class CMultiThreaded_SplashScreenContainerView;

/**
 * @class CMultiThreaded_SplashScreenAppUi MultiThreaded_SplashScreenAppUi.h
 * @brief The AppUi class handles application-wide aspects of the user interface,
including
 *        view management and the default menu, control pane, and status pane.
 */
class CMultiThreaded_SplashScreenAppUi : public CAknViewAppUi, public MWaitObserver
 {
public:
 // constructor and destructor
 CMultiThreaded_SplashScreenAppUi();
 virtual ~CMultiThreaded_SplashScreenAppUi();
 void ConstructL();

public:
 // from CCoeAppUi
 TKeyResponse HandleKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType );

 // from CEikAppUi
 void HandleCommandL( TInt aCommand );
 void HandleResourceChangeL( TInt aType );
```


(C) Copyright Nokia 2014. All rights reserved.

```
 // from CAknAppUi
 void HandleViewDeactivation(const TVwsViewId& aViewIdToBeDeactivated,const TVwsViewId&
aNewlyActivatedViewId );

private:
 void InitializeContainersL();
 void ConstructIconsL();
 // [[[ begin generated region: do not modify [Generated Methods]
public:

 void ThreadNotify();
 void DoReDraw();

private:
 CMultiThreaded_SplashScreenContainerView* iMultiThreaded_SplashScreenContainerView;
 CSplashActive* iActive;
 TInt iCounter;
 //icons for the splash screen
 CFbsBitmap*       iSplashIcon;
 CFbsBitmap*       iSplashIcon_mask;
 RArray<TInt>      iIconHandle;
 };

#endif // MULTITHREADED_SPLASHSCREENAPPUI_H
```

The .cpp file for the appui, the iActive is the thread that we have created which would in turn display the splash screen. The other functions have been omitted for clarity sake and can be had from the attached zip file. The constructicons function allows the user to create a basic splash icon from a mif file which is loaded into the private directory of the application. The ReDraw and ThreadNotify are the main functions to look for which are called from the iActive or the splash thread to update the main thread and notify the latter of the state of the splash thread. The logic to handle the spawned thread's run time can be altered to let the main thread handle it.

```
_LIT(KMBMFile,"SplashDemo.mif");

void CMultiThreaded_SplashScreenAppUi::ConstructL()
 {
 BaseConstructL( EAknEnableSkin );
 InitializeContainersL();
 ConstructIconsL();
 iActive = CSplashActive::NewL(*this, &iIconHandle); // The splash thread which would in
turn display the splash screen
 iActive->IssueRequest(0); // issue the request to start drawing the splash screen from
here
 // After this your main thread is free and you can do what ever you want
 // and then notify the spawned thread to stop when you are ready in the main thread
 }

void CMultiThreaded_SplashScreenAppUi::ConstructIconsL()
 {
    // create and open file server session
 RFs fileSession;
 TBuf<KMaxFileName>  completefilename;
```

 (C) Copyright Nokia 2014. All rights reserved.

```
    User::LeaveIfError(fileSession.Connect());
    fileSession.ShareProtected();

    // set path of the icon files
  User::LeaveIfError(fileSession.PrivatePath(completefilename));

    // append the MBM file name to the private path
  completefilename.Append(KMBMFile);

    // insert the drive to the private path
    TParsePtrC parse((CEikonEnv::Static()->EikAppUi()->Application())->AppFullName());
    completefilename.Insert(0, parse.Drive());

    // Load bitmaps from the resource.
    CIconFileProvider* iconProvider = CIconFileProvider::NewL(fileSession,
completefilename);

 AknIconUtils::CreateIconL(iSplashIcon, iSplashIcon_mask, *iconProvider,
EMbmSplashdemoSplash, EMbmSplashdemoSplash+1);
 AknIconUtils::SetSize(iSplashIcon, TSize(ApplicationRect().Width()
,ApplicationRect().Height()));
 AknIconUtils::SetSize(iSplashIcon_mask, TSize(ApplicationRect().Width()
,ApplicationRect().Height()));

 // for splash screen
 //Append the handles to the icon and the mask so that we can pass it to the Splash
thread
 iIconHandle.Append(iSplashIcon->Handle()); // We need to pass them because a non GUI
thread will not be able to use AknIconUtils
 iIconHandle.Append(iSplashIcon_mask->Handle());
 }


/**
 * Thread functioning if complete, time to cleanup
 */
void CMultiThreaded_SplashScreenAppUi::ThreadNotify()
 {
 if(iActive)
  {
  delete iActive;
  iActive = NULL;
  }
 if(iSplashIcon)
  {
  delete iSplashIcon;
  iSplashIcon = NULL;
  }
 if(iSplashIcon_mask)
  {
  delete iSplashIcon_mask;
  iSplashIcon_mask = NULL;
  }
 iIconHandle.Reset();
 }
```

 (C) Copyright Nokia 2014. All rights reserved.

```
/**
 * Keep redrawing the splash screen again
 */
void CMultiThreaded_SplashScreenAppUi::DoReDraw()
 {
 iCounter++;
 // Right now the counter value is being driven in the splash thread but depending upon
your
 // requirements you can alter this logic to let the main thread drive this counter
 iActive->IssueRequest(iCounter);
 if(iCounter == 10)
  {
  iCounter = 0;
  }
 }
```

The splash thread's .h file which owns an instance of the splash screen which will be used to draw the splash screen.

```
#include "SplashScreen.h"

class CSplashActive;
class MWaitObserver;

class CSplashActive : public CActive
{
public:
 CSplashActive(MWaitObserver& aMyObserver);
 ~CSplashActive();
public:

 void IssueRequest(TInt aProCount);
 void DoCancel();
 void RunL();
 void ConstructL(RArray<TInt>* aBitmapHandle);
 void ThreadSuspend();

public:
 static CSplashActive* NewL(MWaitObserver& aMyObserver, RArray<TInt>* aBitmapHandle);
public:

 TRequestStatus iMyStatus;
 MWaitObserver& iMyObserver;

 RThread actThread;
 RThread iThisThread;
 TInt iCounter;
 CSplashScreen* iSplashScreen;

static TInt MyThread(TAny* aPkg);

};

class MWaitObserver
```

 (C) Copyright Nokia 2014. All rights reserved.

```
{
public:
 virtual void ThreadNotify() = 0;
 virtual void DoReDraw() = 0;
};
```

The cpp of the splash thread class.

```
#include "splashactive.h"

CSplashActive* CSplashActive::NewL(MWaitObserver& aMyObserver, RArray<TInt>*
aBitmapHandle)
 {
 CSplashActive* self=new (ELeave) CSplashActive(aMyObserver);
 self->ConstructL(aBitmapHandle); //bitmaps that were created in the main GUI thread
 return self;
 }

void CSplashActive::ConstructL(RArray<TInt>* aBitmapHandle)
 {
    iSplashScreen = new(ELeave) CSplashScreen(aBitmapHandle);
 }

CSplashActive::CSplashActive(MWaitObserver& aMyObserver)
    :CActive(CActive::EPriorityStandard),
     iMyObserver(aMyObserver)

 {
 CActiveScheduler::Add(this);
 iThisThread.Open(RThread().Id());
 actThread.Create(_L("SplashThread"),CSplashActive::MyThread, KDefaultStackSize,
NULL,this); //create the thread that will do the splash screen drawing
 }

CSplashActive::~CSplashActive()
 {
    actThread.Terminate(KErrNone);
 Cancel();

 if(iSplashScreen)
  {
  delete iSplashScreen;
  iSplashScreen = NULL;
  }
 }


void CSplashActive::ThreadSuspend()
 {
  TRequestStatus* requestStatus = &(iStatus);
 iThisThread.RequestComplete(requestStatus, KErrGeneral);
  actThread.Suspend();
 }
```

 (C) Copyright Nokia 2014. All rights reserved.

```
void CSplashActive::DoCancel()
 {
 }


void CSplashActive::IssueRequest(TInt aCounter)
 {
 iCounter = aCounter;
 iStatus = KRequestPending;
 SetActive(); // Will cause the RunL to be called once the thread has been resumed
 actThread.Resume(); //resume the suspended thread
 }


void CSplashActive::RunL()
 {
// Right now the splash screen runs for 10 seconds you can consider
// modifying the same based on inputs from the main thread
 if(iStatus == KErrNone && iCounter < 10)
  {
  iSplashScreen->Show(); //Show the splash screen from here
  iMyObserver.DoReDraw(); //Notify the user to take action
  }
 else if(iStatus == KErrGeneral)
  {

  }
 else
  {
  iMyObserver.ThreadNotify(); // thread completed execution notify the main thread so
that it can cleanup
  }
 }

TInt CSplashActive::MyThread(TAny* aPkg)
 {
 RThread thisThread;
 while(ETrue)
  {
  CSplashActive* active = static_cast<CSplashActive*>(aPkg);
  User::After(1*1000*1000); //1 Second callback right now
  TRequestStatus* requestStatus = &(active->iStatus);
  (active->iThisThread).RequestComplete(requestStatus, KErrNone);
  thisThread.Suspend(); //puts the thread in suspended state and it will be resumed from
the issuerequest call.
  }
 }
```

The actual splash screen's .h file.

```
class CSplashScreen
{
```

 (C) Copyright Nokia 2014. All rights reserved.

```
public:

 CSplashScreen(RArray<TInt>* aAgitoBitmapHandle);
 ~CSplashScreen();
 void Show();
 void Refresh();
 void StopSplashRefresh();
private:
 RWsSession iWs;
 RWindowGroup iWg;
 CWindowGc* iGc;
 RWindow iWindow;
 CWsScreenDevice* iScreenDevice;
 RArray<TInt>* iAgitoBitmapHandle;
};
```

The splash screen implementation which draws the splash in a non Gui thread by creating the graphic context and using the raw window.

```
#include "SplashScreen.h"

/*
 * CSplashScreen::CSplashScreen(RArray<TInt>* aAgitoBitmapHandle)
 * The aAgitoBitmapHandle contains handles to the icon and the mask
 */
CSplashScreen::CSplashScreen(RArray<TInt>* aBitmapHandle)
 {

 User::LeaveIfError(iWs.Connect());
 iScreenDevice = new(ELeave) CWsScreenDevice(iWs);
 iScreenDevice->Construct();
 TPixelsTwipsAndRotation curPixTwipsRot;
 iScreenDevice->GetDefaultScreenSizeAndRotation(curPixTwipsRot);
 TRect screenRect = curPixTwipsRot.iPixelSize;

 iWg = iWs;
 User::LeaveIfError(iWg.Construct(reinterpret_cast<TUint32>(&iWg), EFalse));
 iWg.SetOrdinalPosition(10, ECoeWinPriorityAlwaysAtFront);

 User::LeaveIfError(iScreenDevice->CreateContext(iGc));

 iWindow = iWs;
 User::LeaveIfError(iWindow.Construct(iWg, reinterpret_cast<TUint32>(&iWg) + 1));
 iWindow.SetBackgroundColor(TRgb(0xff, 0xfa, 0xfa));
 iWindow.Activate();
 iWindow.SetExtent(TPoint(0, 0), TSize(screenRect.Width(), screenRect.Height()));
 iWindow.SetVisible(ETrue);

 iBitmapHandle = aBitmapHandle;
 }

CSplashScreen::~CSplashScreen()
 {
```

 (C) Copyright Nokia 2014. All rights reserved.

```
 iWindow.Close();
 iWg.Close();

 if(iGc)
  {
  delete iGc;
  iGc = NULL;
  }

 if(iScreenDevice)
  {
  delete iScreenDevice;
  iScreenDevice = NULL;
  }
 iWs.Close();
 iBitmapHandle->Reset();
 }


/*
 * void CSplashScreen::Show()
 * Function that begins the redraw and needs to be called only once to avoid flickering
 */
void CSplashScreen::Show()
{
 iGc->Activate(iWindow);
 TRect rect = TRect(iWindow.Size());
 iWindow.Invalidate(rect);
 iWindow.BeginRedraw(rect);

 iGc->SetBrushStyle(CGraphicsContext::ESolidBrush);
 iGc->Clear();
 Refresh();
}

/**
 * Function to refresh the view repeatedly so that the welcome screen holds for sometime
 */
void CSplashScreen::Refresh()
 {
 TRect rect = TRect(iWindow.Size());
 //create the bitmap for this thread using the handle passed. No need to clean up as it
creates only a pointer to the existing icon
 CFbsBitmap* bitmap = new (ELeave) CFbsBitmap();
 bitmap->Duplicate((*iBitmapHandle)[0]);
 //create the bitmap mask for this thread using the handle passed. No need to clean up
as it creates only a pointer to the existing icon
 CFbsBitmap* bitmap_mask = new (ELeave) CFbsBitmap();
 bitmap_mask->Duplicate((*iBitmapHandle)[1]);
 //Draw the icon from the center of the screen
 iGc->BitBltMasked(TPoint(0, 0), bitmap, rect, bitmap_mask, ETrue);
 iGc->DrawRect(TRect(TPoint(50,100), TSize(40,40)));
 StopSplashRefresh();
 }

/**
 * Called to end the redraw and flush the Window Server
```

 (C) Copyright Nokia 2014. All rights reserved.

```
 */
void CSplashScreen::StopSplashRefresh()
 {
 iWindow.EndRedraw();
 iGc->Deactivate();
 iWs.Flush();
 }
```

Using the above code snippet we can draw a splash screen from a non GUI thread thereby freeing up the main thread which can do the other initializations before the main view is ready to be displayed. Since the splash screen is being displayed the user would not be worried about what is going on as s/he would feel that something that he doesn't really need to know at the moment is happening, but since the application per se is responsive in the sense of displaying a splash which is continuously updating it would keep the user involved and would give him an indication to wait.

Such an approach would not only enhance the usability figures of the application but also enable you to pass the Symbian signing criteria.

Download a working S60 3rd Edition, FP1 example from: File:MultiThreaded SplashScreen.zip

 (C) Copyright Nokia 2014. All rights reserved.