



The active idle remains somewhat the sacrosanct space of the device so to speak. The API's to fiddle with the active idle are not available to 3rd party developers and can be had only after opening a partnering case with Nokia.

However there are lot of times when you absolutely must want to play with the active idle in order to provide more visibility to your application or sometimes to display content when the phone is in idle mode i.e. no calls, no user activity per se and the screen on top is the active idle.

One such instance is if we want to display a ticker, within our application and more so on the active idle screen. Though it is not directly possible using any published API but then you can always take the route of drawing directly on the screen. Please see links below for more such instances of the same.

In case there is an incoming call or the user presses any keys to invoke the telephone application or navigates to any other application other than the standby or his/her own application the ticker window is hidden so that it doesn't hamper any user experience and/or functionalities.

The code snippet below details the process of creating a simple working ticker on the active idle screen.

The AppUi.h file looks something like this :-

```
/*
=====
Name          : TickerDemoAppUi.h
Author        : mayank
Copyright     : Your copyright notice
Description   :
=====
*/
#ifndef TICKERDEMOAPPUI_H
#define TICKERDEMOAPPUI_H

// [[[ begin generated region: do not modify [Generated Includes]
#include <aknviewappui.h>
// ]]] end generated region [Generated Includes]

// Observer which is implemented in the appui.cpp
#include "WsChangeObserver.h"

// [[[ begin generated region: do not modify [Generated Forward Declarations]
class CTickerDemoContainerView;
class CWsWatcherActive;
// ]]] end generated region [Generated Forward Declarations]

/**
 * @class          CTickerDemoAppUi TickerDemoAppUi.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 CTickerDemoAppUi : public CAknViewAppUi, public MWsChangeObserver
{
public:
// constructor and destructor
CTickerDemoAppUi();
virtual ~CTickerDemoAppUi();
void ConstructL();

```

## How\_to\_display\_ticker\_on\_active\_idle

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

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

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

// From MWSChangeObserver
void NotifyViewChangeL(TKeyCaptureMode aCaptureMode);

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

public:
// ]]] end generated region [Generated Methods]
void SetTickerMoveInterval();
static TInt TickerMove(TAny* aObject); // Call back function
void MoveTicker();

// [[[ begin generated region: do not modify [Generated Instance Variables]
private:
    CTickerDemoContainer& iTickerView;
    CWsWatcherActive      ;          iWsWatcherActive

// Timer
    CPeriodicTimerMoveTimer;
    TInt iCounter
    TBool iDoShowTicker
    TInt iDuration
    TInt iCategory
// ]]] end generated region [Generated Instance Variables]
// [[[ begin [User Handlers]
protected:
// ]]] end [User Handlers]
};

#endif // TICKERDEMOAPPUI_H
```

The basic .cpp file for the AppUi :-

```
/*
=====
Name      : TickerDemoAppUi.cpp
Author    : mayank
Copyright : Your copyright notice
Description :
=====
*/
// [[[ begin generated region: do not modify [Generated System Includes]

#include <eikmenub.h>
#include <akncontext.h>
```

## How\_to\_display\_ticker\_on\_active\_idle

```
#include <akntitle.h>
#include <TickerDemo.rsg>
// ]]] end generated region [Generated System Includes]

// [[[ begin generated region: do not modify [Generated User Includes]
#include "TickerDemoAppUi.h"
#include "TickerDemo.hrh"
#include "TickerDemoContainerView.h"
#include "WsWatcherActive.h"
// ]]] end generated region [Generated User Includes]

// [[[ begin generated region: do not modify [Generated Constants]
// ]]] end generated region [Generated Constants]

/**
 * Construct the CTickerDemoAppUi instance
 */
CTickerDemoAppUi::CTickerDemoAppUi()
    :iCounter(0),
      (ETrue)          iDoShowTicker
{
// [[[ begin generated region: do not modify [Generated Contents]
// ]]] end generated region [Generated Contents]
}

/**
 * The appui's destructor removes the container from the control
 * stack and destroys it.
 */
CTickerDemoAppUi::~CTickerDemoAppUi()
{
// [[[ begin generated region: do not modify [Generated Contents]
// ]]] end generated region [Generated Contents]
if(iWsWatcherActive)
{
delete iWsWatcherActive;
      = NULL;iWsWatcherActive
}
}

// [[[ begin generated function: do not modify
void CTickerDemoAppUi::InitializeContainersL()
{
    iTickerDemoContainerView = CTickerDemoContainerView::NewL();
    AddView(iTickerDemoContainerView);
    SetDefaultView(iTickerDemoContainerView);
}
// ]]] end generated function

/**
 * Handle a command for this appui (override)
 * @param aCommand command id to be handled
 */
void CTickerDemoAppUi::HandleCommandL( TInt aCommand )
{
// [[[ begin generated region: do not modify [Generated Code]
    TBool commandHandled;
switch ( aCommand )
{ // code to dispatch to the AppUi's menu and CBA commands is generated here
default:
break;
}
}
}
```

## How\_to\_display\_ticker\_on\_active\_idle

```
if ( !commandHandled )
{
if ( aCommand == EAknSoftkeyExit || aCommand == EEikCmdExit )
{
    ( );          Exit
}
}
// ]]] end generated region [Generated Code]

}

/**
 * Override of the HandleResourceChangeL virtual function
 */
void CTickerDemoAppUi::HandleResourceChangeL( TInt aType )
{
    CAknViewAppUi::HandleResourceChangeL( aType );
// [[[ begin generated region: do not modify [Generated Code]
// ]]] end generated region [Generated Code]
if ( aType == KEikDynamicLayoutVariantSwitch )
{
if(iTickerDemoContainerView)
{
// To ensure that change from landscape to portrait and vice versa scales up the UI correctly
if(HasDynamicLayoutVariantSwitch)
}
}
}

/**
 * Override of the HandleKeyEventL virtual function
 * @return EKeyWasConsumed if event was handled, EKeyWasNotConsumed if not
 * @param aKeyEvent
 * @param aType
 */
TKeyResponse CTickerDemoAppUi::HandleKeyEventL(
const TKeyEvent& aKeyEvent,
    TInt aType
)
{
// The inherited HandleKeyEventL is private and cannot be called
// [[[ begin generated region: do not modify [Generated Contents]
// ]]] end generated region [Generated Contents]

return EKeyWasNotConsumed;
}

/**
 * Override of the HandleViewDeactivation virtual function
 *
 * @param aViewIdToBeDeactivated
 * @param aNewlyActivatedViewId
 */
void CTickerDemoAppUi::HandleViewDeactivation(
const TVwsViewId& aViewIdToBeDeactivated,
const TVwsViewId& aNewlyActivatedViewId )
{
    CAknViewAppUi::HandleViewDeactivation(
        aViewIdToBeDeactivated,
        aNewlyActivatedViewId
    );
// [[[ begin generated region: do not modify [Generated Contents]
```

## How\_to\_display\_ticker\_on\_active\_idle

```
// ]]] end generated region [Generated Contents]

}

/**
 * @brief Completes the second phase of Symbian object construction.
 * Put initialization code that could leave here.
 */
void CTickerDemoAppUi::ConstructL()
{
// [[[ begin generated region: do not modify [Generated Contents]

    BaseConstruct(EActiveEnableSkin );

    InitializeContainersL

// To get notified of screen change
    iWsWatcher = CActiveWatcherActive::NewL(*this);
    iWsWatcher->Start();

    SetTickerMoveInterval

// ]]] end generated region [Generated Contents]

}

//Notifies of view change using which we decide to show or not show the ticker
void CTickerDemoAppUi::NotifyViewChangeL(TKeyCaptureMode aDoCapture)
{
    iDoShowTicker = EFalse;
    switch(aDoCapture)
    {
    case EGenericCapture:
    {
        iDoShowTicker = ETrue; iDoShowTicker
        iShowTickerOnInactiveView(iCounter, iDuration, ETrue);
    }
    break;
    case ENoCapture:
    {
        iHideTickerOnContainerView
    }
    }
}

// -----
// void CTickerDemoAppUi::SetTickerMoveInterval
// Reads the periodic interval and sets the callback
// -----
//
void CTickerDemoAppUi::SetTickerMoveInterval()
{
    const TInt iTickInterval = 200000; // After every .2 seconds
    iTickerMoveTimer = CPeriodic::NewL(CActive::EPriorityStandard); // neutral priority
    iTickerMoveTimer->Start(iTickInterval, iTickInterval, TCallBack(TickerMove, this));
}

// -----
// TInt CTickerDemoAppUi:: TickerMove (TAny* aObject)
// Call back function: CPeriodic Timer related functions
// -----
//
TInt CTickerDemoAppUi::TickerMove(TAny* aObject)
```

## How\_to\_display\_ticker\_on\_active\_idle

```
{
    TInt iRetVal
    ((CTickerDemoAppUi*)aObject)->MoveTicker();
    return iRetVal;
}

// -----
// CTickerDemoAppUi:: MoveTicker
// Call back function called on a periodic basis to update the ticker
// -----
//
void CTickerDemoAppUi::MoveTicker()
{
    if(iDoShowTicker)
    {
        iTickerDemoContainer->ShowTickerL();
    }
}
```

From the TickerDemoContainerView we are calling the container's wrapper functions which in turn is calling the actual ticker drawing/updating class as detailed here in a stripped down .cpp file.

```
void CTickerDemoContainerView::ShowTickerL(TInt aCategory, TInt aCounter, TInt aDuration, TBool aDoShow)
{
    if(iTickerDemoContainer)
    {
        iTickerDemoContainer->ShowTickerL(aCategory, aCounter, aDuration, aDoShow);
    }
}

void CTickerDemoContainerView::UpdateTickerL()
{
    if(iTickerDemoContainer)
    {
        iTickerDemoContainer->UpdateTickerL();
    }
}

void CTickerDemoContainerView::HideTicker()
{
    if(iTickerDemoContainer)
    {
        iTickerDemoContainer->HideTicker();
    }
}
```

The Container.cpp file, which owns an instance of the ticker as iTicker

```
void CTickerDemoContainer::ConstructL(
    const TRect& aRect,
    const CCoeControl* aParent,
    MEikCmdObsrvr )
{
    <10>TBuf<10> tempBuf(_L("ticker"));

    // initialize the ticker class
    iTicker = CTicker::NewL();
    iTicker->ShowTickerL(tempBuf, TServices_t(0));
    iTicker->HideTicker();
}
```

## How\_to\_display\_ticker\_on\_active\_idle

```
void CTickerDemoContainer::HandleScreenChangeL()
{
    if(iTicker)
        ->ScaleImage(iTicker); // Will scale the images if any to take into account the mode changes on t
    }

void CTickerDemoContainer::UpdateTickerL()
{
    ->UpdateTickerL(iTickerText);
}
}
```

### The actual ticker.h file

```
#ifndef TICKER_H_
#define TICKER_H_

// System Include
#include <aknview.h>
#include <aknview.h>
#include <akniconutils.h>
#define OFFSCREEN_DISPLAYMODE EColor64K
#define MAX_FILE_LEN 256

#include "iconfileprovider.h"

// FORWARD DECLARATIONS
class RWindowGroup;
class RWsSession;
class CWsScreenDevice;
class CWindowGc;
class RWindow;

typedef enum
{
    ESports,
    EEntertainment,
    EBusiness
}
TServices_t;

class CTicker : public CCoeControl
{
public:

    // @function NewL
    // // @discussion Construct a CTicker
    // @result a pointer to the created instance of CTicker
    static CTicker* NewL();

    // @function NewLC
    static CTicker* NewLC();

    // Destructor.
    virtual ~CTicker();

    //Set sizes for the icons based on screen size and other params
    void ScaleImage();
private: // Constructors
```

## How\_to\_display\_ticker\_on\_active\_idle

```

// C++ default constructor.
    ();      CTicker

// EPOC default constructor.
void ConstructL();
//Create icons from the .mif file
void ConstructIconsL();

public: // New functions

//used to Redraw the Status Window
void WinReDraw();

void ShowTickerL(const TDesC& aText, TServices_t aServices);

void UpdateTickerL(const TDesC& aText);

void HideTicker();

void UpdateText(const TDesC& aText);

private: // Functions from base classes

// From CCoeControl, Draw.
// @param Specified area for drawing

void Draw(const TRect& aRect) const;

//Draws a suitable background for the Ticker
void DrawBackgroundWindow();

void DisplayTickerText(const TDesC& aText);

void GetDrawableWindowSize();

private: // Data
    RWsSession          iWs
    *CWScreenServer    iScreen
    RWindowGroup        iGroup
    * ; CWindowGc        iGc
    RWindow              iWindow
    TBool                iFirstTime

    TSize                ; iCurrentDrawableWindowSize
    TBool                ; iScreenChanged
    TBool                ; iWindowNeedsRedraw
    RFs                  ; iFsSession
    TInt                ; iStartPos
    <255>                ; TBuf                iText
    <255>                ; TBuf                iOriginalText
    TServices_t          iServices // In case you want to show different categories the same can be used, el
};

#endif /*TICKER_H_*/

```

## How\_to\_display\_ticker\_on\_active\_idle

The actual ticker implementation code, find comments inline to understand what is happening where as far as the ticker is concerned.

```
// System Include
#include <akncontext.h>
#include <fbs.h>
#include <eikenv.h>
#include <avkon.hrh>
#include <aknview.h>
#include <APGWGNAM.H>
#include <APGTASK.H>
#include <aknsskininstance.h>
#include <aknscontrolcontext.h>
#include <aknsutils.h>
#include <aknsdrawutils.h>
#include <aknutils.h>
#include <aknlayoutfont.h>
#include <stringloader.h>
#include <EIKAPPUI.H>
#include <EIKAPP.H>
#include <eikbtgpc.h>

// User Include
#include "Ticker.h"
#include <aknfontcategory.hrh>
// ===== MEMBER FUNCTIONS =====

// -----
// CTicker::NewL()
// Two-phased constructor.
// -----
//
CTicker* CTicker::NewL()
{
    *CTicker CTicker::NewLC();
    CleanupStack();
return self;
}

// -----
// CTicker::NewLC()
// Two-phased constructor.
// -----
//
CTicker* CTicker::NewLC()
{
    *CTicker new (ELeave) CTicker();
    CleanupStack();
    ->ConstructL();
return self;
}

// -----
// CTicker::CMyDialog()
//
// C++ default constructor can NOT contain any code, that
// might leave.
// -----
//
CTicker::CTicker()
:iFirstTime(ETrue),
(EIState)EIdleChanged
```

## How\_to\_display\_ticker\_on\_active\_idle

```
void CWindowNeedsRedraw
{
}

// -----
// CTicker::ConstructL(const TRect& aRect)
//
// EPOC default constructor can leave.
// -----
//
void CTicker::ConstructL()
{
// Windows Session
    User::LeaveIfError(iWs.Connect());

// Screen
    iScreen = new(ELeave) CWsScreenDevice(iWs);
    iScreen->Construct();

    // Windows Group
    iGroup = RWindowGroup(iWs);
    User::LeaveIfError(iGroup.Construct(reinterpret_cast<TUint32>(&iGroup), EFalse));
    iGroup.SetOrdinalPosition(0, ECoeWinPriorityAlwaysAtFront); // Play with this priority depend
    // So that the key events dont get hogged by the ticker this is specially
    // important to ensure the One handed navigation to shift from one icon
    // to other on the active idle works fine
    iGroup.EnableReceiptOfFocus(EFalse);

    // Window Graphics
    User::LeaveIfError(iScreen->CreateContext(iGc));

// Window
    iWindow = RWindow(iWs);
    User::LeaveIfError(iWindow.Construct(iGroup, reinterpret_cast<TUint32>(&iGroup) + 1));
    iWindow.Activate();
    GetDrawableWindowSize
    ConstructIconsL();
}

// -----
// CTicker::~CMYDialog()
//
// Destructor
// -----
//
CTicker::~CTicker()
{
    CWndow;
    CIGroup;

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

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

    iWs.Close();
}
```

## How\_to\_display\_ticker\_on\_active\_idle

```
}

void CTicker::ConstructIconsL()
{
    // Create icons if any
}

// -----
// CTicker::Draw(const TRect& aRect) const
//
// Draw function.
// -----
//
void CTicker::Draw(const TRect& aRect) const
{
}

void CTicker::WinReDraw()
{
}

// -----
// CTicker:: ShowTickerL ()
//
// Show the ticker
// -----
//
void CTicker::ShowTickerL(const TDesC& aText, TServices_t aServices)
{
    // Drawing with gc and window
    TRect aRect = TRect::Static()->AppUiFactory()->ClientRect();

    GetDrawableWindow().GetSize(
        aRect);

    // Set the size and extent according to your requirements
    SetExtent(TPoint(20,20), TSize(100, 50));

    if(iWindowNeedsRedraw)
    {
        ->Activate(iWindow);

        = TRect(aWindow.Size());
        InvalidateWindow();
        BeginRedrawWindow();
        SetShadowWindowed(ETrue);
        ->SetBrushColor(TRgb(0x696969));
        ->SetBrushStyle(CGraphicsContext::ENullBrush);
        ->Clear();    iGc

    //Scale images to current resolution
    if(iScreenChanged)
    {
        = EFalse; if screen resolution change taken into account already no need to scale
        ();    ScaleImage
    }

    //Draw a suitable call window background
    = aServices
}

```

## How\_to\_display\_ticker\_on\_active\_idle

```
        DisplayTic@@@t;
        EndRedraw();
    ->Deact@@ivate();
        FlushW$;
}

/*
 * void CCallStatusWindow::ScaleImage()
 * Scales images to suit the screen width
 */
void CTicker::ScaleImage()
{
    // Scales the images if any to fit the screen size change coz of mode change
}

void CTicker::HideTicker()
{
    // Drawing with gc and window
    ->Acti@@ate(iWindow);

        iWn@@@date();
        BeginRedraw();

        S@@@windowable(EFalse);
    ->Cle@@@();

        EndRedraw();

    ->Deact@@ivate();
        FlushW$;
        iWindowNeedsRedraw;
}

void CTicker::UpdateText(const TDesC& aText)
{
    Zer@@@t.
        iOrigin@@@t.
    Append(aText);
        iOrigin@@@t(aText);
        Update@@@t;
}

/*
 * void GetDrawableWindowSize()
 * Checks the current resolution and gets the optimal size
 * for drawing in case the sizes are out of optimal screen sizes
 * towards both extremes
 */
void CTicker::GetDrawableWindowSize()
{
    //Get the current window resolution
    CWScreen@@@Dev = *( CEikonEnv::Static()->ScreenDevice() );
    TPixelsTwipsAndRotation curPixTwipsRot
        sc@@@defaultScreenSizeAndRotation(curPixTwipsRot); // Get screen sizes

    //curPixTwipsRot.iPixelSize; has the screen width and height you can use it to handle correct sca
}

/*
```

## How\_to\_display\_ticker\_on\_active\_idle

```

* void CTicker::DisplayTickerText(const TDesC& aText)
* Displays the ticker text after proper formatting wrt font and position
*/
void CTicker::DisplayTickerText(const TDesC& aText)
{
    CWScreenDevice* screenDev = *( CEikonEnv::Static()->ScreenDevice() );

    ->SetFillColor(TRgb(0x676767)); //Set to obtain a white colored font

    // Get a logical font to base our font on
    const CFont* logicalFont = AknLayoutUtils::FontFromId(EAknLogicalFontPrimaryFont); //logicalFont

    //Setup a font for large text for the single line text of dialpad
    TFontSpec fontSpecLargeFont->FontSpecInTwips();

    fontSpecLargeFont->Height = iWindow.Size().iHeight/2; // Get a font half the height of the window
    fontSpecLargeFont->Style.SetStrokeWeight( EStrokeWeightBold);
    fontSpecLargeFont->Style.SetPosture( EPostureUpright);

    //Setup a font for small text for the multi line text of dialpad
    TFontSpec fontSpecSmallFont->FontSpecInTwips();

    fontSpecSmallFont->Height = iWindow.Size().iHeight/4; //Get a font half the height of the window
    fontSpecSmallFont->Style.SetStrokeWeight( EStrokeWeightNormal);
    fontSpecSmallFont->Style.SetPosture( EPostureUpright);

    // Obtain new font
    CFont fontSmall;

    screenDev->GetFontInPixels((CFont*& )fontSmall, fontSpecSmall);
    TInt smallFontDispWidth = iWindow.Size().iWidth / fontSmall->WidthZeroInPixels();

    ->SetFont((CFont*) fontSmall);
    iStartPos = iWindow.Size().iWidth;
    ZeroText();
    Append(aText);
    iOrigin = iStartPos;
    iOrigin += Append(aText);
    ->DrawText(aText, TPoint(iStartPos, (iWindow.Size().iHeight*3/4) )); // Point at which ticker
    screenDev->SetFont(fontSmall);
}

/**
* Updates the text on the ticker periodically to give a moving effect
*/
void CTicker::UpdateTickerL(const TDesC& aText)
{
    // Drawing with gc and window
    ->Activate(iWindow);
    TRect rect(iWindow.Size());
    iWindow->Update(rect);
    iWindow->Draw(rect);
    ->Clear();

    CWScreenDevice* screenDev = *( CEikonEnv::Static()->ScreenDevice() );

    ->SetFillColor(TRgb(0x676767)); //Set to obtain a white colored font

    // Get a logical font to base our font on

```

## How\_to\_display\_ticker\_on\_active\_idle

```

const CFont* logicalFont = AknLayoutUtils::FontFromId(EAknLogicalFontPrimaryFont); //logicalFont

//Setup a font for small text for the multi line text of dialpad
TFontSpec fontSpecSmall(CFont* logicalFont->FontSpecInTwips());
fontSpecSmall.Height = iWindow.Size().iHeight/4; // Get a font half the height of the window
fontSpecSmall.Style.SetStrokeWeight(EStrokeWeightNormal);
fontSpecSmall.Style.SetPosture(EPostureUpright);

// Obtain new font
CFont fontSmall;

scScreenDeviceFontInPixels((CFont*& )fontSmall, fontSpecSmall);
TInt smallFontDispWidth(iWindow.Size().iWidth / fontSmall->WidthZeroInPixels());

->SetFont((CFont*) fontSmall);

TInt widthSmall->TextWidthInPixels(iOriginalText); // Total width of the text that needs
TBool doRemoveText;

if(width > (iWindow.Size().iWidth))
{
// We have a text that is longer then the window width,
// we need to keep moving out from the start to show more content
doRemoveText = ETrue;
}
if(iStartPos > 0)
{
--=10; //Keep changing the draw position till the draw position is positive
}
else
{
if(doRemoveText)
{
<1> space(_L(" ")); TBuf
= iText.Find(space);pos
if(pos != KErrNotFound)
{
// Since the text length is more then the display width available
// We need to keep removing one word at a time from the start of the
// string to ensure that every word rolls in on the display area
= iText.Right(iText.Length-iText(pos+1));
= 0; iStartPos
}
else
{
// We have looped through bring back the original text now
Zero(); iText.
Append(iOriginalText); iText.
= iWindow.Size().iWidth;startWe have looped through lets start again
}
}
else
{
= iWindow.Size().iWidth; // We have looped through lets start again
}
}

->DrawText(iText, TPoint(iStartPos , (iWindow.Size().iHeight*3/4) )); // Point at which ticker
screenDeviceFont(fontSmall);
EWinRedraw();
->Deactivate();

```

## How\_to\_display\_ticker\_on\_active\_idle

```
FlushW$;  
}  
  
// End of File
```



This is how it looks without a ticker

And this is how it looks after the ticker has been implemented



For details on the implementation of the WsChangeObserver check

## Link

[Notification of application change](#)

For limitations on doing anything directly on the active idle

Link

## Link

[KIS001196 - Restrictions in developing customised Active Idle plug-ins](#)

[KIS001381 - Active idle theme cannot be changed by 3rd party applications](#)

For drawing directly on the screen check

## Link

[Drawing to the screen outside application framework](#)

**Download a working S60 3rd Edition, FP1 example from :-**

[File:TickerDemo.zip](#)

Even though the application hasnt been tested on other versions of S60/S40 I would assume it should work with a little bit of modifications on all/most of them.

The window on this is placed at a hard-coded location, you can customize that according to your needs, also consider using the device screen size/specifications as mentioned in the code above to change the positioning and to optimize it for different modes i.e landscape/portrait.

The window just has a normal white background and some hard-coded text, all of which can be changed as per needs, where for the former you can think about using a image while for the latter you can consider reading from a Database/web server etc.

**--- Added by Mayank 10/06/2009**