



ID	CS001411	Creation date	June 9 2009
Platform	S60 3rd Edition S60 5th Edition	Tested on devices	Nokia 5800 XpressMusic
Category	Symbian C++	Subcategory	Location Based Services

Keywords (APIs, classes, methods, functions): RPositionServer, RPositioner, TPositionInfo, TPositionModuleId

Overview

This snippet demonstrates how to get the GPS location of a device.

The `CLocation` class searches for the GPS location. If it does not get the location, it sleeps for 10 seconds and tries again.

Note: The `Location` capability is included in the self-signing capabilities of S60 3rd Edition, Service Pack 2 and newer platforms.

MMP file

The following libraries and capabilities are required:

```
CAPABILITY    Location
LIBRARY       lbs.lib
```

Header

```
#include <lbsSatellite.h>

class MPositionObserver
{
public:
    virtual void PositionUpdatedL(TPositionInfoBase& aPosInfo) = 0;
    virtual void ErrorL(TInt aError) = 0;
```

CS001411_-_Retrieving_GPS_location

```
};

#include <lbs.h>
#include <LbsSatellite.h>

// FORWARD DECLARATIONS
class MPositionObserver;

class CLocation : public CActive
{
public:

    static CLocation* NewL( TInt aInterval,
        MPositionObserver& aPositionListener ) ;
    virtual ~CLocation();

protected: // from CActive
    void DoCancel();
    void RunL();
    TInt RunError(TInt aError);

private:
    void ConstructL( );
    CLocation( TInt aInterval,
        MPositionObserver& aPositionListener );
    void DoInitialiseL();
    void PositionUpdatedL();
    void Wait();
    void CancelWait();
    void Start();
    static TInt PeriodicTick(TAny* aObject);
    void PositionLost();

public:
    void Pause();
    void Continue();
    TPositionInfoBase* CurrentPosition();

private: // Data

    // The id of the currently used PSY
    TPositionModuleId    iUsedPsy;

    // Position server
    RPositionServer      iPosServer;

    // Positioner
    RPositioner          iPositioner;

    // Basic location info
    TPositionInfo        iPositionInfo;

    TInt                 iInterval;
    TInt                 iUpdateTimeout;

    // Position listener
    MPositionObserver&   iPositionListener;

    // The id of the used psy
    TPositionUpdateOptions iUpdateops;
};
```

CS001411_-_Retrieving_GPS_location

```
// Position info base
TPositionInfoBase*      iPosInfoBase;

// State variable used to mark if we are
// getting last known position
TBool                   iGettingLastknownPosition;

CPeriodic*              iPeriodic;
};
```

Source

```
#include <Lbs.h>
#include <eikenv.h>

// CONSTANTS
const TInt KSecond = 1000000;
const TInt KMaxAge = KSecond;
const TInt KErrBuffer = 100;
// GPS sleeping time after error
const TInt KGPSSleepingMinor = KSecond*10;

//The name of the requestor
_LIT(KRequestor, "MyLocationApp");

CLocation::CLocation(TInt aInterval, MPositionObserver& aPositionListener)
    : CActive(CActive::EPriorityStandard),
      iInterval(aInterval),
      iPositionListener( aPositionListener ),
      iPosInfoBase( &iPositionInfo ),
      iGettingLastknownPosition( ETrue )
    {
    }

void CLocation::ConstructL()
    {
    // Set update interval to one second to receive one position data
    // per second
    iUpdateops.SetUpdateInterval(TTimeIntervalMicroSeconds(iInterval));

    // If position server could not get position
    // In two minutes it will terminate the position request
    iUpdateops.SetUpdateTimeout(TTimeIntervalMicroSeconds(iUpdateTimeout));

    // Positions which have time stamp below KMaxAge can be reused
    iUpdateops.SetMaxUpdateAge(TTimeIntervalMicroSeconds(KMaxAge));

    // Enables location framework to send partial position data
    iUpdateops.SetAcceptPartialUpdates(ETrue);

    // Add this position requestor to the active scheduler
    CActiveScheduler::Add( this );

    // Initialise the position request sequence
    DoInitialiseL();
    }
```

CS001411_-_Retrieving_GPS_location

```
CLocation* CLocation::NewL(TInt aInterval,
MPositionObserver& aPositionListener )
{
    //Create the object
    CLocation* self = new( ELeave ) CLocation(
        aInterval, aPositionListener);

    //Push to the cleanup stack
    CleanupStack::PushL( self );

    //Construct the object
    self->ConstructL();

    //Remove from cleanup stack
    CleanupStack::Pop( self );

    //Return pointer to the created object
    return self;
}

CLocation::~~CLocation()
{
    // Cancel active object (will call DoCancel() if the active object is active)
    Cancel();

    // The timer may be running, cancel it
    CancelWait();

    // Close the positioner
    iPositioner.Close();

    // Close the session to the position server
    iPosServer.Close();
}

void CLocation::DoCancel()
{
    //If we are getting the last known position
    if ( iGettingLastknownPosition )
    {
        //Cancel the last known position request
        iPositioner.CancelRequest(EPositionerGetLastKnownPosition);
    }
    else
    {
        iPositioner.CancelRequest(EPositionerNotifyPositionUpdate);
    }

    iGettingLastknownPosition = ETrue;
}

void CLocation::RunL()
{
    TBuf<KPositionMaxModuleName> buffer;

    //We are not going to query the last known position anymore.
    if ( iGettingLastknownPosition )
    {
        //Mark that we are not requesting NotifyPositionUpdate
        iGettingLastknownPosition = EFalse;
    }
}
```

CS001411_-_Retrieving_GPS_location

```
    }

switch ( iStatus.Int() )
{
    // The fix is valid
    case KErrNone:
    {
        // Pre process the position information
        PositionUpdatedL();
        break;
    }
    // The fix has only partially valid information.
    // It is guaranteed to only have a valid timestamp
    case KPositionPartialUpdate:
    {
        // Send partial data to registered listener
        iPositionListener.PositionUpdatedL(*iPosInfoBase);
        // and wait a while after askin position again
        Wait();
        break;
    }
    // The position data could not be delivered
    case KPositionQualityLoss:
    {
        PositionLost();
        break;
    }
    // Access is denied
    case KErrAccessDenied:
    {
        // Send error to position listener
        iPositionListener.ErrorL(KErrAccessDenied);
        break;
    }
    // Request timed out
    case KErrTimedOut:
    {
        PositionLost();
        break;
    }
    // The request was canceled
    case KErrCancel:
    {
        break;
    }
    // There is no last known position
    case KErrUnknown:
    {
        PositionLost();
        break;
    }
    // Unrecoverable errors.
    default:
    {
        // Send error to position listener
        iPositionListener.ErrorL(iStatus.Int());
        break;
    }
}
}
```

CS001411_-_Retrieving_GPS_location

```
void CLocation::DoInitialiseL()
{
    // Connect to the position server
    TInt error = iPosServer.Connect( );
    TBuf<KErrBuffer> buffer;

    // The connection failed
    if ( KErrNone != error )
    {
        iPositionListener.ErrorL( error );
        return;
    }

    // Open subsession to the position server
    error = iPositioner.Open(iPosServer);

    // The opening of a subsession failed
    if ( KErrNone != error )
    {
        iPositionListener.ErrorL( error );
        iPosServer.Close();
        return;
    }

    // Set position requestor
    error = iPositioner.SetRequestor( CRequestor::ERequestorService,
        CRequestor::EFormatApplication , KRequestor );

    // The requestor could not be set
    if ( KErrNone != error )
    {
        iPositionListener.ErrorL( error );
        iPositioner.Close();
        iPosServer.Close();
        return;
    }

    // Set update options
    error = iPositioner.SetUpdateOptions( iUpdateops );

    // The options could not be updated
    if ( KErrNone != error )
    {
        iPositionListener.ErrorL( error );
        iPositioner.Close();
        iPosServer.Close();
        return;
    }

    // Get last known position. The processing of the result
    // is done in RunL method
    Start();
}

TInt CLocation::RunError(TInt /*aError*/)
{
    return KErrNone;
}

void CLocation::Pause()
{
    if (IsActive())
```

CS001411_-_Retrieving_GPS_location

```
        {
            Cancel();
        }
    }

void CLocation::Continue()
{
    if (!IsActive())
    {
        Start();
    }
}

TPositionInfoBase* CLocation::CurrentPosition()
{
    return iPosInfoBase;
}

void CLocation::PositionUpdatedL()
{
    TPositionUpdateType update = iPosInfoBase->UpdateType();
    // Send GPS position
    if (update!=EPositionUpdateUnknown)
    {
        // Send position information to registered listener
        iPositionListener.PositionUpdatedL(*iPosInfoBase);
        Start();
    }
    else
    {
        Wait();
    }
}

void CLocation::Wait()
{
    if (!iPeriodic)
    {
        // Close GPS handles
        Cancel();

        // Sleep
        iPeriodic = CPeriodic::NewL(CActive::EPriorityIdle);
        iPeriodic->Start(KGPSSleepingMinor, KGPSSleepingMinor,
            TCallBack(PeriodicTick, this));
    }
}

TInt CLocation::PeriodicTick(TAny* aObject)
{
    CLocation* gpslistener = (CLocation*)aObject;
    if (gpslistener)
    {
        // Cancel timer running
        gpslistener->CancelWait();

        // Start listening GPS again after waiting a while
        gpslistener->Start();
    }

    // Does not continue again
    return EFalse;
}
```

CS001411_-_Retrieving_GPS_location

```
    }

void CLocation::Start()
{
    if (!IsActive())
    {
        // Get last known position. The processing of the result
        // is done in RunL method
        if (iGettingLastknownPosition)
        {
            iPositioner.GetLastKnownPosition(*iPosInfoBase, iStatus);
        }
        else
        {
            iPositioner.NotifyPositionUpdate(*iPosInfoBase, iStatus);
        }

        // Set this active object active
        SetActive();
    }
}

void CLocation::CancelWait()
{
    if (iPeriodic)
    {
        iPeriodic->Cancel();
        delete iPeriodic;
        iPeriodic = NULL;
    }
}

void CLocation::PositionLost()
{
    // Wait and then request again
    Wait();
}
```

How to use

```
// Class that use CSearchLocation have to implement
// MPositionObserver interface. Via MPositionObserver are position
// and error messages send.
iSearchLocation = CSearchLocation::NewL(1000000*10, *this);

// Get current position
TPositionInfoBase* pos = iSearchLocation->CurrentPosition();
```

Postconditions

The GPS location is received.

See also

- [CS001377 - Getting position data from TPositionInfoBase](#)
- [S60 Platform Location Example](#)