



These classes demonstrates how to obtain information related to the current location of the mobile device.

### Headers required:

```
#include <lbs.h>
#include <lbspositioninfo.h>
#include <lbssatellite.h>
```

### Link against:

```
LIBRARY lbs.lib
```

### Classes:

```
// helper class-observer
class MPositionReaderObserver
{
public:
    // if successful reading
    virtual void ReadingComplete( TPositionInfoBase& aPosInfo ) = 0;
    // if error occurred
    virtual void ReadingError( TInt aErrorNo ) = 0;
};

// location info reader
class CPositionReader : public CActive
{
public:
    static CPositionReader* NewL( MPositionReaderObserver* anObserver );
    ~CPositionReader();

    // requests for reading
    void ReadPosInfo();
    void ReadCourseInfo();
    void ReadSatelliteInfo();

    // data access
    const TPositionInfo& PosInfo();
    const TPositionCourseInfo& CourseInfo();
    const TPositionSatelliteInfo& SatelliteInfo();

public: // from CActive
    void RunL();
    void DoCancel();

private:
    CPositionReader();
    void ConstructL( MPositionReaderObserver* anObserver );
    void ReadInternal( TPositionInfoBase& anInfo );

private:
    MPositionReaderObserver* iObserver;

    // used to make the primary connection to the location server
    RPositionServer iServer;
    // used to create a sub-session with the server
    RPositioner iPositioner;
```

## Retrieving\_location\_information

```
TPositionInfoBase*   iCurInfo; // points to the current info
TPositionInfo        iPositionInfo; // location info
TPositionCourseInfo iPositionCourseInfo; // course info
TPositionSatelliteInfo iPositionSatelliteInfo; // satellites info
};
```

### The body:

```
// factory
CPositionReader* CPositionReader :: NewL( MPositionReaderObserver* anObserver )
{
    CPositionReader* self = new (ELeave) CPositionReader();
    CleanupStack :: PushL( self );
    self->ConstructL( anObserver );
    CleanupStack :: Pop( self );
    return self;
}

// c++ destructor
CPositionReader :: ~CPositionReader()
{
    Cancel();
    iPositioner.Close();
    iServer.Close();
}

void CPositionReader :: ReadPosInfo()
{
    ReadInternal( iPositionInfo );
}

const TPositionInfo& CPositionReader :: PosInfo()
{
    return iPositionInfo;
}

void CPositionReader :: ReadCourseInfo()
{
    ReadInternal( iPositionCourseInfo );
}

const TPositionCourseInfo& CPositionReader :: CourseInfo()
{
    return iPositionCourseInfo;
}

void CPositionReader :: ReadSatelliteInfo()
{
    ReadInternal( iPositionSatelliteInfo );
}

const TPositionSatelliteInfo& CPositionReader :: SatelliteInfo()
{
    return iPositionSatelliteInfo;
}

void CPositionReader :: RunL()
{
    if( iStatus == KErrNone )
    {
        // location retrieved. Operation finished.
        iObserver->ReadingComplete( *iCurInfo );
    }
    else
        // error occurred
```

## Retrieving\_location\_information

```
        iObserver->ReadingError( iStatus.Int() );
    }

void CPositionReader :: DoCancel()
{
    // cancel update notify
    iPositioner.CancelRequest( EPositionerNotifyPositionUpdate );
}

CPositionReader :: CPositionReader():
    ( CActiveScheduler :: EPriorityStandard )
{
    CActiveScheduler :: Add( this );
}

void CPositionReader :: ConstructL( MPositionReaderObserver* anObserver )
{
    iObserver = anObserver;

    // connect to the location server
    User :: LeaveIfError( iServer.Connect() );

    // open the default positioner
    User :: LeaveIfError( iPositioner.Open( iServer ) );

    _LIT( KAppName, "YourAppName" ); // define your application name
    User::LeaveIfError( iPositioner.SetRequestor(
        CRequestor :: ERequestorService,
        CRequestor :: EFormatApplication,
        KAppName ) );

    // set maximum allowed time for a location request
    const TInt KTimeout = 50000000; // 50 sec
    TTimeIntervalMicroSeconds timeOut( KTimeout );
    TPositionUpdateOptions updateOptions;
    updateOptions.SetUpdateTimeOut( timeOut );
    User::LeaveIfError( iPositioner.SetUpdateOptions( updateOptions ) );
}

void CPositionReader :: ReadInternal( TPositionInfoBase& anInfo )
{
    iCurInfo = &anInfo;

    // prepare active object
    Cancel();
    iStatus = KRequestPending;

    // issuer request
    iPositioner.NotifyPositionUpdate( anInfo, iStatus );
    SetActive();
}
}
```

### How to use:

- implement interface *MPositionReaderObserver* in your class
- include *CPositionReader\* iReader* as a class member
- activate request:

```
iReader = CPositionReader :: NewL( this ); // new reader
iReader->ReadSatelliteInfo(); // request satellites info
```

## Retrieving\_location\_information

- process results of reading:

```
void YourClass :: ReadingComplete( TPositionInfoBase& aPosInfo )
{
    TPositionSatelliteInfo& info = ( TPositionSatelliteInfo& )aPosInfo;
    TInt iSatellitesInView = info.NumSatellitesInView(); // count of satellites in view
    iSatellitesUsed = info.NumSatellitesUsed(); // count of used satellites
    ...
}
```

## Related Links:

- [Landmarks/web client example using Carbide.c++ and UI designer](#)
- [How to use Landmarks API](#)
- [How to select and show a landmark](#)
- [How to compact local landmark databases](#)
- [How to export landmarks from database to file](#)
- [How to import landmarks from file to database](#)
- [Execution of landmark operations](#)
- [How to obtain and save current location](#)
- [How to manage landmark categories](#)