Obtaining device speed

CPositionReader allows to obtain different information, related to the current location of the device. The following class `CDeviceSpeedReader` demonstrates how to get current device speed:

The headers of classes:

```
class MSimpleTimerNotify
{
   public:
      virtual void TimerExpired( TAny* aTimer,TInt aError ) = 0;
};

// class is used polling
// simple timer with the help of the active object
class CSimpleTimer: public CActive
{
   public:
      static CSimpleTimer* NewL(const TInt aPriority, MSimpleTimerNotify& aNotify);
      ~CSimpleTimer();
      void After(TTimeIntervalMicroSeconds32 aInterval);

   protected:
      void RunL();
      void DoCancel();

   private:
      CSimpleTimer(const TInt aPriority,MSimpleTimerNotify& aNotify);
      void ConstructL(void);

   private:
      RTimer iTimer;
      MSimpleTimerNotify& iNotify;
};

class CDeviceSpeedReader : public CBase,
                           public MPositionReaderObserver,
                           public MSimpleTimerNotify
{
   public:
      // factory
      // aTimeDelay - polling period, microseconds
      static CDeviceSpeedReader* NewL( TInt aTimeDelay );
      // C++ destructor
      ~CDeviceSpeedReader();

      // returns KErrNone if successful, otherwise an error code is returned
      // if KErrNone then aSpeed contains current speed, meters per second
      TInt CurrentSpeed( TReal32& aSpeed );

   public: // MPositionReaderObserver
      void ReadingComplete( TPositionInfoBase& aPosInfo );
      void ReadingError( TInt aErrorNo );

   public: // MSimpleTimerNotify
      void TimerExpired( TAny* aTimer,TInt aError );

   private:
      // c++ constructor
      CDeviceSpeedReader( TInt aTimeDelay );
      // second phase constructor
      void ConstructL();

      // wait iTimeDelay and start async. request
      void WaitAndReread();

   private:
      TInt iTimeDelay;
      TInt iLastRqResult;    // result of last reading
      TReal32 iCurrentSpeed; // last known value of speed
      CPositionReader* iReader;
      CSimpleTimer* iTimer;
};
```

The bodies of classes:

```
CSimpleTimer::CSimpleTimer(const TInt aPriority,MSimpleTimerNotify& aNotify)
    :CActive(aPriority),iNotify(aNotify)
{
}

CSimpleTimer::~CSimpleTimer()
{
    Cancel();
    iTimer.Close();
}

CSimpleTimer* CSimpleTimer::NewL(const TInt aPriority,MSimpleTimerNotify& aNotify)
{
    CSimpleTimer* me = new (ELeave) CSimpleTimer(aPriority,aNotify);
    CleanupStack::PushL(me);
    me->ConstructL();
    CleanupStack::Pop();
    return me;
}

void CSimpleTimer::ConstructL(void)
{
    CActiveScheduler::Add(this);
    iTimer.CreateLocal();
}

void CSimpleTimer::After(TTimeIntervalMicroSeconds32 aInterval)
{
    Cancel();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    iTimer.After(iStatus,aInterval);
    SetActive();
}

void CSimpleTimer::DoCancel()
{
    iTimer.Cancel();
}

void CSimpleTimer::RunL()
{
    iNotify.TimerExpired(this,iStatus.Int());
}



CDeviceSpeedReader* CDeviceSpeedReader :: NewL( TInt aTimeDelay )
{
   CDeviceSpeedReader* self = new (ELeave) CDeviceSpeedReader( aTimeDelay );
   CleanupStack :: PushL( self );
   self->ConstructL();
   CleanupStack :: Pop();
   return self;
}

CDeviceSpeedReader :: ~CDeviceSpeedReader()
{
   delete iTimer;
   iTimer = NULL;

   delete iReader;
   iReader = NULL;
}

TInt CDeviceSpeedReader :: CurrentSpeed( TReal32& aSpeed )
{
   if( iLastRqResult == KErrNone )
      aSpeed = iCurrentSpeed;
   return iLastRqResult;
}


void CDeviceSpeedReader :: ReadingComplete( TPositionInfoBase& aPosInfo )
{
   // successful completion of the last request
   iLastRqResult = KErrNone;
   TPositionCourseInfo& info = ( TPositionCourseInfo& )aPosInfo;
   TCourse course;
   info.GetCourse( course );
   iCurrentSpeed = course.Speed();

   // wait and read again
   WaitAndReread();
}

void CDeviceSpeedReader :: ReadingError( TInt aErrorNo )
{
   // error occured
   iLastRqResult = aErrorNo;

   // wait and read again
   WaitAndReread();
}

CDeviceSpeedReader :: CDeviceSpeedReader( TInt aTimeDelay ):
 iCurrentSpeed( -1 ), iLastRqResult( KErrNotReady ), iTimeDelay( aTimeDelay )
{
}

void CDeviceSpeedReader :: ConstructL()
{
   iReader = CPositionReader :: NewL( this );

   // new timer
   iTimer = CSimpleTimer :: NewL( CActive::EPriorityStandard, *this );

   // begin polling
   iReader->ReadCourseInfo();
}

void CDeviceSpeedReader :: WaitAndReread()
{
   // wait ...
   iTimer->After( iTimeDelay );
}

void CDeviceSpeedReader :: TimerExpired( TAny* aTimer*,TInt aError )
{
   // timeout completed -> new request
   iReader->ReadCourseInfo();
}
```

How to use:

```
TInt KPeriod = 1000000; // one second
CDeviceSpeedReader* speedReader = CDeviceSpeedReader :: NewL( KPeriod );
...
// need some time to obtain location information
TReal32 curSpeed;
if( speedReader->CurrentSpeed( curSpeed ) == KErrNone )
{
  // curSpeed contains current device speed
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.