Execution of landmark operations

Short description

CPosLmOperation - this is base class for landmark operations, it supports two variants of execution:

- method *ExecuteL()* allows to run the operation synchronously (batch mode):

Library required:

```
LIBRARY   eposlandmarks.lib //CPosLandmarkDatabase, CPosLmOperation
```

Capabilities needed:

```
Capability  LocalServices NetworkServices ReadDeviceData
Capability  ReadUserData WriteDeviceData WriteUserData
```

```
CPosLmOperation* operation = iDb->InitializeL();
CleanupStack :: PushL( operation );
operation->ExecuteL();
CleanupStack :: PopAndDestroy( operation );
```

This can be simplified by using function *ExecuteAndDeleteLD*. It runs the operation and deletes the object when it is done.

```
   ExecuteAndDeleteLD( iDb->InitializeL() );
```

- method *NextStep()* allows to run the operation asynchronously by using active objects (incrementally execution).

Base class for incrementally execution

You can use class **CLmOperationHandler** as a base class for the incrementally execution operations. Descendants must override only one method *ExecL()*.

CLmOperationHandler header:

```
#include <e32base.h>

// forward declarations
class CPosLandmarkDatabase;
class CPosLmOperation;

// operation execution obserever
class MLmOperationHandlerObserver
{
   public:
      virtual void OnOperationProgress( TInt aProgress ) = 0;
      virtual void OnOperationError( TInt aErrorCode ) = 0;
};

class CLmOperationHandler : public CActive
{
   public:
      CLmOperationHandler( const TDesC& anUri, MLmOperationHandlerObserver* anObserver
);
```


(C) Copyright Nokia 2014. All rights reserved.

```
      virtual ~CLmOperationHandler();

      void ExecuteL();   // start incremental execution
      void NotifyStop(); // notify the need to stop

   protected:
      virtual void ExecL() = 0;

   private:
      void DelObjects(); // free memory

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

   protected: // Data
      TBool iNeedStop;             // stop flag
      CPosLmOperation* iOperation; // current operation
      CPosLandmarkDatabase* iDb;   // landmark database for the operation
      TBuf<128> iUri;              // database URI
      TReal32 iProgress;           // current operation progress
      MLmOperationHandlerObserver* iObserver;

};
```

CLmOperationHandler implementation:

```
#include <epos_landmarks.h>
#include <epos_cposlandmarkdatabase.h>

#include "LmOpHandler.h"


CLmOperationHandler :: CLmOperationHandler( const TDesC& anUri,
                                            MLmOperationHandlerObserver* anObserver ):
    CActive( EPriorityNormal ),
           iObserver( anObserver ), iNeedStop( EFalse ), iUri( anUri )
{
   CActiveScheduler :: Add( this );
}


CLmOperationHandler :: ~CLmOperationHandler()
{
   Cancel();     // cancel request
   DelObjects(); // guaranteed removing
}

void CLmOperationHandler :: ExecuteL()
{
   iObserver->OnOperationProgress( 0 ); // operation start
   ExecL(); // virtual method call
}

void CLmOperationHandler :: NotifyStop()
{
```


(C) Copyright Nokia 2014. All rights reserved.

```
   iNeedStop = ETrue; // stop flag
}


void CLmOperationHandler :: RunL()
{
   switch( iStatus.Int() )
   {
      case KPosLmOperationNotComplete:
      if( iNeedStop )
      {
            // need to stop ("Cancel" was pressed)
            DelObjects();
         iNeedStop = EFalse;
      }
      else
      {
            // notify observer
         iObserver->OnOperationProgress( iProgress * 100 );

         // next iteration
            iOperation->NextStep( iStatus, iProgress );
            SetActive();
      }
         break;

      case KErrNone:
      {
     // The operation has completed.
      iObserver->OnOperationProgress( 100 );
      DelObjects();
      }  break;

      default:
      {
      // error occured
         iObserver->OnOperationError( iStatus.Int() );
      DelObjects();
      }  break;
   }
}

void CLmOperationHandler :: DoCancel()
{
   // Cancel is done by deleting the operation object.
   DelObjects();
}

void CLmOperationHandler :: DelObjects()
{
   delete iOperation;
   iOperation = NULL;

   delete iDb;
   iDb = NULL;
}
```


(C) Copyright Nokia 2014. All rights reserved.

The examples of descendants:

```
// this class allows to initialize database
class CLmOpDbInitHandler : public CLmOperationHandler
{
   public:
      CLmOpDbInitHandler( const TDesC& anUri, MLmOperationHandlerObserver* anObserver ):
         CLmOperationHandler( anUri, anObserver ) {}

   protected:
      void ExecL()
      {
         iDb = CPosLandmarkDatabase :: OpenL( iUri );
         if( iDb->IsInitializingNeeded() )
         {
            iOperation = iDb->InitializeL();
            iOperation->NextStep( iStatus, iProgress );
            SetActive();
         }
         else
         {
            // initialization not needed - forced finishing
            iStatus = KErrNone;
            RunL();
         }
      }
};

// this class allows to compact database
class CLmOpDbCompactHandler : public CLmOperationHandler
{
   public:
      CLmOpDbCompactHandler( const TDesC& anUri, MLmOperationHandlerObserver* anObserver
):
         CLmOperationHandler( anUri, anObserver ) {}

   protected:
      void ExecL()
      {
         iDb = CPosLandmarkDatabase :: OpenL( iUri );
         iOperation = iDb->CompactL();
         iOperation->NextStep( iStatus, iProgress );
         SetActive();
      }
};
```

Progress dialog

Class **CLmProgressDialog** demonstrates how to realize simple progress dialog.

Header:

```
#include <eikprogi.h>
#include <AknProgressDialog.h>
```


(C) Copyright Nokia 2014. All rights reserved.

```
class MProgressDialogObserver
{
   public:
      virtual void CancelOperation() = 0;
};


class CLmProgressDialog : public MProgressDialogCallback
{
   public:
      CLmProgressDialog( MProgressDialogObserver* anObserver ):
         iObserver( anObserver ), iProgressDialog( NULL ), iProgressInfo( NULL ) {}
      virtual ~CLmProgressDialog();

      void StartProgressNoteL();
      void UpdateProcessL( TInt aProgress, const TDesC& aProgressText );
      void ProcessFinishedL();

   protected:
      void DialogDismissedL( TInt aButtonId );

   private:
      CAknProgressDialog*  iProgressDialog;
      CEikProgressInfo*    iProgressInfo;
      MProgressDialogObserver* iObserver;
};
```

Implementation:

```
CLmProgressDialog :: ~CLmProgressDialog()
{
   ProcessFinishedL();
}

void CLmProgressDialog :: StartProgressNoteL()
{
   iProgressDialog = new (ELeave) CAknProgressDialog(
        reinterpret_cast<CEikDialog**>( &iProgressDialog ),
        ETrue );
   iProgressDialog->PrepareLC( R_PROGRESS_NOTE );
   iProgressInfo = iProgressDialog->GetProgressInfoL();
   iProgressDialog->SetCallback( this );
   iProgressDialog->RunLD();
   iProgressInfo->SetFinalValue( 100 );
}

void CLmProgressDialog :: UpdateProcessL( TInt aProgress, const TDesC& aProgressText )
{
   if( iProgressDialog )
      iProgressDialog->SetTextL( aProgressText );

   if( iProgressInfo )
      iProgressInfo->SetAndDraw( aProgress );
}

void CLmProgressDialog :: ProcessFinishedL()
```


(C) Copyright Nokia 2014. All rights reserved.

```
{
   if( iProgressDialog )
   {
      iProgressDialog->ProcessFinishedL();
      iProgressDialog = NULL;
   }
}

void CLmProgressDialog :: DialogDismissedL( TInt aButtonId )
{
    iProgressDialog = NULL;
    iProgressInfo = NULL;
    if( aButtonId == EAknSoftkeyCancel )
     iObserver->CancelOperation();
}
```

Using progress dialog with incrementally execution

Your AppUi class must realize necessary interfaces:

- MProgressDialogObserver,
- MLmOperationHandlerObserver

and contain members:

```
CLmProgressDialog*   iProgressDialog;
CLmOperationHandler* iOperationHandler;
```

Necessary methods can be implemented as follows:

```
void CYourAppUi :: ConstructL()
{
   ...
   iProgressDialog = new (ELeave) CLmProgressDialog( this );
   ...
}

CYourAppUi :: ~CYourAppUi()
{
   ...

   delete iProgressDialog;
   iProgressDialog = NULL;

   delete iOperationHandler;
   iOperationHandler = NULL;

   ...
}


void CYourAppUi :: CancelOperation()
{
   iProgressDialog->ProcessFinishedL();
   iOperationHandler->NotifyStop();
}
```


(C) Copyright Nokia 2014. All rights reserved.

```
_LIT( KProgressMess, "Completeness: " );
void CYourAppUi :: OnOperationProgress( TInt aProgress )
{
   if( aProgress == 0 )
      iProgressDialog->StartProgressNoteL();
   else
   {
      TBuf<32> messWait( KProgressMess );
      messWait.AppendNum( aProgress );
      messWait.Append( '%' );

      if( aProgress < 100 )
         iProgressDialog->UpdateProcessL( aProgress, messWait );
      else
      {
         iProgressDialog->UpdateProcessL( 100, messWait );
         iProgressDialog->ProcessFinishedL();
      }
   }
}

_LIT( KErrorMess, "Error ocurred, code is: " );
void CYourAppUi :: OnOperationError( TInt aErrorCode )
{
   TBuf<64> messError( KErrorMess );
   messError.AppendNum( aErrorCode );
   CAknErrorNote* note = new ( ELeave ) CAknErrorNote( ETrue );
   note->ExecuteLD( messError );
}
```

Now you can start incrementally execution:

```
_LIT( KDbUri, "file://C:eposlm.ldb" );
iOperationHandler = new (ELeave) CLmOpDbCompactHandler( KDbUri, this );
iOperationHandler->ExecuteL();
```

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
- How to obtain and save current location
- Retrieving location information
- How to manage landmark categories

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.