Landmarks/web client example using Carbide.c++ and UI designer

01 Mar 2009

Scope and introduction

This article will walk developers through the steps for developing a small S60 application that sends an HTTP query to a Web service to fetch landmarks information. The application UI is created using the Carbide UI designer. Example code is not intended to be of commercial quality and is provided for educational purposes only. A basic understanding of Carbide.c++ and S60 C++ is assumed.

- Building, setting build configurations, launching the debugger
- Basic S60 C++ development concepts

Functionality

- Querying the last-known position from the default positioning module
- Sending an HTTP query to the GPS Waypoints client API
  - Query requests information from the traffic cameras closest to the last-known position
- Parsing the query results
  - Creating landmark objects
  - Filling UI listbox data with query results
  - Adding query results to the landmarks database on the device

Omissions

- Checking for existing landmarks with the same information
- Error handling
- Code cleanup

Compatibility

- Tested on the Nokia N95 mobile device and S60 3rd Edition, Feature Pack 1 (FP1) emulator
  - Developers need to have a valid access point configured to connect to the Internet.
  - The device should be used to receive proper location data at least once before running the application.

Future plans

- Better integration with Landmarks API classes
- Possible integration with S60 Map&Navigation functionality at a later phase
  - Depending on API availability
- Carbide building and debugging 101 may be added to this article.

Source code and installation file

File:Examplecode.zip

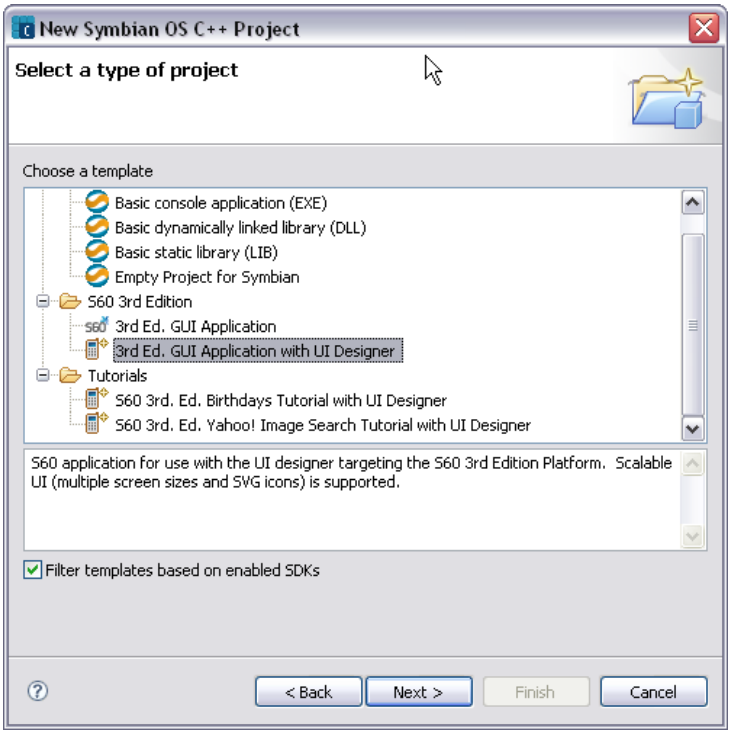
File:Myexample sisfile.zip

Developers will need a valid Developer Certificate from the Symbian Signed programme to sign the installation file if they want to run the application on a real device.

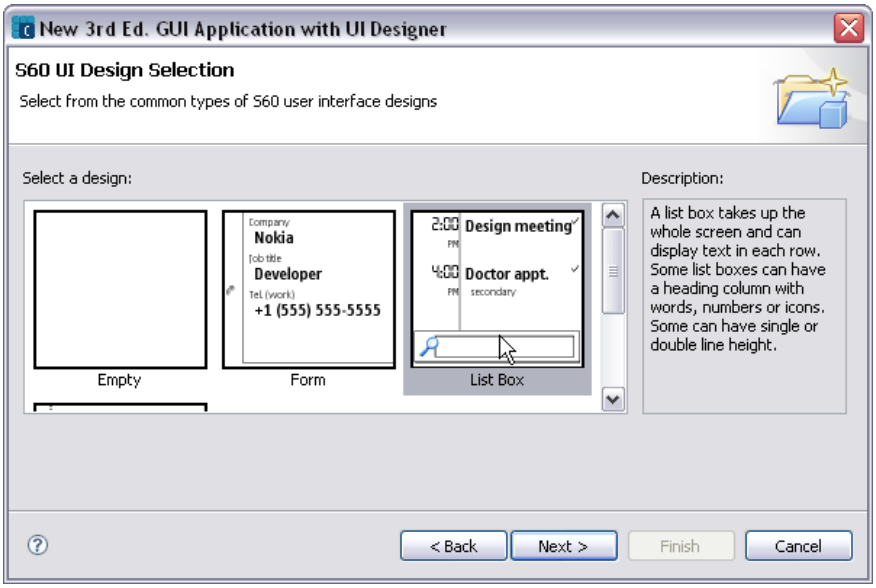
 (C) Copyright Nokia 2013. All rights reserved.

Creating the application project and UI

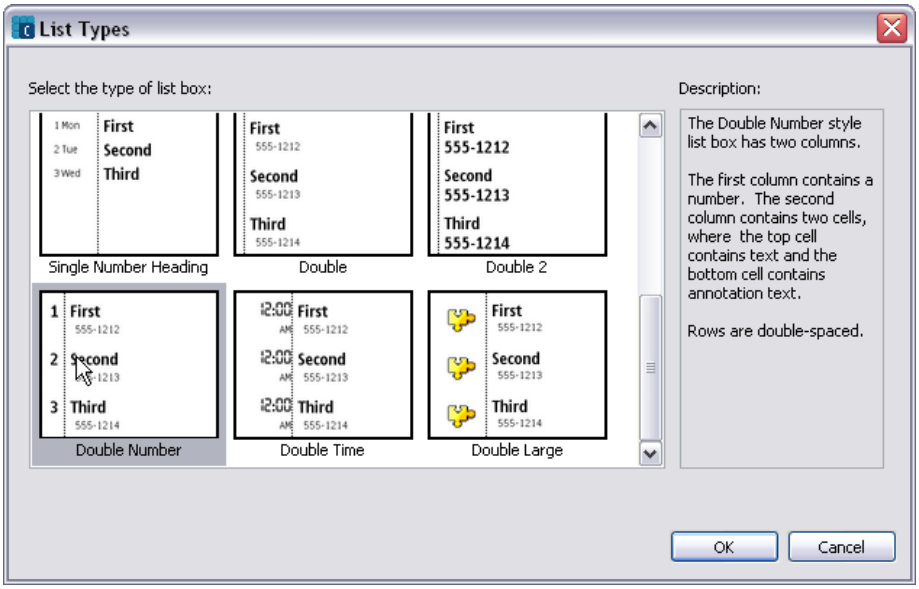
- Create a new project in Carbide.c++ by choosing *File -> New -> Project -> Symbian OS C++ Project*.
- Choose *S60 3rd Edition GUI Application with UI Designer* as the template.
- Enter a name for your project and select S60 3rd Edition, FP1 as the default SDK.



- Select *List Box Design* as the UI design and *Double Number* as the listbox type.



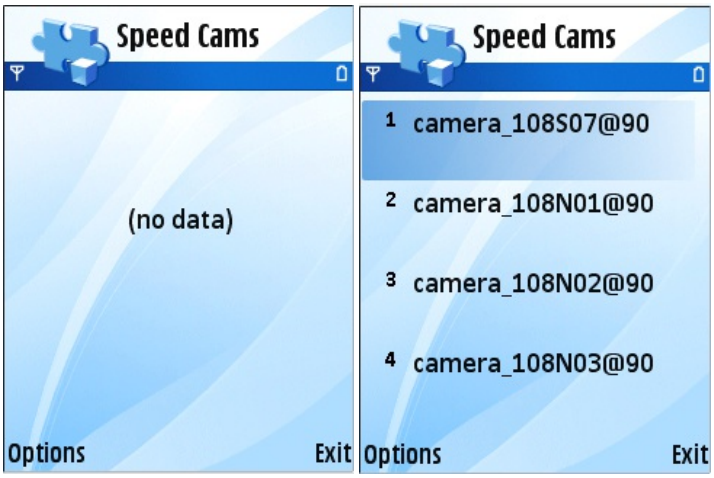
 (C) Copyright Nokia 2013. All rights reserved.

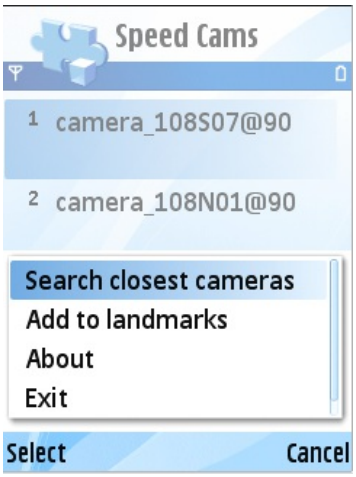


- You may accept the default values for:
  - Name for your base class;
  - View switching support;
  - Application UID;
  - Project directories.

Design the UI

The application UI consists of a single view, which is populated by a listbox control used for showing the landmarks data from a Web query.





 (C) Copyright Nokia 2013. All rights reserved.

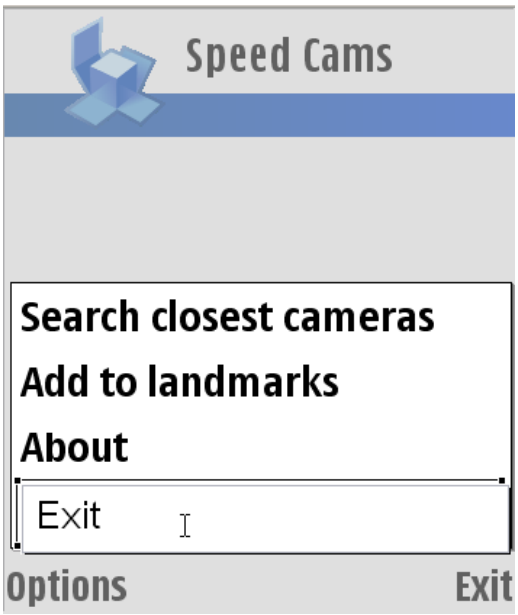
Status pane text manipulation

Your new (still empty) UI design opens up in Carbide after creating the project.

- Click on the status pane title to change the text.

Adding menu items

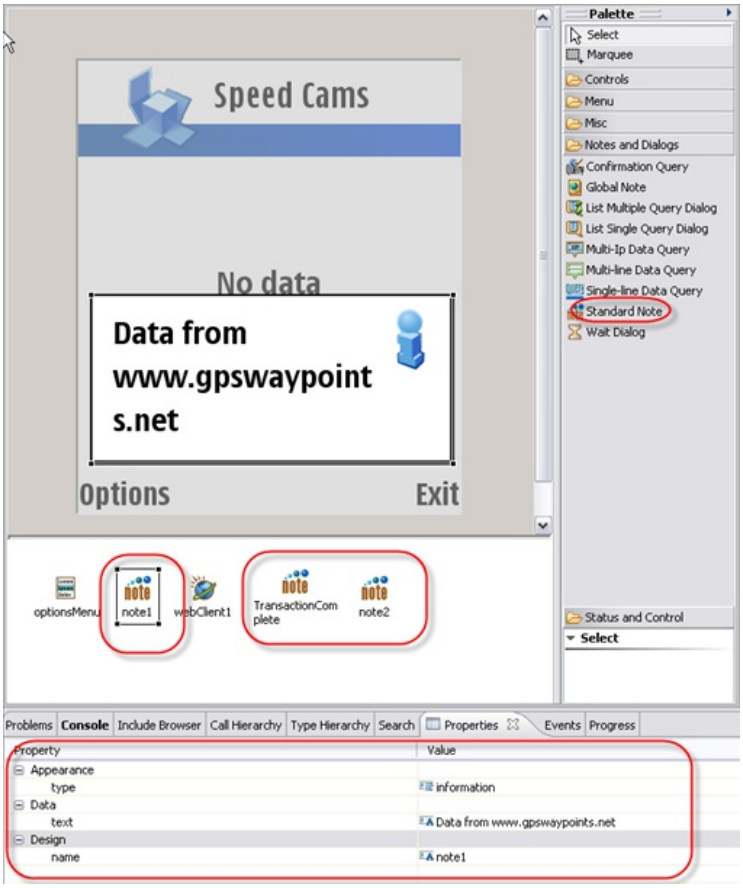
- Add the items shown in the image below to the menu, predefined by the Carbide UI Designer.



Adding notes

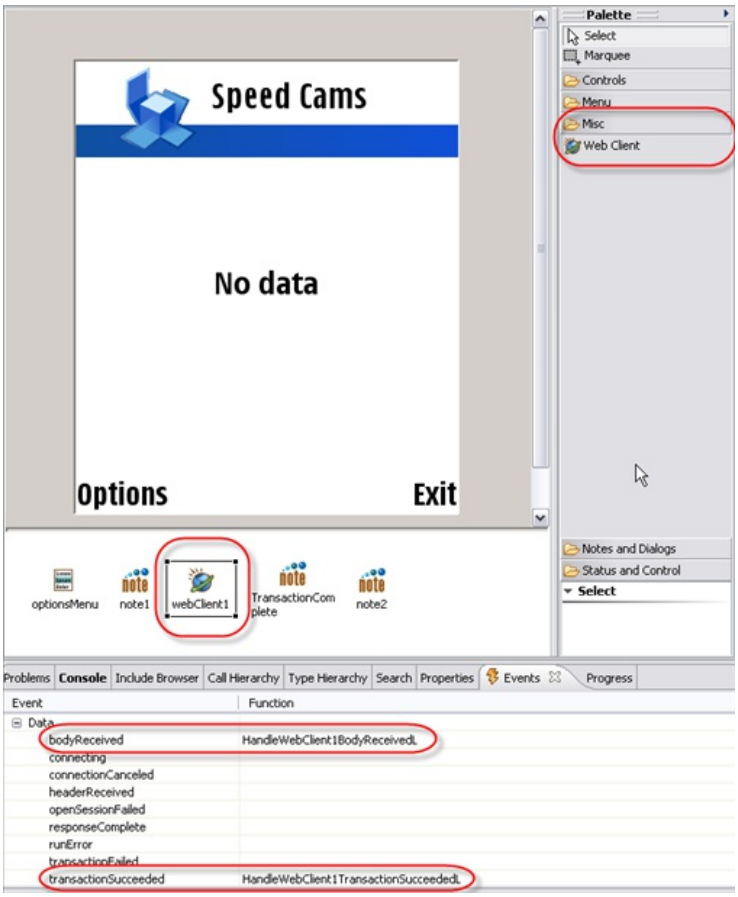
- Add three notes to the UI design by dragging and dropping *Standard Note* objects from the UI control palette:
  - Information note on where the data is downloaded;
  - Confirmation note of a successful HTTP transaction;
  - Error note for the case when user attempts to add landmarks without first performing an HTTP query.
- Use the note property tab to define the note type and text.

 (C) Copyright Nokia 2013. All rights reserved.



Adding a Web Client

- Drag and drop a *Web Client* component (offers painless HTTP connectivity) to the UI design.
  - Select the default values for the event handler functions for *bodyReceived* and *transactionSucceeded* events.



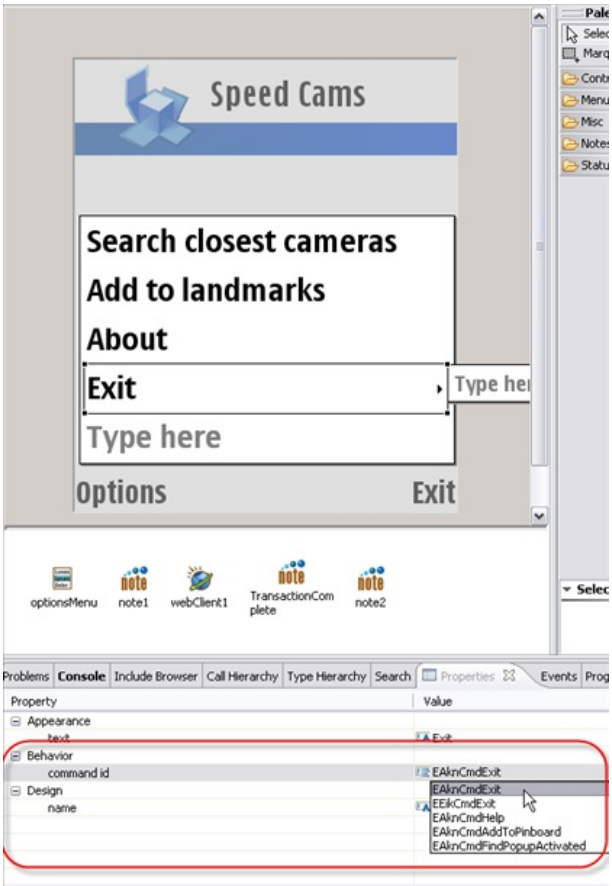
 (C) Copyright Nokia 2013. All rights reserved.

Handling simple menu commands

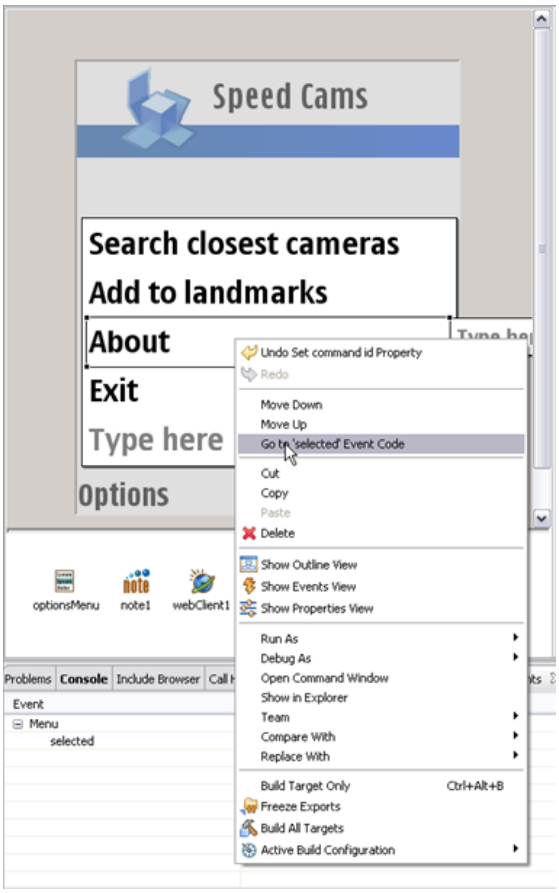
- Set the *Exit* command to use command ID

```
EAknCmdExit
```

.



- Set an event-handling function for the *About* menu item from the *Events* pane and navigate to the handler code.



 (C) Copyright Nokia 2013. All rights reserved.

- Among other code, Carbide has generated wrappers for launching the notes you defined earlier.

Your *About* menu item event handler should be similar to this:

```
/**
 * Handle the selected event.
 * @param aCommand the command id invoked
 * @return ETrue if the command was handled, EFalse if not
 */
TBool CMyExampleListBoxView::HandleAboutMenuItemSelectedL( TInt aCommand )
 {
 RunNote1L();
 return ETrue;
 }
```

- Set default named event handling functions for the *Search...* and *Add...* menu items as well. Don't implement the event handlers just yet.

Adding application logic

Interface

Most of the application logic is handled by a separate class, containing the following public interface:

```
 static CGPSPOIsManager* NewL();
static CGPSPOIsManager* NewLC();
virtual ~CGPSPOIsManager();

/**
* Appends data from the Web query to the data buffer
*
* @return void
* @param aData 8 bit descriptor result of HTTP query
*/
void AppendL( const TDesC8& aData );

/**
* Resets the landmark array and empties the data buffer
*
* @return void
*/
void Reset();

/**
* Handle to the array containing pointers to landmark objects
*
* @return void
*/
RArray< CPosLandmark* > Items();

/**
* Creates landmark items from the search results.
* Calls the parsing functions to parse the comma separated data
*
* @return void
*/
void CreateItemsL();

/**
* Adds the search result landmarks to the landmark database
*
* @return TInt
*/
TInt AddToDbL();
```

Implementation

Source and header files for this class need to be added to the Carbide project manually. The main functionality is explained below, while the link to the full source code of the class is linked to in this article.

Connecting to the default landmarks database

```
void CGPSPOIsManager::ConstructL()
 {
 // Open Landmarks database
 iLmDb = CPosLandmarkDatabase::OpenL();
 ...
```

Appending data incrementally from a running HTTP query

```
void CGPSPOIsManager::AppendL( const TDesC8& aData )
```

 (C) Copyright Nokia 2013. All rights reserved.

```
 {
 // append the received data to the data buffer
 TInt newLength = iDatabuf.Length() + aData.Length();
 if (iDatabuf.MaxLength() < newLength)
  {
  iDatabuf.ReAllocL( newLength );
  }
 iDatabuf.Append( aData );
 }
```

Parsing the HTTP-query result and creating landmark objects

The GPS Waypoints client API provides landmark objects formatted as:

```
#id, lat,lon,name,desc,location,(speed limit) ,(bearing),type,
```

Fields of an individual item are separated by commas and items are separated by a line break.

The sample application logic contains parsing functionality to handle the received data:

- If the following makes you uncomfortable, the *TLex* class offers convenient string parsing.

```
void CGPSPOIsManager::CreateItemsL()
 {
 // create landmarks from the received data
 // parse until all items have called parseiteml
 TInt ret = 0;
 while ( ret != KErrNotFound )
  {
  ret = ParseItemL( ret );
  }
 }

  TInt CGPSPOIsManager::ParseItemL( TInt aPosInBuf )
 {
 if ( aPosInBuf >= iDatabuf.Length() )
  return KErrNotFound;
 return ParseCSVItemL( aPosInBuf );
 }

TInt CGPSPOIsManager::ParseCSVItemL( TInt aPosInBuf )
 {
 // GPS waypoints provides data as comma separated values...
 // ...with line breaks between the result items

 // work with remaining data
 TPtrC8 remainder = iDatabuf.Right( iDatabuf.Length() - aPosInBuf );

 // locate a line change
 TInt ret = remainder.Locate( '\n' );

 // no more line changes (new items) left
 if ( ret == KErrNotFound )
  return ret;

 // single line without the line change
 TPtrC8 singleline = remainder.Left( ret );

 // resume parsing after the \n
 TInt newindex = aPosInBuf + ret + 1;

 // comment lines start with #
 if ( remainder.Left( 1 ).Compare( KMyComment ) == 0 )
  return newindex;

 ...
```

- *singleline* should now contain a string representing a single landmark item.
- Create a landmark object and fill it with values from the string.
- Add the filled landmark object to the array used for containing the items.

```
 // pass singleline to item constructor
 // item should parse and create itself
 // ...add item to itemarray
 //TBool eos = EFalse;
 TInt linepos = 0;
 iItemIndex = EMyLmId;
 iLat = 0;
 iLong = 0;
 CPosLandmark* landmark = CPosLandmark::NewL();
 while ( linepos >= 0 )
  {
  // if linepos == -1 => end of line
  // else continue at linepos
  // => increment linepos beyond the found ,
  linepos = ParseCSVFieldL( linepos, singleline );

  // increase itemindex after setting the value of a field in a landmark

  SetLandMarkItemL( landmark );
  iItemIndex++;
  }

 // add landmark to items
 //landmark->SetPartialL( 0 );
 iItems.Append( landmark );
 return newindex;
 }
```

- Parse attributes from the landmark item.

 (C) Copyright Nokia 2013. All rights reserved.

```
TInt CGPSPOIsManager::ParseCSVFieldL( TInt aLinePos, TPtrC8& aLine )
 {
 // Get an attribute of a search result item to iToken

 TPtrC8 remainder = aLine.Right( aLine.Length() - aLinePos );
 TInt pos = remainder.Locate( TChar(',') );

 // no more fields to read
 if ( pos < 0 )
  return pos;

 // Copy the single token
 if ( iToken.Length() < pos + 1 )
  iToken.ReAllocL( pos + 1 );

 iToken.Copy( remainder.Left( pos ) );

 // if the last ',' was the last character in the line
 TInt ret = aLinePos + pos + 1;
 if ( ret >= aLine.Length() )
  return -1;
 else
  return ret;
 }
```

- Set some attributes of the landmark object.
  - Please check SDK documentation on what attributes are available.

```
TInt CGPSPOIsManager::SetLandMarkItemL( CPosLandmark* aLandmark )
 {
 // only name, desc and coordinates handled in this case

 switch ( iItemIndex )
  {
  case EMyLmName:
   {
   aLandmark->SetLandmarkNameL( iToken );
   break;
   }
  case EMyLmDesc:
   {
   aLandmark->SetLandmarkDescriptionL( iToken );
   break;
   }
  case EMyLmLat:
   {
   TLex lex( iToken );
   lex.Val( iLat );
   break;
   }
  case EMyLmLon:
   {
   TLex lex( iToken );
   lex.Val( iLong );
   TCoordinate coord( iLat, iLong );
   TLocality lmlocality( coord, 10 );
   aLandmark->SetPositionL( lmlocality );
   break;
   }
  case EMyLmLocation:
   {
   break;
   }
  case EMyLmSpeed:
  case EMyLmBearing:
  case EMyLmType:
  case EMyLmId:
  default:
   break;
  }
 return 0;
 }
```

Adding landmarks to the database

```
TInt CGPSPOIsManager::AddToDbL()
 {
 TInt ret = 0;
 // add landmarks to the database
 for ( TInt i = 0; i < iItems.Count(); i++ )
  {
  iLmDb->AddLandmarkL( *iItems[i] );
  ret++;
  }
 return ret;
 }
```

UI changes for using the application logic

- An object of the application logic class has to be owned by some of the UI classes.
  - The sample application creates the object in the application UI class and defines methods for setting a pointer to the model to the listbox class.
    - Usually setting the model pointer would be done in the construction phase, but Carbide may generate some funky code to recreate overloaded constructors if the UI is modified.

Changes to the application UI class

 (C) Copyright Nokia 2013. All rights reserved.

```
// header file
#include "CGPSPOIsManager.h"
...
CGPSPOIsManager* iModel;

// source file
CMyExampleAppUi::~CMyExampleAppUi()
 {
 // [[[ begin generated region: do not modify [Generated Contents]
 // ]]] end generated region [Generated Contents]
 delete iModel;
 iModel = NULL;
 }

// [[[ begin generated function: do not modify
void CMyExampleAppUi::InitializeContainersL()
 {
 iMyExampleListBoxView = CMyExampleListBoxView::NewL();

 // this added after UI code generation
 iMyExampleListBoxView->SetAppModel( iModel );


...

void CMyExampleAppUi::ConstructL()
 {
 iModel = CGPSPOIsManager::NewL();

...
```

Changes to the listbox class

```
// header file
class CGPSPOIsManager;
...
CGPSPOIsManager* iModel;

...

// source file
void CMyExampleListBoxView::SetAppModel( CGPSPOIsManager* aModel )
 {
 iModel = aModel;
 }
```

Updating project settings

The location and landmarks-related functionality requires some link libraries as well as some capabilities to be added to project definitions. This is probably a good time to tweak the settings.

- Open the *Options* tab on your project's mmp file and set the following capabilities for your project:
  - *LocalServices Location NetworkServices ReadDeviceData ReadUserData WriteDeviceData WriteUserData*.
    - Please note that you will need a Developer Certificate from the Symbian Signed programme to sign an application for these capabilities.
- Add *lbs* and *eposlandmarks* libraries to your project on the *Libraries* tab of your mmp file.

Getting position information and issuing the HTTP query

With the application logic mostly in place, you can configure your HTTP-query behaviour.

Carbide UI Designer generates code that eases the pain of issuing and handling the Web queries. The code is generated to the *CWebClientEngine* class, which by default uses the view issuing the queries as an observer to the progress of the queries.

The HTTP query is initiated from the *Search...* menu item. Carbide has generated a function placeholder to the listbox class for handling that menu command.

```
TBool CMyExampleListBoxView::HandleSearch_closest_camerasMenuItemSelectedL( TInt aCommand )
 {
 // TODO: implement selected event handler
 return ETrue;
 }
```

HTTP-query format

The query format for getting traffic camera items from GPS Waypoints client API is:

http://www.gps-waypoints.net/gps/gwnapi/get_closest_waypoints.php?
api_key=[YOURAPIKEY]&lat=[LATCOORDINATE]&lon=[LONCOORDINATE]&type=100006&distance=[RADIUS]&limit=[LIMIT]

You can get an API key by registering to the Web site. The sample application limits the search results to five items and uses 10,000 km as the search radius.

 (C) Copyright Nokia 2013. All rights reserved.

We used a test key for the API calls: NNVALVOGTYHXAQC.

Getting last-known-position information

The sample application contains the position-fetching implementation inside the listbox class. You should probably implement this functionality inside your application model and use active objects to handle the async position-fetching call.

Connecting to the positioning server and opening the default positioning module

```
// header file
#include <lbs.h>

...

RPositionServer iPosServer;
RPositioner  iPositioner;


//source file

#include <lbspositioninfo.h>
#include "CGPSPOIsManager.h"

...

// destructor

 iPositioner.Close();
 iPosServer.Close();

...

// ConstructL

TInt err = iPosServer.Connect();

 TPositionModuleId moduleid;
 err = iPosServer.GetDefaultModuleId( moduleid );
 err = iPositioner.Open( iPosServer, moduleid );

 if ( err == KErrNone )
  {
  _LIT( KMyData, "MyExample");
  iPositioner.SetRequestor(
              CRequestorBase::ERequestorService,
              CRequestorBase::EFormatApplication,
              KMyData );
  }
```

Getting the last-known position

The sample code obtains the last-known position in the event handler of the *Search...* menu item.

```
/**
 * Handle the selected event.
 * @param aCommand the command id invoked
 * @return ETrue if the command was handled, EFalse if not
 */
TBool CGPSPOIsSearchViewView::HandleSurroundingPOIsSelectedL( TInt aCommand )
 {
 // The ResetListL method has been added
 // to empty the listbox
 iMyExampleListBox->ResetListL();
 iMyExampleListBox->ListBox()->SetCurrentItemIndex(0);

 _LIT8( KSearchURL, "http://www.gps-waypoints.net/gps/gwnapi/get_closest_waypoints.php?api_key=NNVALVOGTYHXAQC&lat=%f&lon=%f&type=10

 // get the last known position
 TRequestStatus status;
 TPositionInfo posinfo;

 // use an AO in a real life app
 iPositioner.GetLastKnownPosition( posinfo, status );
 User::WaitForRequest( status );

 TPosition position;
 posinfo.GetPosition( position );

  ...
```

Issuing the HTTP query

```
...


 // create the search url
 RBuf8 rbuf;
```

 (C) Copyright Nokia 2013. All rights reserved.

```
 rbuf.CreateMax( KSearchURL().Length()*2 );

 // format the search string to include the position info
 rbuf.Format( KSearchURL, position.Latitude(), position.Longitude() );

 iModel->Reset();

 // generated by Carbide => encapsulates the HTTP-query
 IssueHTTPGetL( &rbuf );

 rbuf.Close();

 return ETrue;
 }
```

Handling HTTP-query results

The Web Client encapsulation will notify its observer (our listbox view class) on the progress of the query.

In the UI design phase, we indicated that we are interested in handling the *bodyReceived* and *transactionSucceeded* events.

Handling increments of body data

When a part of the HTTP-query result body has been received, you want to add it to the data buffer maintained by your model.

```
/**
 * ClientBodyReceivedL()
 * Called when a part of the HTTP body is received.
 * @param aBodyData:  Part of the body data received. (e.g. part of
 *                    the received HTML page)
 */
void CMyExampleListBoxView::ClientBodyReceivedL(
  CWebClientEngine& anEngine,
  const TDesC8& aBodyData )
 {
 // [[[ begin generated region: do not modify [Generated Code]
 HandleWebClient1BodyReceivedL( anEngine, aBodyData );
 // ]]] end generated region [Generated Code]

 }
/**
 * Handle the bodyReceived event.
 */
void CMyExampleListBoxView::HandleWebClient1BodyReceivedL(
  CWebClientEngine& /*anEngine*/,
  const TDesC8& aBodyData)
 {
 iModel->AppendL( aBodyData );
 }
```

Handling a successful transaction

When the HTTP transaction has completed, ask your model to parse the data and construct the landmark objects.

```
/**
 * ClientTransactionSucceededL()
 * Called to notify that a transaction completed successfully
 * See TTransactionEvent::ESucceeded
 */
void CMyExampleListBoxView::ClientTransactionSucceededL(
  CWebClientEngine& anEngine )
 {
 // [[[ begin generated region: do not modify [Generated Code]
 HandleWebClient1TransactionSucceededL( anEngine );
 // ]]] end generated region [Generated Code]

 }
/**
 * Handle the transactionSucceeded event.
 */
void CMyExampleListBoxView::HandleWebClient1TransactionSucceededL(
  CWebClientEngine& /*anEngine*/ )
 {
 // show a note to indicate transaction completion
 // the note was defined in the UI design phase and
  // Carbide has generated wrapper code for it
 RunTransactionCompleteL();

 // ask the model to parse the data and create landmarks
 iModel->CreateItemsL();

 iMyExampleListBox->ResetListL();

 RArray< CPosLandmark* > items = iModel->Items();

 // populate listbox here
 for ( TInt i = 0; i < items.Count(); i++ )
  {
  TBuf<256> buf;
  TPtrC lmname;
  TPtrC lmdesc;
  CPosLandmark* lmark = items[ i ];
```

 (C) Copyright Nokia 2013. All rights reserved.

```
    lmark->GetLandmarkName( lmname );
    lmark->GetLandmarkDescription( lmdesc );

    // These functions generated by Carbide
    iMyExampleListBox->CreateListBoxItemL( buf, i+1,
             lmname, lmdesc );
    iMyExampleListBox->AddListBoxItemL( iMyExampleListBox->ListBox(), buf );
    }
   iMyExampleListBox->ListBox()->DrawDeferred();
   }
```

Adding landmarks to the database

A menu item for adding the query results to the landmarks database was defined in the UI design phase.

Modify the event handler method to add the landmarks.

```
  /**
   * Handle the selected event.
   * @param aCommand the command id invoked
   * @return ETrue if the command was handled, EFalse if not
   */
  TBool CMyExampleListBoxView::HandleAdd_to_landmarksMenuItemSelectedL( TInt aCommand )
   {
   if ( iModel->Items().Count() == 0 )
    RunNote2L();
   else
    {
    iMyExampleListBox->ResetListL();
    iMyExampleListBox->ListBox()->SetCurrentItemIndex(0);
    iMyExampleListBox->ListBox()->DrawDeferred();

    TInt ret = iModel->AddToDbL();
    iModel->Reset();
    }
   return ETrue;
   }
```

Wrap-up

You should now be able to build, run, or debug the application. As a reminder, you will need a valid Developer Certificate to sign the application if you want to run it on a real device.

Related Links:

- 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
- Retrieving location information

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.