# JAM49 Lab: BBM Integration in the NDK

## Introduction

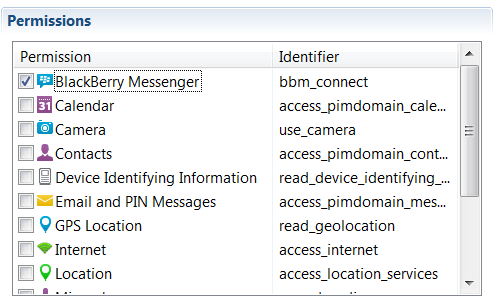
In this lab we will take the Cascades sample called bucketlist and add features from the BlackBerry Messenger Social Platform. We will enable the sample to register with the BlackBerry Messenger Social Platform, create profile box items and update the user’s personal message in their BBM profile when they complete or chicken out on items in their bucket list.

## Configure Project

Open the BlackBerry Native SDK and create a new workspace. Import the partially completed application by going to File -> Import, select General -> Existing Projects into Workspace and choose the location of the bucketlist\_bbm\_incomplete project.

Open **bar-descriptor.xml** and click on the General tab. Click the “Set from Debug Token” to configure the project for your debug token.

Within **bar-descriptor.xml** click on the Application tab and check off BlackBerry Messenger in the Permissions box.



Now we need to add the BBM SP library. Right click on your project and choose **Configure** -> **Add Library.** Click Next and filter for BBM. Add the libbbplatformbbm library. Follow the steps in the wizard and add the line below to the bucketlist\_bbm\_incomplete.pro file

LIBS += -lbbplatformbbm

## Let’s start coding! registration

The first step to creating a BlackBerry Messenger Social Platform (BBM SP) application is registering with the BBM Social Platform. Each application must define its own UUID so that it can uniquely identify itself. This UUID is used to register with the BBM SP servers during testing and development. Applications in BlackBerry App World are assigned their own UUID automatically.

Open **RegistrationHandler.cpp** and locate the TODO item in the constructor. Define your own UUID, altering the value in the comment section.

Locate the **RegistrationHandler::appRegister** method and connect the BBM SP registration signals to our application's registrationStatus slot.

QObject::**connect**(m\_context,

SIGNAL(registrationStateUpdated(bb::platform::bbm::RegistrationState::Type)),

**this**,

SLOT(registrationStatus(bb::platform::bbm::RegistrationState::Type)));

Locate **RegistrationHandler::registrationStatus**, which is called whenever the registration status is changed. Find the case where registration is allowed and update the registration screen after access is allowed. Place a success message in m\_statusText and make m\_continueButton visible. The registration screen is defined in registration.qml.

m\_statusText->**setText**("BBM SP registration succeeded. Press Continue.");

m\_continueButton->**setVisible**(**true**);

Now that we have registration ready to go, let’s modify main.cpp to display the registration page as the first screen in the application. Within main, create a new instance of the RegistrationHandler page and set it as the application scene.

RegistrationHandler\* page;

page = **new** RegistrationHandler();

app.**setScene**(page);

Then move the line that previously displayed the application’s main page to the RegistrationHandler::continueToMainAppScreen method.

BucketListApp mainApp;

Run the application and you should see a successful registration message. Then press continue to go to the main application screen.

## Create a profile box item

BBM SP connected applications appear in the application list of a users BlackBerry Messenger (BBM) profile. Applications can create items within their profile that are shared with a user’s BBM contacts. A maximum of 3 items are displayed at one time. Newer entries push older entries from view.

Each item contains a String and can contain an image. Images must be registered before they can use. Open **ProfileBox.cpp** and observe how images are registered.

Locate the **ProfileBox::createItem** method and add the line of code to create the profile box item based on the supplied String and icon.

**const** **bool** result = m\_profileBox->**requestAddItem**(text, iconId,

QString("cookie"));

Now we create a slot in the ProfileBox class that can be called from QML to create a profile box item. Open **ProfileBox.h** and define the createItem method as a slot.

**void** **createItem**(**const** QString& text, **const** QString& iconPath);

The final step to expose this method to QML is to register the ProfileBox class. Open **bucketlistapp.cpp** and register ProfileBox as com.bucketlist.profilebox version 1.0.

qmlRegisterType<ProfileBox>("com.bucketlist.profilebox", 1, 0, "ProfileBox");

At this point we can connect the QML with our C++ work. Open **main.qml** and import com.bucketlist.profilebox.

import com.bucketlist.profilebox 1.0

Locate the attachedObjects of ListView and uncomment the ProfileBox object that has been named profileBox. We will use this object to call the createItem method.

Call the createItem method in the “todo”, “finished” and “chickened” ActionItem onTriggered methods.

profileBox.**createItem**("Added some items to my bucket list",

"images/todo.png");

profileBox.**createItem**("Finished some items on my bucket list",

"images/finished.png");

profileBox.**createItem**("Chickened out on some items on my bucket list",

"images/chickened.png");

The triggers we just updated only apply when multiple items are selected. We need to do something similar to the onTriggered items in the ListItem. First, let’s expose profileBox to the ListItem using an alias. In the ListView, define the alias as follows.

**property** alias profileBox: profileBox

Open **ToDoItem.qml** located in the assets/items directory. Once again we call the createItem method in the “todo”, “finished” and “chickened” ActionItem onTriggered methods, but this time referencing the alias we just created.

bucketItem.ListItem.view.profileBox.**createItem**(

"Added an item to my bucket list: " +

ListItemData.title, "images/todo.png");

bucketItem.ListItem.view.profileBox.**createItem**(

"Finished an item on my bucket list: " +

ListItemData.title, "images/finished.png");

bucketItem.ListItem.view.profileBox.**createItem**(

"Chickened out on an item on my bucket list: " +

ListItemData.title, "images/chickened.png");

Run the application and use the menu options to change the status of some bucketlist items. Open BlackBerry Messenger, click on your name at the top and select the Apps tab. You should see the application listed. Expand the application view to see the profile box items that were created.

## Update the user’s personal message

The second type of BBM SP integration we will add to this sample is the ability to update the user’s personal status message in BBM. This message is also visible to all of the user’s contacts.

Open **ProfileMessage.cpp** and initialize m\_userProfile in the constructor.

m\_userProfile = **new** bb::platform::bbm::**UserProfile**(

Global::*instance*()->getContext(), **this**);

Next locate the ProfileMessage::setPersonalMessage method and call requestUpdatePersonalMessage on m\_userProfile, passing in the new personal message.

m\_userProfile->**requestUpdatePersonalMessage**(personalMessageString);

If you open up **ProfileMessage.h**, you’ll see that the ProfileMessage::setPersonalMessage method is already configured as a public slot, meaning it is ready to be called from QML.

Once again we have a class to expose in QML. Open **buckelistapp.cpp** and register ProfileMessage so that it can be used within our QML documents.

qmlRegisterType<ProfileMessage>("com.bucketlist.profilemessage", 1, 0, "ProfileMessage");

Open **main.qml** and import com.bucketlist.profilemessage, which we just registered.

import com.bucketlist.profilemessage 1.0

Locate the attachedObjects of ListView and uncomment the ProfileMessage object that has been named profileMessage. Now we can add calls to ProfileMessage::setPersonalMessage to our QML files, similar to what was done with ProfileBox::createItem. Locate the “todo”, “finished” and “chickened” ActionItem onTriggered methods where we used createItem before and add calls to setPersonalMessage.

profileMessage.**setPersonalMessage**("Added some items to my bucket list");

profileMessage.**setPersonalMessage**("Finished some items on my bucket list");

profileMessage.**setPersonalMessage**(

"Chickened out on some items on my bucket list");

That covers the multi select menus. These triggers we also just updated only apply when multiple items are selected. We need to do something similar to the onTriggered items in the ListItem. First, let’s expose profileMessage to the ListItem using an alias. In the ListView, define the alias as follows.

**property** alias profileMessage: profileMessage

Now do the same thing for the single selection menus. Open **TodoItem.qml** and add calls to setPersonalMessage from the “todo”, “finished” and “chickened” ActionItem onTriggered methods defined there. We are going to make use of the profileMessage alias defined in ListView this time.

bucketItem.ListItem.view.profileMessage.**setPersonalMessage**(

"Added an item to my bucket list: " + ListItemData.title);

bucketItem.ListItem.view.profileMessage.**setPersonalMessage**(

"Finished an item on my bucket list: " + ListItemData.title);

bucketItem.ListItem.view.profileMessage.**setPersonalMessage**(

"Chickened out on an item on my bucket list: " + ListItemData.title);

Run the application again. When you select Todo, Chickened or Finished from the menu you should see a prompt requesting permission to change your BBM personal message. Click OK and navigate to BBM to see the updated personal message shown at the top of the screen.