# JAM349 Lab: BBM Integration in a Native Application

## 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.

## Import project

To get started you need to open the BlackBerry Native SDK. You’ll be doing this within a VMWare Session. Within the VMWare Session click on the Start menu -> All Programs -> Research In Motion -> BlackBerry Native SDK 10.0.9 to start the SDK.

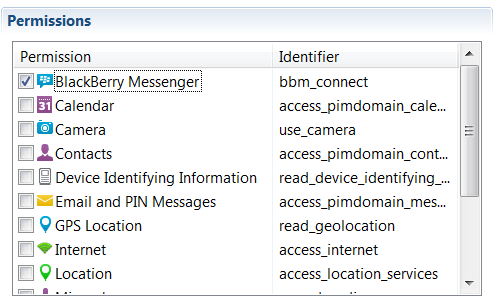
Once it’s open import the project by clicking in the File menu and choosing Import… Then expand General and choose **Existing Projects Into Workspace** and follow the wizard, selecting the location where you extracted bbm\_bucketlist\_incomplete.zip, which is supplied on a USB drive. **Do NOT choose** Import Existing Code as BlackBerry C/C++ Makefile Project.

## 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 and data libraries. Right click on your project and choose **Configure** -> **Add Library.** Click Next and filter for BBM. Add the libbbplatformbbm under the lbbdata library entry. Follow the steps in the wizard and add the line below to the bucketlist\_bbm\_incomplete.pro file

LIBS += -lbbdata

**LIBS += -lbbplatformbbm**

## Configure your BlackBerry 10 dev alpha

To side load an unsigned application on a BlackBerry 10 device, it needs to be put into Developer Mode. To turn on Developer Mode run the Settings application on your BlackBerry 10 Dev Alpha, choose Security and Privacy, Development Mode and toggle Development Mode to On. Swipe up from the bottom bezel to close the Settings application (or any other app).

If you haven’t run BlackBerry Messenger on your BlackBerry 10 Dev Alpha yet, run it now and configure it. Create a new BlackBerry ID when running for the first time. Reusing the same BlackBerry ID as your BlackBerry Smartphone will transfer your BBM contacts from your smartphone, removing them from your smartphone.

Now connect your BlackBerry 10 Dev Alpha to the PC using your USB cable. Within the BlackBerry Native SDK click on File -> New -> Other and choose BlackBerry -> BlackBerry Target. The default IP address for a BlackBerry connected over USB is 169.254.0.1. Enter that in the IP field and your password in the password field and click finish. The SDK will now connect to your device. Note that you’ll need to wait for Windows to finish loading the driver for the device and pass the USB connection to the VM before the connection will succeed. To re-connect, right click on the target you created and choose BlackBerry Tools -> Connect.

## 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**, located in the **src\bucketbbm** folder and locate the TODO item in the constructor. Define your own UUID, altering the value in the comment section. You can generate one here: http://www.guidgenerator.com/

mUuid = QString::**fromUtf8**("YOUR UUID GOES HERE");

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

QObject::**connect**(mContext,

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

**this**,

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

Locate the **RegistrationHandler::registrationFinished** method and observe the different registration statuses that can be returned. Note the signal emitted at the end of the registration process. This is received in **onAppRegisteredChanged** located in **main.qml**.

Now run the application by right clicking on the application and choosing Run As -> BlackBerry C/C++ Application. If you haven’t built the project yet, this will fail with an error that no binaries are present. Right click on the project and select Build. The project will be automatically rebuilt when performing a Run/Debug As if anything is changed in your project.

Run the application, press and hold an item in the list until the menu appears, then make a selection from the menu to move a few items from one type of list (todo, completed or chickened). The application should run without issue, but you won’t see any BBM activity just yet.

## 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** located in **src\bucketbbm** 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, uncommenting the error check.

**const** **bool** result = mProfileBox->**requestAddItem**(text, iconId, QString("cookie"));

Now let’s move to the QML. Open **BucketList.qml** located in the **assets** folder and define the signal that will be fired to trigger the creation of a ProfileBox item and update the user’s personal message in BBM.

**signal** **newBBMStatus**(**string** message, **string** icon)

In the same file locate the **updateBBMStatus** method and call the newBBMStatus signal, passing in the message and image.

**newBBMStatus**(message, image);

Now open **main.qml** and observe the **onNewBBMStatus** slot capturing the newBBMStatus signal you just configured. Note that it’s calling **bbmmanager.updateStatus**. Open bucketbbmmanager.cpp and locate the **updateStatus** method. Here you can see it’s creating an instance of our **profilebox.cpp** class and calling **createItem**, which we just completed in the first step of this section.

Now run the application, press and hold an item in the list until the menu appears, then make a selection from the menu to move a few items from one type of list (todo, completed or chickened). This should trigger the population of a profile box item in BBM. To view it, open BBM, click on your profile heading at the top of the application (your image and white text with blue writing) to bring up the profile page and then click on the Apps tab where you should see bucketlist\_incomplete listed. Expand bucketlist\_incomplete and observe the profile box items.

## 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 locate the **setPersonalMessage** method. Within in initialize the **mUserProfile** variable within the if statement.

**if** (mUserProfile == 0) {

//**TODO**: Grab the user's Profile.

mUserProfile = **new** bb::platform::bbm::**UserProfile**(

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

}

Next call requestUpdatePersonalMessage on m\_userProfile, passing in the new personal message.

mUserProfile->**requestUpdatePersonalMessage**(personalMessageString);

Open bucketbbmmanager.cpp and locate the **updateStatus** method again. Here you can see it’s creating an instance of our **profilemessage.cpp** class and calling **setPersonalMessage**, triggered by the same flow used in the previous section.

Now run the application again and move a task from one list to another. You should be prompted to allow the application to set your BBM personal message. Agree and open BBM and you should see the new personal message at the top of the screen.

Congratulations! You’ve just BBM connected an application!

## BONUS – invocation framework and BBM

In the steps above we’ve made use of the BlackBerry Messenger Social Platform APIs to make access BBM directly within the application. It is also possible to integrate BBM in your application using the Invocation Framework. This can allow you to bring cards (screens) from the BBM Client directly into your application for things such a chat, BBM group chat, share files, invite to BBM and update the user’s BBM Avatar. The invocation framework doesn’t require the use of the BBM registration process we used above either. The code samples you are entering below are 100% stand alone and can be copied directly into your application to add this functionality. Let’s take a look at two ways we can integrate BBM chat into the application using the Invocation Framework.

Open **BucketListPage.qml**, scroll down to the bottom and locate the actions section. Within it create an **InvokeActionItem** to start a BBM Chat with a BBM contact. Make sure you update the PIN in the URI to match a PIN in your BBM contact list. If the PIN doesn’t exist, the item won’t be listed in the menu. Also remember to place a comma after the } from the **ActionItem** listed before this one.

InvokeActionItem {

title: "Start BBM Chat"

//ActionBar.placement: ActionBarPlacement.OnBar

query {

invokeActionId: "bb.action.BBMCHAT"

uri: "pin:2100000A" //Update this PIN.

}

}

Replace **pin:2100000A** with a PIN of someone in your BBM Contact list. If you don’t have any contacts yet use PIN 2a867334, which is the PIN of a BlackBerry Dev Alpha at the front of the room. Come up to the front to add it to your BBM Contact list. If you attempt to use a PIN that isn’t in your contact list the InvokeActionItem will not appear in the menu of your application.

You can also create an **InvokeActionItem** that is pre-populated with some text to share, instead of a contact. This will allow the user to select the contact(s) they wish to start the chat with from your BBM Contact list. Create the **InvokeActionItem** below right after the one you just created. The onTriggered method below is optional. It’s included to show how you could dynamically include text generated dynamically. In a real application, the data value in the onTriggered method would be updated from a field or variable.

InvokeActionItem {

title: "Share Text Over BBM"

query {

mimeType: "text/plain"

invokeTargetId: "sys.bbm.sharehandler"

invokeActionId: "bb.action.SHARE"

data: "This is some text to share."

}

onTriggered: {

data = "This is some new text."

}

}

Now run the application again and try out these two **InvokeActionItems** and see the different workflows they use. The share text method can also be used to share text with other applications. Comment out or remove the **invokeTargetId** and run the application again. You’ll now have the choice to share text with any application registered with the Invocation Framework as a text handler.