Tutorial: Building BlackBerry application to receive BES PUSH notifications on SMP Cloud Edition
Includes code snippets for onboarding and push listener

Author

Buddha Puneeth N.

Contents

What is PUSH Notification? 3
What is BES? 3
Overview on BES PUSH 3
Server side requirements 4
Supported push formats 5
PAP PUSH format -Open standard 5
RIM PUSH format - BB proprietary 5
BES PUSH using SMP Cloud Edition 5
Overview of SMP Cloud Edition 5
Configuring SMP for BES PUSH 5
Client-side listener applications 8
Sample code 8
How to initiate PUSH request from REST Client: 14
References 15

What is PUSH Notification?

Push, or server push, describes a style of Internet-based communication where the request for a given transaction is initiated by the publisher or central server. It is contrasted with pull, where the request for the transmission of information is initiated by the receiver or client.

What is BES?

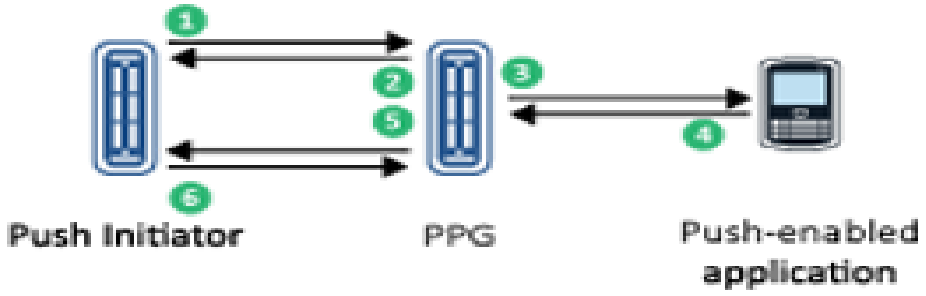
BES (Blackberry Enterprise Server) is a middleware software package that is a part of BlackBerry wireless platform supplied by BlackBerry Ltd. It allows BlackBerry devices to access corporate messaging and collaboration software such as Microsoft Exchange, Lotus. The middleware synchronizes messaging content between enterprise servers and Research in Motion Blackberry smartphones.

BES provides lot of services such as:

- BlackBerry Alert
- BlackBerry Collaboration Service (provides IM services)
- BlackBerry Controller
- BlackBerry Dispatcher
- BlackBerry MDS Connection Service
- BlackBerry Router
- BlackBerry Synchronization Service and few more.

Here 'BlackBerry MDS Connection Service' servers in making PUSH requests from intranet applications.

Overview on BES PUSH



BES request response flow diagram

When the PPG is the BlackBerry Enterprise Server or the BlackBerry Device Service, push messages are sent following this process:

1. The Push Initiator sends a push message to the BlackBerry MDS Connection Service of the BlackBerry Enterprise Server or the BlackBerry Device Service in the form of an HTTPS POST.

   The push message is a MIME multipart message, which contains the following items:

- A WAP PAP 2.0 XML control entity, which describes the delivery parameters and specifies one or more BlackBerry devices to which the content will be delivered
- The content to deliver to the specified BlackBerry devices

2. The BlackBerry MDS Connection Service returns a push response to acknowledge receipt of the push message, and indicates whether the message is accepted for processing or rejected. If the push message is rejected, the BlackBerry MDS Connection Service returns an error code to the content provider that provides the reason for the rejection.
3. The BlackBerry MDS Connection Service sends the push content to the specified BlackBerry devices.
4. Each BlackBerry device notifies the BlackBerry MDS Connection Service when the push message is received. A push message is considered successful if the message is delivered before the specified expiry time and it meets the criteria specified by the <quality-of-service> element in the push message.
5. If the Push Initiator requests notification, the BlackBerry MDS Connection Service sends a result notification message to the push server.
6. The Push Initiator responds to the BlackBerry MDS Connection Service, acknowledging the receipt of the result notification.

   To push data to the BlackBerry devices, two applications are required: a server-side application (push initiator) which submits the push request and a client side application which listens to the push messages.

Server side requirements

1. the application must build a push request and send it as an HTTP POST request to the BlackBerry MDS Connection Service
2. the application must build the push request in either the RIM push format or the PAP push format
3. the application must identify in the push request the port number on the BlackBerry device on which a corresponding client listener application is listening

You can push data to individual users based on either their email addresses or their **device PINs**, or to groups of users created and maintained on the BlackBerry® Enterprise Server.

Supported push formats

PAP PUSH format -Open standard

This push format sends an HTTP POST request to the BlackBerry MDS Connection Service. The request
contains a MIME multipart message that includes two components: an XML-based PAP control entity
that defines the delivery parameters, and the data to be pushed.
The PAP push format is an open standard developed by the Open Mobile Alliance. The BlackBerry
MDS Connection Service supports the WAP PAP 2.0 standard.

RIM PUSH format - BB proprietary

This push format sends an HTTP POST request to the BlackBerry MDS Connection Service. However, in this case, the pushed content is sent as a byte stream.

BES PUSH using SMP Cloud Edition

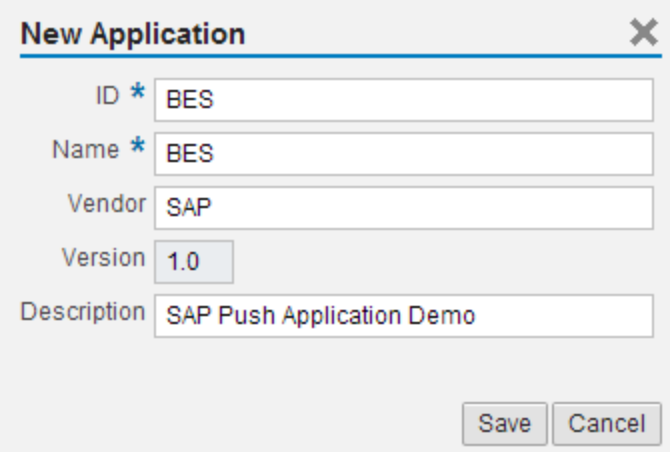
Overview of SMP Cloud Edition

The SAP Mobile Platform, enterprise edition, cloud version enables SAP partners and customers to deploy quality Business-to-Consumer (B2C), Business-to-Employee (B2E), and custom online mobile applications in a cloud environment, which is an essential part of SAP on Demand strategy. The platform is also known as Mobile as a Service (MaaS).

Application developers can use the mobile platform to build lightweight, on-demand applications (for example, productivity applications) that complement and extend existing SAP enterprise solutions. The platform enables interoperability through openness, while at the same time ensuring security and integrity required by mobile applications operating in a distributed network environment. It provides all the required tools, infrastructure and services to get your on-demand applications up and running easily and quickly.

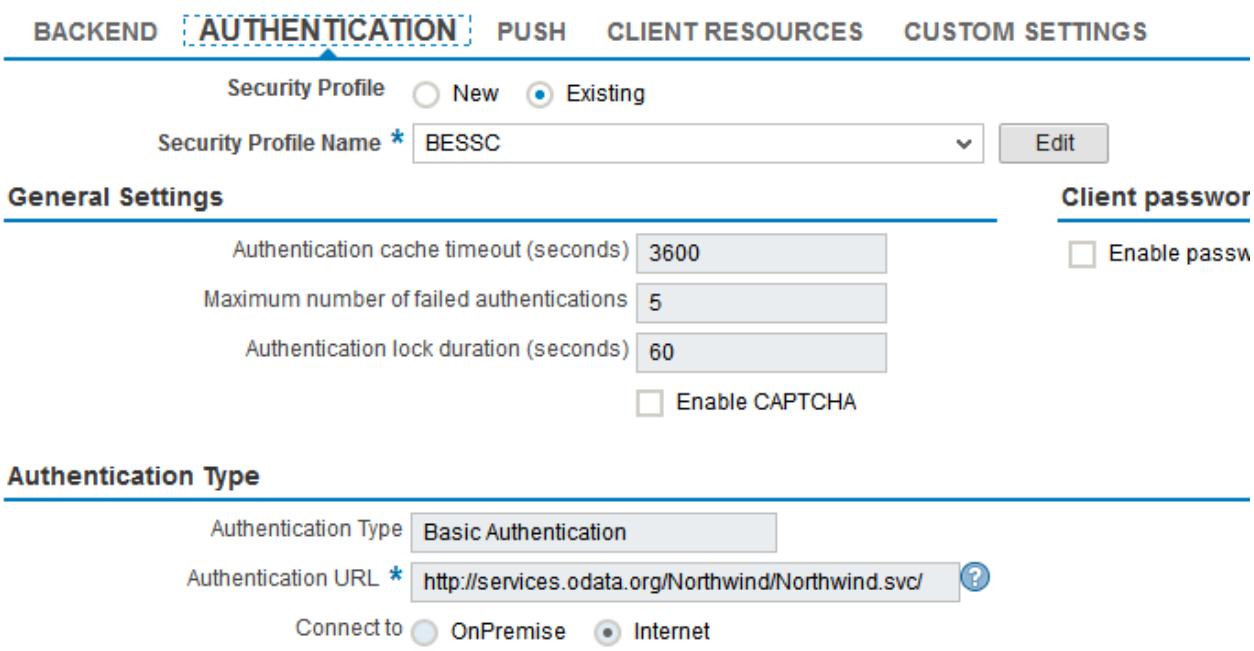
Configuring SMP for BES PUSH

1. Open Admin portal and Under ‘Application’ tab, click on ‘New’ and provide necessary details

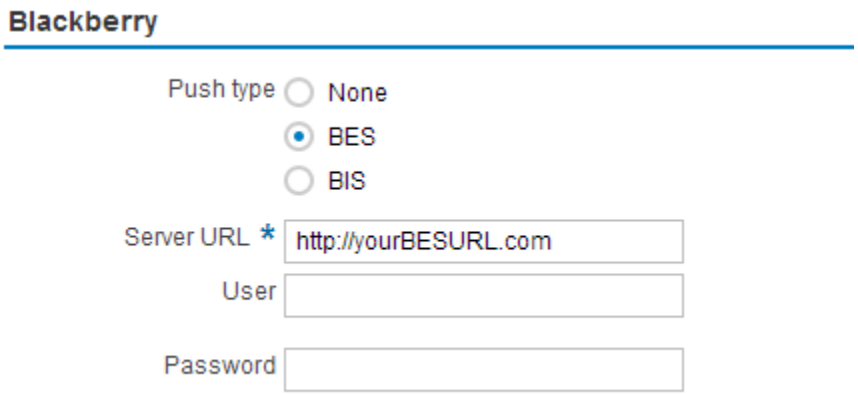


2. Provide Backend URL (Ebay OData Service) and Authentication details

(Making URL as anonymous, as EBay OData service doesn't require any authorization)



3. Configure with BES push details under PUSH tab



We should get the Server URL from the infrastructure team who set up BES server and we need to provide User and Password if requires.

4. Click on SAVE
5. User should have PUSH role in order to perform PUSH operations, Admin has rights to provide this role for any SDN user

Client-side listener applications

Client-side application can be developed in many types:

1. **BB application using JAVA:** You can create a custom BlackBerry Java Application that contains a listening thread that listens on a specified port for incoming data.
2. **BB widget** : You can create a BB widget using web technologies.
3. **BB browser as listener:** The BlackBerry Browser on the BlackBerry device contains a built-in listening thread, which listens on port 7874. You can push web content or other browser-supported data to the BlackBerry Browser, which stores it in a cache dedicated to pushed content. When the BlackBerry device user views the URL associated with the pushed content, the browser retrieves the content from the cache and displays it, rather than making an HTTP request for the content. You can add an icon to the BlackBerry device Home screen that is associated with your push request. To the user, the icon looks like an application, but it simply opens an instance of the BlackBerry Browser to the URL associated with the pushed content. If you use the BlackBerry Browser as your client-side listener application, you must only create the server-side application that makes the push request.

Sample code

Let see an example of BB same application for listening to PUSH messages.

1. First we need to onboard to SMP using any of the onboarding type in order to receive PUSH message
2. In onboarding body we need to pass the necessary details for BES push like BES port, device type and device pin under corresponding properties

```
<?xml version='1.0' encoding='utf-8'?>
<entry xmlns="http://www.w3.org/2005/Atom"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
<title type="text"/>
<updated>2012-06-15T02:23:29Z</updated>
<category
term="applications.Connection"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<content type="application/xml">
<m:properties>
 <d:DeviceType>BlackBerry</d:DeviceType>
 <d:BlackberryDevicePin>XXXXXX</d:BlackberryDevicePin>
 <d:BlackberryBESListenerPort
m:type=\"Edm.Int32\">XXXX</d:BlackberryBESListenerPort>
</m:properties>
</content>
</entry>
```

3. Sample code for onboarding looks like:

```
package bb.sap;


import java.io.IOException;
import java.io.OutputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;

import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.container.MainScreen;

public class home extends MainScreen {
public home()
{
      ButtonField register = new ButtonField("register",ButtonField.CONSUME_CLICK);
      register.setChangeListener(new FieldChangeListener(){

            private int rc;
            private String resp;
       private ListeningThread _listeningThread;
            public void fieldChanged(Field arg0, int arg1) {
                  // TODO Auto-generated method stub
                  onboard();
            }

            private void onboard() {
                  // TODO Auto-generated method stub
```

```
                    String body = "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n"
                      +"<entry xmlns=\"http://www.w3.org/2005/Atom\"\n"

+"xmlns:d=\"http://schemas.microsoft.com/ado/2007/08/dataservices\"\n"

+"xmlns:m=\"http://schemas.microsoft.com/ado/2007/08/dataservices/metadata\">\n"
                      +"<title type=\"text\"/>\n"
                      +"<updated>2012-06-15T02:23:29Z</updated>\n"
                      +"<author><name/></author>\n"
                      +"<category term=\"applications.Connection\"
scheme=\"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme\"/>\n"
                      +"<content type=\"application/xml\">\n"
                      +"<m:properties>\n"
                      +"<d:DeviceType>BlackBerry</d:DeviceType><d:DeviceModel
m:null=\"true\" />\n"
                      +"<d:BlackberryDevicePin>XXXX</d:BlackberryDevicePin>\n"
                      +"<d:BlackberryBESListenerPort
m:type=\"Edm.Int32\">XXXX</d:BlackberryBESListenerPort>"
                      +"</m:properties>\n"
                      +"</content>\n"
                      +"</entry>";
                    HttpConnection httpCon; // httprequest declaration
                    OutputStream os;
                    rc = 0;
                    resp = "";
                    try {
                            // making request
                           httpCon =
(HttpConnection)Connector.open("http://<SMPURL>/odata/applications/latest/<APPNAME/Co
nnections;deviceside=true;interface=wifi", Connector.READ_WRITE);
                          // here BES is the name of SMP application
                           byte[] postDataBytes = body.getBytes(); //converting
sting into bytes
                           httpCon.setRequestMethod(HttpConnection.POST);
//Onboarding is always a post operation
                           httpCon.setRequestProperty("Authorization", "Basic
c3VwdXNlcjI6czNwdXNlcg=="); //Base64 encoded string of UN:PWD
                          httpCon.setRequestProperty("Content-
type","application/atom+xml");
                          os = httpCon.openOutputStream();
                          os.write(postDataBytes);  //making POST request with body as
argument

                           rc = httpCon.getResponseCode(); // reading the response
code
                           resp = httpCon.getResponseMessage();//reading response
message

                           if (rc == 201)   //on successful onboarding
                           {
                                  Dialog.alert("Onboarding successful");
                                  _listeningThread = new ListeningThread();
                                  _listeningThread.start();// calling push listener
```

```
                            }
                            else  // when onboarding is failed
                            Dialog.alert("Onboarding failed with response code: " +
rc + " and response msg: "+ resp);

                    } catch (IOException e) {
                          // TODO Auto-generated catch block
                          e.printStackTrace();
                    }

             }

      });
      add(register);
}
}
```

Once onboarding is done, we are calling PUSH listener thread using the following code:

```
_listeningThread.start();
```

4. The listener class looks like follows:

```
package mypackage;

import java.io.IOException;
import java.io.InputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;

import net.rim.device.api.io.http.HttpServerConnection;
import net.rim.device.api.io.http.MDSPushInputStream;
import net.rim.device.api.system.Application;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.util.DataBuffer;

public class Listenerone extends Thread{
       private boolean _stop = false;
     private StreamConnectionNotifier _notify;
     private static final String URL = "http://:POSR";    // PORT 100.
     private static final int CHUNK_SIZE = 256;
     /**
      * Stops the thread from listening
      */
     private synchronized void stop()
     {
         _stop = true;
```

```
        if(_notify != null)
        {
            try
            {
                _notify.close();
            }
            catch (Exception e)
            {
            }
        }
    }
    public void run()
    {
      StreamConnection stream = null;
        InputStream input = null;
        MDSPushInputStream pushInputStream=null;
        try
        {
            _notify = (StreamConnectionNotifier)Connector.open(URL);
            while (!_stop)
            {
                // NOTE: the following will block until data is received
                stream = _notify.acceptAndOpen();
                try
                {
                    input = stream.openInputStream();
                    pushInputStream = new
MDSPushInputStream((HttpServerConnection)stream, input);

                    // Extract the data from the input stream
                    DataBuffer db = new DataBuffer();
                    byte[] data = new byte[CHUNK_SIZE];
                    int chunk = 0;

                    while ( -1 != (chunk = input.read(data)) )
                    {
                        db.write(data, 0, chunk);
                    }

                    updateMessage(data);

                    // If the push server has application level reliabilty
                    // enabled, this method call will acknowledge receipt
                    // of the push.
                    pushInputStream.accept();

                    data = db.getArray();
                }
                catch (IOException ioe)
                {
                    // A problem occurred with the input stream , however,
the original
                    // StreamConnectionNotifier is still valid.
                    errorDialog(ioe.toString());
```

```
                }
                finally
                {
                    if ( input != null )
                    {
                        try
                        {
                            input.close();
                        }
                        catch (IOException ioe)
                        {
                        }
                    }

                    if ( stream != null )
                    {
                        try
                        {
                            stream.close();
                        }
                        catch (IOException ioe)
                        {
                        }
                    }
                }
            }
            }
            catch (IOException ioe)
            {
                errorDialog(ioe.toString());
            }
        }

        private void updateMessage(final byte[] data)
        {
            Application.getApplication().invokeLater(new Runnable()
            {
                public void run()
                {
                    // Query the user to load the received message
                    String[] choices = {"Ok" , "Cancel" };

                    if ( 0 != Dialog.ask("New message received. Do you want to
render it?" , choices, 0) )
                    {
                        return;
                    }


                    try
                    {
                       Dialog.alert(new String(data));
                    }
                    catch (Exception e)
```

```
                {
                    errorDialog("RichTextField#setText(String) threw " +
e.toString());
                }
            }
        });
    }


    /**
     * Presents a dialog to the user with a given message
     * @param message The text to display
     */
    public static void errorDialog(final String message)
    {
        UiApplication.getUiApplication().invokeAndWait(new Runnable()
        {
            public void run()
            {
                Dialog.alert(message);
            }
        });
    }
}
```

The Push listener is implemented as a thread and we are calling this thread after onboarding. This thread will keep on listening for any PUSH messages, in case if it receives any push message it will parse the push data and pop-up to the user in a separate thread which we will call using 'invokelater' API.

How to initiate PUSH request from REST Client:

In ideal scenario, backend use to trigger the PUSH request. Here in this tutorial we are using Rest Client for making a PUSH request.

Considering that user has PUSH role for this application.

Sample PUSH url: http(s)://<Server Name>:<Run time Port>/Push/<APPCID>
Headers:
Authorization: Base-64 value of SDN credentials in format Username: Password
x-sup-data or x-sup-rim-data or x-sap-poke-data: Data or details which we need to push.

References

1. PUSH overview:
   http://developer.blackberry.com/bbos/java/documentation/push_service_overview.html
2. BES developers guide:
   http://docs.blackberry.com/en/developers/deliverables/12029/Push_applications_for_the_BlackBerry_Enterprise_Server-Development_Guide--915141-1116110032-001-US.pdf
3. SMP Cloud help:
   https://help.hana.ondemand.com/mobile/frameset.htm?SMP_welcome.html

www.sap.com