­­Emergency Contact List

**Software Architecture Document**

**Version 1.0**

Table of Contents

[1 Introduction 3](#_Toc355110953)

[1.1 Overview 3](#_Toc355110954)

[1.2 Scope 3](#_Toc355110955)

[1.3 Acronyms / Definitions 3](#_Toc355110956)

[1.4 References 4](#_Toc355110957)

[2 Software Architecture 5](#_Toc355110958)

[2.1 Architecture Overview 5](#_Toc355110959)

[3 Components Overview 6](#_Toc355110960)

[3.1 ECL Model 6](#_Toc355110961)

[3.2 Push Handling Module 6](#_Toc355110962)

[3.3 Utility Module 6](#_Toc355110963)

[3.4 Application UI 6](#_Toc355110964)

[3.4.1 Main List Screen 6](#_Toc355110965)

[3.4.2 Contact Details Screen 7](#_Toc355110966)

[3.4.3 Phone List screen 7](#_Toc355110967)

[3.4.4 Header List Item 7](#_Toc355110968)

[4 Implementation Details 8](#_Toc355110969)

[4.1 JSON format 8](#_Toc355110970)

[4.2 Data Model 10](#_Toc355110971)

[4.3 Utilities Module 10](#_Toc355110972)

[4.4 Phonebook Integration Module 11](#_Toc355110973)

[4.5 Main Contact List Screen 11](#_Toc355110974)

[4.6 View Details Screen 12](#_Toc355110975)

[4.7 Phone list Screen 12](#_Toc355110976)

[5 Testing 13](#_Toc355110977)

[5.1 Test Environment 13](#_Toc355110978)

[5.2 Test Plan and Results 14](#_Toc355110979)

[5.3 Testing without push infrastructure (Demo) 14](#_Toc355110980)

[6 Issues and Resolution 15](#_Toc355110981)

[6.1 Call Not working on the Work Perimeter: 15](#_Toc355110982)

[6.2 SMS sending is not working: 15](#_Toc355110983)

[6.3 List item with Arrows (>, V) on the right side: 16](#_Toc355110984)

# Introduction

### Overview

The **Emergency Contact List** is a BlackBerry 10 application that stores the contact information of the people in an enterprise who have been designated as “Emergency Contacts”. The contact list is updated by pushing a new list through the BlackBerry Enterprise Server (BES) MDS Connection Service.

The application displays the contact list to the user, where the user can scroll through the list for specific contacts. If the device is online, the application provides easy access to contact the list members by phone or email.

The application allows the user to browse through the emergency contacts, view the details of each contact and contact them via phone, message and Email.

The contact list is accessible to the user whether or not the device is online. The contact list is stored in a .JSON file. This JSON is used to populate the UI, when the application is open.

The application is built with BlackBerry 10 Cascades Qt/QML and C++.

### Scope

This document covers only client side application architecture. It will provide high level architecture of the application and a brief description of the major components.

### Acronyms / Definitions

|  |  |
| --- | --- |
| **Acronym /Definition** | **Description** |
| UI | User Interface |
| XML | eXtensible Markup Language |
| API | Application Programming Interface |
| BB10 | BlackBerry 10 |
| BDS | BlackBerry Device Service |
| JSON | JavaScript Object Notation |
| MDS | Mobile Data System |
| BES | BlackBerry Enterprise Server |
| ECL | Emergency Contact List Application |
| EIS | Enterprise Information System |
| CRUD | **C**reate, **R**ead, **U**pdate, **D**elete |

### References

The Emergency Contact List Application is a BlackBerry 10 Webworks Application that uses the following main features and APIs.

* Cascades is a toolkit to develop Native Blackberry 10 Applications using C++, Qt and QML. <http://developer.blackberry.com/cascades/>
* Cascades Push Service – Receives push message from the BES. <https://developer.blackberry.com/cascades/documentation/device_comm/push/>
* Cellular Call used to directly call a contact.

<http://developer.blackberry.com/cascades/reference/bb__system__phone__phone.html>

* Data Model- Used to populate the main list. <http://developer.blackberry.com/cascades/reference/bb__cascades__datamodel.html>
* Email Service used to send email to a contact.

<http://developer.blackberry.com/cascades/documentation/device_platform/invocation/email.html>

# Software Architecture

### Architecture Overview

1. The Push Test Service sends a request to the BES MDS Connection Service to push a new contact list to the device.
2. The push message is delivered through the Blackberry Push Service API to the application’s Push Handler.
3. The Push Handler adds the notification to the Hub and also adds a splat to the application icon to notify the user of a new Notification.
4. The Push expects a contact list that is encoded in the JSON format.
5. The application then writes this JSON data into a file.
6. The UI then uses this JSON file to populated the all the fields.
7. User can browse the contacts, view the contact details for each contact, and contact them via call, message and Email.

# Components Overview

This Application is divided into the following major components.

### ECL Model

* This is the data model that is used to populate the contact list screen and other screens in the application.
* The Data Model is populated using the contact list details stored in a JSON file.
* There is a filter data model that wraps around this model to provide a collapsible and expandable list which enables the user to see the contact list only department at a time.

### Push Handling Module

* Receives the Emergency contact list through Push
* Notifies the HUB

### Utility Module

* This is a common place for all utility functionality that is required by various other modules of the application.
* It invokes functionalities like making calls, sending messages and email, converting text files to JSON, showing dialogs, retrieving phone number list/email addresses for a contact from the JSON file , etc.

### Application UI

The application has 3 screens and one custom view.

* Main List Screen.
* Contact details screen.
* Phone list screen.
* Custom list item (used in the mail list screen).

### Main List Screen

* This screen is used to show the main Emergency Contact List. This is also the first screen that the user sees after starting the application.
* This screen is a collapsible and expandable list with all the emergency contacts grouped under different Groups.
* The user can expand or collapse each group by tapping on the group header.
* The user can call, text or email the contact by long tapping the contact and selecting the appropriate options from the context menu.

### Contact Details Screen

* This screen is used to show the extended details of each item in the emergency contact list.
* User can call, text or email the contact by tapping the appropriate list item in this screen.

### Phone List screen

* This screen is used to show the list of phone numbers if the contact has multiple phone numbers and the user wants to call the contact.

### Header List Item

* This is a custom view that was created to work as a header for each individual collapsible and expandable group of the emergency contact list.
* The existing header in the list view is too thin to be clicked accurately. Also it shows no visual difference when it is expanded or collapsed.
* The custom view is as wide as the list item and can be clicked easily.
* Also it has an arrow on the right most end that points up when it is expanded and down when it is collapsed.

# Implementation Details

### JSON format

The emergency contact list is stored in JSON format in the file system. This is used by the data model to populate the UI.

The easiest way to imagine this JSON is to look at it as a list of departments/groups. Each group is represented by a structure that has 2 items, a title for the group and a list that holds the list of contacts for that group. This list has many individual contact items. Each contact item is a data structure that has many items corresponding to the various names of Fields and values that a contact item typically has.

This table below gives a description of each of the field used in the JSON

The JSON seen in the next page is a typical contact list in the JSON Format.

|  |  |
| --- | --- |
| Field | Description |
| Title | Title of the Group. E.g. “Operations”, “Boston Team” |
| List | List of emergency contacts that are part of the that group. |
| Name | Name of the contact. |
| Role | Designation / Role of the contact. |
| Office Phone | Office Phone number of the contact. |
| Cell Phone | Cell Phone number of the contact. |
| Email | Email Address of the contact. |
| BBPin | Blackberry Pin of the contact. |
| Backup | The Backup contact who is assigned to the contact. |

JSON Format for Emergency Contact List

[

{

"Title": "Operations Team",

"list": [

{

"Name": "John Doe",

"Role": "Sr. Vice President, Operations",

"OfficePhone": "519 555-1111",

"CellPhone": "519 555-1222",

"Email": "john.doe@email.xyz.com",

"BBPin": "1111111F",

"Backup": "Jason Spencer"

},

{

"Name": "Jack Smith",

"Role": "Manager, Transportation Operations",

"OfficePhone": "519 555-3111",

"CellPhone": "519 555-3222",

"Email": "jack.smith@email.xyz.com",

"BBPin": "3333333F",

"Backup": "Julian Migs"

}

]

},

{

"Title": "Testing Team",

"list": [

{

"Name": "Steven Cho",

"Role": "Head Pachyderm Handler",

"OfficePhone": " 519 555-8111",

"CellPhone": " 519 555-8222",

"Email": "steven.cho@email.xyz.com",

"Backup": " Lucy"

},

{

"Name": "Rakesh Nambiar",

"Role": "Canine Specialist",

"OfficePhone": " 519 555-9111",

"CellPhone": " 519 555-9222",

"Email": "rakesh.nambiar@email.xyz.com",

"BBPin": "9996999F"

},

{

"Name": "Mika Sarinen",

"Role": "T-Rex Wrangler",

"OfficePhone": " 519 555-9111",

"CellPhone": " 519 555-9222",

"Email": "mika.sarinen@email.xyz.com",

"BBPin": "9925469F"

}

]

}

]

### Data Model

This module implements Data Modeling functionalities for the UI. This module consists of the ECLDataModel and the FilteredDataModel Classes. The ECL Data Model takes care of Parsing the contact info from the JSON file and supplying the information to the UI. The Filter Data Model wraps around the ECLDataModel and takes care of filtering the data corresponding to the collapsed list view items.

Both the data model classes are derived from *bb::cascades::DataModel*. They expose methods to retrieve the child count, item type, and actual data for each item in the list view. These methods are called when the QML Main List view needs to populate itself and the Contact Details View.

On Initialization the ECLData Model parses the JSON file to using *bb::data::JsonDataAccess* and populates an internal JSON-List which is a QVariantList. Then whenever there is a call to retrieve the child count, item-type or item-data for any item in the list, the ECLDataModel traverses this JSON-List and returns the required values.

When the Filter Data model is instantiated it takes an instance of the ECLDataModel also. It maintains the expansion states of each and every list item in the list. It exposes the same APIS that are exposed by the ECLDataModel. These APIS are what are called from the QML side.

When a particular group is expanded, the filter Data Model calls the corresponding methods in the ECLData Model. When the group is not expanded, the Filter Data model returns the child count for that item as Zero, thereby stopping all further requests for that Item.

When the Group is selected the Filter Data Model toggles the expansion state for that group, thereby enabling dynamic expansion/collapsing of the list view.

### Utilities Module

This module consists of a common class (EclUtils) where all the utilities are placed. These utilities methods are used by the other modules of the application. Following are the main few APIs exposed by it.

***API***: void makeCall(const QString& number);

* Invokes the call application to make a call to the given number.

***API***: bool sendEmail(const QString& recipient);

* This API invokes the email composer with the ‘to’ field populated with the email address of the intended recipient.

***API***: void sendSMS(const QString& number);

* This API has a dummy implementation, it just shows alert message

### Phonebook Integration Module

This module is a class that encapsulates the ‘saving-to-address-book’ functionality.

***API***: bool saveToAddressBook(QVariant eclData);

* This API parses through the QVariant Data that comes from the UI, retrieves the fields and values that need to be saved and saves all those fields together as one ‘Contact’ object into the device Address Book.

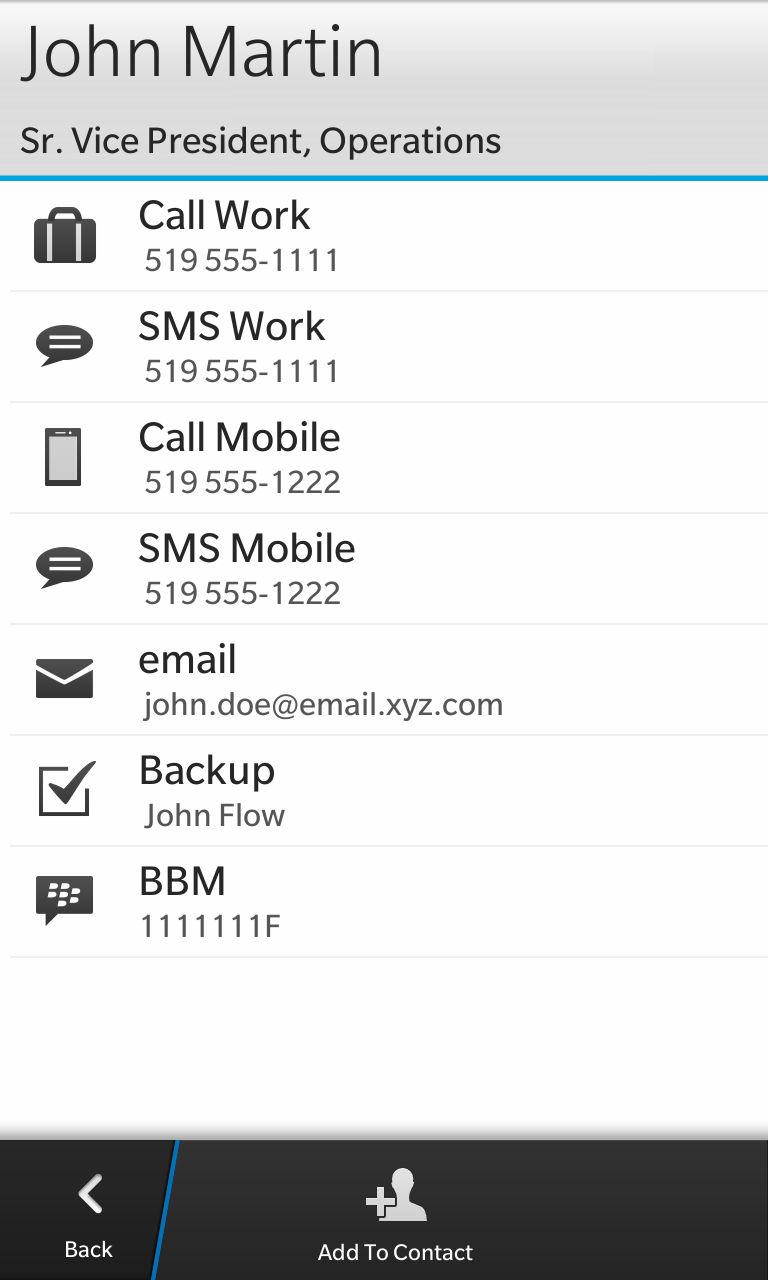
### Main Contact List Screen

This screen is the first screen that the user sees on entering the application. This screen is used to show the main Emergency Contact List. This is also the first screen that the user sees after starting the application. This screen is a collapsible and expandable list with all the emergency contacts grouped under different Groups. The user can expand or collapse each group by tapping on the group header. The user can call, text or email the contact by long tapping the contact and selecting the appropriate icon.

|  |  |  |
| --- | --- | --- |
| C:\Users\p.muralidharan.nair\Desktop\BBCodeMail\images\IMG_00000019.png | C:\Users\p.muralidharan.nair\Desktop\BBCodeMail\images\IMG_00000020.png | C:\Users\p.muralidharan.nair\Desktop\BBCodeMail\images\IMG_00000021.png |
| Main List Screen with all headers expanded. | Main List screen with only one header expanded | Main List Screen with context menu invoked |

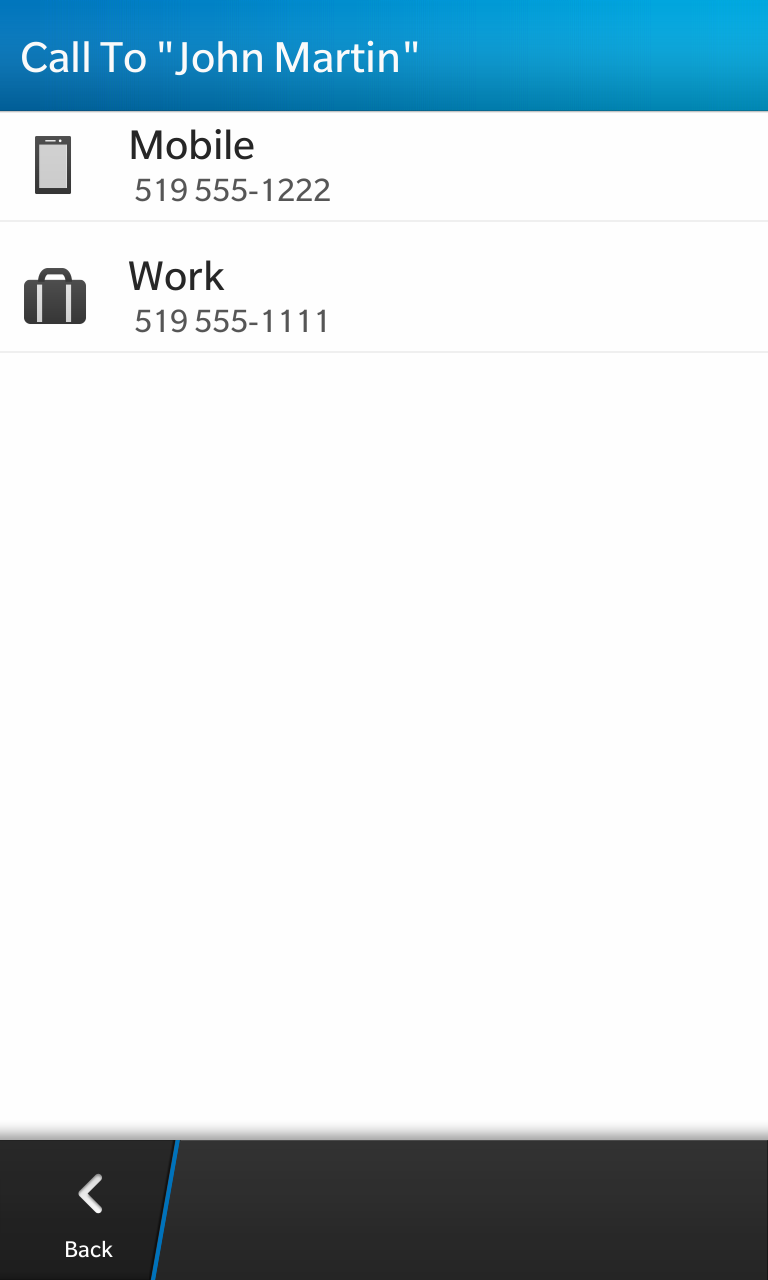
### View Details Screen

On the main screen when the user taps on any contact item, the details screen is opened. This screen contains the detailed contact info of the person. The user can call, send a message, or send an email to that contact from this screen.



### Phone list Screen

This screen is invoked when the user tries to call the contact from the context menu and if the contact has more than one phone number.



# Testing

### Test Environment

To test the application, a simple HTML page is used. Using this test page AJAX request submitted to the BDS to send a Push message to a device. See “server/index.html” under the project.

Here is an example of an HTTP request to the BDS server to send a Push message to the device identified by the email address (first.last@company.com).

*POST* [*http://1.1.1.1:8080/push?DESTINATION=first.last@company.com&PORT=requestlist&REQUESTURI=/*](http://1.1.1.1:8080/push?DESTINATION=first.last@company.com&PORT=requestlist&REQUESTURI=/) *HTTP/1.1*

*Content-Type: text/plain*

*Request Payload:*

*[*

*{*

*"Title": "Operations Team",*

*"list": [*

*{*

*"Name": "John Doe",*

*"Role": "Sr. Vice President, Operations",*

*"OfficePhone": "519 555-1111",*

*"CellPhone": "519 555-1222",*

*"Email": "john.doe@email.xyz.com",*

*"BBPin": "1111111F",*

*"Backup": "Jason Spencer"*

*},*

*{*

*"Name": "Jack Smith",*

*"Role": "Manager, Transportation Operations",*

*"OfficePhone": "519 555-3111",*

*"CellPhone": "519 555-3222",*

*"Email": "jack.smith@email.xyz.com",*

*"BBPin": "3333333F",*

*"Backup": "Julian Migs"*

*}*

*]*

*},*

*{*

*"Title": "Testing Team",*

*"list": [*

*{*

*"Name": "Steven Cho",*

*"Role": "Head Pachyderm Handler",*

*"OfficePhone": " 519 555-8111",*

*"CellPhone": " 519 555-8222",*

*"Email": "steven.cho@email.xyz.com",*

*"Backup": " Lucy"*

*},*

*{*

*"Name": "Rakesh Nambiar",*

*"Role": "Canine Specialist",*

*"OfficePhone": " 519 555-9111",*

*"CellPhone": " 519 555-9222",*

*"Email": "rakesh.nambiar@email.xyz.com",*

*"BBPin": "9996999F"*

*},*

*{*

*"Name": "Mika Sarinen",*

*"Role": "T-Rex Wrangler",*

*"OfficePhone": " 519 555-9111",*

*"CellPhone": " 519 555-9222",*

*"Email": "mika.sarinen@email.xyz.com",*

*"BBPin": "9925469F"*

*}*

*]*

*}*

*]*

### Testing without push infrastructure (Demo)

* There are also 2 “demo” action items in the overflow area of the Action Bar which can be used to test the application without Push infrastructure
* The “Demo with JSON File” option uses a pre-configured JSON file packaged with application (*assets/data/ecl\_data.json*) to populate the Contact List.
* The “Demo with Text File” option uses a text file packaged with the application (*assets/data/ecl\_data.txt*) which is converted to JSON format and then used to populate the contact List. This Option is to show that any format can be supported by application by writing a small function to convert the text data to JSON. The text data format packaged with application was used by the Emergency Contact List application on BB 7 platform. The *assets/data/config.json* is used to map the fields from text file to JSON file

# Issues and Resolution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl No.** | **Issue description** | **Resolution** | **Update/Comments** | **Current status** |
| 1 | Call not working on the Work Perimeter. | Used another solution: invoked the dialer which is used to make a call | Solution suggested by BB not working | OPEN |
| 2 | SMS sending is not working. | No Solution yet. | At the time of implementation the platform not supported it | OPEN |
| 3 | List item with Arrows (>, V) on the right side. | Created Custom List item. | A custom List item was created to fulfill this purpose. However it doesn’t give a native look and feel. | OPEN |

### Call Not working on the Work Perimeter:

**Challenge**:

Call feature not working in the work perimeter with the following Cascades API

*bb::system::phone::Phone phone;*

*phone.initiateCellularCall(number);*

**Solution from BB:**

Apply the "<permission>access\_phone</permission>" line to the bar-descriptor.xml file to allow phone access in the Work Perimeter from Cascades.

However this does not work.

### SMS sending is not working:

**Challenge**:

Send SMS feature is not working in the present version of the platform.

**Workaround:**

No workaround

### List item with Arrows (>, V) on the right side:

**Challenge**:

There was need for a list item that had an arrow on the right side to indicate that tapping this item would open another screen. There is no support for this in cascades though such support is there in Webworks.

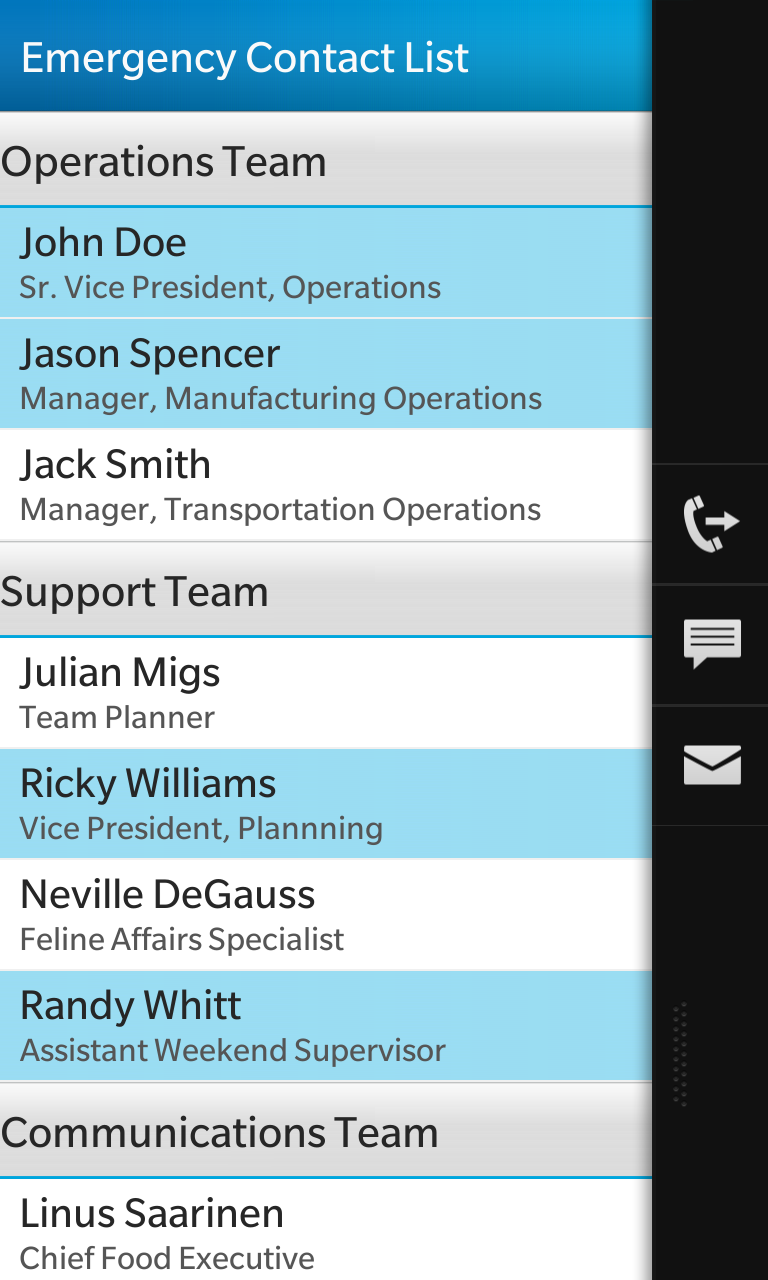
**Workaround:**

Since there was no direct solution to this problem, the work around used a custom item.

**Approach 1: using QML**

created the custom list Item using the QML, the CustomListItem.qml is attached here

**issue**: once we select the item it is highlighted forever, this gives impression of multi-item selection in the list view, snapshot is given below

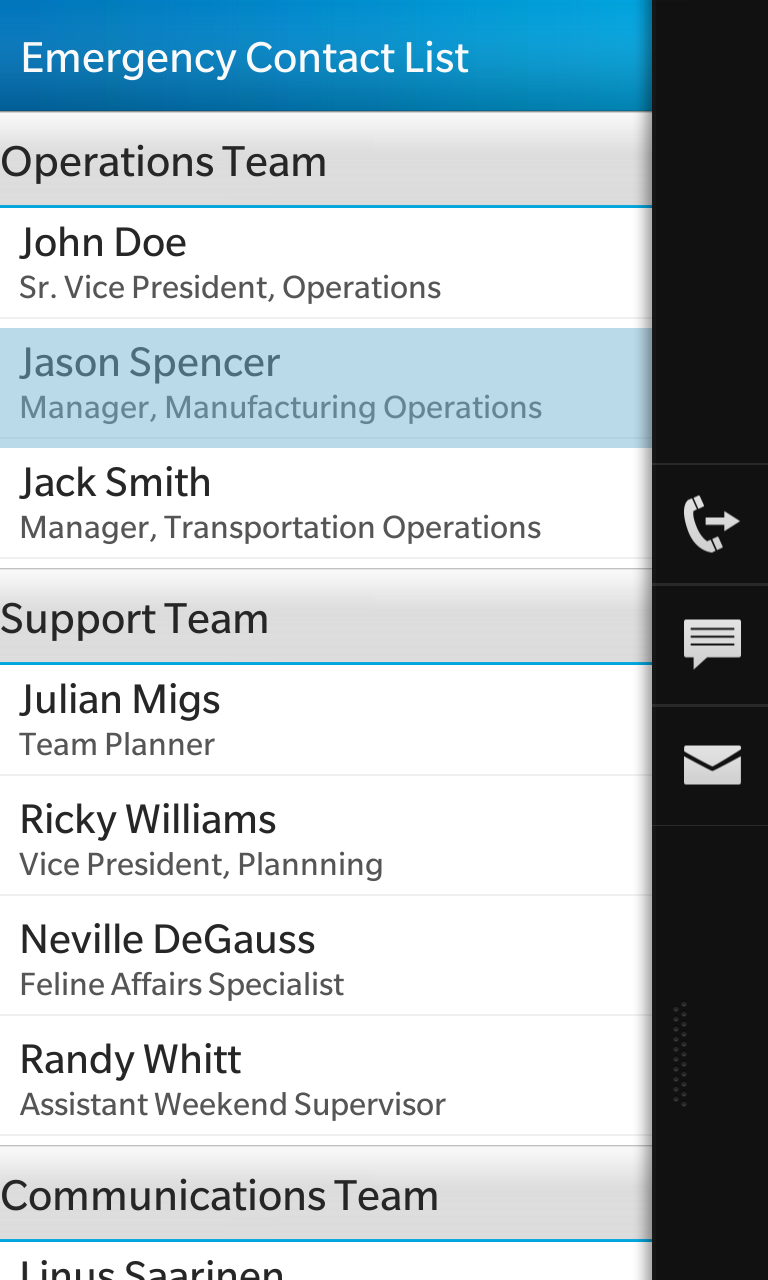
 

**Approach 2: using custom QT class**

Created custom class, ecllistitem.cpp and ecllistitem.hpp

Used an highlighter container for responding to the user tapping/selection by playing with the opacity of the highlighter

**Issue**: this doesn’t give the native look and feeling for the tapping/selection part of the item



The approach 2 is implemented in the application