Ubuntu Component Store Documentation

Release 0.1

Nekhelesh Ramananthan

April 14, 2015

Contents

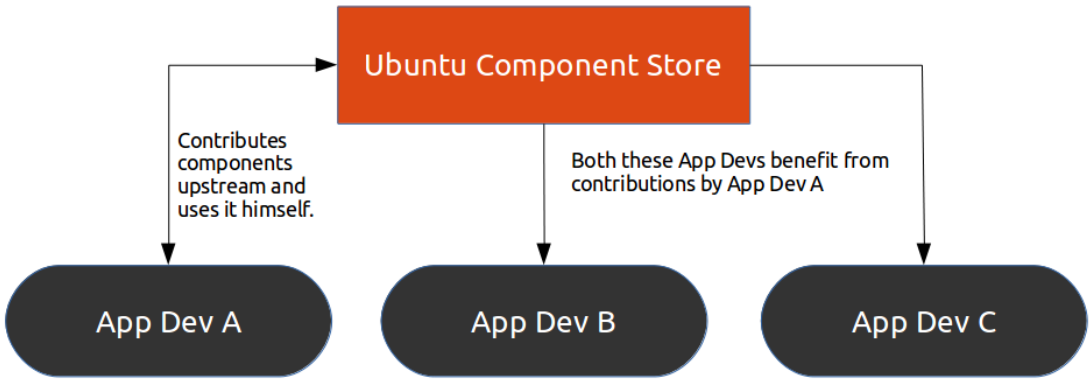
The Ubuntu Component Store (UCS) is intended to be a library hosting a collection of components created by community developers for building Ubuntu SDK apps. UCS will enable new developers by providing plug-and-play components/modules and thereby preventing the need to start from scratch.

CHAPTER 1

Purpose

What's the usecase for this? Let's take a look at an example. Assume *App Dev A* creates a component that is intuitive and helps improve the user experience. *App Dev B* would like to incorporate a similar component in his own app and either borrows the code from *App Dev A* or creates his own. Both choices have serious drawbacks. If he borrows the code, then he has to keep track of any upstream improvements. On the other hand, if he creates his own component, then it is just a waste of effort since there is now code duplication throughout the community. This is what the Ubuntu Component Store is aimed at solving.

By hosting a set of components, UCS will enable app developers to share components with their peers and reduce code duplication.



CHAPTER 2

How will this work?

A developer who just wants to use these components will be able to grab the components easily and keep them updated by simply running the ucs program.

You can find the installation guide, to install *ucs* so that you can use it to add components to your Ubuntu SDK apps, at *Installation Guide*.

A component developer can create a component and contribute it to the Ubuntu Component Store. There are two types of components: *Curated* and *Community*. A component developer can publish their component to the Community store by themselves, with no oversight. Publishing to the Curated store requires approval that the curated component is well-constructed: it must have good documentation and tests, and the component developer will join the store team and maintain their component. This leads to the Curated components being superior.

If you want to contribute a component, you can find the contribution guide at *Contributing to UCS*.

CHAPTER 3

Main Contents

3.1 Installation Guide

3.1.1 Installing UCS itself

Before using *ucs* to install components, you first need to install *ucs* itself!

To install *ucs*, we need to add a PPA

```
sudo add-apt-repository ppa:ubuntu-touch-community-dev/ppa
sudo apt-get update
sudo apt-get install ucs
```

3.1.2 Finding Components

Use the *search* command to search for available components by name or description.:

```
$ ucs search state
Matching curated components:
                      EmptyState
Matching community components:
                     (no matches)
$ ucs search podcast
Matching curated components:
                     (no matches)
Matching community components:
           sil/GenericPodcastApp  A component which manages a podcast RSS feed, with playback and dis
```

You can list all components in the store, both curated and community components, with *ucs search*.

3.1.3 Installing Components

You can install components individually using the *install* command,

```
ucs install EmptyState
ucs install PageWithBottomEdge
ucs install sil/GenericPodcastApp
```

Note: Remember to run the *ucs* command from inside your application's component folder!

You can then use the components by importing it in your qml files by,

```
import ubuntu_component_store.Curated.EmptyState 1.0
import ubuntu_component_store.Curated.PageWithBottomEdge 1.0
import ubuntu_component_store.sil.GenericPodcastApp 1.0
```

Note: Community components are named as *developer/ComponentName*. Curated components are installed with just a ComponentName (like *ucs install EmptyState*), but when importing them in QML you must call them *ubuntu_component_store.Curated.EmptyState* as above.

3.1.4 Updating Components

You can also update the UCS components using the *ucs* script. It will automatically fetch the latest upstream code.

```
ucs update EmptyState
```

Note: Remember to run the *ucs* command from inside your application's component folder!

You can see the full ucs script help using the help argument,

```
ucs help
```

3.2 Contributing to UCS

There are two sorts of components in the Ubuntu Component Store: *Curated* components and *Community* components. As a developer, you can publish your component to either. Components in the Curated store have higher requirements to ensure that they are high-quality; anyone can publish to the Community store without approval.

3.2.1 Contributing a component

Contributing a component to the Curated store

Ubuntu Component Store is intended to be a community project where app developers are encouraged to contribute any component that they think would be useful to other developers. To contribute a component to UCS, we recommend first joining the team by applying here. This would then make it possible for you to maintain your component by pushing bug fixes and improvements as and when you choose.

By moderating the members of the team and components in UCS, we can promise a well tested component with a stable API. We recognize that this can impede the rate at which new components can be added to UCS, but we feel that this is an acceptable drawback to ensure good quality components to our userbase.

Let's go through the steps required to upload a new component to the store.

Getting Started

You can technically contribute new components to the store without being a member of the team by requesting someone from the team to maintain your component. However we highly recommend that you join the team since that would allow you to push bug fixes and improvements to your component quickly. You can apply to become a member by applying here. The approval should be done within a day or two.

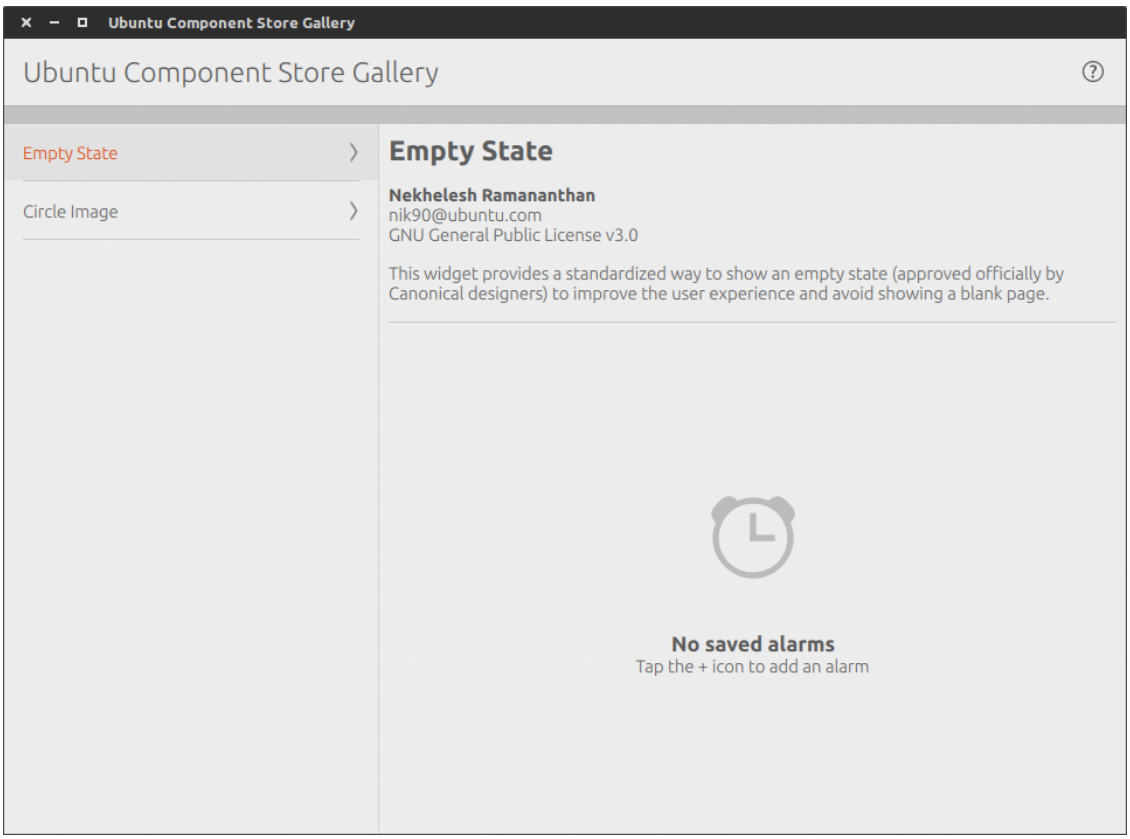
UCS is hosted on launchpad and requires bazaar (bzr) to grab or push the code. As such you would need to have an account on launchpad and be familiar with bzr. You can grab the code by,

```
bzr branch lp:component-store
```

UCS by itself is an ordinary QML project. You should be able to open it using qtcreator like any other project. Run the UCS Gallery app by either pressing the green button in qtcreator or via the command line as shown below,

```
qmlscene main.qml
```

This should open the UCS Gallery app as shown below. It provides a visual overview of all the components in the store.



Note: At the time of writing this documentation, you would need a 14.10 desktop to run the component store gallery. You can run the gallery app on the phone or emulator using a 14.04 desktop, however the gallery app doesn't converge well on the phone yet. This should be fixed soon.

Adding a new component to the store

Adding a new component to the store involves 3 main steps which are,

1. Adding a new component to the ComponentStore folder
2. Updating the gallery app to showcase your new component
3. Updating the documentation

Adding a new component to the ComponentStore Folder The components are stored in their own folders in the ComponentStore folder. Let's assume for illustration purposes, we are trying to add a new component *MyNewComponent* to UCS. Let's grab UCS first and then add the new component.

```
bzr branch lp:component-store MyNewComponentBranch
```

Let create a new folder for our component in the ComponentStore folder,

```
cd MyNewComponentBranch/ComponentStore
mkdir MyNewComponent && bzr add MyNewComponent
cd MyNewComponent
```

Inside your *MyNewComponent* folder, add your generic component files. Everything that is required to use your component must be included in this folder. Let's now add these files to the source control bzr and commit them.

```
bzr add *
bzr commit -m "Added my new component"
```

Updating the gallery app Now that we have our component ready and added, let's update the gallery app to show that. Create a new file in the GallerySRC folder called MyNewComponentWidget.qml. This file is rather easy to fill out. Just copy the code from the other widget template files. It basically involves filling in information like the author, license, contact and a description of your component which is rather simple. Once completed, add MyNewComponentWidget.qml to the version control.

```
cd GallerySRC
bzr add MyNewComponentWidget.qml
bzr commit -m "Showcase new component in the gallery"
```

Next, open WidgetsModel.qml and add your component to the list. That's it! Run the gallery app and see if your component shows up as expected.

Updating the documentation This is one of the most important steps and benefits of adding your component to UCS. By providing a well written and clear documentation you make it easier for other app developers to use your component in their app. All the documentation is hosted in the docs folder. Create a new documentation file in the _components folder and fill in the necessary documentation. Use the existing documentation files to help you with setting it up.

Once done, add your component to the list in index.rst. Now let's test if the documentation looks good. To build the documentation, you need sphinx and pip packages to be installed. Let's install that for the first time.

```
sudo apt-get install pip
sudo pip install sphinx sphinx-autobuild
```

Building your documentation is now really simple. Once the build process is complete, open _build/index.html in your browser to see your documentation.

```
make html
```

Once again, let's add this to the version control.

```
cd docs/_components
bzr add MyNewComponent.rst
bzr commit -m "Added documentation for the new component"
```

That was it! Your component is ready. Let's push this online to UCS.

```
bzr push lp:~launchpad-id/component-store/MyNewComponentBranch
```

From there on, it is just a matter of reviewing the code (for new components, we try to ensure everything is in order) and then merging to trunk.

Contributing a component to the Community store

To publish an Ubuntu SDK component to the Community store, you will need a Launchpad account, and you will need to be familiar with using bzr to push code to Launchpad.

Community components can be pure QML, or they can be compiled binary components.

Publishing a pure QML component

Create a branch on Launchpad (in any project of your choice) which is named how you plan to name the component, with a top-level *qml* folder, and put your QML file and any required assets for your component in that qml folder. Add an ubuntu_component_store.json file. Then, submit the component to the community store with *ucs submit lp:~username/project/branch*.

So, your branch should look like:

```
/qml/
     MyComponent.qml
     required_image.png
     included_script.js
/ubuntu_component_store.json
```

If your component does not have any external assets, it is fine to have a branch with *qml/MyComponent.qml* and nothing else in it. Your branch may also contain any other files you choose outside the *qml* top-level folder; these files are not installed with your component, but may provide useful guidance, READMEs, or other code for people who want to make changes to the component and contribute them back to you.

The name of the component itself is defined by ubuntu_component_store.json; the QML filename is what names the QML Item that it provides. So, if you are Launchpad user *sil*, and you push your component to *lp:~sil/SomeProject/mything*, it defines its own name in ubuntu_component_store.json as *UsefulComponent*, and it contains *qml/RedRectangle.qml*, an app developer will use it like this:

```
import QtQuick 2.0
import ubuntu_component_store.sil.UsefulComponent 1.0
....
MainView {
    RedRectangle {
        ....
    }
}
```

Most components should have component name and QML file name be the same thing to avoid confusion.

Do not publish two unrelated QML components in one UCS component; publish them separately, so they can be used separately.

Note: if you do not want to create a whole Launchpad project just for this component, you can push to a Launchpad "junk" branch: *lp:~username/+junk/somename*

Once your branch is created, publish it to the Community store with

```
$ ucs submit lp:~username/project/somename
(submitting to community repository)
Checking Launchpad branch lp:~username/project/somename
```

```
Checks passed OK
Calculated package summary data
Updating master record
Component username/ComponentName updated successfully
```

It should then be visible to *ucs search*.

Note: Once added to the community store, there is no way (yet) to remove your component.

Publishing a compiled component

Publishing a compiled component is a little more complicated, for CPU architecture reasons. Your component must be an Extension Plugin, "a plugin library to make it available to the QML engine as a new QML import module". Creating such a plugin is currently beyond the scope of this document. (We are hoping to provide an Ubuntu SDK IDE template for creating such components with the proper filesystem layout; before then, the "App with QML Extension Library" option creates an appropriate type of component.)

You *must* compile your component for three different architectures: ARM, x86, and amd64 (for Ubuntu phones, the desktop emulator, and Ubuntu desktops). Once you have compiled it as such, you will have three different *libMyComponent.so* files.

Assemble a Launchpad branch with a top-level *qmllib* folder, and in it put a folder for each architecture, named for the GNU architecture triplet, and then the *.so* file within. So:

```
/qmllib
    /x86_64-linux-gnu
                    /libMyComponent.so
    /i386-linux-gnu
                    /libMyComponent.so
    /arm-linux-gnueabihf
                    /libMyComponent.so
/ubuntu_component_store.json
```

Add an ubuntu_component_store.json file to the root of the branch.

Your branch may contain any other files of your choice outside the */qmllib* folder; in particular, it should contain the source code for the plugin so others can build it themselves if they choose!

The name of the component itself is defined by ubuntu_component_store.json; your component is expected to use qmlRegisterType to provide QML Item types. So, if you are Launchpad user *sil*, and you push your component to *lp:~sil/SomeProject/Whatever*, it defines its name in ubuntu_component_store.json as *SomeComponent*, and it registers a *Triangle* type, an app developer will use it like this:

```
import QtQuick 2.0
import ubuntu_component_store.sil.SomeComponent 1.0
....
MainView {
    Triangle {
        ....
    }
}
```

Do not publish two unrelated components in one UCS component; publish them separately, so they can be used separately.

Note: if you do not want to create a whole Launchpad project just for this component, you can push to a Launchpad "junk" branch: *lp:~username/+junk/somename*

Once your branch is created, publish it to the Community store with

```
$ ucs submit lp:~username/project/somename
(submitting to community repository)
Checking Launchpad branch lp:~username/project/somename
Checks passed OK
Calculated package summary data
Updating master record
Component username/ComponentName updated successfully
```

It should then be visible to *ucs search*.

A component can contain both *qml* and *qmllib* folders and so contain both QML parts and binary parts; both will be installed when a developer uses *ucs install* to install your component.

ubuntu_component_store.json

A community component must contain a file *ubuntu_component_store.json* describing metadata about the component. It must be valid JSON, with keys *name*, *version*, and *description*:

```
{
    "name": "GenericPodcastApp",
    "version": "1.0",
    "description": "A component which manages a podcast RSS feed, with playback and display of episo
}
```

Other keys may be added in future. Note that the public name of the component will be launchpadusername/name, where *name* is the name taken from *ubuntu_component_store.json*. The branch name in Launchpad is ignored.

3.3 Release Notes

13th April 2015

- Reorganized UCS by adding a Curated and Community Store.
- Added Bottom Edge Tabs component

23rd March 2015

- Added haptic feedback to radial action buttons
- Added drop shadow to radial bottom edge hint

13th March 2015

- Improved ListItemWithActions components by adding the following new properties,
  - showDivider - mimics SDK ListItem property
  - showUnderscore - Shows an underscore under the active right side action
  - enableHaptics - enables haptic effects on actions triggered

11th March 2015

- Added expandAngle property to radial bottom edge component to allow developers to specify the spread angle.

3rd March 2015

- Added WelcomeWizard component

- Fixed a typo in listitemwithactions documentation

23rd February 2015

- ListItemWithActions: Reset active action after executing it.
- ListItemWithActions has been fixed to use actions's iconSource property to allow using icons that are not in the theme

4th January 2015

- Add ubuntu_component_store.json metadata files for curated components

15th November 2014

- Added radial bottom edge component

12th November 2014

- Fixed typos in the installation guide
- Added ListItemWithActions component

9th November 2014

- Fixed build warnings in CircleImage and PageWithBottomEdge documentation
- Added release notes to the documentation
- Released ucs 0.1.2
  - Improved intall and update command to use less network bandwidth
  - Fixed typos in the help command

8th November 2014

- Added Fastscroll Component
- Packaged ucs 0.1.1 and released to trusty, utopic and vivid using a PPA
- Improved index.rst and fixed typos

6th November 2014

- Added PageWithBottomEdge component
- Added iconColor property to EmptyState component
- Added images to EmptyState and CircleImage to make it more descriptive for users

4th November 2014

- First release of Ubuntu Component Store
- Added UCS Gallery
- Added EmptyState and CircleImage components

CHAPTER 4

Components

4.1 Bottom Edge Tabs

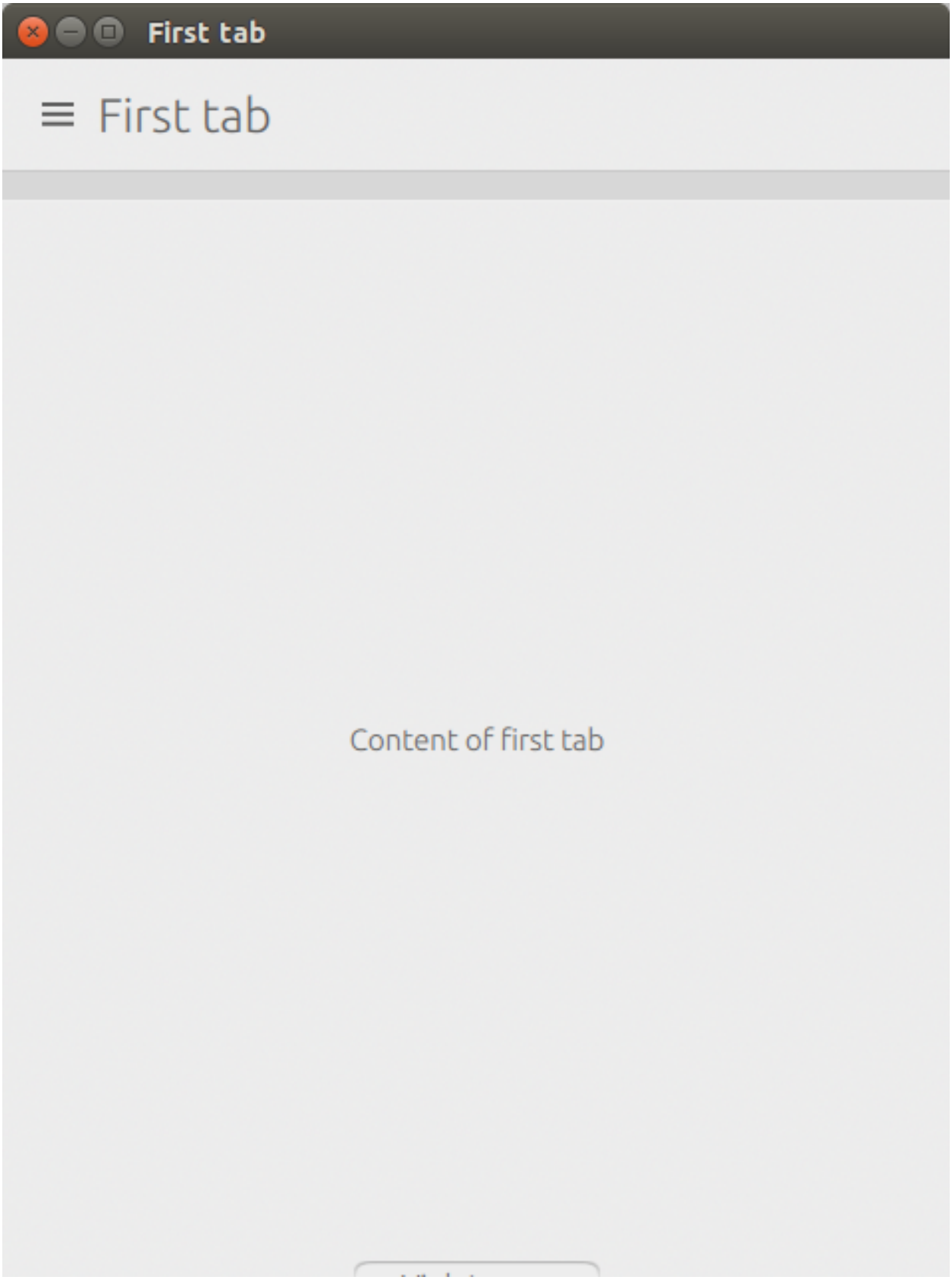
Author	Roman Shchekin
License	GNU General Public License v3.0
Contact	mrqtros@gmail.com
Framework	ubuntu-sdk-14.10

BottomEdgeTabs is very similar to PageWithBottomEdge but uses Tabs as root component instead of Page. Check it's documentation.

Example:

```
MainView {
            objectName: "mainView"

            applicationName: "com.ubuntu.developer.qtros.tabsbottomedge"

            width: units.gu(50)
            height: units.gu(75)
            useDeprecatedToolbar: false

            PageStack {
                id: stack
                Component.onCompleted: push(tabs)

                BottomEdgeTabs {
                    id: tabs

                    bottomEdgePage: secondPage
                    bottomEdgeTitle: "Violet page"

                    Tab {
                        title: "First tab"
                        page: Page {
                            Label {
                                anchors.centerIn: parent
                                text: "Content of first tab"
                            }
                        }
                    }

                    Tab {
```

```
                    title: "Second tab"
                    page: Page {
                        Label {
                            anchors.centerIn: parent
                            text: "Centered label"
                        }
                    }
                }
            } // BottomEdgeTabs

            Page {
                id: secondPage
                title: "Violet page"
                visible: false

                Rectangle {
                    anchors.fill: parent
                    color: "darkviolet"
                }
            }
        }
    }
```


First tab
First tab
Content of first tab

4.1.1 Properties

- *bottomEdgeTitle*: string
- *bottomEdgePage*: Page
- *bottomEdgeEnabled*: boolean

4.1.2 Signals

- *bottomEdgeReleased()*
- *bottomEdgeDismissed()*

4.1.3 Property Documentation

bottomEdgeTitle

The text to be displayed in the bottom edge action.

bottomEdgePage

The page to be shown when swiping the bottom edge up.

bottomEdgeEnabled

Boolean property to enable/disable the bottom edge

4.1.4 Signal Documentation

bottomEdgeReleased()

This handler is called when the bottom edge is let go.

bottomEdgeDismissed()

Called when the bottom edge is dismissed (hidden).

4.2 Circle Image

Author	Michael Spencer
License	GNU General Public License v3.0
Contact	mspencer@sonrisesoftware.com
Framework	ubuntu-sdk-14.10

This widget converts any image into a circular sized image which can be useful for showing user profile pictures. As trivial as this might look, it is not a straightforward solution in QML.

Example:

```
CircleImage {
    width: units.gu(10)
    height: width
    source: Qt.resolvedUrl("assets/flashback.png")
}
```

4.2.1 Properties

- *source*: url
- *status*: enum

4.2.2 Property Documentation

source

The source url of the image to display.

status

The loading status of the image

4.3 Empty State

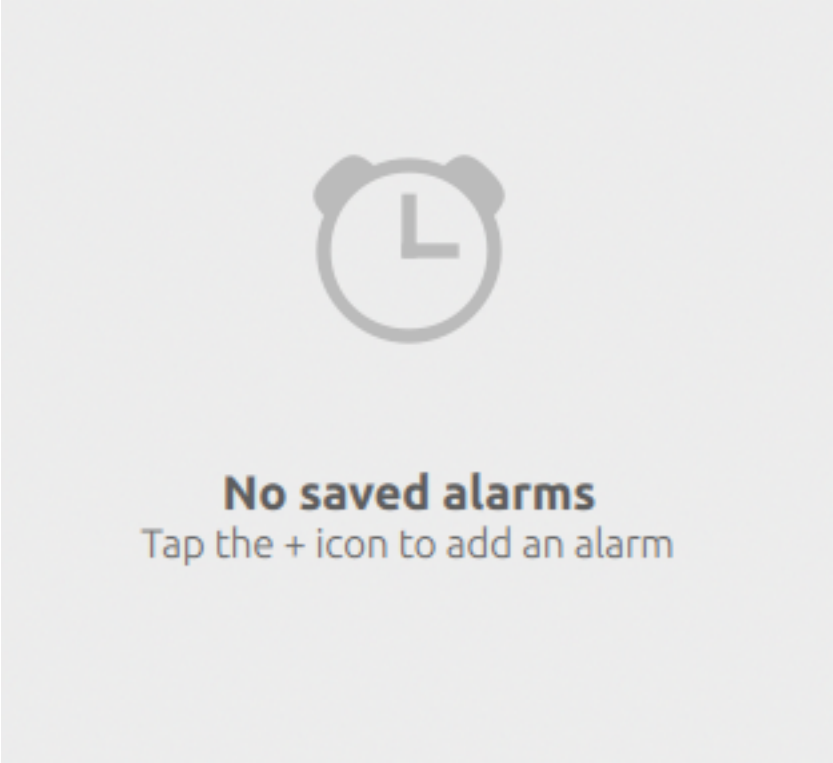
Author	Nekhelesh Ramananthan
License	BSD License
Contact	nik90@ubuntu.com
Framework	ubuntu-sdk-14.10

This widget provides a standardized way to show an empty state (approved by Canonical designers) to improve the user experience and avoid showing a blank page. This way the user is not left starring a blank page but instead is shown the empty state which informs the user of situation along with some recommendations.

For instance, in the clock app if the user has no alarms saved, then an empty state is shown and encourages the user to create a new alarm.

Example:

```
EmptyState {
    iconName: "alarm-clock"
    title: i18n.tr("No saved alarms")
    subTitle: i18n.tr("Tap the + icon to add an alarm")
    anchors.centerIn: parent
}
```



4.3.1 Properties

- *iconName*: string
- *iconSource*: url
- *iconColor*: color
- *title*: string
- *subTitle*: string

4.3.2 Property Documentation

iconName

The name of the icon to display. Valid icon names can be found in the suru-icon-theme.

Note: If both *iconName* and *iconSource* are specified, then *iconName* will be ignored.

iconSource

The source url of the icon to display. It has precedence over *iconName*.

iconColor

The color of the icon.

Note: This property only works if iconName is used.

---

title

The main message of the empty state. Try to keep this short and precise.

subTitle

A more descriptive message of the empty state. Usually the subtitle provides tips or instructions to perform an action to avoid the empty state.

4.4 Fast Scroll

Author	Renato Araujo Oliveira Filho
License	BSD License
Contact	renato.filho@canonical.com
Framework	ubuntu-sdk-14.10

This widget provides a quick way to navigate long list views by providing a fast scroll. Example use cases are scrolling through an address book with a long list of contacts. It shows all the characters from A-Z but only the ones shown in the listview are differentiated with a bold font. The currently selected section is highlighted with a black rectangle. As the user scrolls the listview, the selected section rectangle also scrolls automatically with an animation.

Note: At the moment, fast scroll only accepts listviews to be ordered alphabets and does not work well with special characters.

Example:

```
Page {
    id: fastscrollpage

    ListModel {
        id: testModel
        ListElement { title: "Alan Pope" }
        ListElement { title: "Aditya Urs" }
        ListElement { title: "Akiva Shammai Avraham" }
        ListElement { title: "Andrew Hayzen" }
        ListElement { title: "Carla Sella" }
        ListElement { title: "Daniel Holm" }
        ListElement { title: "Daniel Holbhach" }
        ListElement { title: "David Planella" }
        ListElement { title: "Lento Manickathan" }
        ListElement { title: "Leo Arias" }
```

```
        ListElement { title: "Michael Hall" }
        ListElement { title: "Michael Zanetti" }
        ListElement { title: "Mihir Soni" }
        ListElement { title: "Michael Spencer" }
        ListElement { title: "Nicholas Skaggs" }
        ListElement { title: "Nekhelesh Ramanthan" }
        ListElement { title: "Oliver Gravert" }
        ListElement { title: "Jenkins" }
        ListElement { title: "Kunal Parmar" }
        ListElement { title: "Riccardo Padovani" }
        ListElement { title: "Robert Schroll" }
        ListElement { title: "Victor Thompson" }
    }

    ListView {
        id: nameListView
        anchors.fill: parent
        anchors.rightMargin: fastScroll.showing ? fastScroll.width - units.gu(1)
                                                : 0
        clip: true
        currentIndex: -1
        model: testModel

        function getSectionText(index) {
            return testModel.get(index).title.substring(0,1)
        }

        delegate: ListItem.Standard {
            text: title
        }
    }

    FastScroll {
        id: fastScroll
        listView: nameListView
        anchors {
            top: nameListView.top
            bottom: nameListView.bottom
            right: parent.right
        }
    }
}
```

4.4.1 Properties

- *listview*: listview
- *showing (readonly)*: bool
- *enabled*: bool

4.4.2 Methods

- *function getSectionText(index)*

4.4.3 Property Documentation

listview

The listview which requires the fastscroll component

showing (readonly)

Readonly property to return the visiblity status of the fast scroll component. The fast scroll is automatically hidden after few seconds. This property will help with defining behaviour based on the visiblity of the component.

enabled

This property can be used to enable/disable the fastscroll. For instance when the listview has too few elements, it might be better to disable the fastscroll.

Here is a code sample that disables the fastscroll when the listview has too few elements,

```
enabled: (listview.contentHeight > (listview.height * 2)) && (listview.height >= minimumHeight)
```

4.4.4 Method Documentation

function getSectionText(index)

This function returns the available section headers of a listview to the fast scroll and is required for the fastscroll to know which indexes to highlight. See the example above to get a better idea of how to use this.

4.5 List Item With Actions

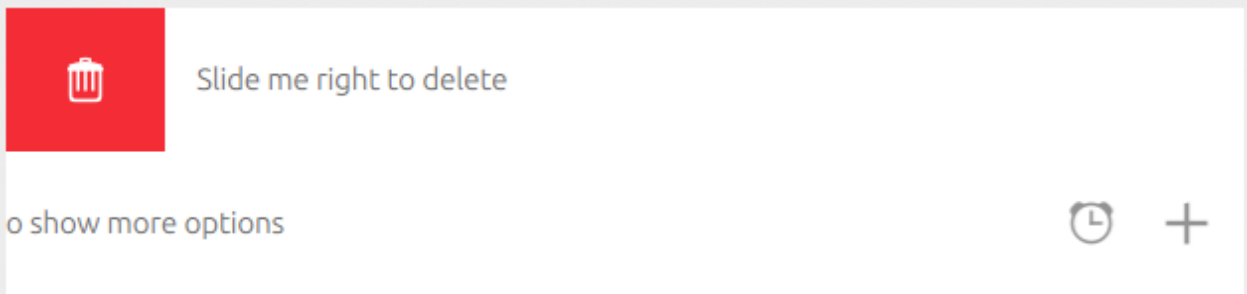
Author	Renato Araujo Oliveira Filho
License	GNU General Public License v3.0
Contact	renato.filho@canonical.com
Framework	ubuntu-sdk-14.10

This widget provides an updated listitem which is what the core apps currently use. These listitems will be provided by the SDK with vivid onwards. However current RMT devices will ship with the ubuntu-sdk-14.10 framework which wouldn't have these latest listitems.

Example:

```
Page {
    id: listitemwithactionspage

    ListModel {
        id: testModel
        ListElement { title: "Slide me right to delete" }
        ListElement { title: "Slide me left to show more options" }
    }

    ListView {
        id: nameListView
        anchors.fill: parent

        clip: true
        model: testModel

        delegate: ListItemWithActions {
            height: units.gu(9)
            width: parent.width
            color: "White"
            triggerActionOnMouseRelease: true

            leftSideAction: Action {
                iconName: "delete"
                text: i18n.tr("Delete")
                onTriggered: {
```

```
                testModel.remove(nameListView.index)
            }
        }

        rightSideActions: [
            Action {
                iconName: "alarm-clock"
                text: i18n.tr("Alarm")
            },

            Action {
                iconName: "add"
                text: i18n.tr("Add")
            }
        ]

        contents: Label {
            text: title
            anchors.left: parent.left
            anchors.verticalCenter: parent.verticalCenter
        }
    }
  }
}
```



4.5.1 Properties

- *color*: color
- *contents*: Item
- *internalAnchors*: anchors
- *leftSideAction*: Action
- *rightSideActions*: Action <List>
- *locked*: boolean (false by default)
- *triggerActionOnMouseRelease*: boolean (false by default)
- *showDivider*: boolean (false by default)
- *showUnderscore*: boolean (false by default)
- *enableHaptics*: boolean (false by default)

4.5.2 Property Documentation

color

The background color of the list item.

contents

This property is used to define the contents of the list item. Unlike the SDK list items which only assigning values to the text property, the contents allows you to define whatever content you so wish. In the example above, a Label is used to show content in the list item. This could very well be replaced with a column with multiple labels with whatever formatting you choose. For example,

```
contents: Column {
    spacing: units.gu(2)
    anchors.fill: parent
    Label {
       id: title
       text: Test"
       font.bold: true
    }

    Label {
        id: subTitle
        text: "SubTitle"
    }
}
```

internalAnchors

Internal anchors allows you to define the anchors of the contents. The values are already defined by default, but if you so wish, you could change the anchors to better suit your application. For instance changing the top anchor of the contents can be done by,

```
internalAnchors.topMargin: units.gu(0)
```

leftSideAction

According to design, the left side of a list item can include only one action. An action can be defined easily by,

```
Action {
    iconName: "add"
    text: "Add"
    onTriggered {
        doSomething()
    }
}
```

rightSideActions

On right side of a list item, one can include a list of actions. Obviously it would be recommended to not add more than 3 actions since space is limited.

```
rightSideActions: [
    Action {
        iconName: "alarm-clock"
        text: i18n.tr("Alarm")
    },

    Action {
        iconName: "add"
        text: i18n.tr("Add")
    }
]
```

locked

This property can be used to lock the actions of a list item (essentially enabling/disabling them).

triggerActionOnMouseRelease

This property affects the right side actions behavior. If set to true, the user can swipe left to reveal the right side actions and execute an action by just hovering over it. By default, this is set to false meaning that the user needs to press on the action to trigger it.

showDivider

This property can be used to display a thin divider along the bottom of the list item.

showUnderscore

This property when enabled displays an underscore underneath the active action in the rightSideActions.

enableHaptics

This property enables haptic feedback when triggering actions in the list item.

4.6 Page With Bottom Edge

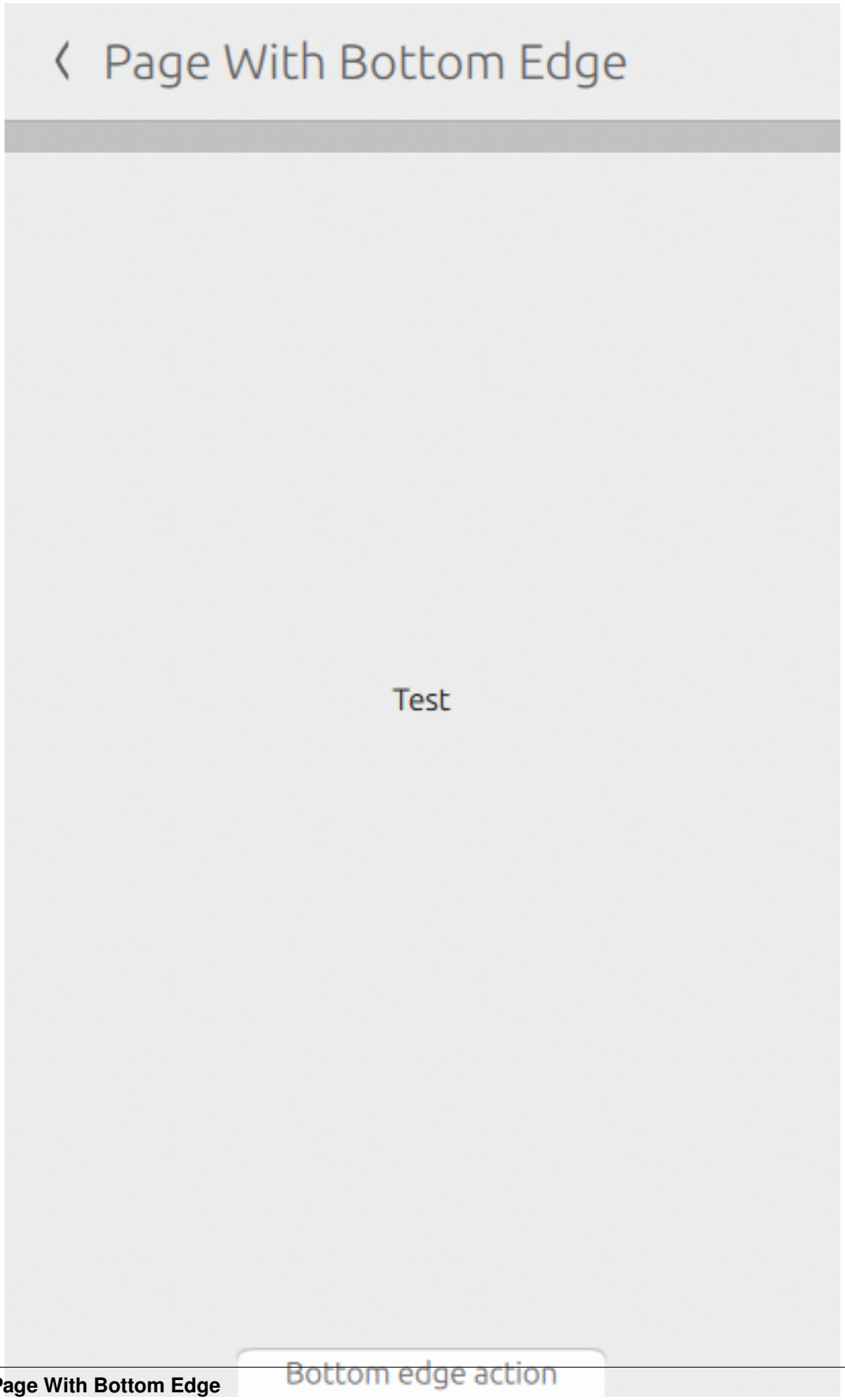
Author	Renato Araujo Oliveira Filho
License	GNU General Public License v3.0
Contact	renato.filho@canonical.com
Framework	ubuntu-sdk-14.10

This component provides a bottom edge which can be used to house common actions. There is only one bottom edge available for a page. Only one. The user can use it without looking where they are pressing. Think carefully about which action to put there.

For instance, the clock app uses the bottom edge to show the alarms saved by the user. While a memo creating app like Quick Memo uses it to create a memo.

Example:

```
MainView {
    objectName: "mainView"

    applicationName: "com.ubuntu.developer.boiko.bottomedge"

    width: units.gu(100)
    height: units.gu(75)

    Component {
        id: pageComponent

        PageWithBottomEdge {
            id: mainPage
            title: i18n.tr("Main Page")

            bottomEdgePageComponent: Page {
                title: "Contents"
                anchors.fill: parent

                ListView {
                    anchors.fill: parent
                    model: 50
                    delegate: ListItems.Standard {
                        text: "One Content Item: " + index
                    }
                }
            }
            bottomEdgeTitle: i18n.tr("Bottom edge action")
        }
    }

    PageStack {
        id: stack
        Component.onCompleted: stack.push(pageComponent)
    }
}
```


Page With Bottom Edge
Test
Bottom edge action

4.6.1 Properties

- *bottomEdgeTitle*: string
- *bottomEdgeEnabled*: boolean
- *bottomEdgePageComponent*: Page

4.6.2 Signals

- *bottomEdgeReleased()*
- *bottomEdgeDismissed()*

4.6.3 Property Documentation

bottomEdgeTitle

The text to be displayed in the bottom edge action.

bottomEdgePageComponent

The page to be shown when swiping the bottom edge up.

bottomEdgeEnabled

Boolean property to enable/disable the bottom edge

4.6.4 Signal Documentation

bottomEdgeReleased()

This handler is called when the bottom edge is let go.

bottomEdgeDismissed()

Called when the bottom edge is dismissed (hidden).

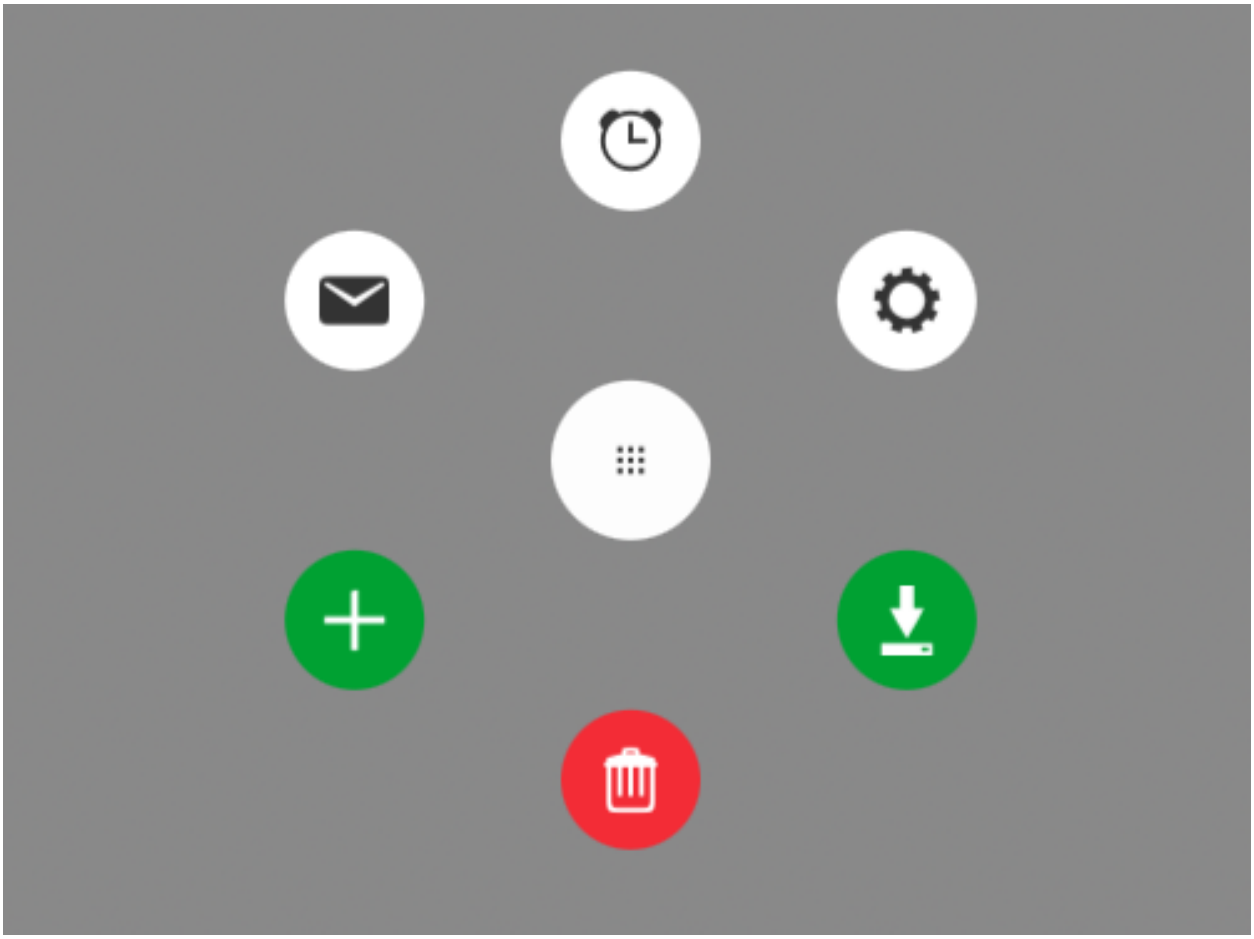
4.7 Radial Bottom Edge

Author	Nekhelesh Ramananthan
License	BSD License
Contact	nik90@ubuntu.com
Framework	ubuntu-sdk-14.10

This component provides a unique way to show actions buttons using the bottom edge. It allows app developers to decide how many actions they want to show and customize it to their liking.

Example:

```
RadialBottomEdge {
    actions: [
        RadialAction {
            iconName: "alarm-clock"
            iconColor: UbuntuColors.coolGrey
            onTriggered: console.log("Alarm..zZz")
        },

        RadialAction {
            iconName: "settings"
            iconColor: UbuntuColors.coolGrey
        },

        RadialAction {
            iconName: "save"
            iconColor: "white"
            enabled: false
            backgroundColor: UbuntuColors.green
            onTriggered: console.log("save")
        },

        RadialAction {
            iconName: "delete"
            iconColor: "white"
            enabled: false
            backgroundColor: UbuntuColors.red
            onTriggered: console.log("delete")
        },

        RadialAction {
            iconName: "add"
            iconColor: "white"
            backgroundColor: UbuntuColors.green
        },

        RadialAction {
            iconName: "stock_email"
            iconColor: UbuntuColors.coolGrey
        }
    ]
}
```



4.7.1 Properties

- *hintSize*: int
- *hintColor*: color
- *hintIconName*: string
- *hintIconSource*: url
- *hintIconColor*: color
- *bottomEdgeEnabled*: boolean
- *actions*: RadialAction <list>
- *actionButtonSize*: int
- *actionButtonDistance*: int
- *expandAngle*: int (defaults to 360 deg)

Note: All properties except for *hintIconSource* have well defined defaults. As a developer, you could choose to go with the defaults or change them to your liking.

4.7.2 Property Documentation

hintSize

The size of the hint shown in the bottom edge. It defaults to 8 grid units.

hintColor

The background color of the hint shown in the bottom edge. By default it uses the ubuntu palette's overlay color.

hintIconName

The name of the icon to display. Valid icon names can be found in the suru-icon-theme.

Note: If both *hintIconName* and *hintIconSource* are specified, then *hintIconName* will be ignored.

hintIconSource

The source url of the icon to display. It has precedence over *hintIconName*.

hintIconColor

The color of the icon displayed in the hint.

bottomEdgeEnabled

Property to enable/disable the bottom edge. When disabled, the bottom edge hint will be hidden.

actions

This property is used to define a list of actions to be shown in the radial menu. The list takes a **RadialAction** which inherits **Action**. A RadialAction adds 3 properties on top of what Action provides which are iconName, iconColor, enabled and backgroundColor.

```
RadialAction {
    iconName: "add"
    iconColor: "white"
    enabled: false
    backgroundColor: "green"
}
```

This helps defining properties for each action separately and allow for customization.

actionButtonSize

The size of all the actions buttons. By default it is set to 7 grid units. Increasing this size would also require increasing the *actionButtonDistance* value as well.

actionButtonDistance

The distance (separation) between the action buttons and the center of the radial menu.

expandAngle

The expand angle defines the spread angle. By default, it is set to 360 degrees which places the buttons in a full circle pattern. If it was set to 180 degrees, then the buttons would follow a semi-circle pattern.

4.8 Welcome Wizard

Author	Nekhelesh Ramananthan, Michael Spencer
License	BSD License
Contact	nik90@ubuntu.com
Framework	ubuntu-sdk-14.10

This component shows users a welcome wizard which can be used to introduce and showcase the features of the app to guide the user. The API of this component is rather simple and provides a lot of freedom to the developer to present his content.

It is recommended to put the Walkthrough{} component inside a qml Component{} since it is not run frequently. Walkthrough derives from a Page. So push it into a pagestack to show the welcome wizard as shown below in the example.

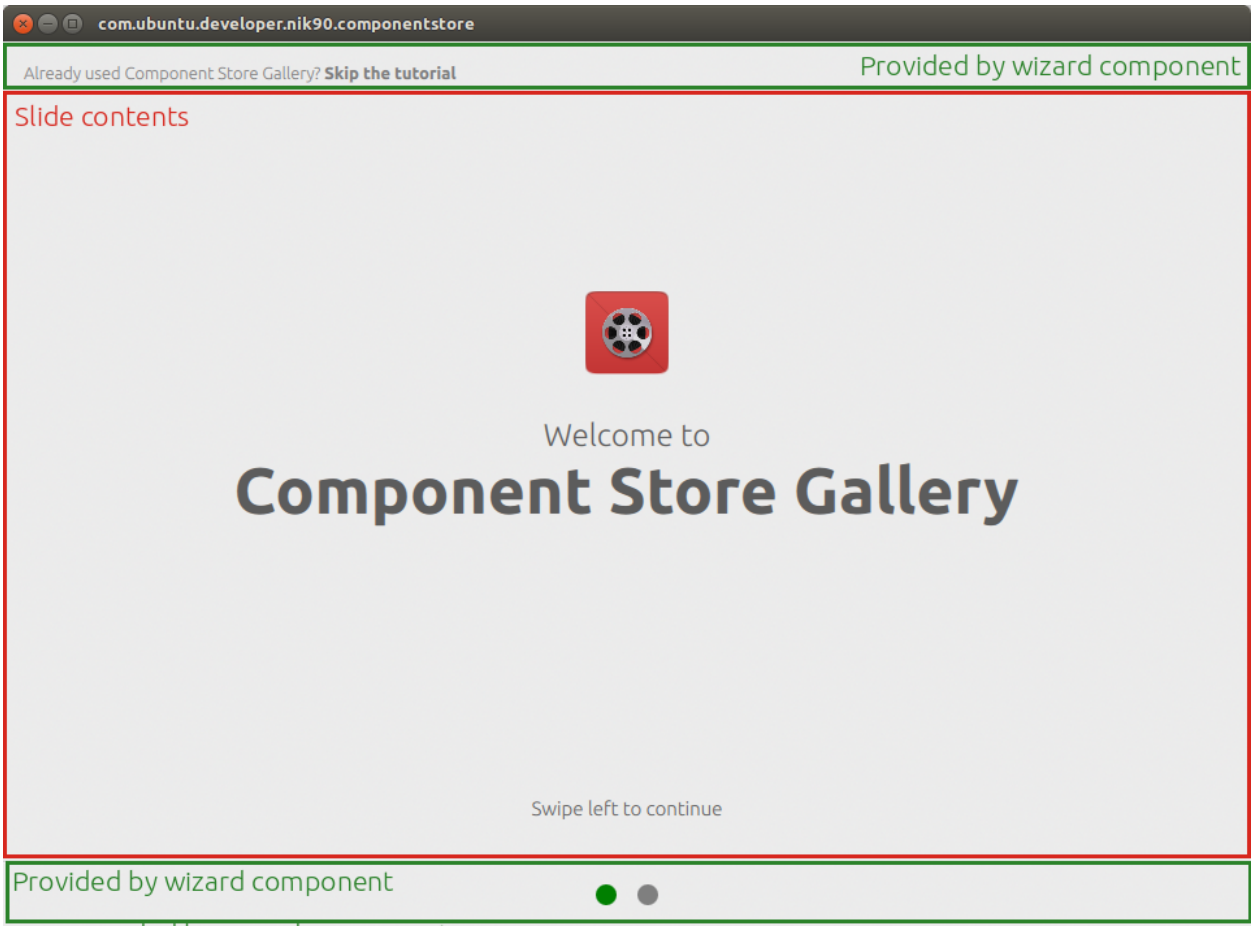
Example:

MainView.qml

```
MainView
{
    Component {
        id: pageComponent
        Walkthrough {
            id: walkthrough

            appName: "Component Store Gallery"

            onFinished: {
                console.log("Welcome Wizard Complete!")
                // Here perhaps save isFirstRun variable to the disk
                stack.pop()
            }

            model: [
                Slide1{},
                Slide2{}
            ]
        }
    }

    component.onCompleted: pagestack.push(pageComponent)
}
```

Slide1.qml

```
import QtQuick 2.3
import Ubuntu.Components 1.1

// Slide 1
Component {
    id: slide1
    Item {
        id: slide1Container

        UbuntuShape {
            anchors {
                bottom: textColumn.top
                bottomMargin: units.gu(4)
                horizontalCenter: parent.horizontalCenter
            }

            image: Image {
                smooth: true
                antialiasing: true
                fillMode: Image.PreserveAspectFit
                source: Qt.resolvedUrl("assets/flashback.png")
            }
        }

        Column {
            id: textColumn

            anchors.centerIn: parent

            Label {
                text: "Welcome to"
                fontSize: "x-large"
                height: contentHeight
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Label {
                text: "Component Store Gallery"
                font.bold: true
                height: contentHeight
                font.pixelSize: units.dp(50)
                anchors.horizontalCenter: parent.horizontalCenter
            }
        }

        Label {
            id: swipeText
            text: "Swipe left to continue"
            horizontalAlignment: Text.AlignHCenter
            anchors.bottom: parent.bottom
            anchors.bottomMargin: units.gu(2)
            anchors.horizontalCenter: parent.horizontalCenter
        }
    }
}
```



4.8.1 Properties

- *appName*: string
- *isFirstRun*: boolean
- *model*: Item<list>
- *completeColor*: color
- *inCompleteColor*: color
- *skipTextColor*: color

4.8.2 Signals

- *finished()*

4.8.3 Property Documentation

appName

Name of the application that is shown in some parts of the welcome wizard.

isFirstRun

Boolean property to determine if the welcome wizard was run for the first time or not. It is recommended to store this variable to the disk using U1dB or Qt.labs.settings to remember the welcome wizard run state.

model

This property stores the welcome wizards slides that are shown to the user. Create your content and store them in separate files per slide. So if you have 3 slides in your welcome wizard, you could define them as Slide1.qml, Slide2.qml and Slide3.qml and then reference them as,

```
model: [
    Slide1{},
    Slide2{},
    Slide3{}
]
```

The slides should only contain the content you want to show. Everything else like the dots, divider, title etc are handled by the welcome wizard component itself. Think of these slides as delegates in a listview which only house the content itself.

completeColor

This property sets the color of the bottom circle to indicate the progress of the user. By default it is green.

inCompleteColor

This property sets the color of the bottom circle to indicate the slides left in the wizard. By default it is grey.

skipTextColor

This property sets the color of the skip text shown at the top of the welcome wizard.

4.8.4 Signal Documentation

finished()

This signal is fired automatically when the user press the skip button. It can also be made to fire manually to exit the welcome wizard. You can perform exit tasks when this signal is fired like updating the *isFirstRun* variable and storing to disk etc.

```
Button {
    id: continueButton
    color: "Green"
    height: units.gu(5)
    width: units.gu(25)
    text: "Exit Welcome Wizard"
    anchors.horizontalCenter: parent.horizontalCenter
    onClicked: finished()
}
```