Technical overview on widget relocation (aka Intelligent Scrolling)

Introduction

This document describes input widget relocation in MeeGo Touch Application Framework (MTF), why is it needed and how is it done. The term "widget relocation" means determining a focused input-accepting widget (text field from here on) and ensuring its visibility to user, moving it if necessary. Moving the input widget is often necessary because of limited space in screen and software input panel (SIP) occupying roughly half of it.

For an introduction into basic text input interactions please refer to [1].

Floating SIP

In MTF, SIP is shown in a window on top of active application window. In contrast to fullscreen SIP, a floating one does not cover the active application fully, and user can still see and interact with it with the SIP being always aligned to bottom of screen. Also, a floating SIP cannot be fully transparent to applications without strict design rules for application GUI. Text fields would have to be always at the very top of application windows, for example. If they are not, then a mechanism is needed to move focused text field always to visible space above the SIP.

Fullscreen SIP does not have this basic visibility problem since it has a text field of its own and merely copies the content over to the underlying application. Application developers don't have to take fullscreen SIP into account in any way.

Ensuring cursor visibility

For users to be able to write text at least the cursor of a text field needs to be visible, preferably the whole text field. While applications usually don't have their text fields hidden, a SIP easily covers them. Let's consider a simple example from MTF which will illustrate the cursor visibility issue.

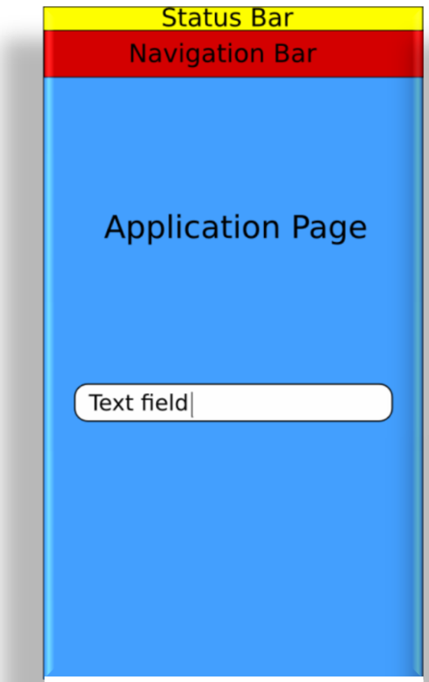

Figure 2 Application page with a text field

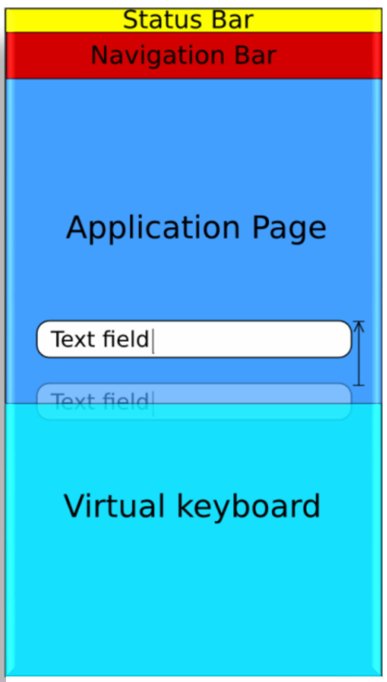

Figure 1 Relocated text field after keyboard is shown

Text field, when not focused, is fully visible to user. After focusing in, it can get either partially or fully obscured by the SIP. In this example, it will get partially hidden, and MTF needs to move text field up by applying a negative vertical offset to text field's global position.

In this example, the offset is applied to the contents of a container widget called MPannableViewport. It is a clipping viewport whose contents can be scrolled, and it sits in every application page in MTF. The range of the viewport is artificially grown to be able to achieve this (more on this in chapter Nested viewports).

Nested viewports

When text field is inside nested viewport it may be required to scroll all of the viewports. In the setup shown in Figure 3, the text field cannot be made visible by scrolling only one of the two viewports. Both are needed: The smaller one (positioned at the bottom in Figure 3) is needed to scroll its contents down while the larger parent viewport is needed to scroll its contents up. This way the text field is not clipped by the smaller viewport and it can be on top of virtual keyboard when it comes up.

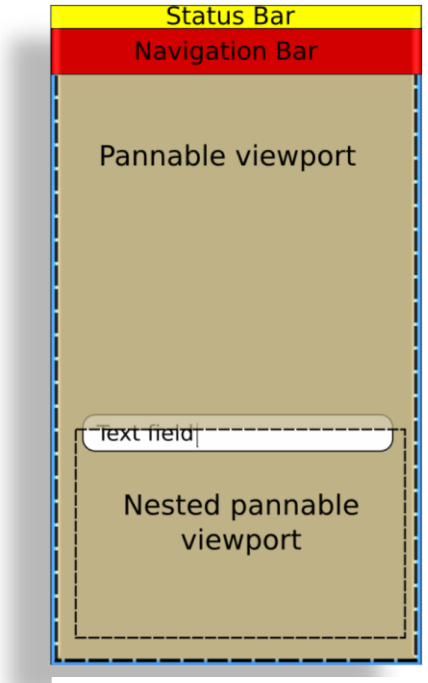

Figure 3 Nested viewports

In MTF, the only container widget that serves as a clipping viewport is MPannableViewport.

The fundamental mechanism which allows widget relocation inside (nested) viewports is to dynamically extend the scrollable range of the root viewport. This guarantees that even the lowest widget in the container's hierarchy can be moved above the area occupied by the appeared SIP. The extra range corresponds to adding empty area at the bottom of the widget clipped by the viewport. User will never see this empty area since it is either outside screen or covered by SIP. No more than one range extension can be active at a time.

Another way of managing text field visibility would be to relayout/resize windows to available space when SIP is shown. This approach was not taken in MTF because it would require applications to implement yet another layout policy and because range extension is just plain simpler.

With conforming viewports, any number of nested viewports can be handled. If clipping by screen area, SIP, or viewports is the only reason a text field is hidden, it can be relocated to a location visible to user.

Choosing distribution of work among viewports plays an important role on what the end-result looks like. A simple bottom-up approach works where total text field offset is first calculated and then work is divided so that innermost viewport, closest to text field, always tries to do the most of it. This guarantees text field's visibility but resulting location might not be optimal. In many cases the text field might get just barely visible without user seeing any surrounding content in the same container. Note that otherwise same but top-to-bottom approach is not guaranteed to work. To obtain optimal results, more refined ways of distributing the work can be made but in the end there is no single correct way of doing it.

Relocator logic in MTF

Relocation logic is in class MInputWidgetRelocator (referred to as "relocator" from here on), and is owned by MSceneManager. The scene manager controls the appearance and positioning of scene windows, which in turn are the basic building blocks for application pages, dialogs, sheets, and others, inside an MTF application. Relocation is therefore only used in when text field is in MSceneManager managed scene. In case of multiple MWidows, each of which has its own scene, there will be multiple relocators, too. What the relocation operation does is roughly the following:

1. Check whether there is focused input field (abort/restore if not)
2. Possibly hide navigation controls to gain more screen space
3. Relocate the input field

Relocation is triggered from various events that occur. Those include:

- New text field is focused
- Text cursor position is moved
- SIP changes shape
- Device orientation changes
- Physical keyboard opened/closed, if applicable
- Scene window animations are finished

To avoid annoying users, excess movement due to scrolling should be minimized. For example, relocation should not take place if cursor is already at center area of visible part of the screen.

Scroll chain helper class

Relocation is done by scrolling all the viewports that contains the text field. Upon focusing into a text field, its parents are analyzed bottom-up and scrollable widgets are added to something called a *scroll chain*. Continuing a previous example setup in Figure 3, Figure 4 below shows the text field's parent widgets (roughly). Those able to do scrolling make up the scroll chain and are marked with green.

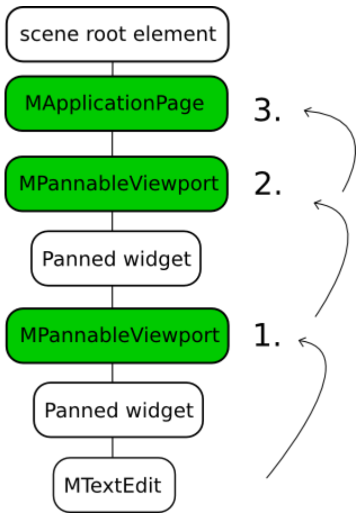

Figure 4 Text field's parent widget chain

The chain is built searching for scrollers bottom-up, from the text field to scene root element. In this case we get an MApplicationPage and two MPannableViewports and that are able to do scrolling. While application page itself is not a viewport-like component it inherits MSceneWindow which is sometimes displaced as a last resort to cover more cases.

In MTF, scroll chain is represented by class MScrollChain, and it is used to pre-calculate required translations, applying and restoring them. When relocation is needed, relocator restores any previous scroll chain and creates a new. Restoring skips widgets that are common to previous and new chain.

There is nothing that bounds scroll chain to input widgets. It can be used to scroll any widget to a new position.

Input widget

Accepted input field for relocation is any QGraphicsItem that has the flag QGraphicsItem::ImAcceptsInputMethod and has keyboard focus. The item also has to provide micro focus rectangle via input method query. This is the cursor rectangle for basic text entries and that is what's used for actual visibility testing and relocation.

Common use-cases and implementation status

MApplicationPage: Text field on an application page. Every application page has MPannableViewport which is used for relocation. The scene window itself is not required to be moved. In certain cases, depending on orientation and other factors, navigation controls are hidden. Logic is missing to hide navigation controls always if there is otherwise not enough space.

MSheet: Text field on a sheet. In content slot of every sheet there is MPannableViewport which is used for relocation. Scene window is not moved since header slot is required to stay in place. Logic is missing for hiding status bar and header slot if there is otherwise not enough space.

MDialog: Uses scene window displacement to move dialog box to visible. Fine-tuning of the dialog box position is required. This is currently low-priority because text edits in dialogs are discouraged.

References

[1] MeeGo Basics, Text Input, http://meego.com/developers/ui-design-guidelines/handset/meego-basics, Apr 6th 2011