ADF Mobile Code Corner

m03. How-to dynamically show-hide mobile UI components

twitter.com/adfcodecorner

Abstract:

A requirement in software development is to conditionally enable/disable or show/hide UI components. Usually, to accomplish this, you dynamically look-up a UI component to change its visibility state. In ADF Mobile v 1.0 however there is no such component look up and the requirement thus needs to be implemented differently.

In this article I explain how to dynamically change the component visibility and rendering state from Java.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
04-MAR-2013

Oracle ADF Mobile Code Corner is a blog-style series of how-to documents targeting at Oracle ADF Mobile that provide solutions to real world coding problems. ADF Mobile Code Corner is an extended offering to ADF Code Corner

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

In ADF Mobile there are two visible states for user interface components, *visible* and *rendered*. If you set a component's visible state to false, then the component still is rendered but hidden on a page If you set the rendered property to false then a component is not added to a page or view at all. Both however have a similar effect in that users can either see or don't see a UI component. Components that are disabled always render on the screen but cannot be used for data input or invoking an action.

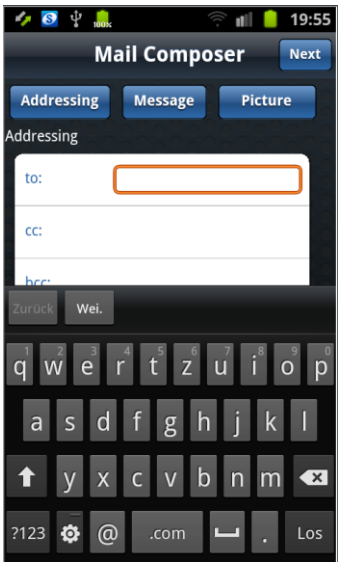
In this article I show you how you can dynamically set a component's visible and disabled state from Java. ADF Mobile v 1.0 does not provide a lookup facility to obtain a component handle to directly manipulate the visible and disabled state so that we need to be a bit creative.

To unveil the solution upfront: The change in the UI (at least for the hide/unhide) functionality is implemented using CSS on the inlineStyle property. The secret sauce however is the use of the `PropertyChangeSupport` support added to the class that exposes the method to switch the UI visibility state.

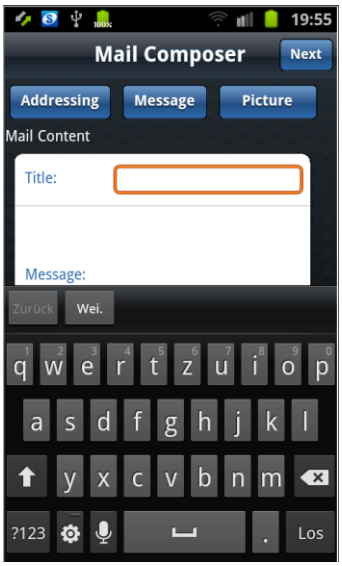
The screen shots for this sample, as well as the source code, are taken from a later ADF Code Corner mobile sample *How-to send Emails with Attachments"*. To download the sample code, refer to this sample when it is out.

Use Case

The images below show a train-style menu for users to navigate between different areas in a long form to compose and send a mail with attachment. Instead of navigating between different views in a task flow, this sample uses show/hide switches to display the information to edit.

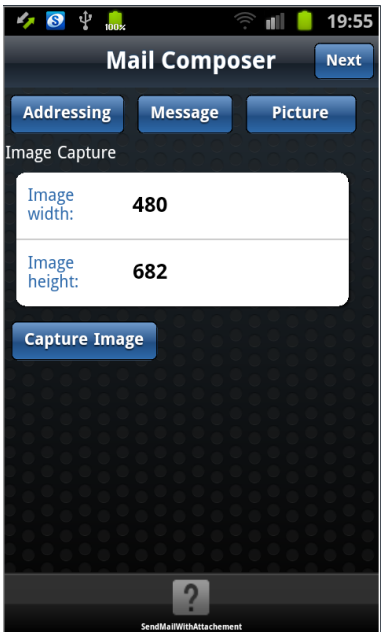


Pressing the *Addressing* command button e.g. hides all input fields that are not related to defining the mail addressing (to, cc and bcc)



Pressing the command button to compose the mail title and message body simply disables all other UI components. Note that if there is a need to switch back to the addressing section, then this can be done without losing data.

Another menu option navigates to the image area for users to take a picture with the on-device camera.



Similar to the solution explained in this article you can use the same approach to render/not render or enable/disable component. To simplify the task of switching content areas, the related input text fields can be grouped in `panelFormLayout` or `panelGroupLayout` components.

Implementation

At runtime, ADF mobile user interfaces developed with AMX (ADF Mobile XML) components render in HTML and JavaScript, which means that what you see on your mobile display in fact is a web view. So if the task is to show and hide components (without removing them from the generated HTML) then CSS will do: *display:none;*

Note that if you want to ensure your mobile displays render fast, you should make sure that a minimum of UI components are rendered in a view. Here, using CSS would be sub-optimal because the component HTML is still rendered – just hidden. In cases where you need to switch between very long forms, using bounded task flow navigation or changing the *rendered* property state is more efficient. For small forms, as indicate in this sample, using CSS provides reasonably good performance – so let's continue with a look at the *amx* markup of the UI.

```
<amx:commandButton id="cb5" text="Picture"
                                  action="#{viewScope.ComposeMailViewBean.showCaptureImage}" >

 <amx:commandButton id="cb4" text="Message"
                                   action="#{viewScope.ComposeMailViewBean.showMailBody}"/>
```

…

The command buttons in the header area of the view invoke a managed bean methods to indicate the area that should be made visible.

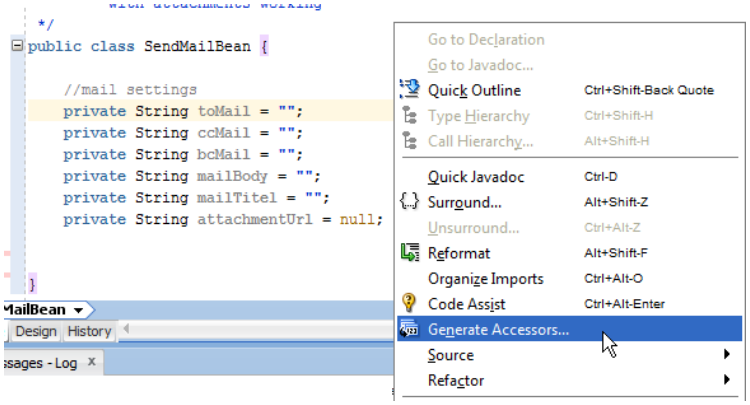
The show/hide areas are defined by `panelFormLayout` tags as shown below.

```
<amx:panelFormLayout id="pfl1"
        inlineStyle="#{viewScope.ComposeMailViewBean.showAddressingHeader}">
        <amx:inputText value=" …"/>
        <amx:inputText value=" …"/>
        <amx:inputText value=" …"/>
</amx:panelFormLayout>
```

As shown, the *inlineStyle* property of the `panelFormLayout` component references the same managed bean that is referenced by the command buttons. This way, in response to a button press action, the UI components in a panel form are shown or hidden. All that is missing now is the information of how to tell the UI components that they need to refresh. This information is part of the managed bean and uses the `PropertyChangeSupport` for the setter/getter pair referenced in the *inlineStyle* property.

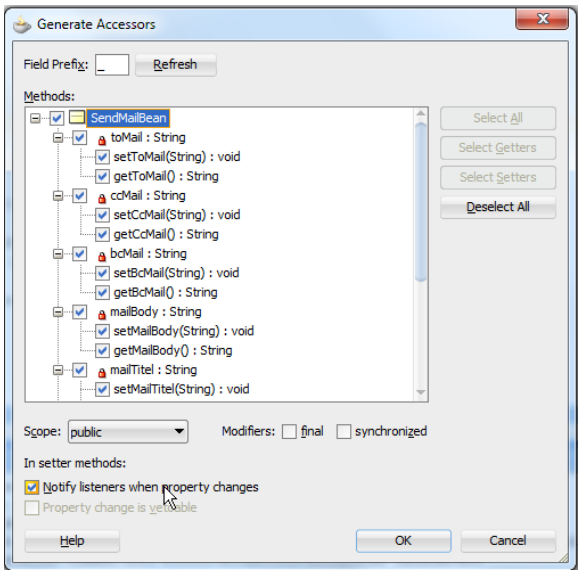
About PropertyChangeSupport

JavaBeans properties become bounded properties when they communicate value changes to interested listeners. The `PropertyChangeSupport` class implements this change notification in a thread safe manner. The event receiving part implements the `PropertyChangeListener` interface, which also is implemented by ADF Mobile.



To add `PropertyChangeSupport` to a bean in Oracle JDeveloper, right mouse click into the bean's source code and choose *Generate Accessors* from the menus as shown in the image above.

Select the properties for which you want to generate setter/getter methods and check the *Notify listeners when property changes* check box (shown in the image below) for JDeveloper to generate all the requires property change notification code. This adds a change notification line onto each setter methods in which listeners are informed about the new and the old value of a property. In addition, methods are added for the mobile framework to register a change listener.



The bean code below contains generated property change support to notify the ADF Mobile framework about changes to the different mail areas for the components to refresh:

```
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;
…

public class ComposeMailViewBean {

  //default settings for the individual show/hide areas
  private String showPhotoButton = "display:none;";
  private String showAddressingHeader = "";
  private String showMailBodyHeader = "display:none;";
  private String showCaptureHeader = "display:none;";

  //The secret sauce that makes it possible to dynamically show/hide
  //UI components in ADF mobile

  private transient PropertyChangeSupport propertyChangeSupport =
                                    new PropertyChangeSupport(this);

  public ComposeMailViewBean() {
      super();
  }


  /**
   * Method to display all fields that are required for editing the
   * mail to, cc and bcc options. Disables all the other edit options
   * @return null. The method is called from a command action and
   *  thus needs to return a String
   */
```

```
public String showAddressing() {
  setShowPhotoButton("display:none;");
  setShowAddressingHeader("");
  setShowMailBodyHeader("display:none;");
  setShowCaptureHeader("display:none;");

 return null;
}

/**
 * Method to display all fields that are required for editing the
 * mail title and message options. Disables all the other edit
 * options
 * @return null. The method is called from a command action and thus
 * needs to return a String
 */

 public String showMailBody() {
   setShowPhotoButton("display:none;");
   setShowAddressingHeader("display:none;");
   //display the mail body and header section
   setShowMailBodyHeader("");
   setShowCaptureHeader("display:none;");
   return null;
 }

 /**
  * Method to display all fields and button that are required for
  * capturing an image using the camera. Disables all the other edit
  * options
  * @return null. The method is called from a command action and
  * thus needs to return a String
  */
 public String showCaptureImage() {
   //show button to capture photo
   setShowPhotoButton("");
   setShowAddressingHeader("display:none;");
   setShowMailBodyHeader("display:none;");
   //show photo width and height settings
   setShowCaptureHeader("");
   return null;
 }
  …

public void setShowAddressingHeader(String showAddressingHeader) {
  String oldShowAddressingHeader = this.showAddressingHeader;
  this.showAddressingHeader = showAddressingHeader;
  propertyChangeSupport.firePropertyChange("showAddressingHeader",
```

```
                                                    oldShowAddressingHeader,
                                                      showAddressingHeader);
    }


    /**
     * Method that returns CSS to display the addressing components.
     * This method is referenced from a panelFormLayout component's
     * inlineStyle property
     * @return
     */
     public String getShowAddressingHeader() {
       return showAddressingHeader;
    }
    …
    public void addPropertyChangeListener(PropertyChangeListener l) {
       propertyChangeSupport.addPropertyChangeListener(l);
    }


   public void removePropertyChangeListener(PropertyChangeListener l) {
     propertyChangeSupport.removePropertyChangeListener(l);
   }
   public void setPropertyChangeSupport(PropertyChangeSupport
                                        propertyChangeSupport) {
  PropertyChangeSupport oldPropertyChangeSupport =
                                        this.propertyChangeSupport;
  this.propertyChangeSupport = propertyChangeSupport;

  propertyChangeSupport.firePropertyChange("propertyChangeSupport",
                                          oldPropertyChangeSupport,
  propertyChangeSupport);
 }
  public PropertyChangeSupport getPropertyChangeSupport() {
    return propertyChangeSupport;
  }
}
```

Conclusion

In this article I explain how UI components in ADF Mobile can be partially refreshed using a JavaBean property and the `PropertyChangeSupport` class to programmatically and thus dynamically set the inlineStyle property, dispabled or rendered property. The complete source code and sample used in this article become available in a later article *"How-to send Emails with Attachments"*.

RELATED DOCOMENTATION

⊠	
⊠	
⊠	