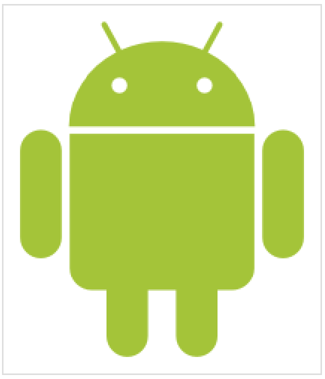
← Previous Next →

How Do I Validate An Android Form?

Posted on July 12, 2013 by **Excella**

Tweet 1 Like 2



I have heard lots of questions about how to develop for Android OS lately, so I've made a point to explore more about **native Android programming.** Specifically, effective and efficient (and cool looking!) **methods to perform form field validation.**

I started by working on a small mobile application that would require individuals to enter contact information. Through this exercise I was able to find some really great resources from all over the web that helped move the process forward. I have consolidated what I learned here to help future Android developers down the path of validation.

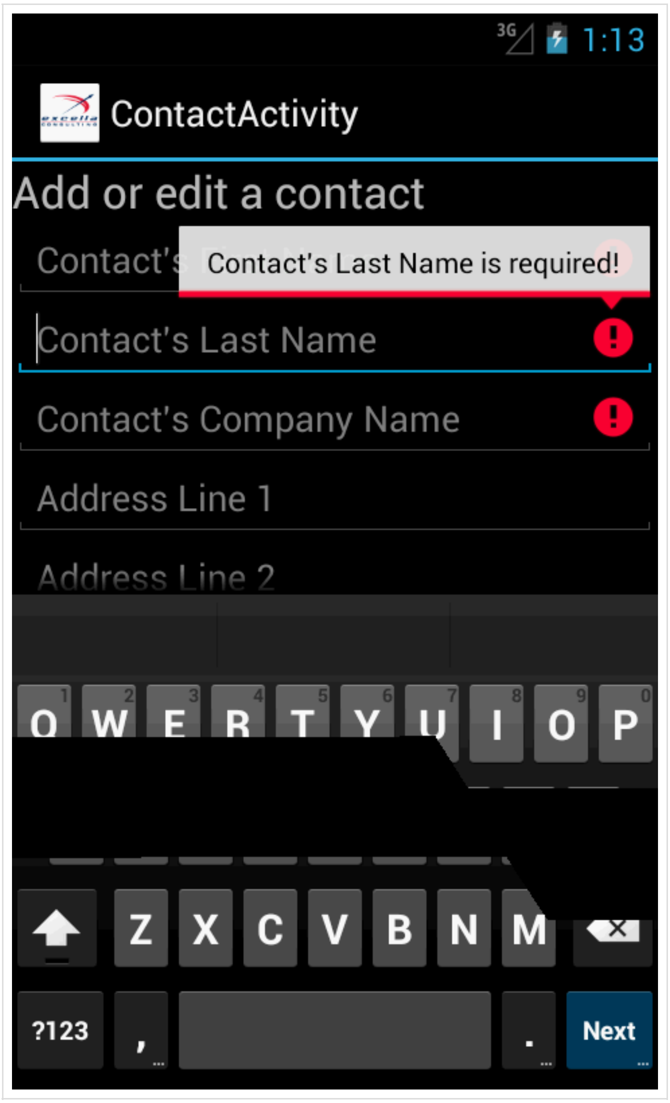
Read on for what I did, and the tips and tricks I learned along the way!

Getting Started

The first question I had to answer was "when" did I want to validate fields? I came up with four different scenarios:

1) Before the end user even enters data by "setting them up for success"
2) When the end user changes text in a text field
3) When a text field loses focus
4) When a form is submitted

The next question I asked myself was how did I want the validation messages to look? After looking at various options like **Toasts**, I decided that **the built-in "setError" method on a TextView object was the coolest and most effective way to display validation errors:**

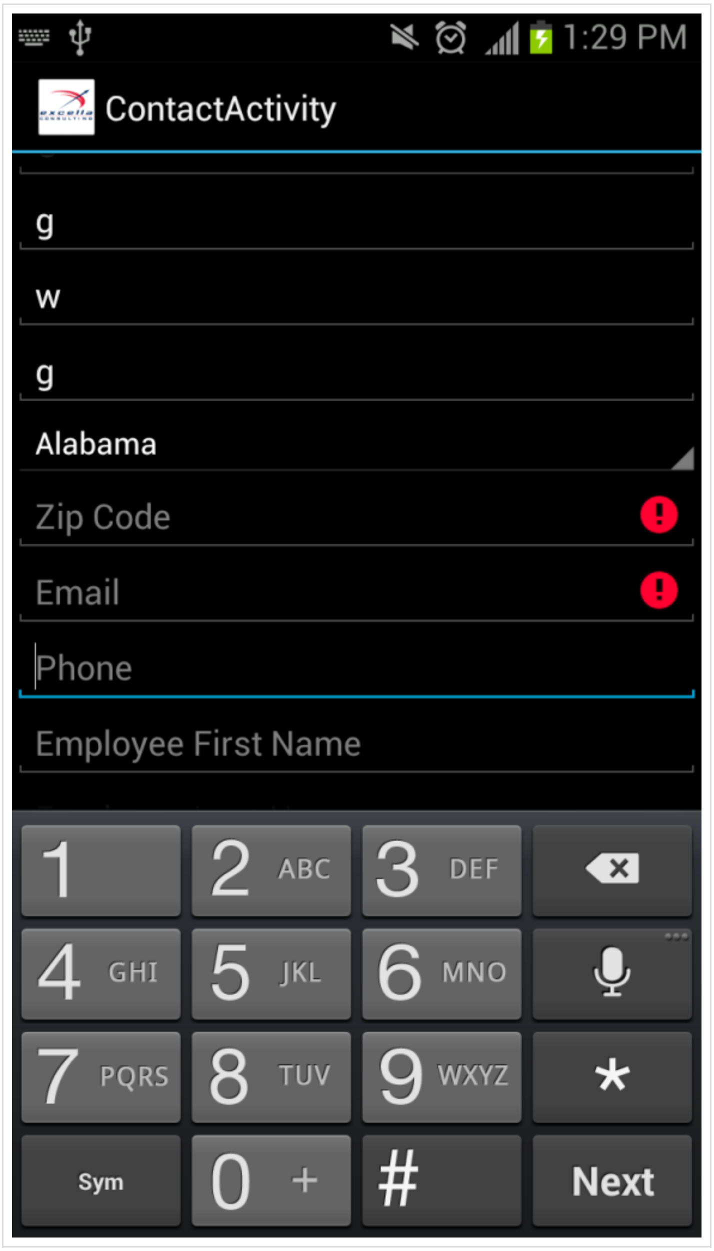


I like the display cue of the red icon (which can be customized) and the hover text which displays when each field has focus.

So now that I knew what I wanted to do, the next important question was: how to do it? So let me answer that based on my list of “whens” from above.

Scenario 1: Before the User Enters Any Text

Android provides a long list of “input types” that you can set in the layout XML to “set the user up for success” so that they can only enter permissible characters or numbers. For example: an input type of “phone” leads to only numbers, and characters such as -, (,) being available for the user to enter:



The XML to achieve this is shown below:

```
<EditText
      android:id="@+id/EditTextContactPhone"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="@string/contact_phone"
      android:inputType="phone" >
</EditText>
```

The challenge is finding the proper list of input type values to use in the layout XML and what they make the Android device do. **This link** will get you started.

Scenario 2: When the End User Changes Text in a Text Field

Next up: what you can do when the user changes the text in a field and still manages to enter something incorrect, even though we set them up for success above. Well, if it can be done unobtrusively, the next best time to validate is when the user is actually entering the text (this is also where we can implement the cool icon and error message shown in the first screen).

To do this you have to implement a **TextChangedListener** which must implement the **TextWatcher** interface. The annoyance here is that you have to do this on a field by field basis so **you need a way to “homogenize” the implementation or you will have tons of repetitive code.** I did this by creating an abstract class called *TextValidator* which implements the TextWatcher interface (credit goes to **Christopher Perry on Stack Overflow** for this idea).

The key method in this class is **afterTextChanged.** My flavor of it is shown below (I don’t pass the text separately to the validate method).

```
final public void afterTextChanged(Editable s) {
      validate(textView);
}
```

Then I created a concrete class that extends the TextValidator class and implements the validate method:

```
public class ContactValidator extends TextValidator implements OnFocusChangeListener
{
      public ContactValidator(TextView textView) {
            super(textView);
      }

      @Override
      public boolean validate(TextView textView) {

            if (StringUtils.isBlank(textView.getText().toString())){
                        textView.setError( textView.getHint()+" is required!" );
                        return false;
            } else {
                  textView.setError(null);
                  return true;
              }
      }

      public void onFocusChange(View textView, boolean hasFocus) {

            if (!hasFocus){
                  validate((TextView)textView);
            }
      }
}
```

I highlighted in red the two key elements of this class, that it extends TextValidator, and how to get the cool “icon” and error message. Using TextView.setError gives us the desired visual validation.

Now, the last step is to set this validator on to the text fields of interest. Since I wanted it set on all the text fields in my Activity, I created a “factory” class to set up the validation on all the text fields:

```
public class ContactValidatorFactory<T extends Activity> {

      private ViewGroup viewGroup;

      private View view;

      private List<View> possibleTextViews = new ArrayList<View>();

      public void setUpValidators(T t) {

            viewGroup = ((ViewGroup) t.findViewById(android.R.id.content));
            view = viewGroup.getChildAt(0);
            possibleTextViews = view.getFocusables(0);


            for (View possibleTextView : possibleTextViews) {

                  if (possibleTextView instanceof EditText) {
                        ((EditText) possibleTextView).addTextChangedListener(new
ContactValidator((EditText) possibleTextView));
                        ((EditText) possibleTextView).setOnFocusChangeListener(new
ContactValidator(((EditText) possibleTextView)));

                  }
            }
      }
}
```

The 3 lines of code highlighted in blue are used to retrieve a list of all the elements on the screen. The code then loops through all of them looking for text boxes (EditText). For each EditText it finds, the line highlighted in red adds the on text change listener to the text box. The listener is used to trigger the validation logic whenever the user changes text in the field.

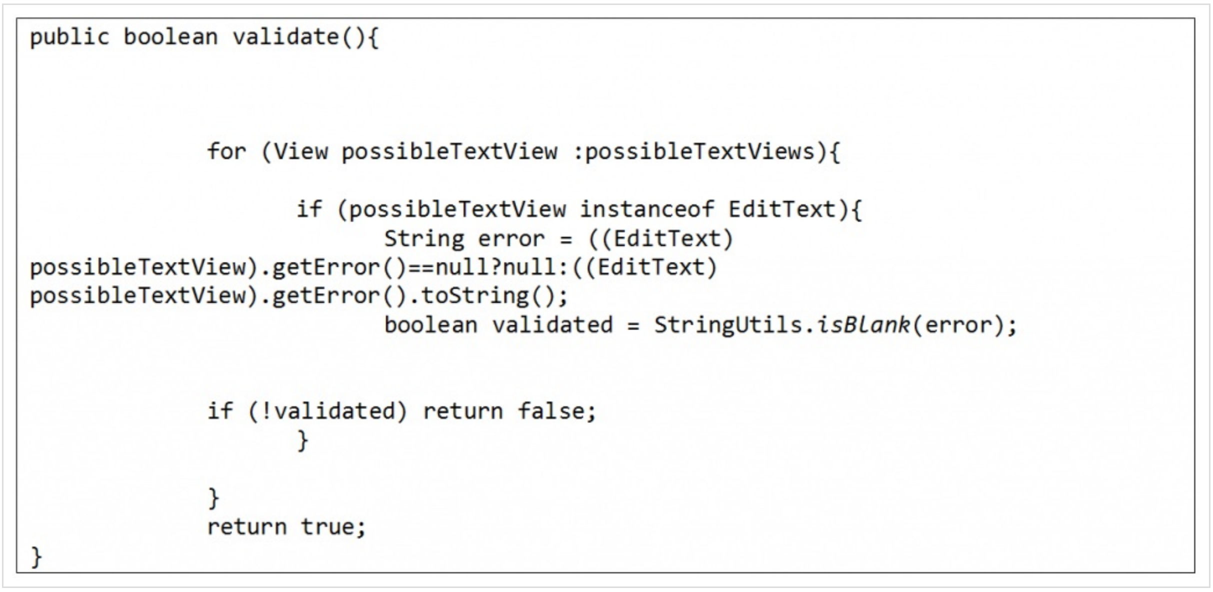
Scenario 3: When a Text Field Loses Focus

The setup for validation when a text field loses focus is very similar to the one described above. There are just two small additions:

- The line in green above adds the on focus change listener which works the same way as the text change listener.
- The ContactValidator class has an OnFocusChange method which calls the validate method and it implements the OnFocusChangeListener interface.

Scenario 4: When a Form is Submitted

The final step in the validation is to make sure that the form is only submitted for further processing if there are no validation errors. Since the Activity keeps track of the state of all of its component elements, this becomes an easy task. You just need to check if an error message is set for any of the elements or not. If it is, then the form is not valid:



```
public boolean validate(){


            for (View possibleTextView :possibleTextViews){

                  if (possibleTextView instanceof EditText){
                        String error = ((EditText)
possibleTextView).getError()==null?null:((EditText)
possibleTextView).getError().toString();
                        boolean validated = StringUtils.isBlank(error);


            if (!validated) return false;
                  }

            }
            return true;
}
```

I created this method in the ContactValidatorFactory class, and then called it from the submit method tied to the submit button on the form.

So what do you think? How would you improve this validation? How would you handle validation of other elements on a form?

Author: *Janice G., Senior Consultant, Excellian since 2012*

SHARE THIS:

Twitter 1 | Facebook 2 | LinkedIn | Email | Print | More

This entry was posted in **Software Development** and tagged **Mobile** by **Excella**. Bookmark the **permalink**.

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

Post Comment

☐ Check to leave a reply

☐ Notify me of followup comments via e-mail

☐ Notify me of new posts by email.

S E A R C H E X C E L L A ' S B L O G

Search this Blog

S U B S C R I B E B Y E M A I L

Enter your email address to subscribe to this blog and receive notifications of new posts by email.
Join 37 other subscribers

Email Address

Subscribe

R E C E N T P O S T S

- **Where does Agile Fit in Federal IT?**
- **What is Agile?**
- **Best IT Events For June**
- **Is Microsoft right for your business? Insights from Build 2014**
- **How to Modernize Your .NET Software Dev Environment**
- **Best IT Events for May**
- **Do Business Analysts Need Another Certification?**

RSS - Posts

A R C H I V E S

- **June 2014** (3)
- **May 2014** (3)
- **April 2014** (4)
- **March 2014** (4)
- **February 2014** (5)
- **January 2014** (2)
- **December 2013** (3)
- **November 2013** (3)
- **October 2013** (5)
- **September 2013** (3)
- **August 2013** (4)
- **July 2013** (4)
- **June 2013** (4)
- **May 2013** (5)
- **April 2013** (3)
- **March 2013** (5)
- **February 2013** (8)
- **January 2013** (9)
- **December 2012** (6)
- **November 2012** (7)

C A T E G O R I E S

- **Business Intelligence** (10)
- **Events & Training in DC, MD, VA** (11)
- **Excella** (1)
- **Program Management** (31)
- **Software Development** (41)
- **Training** (1)

T A G S

.NET Agile Agile BI Best Practices Business Analysis CBAP Certifications Change

Management data governance Data Management Data visualization Developer Tools Federal Government information culture IT events Java Microsoft Mobile Open Source PMP Project Management Python Reporting & Visualization Requirements Scrum UI/UX