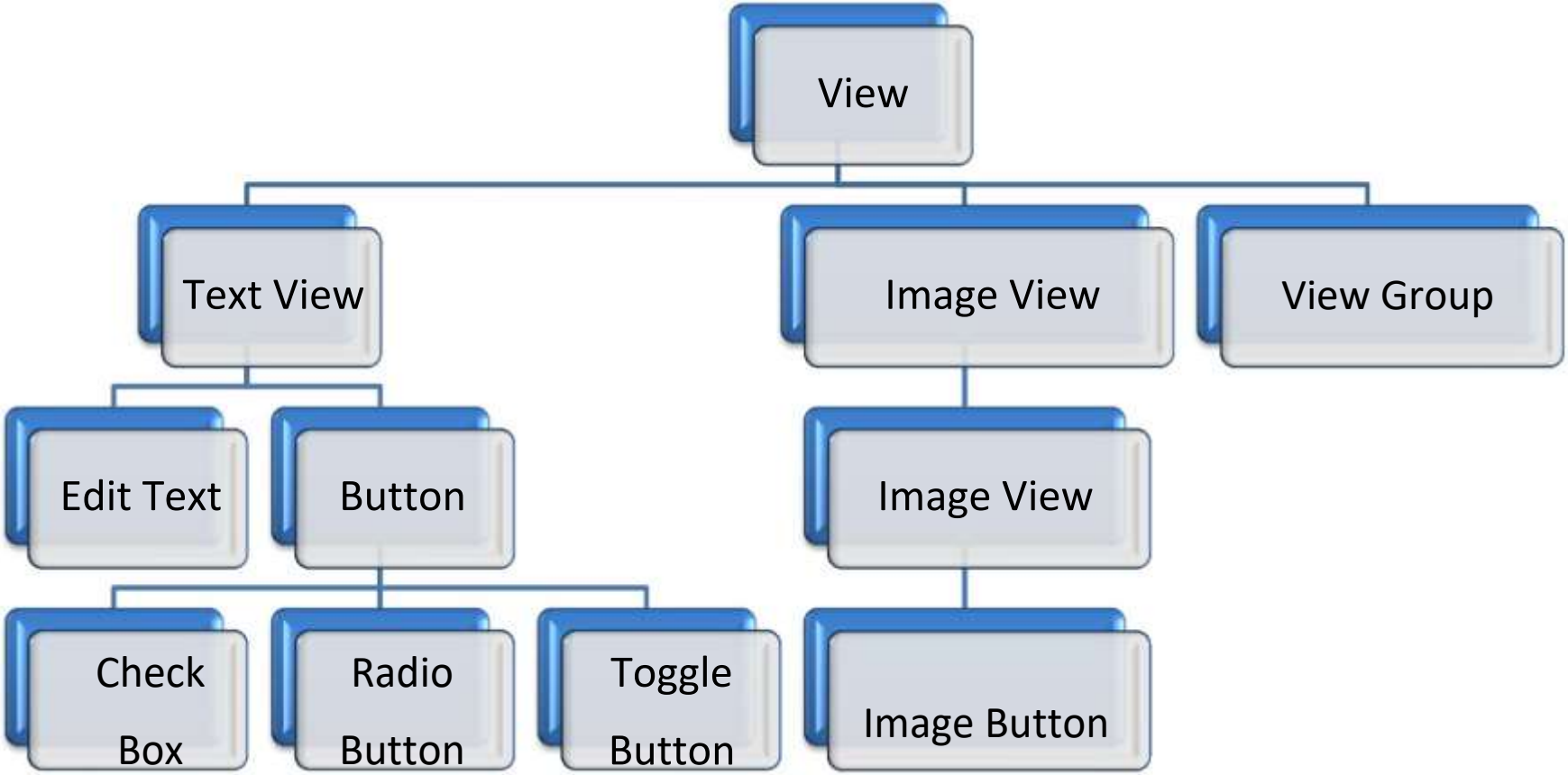
Input Controls in Android

Ref: http://developer.android.com/

View Hierarchy



Ref: http://developer.android.com/

Text View

- A text view allows the user to type text into your app.
- It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard.
- In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.
- You can add a text view to you layout with the EditText object.
- You should usually do so in your XML layout with a <EditText> element

Ref: http://developer.android.com/

Specifying the Keyboard Type

- Text fields can have different input types, such as number, date, password, or email address.
- The type determines what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.
- You can specify the type of keyboard you want for your EditText object with the android:inputType attribute.
  - For example, if you want the user to input an email address, you should use the textEmailAddress input type:

```
<EditText android:id="@+id/email_address"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="@string/email_hint"
   android:inputType="textEmailAddress" />
```

Ref: http://developer.android.com/

Specifying the Keyboard Type

- There are several different input types available for different situations.
- Here are some of the more common values for android:inputType:
  - "**text**" Normal text keyboard.
  - "**textEmailAddres**s" Normal text keyboard with the @ character.
  - "**textUri**" Normal text keyboard with the / character.
  - "**numbe**r" Basic number keypad.
  - "**phone**" Phone-style keypad.

Ref: http://developer.android.com/

Controlling other behaviors

- The android:inputType also allows you to specify certain keyboard behaviors, such as whether to capitalize all new words or use features like auto-complete and spelling suggestions.
- The android:inputType attribute allows bitwise combinations so you can specify both a keyboard layout and one or more behaviors at once.
- For example, here's how you can collect a postal address, capitalize each word, and disable text suggestions:

```
<EditText
  android:id="@+id/postal_address"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:hint="@string/postal_address_hint"
  android:inputType="textPostalAddress|
                     textCapWords|
                     textNoSuggestions" />
```

Ref: http://developer.android.com/

Controlling other behaviors

Here are some of the common input type values that define keyboard behaviors:

- "**textCapSentences**" Normal text keyboard that capitalizes the first letter for each new sentence.
- "**textCapWords**" Normal text keyboard that capitalizes every word. Good for titles or person names.
- "**textAutoCorrect**" Normal text keyboard that corrects commonly misspelled words.
- "**textPassword**" Normal text keyboard, but the characters entered turn into dots.
- "**textMultiLine**" Normal text keyboard that allow users to input long strings of text that include line breaks (carriage returns).

Ref: http://developer.android.com/

Buttons

- A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.

Ref: http://developer.android.com/

Buttons: Text, Image, Both

- you can create the button in your layout in three ways:
- With text, using the Button class:

```
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/button_text"
   ... />
```

- With an icon, using the ImageButton class:

```
 <ImageButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/button_icon"
   ... />
```

- With text and an icon, using the Button class with the android:drawableLeft attribute:

```
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
   ... />
```

Ref: http://developer.android.com/

Handling Click Event

- To define the click event handler for a button, add the android:onClick attribute to the <Button> element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event.
- The Activity hosting the layout must then implement the corresponding method.
- For example, here's a layout with a button using android:onClick:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/button_send"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/button_send"
  android:onClick="sendMessage" />
```

- Within the Activity that hosts this layout, the following method handles the click event:

```
public void sendMessage(View view) {
   // Do something in response to button click
}
```

Ref: http://developer.android.com/

Using an OnClickListener

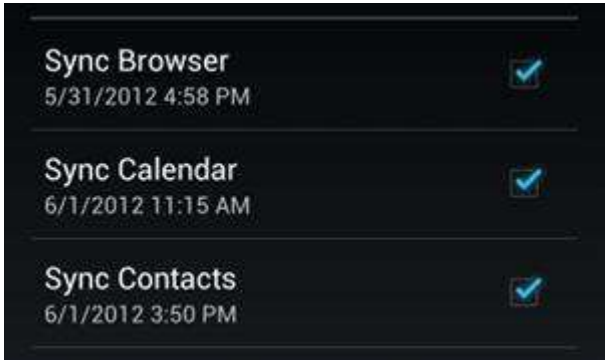
- You can also declare the click event handler programmatically rather than in an XML layout.
- This might be necessary if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.
- To declare the event handler programmatically, create an View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener).
- For example:

```
Button button = (Button) findViewById(R.id.button_send);
    button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            // Do something in response to button click
        }
    });
```

Ref: http://developer.android.com/

Check Boxes

- Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.



- To create each checkbox option, create a CheckBox in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

Ref: http://developer.android.com/

Responding to Click Events

- To define the click event handler for a checkbox, add the android:onClick attribute to the <CheckBox> element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event.
- The Activity hosting the layout must then implement the corresponding method.

Ref: http://developer.android.com/

XML file defining Check Box

- <?xml version="1.0" encoding="utf-8"?> <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"> <CheckBox
    android:id="@+id/checkbox_meat"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/meat"
      android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cheese"
      android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Ref: http://developer.android.com/

Event Handling for Check Box

Within the Activity that hosts this layout, the following method handles the click event for both checkboxes:

```
public void onCheckboxClicked(View view)
    { // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

Ref: http://developer.android.com/

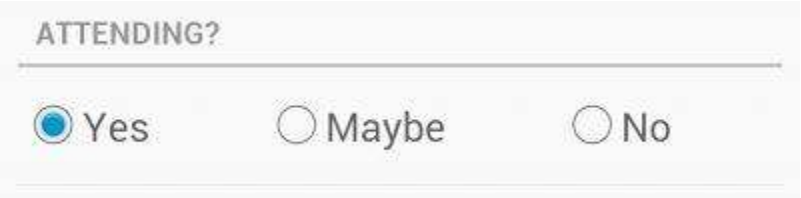
Check Box state Change

- The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must:
  - Be public
  - Return void
  - Define a View as its only parameter (this will be the View that was clicked)
- If you need to change the checkbox state yourself use the setChecked(boolean) or
- toggle() method.
  - Change the checked state of the view to the inverse of its current state

Ref: http://developer.android.com/

Radio Buttons

- Radio buttons allow the user to select one option from a set.
- You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side.
  - If it's not necessary to show all options side-by-side, use a spinner instead.



  - To create each radio button option, create a RadioButton in your layout.
  - However, because radio buttons are mutually exclusive, you must group them together inside a RadioGroup.
    - By grouping them together, the system ensures that only one radio button can be selected at a time.

Ref: http://developer.android.com/

Radio Button Event Handling

- When the user selects one of the radio buttons, the corresponding RadioButton object receives an on-click event.
- To define the click event handler for a button, add the android:onClick attribute to the <RadioButton> element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event.
- The Activity hosting the layout must then implement the corresponding method.

Ref: http://developer.android.com/

Radio Button Example

- <?xml version="1.0" encoding="utf-8"?> <RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"> <RadioButton
    android:id="@+id/radio_pirates"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/pirates"
      android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/ninjas"
      android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

Ref: http://developer.android.com/

Event Handling

```
• public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was
    clicked switch( view.getId() ) {
      case R.id.radio_pirates:
        if (checked)
          // Pirates are the
        best break;
      case R.id.radio_ninjas:
        if (checked)
          // Ninjas
        rule break;
    }
  }
```

Ref: http://developer.android.com/

Toggle Buttons

- A toggle button allows the user to change a setting between two states.
- You can add a basic toggle button to your layout with the ToggleButton object.
- Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a Switch object.

- The ToggleButton and Switch controls are subclasses of CompoundButton and function in the same manner, so you can implement their behavior the same way.

Ref: http://developer.android.com/

Event Handling: Xml and Java File

```
<ToggleButton
  android:id="@+id/togglebutton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textOn="Vibrate on"
  android:textOff="Vibrate off"
  android:onClick="onToggleClicked"/>
```

- ```
  public void onToggleClicked(View view) {
     // Is the toggle on?
     boolean on = ((ToggleButton)
     view).isChecked(); if (on) {
        // Enable
     vibrate } else {
        // Disable vibrate
     }
  }
  ```

Ref: http://developer.android.com/
  ```

Using an OnCheckedChangeListener

- To declare the event handler programmatically, create an CompoundButton.OnCheckedChangeListener object and assign it to the button by calling setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener). For example:

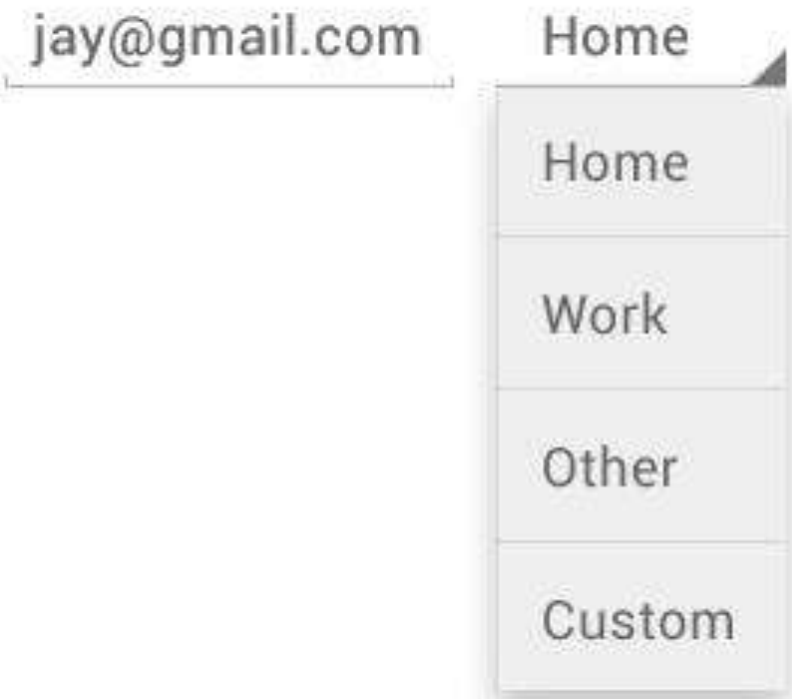
```
ToggleButton toggle = (ToggleButton)
findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

Ref: http://developer.android.com/

Spinners

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value.

Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.



Ref: http://developer.android.com/

XML Design

- You can add a spinner to your layout with the Spinner object. You should usually do so in your XML layout with a <Spinner> element. For example:

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

- To populate the spinner with a list of choices, you then need to specify a SpinnerAdapter in your Activity or Fragment source code.

Ref: http://developer.android.com/

Populate the Spinner with User Choices

- The choices you provide for the spinner can come from any source, but must be provided through an **SpinnerAdapter**
  - **ArrayAdapter** if the choices are available in an array
  - **Cursor Adapter** if the choices are available from a database query.

Ref: http://developer.android.com/

Populate the Spinner with User Choices

- If the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```
<?xml version="1.0" encoding="utf-
    8"?> <resources>
        <string-array name="planets_array">
            <item>Mercury</item>
            <item>Venus</item>
            <item>Earth</item>
            <item>Mars</item>
            <item>Jupiter</item>
            <item>Saturn</item>
            <item>Uranus</item>
            <item>Neptune</item>
        </string-array>
    </resources>
```

Ref: http://developer.android.com/

Populate the Spinner with User Choices

Java Code for handling spinner

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

- The **createFromResource()** method allows you to create an **ArrayAdapter** from the string array. The third argument for this method is a layout resource that defines how the selected choice appears in the spinner control. The **simple_spinner_item** layout is provided by the platform and is the default layout you should use unless you'd like to define your own layout for the spinner's appearance.
- You should then call **setDropDownViewResource(int)** to specify the layout the adapter should use to display the list of spinner choices **(simple_spinner_dropdown_item** is another standard layout defined by the platform).
- Call **setAdapter()** to apply the adapter to your **Spinner**.

Ref: http://developer.android.com/

Event handling

- Then you need to specify the interface implementation by calling setOnItemSelectedListener():

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

- Handling selection of item

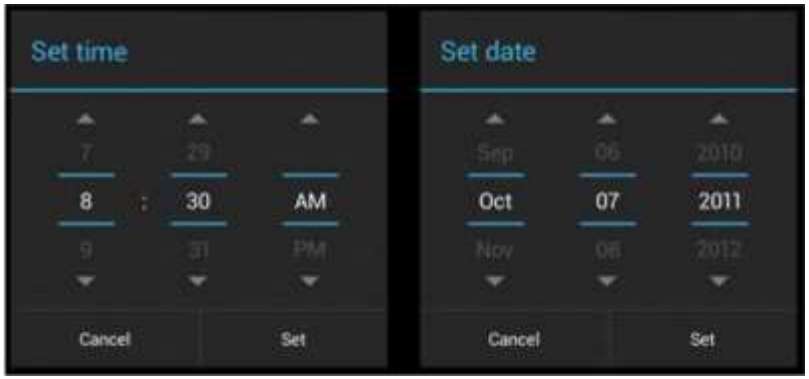
```
public class SpinnerActivity extends Activity implements OnItemSelectedListener
    {
        ...

        public void onItemSelected(AdapterView<?> parent, View
              view, int pos, long id) {
           // An item was selected. You can retrieve the selected item using
           // parent.getItemAtPosition(pos)
        }

        public void onNothingSelected(AdapterView<?> parent)
           { // Another interface callback
        }
    }
```

Ref: http://developer.android.com/

Pickers



- Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs.
- Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year).
- Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.
- It is recommend that you use DialogFragment to host each time or date picker.
- The DialogFragment manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.

Ref: http://developer.android.com/

Creating a Time Picker

- To display a TimePickerDialog using DialogFragment, you need to define a fragment class that extendsDialogFragment and return a TimePickerDialog from the fragment's onCreateDialog() method.
- To define a DialogFragment for a TimePickerDialog, you must:
  - Define the onCreateDialog() method to return an instance of TimePickerDialog
  - Implement the TimePickerDialog.OnTimeSetListener interface to receive a callback when the user sets the time.

Ref: http://developer.android.com/

```
public static class TimePickerFragment extends DialogFragment
                                implements
TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
     // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

    // Create a new instance of TimePickerDialog and return it
    return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int
minute) {
        // Do something with the time chosen by the user
    }
}
```

Ref: http://developer.android.com/

Time Picker...

- Once you've defined a DialogFragment like the one shown above, you can display the time picker by creating an instance of the DialogFragment and calling show().
- For example, here's a button that, when clicked, calls a method to show the dialog:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />
```

Ref: http://developer.android.com/

Time Picker...

- When the user clicks this button, the system calls the following method:

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(),"timePicker");
}
```

- This method calls show() on a new instance of the DialogFragment defined above. The show() method requires an instance of FragmentManager and a unique tag name for the fragment.

Ref: http://developer.android.com/

Date Picker

- Creating a DatePickerDialog is just like creating a TimePickerDialog. The only difference is the dialog you create for the fragment.
- To display a DatePickerDialog using DialogFragment, you need to define a fragment class that extendsDialogFragment and return a DatePickerDialog from the fragment's onCreateDialog() method.
- To define a DialogFragment for a DatePickerDialog, you must:
  - Define the onCreateDialog() method to return an instance of DatePickerDialog
  - Implement the DatePickerDialog.OnDateSetListener interface to receive a callback when the user sets the date.

Ref: http://developer.android.com/

Date Picker..

```
public static class DatePickerFragment extends
                            DialogFragment implements
DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the
        picker final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR); int
        month = c.get(Calendar.MONTH); int day
        = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
     return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day)
{
        // Do something with the date chosen by the user
    }
}
```

Ref: http://developer.android.com/

Date Picker...

- Once you've defined a DialogFragment like the one shown above, you can display the date picker by creating an instance of the DialogFragment and calling show().
- For example, here's a button that, when clicked, calls a method to show the dialog:

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/pick_date"
  android:onClick="showDatePickerDialog" />
```

Ref: http://developer.android.com/

Date Picker...

- When the user clicks this button, the system calls the following method:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

- This method calls show() on a new instance of the DialogFragment defined above. The show() method requires an instance of FragmentManager and a unique tag name for the fragment.

Ref: http://developer.android.com/