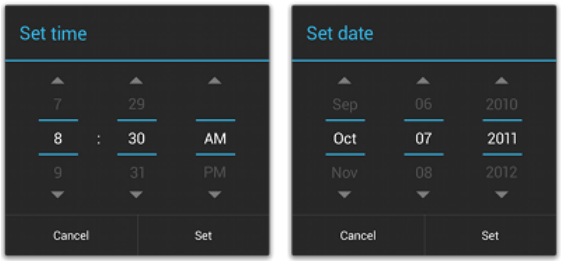
Pickers

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.



We recommend that you use `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) to host each time or date picker. The `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) 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.

Although `DialogFragment` (https://developer.android.com/reference/android/app/DialogFragment.html) was first added to the platform in Android 3.0 (API level 11), if your app supports versions of Android older than 3.0—even as low as Android 1.6—you can use the `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) class that's available in the support library (https://developer.android.com/tools/support-library/index.html) for backward compatibility.

Note: The code samples below show how to create dialogs for a time picker and date picker using the support library (https://developer.android.com/tools/support-library/index.html) APIs for `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html). If your app's `minSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#min) is 11 or higher, you can instead use the platform version of `DialogFragment` (https://developer.android.com/reference/android/app/DialogFragment.html).

Key classes are the following:

- `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html)
- `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html)

Also see the Fragments overview (https://developer.android.com/guide/components/fragments.html).

Creating a Time Picker

To display a `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html) using `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html), you need to define a fragment class that extends `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) and return a `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html) from the fragment's `onCreateDialog()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#onCreateDialog(android.os.Bundle)) method.

Note: If your app supports versions of Android older than 3.0, be sure you've set up your Android project with the support library as described in Setting Up a Project to Use a Library (https://developer.android.com/tools/support-library/setup.html).

Extending DialogFragment for a time picker

To define a `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) for a `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html), you must:

- Define the `onCreateDialog()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#onCreateDialog(android.os.Bundle)) method to return an instance of `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html)
- Implement the `TimePickerDialog.OnTimeSetListener` (https://developer.android.com/reference/android/app/TimePickerDialog.OnTimeSetListener.html) interface to receive a callback when the user sets the time.

Here's an example:

KOTLIN JAVA

```
class TimePickerFragment : DialogFragment(), TimePickerDialog.OnTimeSetListener {

    override fun onCreateDialog(savedInstanceState: Bundle): Dialog {
        // Use the current time as the default values for the picker
        val c = Calendar.getInstance()
        val hour = c.get(Calendar.HOUR_OF_DAY)
        val minute = c.get(Calendar.MINUTE)

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

    override fun onTimeSet(view: TimePicker, hourOfDay: Int, minute: Int) {
        // Do something with the time chosen by the user
    }
}
```

See the `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html) class for information about the constructor arguments.

Now all you need is an event that adds an instance of this fragment to your activity.

Showing the time picker

Once you've defined a `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) like the one shown above, you can display the time picker by creating an instance of the `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) and calling `show()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#show(android.support.v4.app.FragmentManager, java.lang.String)).

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" />
```

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

KOTLIN JAVA

```
fun showTimePickerDialog(v: View) {
    TimePickerFragment().show(supportFragmentManager, "timePicker")
}
```

This method calls `show()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#show(android.support.v4.app.FragmentManager, java.lang.String)) on a new instance of the `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) defined above. The `show()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#show(android.support.v4.app.FragmentManager, java.lang.String)) method requires an instance of `FragmentManager` (https://developer.android.com/reference/android/support/v4/app/FragmentManager.html) and a unique tag name for the fragment.

Caution: If your app supports versions of Android lower than 3.0, be sure that you call `getSupportFragmentManager()` (https://developer.android.com/reference/android/support/v4/app/FragmentActivity.html#getSupportFragmentManager()) to acquire an instance of `FragmentManager` (https://developer.android.com/reference/android/support/v4/app/FragmentManager.html). Also make sure that your activity that displays the time picker extends `FragmentActivity` (https://developer.android.com/reference/android/support/v4/app/FragmentActivity.html) instead of the standard `Activity` (https://developer.android.com/reference/android/app/Activity.html) class.

Creating a Date Picker

Creating a `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html) is just like creating a `TimePickerDialog` (https://developer.android.com/reference/android/app/TimePickerDialog.html). The only difference is the dialog you create for the fragment.

To display a `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html) using `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html), you need to define a fragment class that extends `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) and return a `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html) from the fragment's `onCreateDialog()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#onCreateDialog(android.os.Bundle)) method.

Extending DialogFragment for a date picker

To define a `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) for a `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html), you must:

- Define the `onCreateDialog()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#onCreateDialog(android.os.Bundle)) method to return an instance of `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html)
- Implement the `DatePickerDialog.OnDateSetListener` (https://developer.android.com/reference/android/app/DatePickerDialog.OnDateSetListener.html) interface to receive a callback when the user sets the date.

Here's an example:

KOTLIN JAVA

```
class DatePickerFragment : DialogFragment(), DatePickerDialog.OnDateSetListener {

    override fun onCreateDialog(savedInstanceState: Bundle): Dialog {
        // Use the current date as the default date in the picker
        val c = Calendar.getInstance()
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

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

    override fun onDateSet(view: DatePicker, year: Int, month: Int, day: Int) {
        // Do something with the date chosen by the user
    }
}
```

See the `DatePickerDialog` (https://developer.android.com/reference/android/app/DatePickerDialog.html) class for information about the constructor arguments.

Now all you need is an event that adds an instance of this fragment to your activity.

Showing the date picker

Once you've defined a `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) like the one shown above, you can display the date picker by creating an instance of the `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) and calling `show()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#show(android.support.v4.app.FragmentManager, java.lang.String)).

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" />
```

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

KOTLIN JAVA

```
fun showDatePickerDialog(v: View) {
    val newFragment = DatePickerFragment()
    newFragment.show(supportFragmentManager, "datePicker")
}
```

This method calls `show()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#show(android.support.v4.app.FragmentManager, java.lang.String)) on a new instance of the `DialogFragment` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html) defined above. The `show()` (https://developer.android.com/reference/android/support/v4/app/DialogFragment.html#show(android.support.v4.app.FragmentManager, java.lang.String)) method requires an instance of `FragmentManager` (https://developer.android.com/reference/android/support/v4/app/FragmentManager.html) and a unique tag name for the fragment.

Using pickers with autofill

Android 8.0 introduces the Autofill Framework (https://developer.android.com/guide/topics/text/autofill.html), which allows users to save data that can be later used to fill out forms in different apps. Pickers can be useful in autofill scenarios by providing a UI that lets users change the value of a field that stores date or time data. For example, in a credit card form, a date picker would allow users to enter or change the expiration date of their credit card.

Because pickers are dialogs, they aren't displayed in an activity along with other fields. To display the picker data when the picker isn't visible, you can use another view, such as an `EditText` (https://developer.android.com/reference/android/widget/EditText.html), which can display the value when the picker isn't visible.

An `EditText` (https://developer.android.com/reference/android/widget/EditText.html) object natively expects autofill data of type `AUTOFILL_TYPE_TEXT` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_TEXT). In contrast, autofill services should save the data as `AUTOFILL_TYPE_DATE` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_DATE) to be able to create an appropriate representation of it. To solve the inconsistency in types, it's recommended that you create a custom view that inherits from `EditText` (https://developer.android.com/reference/android/widget/EditText.html) and implements the methods required to correctly handle values of type `AUTOFILL_TYPE_DATE` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_DATE).

You should take the following steps to create a subclass of `EditText` (https://developer.android.com/reference/android/widget/EditText.html) that can handle values of type `AUTOFILL_TYPE_DATE` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_DATE):

1. Create a class that inherits from `EditText` (https://developer.android.com/reference/android/widget/EditText.html).
2. Implement the `getAutofillType()` (https://developer.android.com/reference/android/view/View.html#getAutofillType()) method, which should return `AUTOFILL_TYPE_DATE` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_DATE).
3. Implement the `getAutofillValue()` (https://developer.android.com/reference/android/view/View.html#getAutofillValue()) method, which should return an `AutofillValue` (https://developer.android.com/reference/android/view/autofill/AutofillValue.html) object that represents the date in milliseconds. To create the return object, use the `forDate()` (https://developer.android.com/reference/android/view/autofill/AutofillValue.html#forDate(long)) method to generate an `AutofillValue` (https://developer.android.com/reference/android/view/autofill/AutofillValue.html) object.
4. Implement the `autofill()` (https://developer.android.com/reference/android/view/View.html#autofill(android.view.autofill.AutofillValue)) method. This method provides the logic to handle the `AutofillValue` (https://developer.android.com/reference/android/view/autofill/AutofillValue.html) parameter, which is of type `AUTOFILL_TYPE_DATE` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_DATE). To handle the parameter, create a proper string representation of it, such as `mm/yyyy`. Use the string representation to set the `text` property of your view.
5. Implement functionality that displays a picker when the user wants to edit the date in the custom subclass of `EditText` (https://developer.android.com/reference/android/widget/EditText.html). The view should update the `text` property with a string representation of the value that the user selected on the picker.

For an example of a subclass of `EditText` (https://developer.android.com/reference/android/widget/EditText.html) that handles `AUTOFILL_TYPE_DATE` (https://developer.android.com/reference/android/view/View.html#AUTOFILL_TYPE_DATE) values, see the `CreditCardExpirationDatePickerView` (https://github.com/googlesamples/android-AutofillFramework/blob/master/Application/src/main/java/com/example/android/autofillframework/app/CreditCardExpirationDatePickerView.java) class in the Android Autofill Framework sample (https://github.com/googlesamples/android-AutofillFramework).

To learn more about proving autofill support for your custom views, see Support for custom views (https://developer.android.com/guide/topics/text/autofill.html#support_for_custom_views).

Dernière mise à jour : septembre 18, 2018