Day 14: Controls: DateTime Controls

Today we will take a look at different DateTime Controls in Xamarin.Android. We will explore the following controls –

1. CalendarView
2. DatePicker
3. TimePicker

This article will also serve as a quick introduction to Fragments in Android, although, we will explore Fragments in more detail in the upcoming blog posts.

# CalendarView

CalendarView control in Android is used to showing and selecting dates. Let’s take a look at a simple example of a CalendarView control in action –

To get started, drag and drop a CalendarView Control on to the Main.axml (or any other activity) file, in the snippet below, I also included a button to get the selected date from CalendarView –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:p1="http://schemas.android.com/apk/res/android"  p1:orientation="vertical"  p1:minWidth="25px"  p1:minHeight="25px"  p1:layout\_width="match\_parent"  p1:layout\_height="match\_parent"  p1:id="@+id/linearLayout1">  <CalendarView  p1:layout\_width="match\_parent"  p1:layout\_height="500dp"  p1:id="@+id/calendarView1" />  <Button  p1:text="Get Selected Date"  p1:layout\_width="match\_parent"  p1:layout\_height="wrap\_content"  p1:id="@+id/getSelectedDateButton" />  </LinearLayout> |

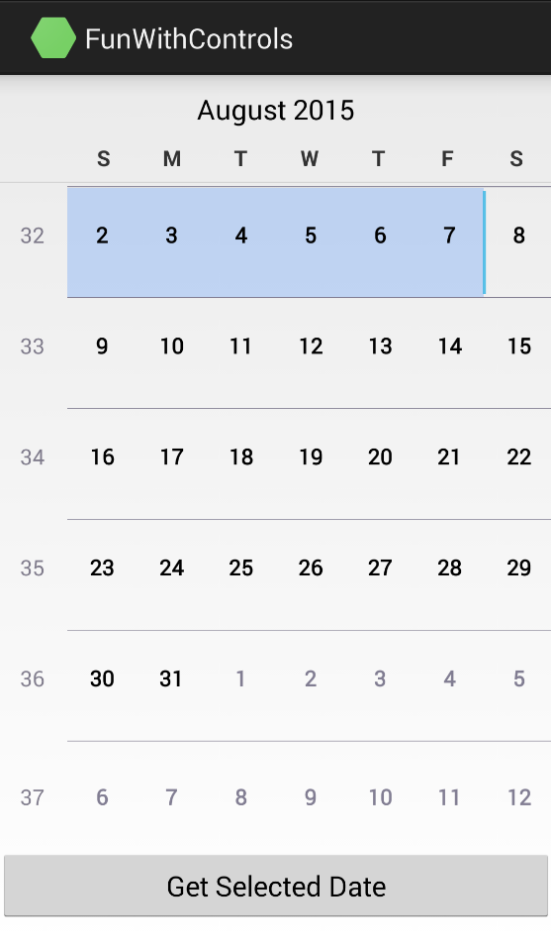
Gist file link: <https://gist.github.com/vkoppaka/4782551da86bac6c1783>

And as always, to find the calendarView in the Activity file, we use the FindViewById method –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", MainLauncher = true, Icon = "@drawable/icon")]  public class MainActivity : Activity  {  private CalendarView calendarView;  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.Main);  calendarView = FindViewById<CalendarView>(Resource.Id.calendarView1);  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/e58b813850671e25bcf5>

If I were to run the application, we should now see a Calendar widget that lets us see and select dates.

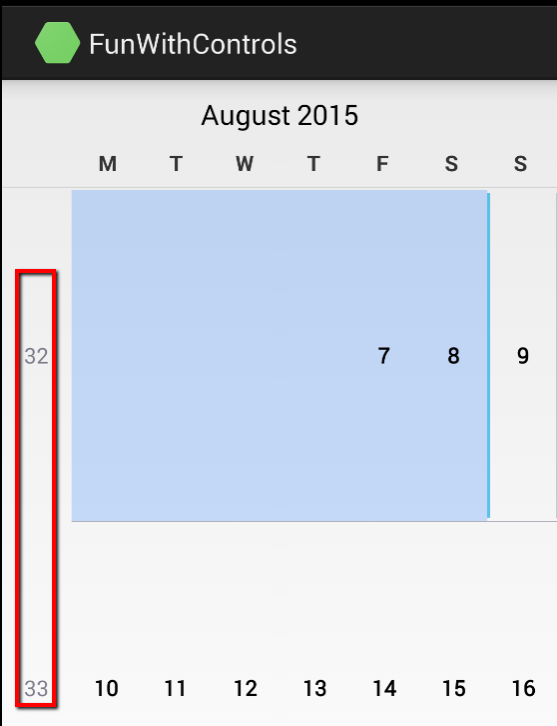


And in order to get the selected date and see what other options that CalendarView exposes let’s take a look at different properties and Events CalendarView exposes.

## Properties

The most common properties that CalendarView exposes are –

* **android:maxDate:** Sets the maximum date shown by the CalendarView widget, data entry format is “mm/dd/yyyy”
* **android:minDate:** Sets the minimum date shown by the CalendarView widget, data entry format is “mm/dd/yyyy”
* **android:firstDayOfWeek:** Sets an integer notation of the first day of the week that CalendarView should be showing.
* **android:showWeekNumber:** Shows week number in the CalendarView left column.
* **android:shownWeekCount:** Limits the view of Calendar to an integer based week count.
* **Bunch of color properties like:** focusedMonthDateColor, selectedWeekBackgroundColor, unfocusedMonthDateColor, weekNumberColor, weekSeperatorLineColor all these properties are pretty self-explanatory and are used to set the various color attributes of CalendarView



## Events

The most common event that CalendarView exposes is –

**DateChange:** Event that gets raised by the CalendarView widget when a Date selection is changed, the event exposes the new day of the month, month and year to get what the new date that is selected is

## Getting selected date:

To get selected date of a CalendarView widget we would be calling the .Date property. The .Date property gets the number of milli-seconds that have passed since 1/1/1970 to the selected date. So we just need to add those milli-seconds to get the actual date, like –

|  |
| --- |
| private void GetSelectedDateButton\_Click(object sender, EventArgs e)  {  var milliSeconds = calendarView.Date;  var oldDateTime = new DateTime(1970, 1, 1);  var newDate = oldDateTime.AddMilliseconds(milliSeconds);  Toast.MakeText(this, newDate.ToShortDateString(), ToastLength.Long);  } |

Gist file link: <https://gist.github.com/vkoppaka/9764cc89a17980ffe3a5>

# DatePicker

DatePicker View is pretty similar to CalendarView but instead of showing an entire calendar for selecting the date which takes up a lot of screen real estate, the DatePicker shows a simple and modern picker for Selecting the Date.

To get started, drag and drop a simple TextView (to show the selected date) and Button (to bring up the DatePicker). Here is how the AXML looks like –

|  |
| --- |
| <?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"  android:minWidth="25px"  android:minHeight="25px">  <TextView  android:id="@+id/dateDisplay"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="" />  <Button  android:id="@+id/pickDate"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Pick date" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/ef0473d497d671043c3f>

On the activity side, we will first get references for the TextView and Button controls. And when the Pick Date Button is clicked, we will be showing a DatePicker Dialog. This Dialog is a “Fragment” (more on what a fragment is in coming blog posts), which is a C# class that inherits from the “**DialogFragment**” class. And inside, this DatePickerFragment we will be overriding the “**OnCreateDialog**” and we will be returning a DatePickerDialog with the date initialized. Let’s see the DatePickerFragment in action –

|  |
| --- |
| using System;  using Android.App;  using Android.Content;  using Android.OS;  namespace FunWithControls  {  public class DatePickerFragment : DialogFragment  {  private readonly Context context;  private DateTime date;  private readonly DatePickerDialog.IOnDateSetListener listener;  public DatePickerFragment(Context context, DateTime date, DatePickerDialog.IOnDateSetListener listener )  {  this.context = context;  this.date = date;  this.listener = listener;  }  public override Dialog OnCreateDialog(Bundle savedState)  {  var dialog = new DatePickerDialog(context, listener, date.Year, date.Month - 1, date.Day);  return dialog;  }  }  } |

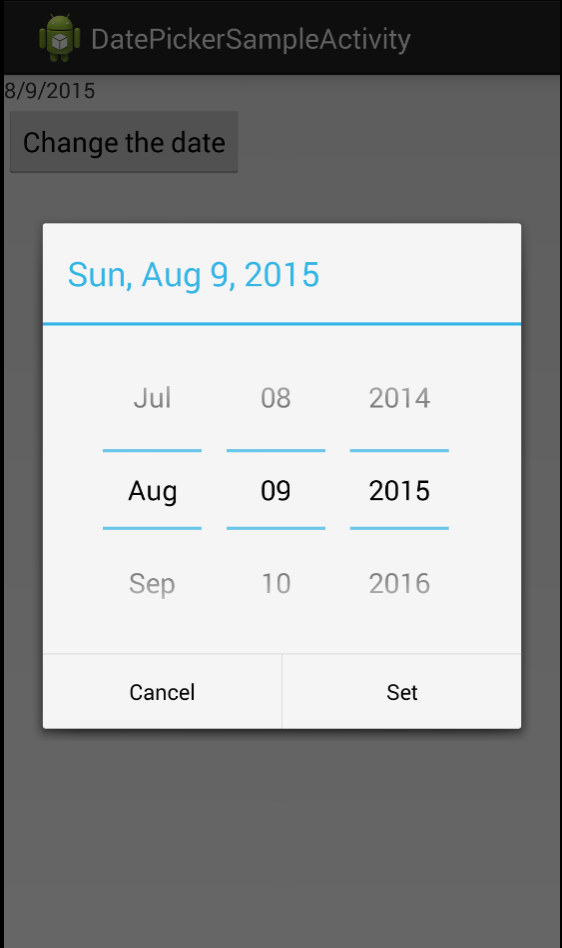
Gist file link: <https://gist.github.com/vkoppaka/8e20368cb4bc17d2b4ed>

And the Activity that calls this Fragment should also inherit from “**IOnDateSetListener”.** This interface exposes a **OnDateSet** method as a listener callback to set any visual elements (or do any additional processing) after a date is set from the Fragment. Let’s see the Activity in action –

|  |
| --- |
| using System;  using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "DatePickerSampleActivity", MainLauncher = true)]  public class DatePickerSampleActivity : Activity, DatePickerDialog.IOnDateSetListener  {  private TextView dateDisplay;  private Button pickDate;  private DateTime date;  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Create your application here  SetContentView(Resource.Layout.DatePickerSample);  dateDisplay = FindViewById<TextView> (Resource.Id.dateDisplay);  pickDate = FindViewById<Button> (Resource.Id.pickDate);  // add a click event handler to the button  pickDate.Click += delegate { ShowDatePickerDialog (); };  // get the current date  date = DateTime.Today;  // display the current date (this method is below)  UpdateDisplay (date);  }  void ShowDatePickerDialog()  {  var dialog = new DatePickerFragment(this, DateTime.Now, this);  dialog.Show(FragmentManager, null);  }    public void OnDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth)  {  var newDate = new DateTime(year, monthOfYear + 1, dayOfMonth);  UpdateDisplay(newDate);  }    void UpdateDisplay(DateTime selectedDate)  {  dateDisplay.Text = selectedDate.ToString("d");  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/451fbb9d922c031adbc5>

And when you run the application, you will first see a TextView and Pick Date button, and when you click the Pick Date button, you will then see our DatePicker control.



And if you change any date and click “Set” button on the DatePicker, you will see the underlying TextView update because of the OnDateSet method sets the current date selected from DatePicker.

# TimePicker

TimePicker View is pretty similar to DatePicker but instead of showing an interface for selecting the date, the TimePicker shows a simple and modern picker for selecting, well, Time.

To get started, drag and drop a simple TextView (to show the selected Time) and Button (to bring up the TimePicker). Here is how the AXML looks like –

|  |
| --- |
| <?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"  android:minWidth="25px"  android:minHeight="25px">  <TextView  android:id="@+id/timeDisplay"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="" />  <Button  android:id="@+id/pickTime"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Pick time" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/3346c5549d3f97b1175b>

On the activity side, we will first get references for the TextView and Button controls. And when the Pick Time Button is clicked, we will be showing a TimePicker Dialog. This Dialog is a “Fragment” (just like the one from above), which is a C# class that inherits from the “**DialogFragment**” class. And inside, this TimePickerFragment we will be overriding the “**OnCreateDialog**” and we will be returning a TimePickerDialog with the date initialized. Let’s see the TimePickerFragment in action –

|  |
| --- |
| using Android.App;  using Android.Content;  using Android.OS;  namespace FunWithControls  {  public class TimePickerFragment : DialogFragment  {  private readonly Context context;  private int hour;  private int minute;  private readonly TimePickerDialog.IOnTimeSetListener listener;  public TimePickerFragment(Context context, int hour, int minute, TimePickerDialog.IOnTimeSetListener listener )  {  this.context = context;  this.hour = hour;  this.minute = minute;  this.listener = listener;  }  public override Dialog OnCreateDialog(Bundle savedState)  {  var dialog = new TimePickerDialog(context, listener, hour, minute, false);  return dialog;  }  }  } |

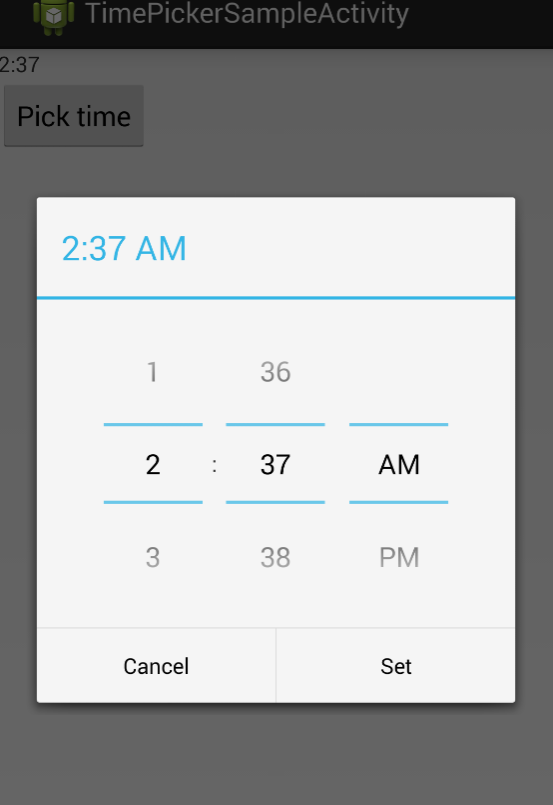
Gist file link: <https://gist.github.com/vkoppaka/9f550ad78f7e5d5b4bde>

And the Activity that calls this Fragment should also inherit from “**IOnTimeSetListener”.** This interface exposes an **OnTimeSet** method as a listener callback to set any visual elements (or do any additional processing) after a time is set from the Fragment. Let’s see the Activity in action –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "TimePickerSampleActivity", MainLauncher = true)]  public class TimePickerSampleActivity : Activity, TimePickerDialog.IOnTimeSetListener  {  private TextView timeDisplay;  private Button pickTime;  private int hours;  private int minutes;  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Create your application here  SetContentView(Resource.Layout.TimePickerSample);  timeDisplay = FindViewById<TextView> (Resource.Id.timeDisplay);  pickTime = FindViewById<Button> (Resource.Id.pickTime);  // add a click event handler to the button  pickTime.Click += delegate { ShowTimePickerDialog (); };  // get the current date  hours = 2;  minutes = 37;  // display the current date (this method is below)  UpdateDisplay (hours, minutes);  }  void ShowTimePickerDialog()  {  var dialog = new TimePickerFragment(this, hours, minutes, this);  dialog.Show(FragmentManager, null);  }    public void OnTimeSet(TimePicker view, int hourOfDay, int minute)  {  UpdateDisplay(hourOfDay, minute);  }  void UpdateDisplay(int selectedHours, int selectedMinutes)  {  timeDisplay.Text = selectedHours + ":" + selectedMinutes;  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/2509d24da6051893d52e>

And when you run the application, you will first see a TextView and Pick Time button, and when you click the Pick Time button, you will then see our TimePicker control.



That’s it for today, tomorrow, we will explore Media Controls in Xamarin.Android