Activity Lifecycle

Activity Lifecycle Concepts and Overview

Xamarin Inc.

Related Articles:

- Android.App.Activity Class
- Android.Content.Intent Class
- Android.Content.Context Class

Related SDK:

- Android Activity Class
- Android Intent Class
- Android Context Class

Activities are a fundamental building block of Android applications and they can exist in a number of different states. The activity lifecycle begins with instantiation and ends with destruction, and includes many states in between. When an activity changes state, the appropriate lifecycle event method is called, notifying the activity of the impending state change and allowing it to execute code in order to adapt to that change. This article examines the lifecycle of activities and explains the responsibility that an activity has during each of these state changes in order to be part of a well-behaved, reliable application.

Overview

Activities are an unusual programming concept specific to Android. In traditional application development there is usually a static main method, which is executed to launch the application. With Android, however, things are different; Android applications can be launched via any registered activity within an application. In practice, most applications will only have a specific activity that is specified as the application entry point. However, if an application crashes, or is terminated by the OS, the OS can try to restart the application at the last open activity or anywhere else within the previous activity stack. Additionally, the OS may pause activities when they're not active, and reclaim them if it is low on memory. Careful consideration must be made to allow the application to correctly restore its state in the event that an activity is restarted and in case that activity depends on data from previous activities.

The activity lifecycle is a collection of methods the OS calls throughout the lifecycle of an activity. These methods allow developers to implement the functionality that is necessary to satisfy the state and resource management requirements of their applications.

It is extremely important for the application developer to analyze the requirements of each activity to determine which methods exposed by the activity lifecycle need to be implemented. Failure to do this can result in application instability, crashes, resource bloat, and possibly even underlying OS instability.

This article examines the activity lifecycle in detail, including:

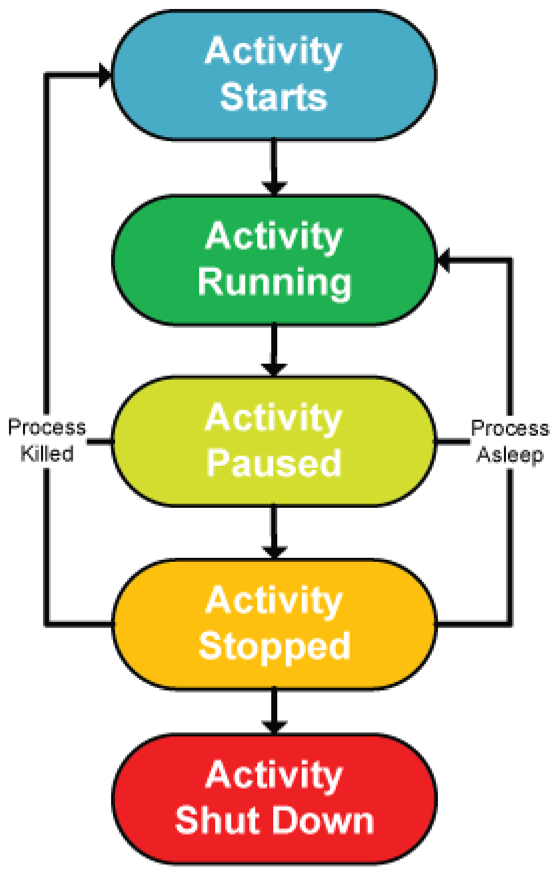
- Activity States
- Lifecycle Methods
- Lifecycle Best Practices

Activity Lifecycle

The Android activity lifecycle comprises a collection of methods exposed within the Activity class that provide the developer with a resource management framework. This framework allows implementers to meet the unique state management requirements of each activity within an application. The activity lifecycle thus assists the developer by providing a consistent framework in which to handle resource management within the application.

Activity States

The Android OS uses a priority queue to assist in managing activities running on the device. Based on the state a particular Android activity is in, it will be assigned a certain priority within the OS. This priority system helps Android identify activities that are no longer in use, allowing the OS to reclaim memory and resources. The following diagram illustrates the states an activity can go through, during its lifetime:



These states can be broken into 3 main groups as follows:

- **Active or Running** - Activities are considered active or running if they are in the foreground, also known as the top of the activity stack. This is considered the highest priority activity in the Android Activity stack, and as such will only be killed by the OS in extreme situations, such as if the activity tries to use more memory than is available on the device as this could cause the UI to become unresponsive.
- **Paused** - When the device goes to sleep, or an activity is still visible but partially hidden by a new, non-full-sized or transparent activity, the activity is considered paused. Paused activities are still alive, that is, they maintain all state and member information, and remain attached to the window manager. This is considered to be the second highest priority activity in the Android Activity stack and, as such, will only be killed by the OS if killing this activity will satisfy the resource requirements needed to keep the Active/Running Activity stable and responsive.
- **Stopped** - Activities that are completely obscured by another activity are considered stopped or in the background. Stopped activities still try to retain their state and member information for as long as possible, but stopped activities are considered to be the lowest priority of the three states and, as such, the OS will kill activities in this state first to satisfy the resource requirements of higher priority activities.

Multitasking and the Activity Lifecycle

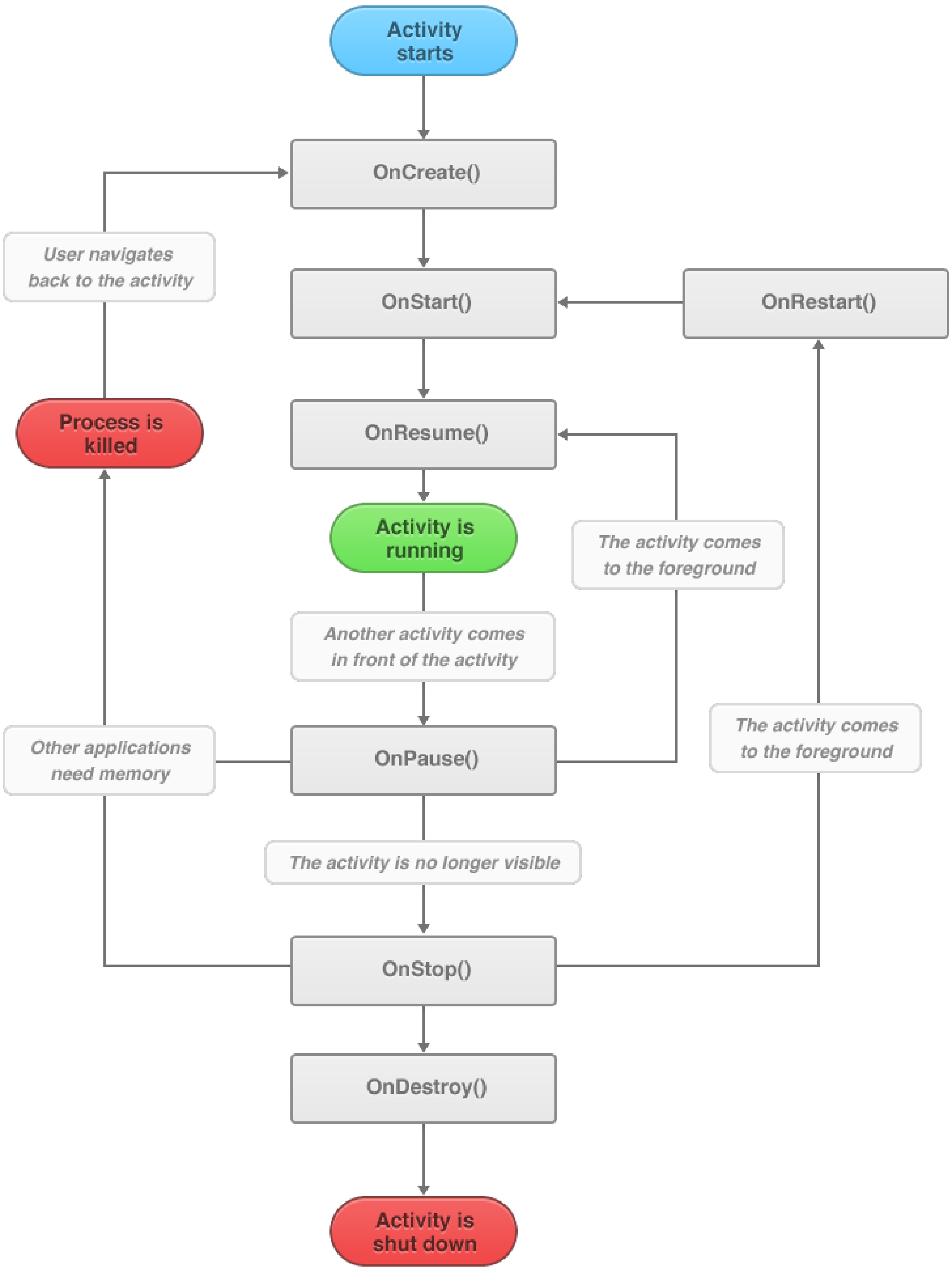
UI and multitasking functionality behaves differently in Android than on other mobile platforms. In Android, if a user navigates backwards using the back button on a device, this means that an activity has been navigated away from and the OS will then destroy that activity and reclaim its resources.

To use the multitasking capabilities of the operating system (for example, switch foreground applications), the user must press the home key to put their currently executing activity into the background. When this occurs, activities and their resources will be reclaimed, based on the operating system’s resource management heuristics triggered by the activity state condition (as described earlier).

Activity Lifecycle Methods

The Android SDK and, by extension, the Xamarin.Android framework provide a powerful model for managing the state of activities within an application. When an activity’s state is changing, the activity is notified by the OS, which calls specific methods on that activity.

The following diagram illustrates those methods and their names:


Activity starts
OnCreate()
User navigates back to the activity
OnStart()
OnRestart()
Process is killed
OnResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
OnPause()
The activity is no longer visible
OnStop()
OnDestroy()
Activity is shut down

As a developer, you can handle state changes by overriding these methods within an activity.

OnCreate

This method is called when the activity is first created. An activity should override this method to setup its main content view and perform any other initial setup, such as creating views, binding data to lists, etc. This method also provides an activity with a Bundle parameter, which is a dictionary for storing and passing state information and objects between activities. If the bundle is non-null, this indicates the activity is resuming and can be rehydrated. Activities can also make use of the Extras container of the Intent object to restore data previously saved, or data that is being passed between activities. For example, the following code illustrates how to retrieve values from the bundle, and also how to retrieve values from the Extras container:

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    string intentString;
    bool intentBool;
    string extraString;
    bool extraBool;
    if (bundle != null)
    {
        //if the activity is being resumed...
        intentString = bundle.GetString("myString");
        intentBool = bundle.GetBoolean("myBool");
    }

    //Check for the values in the Intent Extras...
    extraString = Intent.GetStringExtra("myString");
    extraBool = Intent.GetBooleanExtra("myBool", false);
    // Set our view from the "main" layout resource
    SetContentView(Resource.Layout.Main);
}
```

OnStart

This method is called when the activity is about to become visible to the user. Activities should override this method if they need to perform any specific tasks right before an activity becomes visible, such as: refreshing current values of views within the activity, or ramping up frame rates (a common task in game building).

OnPause

This method is called when the system is about to put the activity into the background. Activities must override this method if they need to commit unsaved changes to persistent data, destroy or cleanup other objects consuming resources, or ramp down frame rates, etc. Implementations of this method should return as quickly as possible, as no successive activity will be resumed until this method returns. The following example illustrates how to override the OnPause method to save data in the Extras container, to enable the data to be passed between activities:

```
protected override void OnPause()
{
    Intent.PutExtra("myString", "Hello Xamarin.Android OnPause");
    Intent.PutExtra("myBool", true);
    base.OnPause();
}
```

OnResume

This method is called when the activity will start interacting with the user after being in a pause state. When this method is called, the activity is moving to the top of the activity stack, and it is receiving user input. Activities can override this method if they need to perform any tasks after the activity begins accepting user input.

OnStop

This method is called when the activity is no longer visible to the user, because another activity has been resumed or started and is covering this one. This can happen because the activity is completing (Finish method was called) because the system is destroying this instance of the activity to save resources, or because an orientation change has occurred for the device. You can distinguish between these two scenarios by using the IsFinishing property. An activity should override this method if it needs to perform any specific tasks before it is destroyed, or if it’s about to initiate a UI rebuild after an orientation change.

OnRestart

This method is called after your activity has been stopped, prior to it being started again. This method is always followed by OnStart. An activity should override OnRestart if it needs to perform any tasks immediately before OnStart is called. For instance, if the activity has previously been sent to the background and OnStop has been called, but the activity’s process has not yet been destroyed by the OS, then the OnRestart method should be overridden.

A good example of this would be when the user presses the home button while on an activity in the application. The OnPause, and then the OnStop methods are called, but the activity is not destroyed. If the user were then to restore the application by using the task manager or a similar application, the OnRestart method of the activity would be called by the OS, during the activities reactivation.

OnDestroy

This is the final method that is called on an activity before it’s destroyed. After this method is called, your activity will be killed and purged from the resource pools of the device. The OS will permanently destroy the state data of an activity after this method runs, so an activity should override this method as a final means to save state data.

OnSaveInstanceState

This method is provided by the Android activity lifecycle framework to give an activity the opportunity to save its data when a change occurs, for example, a screen orientation change. The following code illustrates how activities can override the OnSaveInstanceState method to save data for rehydration when the OnCreate method is called:

```
protected override void OnSaveInstanceState(Bundle outState)
{
    outState.PutString("myString", "Hello Xamarin.Android OnSaveInstanceState");
    outState.PutBoolean("myBool", true);
    base.OnSaveInstanceState(outState);
}
```

Summary

The Android activity lifecycle provides a powerful framework for state management of activities within an application. This article introduced the different states that an activity can go through during the lifecycle. Next, it explored the different methods exposed by the activity lifecycle, and covered some use cases showing when a developer might want to override these methods. Finally, some sample code was provided that illustrated how to use the provided activity lifecycle methods to take advantage of some of the key state storage features in Android.

Source URL: http://docs.xamarin.com/guides/android/application_fundamentals/activity_lifecycle