

**Mobile Computing**  
**Professor Pushendra Singh**  
**Indraprastha Institute of Information Technology Delhi**  
**App Fundamentals**  
**Lecture 12**

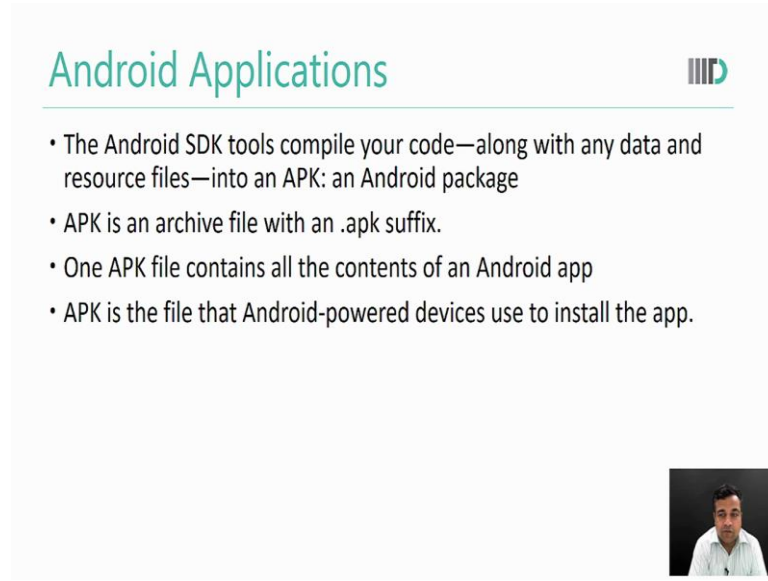
Hello, you have already developed an application and you already know few of the basics. In this lecture we will go deeply into some application fundamentals and we will cover some of the details that are needed to and we will cover some of the details that are needed to master android program.

(Refer Slide Time: 0:37)




For this lecture, my reference is android API Guide which is available at the given link.

(Refer Slide Time: 0:44)



## Android Applications

- The Android SDK tools compile your code—along with any data and resource files—into an APK: an Android package
- APK is an archive file with an .apk suffix.
- One APK file contains all the contents of an Android app
- APK is the file that Android-powered devices use to install the app.



Let us look at android applications once again. The android SDK tools compile your code along with any data and resource files into an APK file. An APK file is an android package file with the suffix .apk. As u may have seen when you developed your application.


This is in the same line as we have JAR for JAVA type files or other archive files that you may have used earlier in your programming career. In android the one APK file that you create for your application contains all the contents of an android app that means all the resources that had been using all the executable code etc.

APK is the file that any android powered device will use to install the app whether it is on tablet, whether it is on phone, whether it is on Google glass, or whether it is in TV that run on android. They all need an APK file.

(Refer Slide Time: 1:49)

## Android Applications

- Once installed on a device, each Android app lives in its own security sandbox
- The Android operating system is a multi-user Linux system in which each app is a different user.
  - By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app).
  - The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.



Android applications once they are installed on a device, they live in their own security sandbox. This is a very good approach to make sure that one application does not destroy the whole system. Earlier in computing systems, you may have seen that if one application crashed, sometimes your system needed to be rebooted.

In fact you may have experienced the same thing with your PC from time to time. However this situation is not desirable with a mobile phone and specially with an open market where multiple applications are developed from various entities so we cannot allow the situation in which a crash by an application requires your phone to reboot.

So the Android operating system which is an approach that it creates a sandbox for each application. A sandbox is like a virtual box in which your application runs and if it crashes the effects remain limited to the box. The Android sandbox approach comes from the JAVA sandbox techniques.

The Android operating system itself is a multi-user Linux system and each app acts as a different user. So as you may have seen in the multi-user system each user is separated by another user. And each user is uniquely identified by the system. Similarly in Android each Android application is identified by a unique Linux user ID and one application is separated from another application and the

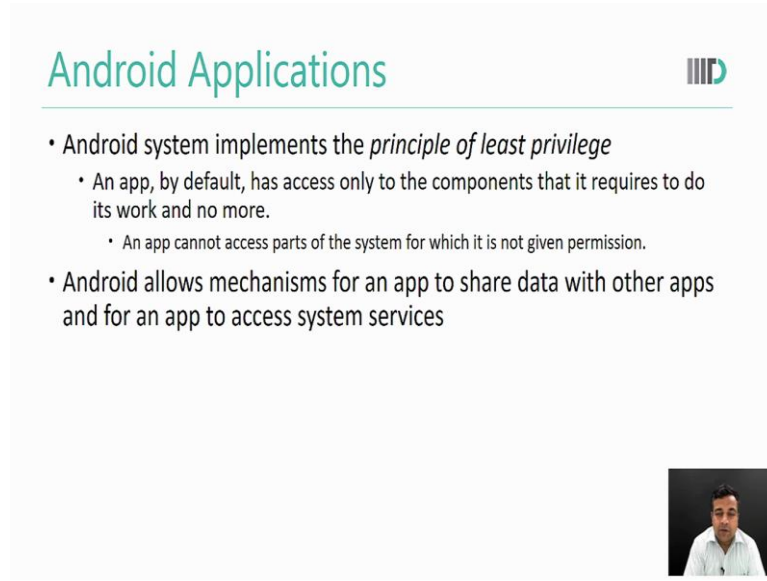
system sets permission for all the files in an app so that only user ID assigned to that app can access them.

These are some very sound operating system principles. You may have studied some of them while doing your operating system codes. Android applies all of them to ensure that application run in a linked environment where if an application crises all wants to maliciously access data for other application. It is not allow to impact the system.

So each process in android has its own virtual machine and an app code runs in complete isolation from other apps. So by default an application will run as its own Linux process. Android starts the process when any of the app's components need to be executed. We will soon learn about the different app components.

And when the application stops then the system shuts down the process when is no longer needed and the system recover memory and other resources for other applications that are running. You may have studied some of the principles in the operating system if not then maybe you would like to revise you operating system concept once again. Overall android is very very similar to what a Linux operating system is because that is the base on which the android is build up on.

(Refer Slide Time: 5:07)



The slide features the title "Android Applications" in a teal font at the top left, with a logo consisting of three vertical bars and a stylized 'D' at the top right. Below the title, there are two main bullet points. The first bullet point is "Android system implements the *principle of least privilege*", which is followed by two sub-bullets: "An app, by default, has access only to the components that it requires to do its work and no more." and "An app cannot access parts of the system for which it is not given permission." The second main bullet point is "Android allows mechanisms for an app to share data with other apps and for an app to access system services". In the bottom right corner of the slide, there is a small video thumbnail showing a man in a white shirt.

- Android system implements the *principle of least privilege*
  - An app, by default, has access only to the components that it requires to do its work and no more.
  - An app cannot access parts of the system for which it is not given permission.
- Android allows mechanisms for an app to share data with other apps and for an app to access system services


Android system implements what we call as the principle of least privilege which means that an application will have just as much privilege as it needs. That is an application by default can access only the components that it requires to do its work and nothing more. So an app cannot access parts of the system for which it is not given permission.

This ensures that an application maliciously or non maliciously do not affect the system at large. However with the principle of least privilege android also allows mechanisms for an application to share data with other app and for an app to access system services. As you may imagine this is very important and necessary to develop wonderful applications for example if you want to develop an application which access its camera that application must have some way to communicate with the camera app.

(Refer Slide Time: 6:12)

## Applications – Resource Access

- An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more.
- The user has to explicitly grant these permissions.
- It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files.
- To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM (the apps must also be signed with the same certificate).



So an app can request permission to access device data whenever it wants. For example user's contacts, SMS messages, the mountable storage, camera, Bluetooth and more. Android requires that the user explicitly grant these permissions. You have already experienced that when you installed an app, and the app asks your permission for example if you install an app such as PayTm which many of us use for paying uhh which many of us use for paying bills.

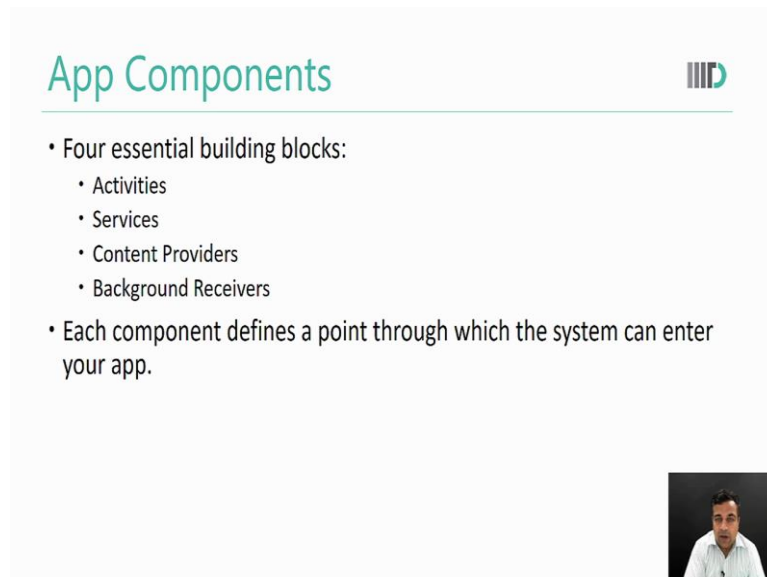
It requires access to our SMS messages because sometimes it wants to access SMS to authenticate itself and to validate its user. Similarly there are other applications which require access to your contact list or to your camera or to your GPS or any other system resource and every time an explicit permission from the user is required at a time of the installation.

It is also possible in android that two apps share the same Linux user ID because they can share the same Linux user ID they can access each other's files. We will learn more about it in later weeks when we have term then. We will learn about later weeks, once we developed to our advanced programming environment.

We will learn more about it in later weeks, once we have developed our basic understanding and would develop more advanced applications. Now to conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM. But these apps must then also signed with the same certificate.


Don't worry about these details, we will be discussing them much more in (det)  
don't worry about these details we will be discussing them in later chapters.  
Don't worry about details, don't worry about these details we will be discussing  
them in more detail in later.

(Refer Slide Time: 8:28)



## App Components

- Four essential building blocks:
  - Activities
  - Services
  - Content Providers
  - Background Receivers
- Each component defines a point through which the system can enter your app.




Now, let us look at app components. An android application (th) now let us look  
at app components. There are essentially four building blocks for any android  
application. An android application may have either one of them or two of them  
or three of them or all four of them.

You have already experienced one of it. It's called activities. The other building  
blocks are services, content providers and background receivers. Each of these  
components defines a point through which the system can enter your application.

(Refer Slide Time: 9:12)

## App Component

- Activity
  - An *activity* represents a single screen with a user interface.
  - An activity is implemented as a subclass of Activity
- Services
  - A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes.
  - A service does not provide a user interface.
  - Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.
  - A service is implemented as a subclass of Service



Activity. As you already have seen activity represents a single screen with a user interface. An activity is implemented as a subclass of Activity class. For example if you have an email app then there may be an activity that shows you the list of emails. Another activity to compose an email. And another activity for reading email. So the single email app may consist of three activities.

Currently in our math quiz application, we are only using one activity but very soon we will move to implement more activities into same applications. And all though the activities work together uhh and all though the activities work together (prohide). And all though the activities work together provide an osc experience to the user.

They are independent of others. Uhh and as such a different application can start any one of the activities. If the parent applications allows and as such a different application can start any one of the activity of another application as long as it is provided by the permissions. For example a camera app can start the activity in the email app that composes new email in order a user to share a picture.

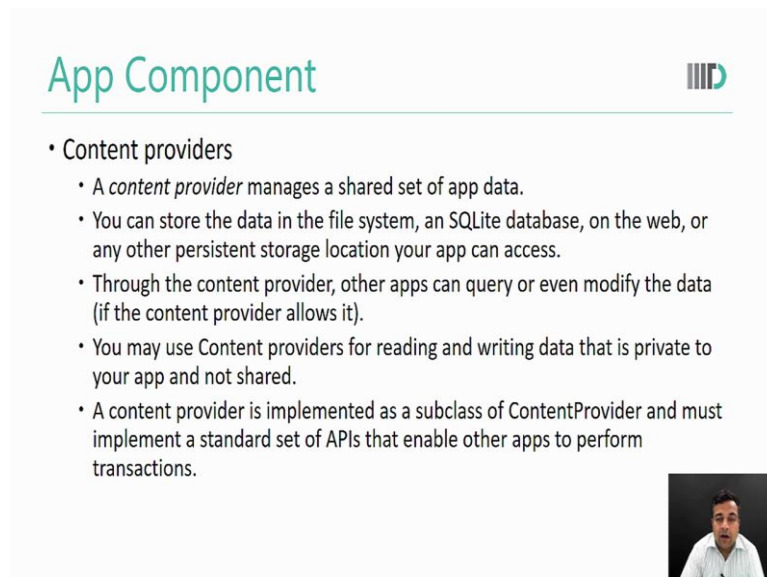
You may have experienced some of these when you open your Whatsapp or your Facebook application and you want to share something. An activity is also implemented. An activity is implemented as a subclass of activity and as you have already developed one activity in your program we will be learning more about activities later on.



The another building block is one has called a service. A service is a component that runs in the background to perform long running operations or to perform work for remote processes. A service does not provide a user interface. Let us take some example of services. One of the example could be a service that downloads data and the background as you may imagine (down) as you may imagine downloading data and the background does not fully require a user interface.


Another example is a service that plays music in the background while the user is in a different app. So there could be various services that you can implement which work in the background. Another component such as an activity can start the service and let it run or bind it to others in order to interact with it. A service is implemented as a subclass of a service. Let's look at the another app component.

(Refer Slide Time: 12:46)



**App Component**

- Content providers
  - A *content provider* manages a shared set of app data.
  - You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access.
  - Through the content provider, other apps can query or even modify the data (if the content provider allows it).
  - You may use Content providers for reading and writing data that is private to your app and not shared.
  - A content provider is implemented as a subclass of `ContentProvider` and must implement a standard set of APIs that enable other apps to perform transactions.



Content providers. A content provider manages a shared set of app data. As you can imagine from the name that it is providing some content. Now you can store this data in the file system. In a (sik in a) Now you can store this data in a file system an SQLite database, on the web or any other persistent storage location your app can access.


And through the content provider other apps can query or even modify the data as long as the content provider allows it. You may also use content providers for reading and writing data that is private to your app and not shared.

That is a content provider can be used both for saving private data and also the data that you may wish to share others. Content provider is implemented as a subclass of content provider and it must implement a standard set of API's that enable other apps to perform transactions.

(Refer Slide Time: 13:57)

## App Component

- Broadcast receiver
  - It is a component that responds to system-wide broadcast announcements.
    - E.g. a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
  - Apps can also initiate broadcasts
    - E.g. to let other apps know that some data has been downloaded to the device and is available for them to use.
  - Broadcast receivers may create a status bar notification to alert the user when a broadcast event occurs.
  - A broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work.
    - E.g. initiate a service to perform some work based on the event.
- A broadcast receiver is implemented as a subclass of `BroadcastReceiver` and each broadcast is delivered as an *intent*.



The fourth component is the broadcast receiver. Broadcast receiver is a component that responds to system wide broadcast announcements. For example a broadcast announcing that the screen has turned off, or that the battery is now low sometimes you may have seen a notification says battery is low 15% or battery is low 3% please plug in your phone.

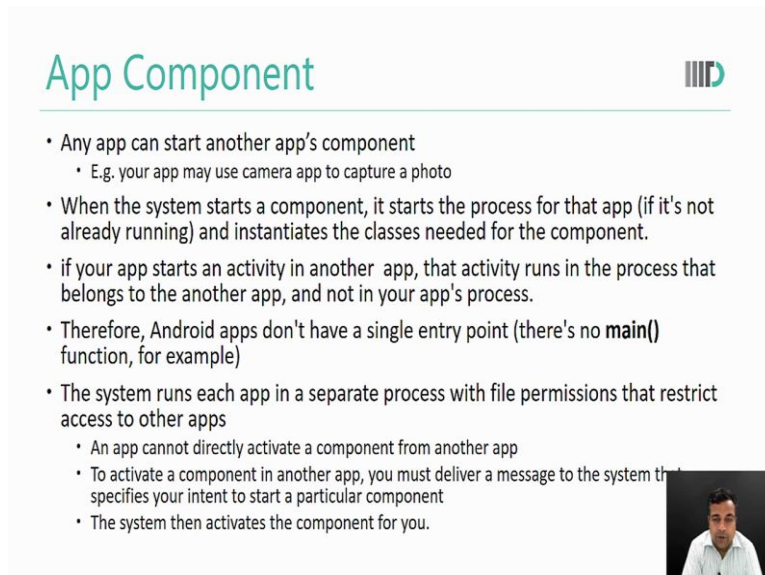
So these are the kind of broadcast that we are talking about it. An app can also initiate broadcasts so for example an app may want to broadcast that has downloaded the data that you wanted to download. The broadcast receiver may create a status bar notification they do not really need a screen. But status bar notification is very common for broadcast receiver to implement so that they can alert the user when the correspondent broadcast event has occurred.

A broadcast receiver is just a gateway to other components and is intended to do a minimal amount of work. Broadcast receiver is implemented as a subclass of

broadcast receiver and each broadcast is delivered as an intact. So far we have not covered intent in our lectures neither our application has any intent building.

As of now you can understand intent as only a message object. That is it is a message which is delivered to get some action done. We will learn more about intent in our later classes.

(Refer Slide Time: 15:38)



## App Component

- Any app can start another app's component
  - E.g. your app may use camera app to capture a photo
- When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component.
- If your app starts an activity in another app, that activity runs in the process that belongs to the other app, and not in your app's process.
- Therefore, Android apps don't have a single entry point (there's no `main()` function, for example)
- The system runs each app in a separate process with file permissions that restrict access to other apps
  - An app cannot directly activate a component from another app
  - To activate a component in another app, you must deliver a message to the system that specifies your intent to start a particular component
  - The system then activates the component for you.

Now the beauty of android lies in that in any app can start another apps component. Just as I give previous example your app may want to use the camera app to capture a photo. And when the system starts a component it is starts the process for that app if it is not running. And instantiates the classes needed for the component.

If your app starts an activity in another app and that activity runs in the process that belongs to another app and not in your app's process. Let's go through these points again. Suppose you want to run your application and your application has a component let's take activity. The moment you want to start this activity there will be a corresponding process for your app.

And all the classes needed for this component activity will be instantiated. Now this activity wants to initiate another activity in another app. Now the new activity will run in the process that belongs to the other app and not your process. Because essentially you are running an activity of an another application.

Due to this feature android apps do not have a single entry point. As you may have may imagine as you may imagine an apps component can be instantiated by any other app so there is not a single entry point. A camera app may have many components and these components may be initiated in a different order by different apps.


And this is also the reason why there is no main function in an android program. Some of you already have noticed that they could not find the main function. Others may have not looked closely. But now you can look for this and you will not find the main function. Because the main function essentially implies that that's where the execution starts.

This is not true for the android application. An android application may start whenever any component is initialized. As told earlier the system runs each app in a separate process with permission that restricts access to other app. So an app cannot directly activate a component from another app. But what it does is that it makes a request to the system. And when it makes a request to the system, the system activates that component for the app.

(Refer Slide Time: 18:36)

## Intent & Activating Components

- Activity, Services, and Broadcast receivers—are activated by an asynchronous message called an intent.
  - For activities and services, an intent defines the action to perform
  - For broadcast receivers, the intent simply defines the announcement being broadcast
- Content provider, is not activated by intents. It is activated when targeted by a request from a ContentResolver

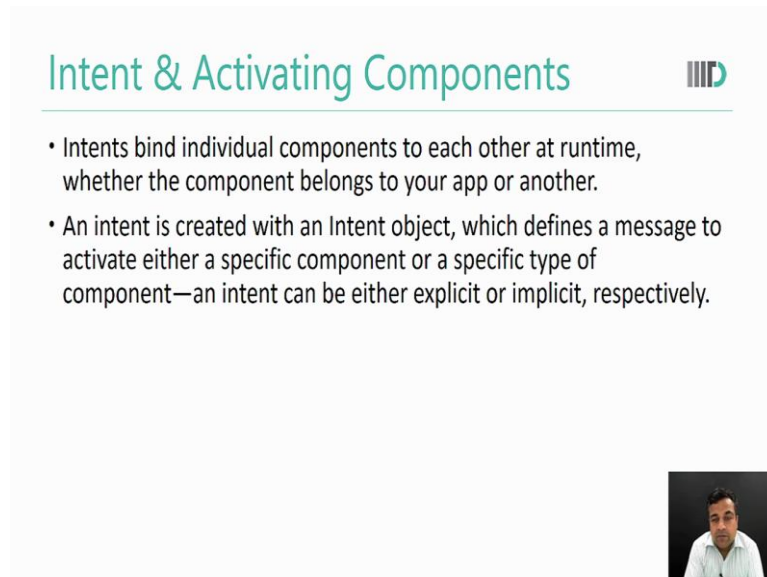


Let us learn a little bit more about intent and activity components. Activity, services and broadcast receivers these three components are activated by an asynchronous message which we called an intent. Like I told you earlier intent is

more like a message object. For activity and services an intent defines the action to perform.

While for broadcast receivers, the intent defines the announcements being broadcast. For the fourth component the content provider it is not activated by intents. It is activated when targeted by a request from a content resolver. We will read more about it in later lectures.

(Refer Slide Time: 19:32)




The slide features a title 'Intent & Activating Components' in teal text at the top left, followed by a teal Android logo icon at the top right. Below the title, there are two bullet points: 'Intents bind individual components to each other at runtime, whether the component belongs to your app or another.' and 'An intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component—an intent can be either explicit or implicit, respectively.' In the bottom right corner of the slide, there is a small, square video inset showing a man in a light blue shirt speaking.

So in intent bind individual components to each other at runtime, whether the component belongs to your app or another. These are like the messenger which goes from one component to another component. You create an intent in your program with an intent object. An intent object defines a message to activate either a specific component or a specific type of component.


In android we can define both implicit and explicit intents. Do not worry too much we will soon be implementing intents in our program and then you can come back to these lectures and make more sense of what we have learnt. And there will also be more details in later chapters about intents.

(Refer Slide Time: 20:28)

## The Manifest File



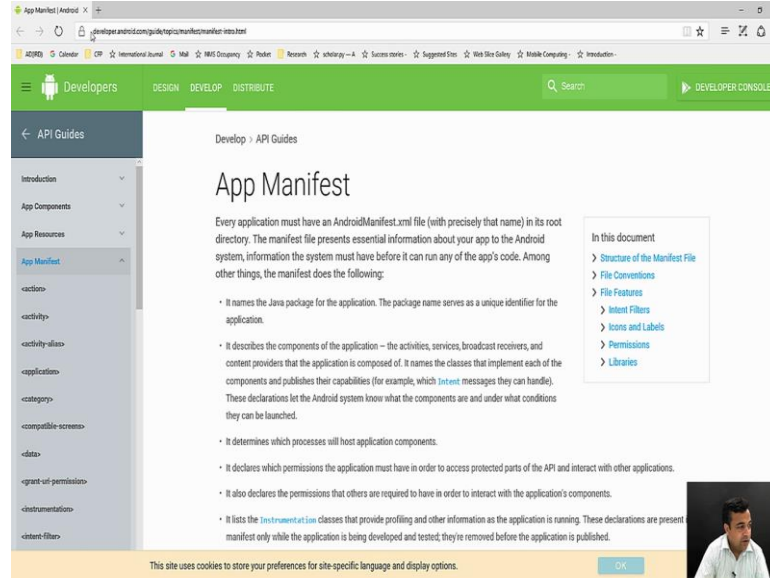
- Declares all the components of the app
- Used by Android System to know what components exist in an app
- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API Level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
- Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.
- <https://developer.android.com/guide/topics/manifest/manifest-intro>



Now the last topic of this lecture the manifest file. We have already see the manifest file in project. The manifest file declares all the components of the app. It is used by the android system to know what components exist in an app. Manifest file identifies the user permissions that app requires.

For example X is to (( ))(20:55) or to use user's contact or to use user's SMS services, camera app, GPS etc etc. The manifest file also declares minimum API level required by the app. And it also declares hardware and software features used or required by the app, it also declares API libraries that a app needs to be linked against, such as if you are using Google maps library. A very good detail instruction of a manifest of manifest file is available at this link. Let's click it and try to see.

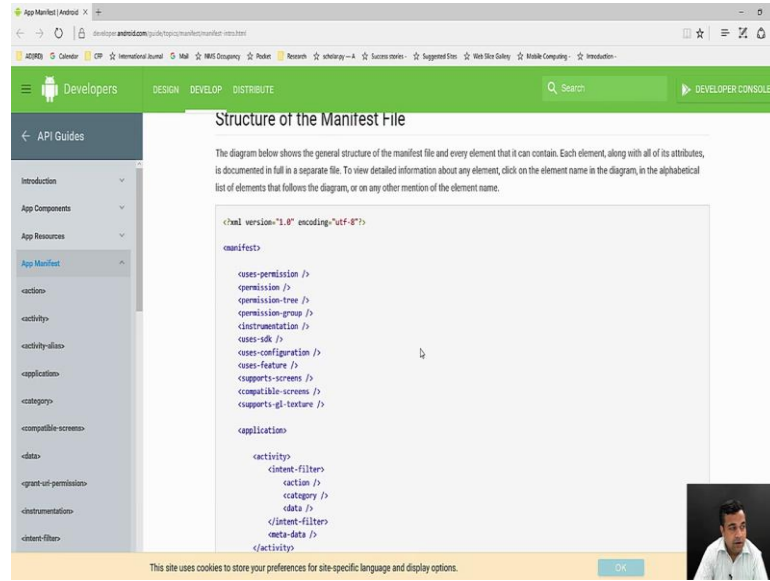
(Refer Slide Time: 21:42)



You can see this the same link which I had given. It defines app manifest. As you can see that this is the same link that I had given if you click on it this page below. Let's go through it to understand the manifest file. So every android application must have a manifest file. Every android application must have a manifest file and that manifest file must be name androidmanifest.xml.

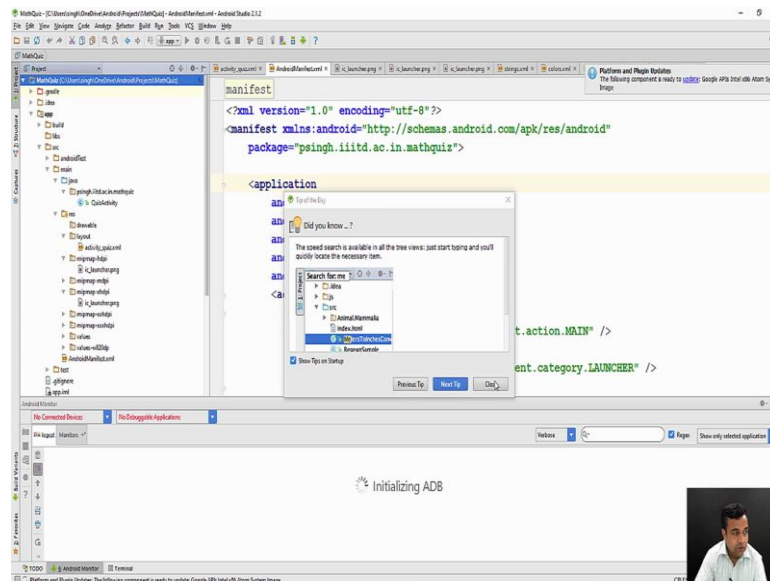
This name is precise that is you cannot change it. The manifest file uses all the essential information about your application that the android system requires. It needs the JAVA package describes the components of the application determines which processes will goes to application components declares permission etc etc. Now let us look at the structure of manifest file.

(Refer Slide Time: 22:53)

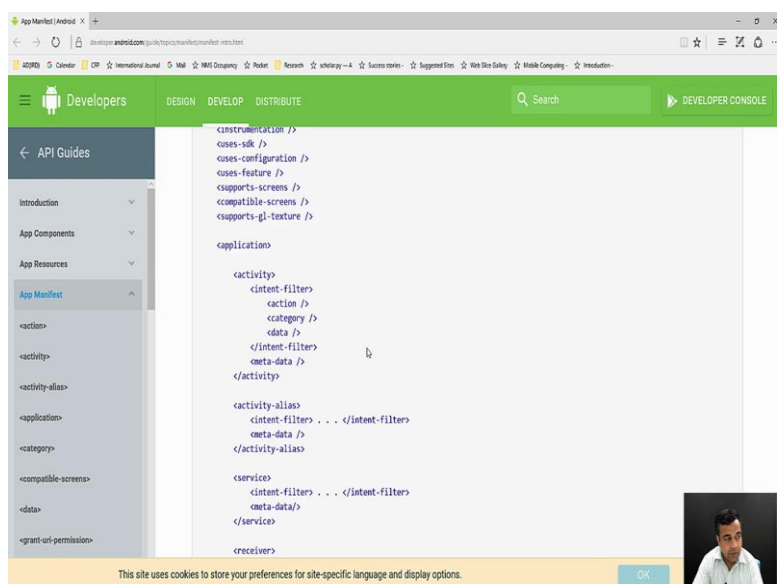
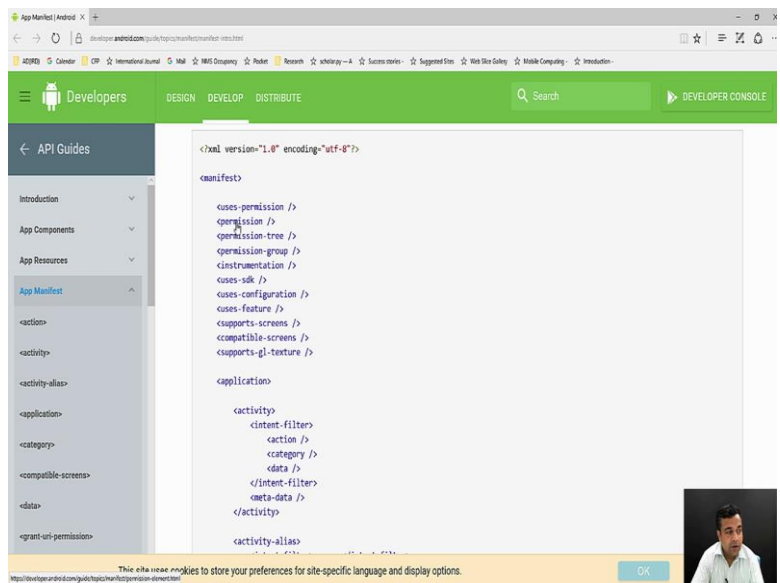
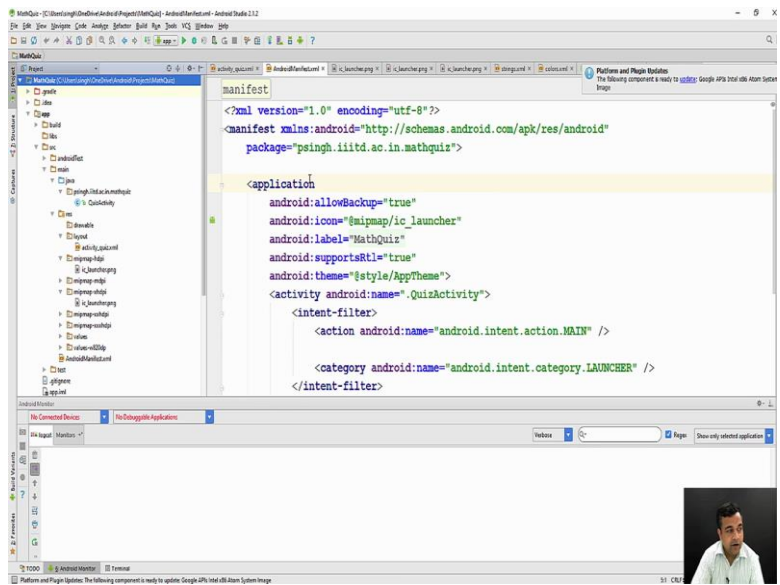


As you can see that the manifest start. As you can see that manifest file starts from the manifest tag and then it goes into user formations. User SDK configuration, features etc etc. And then it comes to application. Let us try to see what these fields are I will also open our program that we have developed.

(Refer Slide Time: 23:22)







```
<action />
<category />
<data />
</intent-filter>
<meta-data />
</activity>

<activity-alias>
<intent-filter> . . . </intent-filter>
<meta-data />
</activity-alias>

<service>
<intent-filter> . . . </intent-filter>
<meta-data />
</service>

<receiver>
<intent-filter> . . . </intent-filter>
<meta-data />
</receiver>

<provider>
<grant-uri-permission />
<meta-data />
<path-permission />
</provider>

<uses-library />
```

```
<meta-data />
</service>

<receiver>
<intent-filter> . . . </intent-filter>
<meta-data />
</receiver>

<provider>
<grant-uri-permission />
<meta-data />
<path-permission />
</provider>

<uses-library />

</application>

</manifest>
```

All the elements that can appear in the manifest file are listed below in alphabetical order. These are the only legal elements; you cannot add your own elements or attributes.

- <action>
- <activity>
- <activity-alias>
- <application>

This is our manifest file. As it is given here. Without having anything on top we don't have anything on top except the package given. Then we have the application. Inside the application some fields are given. We will try to understand them. But let's go back to the original structure of the manifest file.

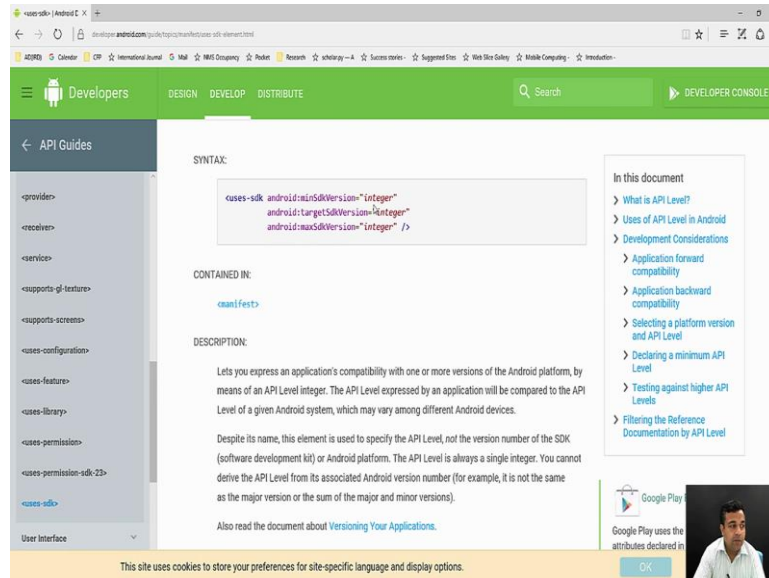
A manifest file starts with following fields. User permissions, SDK, configurations, support system etc. Then it comes to application components, activity, services, broadcast receivers and content providers. If you click on any of these fields you will get the detailed value of that.

(Refer Slide Time: 24:34)

The screenshot shows the Android Developer website's API Guides section for the `<permission-tree>` element. The page includes a navigation sidebar on the left with categories like Introduction, App Components, App Resources, App Manifest, User Interface, Animation and Graphics, Computation, Media and Camera, Location and Sensors, Connectivity, Text and Input, Data Storage, and Libraries. The main content area displays the element name, its syntax, and a list of elements it contains. The syntax is shown as `<permission-tree android:icon="drawable resource" android:label="string resource" android:name="string" />`. The 'CONTAINED IN:' section lists `<manifest>`. The 'DESCRIPTION:' section explains that it declares the base name for a tree of permissions and provides an example of permissions like `com.example.project.taxes.CALCULATE` and `com.example.project.taxes.deductions.MAKE_SOME_UP`.

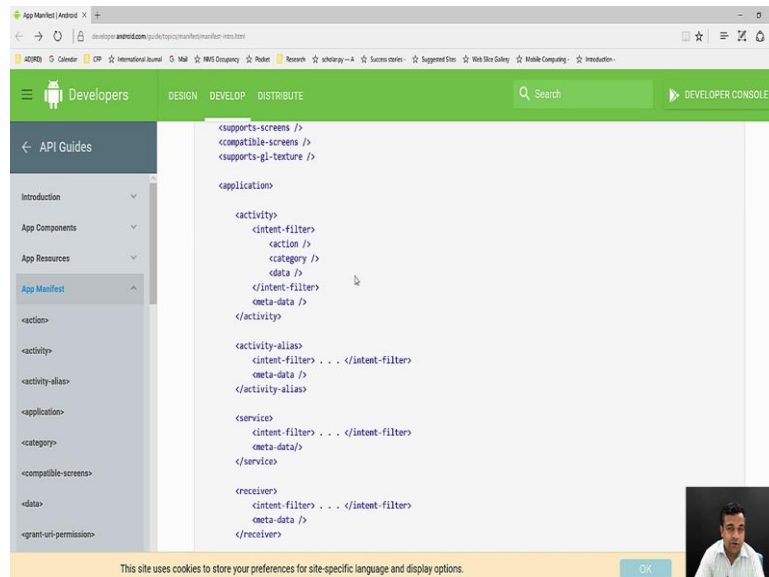
The screenshot shows the Android Developer website's API Guides section for the `<manifest>` element. The page includes a navigation sidebar on the left with categories like Introduction, App Components, App Resources, App Manifest, User Interface, Animation and Graphics, Computation, Media and Camera, Location and Sensors, Connectivity, Text and Input, Data Storage, and Libraries. The main content area displays the element name, a diagram showing its structure, and its syntax. The diagram shows a tree structure of elements including `<uses-permission />`, `<permission />`, `<permission-tree />`, `<permission-group />`, `<instrumentation />`, `<uses-sdk />`, `<uses-configuration />`, `<uses-feature />`, `<supports-screens />`, `<compatible-screens />`, `<supports-gl-texture />`, and `<application>`. The 'CONTAINED IN:' section lists `<manifest>`. The 'DESCRIPTION:' section explains that the diagram shows the general structure of the manifest file and every element it can contain.

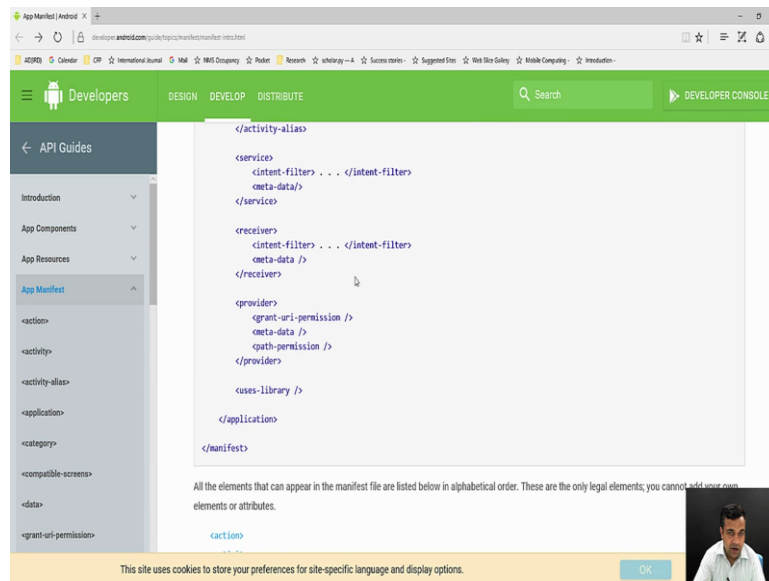
The screenshot shows the Android Developer website's API Guides section for the `<uses-sdk>` element. The page includes a navigation sidebar on the left with categories like Introduction, App Components, App Resources, App Manifest, User Interface, Animation and Graphics, Computation, Media and Camera, Location and Sensors, Connectivity, Text and Input, Data Storage, and Libraries. The main content area displays the element name, its syntax, and a list of elements it contains. The syntax is shown as `<uses-sdk android:minSdkVersion="integer" android:targetSdkVersion="integer" android:maxSdkVersion="integer" />`. The 'CONTAINED IN:' section lists `<manifest>`. The 'DESCRIPTION:' section explains that it lets you express an application's compatibility with one or more versions of the Android platform, by means of an API Level integer. The API Level expressed by an application will be compared to the API Level of a given Android system, which may vary among different Android devices. A sidebar on the right titled 'In this document' lists links to other sections like 'What is API Level?', 'Uses of API Level in Android', 'Development Considerations', 'Application forward compatibility', 'Application backward compatibility', 'Selecting a platform version and API Level', 'Declaring a minimum API Level', 'Testing against Levels', and 'Filtering the Reference Documentation'.



So in your free time you may want to click on it and find out what it means. For example user SDK must be main SDK version target as to (())(24:48) SDK version . You have already seen some of these earlier when we were setting our project.

(Refer Slide Time: 24:55)

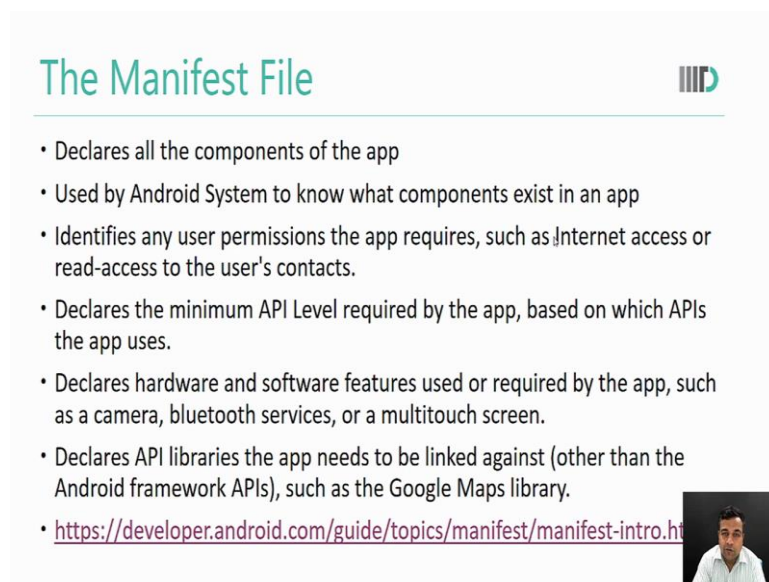




```
</activity-alias>
<service>
  <intent-filter> . . . </intent-filter>
  <meta-data />
</service>
<receiver>
  <intent-filter> . . . </intent-filter>
  <meta-data />
</receiver>
<provider>
  <grant-uri-permission />
  <meta-data />
  <path-permission />
</provider>
<uses-library />
</application>
</manifest>
```

Whatever settings you did in your project come in a manifest file and your manifest file is used by android to understand your application requirements.

(Refer Slide Time: 25:07)



## The Manifest File

- Declares all the components of the app
- Used by Android System to know what components exist in an app
- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API Level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
- Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.
- <https://developer.android.com/guide/topics/manifest/manifest-intro.html>

That is it for today's lecture. In this lecture we went through some more fundamental knowledge of android application and we also went through the manifest file and you also came to know that why android application is do not have a main function. For this lecture I am using the reference of android API guides which are available on developer.android.com. Thank you!