

CS710

Mobile & Pervasive Computing



Lecture 1

“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”

-- Mark Weiser in his seminal paper on ubiquitous computing

Pervasive Computing

“Computing that is omnipresent and is, or appears to be, everywhere all the time; may involve many different computing devices that are embedded in various devices or appliances and operate in the background.”

*“The age of **calm technology**, when technology recedes into the background of our lives.” “The practice of making computers so common and accessible that users are not even aware of their physical presence. The ideal of ubiquitous computing could be defined as a high-speed network that covers any kind of geography and is easily installed and automatically maintained.”*

Ubiquitous Computing (ubicom): *is a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities.*

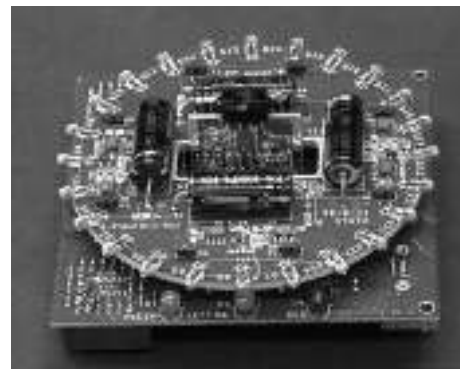
An integration of microprocessors into everyday objects like furniture, clothing, white goods, and toys even paints.

- The original term ubiquitous computing was coined by Mark Weiser in 1988 at Xerox PARC
- The essence of Weiser’s vision is that mobile and embedded processors can communicate with each other and the surrounding infrastructure, seamlessly coordinating their operation to provide support for a wide variety of everyday work practices.

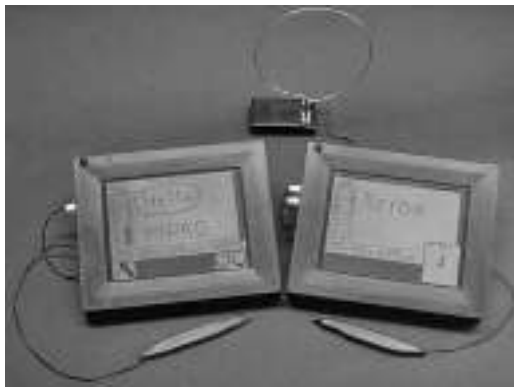
Evolution of Pervasive Computing

- Weiser believed that in a ucomp world, computation could be integrated with common objects that you might already be using for everyday work practices, rather than forcing computation to be a separate activity.
- If the integration is done well, you may not even notice that any computers were involved in your work.
- Weiser sometimes also referred to this as *invisible computing*
- ParcTab, or Tab, an inch-scale computer that represented a pocket book or wallet
- Tabs communicated wirelessly with a ceiling-mounted base station using 10 kbps

- diffuse infrared signaling



- ParcPad, or Pad, a foot-scale device, serving the role of a pen-based notebook or e-book reader
- ParcPads employed a similar design approach using a low-bandwidth X-protocol across a radio link, communicating with a basestation through a proprietary short-range near-field radio



- Liveboard provides the functionality of a whiteboard
- Liveboards were designed around standard computer workstations, but with much larger pen-based displays, and pen-based input



- In the mid-1990s, IBM began a research direction it called *pervasive computing* (IBM Mobile and Pervasive Computing)
- IBM, to its credit, was one of the first companies to investigate the business opportunity around pervasive systems, and created a business unit dedicated to the task.
- One of the first commercial deployments of a pervasive computing system was born from a collaboration between IBM Zurich and Swissair in 1999 (IBM Swissair), enabling passengers to check-in using Web-enabled (WAP) cell phones
- Once the passengers had accessed the service, the phone also served as a boarding pass, showing gate seat and flight departure information, and identifying the traveler as having valid flight credentials.
- Although this was one of the most publicized projects, IBM also applied these technologies to other service opportunities in banking and financial services, gaining early experience in this area.



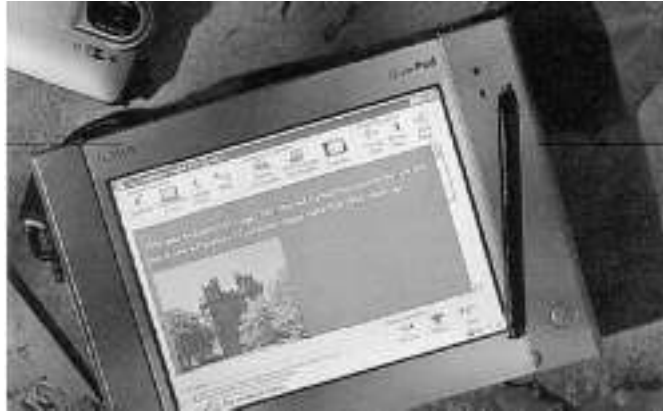
- Taking another approach to ubiquitous computing, *wearable computing* puts the emphasis on a portable computer that can be unobtrusively integrated with a person's clothing, while still being comfortable for the wearer
- An important related topic is augmented reality in which a computer is able to overlay information on top of what a user sees in order to improve ability to carry out a task.
- In an expanded view of ubiquitous computing with a suitably unobtrusive heads-up display embedded in our eyewear, augmented reality could become an indispensable tool of the future in much the same way we have come to rely on the cell phone today

Pervasive Computing Projects

- Classroom 2000 began in July 1995 with a bold vision of how ubiquitous computing could be applied to education and provide added value to standard teaching practices in the future
- Classroom 2000 investigated the possibility of capturing the entire lesson in a form that would be a useful reference itself.
- Key challenge was to create index points that enabled students to skip over a block of video of little interest and be able to jump to the exact point in time that might provide the answer to a question
- Furthermore, these index points needed to be automatically generated, with clear meaning to anybody who wanted to use them
- With an electronic board (the Xerox Liveboard was used for some of the work), it is possible to timestamp all the annotations made by a teacher during the lesson, along with slide transitions during a presentation, and other user-generated input, and these were used to index the audio and visual record of the lesson.
- Thus, the combined media, timeline, and indices represent a powerful summary that can be immediately made available to the students when the class finishes.

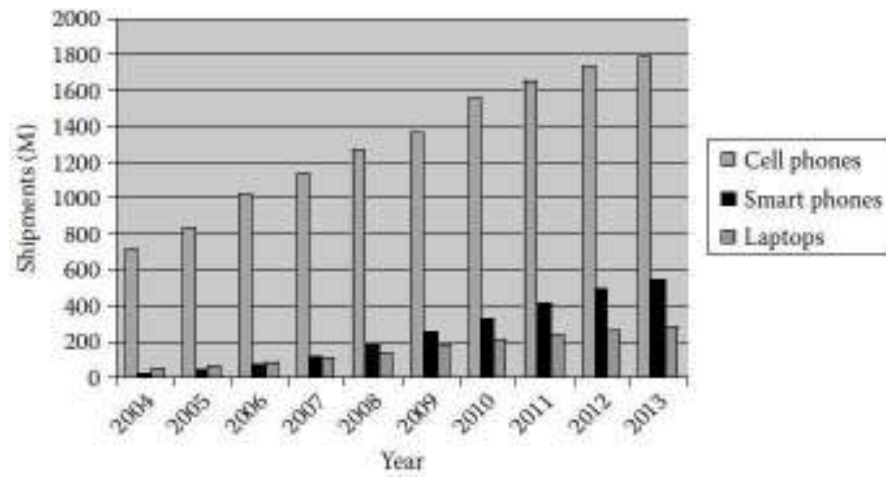


- In 1999, the Aware Home project was founded and set out to explore how computation and embedded technologies could support everyday activities in a home.
- In the spirit of Living Laboratories, a complete residential building was designed from scratch, providing all of the expected features in a modern home, but with additions to support embedded computation and sensing, wiring conduits, and a control center.
- Systems introduced into the Aware Home to support the research included cameras and RFID tags to identify and track an occupant's location, and various forms of sensors. For example, the house included a smart floor composed of a network of pressure sensors that could identify the characteristic ambulatory gait of individuals as they moved between rooms, thus providing additional means of occupant identification.
- The GUIDE project was created, obtained a government grant, and captured the imagination of many researchers interested in location-based services in the wild.
- It was the first mobile electronic guidebook designed and optimized from concept to implementation for use by tourists.



Modern Directions

- SenseCam is a small wearable computer that periodically captures images of the world as a user moves around.
- In collaboration with the MyLifeBits project, SenseCam provides a wealth of contextual data about the wearer, augmented by a database describing documents and other electronic media that the individual has accessed.
- The result is a prosthetic memory aid that can be used to answer basic questions about an individual's life, enabling more detailed recall than most of us could achieve by unaided means
- RADAR was the first example of a wireless system allowing mobile computers to locate themselves in a building [an indoor global positioning system (GPS)].
- It was designed around the first WiFi (802.11b) radios that were commercially available, and made use of reception maps in a Microsoft building hosting several access points (APs) with the data collected using a wireless survey tool. By comparing the received signal strength indication (RSSI) reading for each AP measured at a point of interest with the RSSI signal maps on record, the system could automatically determine a mobile computer's most likely location to an accuracy of 2–3 meters.
- EasyLiving, established in 1999, was MS's closest project to the spirit of the original ubicomp vision.
- The research was centered on a smart room designed to support both work and recreational activities.
- A key ingredient was the use of image processing to recognize activities in the space, making use of multiple cameras to track the occupants, and objects situated in the room.
- Of particular note was the ability to migrate computing sessions from screen to screen as people moved around, and mechanisms to automatically control the lighting and music in the space to best suit all the occupants
- Place Lab was the best known of the projects from IRS, exploring and building a system that could determine the location of a mobile device by cataloguing and mapping WiFi access points throughout a city
- This was later extended to GSM towers, and demonstrated effective location-based services for cell phone applications.
- The project was rolled out to the research community as a sandbox to experiment with location based services on standard mobile platforms.
- It had a distinct advantage over GPS, in that it worked well indoors and did not require the purchase of additional GPS equipment for operation outdoors.



Lecture 2

Wearable Computing?



“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”

-- Mark Weiser in his seminal paper on ubiquitous computing

- Omnipresence, background execution, Accessibility, embed-ability

Example Scenarios – Aura

- XYZ is at Gate 23 in the Lahore airport, waiting for her connecting flight. She has edited many large documents, and would like to use her wireless connection to e-mail them. Unfortunately, bandwidth is miserable because many passengers at Gates 22 and 23 are surfing the Web. Aura observes that at the current bandwidth XY) won't be able to finish sending her documents before her flight departs. Consulting the airport's network weather service and flight schedule service, Aura discovers that wireless bandwidth is excellent at Gate 15, and that there are no departing or arriving flights at nearby gates for half an hour. A dialog box pops up on XY's screen suggesting that she go to Gate 15, which is only three minutes away. It also asks her to prioritize her e-mail, so that the most critical messages are transmitted first. XY) accepts Aura's advice and walks to Gate 15. She watches TV there until Aura informs her that it is close to being done with her messages, and that she can start walking back. The last message is transmitted during her walk, and she is back at Gate 23 in time for her boarding call.
- Fred is in his office, frantically preparing for a meeting at which he will give a presentation and software demonstration. The meeting room is a 10-minute walk across campus. It is time to leave, but Fred is not quite ready. He grabs his wireless handheld computer and walks out of the door. Aura transfers the state of his work from his desktop to his handheld, and allows him to make his final edits using voice commands during his walk. Aura infers where Fred is going from his calendar and the campus location tracking service. It downloads the presentation and the demonstration software to the projection computer, and warms up the projector. Fred finishes his edits just before he enters the meeting room.
- As he walks in, Aura transfers his final changes to the projection computer. As the presentation proceeds, Fred is about to display a slide with highly sensitive budget information. Aura senses that this might be a mistake: the room's face detection and recognition capability indicates that

there are some unfamiliar faces present. It therefore warns Fred. Realizing that Aura is right, Fred skips the slide. He moves on to other topics and ends on a high note, leaving the audience impressed by his polished presentation.

Pervasive Computing

- These scenarios embody two key ideas in pervasive computing
- Proactivity
 - Combining knowledge from different layers of the system
- Self-Tuning
 - automatically adjusting behavior to fit circumstances
 - ability to move execution state effortlessly across diverse platforms
- Consulting the airport's network weather service and flight schedule service
- ... The last message is transmitted during her walk, and she is back at Gate 23 in time for her boarding call.
- ... Aura infers where Fred is going from his calendar and the campus location tracking service

Composition of Pervasive Computing

- Pervasive computing represents a major evolutionary step in a line of work dating back to the mid-1970s.
- Two distinct earlier steps in this evolution are distributed systems and mobile computing.
- Some of the technical problems in pervasive computing correspond to problems already identified and studied earlier in the evolution.
 - In some of those cases, existing solutions apply directly;
 - In other cases, the demands of pervasive computing are sufficiently different that new solutions have to be sought.
 - There are also new problems introduced by pervasive computing that have no obvious mapping to problems studied earlier.

Distributed Systems

- The field of distributed systems arose at the intersection of personal computers and local area networks.
- Conceptual framework and algorithmic base developed in this paradigm later became basis for all work involving two or more computers connected by a network
- The knowledge base developed spans many areas that are foundational to pervasive computing
- Remote communication, including protocol layering, remote procedure call, the use of timeouts, and the use of end-to-end arguments in placement of functionality.
 - Protocol layering is a common technique to simplify networking designs by dividing them into functional layers, and assigning protocols to perform each layer's task.
 - The principle states that, whenever possible communication protocol operations should be defined to occur at the end-points of a communications system, or as close as possible to the resource being controlled
- *Fault tolerance, including atomic transactions, distributed and nested transactions, and two-phase commit.*

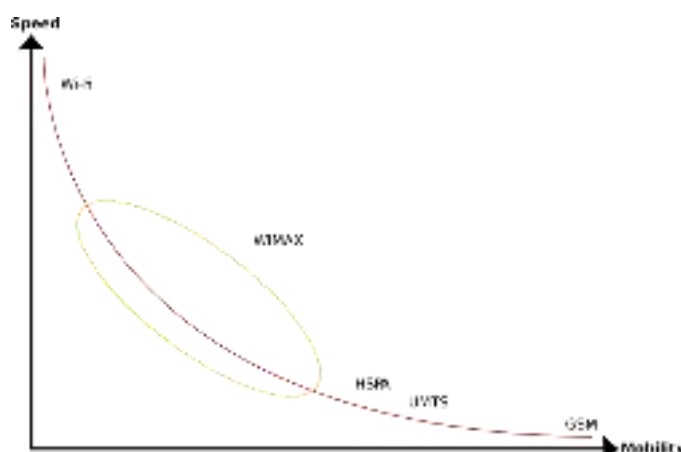
- In an **atomic transaction**, a series of database operations either *all* occur, or *nothing* occurs.
- a **nested transaction** occurs when a new transaction is started by an instruction that is already inside an existing transaction.
- that coordinates all the processes that participate in a [distributed atomic transaction](#) on whether to [commit](#) or *abort (roll back)* the transaction
- *High availability, including optimistic and pessimistic replica control, mirrored execution, and optimistic recovery.*
 - Traditional pessimistic replication systems try to guarantee from the beginning that all of the replicas are identical to each other, as if there was only a single copy of the data all along.
 - In optimistic replication replicas are guaranteed to converge under certain conditions
 - *Optimistic Recovery* is a technique supporting application-independent transparent recovery from processor failures in distributed systems
- *Remote information access, including caching, function shipping, distributed file systems, and distributed databases.*
 - This makes it possible for multiple users on multiple machines to share files and storage resources over network
- *Security, including encryption-based mutual authentication and privacy.*
 - *Mutual Authentication* is a security feature in which a client process must prove its identity to a server, and the server must prove its identity to the client, before any application traffic is sent over the client-to-server connection.

Lecture 3

Mobile Computing

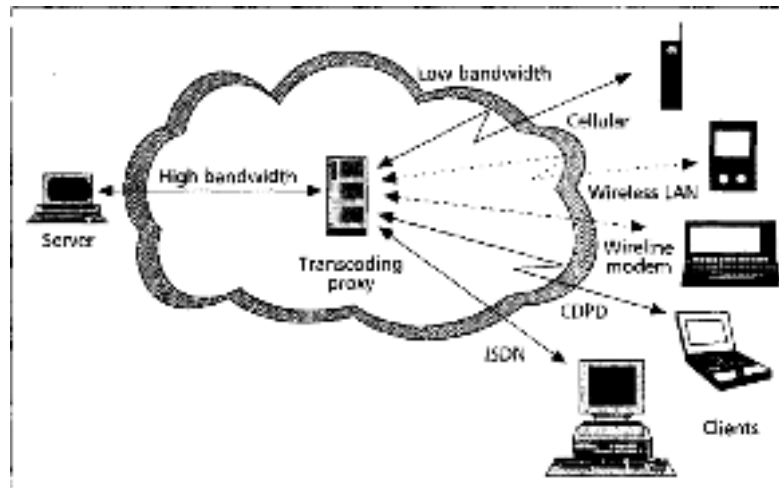
- The appearance of full-function laptop computers and wireless LANs in the early 1990s
- ... distributed system with mobile clients.
- Although many basic principles of distributed system design continued to apply, four key constraints of mobility forced the development of specialized techniques.
- unpredictable variation in network quality,
- lowered trust and robustness of mobile elements,
- limitations on local resources imposed by weight and size constraints, and
- concern for battery power consumption

Technology	Coverage	Speed
WiFi (802.11g)	38, 140 (m)	54Mbps
WiFi (802.11n)	70, 250 (m)	72.2 / 150 Mbps
WiMax	5-10 Miles	40 Mbps
UMTS		2Mbps
GSM		114kbps



- Mobile computing is still a very active and evolving field of research
- **Mobile networking**, including Mobile IP, IPv6, ad hoc protocols, and techniques for improving TCP performance in wireless networks
 - The IP address of a node consists of two portions network identifier (network ID) and a host identifier (host ID). The network ID specifies which network a host is on, and the host ID uniquely specifies hosts within a network.
- **Mobile information access**, including disconnected operation, bandwidth-adaptive file access, and selective control of data consistency
 - **Data consistency** summarizes the validity, accuracy, usability and integrity of related data between applications and across the IT enterprise
- *Support for **adaptive applications**, including transcoding by proxies and adaptive resource management*

- Application adaptation is a means by which applications can respond to a changing operating environment.
- Transcoding proxies are used as intermediaries between generic World Wide Web and a variety of client services in order to adapt to greatly varying bandwidths of different client communication links and handle the heterogeneity of possible small screen devices



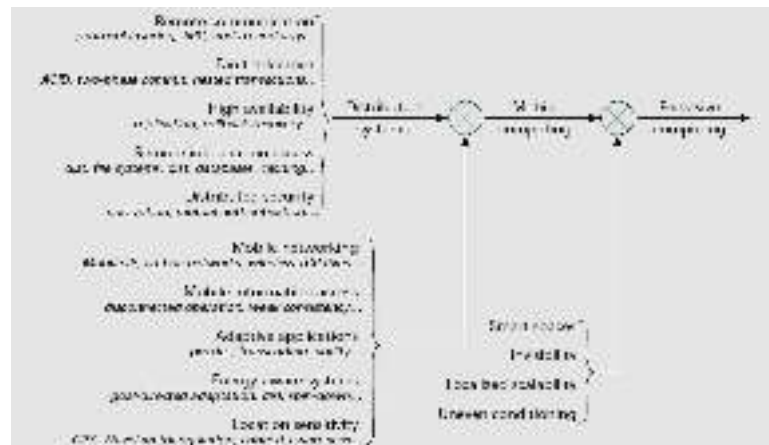
- System-level energy saving techniques, such as energy-aware adaptation, variable-speed processor scheduling, and energy-sensitive memory management
 - how applications can dynamically modify their behavior to conserve energy
- Location sensitivity, including location sensing and location-aware system behavior.

Lecture 4

Pervasive Computing

- One saturated with computing and communication capability, yet so gracefully integrated with users that it becomes a “technology that disappears.”
- The agenda of pervasive computing subsumes that of mobile computing, but goes much further
- **Effective Use of Smart Spaces:**
 - smart space brings together two worlds that have been disjoint until now
 - computing infrastructure
 - building infrastructure
- **Invisibility:**
 - The ideal is complete disappearance of pervasive computing technology from a user’s consciousness.
 - In practice, a reasonable approximation to this ideal is *minimal user distraction*.
 - *If a pervasive computing environment continuously* meets user expectations and rarely presents him with surprises, it allows him to interact almost at a subconscious level
- **Localized Scalability**
 - As smart spaces grow in sophistication, the intensity of interactions between a user’s personal computing space and his/her surroundings increases
 - This has severe band-width, energy, and distraction implications for a wireless mobile user. The presence of multiple users will further complicate this problem. Scalability, in the broadest sense, is thus a critical problem in pervasive computing.
 - Like the inverse square laws of nature, good system design has to achieve scalability by severely reducing interactions between distant entities.
- **Masking Uneven Conditions**
 - there exists huge differences in the “smartness” of different environments
 - The large dynamic range of “smartness” can be jarring to a user, detracting from the goal of making pervasive computing technology invisible

Composition of Pervasive Computing

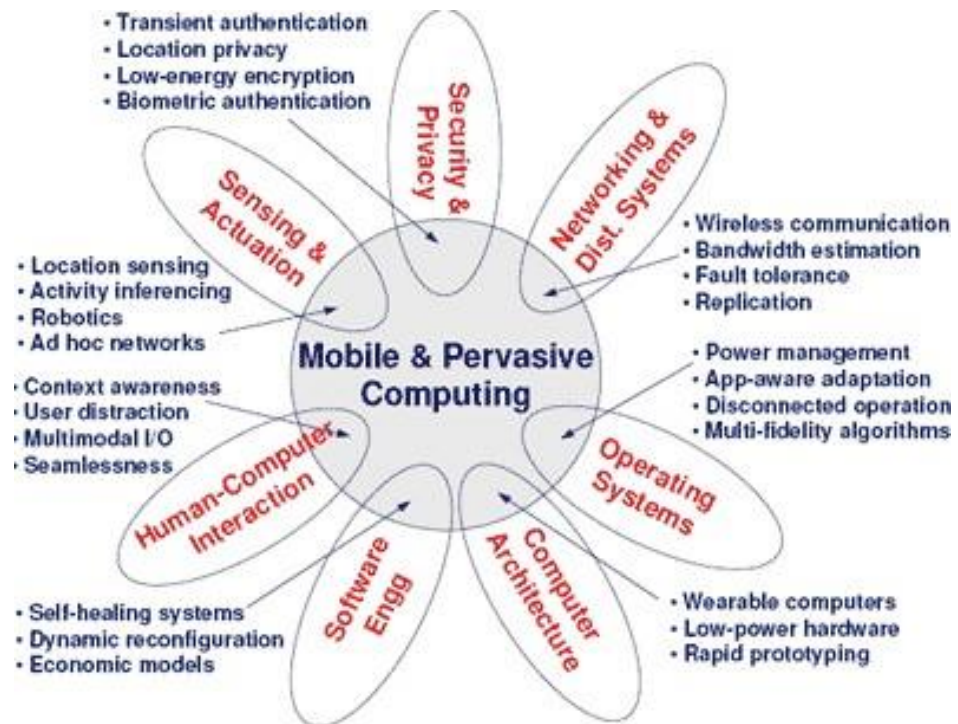


- Some of the technical problems in pervasive computing correspond to problems already identified and studied earlier in the evolution.
 - In some of those cases, existing solutions apply directly;
 - in other cases, the demands of pervasive computing are sufficiently different that new solutions have to be sought.
 - There are also new problems introduced by pervasive computing that have no obvious mapping to problems studied earlier.

Lecture 5

Composition of Pervasive Computing

- Some of the technical problems in pervasive computing correspond to problems already identified and studied earlier in the evolution.
 - In some of those cases, existing solutions apply directly;
 - in other cases, the demands of pervasive computing are sufficiently different that new solutions have to be sought.
 - There are also new problems introduced by pervasive computing that have no obvious mapping to problems studied earlier.
- **Sensing and Actuating**
 - Location Sensing
 - Activity Inference
 - Robotics
 - Ad-hoc Networks
- **Operating Systems**
 - Operating System of Small Things
 - Power Management
 - Application Aware Adaption
 - Disconnected Operation
- **Computer Architecture**
 - Wearable Computer
 - Low Power Hardware
 - Rapid Prototyping
- **Software Engineering**
 - Self-Healing Systems
 - Dynamic Reconfiguration
 - Economic Models
- **Security and Privacy**
 - Location Privacy
 - Low-power Encryption
 - Biometric Authentication
- **Human Computer Interaction**
 - Context Awareness
 - Seamlessness
 - Multimodal I/O



Pervasive Application Development

- This vision of pervasive computing leads to two fundamental characteristics of pervasive applications — mobility and context-awareness
- Both of these characteristics are a result of the extremely dynamic nature of pervasive computing environments
- Mobility has three implications. First, applications must run on a wide variety of devices, including the devices embedded in various environments and devices carried by users.
- Second, because devices may be transported to locations where a high-bandwidth network connection is not available, applications must work (perhaps in a degraded mode) with low-bandwidth network connections or in the absence of any network connection.
- Third, applications that make use of a user's location must account for the possibility that the location will change.

Lecture 6

Pervasive Application Development

- A context-aware application is one that is sensitive to the environment in which it is being used (e.g., the location or the particular user of the application).
- The application can use this information to customize itself to the particular location or the user. This implies the following technological requirements
- Identifying and binding to data sources that provide the right information
- Composing the information from these sources to create information that is useful for an application
- Using that information in meaningful ways within the application itself
- Identifying and binding to data sources that provide the right information
- Composing the information from these sources to create information that is useful for an application
- Using that information in meaningful ways within the application itself

Some Definitions

- A multi-device application is one that is able to execute on devices with different capabilities.
- A multimodal application is one that supports multiple user interface modalities such as GUI, voice, and a combination of the two.
- we consider only networked applications, because they represent the bulk of interesting and useful pervasive applications
- A device platform is the distributed software platform to which a pervasive application is targeted.
- A thin-client application is a networked application in which the user interface rendering component is executing on the user's device, whereas the rest of the application is executing on a networked computer.
- A thick-client application, on the other hand, has significant application components executing on the user's device.
- well-known MVC application structure
- The view represents the presentation, and the controller represents the application flow, including the navigation, validation, error handling, and event handling.
- The view and the controller together deal with the user interaction of the application.
- The model component includes the application logic as well as the data underlying the application logic.
- A disconnect able application is one that is able to continue to execute when there are different levels of connectivity between the different components of the application.
- The attributes of the environment of an application are referred to as the context of the application.
- The context of an application includes some of the user's significant attributes, such as location, destination, the identities of other people in the vicinity, and the attributes of the task being performed, such as the objective and the artifacts necessary for the task.
- A context-aware application is one that is able to sense some aspects of the environment in which it is executing and adapt its behavior to the sensed environment.

Pervasive Application Development

- There are fundamental reasons why pervasive application development is more difficult than conventional application development.
- End user devices, such as smart phones and PDAs, come in many varieties and have widely varying capabilities, both hardware (form factor, user interface hardware, processor, memory, and network bandwidth) and software (operating system, user interface software, services, and applications).

Lecture 7

Pervasive Application Development

- There are fundamental reasons why pervasive application development is more difficult than conventional application development.
- End user devices, such as smart phones and PDAs, come in many varieties and have widely varying capabilities, both hardware (form factor, user interface hardware, processor, memory, and network bandwidth) and software (operating system, user interface software, services, and applications).

Heterogeneity of Device Platforms

- The impact of device heterogeneity on application developers is that applications need to be developed (or ported) to each device and maintained separately for each device.
- **User Interface**
 - output capabilities, such as the screen characteristics (e.g., size and color)
 - the input capabilities, such as the number of hard buttons, rollers, and other controls;
 - and the software toolkit available to manipulate these input and output capabilities
- Because of differences in these capabilities from one device to another, the view component of an application will have to be rewritten for each device
- **Interaction Modalities**
 - significant method of user interaction
 - Examples are keyboard or mouse, speech, pen, and tactile interfaces
- view and controller portions of applications may need to be significantly rewritten to enable each modality
 - a speech based application could have a different structure from a GUI-based application
- Furthermore, *multimodal* interfaces can use multiple modalities within a single application
- **Platform Capabilities**
 - distributed software infrastructure on which an application executes
 - including the device software infrastructure and the server software infrastructure
 - the programming models on the device and the server are different
- An application may need to be partitioned differently between the device and the server depending on the processor, memory, and network capabilities of a device
- **Connectivity**
 - Execute in a dynamic environment that supports multiple levels of connectivity
 - worry about dynamically varying the partitioning of the application between the various connectivity scenarios
 - Resynchronizing partitioned components after reestablishing connectivity
- adds a significant amount of complexity

Dynamics of Application Environments

- Pervasive applications should be customized to the user and task at hand — also referred to as the context of the application.
- The context can be highly dynamic. The data sources that provide information about the application's environment are called context sources
- Consider the complexities of application development in the face of dynamic and heterogeneous context sources.
- The context data from different context sources could have different schemas and formats.
 - For example, location data from a cell tower is different from the location data from an IEEE 802.11 base station.
- If each pervasive application that uses context data were responsible for collecting and normalizing context data from different sources, applications would indeed be quite complex.
- The context information from any one source could be too low-level to be useful for an application.
- The actual context sources themselves could be highly dynamic. For example, the location of a person can be obtained by a multitude of sources, including a cell tower, a telematics gateway, a wireless local area network (LAN) hub, and an active-badge access point.
- Each of these sources of location may have a different API and may be more or less applicable to different locations. Applications should not be responsible for discovering these context sources and explicitly binding to them.

Approaches for Developing Pervasive Applications

- The basic problem of mobile application development to multiple devices, modalities, and connectivity environments is that of complexity, because the same application may have to be rewritten multiple times.

Lecture 8

Heterogeneity of Device Platforms

- The impact of device heterogeneity on application developers is that applications need to be developed (or ported) to each device and maintained separately for each device.

Dynamics of Application Environments

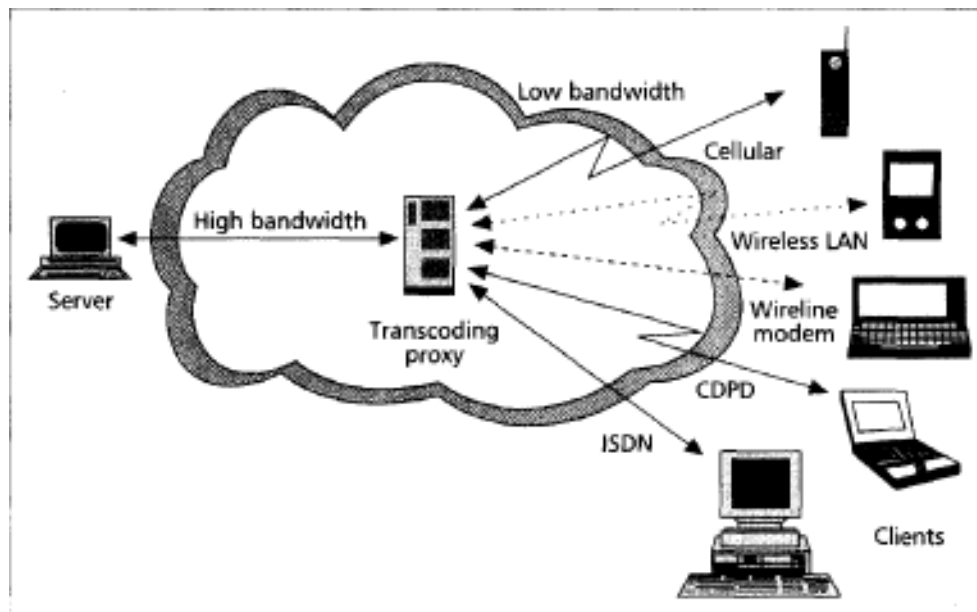
- Pervasive applications should be customized to the user and task at hand — also referred to as the context of the application.
- The context can be highly dynamic. The data sources that provide information about the application's environment are called context sources
- Consider the complexities of application development in the face of dynamic and heterogeneous context sources.
- The context data from different context sources could have different schemas and formats.
 - For example, location data from a cell tower is different from the location data from an IEEE 802.11 base station.
- If each pervasive application that uses context data were responsible for collecting and normalizing context data from different sources, applications would indeed be quite complex.
- The context information from any one source could be too low-level to be useful for an application.
- The actual context sources themselves could be highly dynamic. For example, the location of a person can be obtained by a multitude of sources, including a cell tower, a telematics gateway, a wireless local area network (LAN) hub, and an active-badge access point.
- Each of these sources of location may have a different API and may be more or less applicable to different locations. Applications should not be responsible for discovering these context sources and explicitly binding to them.
- The actual context sources themselves could be highly dynamic. For example, the location of a person can be obtained by a multitude of sources, including a cell tower, a telematics gateway, a wireless local area network (LAN) hub, and an active-badge access point.
- Each of these sources of location may have a different API and may be more or less applicable to different locations. Applications should not be responsible for discovering these context sources and explicitly binding to them.

Approaches for Developing Pervasive Applications

- The basic problem of mobile application development to multiple devices, modalities, and connectivity environments is that of complexity, because the same application may have to be rewritten multiple times.

Platform-Independent View Component

- **Presentation Transcoding**



Platform-Independent Controller Component

- The controller of an application represents the control flow, including data validation and error handling, typically via event handlers.
- To address the full range of applications, it is necessary to consider the role of the controller in modern interactive applications.
- There are several reasons why the controller of an application needs to be targeted to multiple devices

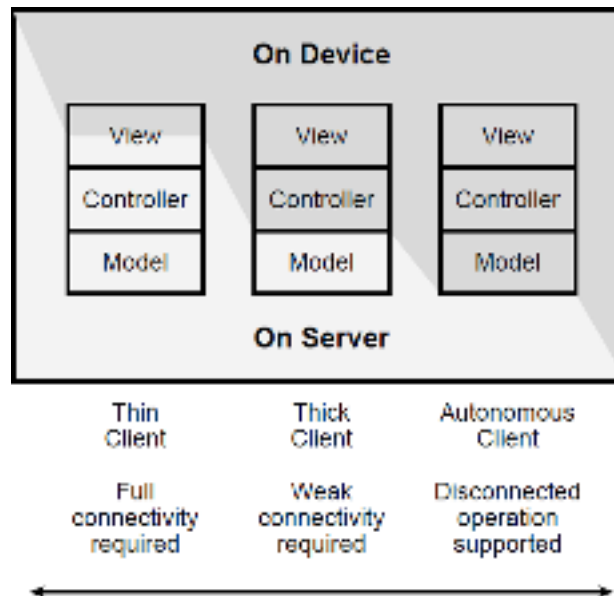
Host-Independent Model Component

- How to deal with the heterogeneity of connectivity environments
- Networked mobile applications vary in the distribution of logic and data between the mobile device and the server
- In a thin-client application, views are generated on the server and then rendered on the client device by a component such as a Web browser.
- Controller logic, model logic, and model data all reside on the server, so disconnected operation is impossible.
- In a thick-client application, the model still resides on a server, perhaps accessed through Web services, but the rest of the application resides on the client device.
- Caching of data before connection and queuing of updates to be performed upon reconnection enable limited forms of offline operation in a weakly connected environment.
- The operations allowed are those that can proceed sensibly in the absence of a complete and current model.
- An autonomous-client application resides entirely on the client device.
- It maintains its own fully functional model, which may be synchronized from time to time with replicas of the model on a server.

Lecture 9

Host-Independent Model Component

- An autonomous-client application resides entirely on the client device.
- It maintains its own fully functional model, which may be synchronized from time to time with replicas of the model on a server.



- It should be noted that there is a significant level of runtime infrastructure needed for disconnectable applications:
 - An application hosting and execution environment is needed on the mobile device.
 - If application code is to be downloaded from the server to clients upon demand, a code-migration component is needed on the server and device sides to coordinate the partitioning and loading of application components.
 - A data synchronization component is needed for updating both the device and server instances of the application with changes to the data on the other sites and to resolve any possible conflicts.

Developing Context-Aware Applications

- One can think of a context-aware application as having a triggering aspect and an effecting aspect
- The triggering aspect binds to data sources, collects data, analyzes the data, and ensures that the data is relevant to the application.
- If so, it notifies the effecting aspect, which takes the action corresponding to the trigger.
- Context-aware applications have three sources of complexity:
 1. The heterogeneous nature of data sources
 2. The dynamic nature of context sources
 3. The multiple sources of potentially low-level context data
- Observe that these are all in the triggering component of applications.

Source-Independent Context Data

- An application obtaining data from heterogeneous sources with inconsistent availability and quality of service should not name a specific source of data.
- Rather, it should describe the kind of data that is required, so that the underlying infrastructure can discover an appropriate source for the data.
- The basic idea of this approach is for an application to specify the desired context data without specifying the exact location and data type of the source, or whether it is coming from multiple sources.
- These are considerations that will be handled transparently by the infrastructure.
- In some cases, the infrastructure may discover a data source, such as a device or a Web service that directly provides the described data.
- Some data sources, such as request-response Web services, are passive or pull-based.
- Other data sources, such as sensors that trigger alarms, are active or push-based.
- Flexible infrastructure is capable of discovering both kinds of data sources.
- An application can then pull the current value from a passive data source or subscribe to be notified each time an active data source generates a new value.

Developing Pervasive Software

1. **Device-independent views** — these allow an application to capture the basic interaction structures that should be reused across multiple devices and modalities. They should be combined with the ability to fine-tune the presentation when necessary.
2. **Platform-independent controllers** — these allow an application to specify the overall control flow across multiple execution platforms, but still allow an application to have different control flow structures for different devices and uses.
3. **Host-independent models** — these allow an application to encapsulate the business logic and data in a manner that can be reused regardless of which host a component is instantiated on.
4. **Source-independent context data** — this allows an application to specify the intended context data to be supplied by reusable infrastructure components, which in turn are concerned with the specific data formats, locations, and combinations of physical data sources that provide the actual data.

Lecture 10

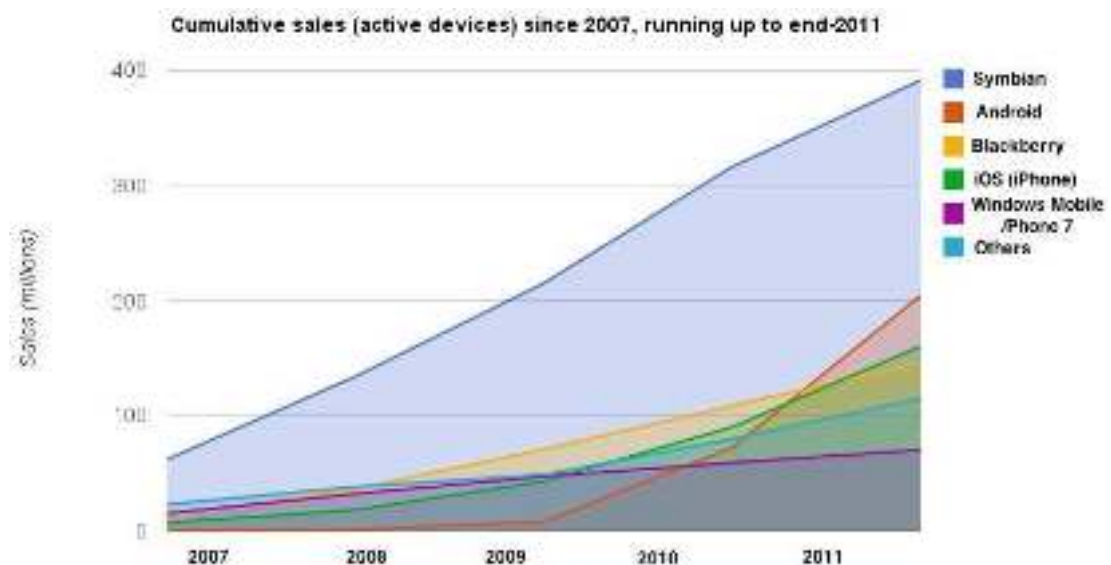
Hardware Platforms

- Projected Shipment
 - Cell Phones: Little above 1600M
 - Smart Phones: 400M
 - Laptops: 200M

Operating Systems for Smartphones

- Symbian
- Windows Mobile
- Blackberry OS
- Android
- iOS

Hardware Platforms



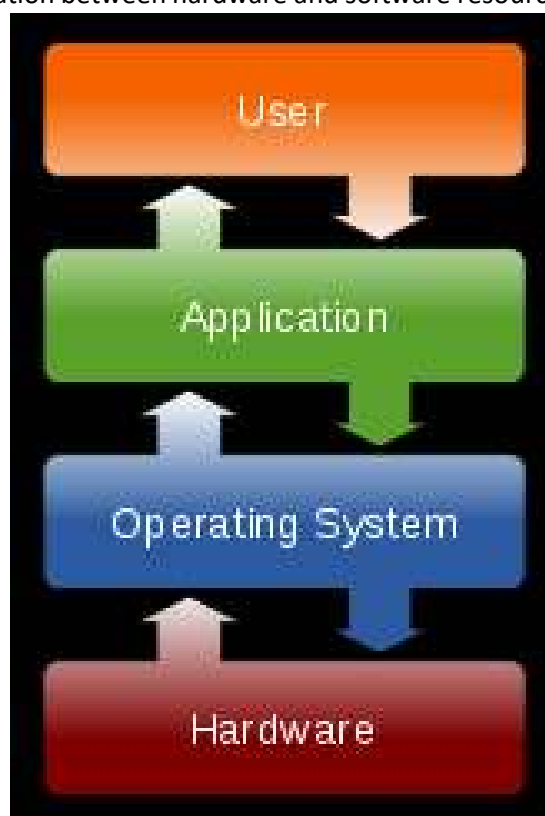
Symbian

- Developed by Symbian Ltd and runs exclusively on ARM processors
 - An unreleased x86 version exists
- Successor to Symbian OS and Nokia Series 60
- Designed for Smartphones
- Maintained by Nokia
- Open Source Operating System and Software Development Platform
- Embedded Operating Systems Family

- Symbian releases are styled **Symbian^1**, **Symbian^2**
 - **Symbian^1**, as the first release, forms the basis for the platform. It incorporates Symbian OS and S60 5th Edition (Symbian OS 9.4)
 - **Symbian^2** was the first royalty-free version of Symbian. On June 1, 2010, a number of Japanese companies including DoCoMo and Sharp announced smartphones using Symbian^2
 - **Symbian^3** was announced on 15 February 2010. It was designed to be a more 'next generation' smartphone platform. The Symbian^3 release introduced new features such as a new 2D and 3D graphics architecture, UI improvements, and support for external displays through HDMI.
 - **Symbian^4** was expected to be released in the first half of 2011. However, Nokia announced in October 2010 that Symbian^4 will not ship as a separate release. Instead, improvements to Symbian will be delivered as software updates to all current Symbian^3 devices

Crash Course on Operating Systems

- An operating system is a software, consisting of programs and data that runs on computers and manages computer hardware resources and provide common service for efficient execution of various application software
- For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between the application programs and the computer hardware
- Kernel is a bridge between applications and the actual data processing done at the hardware level. The kernel's responsibilities include managing the system's resources (communication between hardware and software resources)



- Application Programming Interfaces (APIs)
 - The functionality provided by the Windows API can be grouped into eight categories
 - Base Services
 - Advanced Services
 - Graphics Device Interface
 - User Interface
 - Common Dialog Box Library
 - Common Control Library
 - Windows Shell
 - Network Services
-
- Application Programming Interfaces (APIs)
 - Web
 - Multimedia
 - Program interaction
 - Wrapper Libraries

Symbian – Architecture

- System Model Structure - Technology domains and packages
 - The system model reflects the organization of a modular software stack
- **Layers** are the fundamental stacking of software
 - Each layer abstracts the ones below
- **Levels** within layers group packages by intended audience
- **Packages** are self-contained technologies
 - Levels within packages show their internal software stack
- **Collections** are logical groupings of related components
- **Components** are the atomic unit of software architecture

Symbian – Foundation Layers

- Three Symbian Foundation Device layers:
OS, Middleware and Applications
- Additional layers may be added with Architecture Council approval
- Vendors may add their own layers if needed
 - Vendor's layers can only contain non-contributed, vendor-specific packages



Foundation Device Layers – OS

- Provides APIs that abstract the hardware platform
- Allowing higher layers to be isolated from hardware changes
- Contains lower-level APIs that are not hardware abstractions, but are used within the OS layer
- Sufficient to develop a test platform
- Two levels:
 - hw: The kernel, interfaces to the hardware and user-side services
- This level is sufficient to be an operating system (filesystems, essential drivers, program execution model)
 - services: Essential services necessary for a phone
- Contains communications, text and data handling, graphics, etc

Foundation Device Layers – Middleware

- Provides higher-level generic APIs usable by programs in the application layer
- Includes native UI frameworks, application lifecycle, higher-level protocols and data handling, etc
- Middleware components are independent of the hardware platform
- MW APIs are not used by the OS layer
- **Two levels:**
 - **generic:** Services intended for any class of application
- **Examples:** Text entry, security and GUI widgets are expected to be used by most apps
 - This level and below is a general purpose computer
- **specific:** Services intended for a *specific* class of application
- **Examples:** Presence is mainly used by instant messaging apps, the Phone Server is expected to only be used by a Phone application
 - This level and below is sufficient for phone application development

Foundation Device Layers – Applications

- Primarily contains interactive UI applications
- Includes non-interactive applications (daemons) that respond to events from other than the device user (e.g. from a PC).
- **Two levels:**
 - **services:** Non-interactive applications, services provided to other packages
 - This level and below is sufficient to create applications which work together
 - **apps:** Applications which interact with the user
 - This level and below is the full set of software on the device
- Packages which span both levels provide user-facing applications and services to other applications.

- **Example:** Contacts Apps provides the user-facing Phonebook application, plus the Contact Model which has an API for programmatically accessing contacts

Lecture 11

Symbian Design Patterns

- The microkernel pattern
- The client–server pattern
- Frameworks
- The graphical application model
- An event-based application model
- Specific idioms aimed at improving robustness
- Streams and stores for persistent data storage
- The class library

Symbian – Architecture

- System Model Structure - Technology domains and packages
- The system model reflects the organization of a modular software stack
- **Layers** are the fundamental stacking of software
 - Each layer abstracts the ones below
- **Levels** within layers group packages by intended audience
- **Packages** are self-contained technologies
 - Levels within packages show their internal software stack
- **Collections** are logical groupings of related components
- **Components** are the atomic unit of software architecture

Symbian – Foundation Layers

- Three Symbian Foundation Device layers:
OS, Middleware and Applications
- Additional layers may be added with Architecture Council approval
- Vendors may add their own layers if needed
 - Vendor's layers can only contain non-contributed, vendor-specific packages

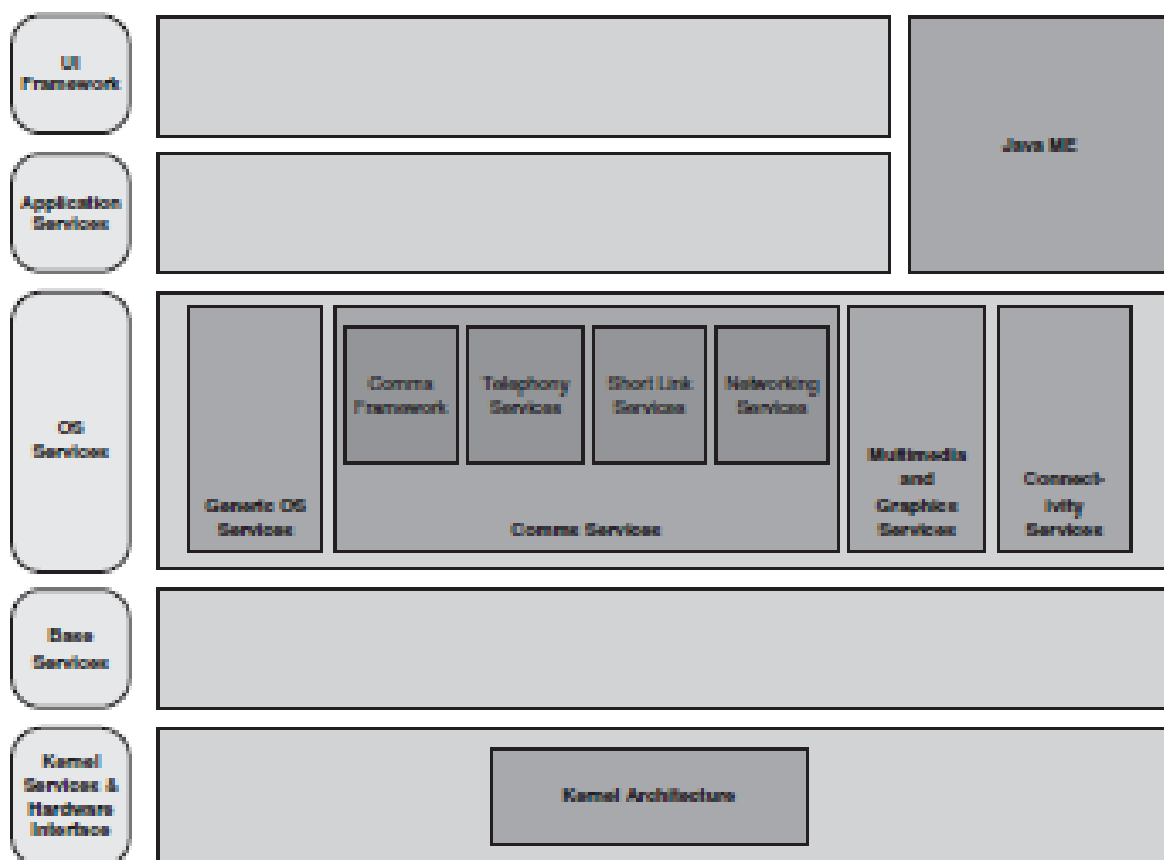
Symbian OS Model

- **UI Framework Layer**
 - UI Framework layer provides the frameworks and libraries for constructing a user interface
- **The Application Services Layer**
 - **System-level services** Text Handling
 - **Services that support generic types of application**, Alarm Server, data synchronization services, Printing Support
 - **Services based on more generic but application-centric technologies**, mail, messaging and browsing

- **The OS Services Layer**
 - generic operating system services
 - communications services
 - multimedia and graphics services
 - connectivity services.
- The Base Services Layer
- The Kernel Services and Hardware Interface Layer

Symbian – Architecture

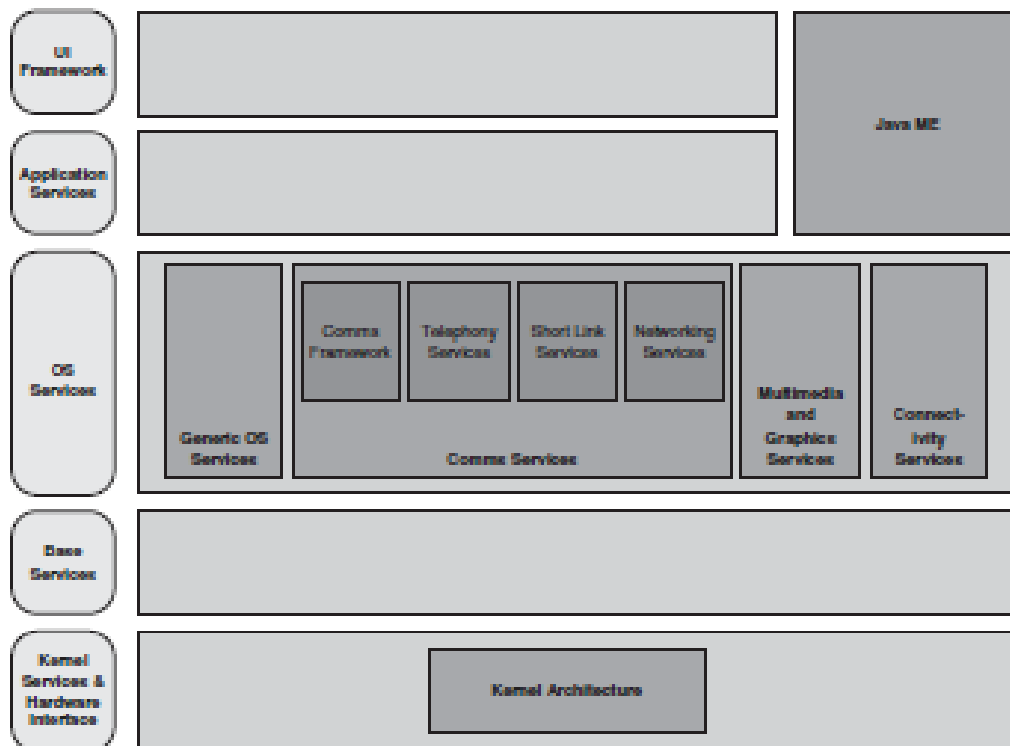
- All the services provided by a layer are at a similar level of abstraction
- A layer provides services to higher layers ('upwards')
- A layer delegates tasks to lower layers ('downwards')
- Dependencies flow consistently from higher layers to lower layers (but dependencies are allowed sideways within layers)
- Requests travel downwards
- Notifications travel upwards
- Higher layers abstract the services of lower layers away from machine centric services towards user-visible functionality
- A layer provides services as far as possible via well-defined external interfaces, which can be separated from the internal interfaces available within the layer



Lecture 12

Symbian – Architecture

- System Model Structure - Technology domains and packages
 - The system model reflects the organization of a modular software stack
- **Layers** are the fundamental stacking of software
 - Each layer abstracts the ones below
- **Levels** within layers group packages by intended audience
- **Packages** are self-contained technologies
 - Levels within packages show their internal software stack
- **Collections** are logical groupings of related components
- **Components** are the atomic unit of software architecture



Symbian – Components

- The smallest architectural entity of the system.
- A component is an implementation unit that provides a discrete, re-usable piece of the system. packages.
 - Build and packaging data
 - binaries, data, tests, documentation and source code
 - Informative data
- Information provided for documentation or analysis purposes

- Age
- What devices this can should or must appear it
- Intended target (ie device or desktop)
- Class of contents (ie, what is in it)

Symbian – Collections

- A coherent set of collaborating components which together deliver a complete, discrete, and identifiable part of the system functionality
- Components in a collection will generally be strongly coupled or implement the same interface (e.g. a family of plugins)
- A component with no strong ties in the package can live alone in its collection
- Collections can group components to reflect ...
 - the software or communication stack
 - a sub-technology of functionality
- Collections are stacked in levels
 - The primary package exposed interfaces are on the top
 - The interface to lower layers or hardware are on bottom
 - Middle levels tend to reflect dependencies and technology-specific relationships

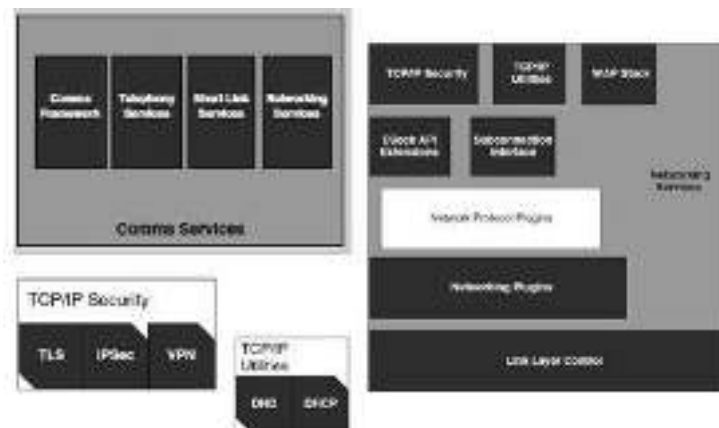
Symbian Design Patterns

- The microkernel pattern
- The client–server pattern
- Frameworks
- The graphical application model
- An event-based application model
- Specific idioms aimed at improving robustness
- Streams and stores for persistent data storage
- The class library

Symbian – Architecture

- Comm Services

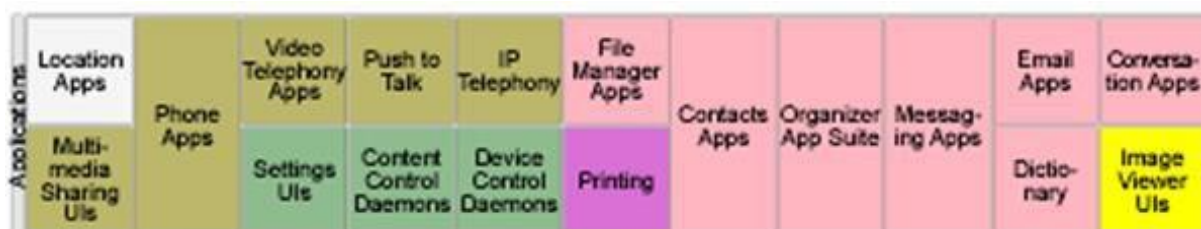


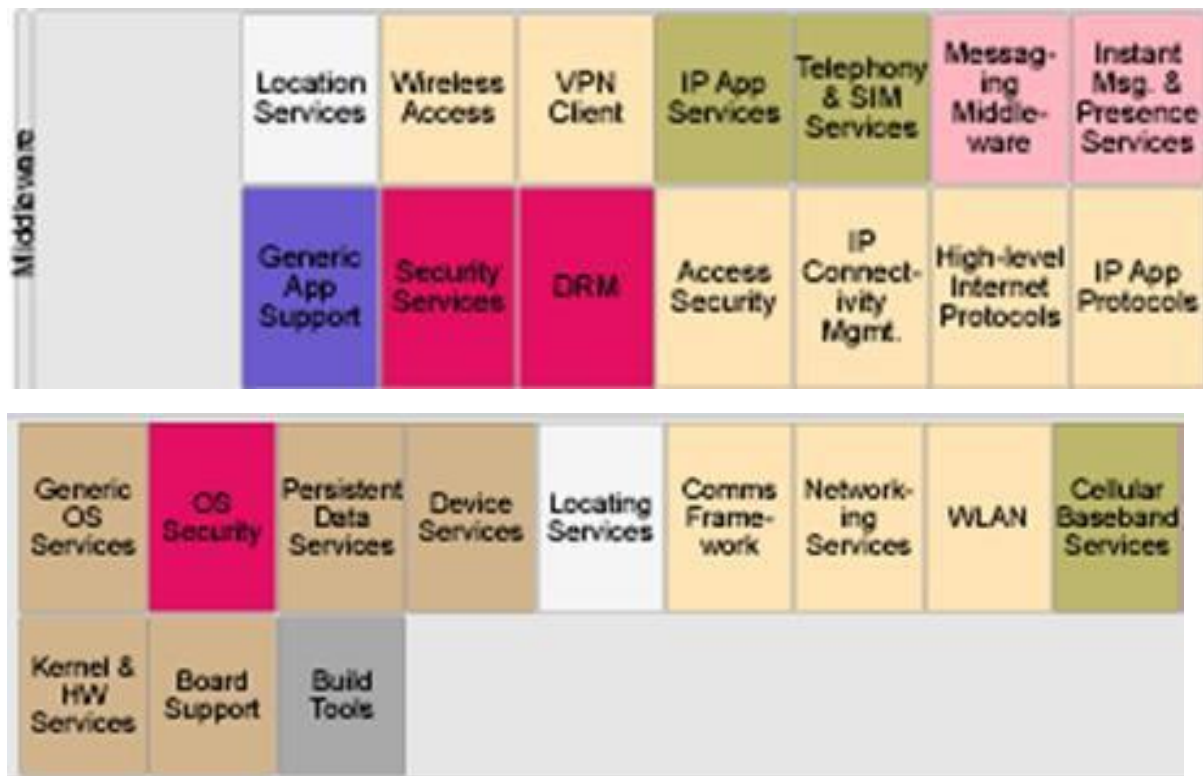


- What about package ?
- OS Security package: provides cryptography services to the layers above including applications written on the OS.
- The contentmgmt collection
- crypto collection
- cryptomgmtlibs collection
- cryptoservices collection
- securityanddataprivacytools

Symbian – Packages

- A modular group of component collections that is sufficient to develop, build, and test a technology
- A high-level application or application suite, e.g. Phone Apps, Messaging Apps
- Services with a common or interrelated code base, e.g. Security Services, Input Methods
- A core technology, e.g. Graphics, Networking Services
- Packages are the basic unit of Symbian Foundation organisational control
- Package scope and model location is agreed with the Foundation Architecture council
- Packages are owned and maintained by a package owner (PkO)
- PkO has overall responsibility for the package and its contents
- Contributions also come from the wider Symbian Foundation community
- A package belongs to a single technology domain – a vertical grouping of related packages





- Packages split or combine over time
- Generic technologies split when they gain sufficient “mass” over time to warrant their own package.
- New technologies tend to start development in the generic packages: Generic App Support, Generic OS Services.
- This avoids the overhead of creating a package for a technology with an unclear future
- **Example:** Device Services split out of Generic OS Services
- Subtechnologies split when large or sufficiently self-contained enough to be developed separately
- **Example:** Shortlink Services split into Bluetooth and USB

Symbian – Architecture

• Design

- Symbian OS was created with three systems design principles in mind:
 - the integrity and security of user data is paramount,
 - user time must not be wasted, and
 - all resources are scarce.
- features pre-emptive multitasking and memory protection
- follow an object-oriented design: Model-View-Controller
- strong emphasis on conserving resources, cleanup stacks, disk space, low power modes etc
- Event-based

Symbian – Kernel

- Bootstrapping the physical or emulated device to provide the basic initialization of the hardware
- Creating and managing the fundamental operating system kernel abstractions, for example, threads, processes, memory address spaces, and other resources including timers, mutexes, and so on
- Scheduling, pre-emption and interrupt handling
- Access to devices, providing the device-driver framework and device drivers that abstract device hardware and implement the two-tier logical and physical device driver model
- Encapsulating the kernel–user boundary; all processes which run in privileged mode originate from this layer
- Encapsulating the lowest level of an operating system port ('base port') to new hardware insulating all higher layers from actual hardware.
- **EKA2** (EPOC Kernel Architecture 2) is the second-generation Symbian platform Kernel
- Like its predecessor it has pre-emptive multithreading and full memory protection. The main differences are:
 - **Real-Time guarantees** (each API call is quick, but more importantly, time-bound)
 - **Multiple threads** inside the kernel as well as outside
 - **Pluggable memory models**, allowing better support for later generations of ARM Instruction set
- A "nanokernel" which provides the most basic OS facilities upon which other "personality layers" can be built
- Symbian kernel (EAK2) supports sufficiently-fast real time response
- single processor core executes both the user application and the signaling stack
- microkernel architecture that contains only the minimum, most basic primitives and functionality
- for maximum robustness, availability and responsiveness

Symbian - Features

- **Application Development**
 - From 2010, Symbian switched to using standard C++ with Qt as the SDK, which can be used with either Qt creator or Carbide.
 - Qt supports the older Symbian S60 3rd and 5th editions, as well as the new Symbian platform.
 - Alternative application development can be done with using Python (Python for S60), Adobe Flash or JavaME
 - Symbian OS previously used a Symbian specific C++ version along with Carbide.c++ Integrated development environment as the native application development environment.

Symbian – Devices

- As of 21 July 2009, more than 250 million devices running Symbian OS had been shipped
- In addition to Nokia , Sony Ericsson and Samsung have used Symbian OS.



Symbian – Packages

- Package's internal software stack described with levels
 - Lowest provides interface to hardware or layers below
 - Highest provides, UIs, APIs, metadata and documentation
 - Mid-levels tend to be frameworks, engines, servers, plugins, internal tools, etc
- Meaning differs between layers
- By convention, packages in the same layer and level have roughly the same number of internal levels
 - OS Layer, hw level: ≤ 5 levels. Bottom level reserved for development boards.
 - OS Layer, services level: ≤ 6 levels. Bottom level reserved for device drivers
 - Middleware Layer, generic level: ≤ 6 levels
 - Middleware Layer, specific level: ≤ 5 levels. UI, if any, on top level
 - App Layer, services level: ≤ 4 levels
 - App Layer, apps level: ≤ 4 levels.
- Spanning packages ≤ 8 levels
- Applications and UIs on top level, frameworks and engines below, plugins and other support on bottom (guidelines, not binding)

Symbian – Architecture

- Operating system
- The All over Model contains the following layers, from top to bottom:
- UI Framework Layer
- Application Services Layer
- J2ME
- OS Services Layer
 - generic OS services
 - communications services
 - multimedia and graphics services
 - connectivity services
- Base Services Layer
- Kernel Services & Hardware Interface Layer

Lecture 13

Android Linux Kernel

- Great memory and process management
- Permissions-based security model
- Proven driver model
- Support for shared libraries
- It's already open source!

Android Linux Kernel Enhancement

- Alarm
- Ashmem
- Binder
- Power Management
- Low Memory Killer
- Kernel Debugger
- Logger

Android Kernel – Binder

- Applications and Services may run in separate processes but must communicate and share data
- IPC can introduce significant processing overhead and security holes
- Driver to facilitate inter-process communication (IPC)
- High performance through shared memory
- Per-process thread pool for processing requests
- Reference counting, and mapping of object references across processes
- Synchronous calls between processes

Android Kernel – Power Management

- Mobile devices run on battery power
- Batteries have limited capacity
- Built on top of standard Linux Power Management (PM)
- More aggressive power management policy
- Components make requests to keep the power on through “wake locks”
- Supports different types of wake locks

Android Native Libraries

- Bionic Libc
- Function Libraries
- Native Servers
- Hardware Abstraction Libraries

Android Native Libraries – libc

- Bionic libc is Custom c run time implementation, optimized for embedded use.
- **License:** keep GPL out of user-space
- **Size:** will load in each process, so it needs to be small
- **Fast:** limited CPU power means we need to be fast

Android Native Libraries – Function Libraries

- These are libraries that do most of the heavy lifting
- Providing a lot of power behind Android platform
- Abstracted by the higher level API seen in application framework.
- **WebKit** – Browser Engine
- **Media Framework** - built on top of a set of media libraries, including OpenCore
- **SQLite** – Open Source Relational Database

Android Native Servers

- Surface Flinger
- Audio Flinger

Surface Flinger

- Provides system-wide surface “composer”, handling all surface rendering to frame buffer device
- Can combine 2D and 3D surfaces and surfaces from multiple applications Libraries
- Surfaces passed as buffers via Binder IPC calls
- Can use OpenGL ES and 2D hardware accelerator for its compositions



Audio Flinger

- Manages all audio output devices
- Processes multiple audio streams into PCM audio out paths
- Handles audio routing to various outputs



Hardware Abstraction Libraries

- Native Libraries to provide a better abstraction between the hardware and upper OS layers
- User space C/C++ library layer
- Defines the interface that Android requires hardware “drivers” to implement
- Separates the Android platform logic from the hardware interface
- Why do we need a user-space HAL?
 - Not all components have standardized kernel driver interfaces
 - Kernel drivers are GPL which exposes any proprietary IP
 - Android has specific requirements for hardware drivers

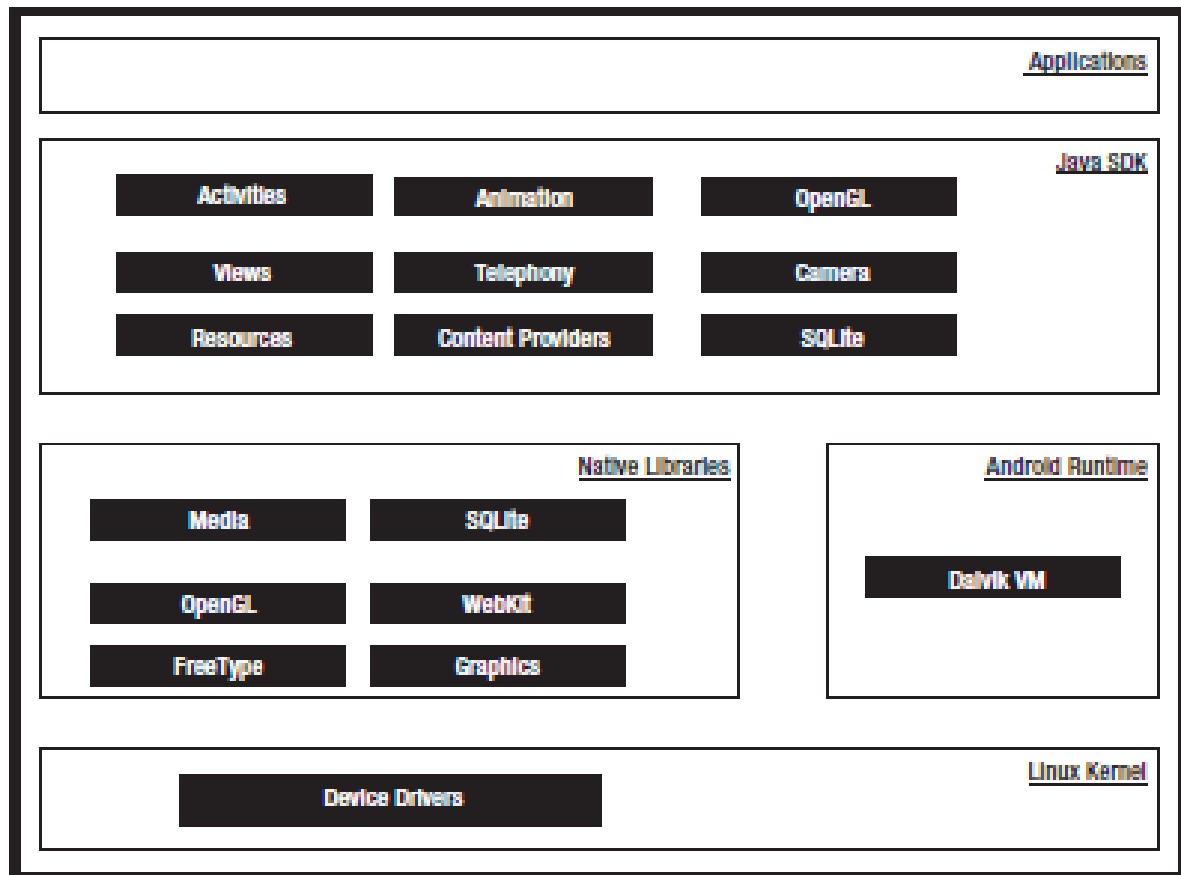
Android Runtime

- Composed of two elements
 - Core Libraries
 - Dalvik Virtual Machine (DVM)
- Android’s custom clean-room implementation virtual Machine
- Provides application portability and runtime consistency
- Runs optimized file format (.dex) and Dalvik bytecode
- Java .class / .jar files converted to .dex at build time

Android Virtual Machine (DVM)

- Performance requirements on handsets are severe
- Packages in Android, are full-featured and extensive
- These system libraries might use as much as 10 to 20MB (even with the optimized JVM)
 - Welcome to Dalvik JVM :It reuses duplicate information from multiple class files, effectively reducing the space requirement (uncompressed) by half from a traditional .jar file
 - Many of Android’s core libraries, including the graphics libraries, are implemented in C and C++
 - Fine-tuned the garbage collection
 - Dalvik VM uses a different kind of assembly-code generation use of registers rather than stack
- Used to write all the classes or core system services
- Services that are essential to the Android platform
- Behind the scenes - applications typically don’t access them directly

Android Software Stack



Lecture 14

Android Kernel – Binder

- Applications and Services may run in separate processes but must communicate and share data
- IPC can introduce significant processing overhead and security holes
- Driver to facilitate inter-process communication (IPC)
- High performance through shared memory
- Per-process thread pool for processing requests
- Reference counting, and mapping of object references across processes
- Synchronous calls between processes

Android Kernel – Power Management

- Mobile devices run on battery power
- Batteries have limited capacity
- Built on top of standard Linux Power Management (PM)
- More aggressive power management policy
- Components make requests to keep the power on through “wake locks”
- Supports different types of wake locks

Android Native Libraries

- Bionic Libc
- Function Libraries
- Native Servers
- Hardware Abstraction Libraries

Android Native Libraries – libc

- Bionic libc is Custom c run time implementation, optimized for embedded use.
- **License:** keep GPL out of user-space
- **Size:** will load in each process, so it needs to be small
- **Fast:** limited CPU power means we need to be fast

Lecture 15

Android Native Libraries – libc

- Bionic libc is Custom c run time implementation, optimized for embedded use.
- **License:** keep GPL out of user-space
- **Size:** will load in each process, so it needs to be small
- **Fast:** limited CPU power means we need to be fast

Android Native Libraries – Function Libraries

- These are libraries that do most of the heavy lifting
- Providing a lot of power behind Android platform
- Abstracted by the higher level API seen in application framework.
- **WebKit** – Browser Engine
- **Media Framework** - built on top of a set of media libraries, including OpenCore
- **SQLite** – Open Source Relational Database

Android Native Servers

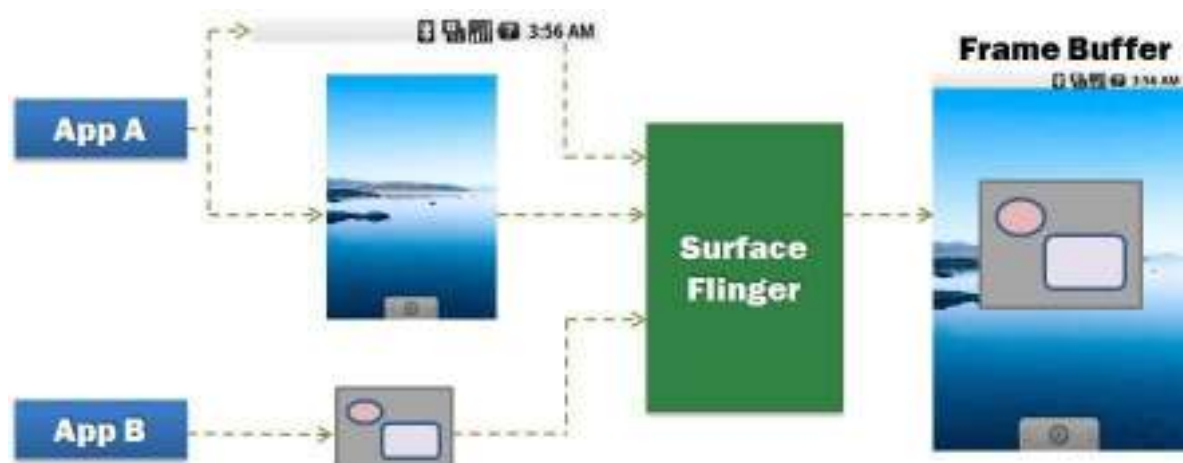
- Surface Flinger
- Audio Flinger

Surface Manager/Flinger

- Provides system-wide surface “composer”, handling all surface rendering to frame buffer device
- Can combine 2D and 3D surfaces and surfaces from multiple applications Libraries
- Surfaces passed as buffers via Binder IPC calls
- Can use OpenGL ES and 2D hardware accelerator for its compositions



Surface Manager/Flinger



Audio Manager/Flinger

- Manages all audio output devices
- Processes multiple audio streams into PCM audio out paths
- Handles audio routing to various outputs



Hardware Abstraction Libraries

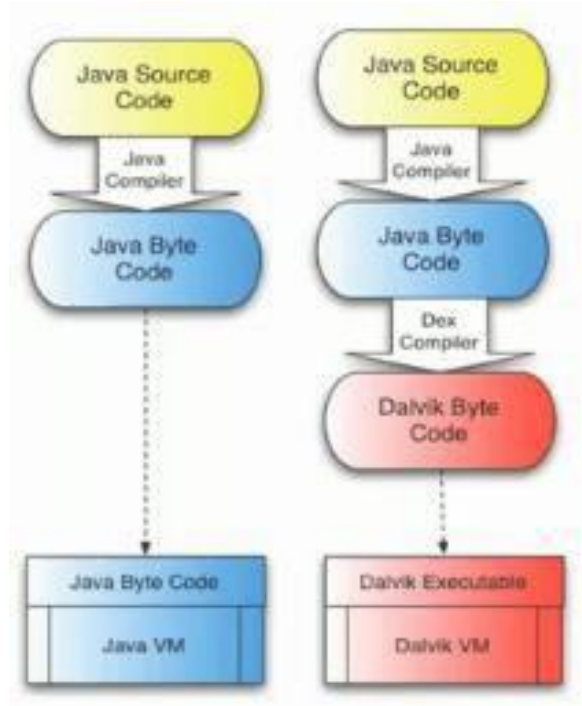
- Native Libraries to provide a better abstraction between the hardware and upper OS layers
- User space C/C++ library layer
- Defines the interface that Android requires hardware “drivers” to implement
- Separates the Android platform logic from the hardware interface

Android Runtime – Dalvik

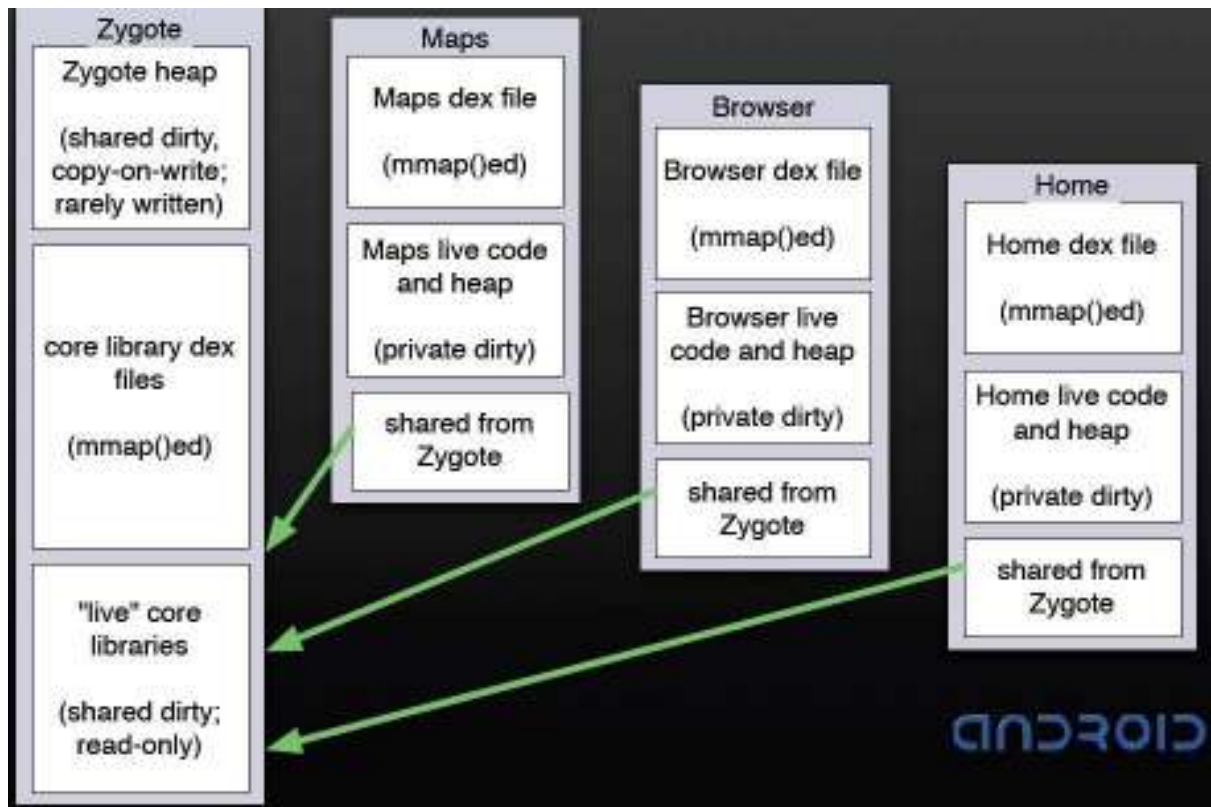
- Android’s custom clean-room implementation virtual Machine
- Provides application portability and runtime consistency
- Runs optimized file format (.dex) and Dalvik bytecode
- Java .class / .jar files converted to .dex at build time

Lecture 16

Android Runtime - Dalvik



- 4 Kinds of Memory
- clean vs. dirty
 - clean: mmap()ed and unwritten
 - dirty: malloc()ed
- shared vs. private
 - shared: used by many processes
 - private: used by only one process
- clean (shared or private)
 - common dex files (libraries)
 - application-specific dex files
- shared dirty
 - ???
- private dirty
 - application “live” dex structures
 - application heap



- CPU speed: 250-500MHz
- bus speed: 100MHz
- data cache: 16-32K
- available RAM for apps: 20 MB
- No JIT (Just In Time) Compiler
 - Omitted at first, Back in release 2.2
- Install Time Work
 - Verification
 - valid indices
 - valid offsets
 - Code Can't misbehave
- Optimization
 - byte-swapping and padding (unnecessary on ARM)
 - static linking
 - "inlining" special native methods
 - pruning empty methods
 - adding auxiliary data

- Register Machine
 - avoid instruction dispatch
 - avoid unnecessary memory access
 - consume instruction stream efficiently
 - higher semantic density per instruction
- Core APIs for Java language provide a powerful, yet simple and familiar development platform
 - Data structures
 - Utilities
 - File access
 - Network Access
 - Graphics
 - ...

Application Framework

- Written in Java
- Services that are essential to the Android platform
- Behind the scenes - applications typically don't access them directly Application

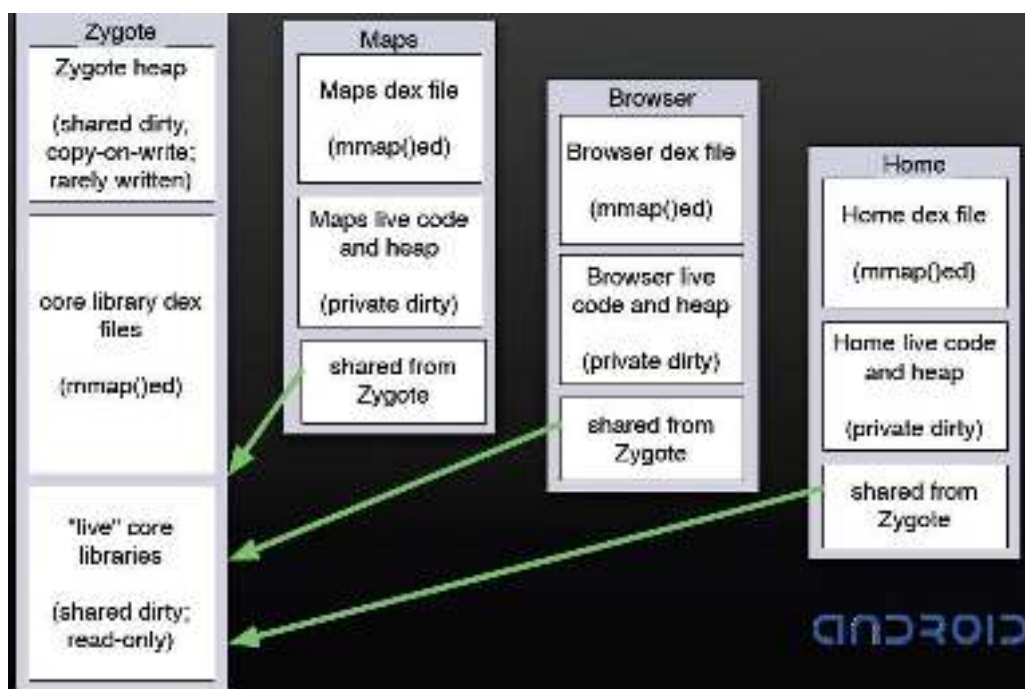
Core Application Services

- Activity Manager
- Package Manager
- Window Manager
- Resource Manager
- Content Providers
- View System

Lecture 17

Android Runtime – Dalvik

- clean (shared or private)
 - common dex files (libraries)
 - application-specific dex files
- shared dirty
 - ???
- private dirty
 - application “live” dex structures
 - application heap



- No JIT (Just In Time) Compiler
 - Omitted at first, Back in release 2.2
- Install Time Work
 - Verification
 - valid indices
 - valid offsets
 - Code Can't misbehave
- Optimization
 - byte-swapping and padding (unnecessary on ARM)
 - static linking
 - “inlining” special native methods
 - pruning empty methods
 - adding auxiliary data

- Register Machine
 - avoid instruction dispatch
 - avoid unnecessary memory access
 - consume instruction stream efficiently
 - higher semantic density per instruction
- Core APIs for Java language provide a powerful, yet simple and familiar development platform
 - Data structures
 - Utilities
 - File access
 - Network Access
 - Graphics
 - ...

Lecture 18

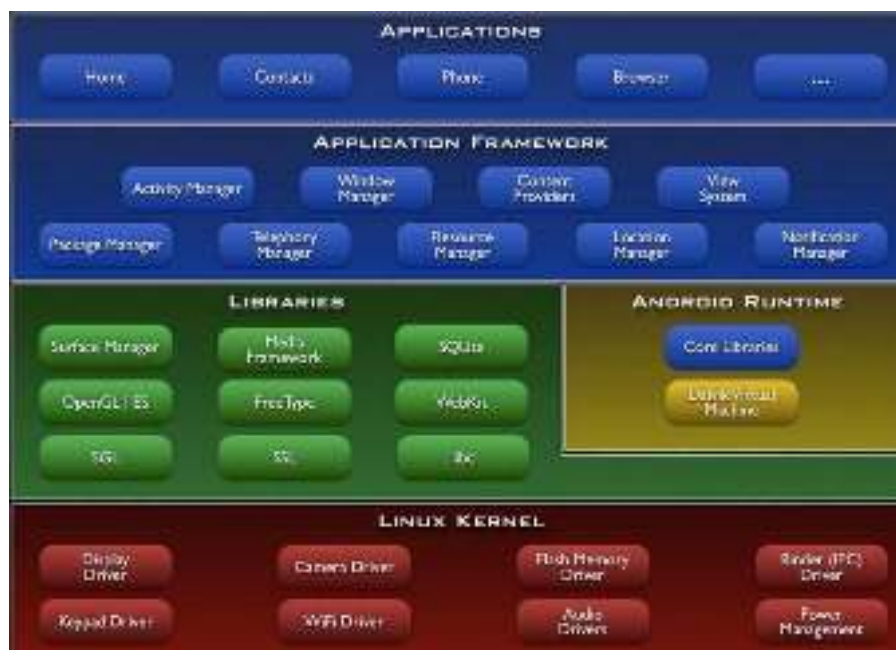
Application Framework

- Written in Java
- Services that are essential to the Android platform
- Behind the scenes - applications typically don't access them directly

Core Application Services

- Activity Manager
- Package Manager
- Window Manager
- Resource Manager
- Content Providers
- View System
- Hardware Services
 - Provide access to lower-level hardware APIs
 - Typically accessed through local *Manager* object
- Telephony Service
- Location Service
- Bluetooth Service
- WiFi Service
- USB Service
- Sensor Service

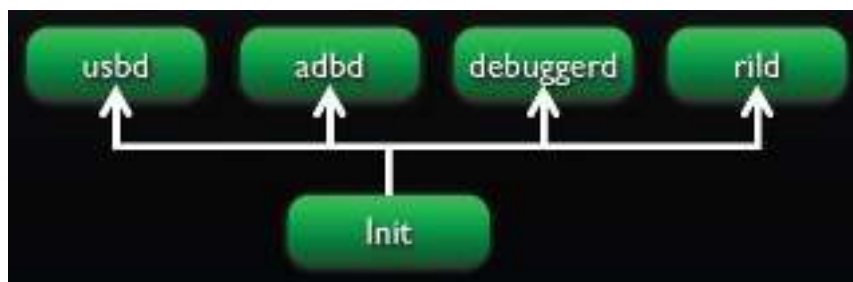
Applications



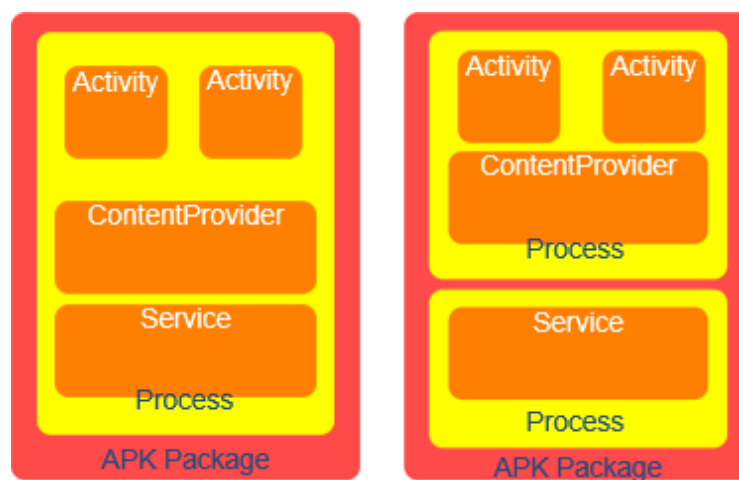
Runtime Walkthrough

- **System Startup**
 - Bootloader
 - Kernel
 - Init
 - Zygote
 - System Server
 - Activity Manager
 - Launcher (Home)

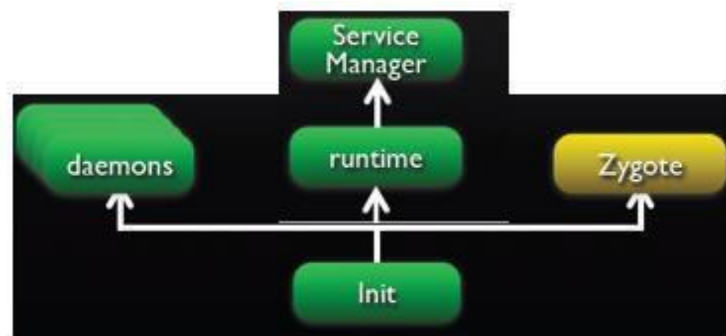
Runtime Walkthrough – Zygote



Package, Service and Process



Runtime Walkthrough – Run Time Process



Lecture 19

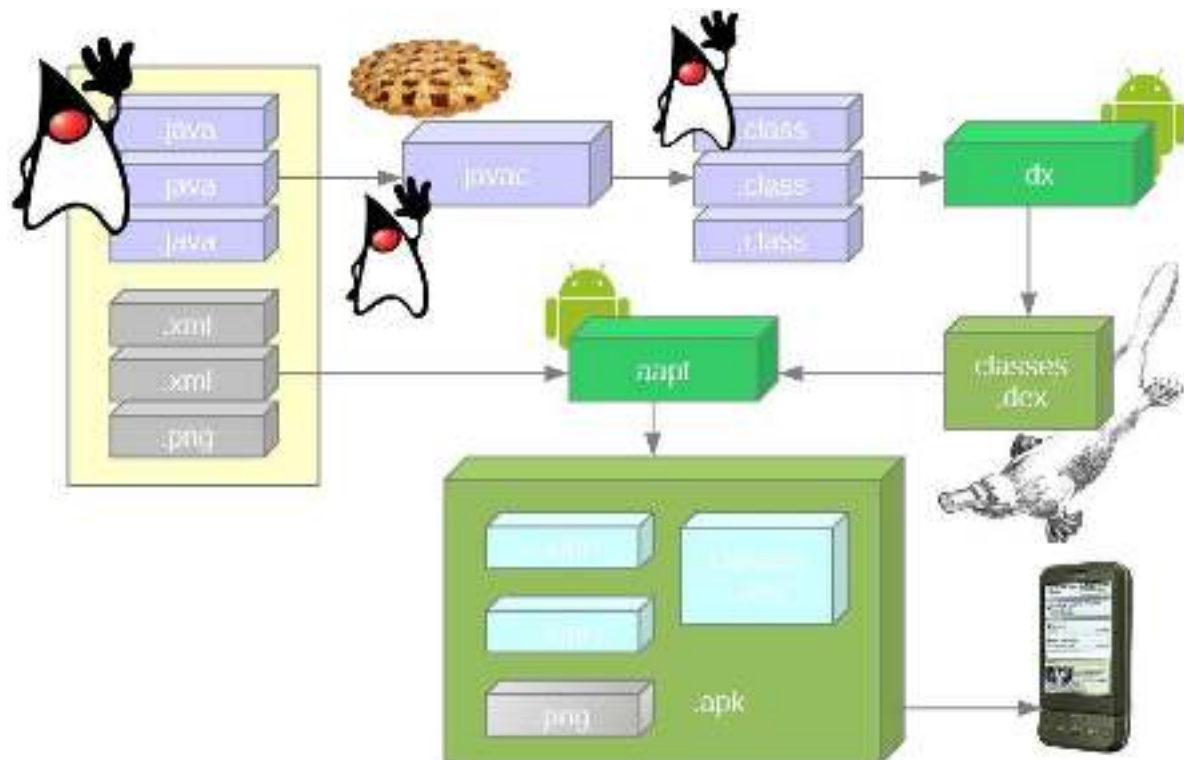
Core Application Services

- Activity Manager
- Package Manager
- Window Manager
- Resource Manager
- Content Providers
- View System
- Hardware Services
 - Provide access to lower-level hardware APIs
 - Typically accessed through local *Manager* object
- Telephony Service
- Location Service
- Bluetooth Service
- WiFi Service
- USB Service
- Sensor Service

Applications



Recap of Lecture 18

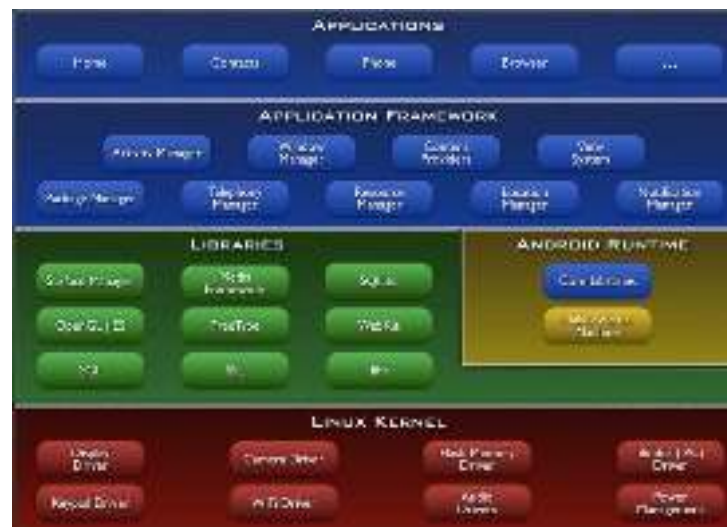


Lecture 20

Core Application Services

- Activity Manager
- Package Manager
- Window Manager
- Resource Manager
- Content Providers
- View System
- Hardware Services
 - Provide access to lower-level hardware APIs
 - Typically accessed through local *Manager* object
- Telephony Service
- Location Service
- Bluetooth Service
- WiFi Service
- USB Service
- Sensor Service

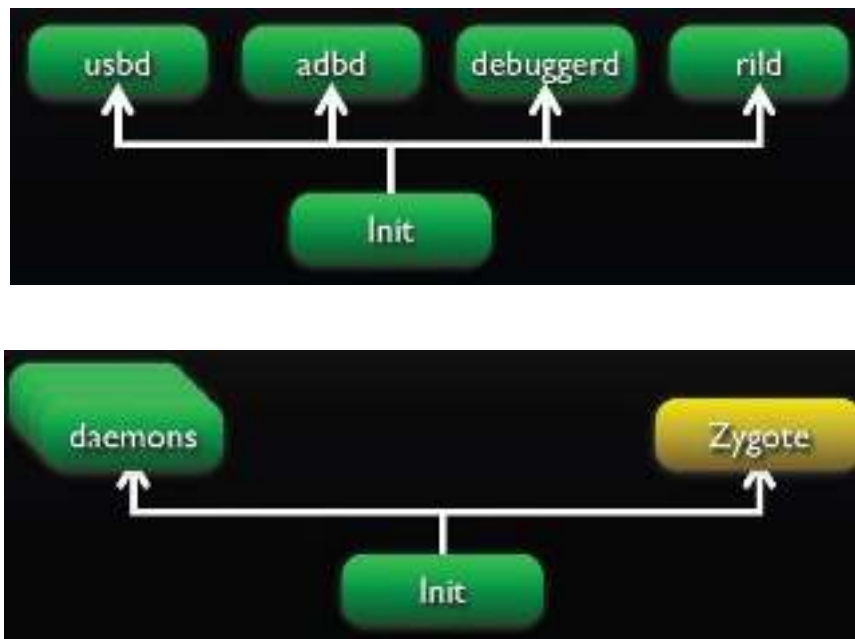
Applications



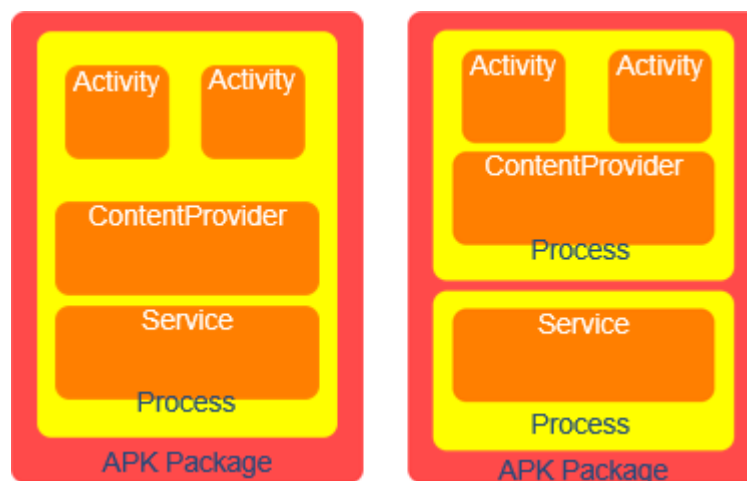
Runtime Walkthrough

- **System Startup**
 - Bootloader
 - Kernel
 - Init
 - Zygote
 - System Server
 - Activity Manager
 - Launcher (Home)

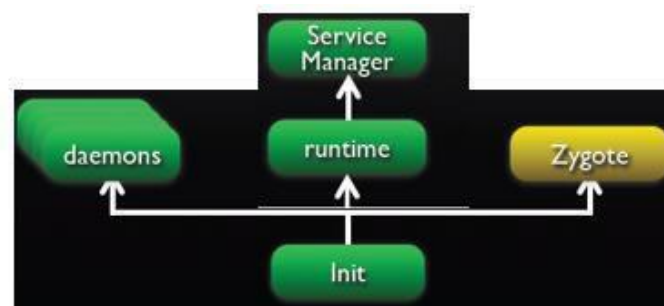
Runtime Walkthrough – Zygote



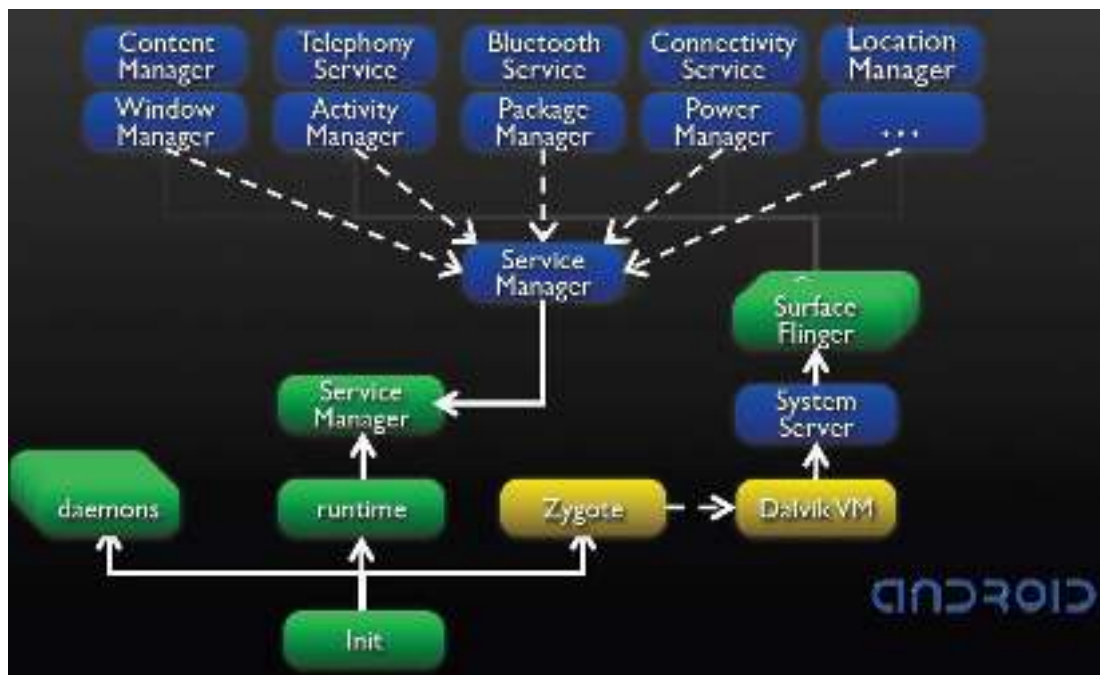
Package, Service and Process

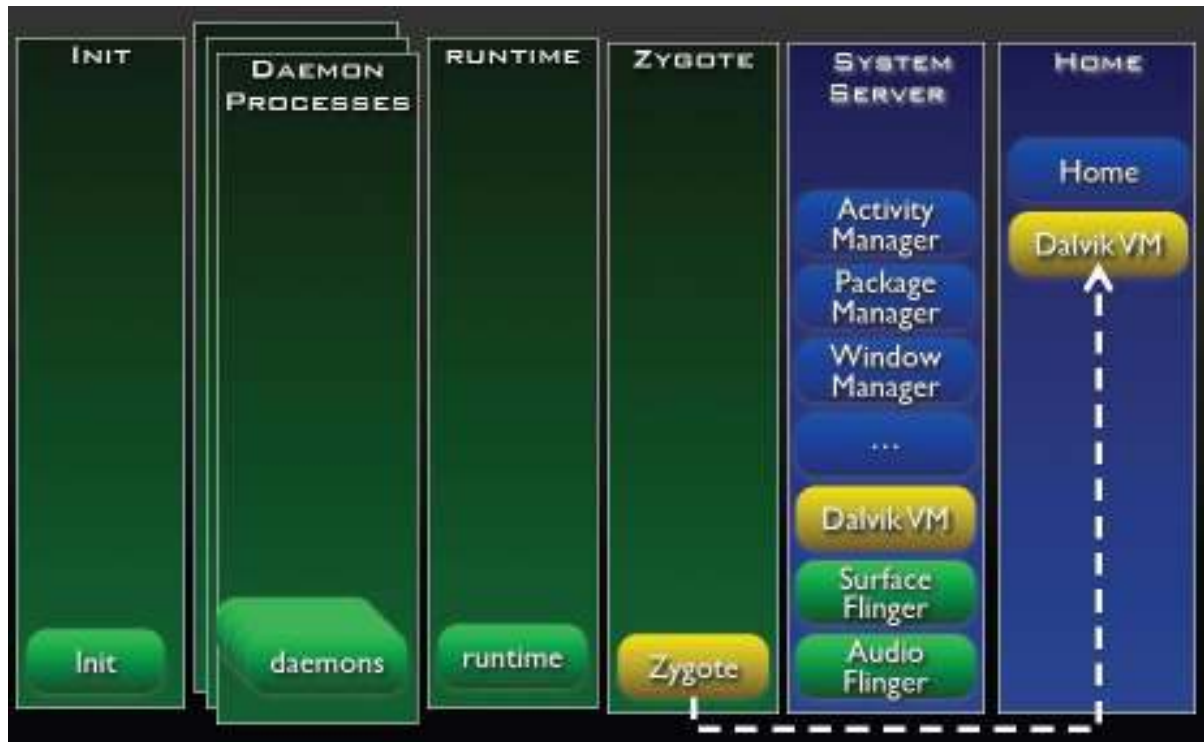


Runtime Walkthrough – Run Time Process



Runtime Walkthrough

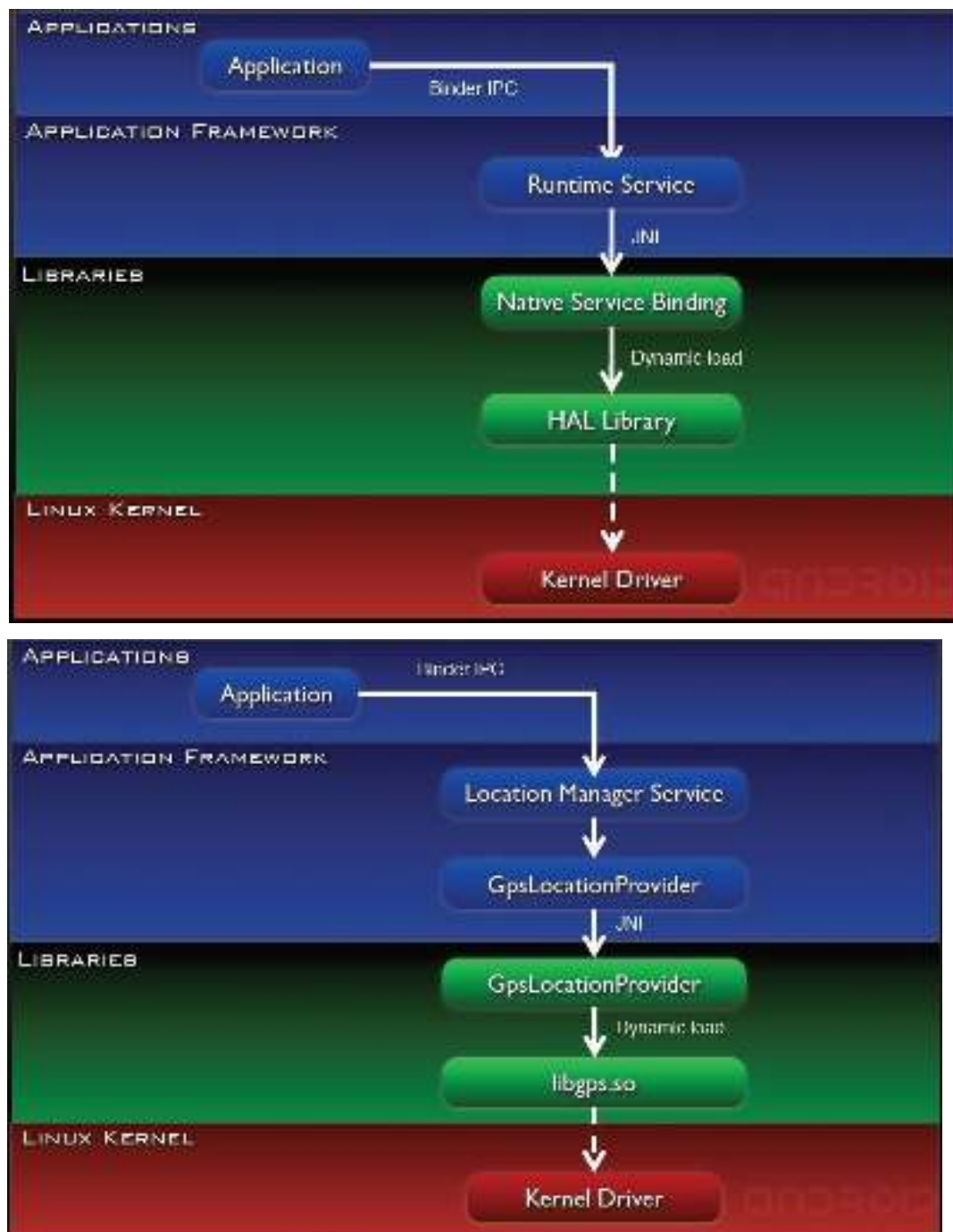


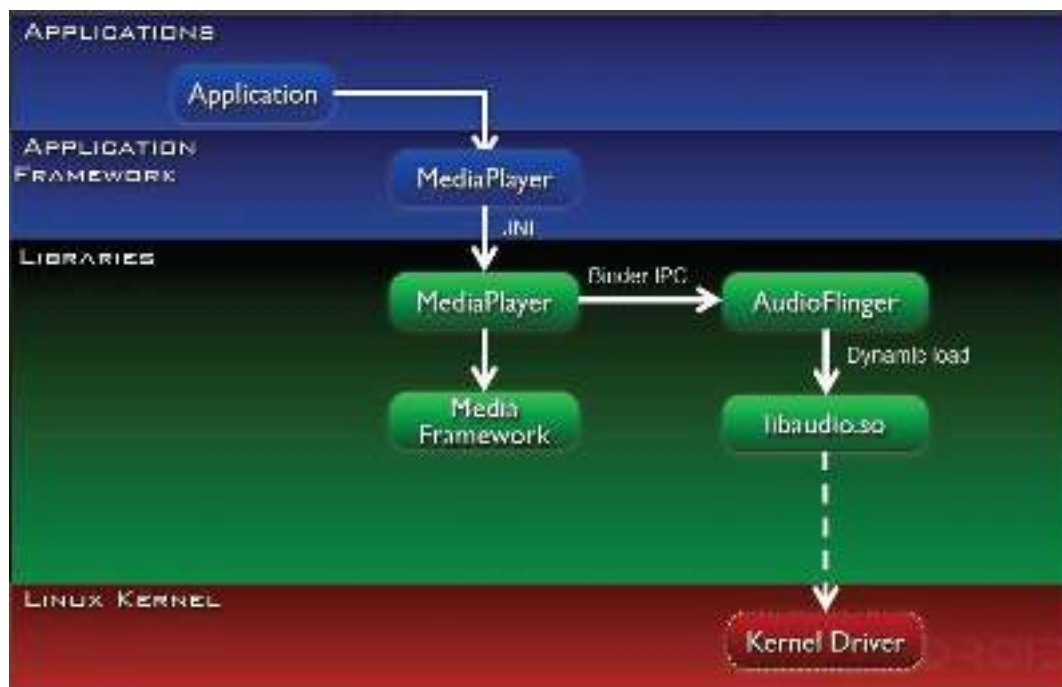
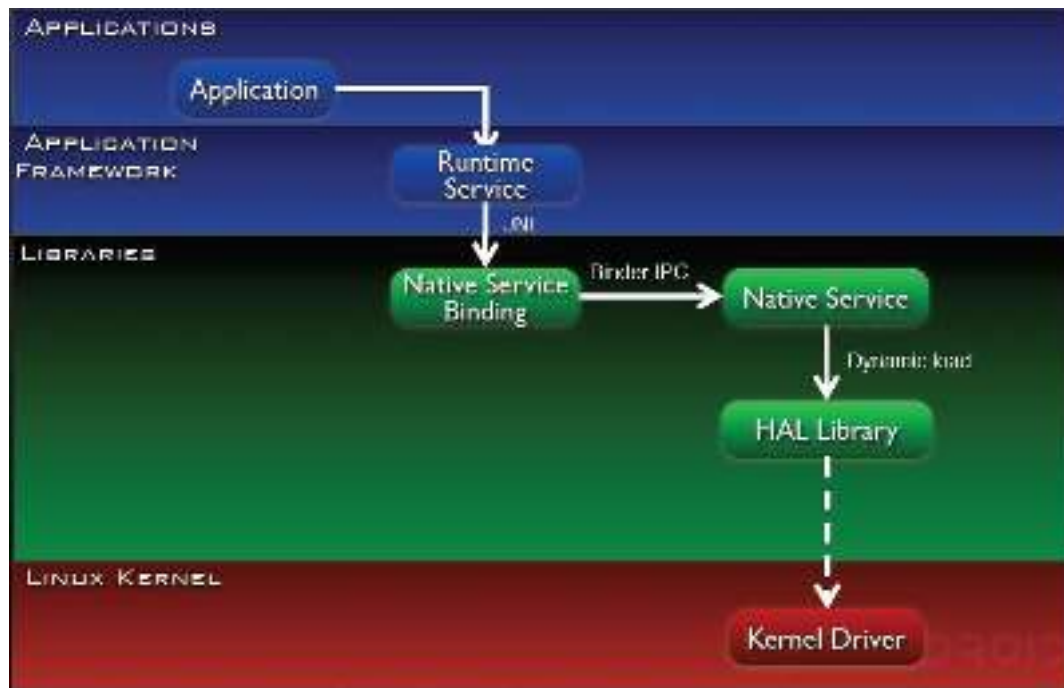


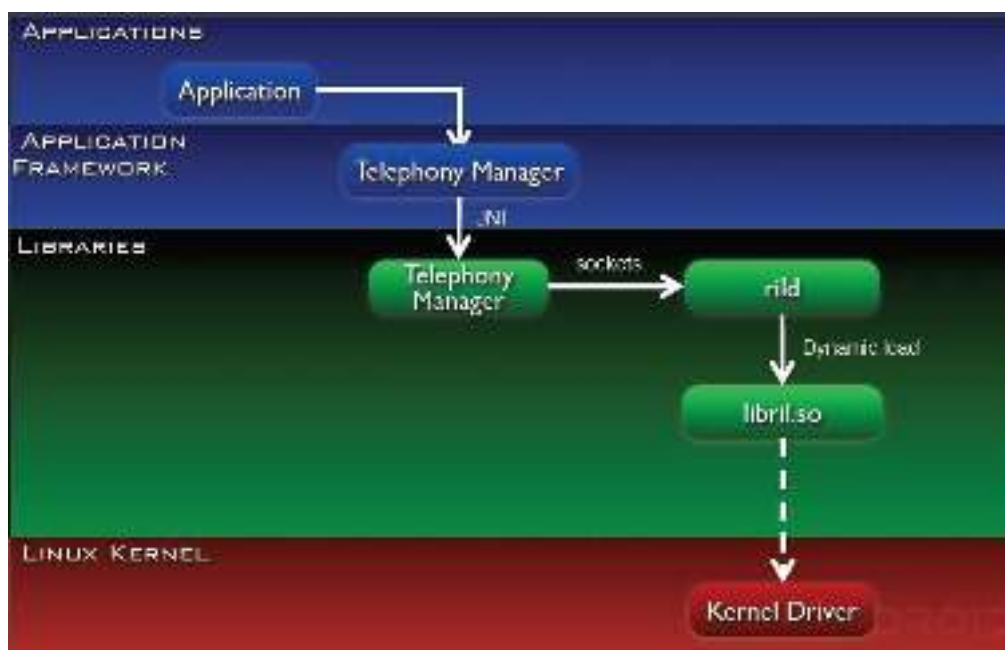
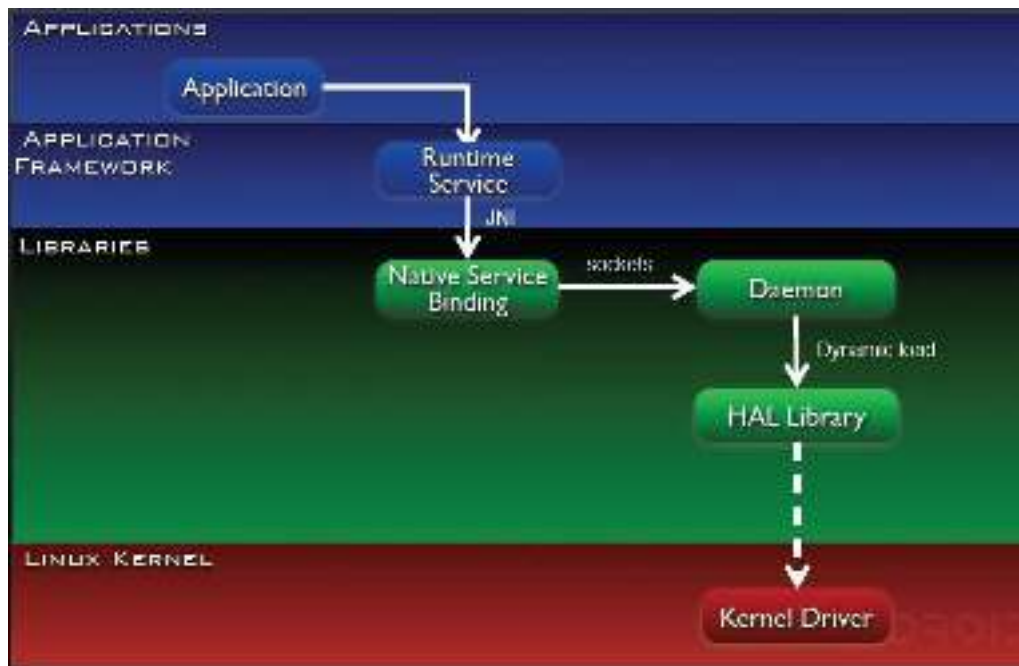


Lecture 21

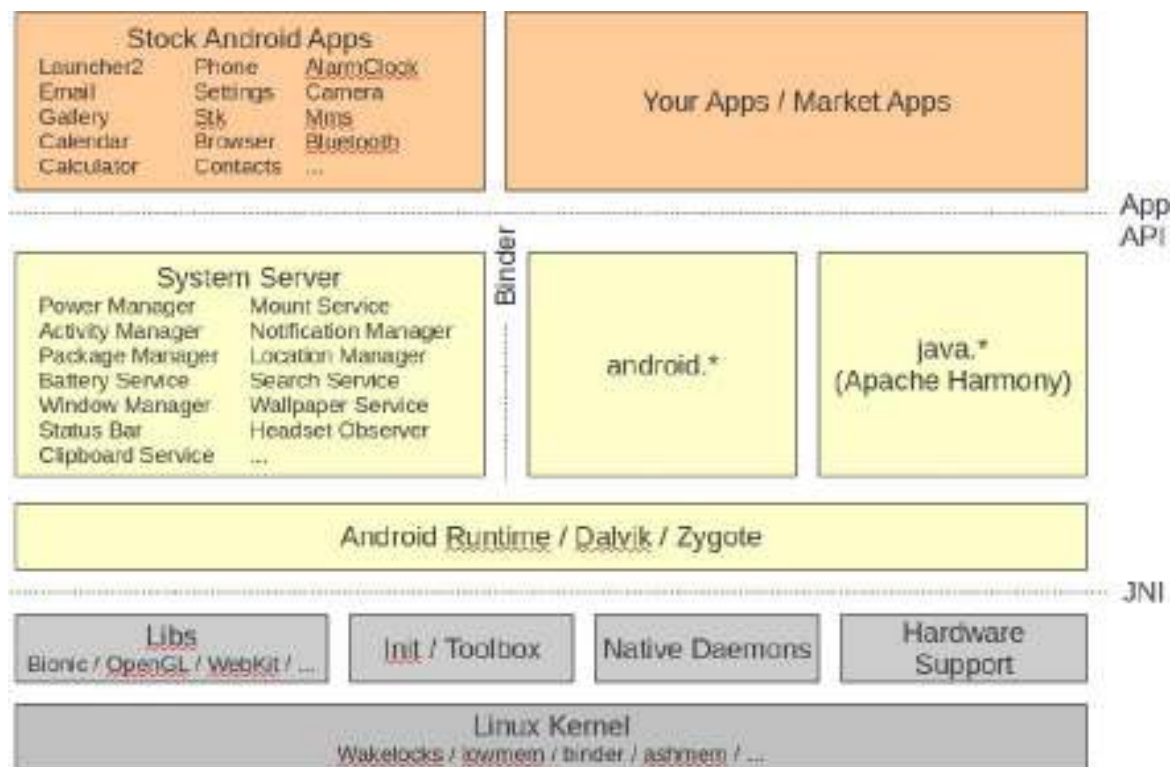
Layer Interaction







Overall Architecture

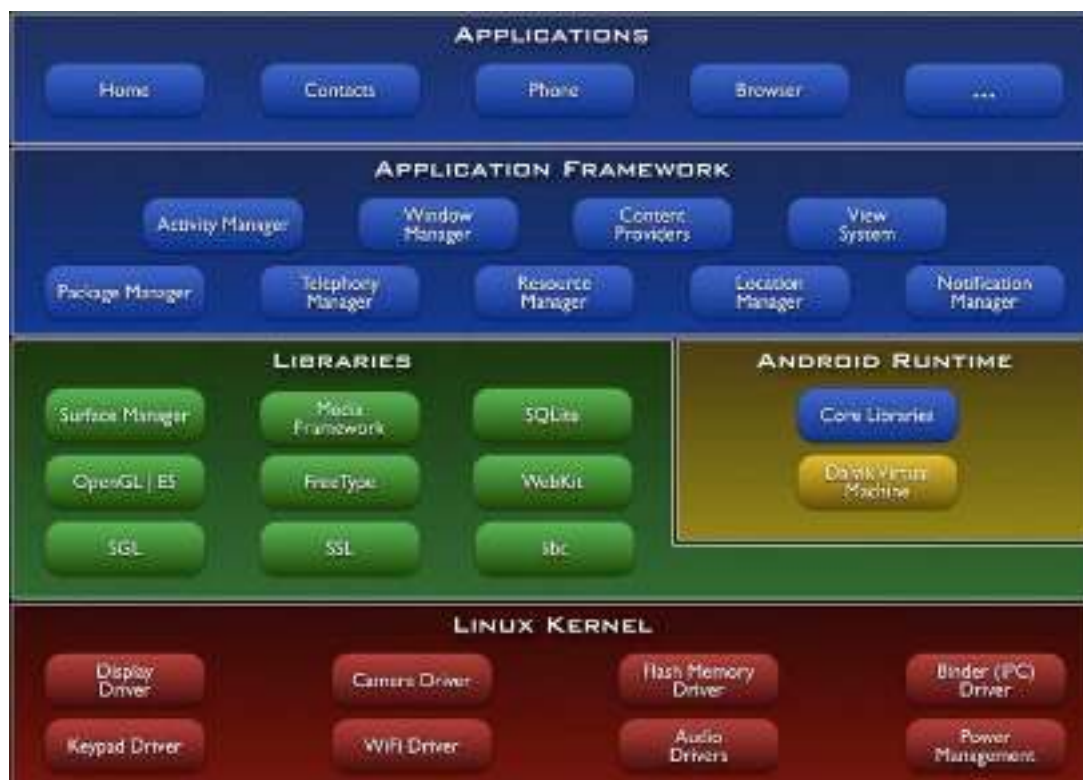


Lecture 22

Android

- Android is an open source software stack for mobile devices that includes an operating system, middleware and applications.
- Google, Inc. purchased the original developer of the software Android Inc. in 2005
- Google and other members of the Open Handset Alliance collaborated on Android's development and release
- The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android
- Android's mobile operating system is based upon a modified version of the linux kernel
- Software Stack consists of
 - Java applications running on java based, object oriented application framework on the top of java core libraries running on a Dalvik virtual machine featuring a JIT compilation

Android Software Stack



Windows Phone 7

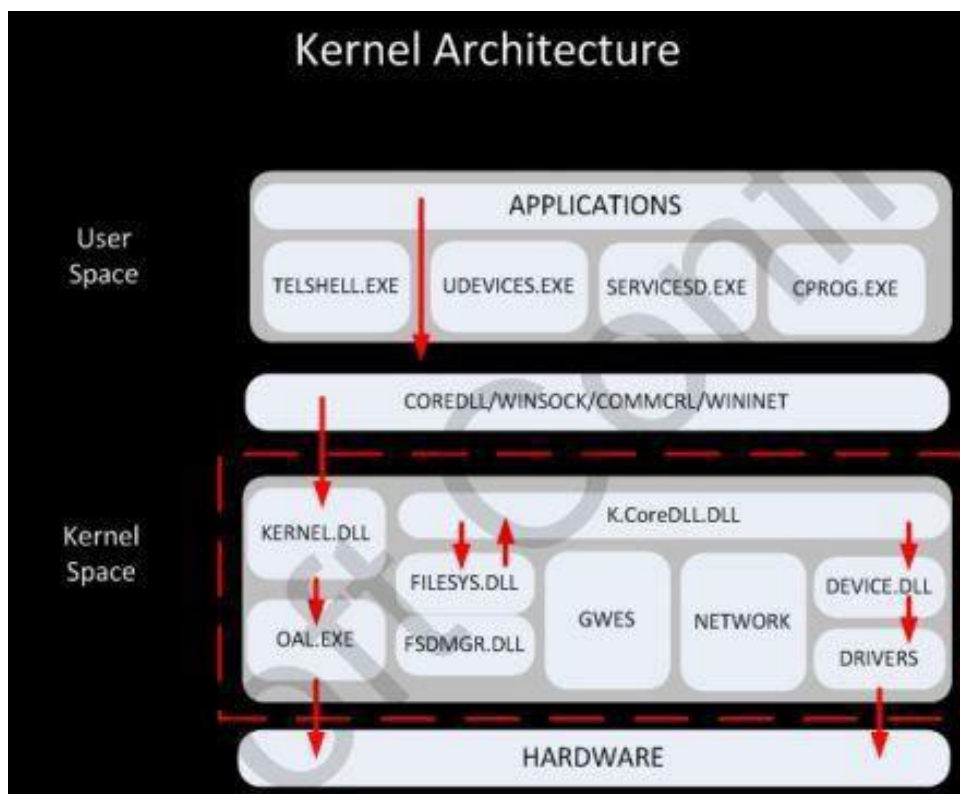
- Windows Phone 7 is a mobile operating system developed by Microsoft and is the successor to its Windows Mobile platform.
- Unlike its predecessor, it is primarily aimed at the consumer market rather than the enterprise market
- It was launched in Europe, Singapore, Australia and New Zealand on October 21, 2010, and in the US and Canada on November 8, 2010, Mexico on November 24, 2010, with Asia to follow in 2011
- With Windows Phone 7, Microsoft offers a new user interface with its design language named Metro, integrates the operating system with 3rd party and other Microsoft services, and controls the hardware it runs on
- Work on a major Windows Mobile update may have begun as early as 2004 under the codename "Photon", but work moved slowly and the project was ultimately cancelled.
- In 2008, Microsoft reorganized the Windows Mobile group and started work on a new mobile operating system.
- The product was to be released in 2009 as Windows Phone, but several delays prompted Microsoft to develop Windows Mobile 6.5 as an interim release.
- Windows Phone 7 was developed quickly. One result was that Windows Mobile applications do not run on it.
- On October 11, 2010, Microsoft's CEO announced 10 devices operating Windows Phone 7, made by HTC, Dell, Samsung and LG.
- Microsoft reported on December 21, 2010 that in the first 6 weeks phone manufacturers sold 1.5 million Windows Phone 7 devices to mobile operators and retailers.
- On 11 February 2011, at a press event in London, Microsoft CEO and Nokia CEO announced a partnership between their companies in which Windows Phone would become the primary smartphone operating system for Nokia.
- The event was largely focused on creating "a new global mobile ecosystem", suggesting competition with Android and iOS, by saying "It is now a three horse race".
- Integration of Microsoft services with Nokia's own services were announced specifically that Bing would power search across Nokia devices, and an integration of Nokia Maps with Bing Maps as well as Nokia's application store being integrated with the Windows Phone Marketplace.

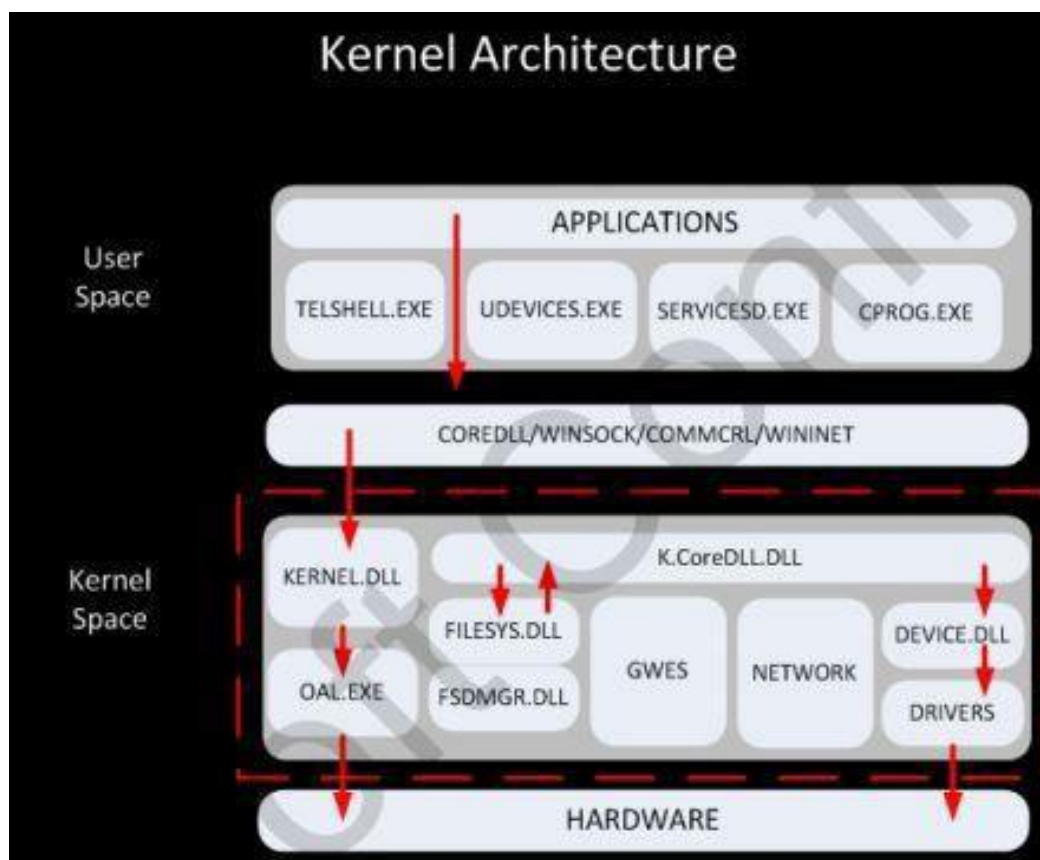
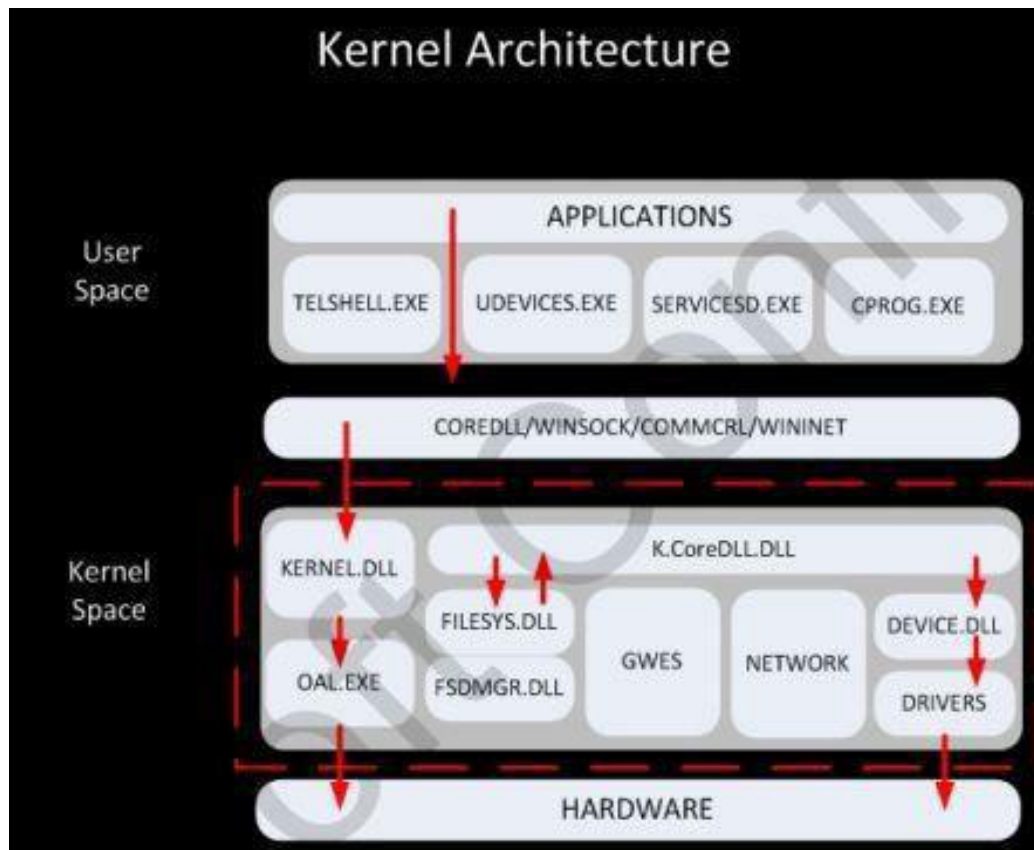
Lecture 23

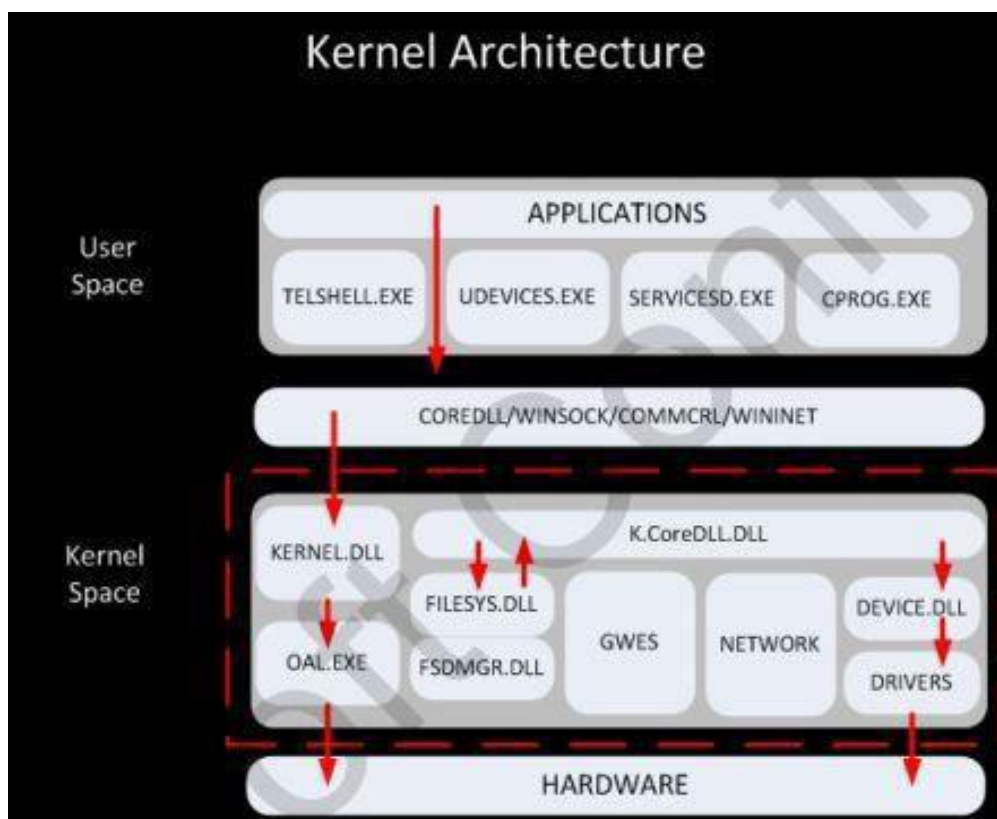
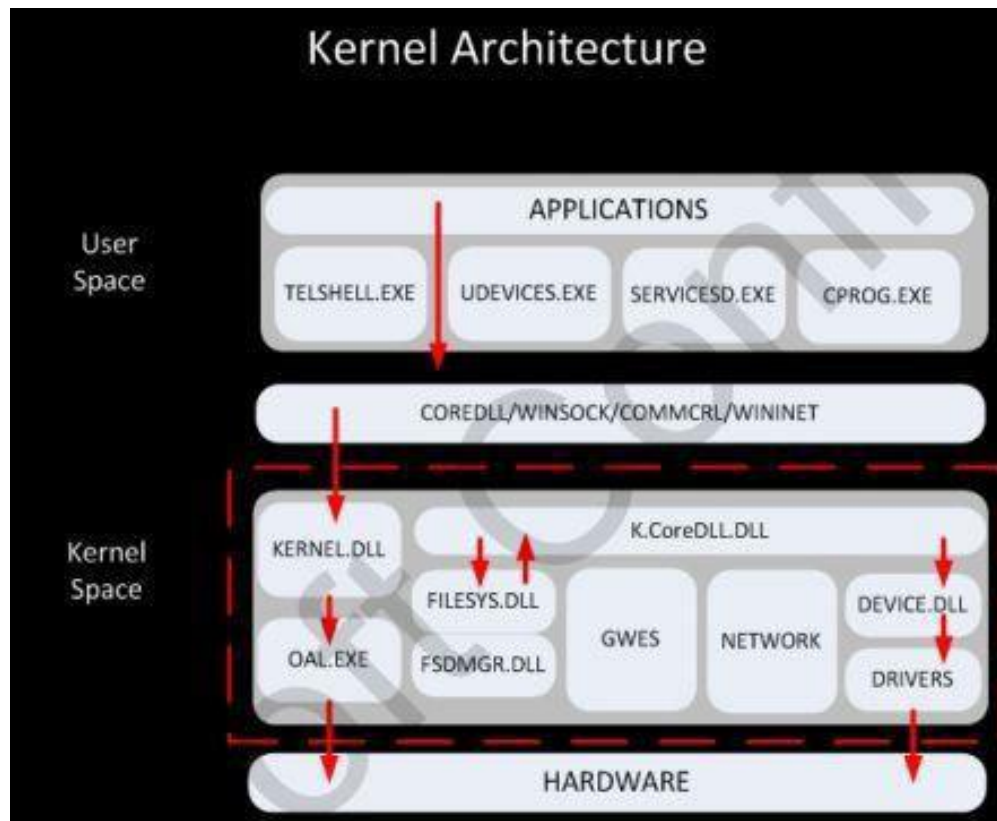
Version History

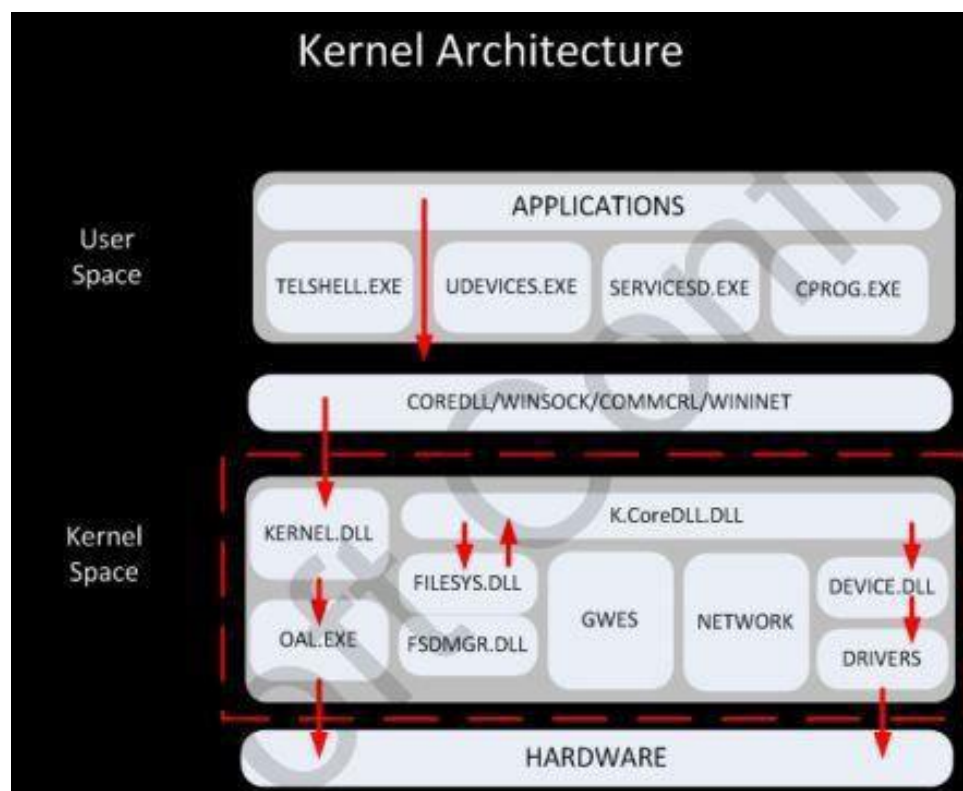
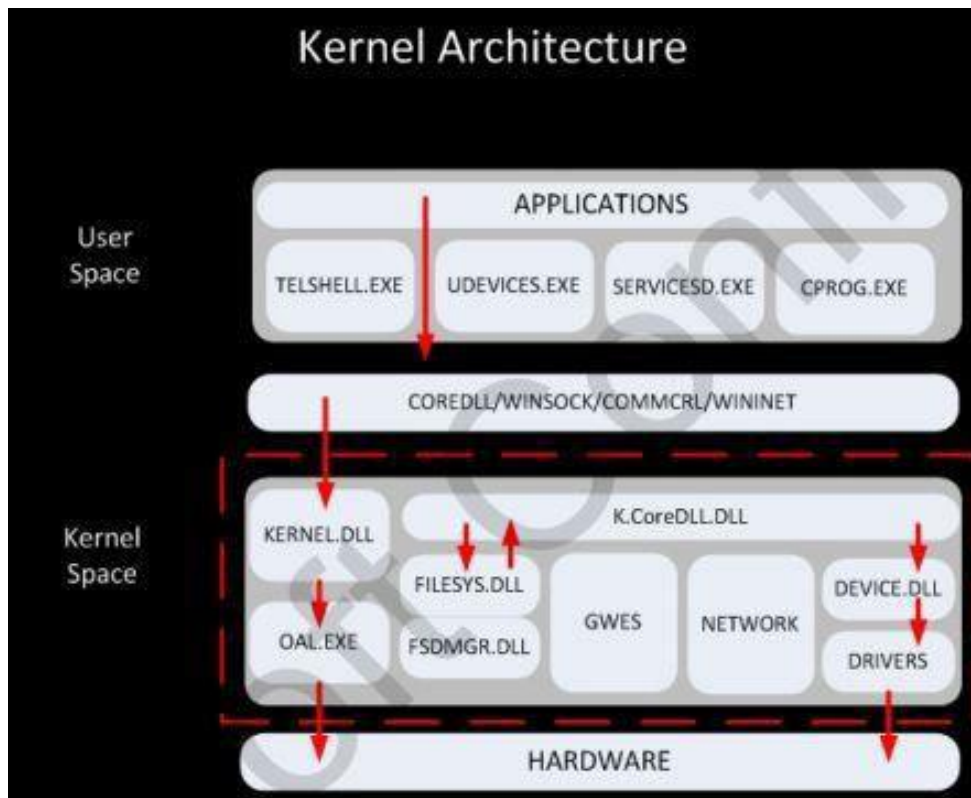
- 7.0.7004.0 - Original WP7 RTM version.
- 7.0.7008.0 - Intermediate test update (no functionality changes).
- 7.0.7389.0 - New phones installed with this version includes all the features of OS version 7.0.7390.0.
- 7.0.7390.0 - "NoDo" update (copy/paste, performance improvements, market search improvements.)
- 7.?.?????.? - "Mango" update (3rd party multitasking, IE9, Twitter integration, etc...)

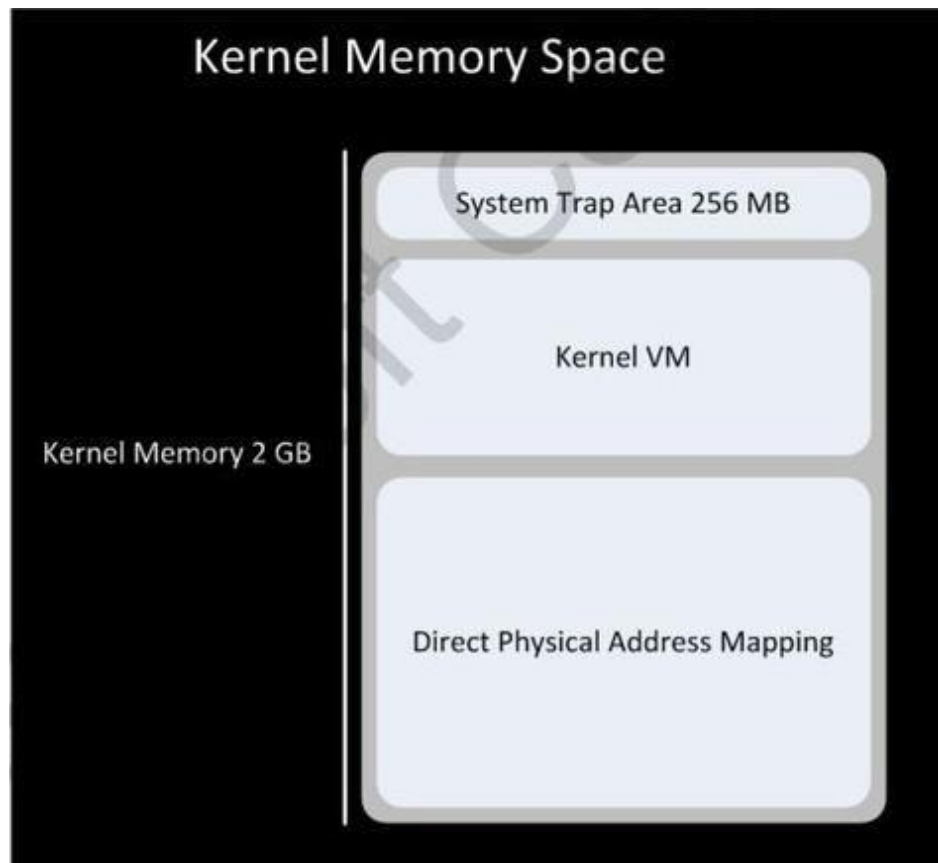
Structure of the OS – Building











Structure of the OS – File

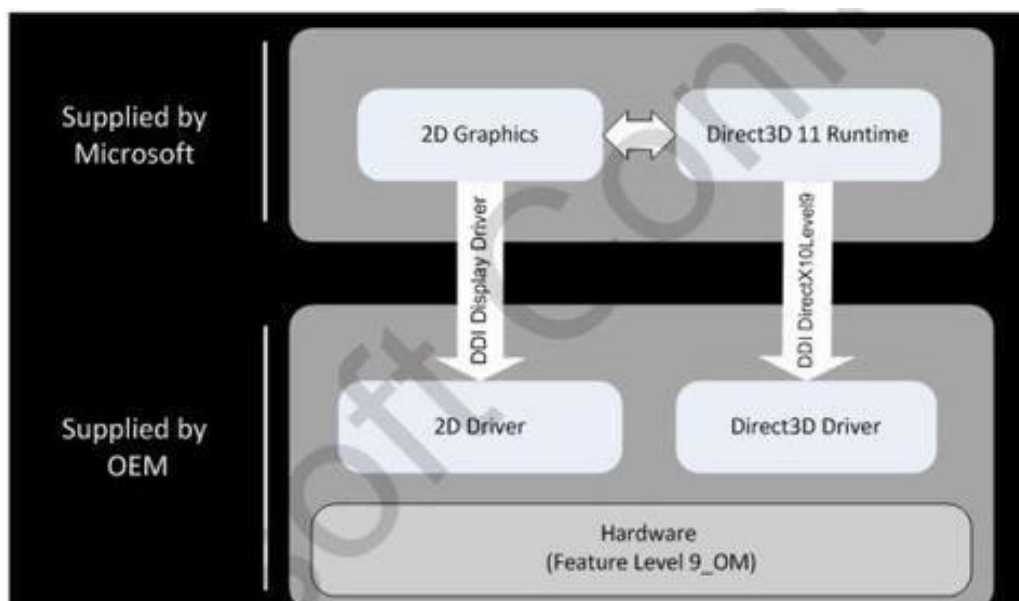
- Phone OS Windows 7 uses two file systems
- IMGFS and TexFAT.
 - IMGFS is intended for system files
 - TexFat is an 'extended' version of FAT can handle those files larger than 4GB
- For user files, Microsoft Unified Storage System.
 - This system ensures that applications and users cannot distinguish between files in the internal flash memory and files on a memory card.

Lecture 24

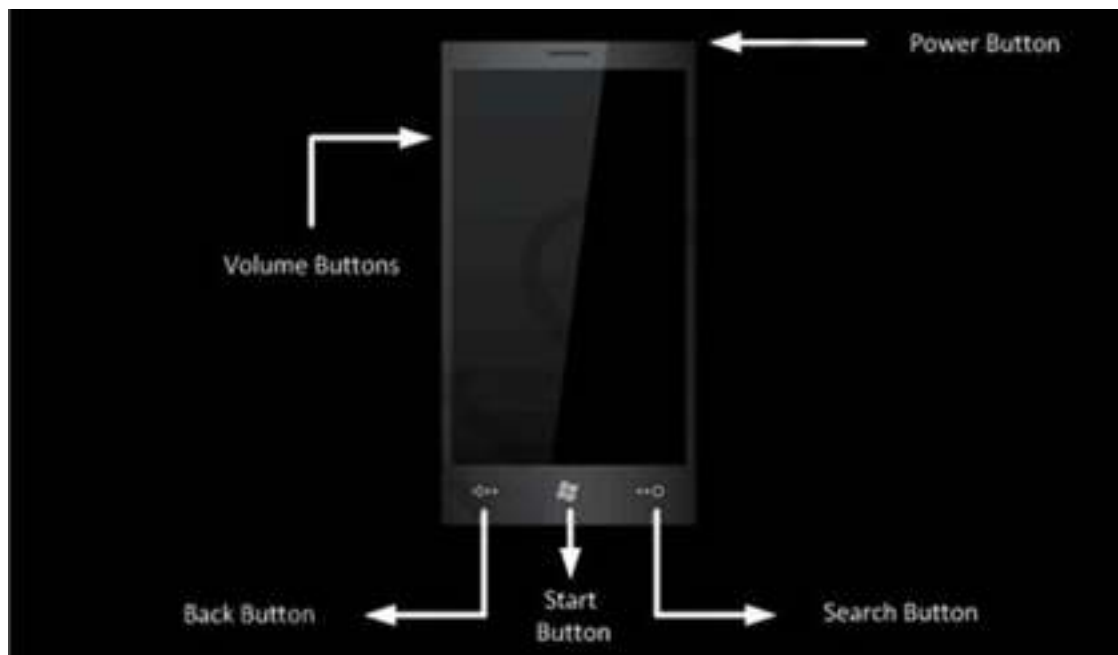
Structure of the OS – File

- Phone OS Windows 7 uses two file systems
- IMGFS and TexFAT.
 - IMGFS is intended for system files
 - TexFat is an 'extended' version of FAT can handle those files larger than 4GB
- For user files, Microsoft Unified Storage System.
 - This system ensures that applications and users cannot distinguish between files in the internal flash memory and files on a memory card.

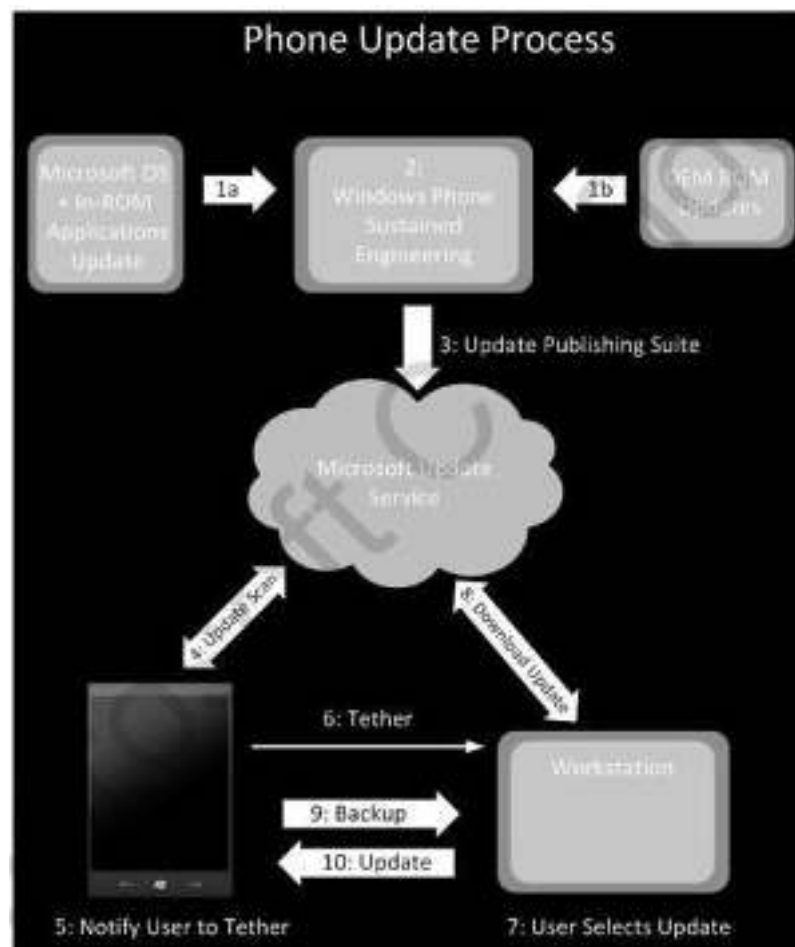
Structure of the OS – Graphics



Windows 7 – Requirements

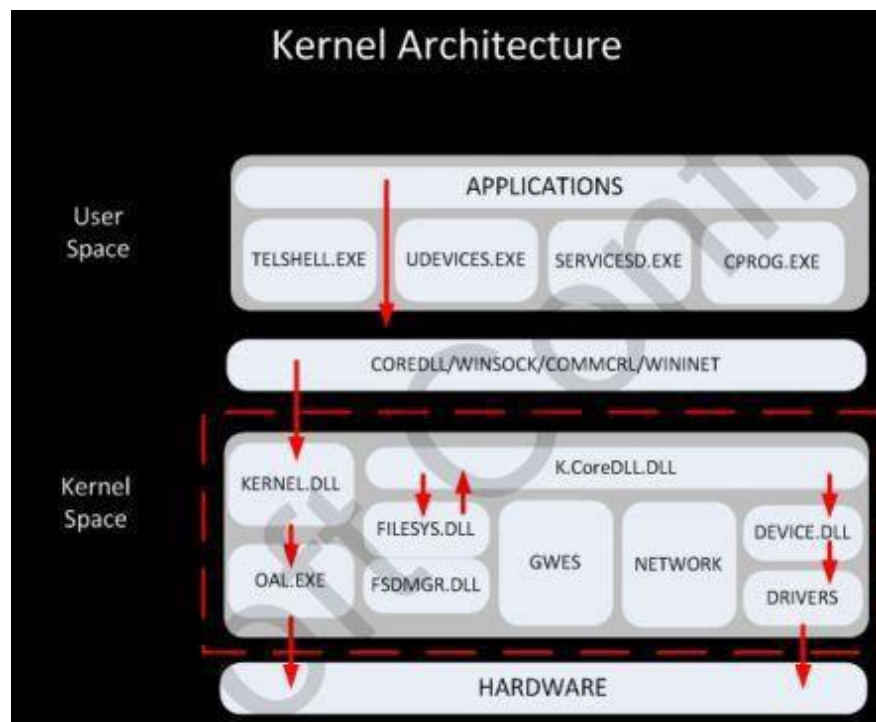


Windows 7 – Updates



Lecture 25

Structure of the OS – Building

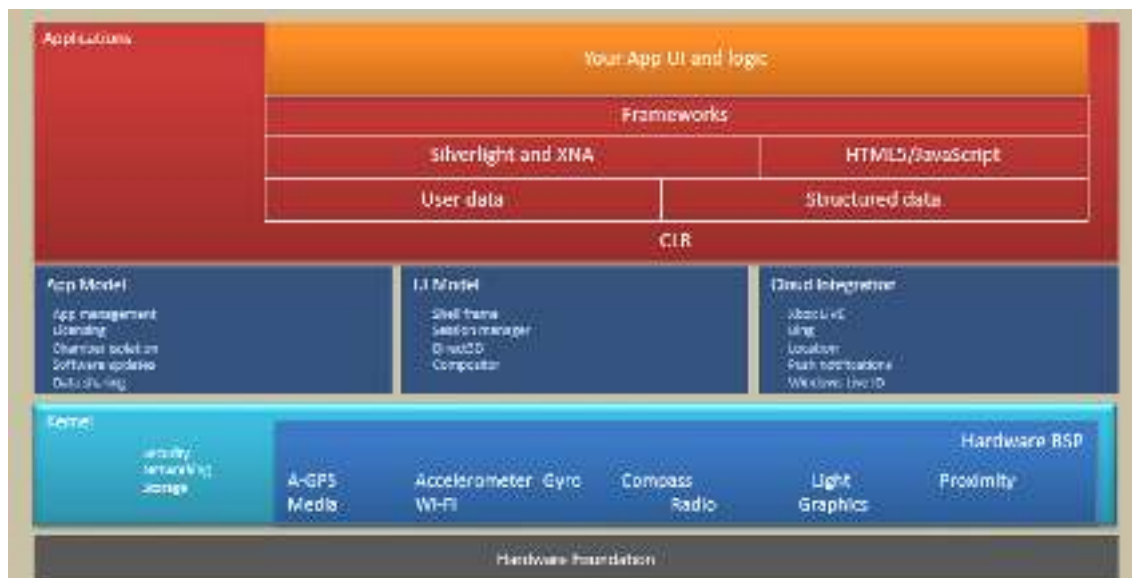


Windows 7 – Requirements

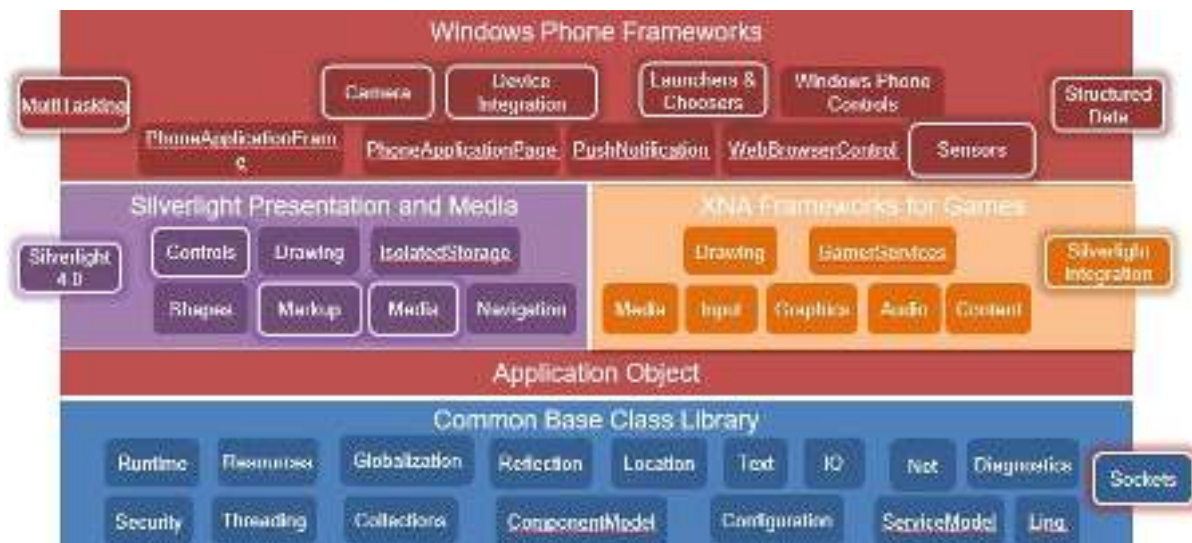




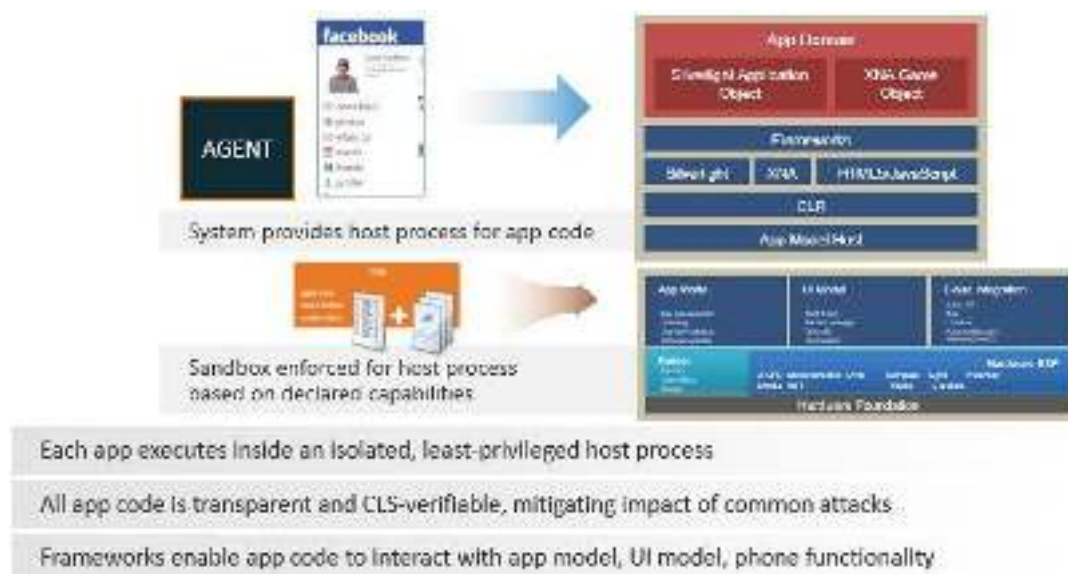
Software Architecture



Framework Details.

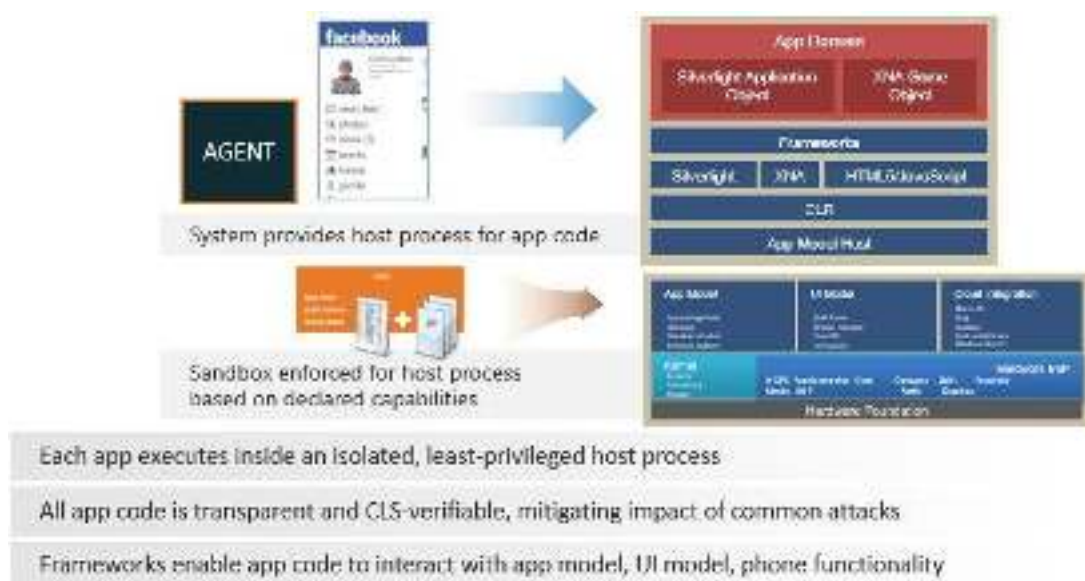


App Hosting and Run Time



Lecture 26

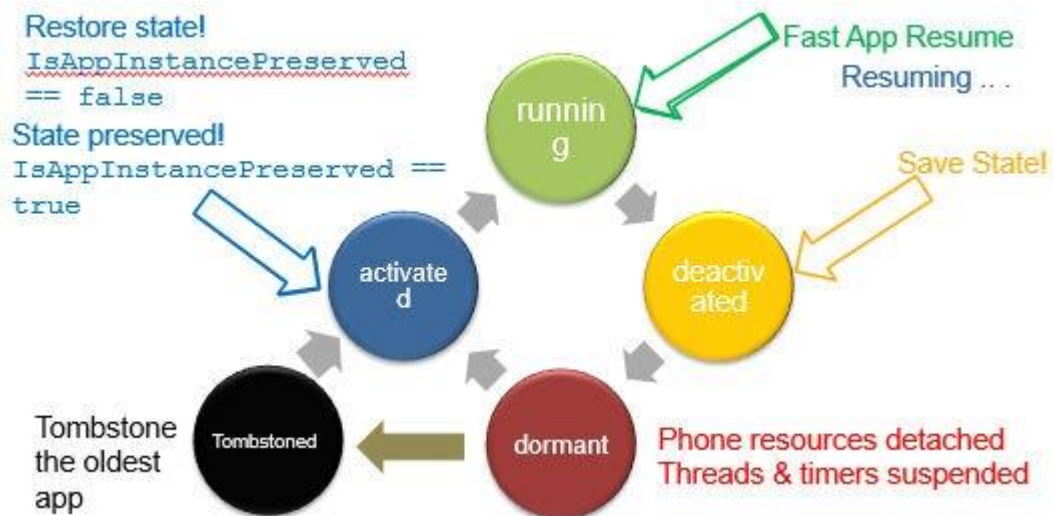
App Hosting and Run Time



WP7 Execution Model

- There are four application states:
 - Launch
 - Running
 - Closing
 - Deactivated
 - Activated
 - Dormant
 - Tombstoned

Application Life Cycle



Deactivation Resource Management

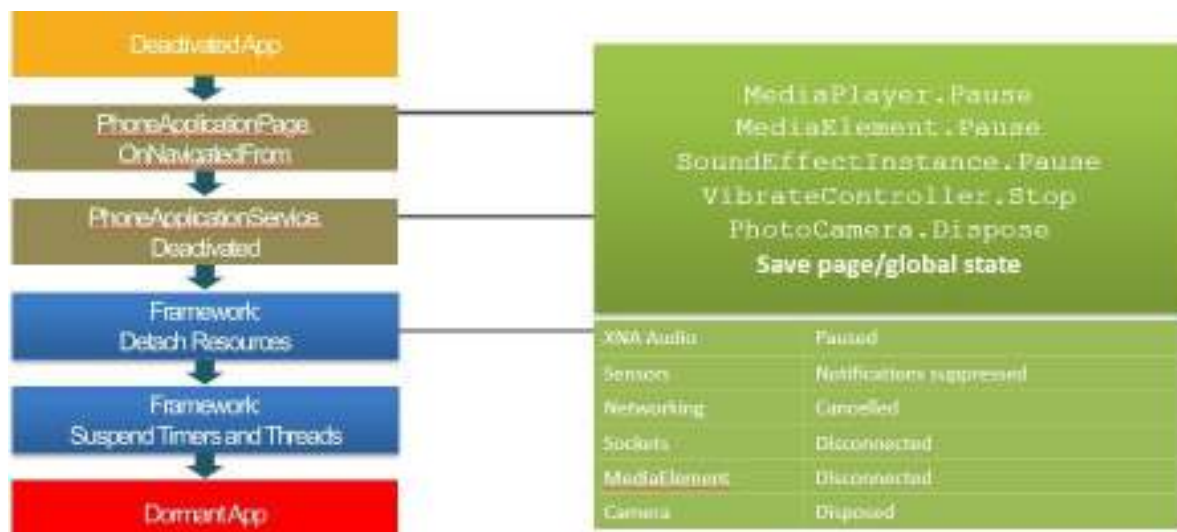


Activation Resource Management



Lecture 27

Deactivation Resource Management



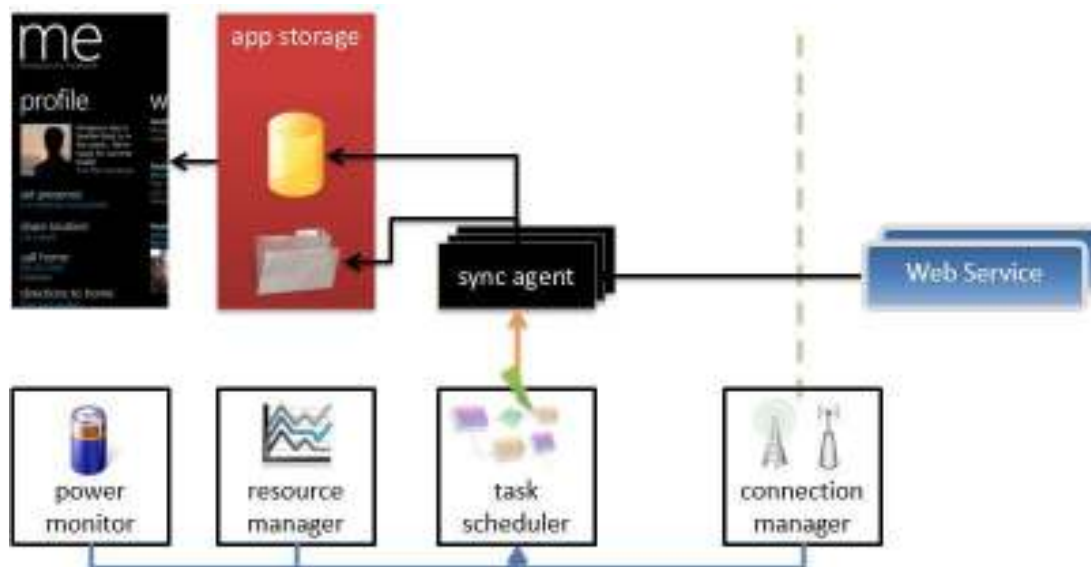
Activation Resource Management



Content Centric Experience: Challenges



Content Centric Experience: Working



Structured Data and IO Performance



SD Metric	Target	Card A	Card B	Card C
64KB seq. writes/sec	4	17	9	2
64KB seq. reads/sec	8	26	15	7
4KB write IOPs	20	87	3	57
4KB read IOPs	500	1054	1022	373



Structured Data in Mango



app agent



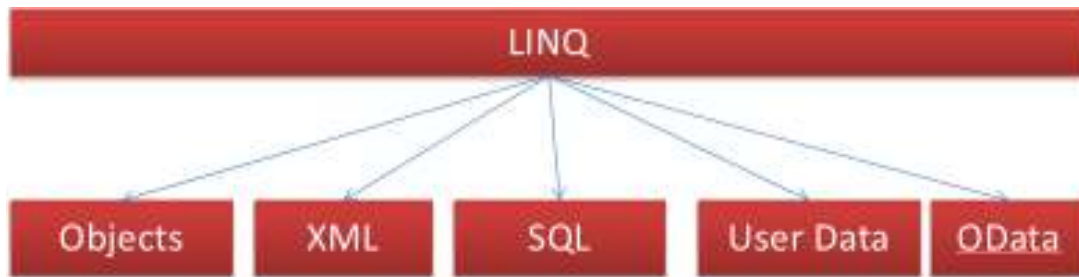
Linq



Web Service

Lecture 28

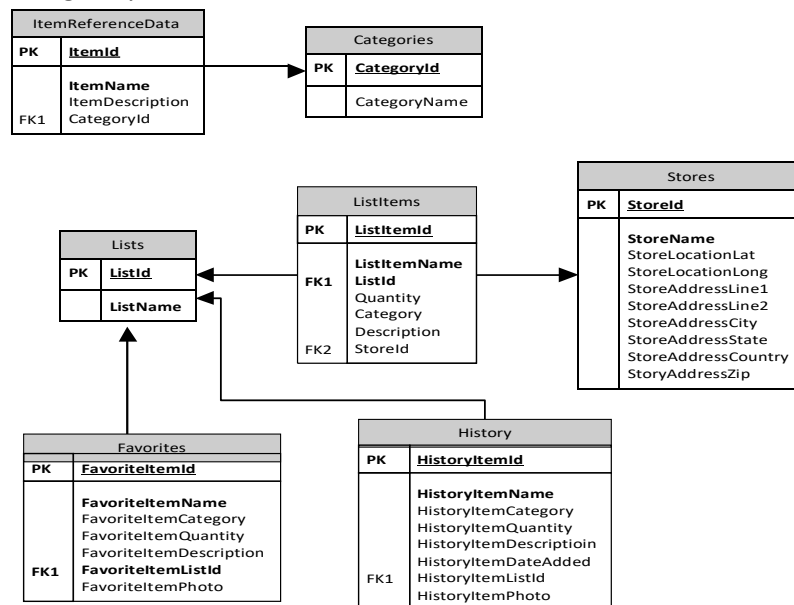
LINQ to Everything



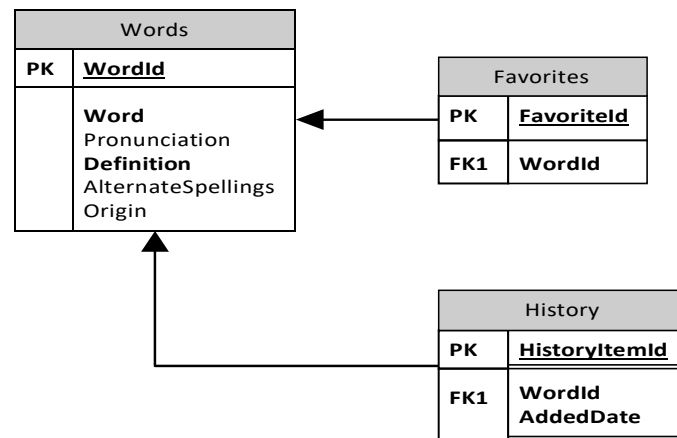
7 → Mango

Complex Scenarios

- Numerous relationships and constraints
- Example: Shopping List
 - 7 tables
 - 100s of records
 - 5 foreign keys

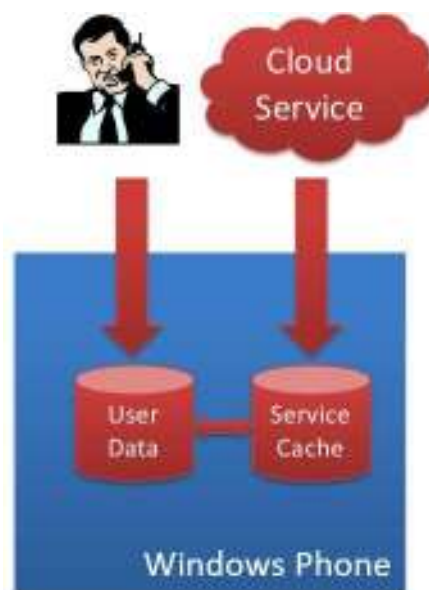


- Huge amounts of static reference data
- Example: dictionary app
 - 3 tables
 - 1 table with 500k rows



Web Services Cache

- Fetch reference data from cloud
- Cache it locally
- Combine with user-specific data



Users Data

- Filter contacts
 - Birthdays in the next month
- Query all appointments
 - Find an available time for a meeting

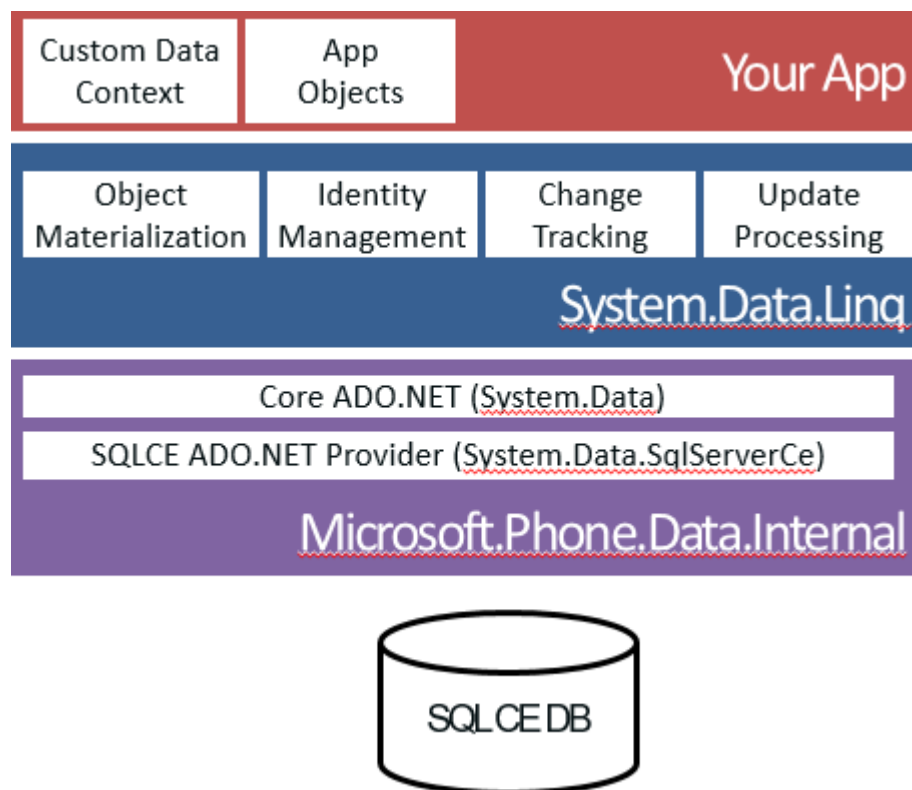


Filter

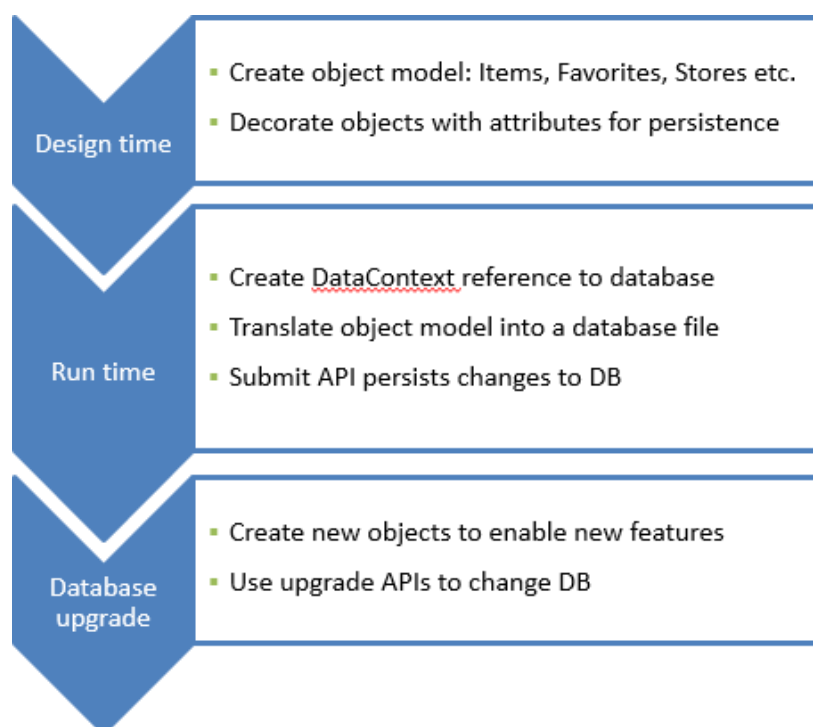
Local Storage Data – Overview



Architecture



Code First Development



User Data – New and Updated APIs

- Chooser Tasks related to user data
 - EmailAddressChooserTask
 - PhoneNumberChooserTask
 - AddressChooserTask
- Microsoft.Phone.UserData for direct access
 - Contacts
 - Appointments

Microsoft.Phone.UserData

- Important points
 - Contacts and Appointments APIs are *read only*
 - Third party social network data cannot be shared

Contact/Appointment Data Share

	Contact Name and Picture	Other contact data	Appointments/Events
Windows Live Social	YES	YES	YES
Windows Live Rolodex (user created and SIM import)	YES	YES	n/a
Exchange accounts (corporate plus Google, etc.)	YES	YES	YES
Operator Address Books	YES	YES	n/a
Facebook	YES	NO	NO
Other networks in the People Hub (e.g., Twitter)	NO	NO	NO

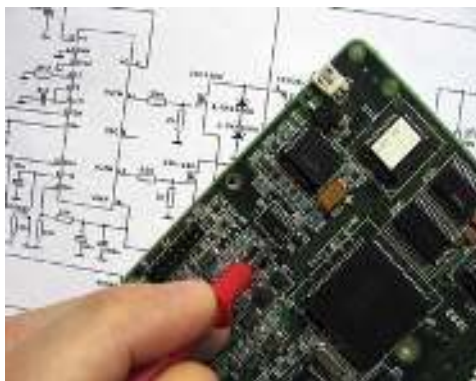
Lecture 29

Rethinking Multitasking

- Keep the UX great



- Get more out of the phone

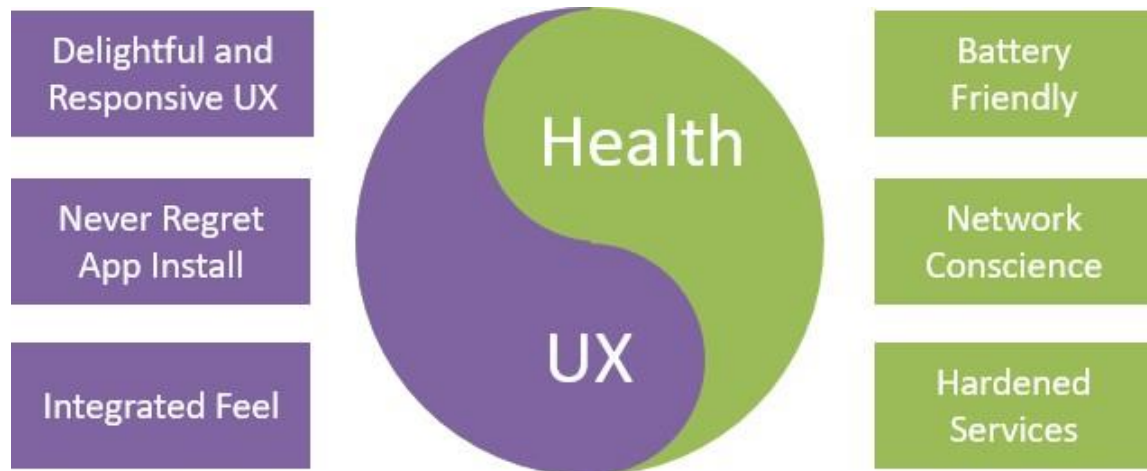


- Don't keep users waiting



Lecture 30

Windows Phone Harmony

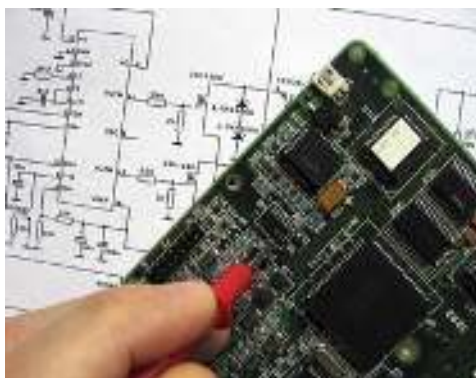


Rethinking Multitasking

- Keep the UX great



- Get more out of the phone



- Don't keep users waiting



Multitasking and Phone Health

New OS services

Processing on demand

Balance foreground and background

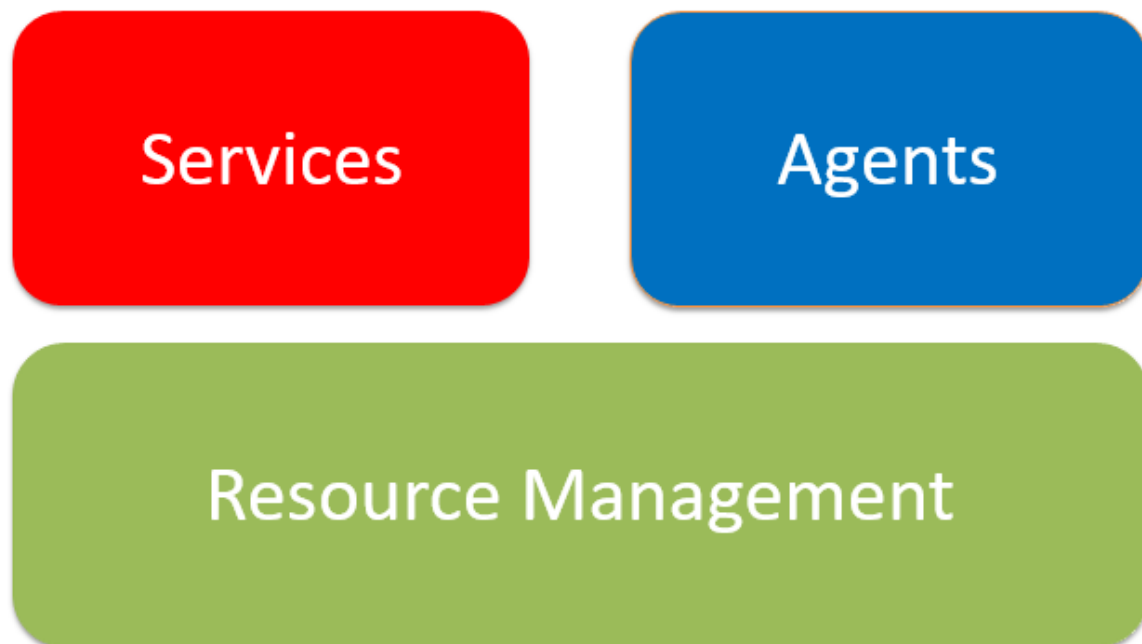
Align with system and user activity

Monitor resource usage

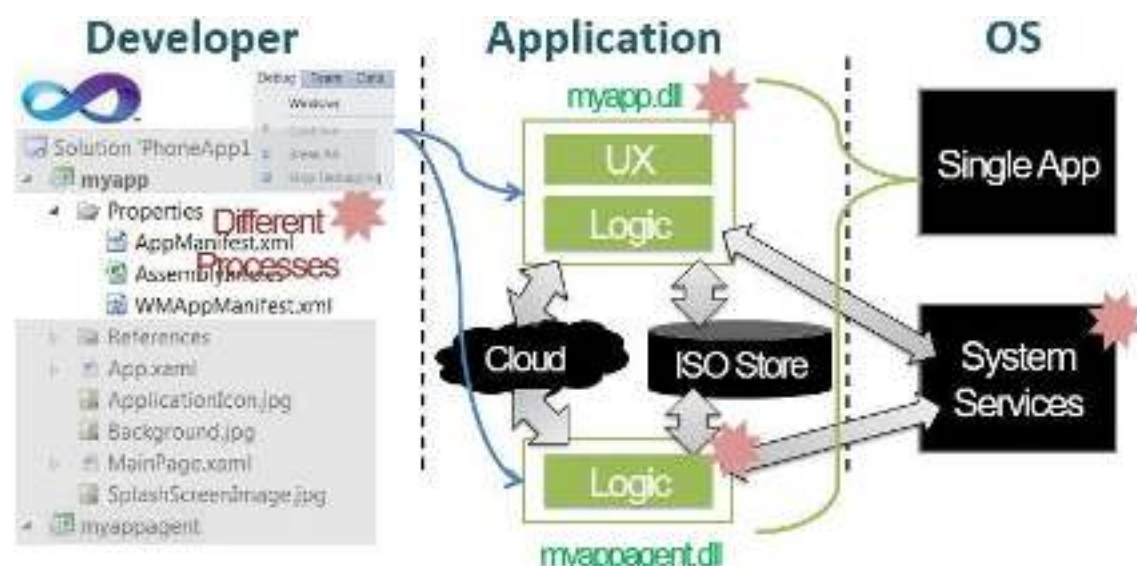
Getting more out of the phone: agents



Multitasking Components



End to End Architecture

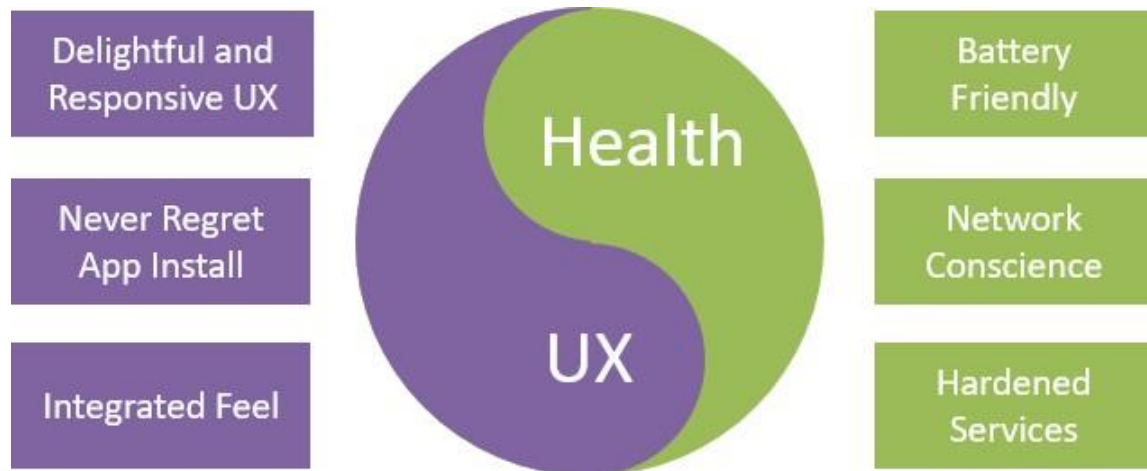


Background Agent Functionality

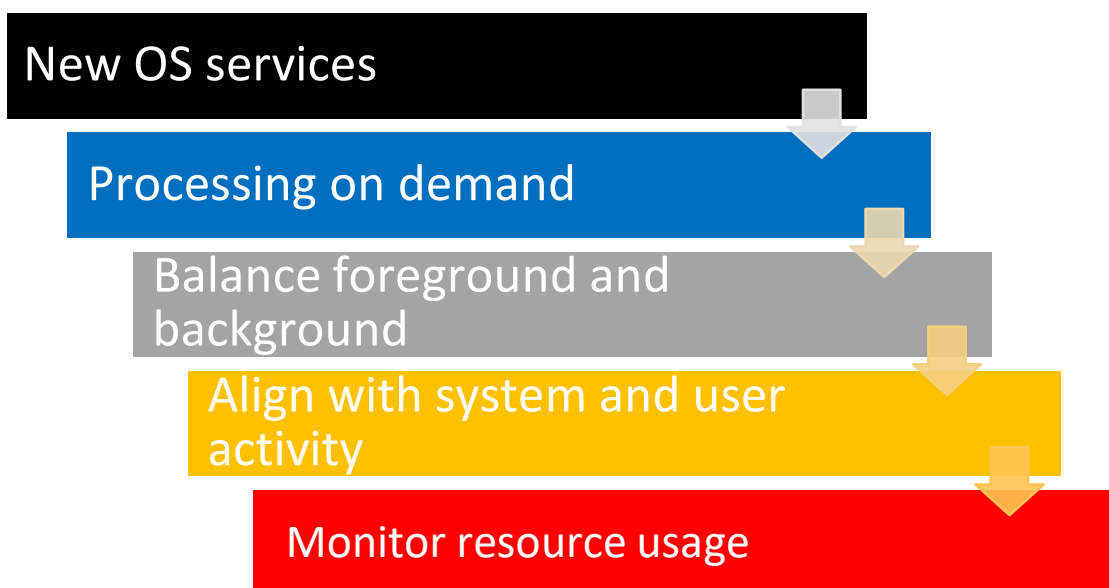
Allowed	Restricted
▪ Tiles ▪ Toast ▪ Location ▪ Network ▪ R/W ISO store ▪ Structured storage ▪ Sockets ▪ Most framework APIs	▪ Display UI ▪ XNA libraries ▪ Microphone and Camera ▪ Sensors ▪ Play audio (may only use background audio APIs)

Lecture 31

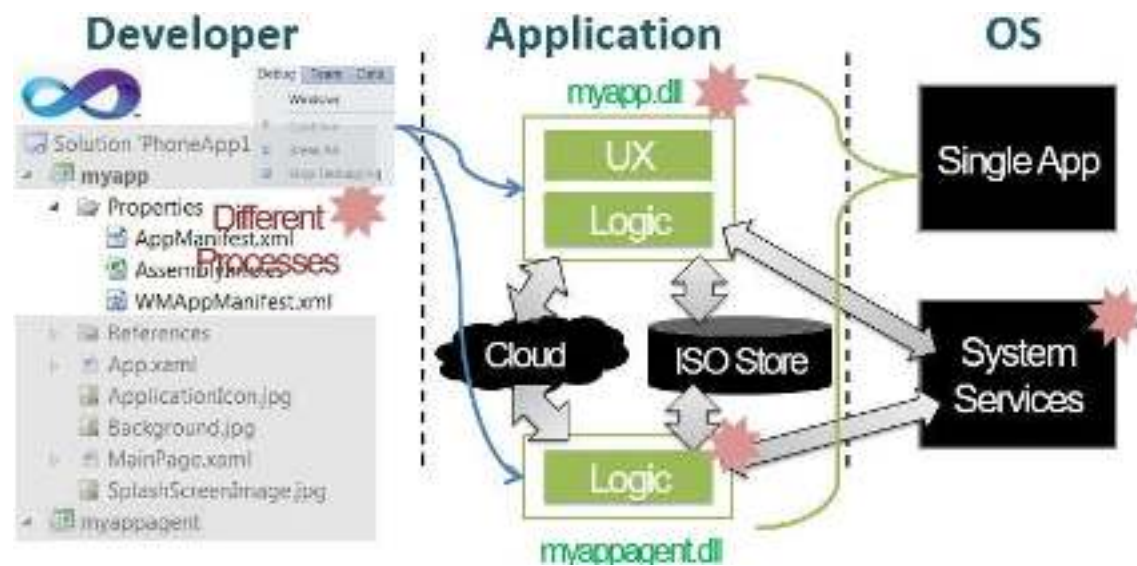
Windows Phone Harmony



Multitasking and Phone Health



End to End Architecture



Background Agent Functionality

Allowed	Restricted
▪ Tiles ▪ Toast ▪ Location ▪ Network ▪ R/W ISO store ▪ Structured storage ▪ Sockets ▪ Most framework APIs	▪ Display UI ▪ XNA libraries ▪ Microphone and Camera ▪ Sensors ▪ Play audio (may only use background audio APIs)

Additional Functionality



Generic Background Agents

- Agents
 - Periodic
 - On Idle
 - May have one or both
- Initialized in foreground, run in background
 - Persisted across reboots
- User control through CPL
 - Up to a maximum of 18 periodic agents
- Synchronize with foreground through mutex
- Agent runs for up to 14 days (can be renewed)

Lecture 32

Generic Background Agents

Periodic Agents

- Occurrence
 - Every 30 min
- Duration
 - 15 seconds
- Scenarios
 - Incremental data sync
 - Location
 - Others...

On Idle Agents

- Occurrence
 - External power, non-cell network
- Duration
 - 10 minutes
- Scenarios
 - Data feasting
 - Initial sync
 - Others...

Don't Keep Users Waiting: Background Transfers



Multitasking Cheat Sheet

Job	Tool
Resume quickly from the lock screen	Fast App Switching (it's free!)
Set an alarm or reminder at a precise time	Background Notification
Large file downloads	Background Transfer
Event-based toast/tile updates	Push Notifications
Location-based services; regular toast/tile updates; data pre-caching; etc.	Periodic Background Agent
Play music in the background	Background Audio Player
Synchronize data; SETI@home; etc.	Resource-Intensive Agent
Real-time GPS tracking	Run under the lock screen



Services and Frameworks



WP7 Security Model

- The need for protection
- Chambers
- Capabilities
- Sandbox
- Application deployment

Chambers

- There are four chamber types
 - Trusted Computing Base (TCB)
 - Elevated Rights Chamber (ERC)
 - Standard Rights Chamber (SRC)
 - Least Privileged Chamber (LPC)

Capabilities

- Each application discloses its capabilities to the user, including:
 - Disclosure on the application details page in the Windows Phone Marketplace.
 - Disclosure with an explicit prompt upon application purchase, for those capabilities that have legal requirements for explicit disclosure and specific consent collection.
 - Disclosure within the application, when the user is about to use the location capability for the first time.

Sandbox

- Every application on Windows Phone 7 runs in its own isolated chamber, and is defined by the declared capabilities that the application needs to function.
- Applications developed by other companies that are distributed via the Windows Phone Marketplace cannot remain active in the background.

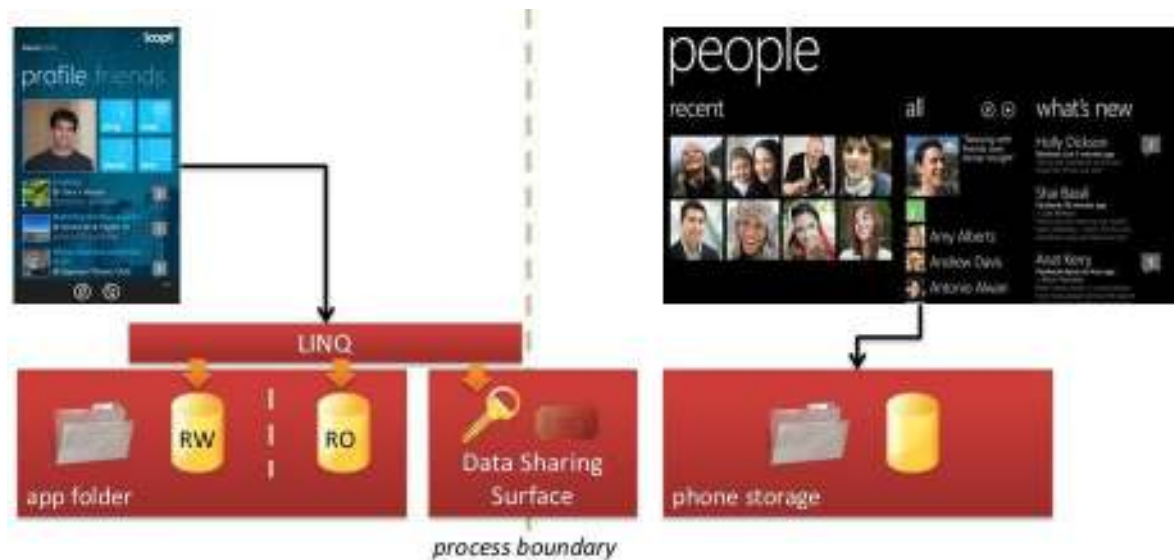
Application Deployment

- Application developers must register with Microsoft before an application can be submitted to the Marketplace Hub.
- All applications are code-signed by VeriSign.
- The application development model's use of "managed code only" in addition to the least privilege and isolation aspects of the Windows Phone OS 7.0 security model provide strong protections against security attacks

Apps, Contents and Isolation



Content Sharing for Applications



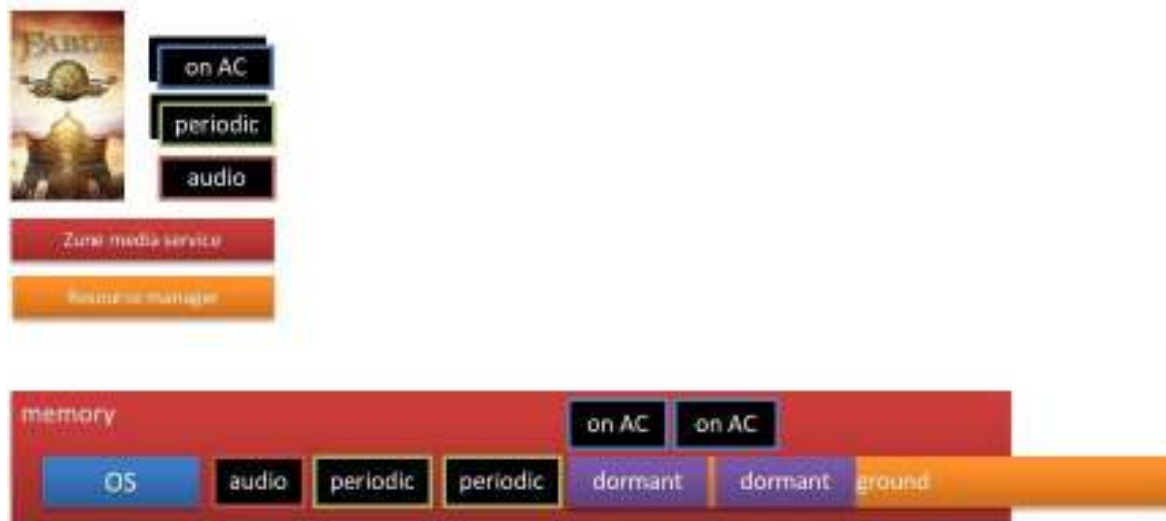
Getting More Out of the Phone: Audio Agents



Getting More Out of the Phone: Generic Agents



Don't Keep Users Waiting: Dormant Apps



Graphics Composition



Shell frame composes all UI into a single screen.

Central page management enables cross app UI transitions and other effects.

System wide Z order enforcement emphasizes core phone functionality.

Lecture 33

Research In Motion

- BlackBerry and its OS are produced by Research In Motion (RIM)
- Recognized as the cultural icon of the office environment / workforce
- Ontario based founded in 1984 by Mike Lazaridis
 - Entered the wireless market making pagers
 - Like the inter@active (RIM-900) for Ericsson
 - Wireless Email on GSM / CDMA networks
- Released the BlackBerry in 2002

Market Share

- Actual handset shipment numbers place the BlackBerry closer to 16%
- Yearly growth of 80% in 2007-08.
- RIM also offers its push email services to other platforms
 - Symbian, Windows Mobile and Palm OS.
- 1.2 million new users [Q4, 2007]
- US user base had reached 12 million users [2008]
- The consumer BlackBerry Internet Service is available in 91 countries worldwide on over 500 mobile service operators using various mobile technologies

Development Model

- BlackBerry applications are written using Java ME
 - RIM provides a JDE and Eclipse plug-in
 - Mobile Information Device Profile (MIDP)
- Defines common interface for low lever features on mobile devices
- Users can download and run any application
- Code signing is needed for certain functions
 - Does not guarantee correct code

Applications

- Many popular apps
- Obtain applications through web, desktop connect, or 'IM's homepage.
- In October 2008, RIM announced their own Application Store built directly into the BlackBerry
 - Competes with Apple and Google's app stores.
 - Offer free and paid applications

OS Architecture

- Blackberry OS is proprietary
- No significant information publicly
- Hardware access through RIM developed JVM via
 - Standard JavaME applications
 - MDS (Mobile Data System) applications
- Legacy devices supported C++, but no longer
- 'IM's attention to building cross-platform functionality
 - Symbian-OS, Windows Mobile, Desktop Connect
- Suggests nothing specific about the underlying OS
- Focus mainly on the RIM API that wraps MIDP

Lecture 34

Development Model

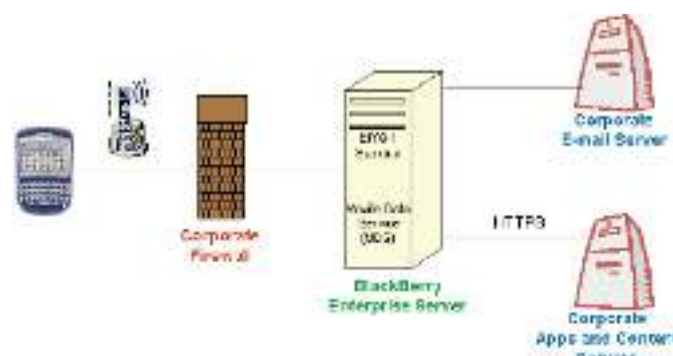
- BlackBerry applications are written using Java ME
 - RIM provides a JDE and Eclipse plug-in
 - Mobile Information Device Profile (MIDP)
- Defines common interface for low level features on mobile devices
- Users can download and run any application
- Code signing is needed for certain functions
 - Does not guarantee correct code

OS Architecture

- Blackberry OS is proprietary
- No significant information publicly
- Hardware access through RIM developed JVM via
 - Standard JavaME applications
 - MDS (Mobile Data System) applications
- Legacy devices supported C++, but no longer
- 'IM's attention to building cross-platform functionality
 - Symbian-OS, Windows Mobile, Desktop Connect
- Suggests nothing specific about the underlying OS
- Focus mainly on the RIM API that wraps MIDP

Network Architecture

- All BB connect to 'IM's central NOC through Carrier
- NOC connected to all BES on site
- BES can attach to additional middleware services



MDS (Mobile Data Services)

- MDS focuses mainly on web and enterprise services
 - MDS is a runtime container for processing pushed data
 - Ideal for Rapid Application Development
 - Minimal coding required (sometimes none)
- BB enterprise servers perform all the data processing and management
 - WSDL, Database, etc
- Can be arbitrarily complex on the backend

Application Structure

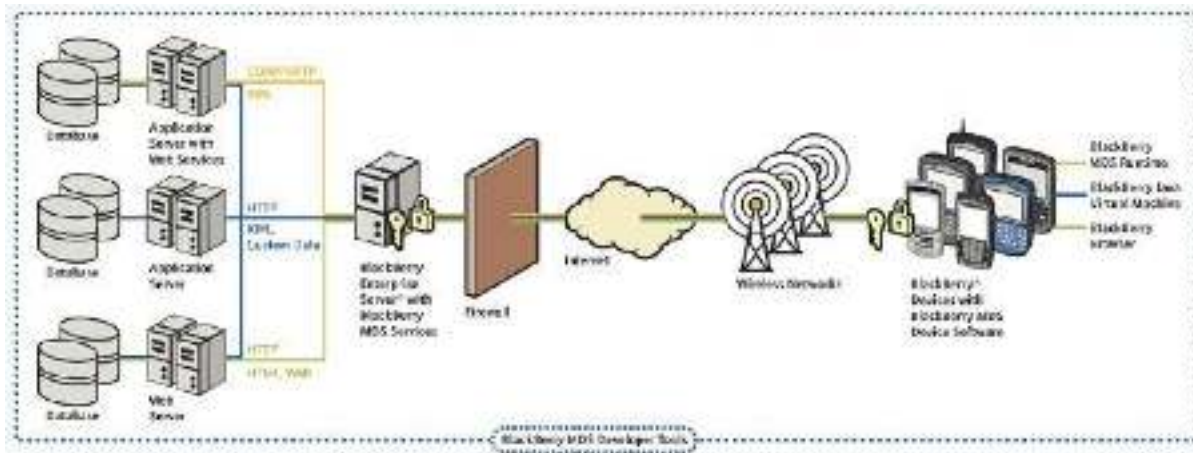
- JavaME applications like MIDlets contain
 - .jar and .jad files
- Converted to proprietary .cod files for BB
 - Compiled code (optionally signed) and preverified
 - Raps tool converts .jar .jad to .cod
- .alx files used to load applications via the BlackBerry Desktop Manager software
 - XML-based BlackBerry application descriptor

Over the Air (OTA) Deployment

- Standard MIDlets and .cod files can be obtained over the air (OTA)
- Provider puts both a .jad file and either a .cod or a .jar file
- Select the .jad file from a browser
- MDS server feature provides a built-in transcoder that converts .jar files into .cod files.

MDS Applications

- MDS is the simplest to develop for the client side
 - Arbitrarily complex on the server end
- Considered “Browser Based”
 - Essentially laying out html forms
 - All data is received in expected form
 - Can be developed into a browse-able page (cHTML)
- MDS Server defines a WSDL or SQL-DB Schema
 - MDS uses standard wrappers to call these accessors
- MDS apps use the compression and encryption features to securely send data through the cell carriers



Security Model in Brief

- Class files verified for interface compliance
- Limited API set (CLDC)
- Downloading and management within the JVM
- No user-defined class loaders
- No Java Native Interface or user extensions
- System classes cannot be overridden.

Sensitive APIs

- Use of the BlackBerry and RIM API is restricted
 - Tracked for security and export reasons
 - Not in simulator
- The JVM checks for valid code signatures
 - Developer just sends code hashes to a webservice
 - Gets a RIM signed signature
 - Linktime verification
 - Runtime verification

Company History

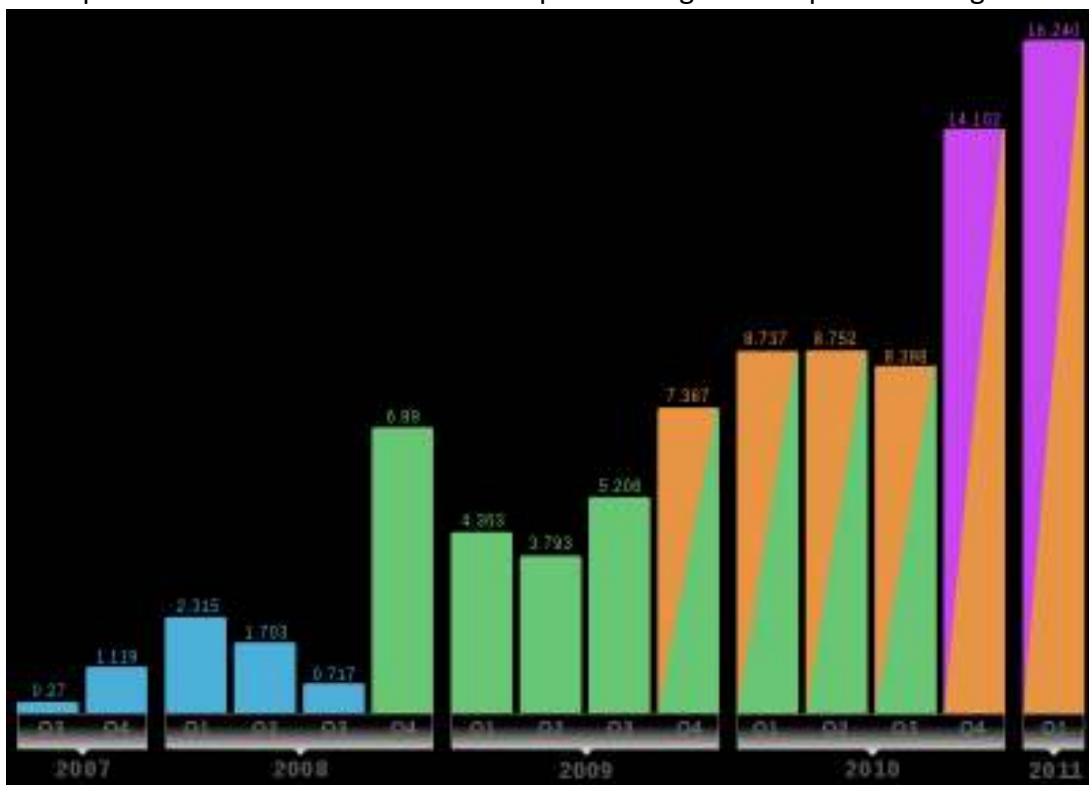
- April 1, 1976: Apple Computer, Inc. is founded (Apple I)
- Apple II, Apple III, Lisa
- 1984: Macintosh: mouse, GUI; first version of Apple's "ease of use"
- 1998: iMac: designed by J. Ive (designed iPod and iPhone too)

iPhone History

- 1999: Apple starts registering trademarks and domain names: iPhone.org, iPhone TM
- 2004: Apple and Motorola work on phone with iTunes
- Sept. 7, 2005: ROKR released
- Jan 9, 2007: iPhone announced, Apple Computer, Inc. becomes Apple, Inc.
- June 29, 2007: iPhone launched
- March 6, 2008: SDK
- Before that: only Web applications
- July 11, 2008: iPhone 3G, AppStore

Market Share and Predictions

- 12.9% in Q3 2008; one year before: 3.4%
- 3rd after Symbian (49.8%) and RIM (15.9%)
- Q4 2008: 6.9 million units sold; one year before: 1.1 million
- Prediction: 45 million total in 2009
- By the end of [fiscal year](#) 2010, a total of 73.5 million iPhones were sold
- By 2010/2011, the iPhone has a market share of barely 4% of all cellphones, but Apple still pulls in more than 50% of the total profit that global cellphone sales generate



Development Model

- Free (with registration) SDK available
 - Requires Intel-based Mac
- SDK allows code development and testing
 - Includes a simulator for the iPhone
- However, loading code onto an actual iPhone is not possible without joining the iPhone developer program

iPhone Developers Program

- iPhone Developer Program allows developers to load applications on the iPhone
- Developers can publish applications in the AppStore
 - Developer sets price
 - Can be free; minimum price \$0.99, maximum price \$999
 - Apple takes 30% of application cost for each copy sold
- Three different “levels” for the developer program

Development Tools

- SDK is based around Xcode
 - XCode is used to develop Mac OS X, iPhone OS, Apple applications
- Other included utilities
 - Instruments -- real time code profiling
 - Dashcode -- rapid development of widgets
 - Web apps for iPhone
 - Dashboard widgets for Mac OS X
 - Simulator -- used for testing
 - Interface Builder -- drag and drop interface creation tool

iPhone Applications

- Written in Objective-C
 - Superset of C
 - “Another object-oriented C-like language”
 - SmallTalk influence
 - “Messages” are sent to objects
- Event driven model
- Note: Even though the SDK is available, Apple does not release all API information
 - Things like CoverFlow and certain hardware APIs are not published

Applications

- AppStore provides over 20,000 different applications
 - Requires Apple approval to be published in the AppStore
 - Wide variety of applications available
- Everything from games and social networking to finance and business

Application Framework

- OS Architecture
 - Based on Darwin
- Darwin Information
 - Open Source
 - BSD-like system
 - Maintained by Apple and community
- Three different “variants”
 - PowerPC and Intel x86 (Mac OS X)
 - Released as open source
 - ARM (iPhone OS)
 - Not released

Darwin

- Darwin kernel is XNU
- XNU is a hybrid kernel based on the Mach 3 microkernel and various FreeBSD components
- Object-oriented device driver framework/API
 - I/O Kit
- Binary format: Mach-O
 - Mach-O is a binary format similar to ELF, a.out, EXE, COM, etc.

Mach-O

- File can contain binary code for multiple CPU architectures
 - For example, 32-bit (G4) and 64-bit (G5) PowerPC along with Intel x86.
- Known as a “Universal Binary”
- Seamless transition between architectures
- Rosetta (used during PowerPC to Intel switch)

iPhone Runtime

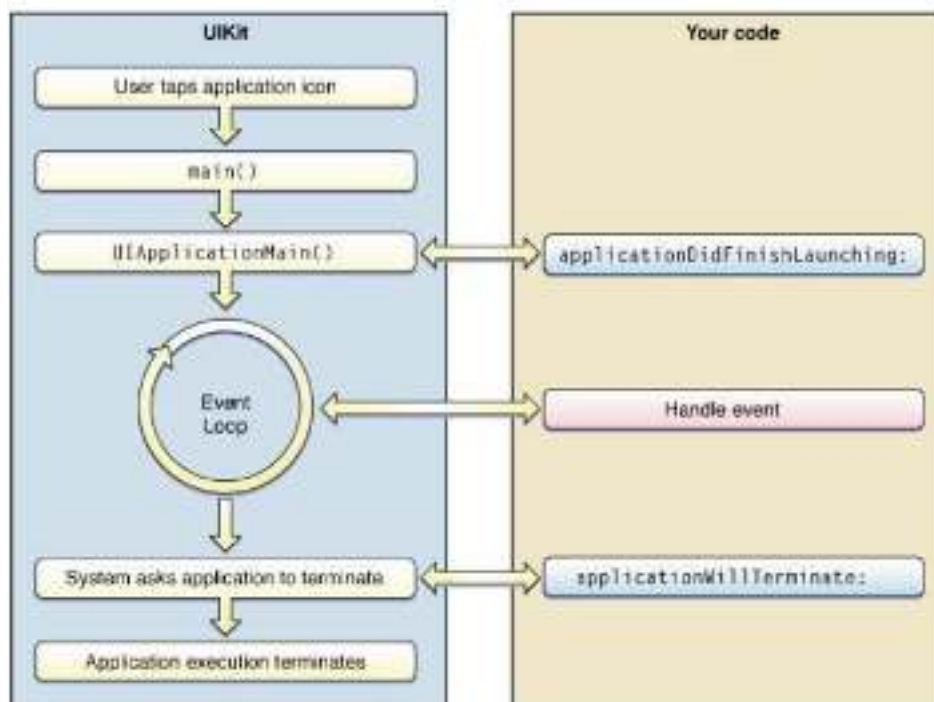
- Model different from PC
 - Short (“bursty”) usage
 - Single foreground application
- Currently no “background” processes allowed
- This really only applies to 3rd party app developers
- Apple has released guidelines on writing applications that fit this usage model
- App is notified of pending termination and is advised to save state
- Applications are sandboxed
- Applications run in their own virtual address space
 - No swapping is done
 - Event is sent to app to notify it to free some memory
 - If more memory is still needed, application can be killed by OS

Application Structure

- Distribution in bundle
 - Executable file
 - png files
 - .nib files
 - Info.plist
 - .lproj

Application Lifecycle

- Applications are based on event handling



MVC Design Pattern

- View: user interface
- Model: engine
- Controller: link between the two

Objective-C

- Object-oriented superset of C
 - Different from C++
- Some nice features
 - Automatic generation of setters and getters according to properties
- Some difficulties
 - The message-passing system...

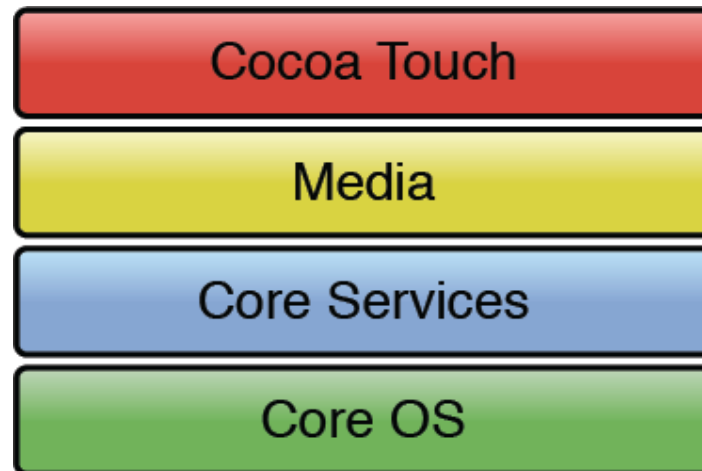
Memory Management

- No garbage collector
- Instead, reference count
 - Alloc
 - Release
 - Retain
 - Autorelease
 - copy

Lecture 35

Frameworks and APIs

- iPhone OS based on a layered architecture
 - Four basic layers



Core OS Layer

- Three basic frameworks at this layer
 - CFNetwork -- APIs related to networking
 - Sockets, FTP, HTTP, Bonjour, etc.
 - Security
 - Certificate handling, random number generation, crypto related functions
 - System
 - BSD and POSIX related functions

Cross Layer Services

- Address Book
 - Contact information used by things like SMS and Phone applications
- Core Foundation
 - Access to basic data structures like strings and other basic system functions
- Core Location
 - Location based information (GPS access)
- Foundation
 - Base for all Objective-C objects like the root NSObject class, NSString, and NSArray for example
- System Configuration

Media Layer

- Audio Toolbox
- Audio Unit
- AV Foundation
- Core Audio
- Core Graphics
- Media Player
- OpenGL ES
- Quartz Core

Cocoa Touch Layer

- Address Book UI
 - UI for the Address Book
- UIKit
 - All user interface components

Undocumented APIs

- Not all APIs are documented
 - CoverFlow, hardware (proximity sensor for example)
 - Using undocumented APIs is a reason for being rejected from the AppStore
 - Google uses such APIs in their voice search application
 - Accepted into AppStore
 - Eric Schmidt (CEO, Google) sits on Apple's board of directors

System Protection

- Sandboxed applications
- Code signing
 - Applications signed by Apple
 - Developer's certificate signed with Apple's root certificate used for development and testing
 - Code reviewed by Apple before publication in the App Store
 - Review process: mystery, but seems thorough
 - Jailbreaking circumvents this protection

JailBreaking

- Two firmwares to modify with different results
 - Application processor firmware modification to allow unsigned code to run (i.e. applications from outside the AppStore)
 - This is jailbreaking
 - Baseband processor firmware modification to allow carriers other than the intended one
 - This is SIM unlocking
- Similar to other OSes, the iPhone OS is loaded in stages

- Boot ROM loads the LLB
 - LLB loads the Firmware
 - At each stage, signature checks performed to validate the next stage
- In theory...
- The boot ROM does not perform a signature check on the LLB
 - Oops!
- More recently, the LLB is subject to a buffer overflow which allows unsigned code to be loaded that can override the signature checks of *all* subsequent checks.

SIM Unlocking

- Jailbreaking is only half the battle.
- Users still tied to a specific carrier
- Jailbreaking can lead to more vulnerabilities...

Android Development Framework

- Android applications are written with Java as a programming language but executed by means of a custom virtual machine called Dalvik rather than a traditional Java VM.
- Each Android application runs in a separate process within its own Dalvik instance, relinquishing all responsibility for memory and process management to the Android run time, which stops and kills processes as necessary to manage resources.
- Dalvik and the Android run time sit on top of a Linux kernel that handles low-level hardware interaction, including drivers and memory management, while a set of APIs provides access to all the underlying services, features, and hardware.

Dalvik Virtual Machine

- One of the key elements of Android is the Dalvik virtual machine.
- The Dalvik VM uses the device's underlying Linux kernel to handle low-level functionality including security, threading, and process and memory management. It's also possible to write C/C++ applications that run directly on the underlying Linux OS.

Android Software Development Kit (SDK)

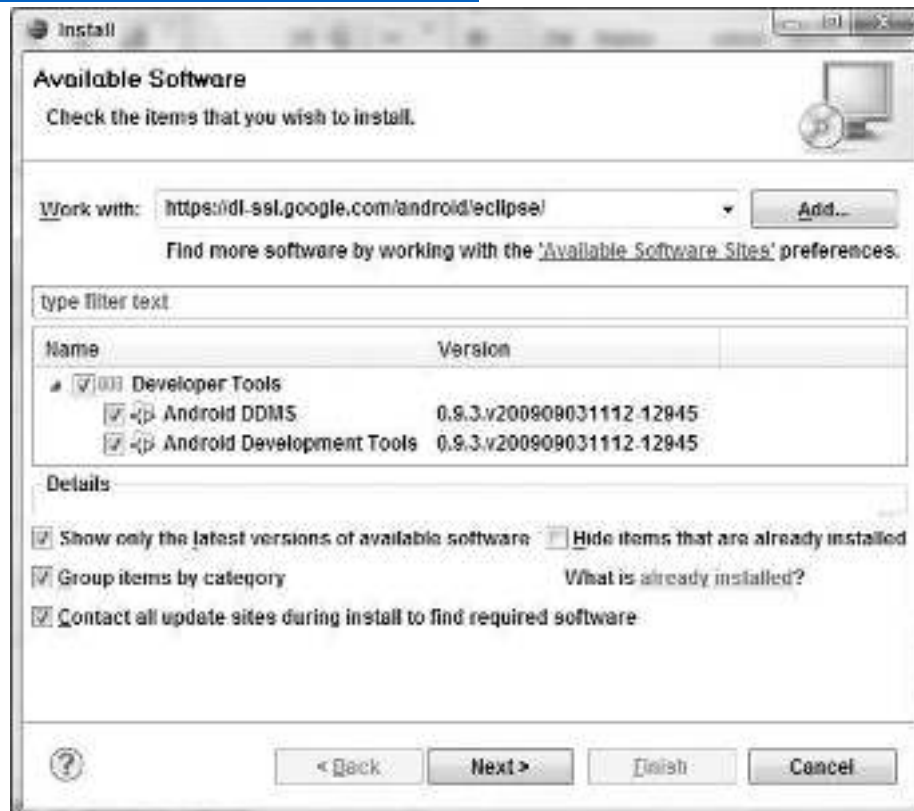
- The Android software development kit (SDK) includes everything you need to start developing, testing, and debugging Android applications.
 - The Android APIs
 - Development Tools
 - Android Virtual Device Manager and Emulator
 - Full Documentation
 - Sample Code

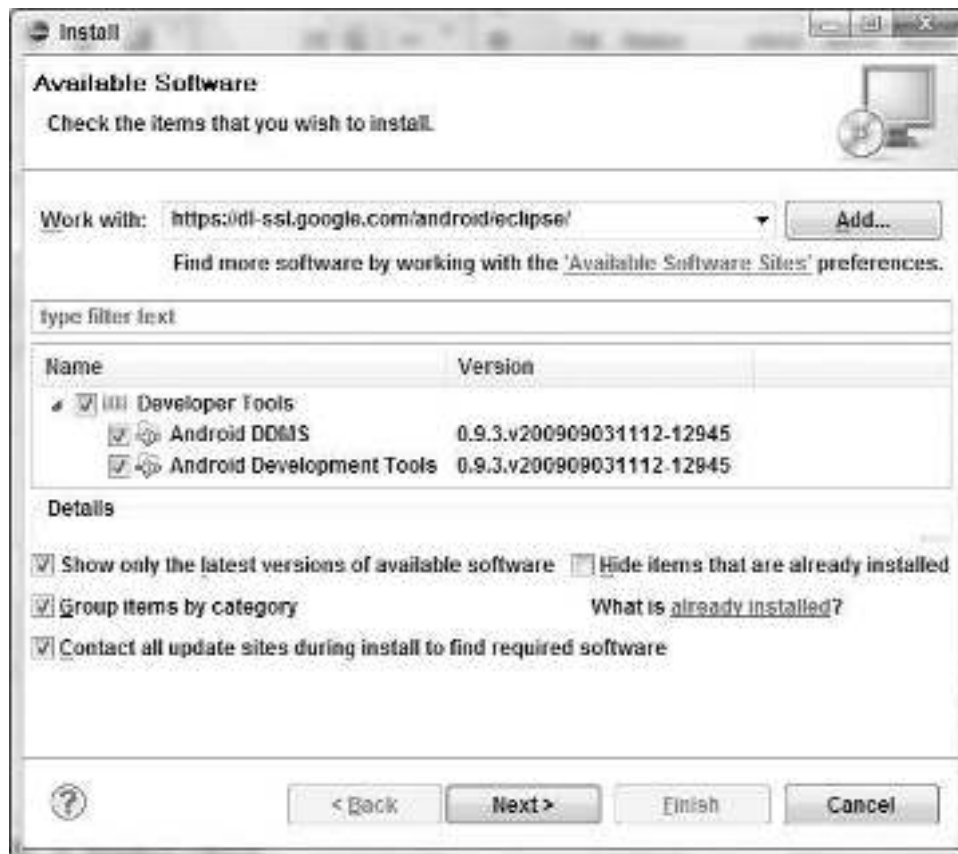
Application Development Environment

- Android SDK and
- The Java development kit.
- IDE of Choice – Eclipse
- Versions of the SDK, Java, and Eclipse are available for Windows, MacOS, and Linux

Setting Up the Environment

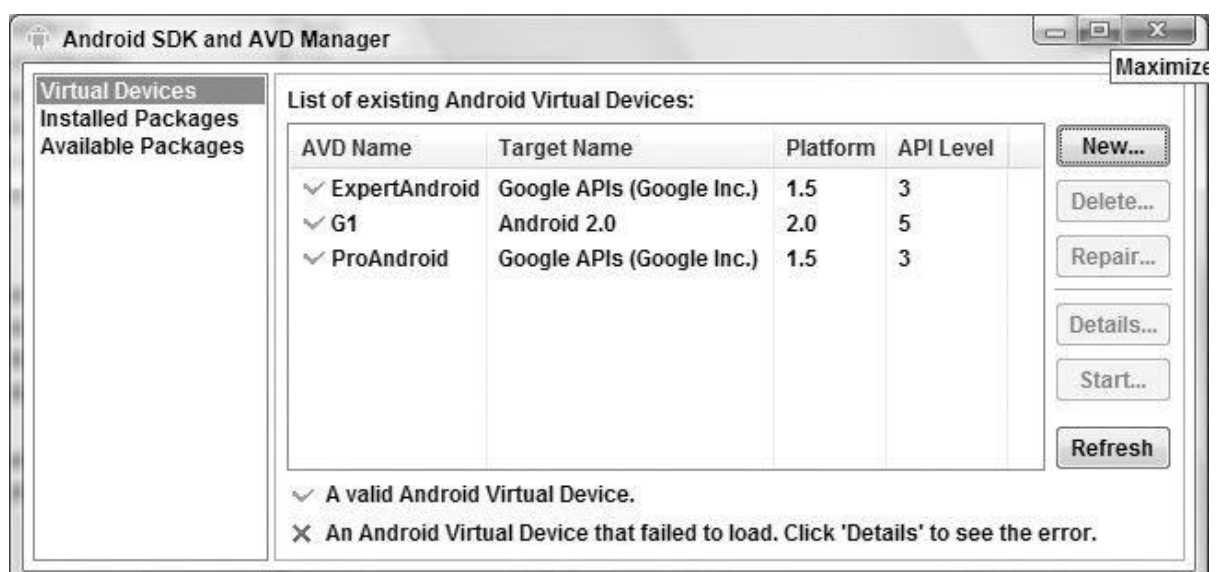
- Download Eclipse
- Download ADT
 - <https://dl-ssl.google.com/android/eclipse/>

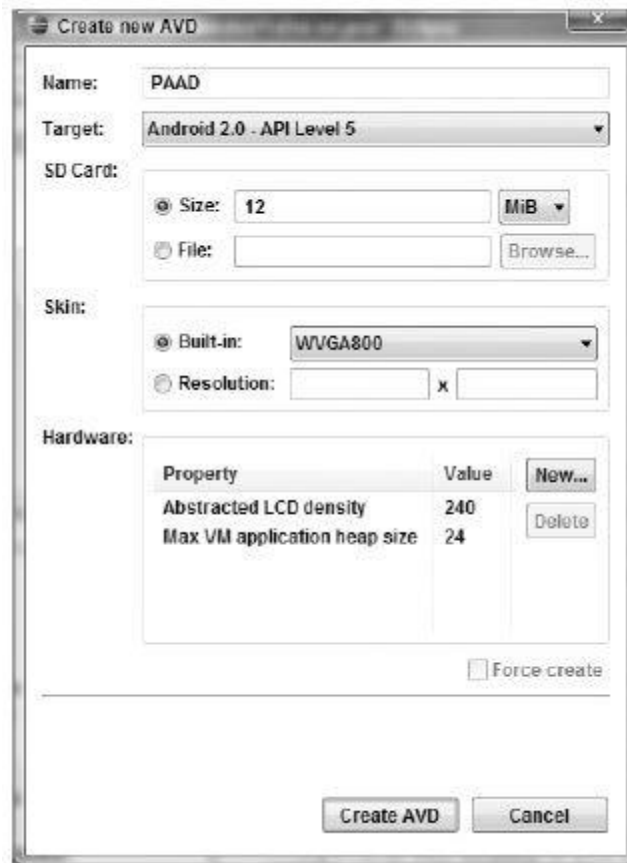




Create an AVD

- Maximum virtual machine heap size
- Screen pixel density
- SD Card support
- The existence of DPad, touchscreen, keyboard, and trackball hardware
- Accelerometer and GPS support
- Available device memory
- Camera hardware (and resolution)

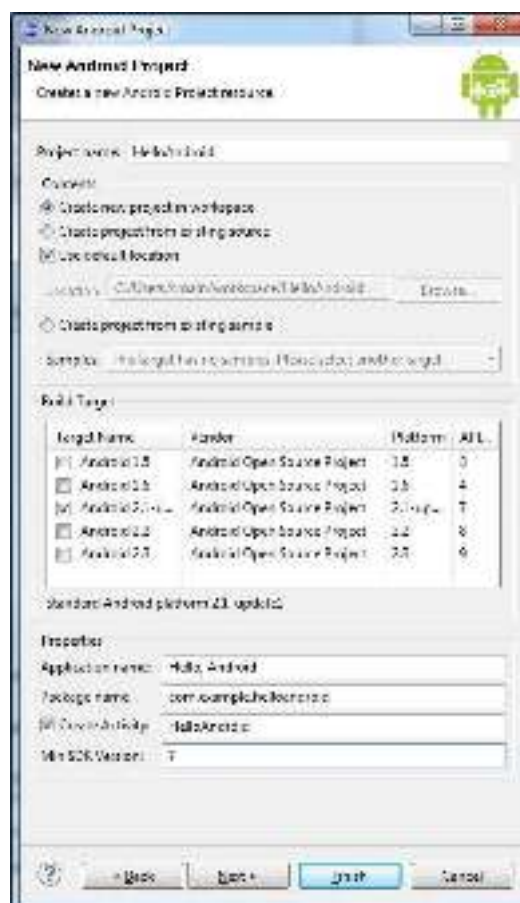




Emulator



Creating First Android Application



Lecture 36

Android Development Framework

- Android applications are written with Java as a programming language but executed by means of a custom virtual machine called Dalvik rather than a traditional Java VM.
- Each Android application runs in a separate process within its own Dalvik instance, relinquishing all responsibility for memory and process management to the Android run time, which stops and kills processes as necessary to manage resources.

Dalvik Virtual Machine

- One of the key elements of Android is the Dalvik virtual machine.
- The Dalvik VM uses the device's underlying Linux kernel to handle low-level functionality including security, threading, and process and memory management. It's also possible to write C/C++ applications that run directly on the underlying Linux OS.

Android Software Development Kit (SDK)

- The Android software development kit (SDK) includes everything you need to start developing, testing, and debugging Android applications.
 - The Android APIs
 - Development Tools
 - Android Virtual Device Manager and Emulator
 - Full Documentation
 - Sample Code

Application Development Environment

- Android SDK and
- The Java development kit.
- IDE of Choice – Eclipse
- Versions of the SDK, Java, and Eclipse are available for Windows, MacOS, and Linux

Hello World

```
package com.example.helloworld;
import android.app.Activity;
import android.os.Bundle;
public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Views

@Override

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Defining UI – XML Way

- UI layout defined in the main.xml file created by the Android project template

```
<?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">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello World, HelloWorld"
    />
</LinearLayout>
```

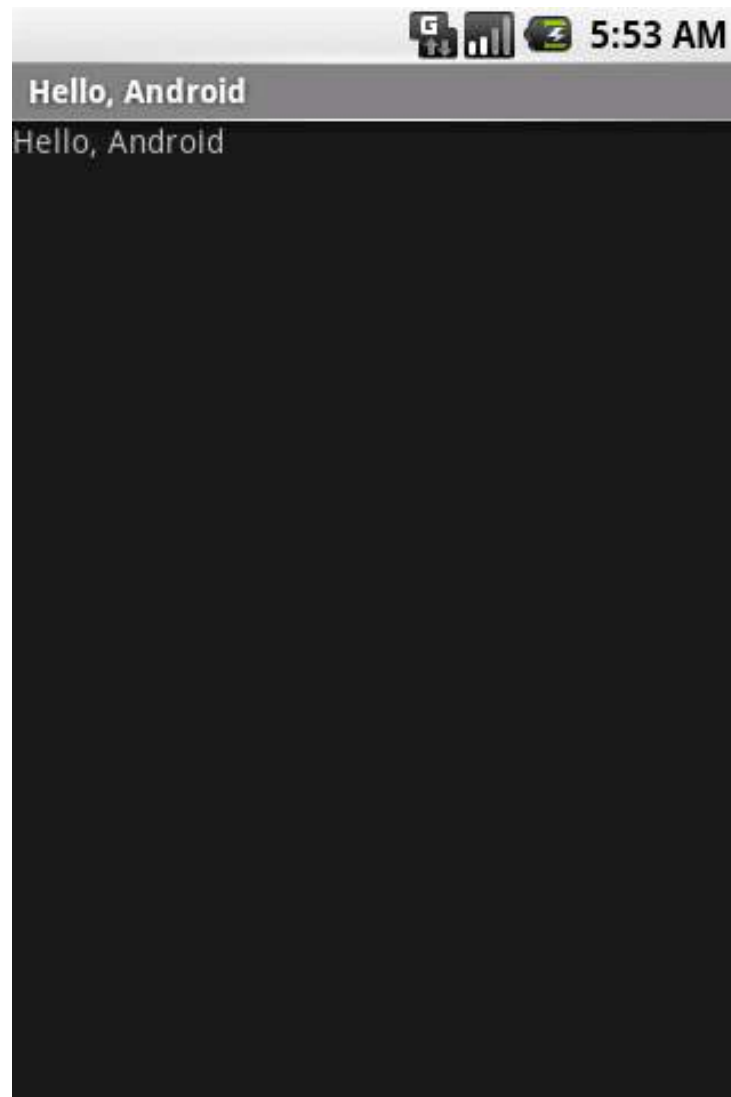


Defining UI – Coding Way

```
package com.example.helloandroid;

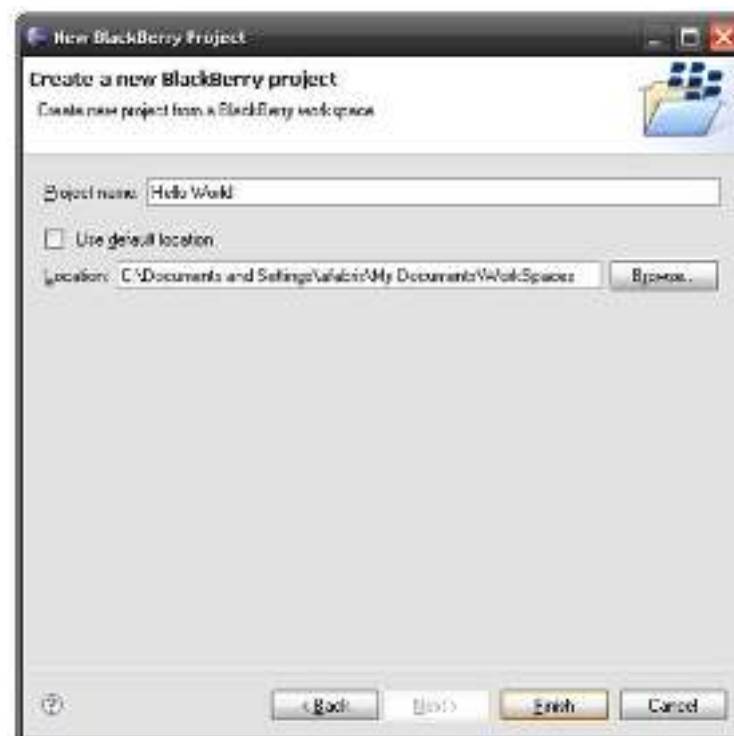
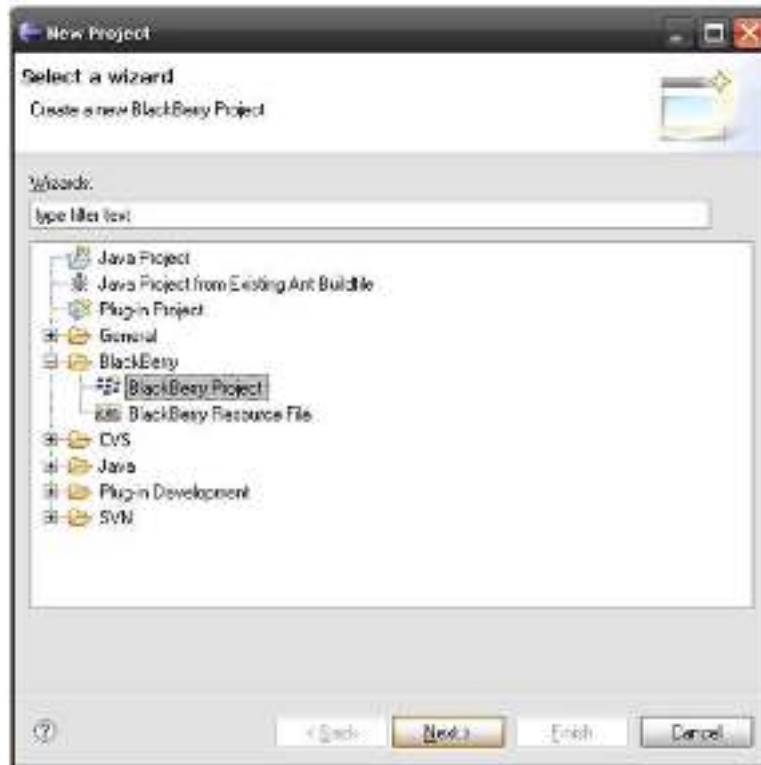
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

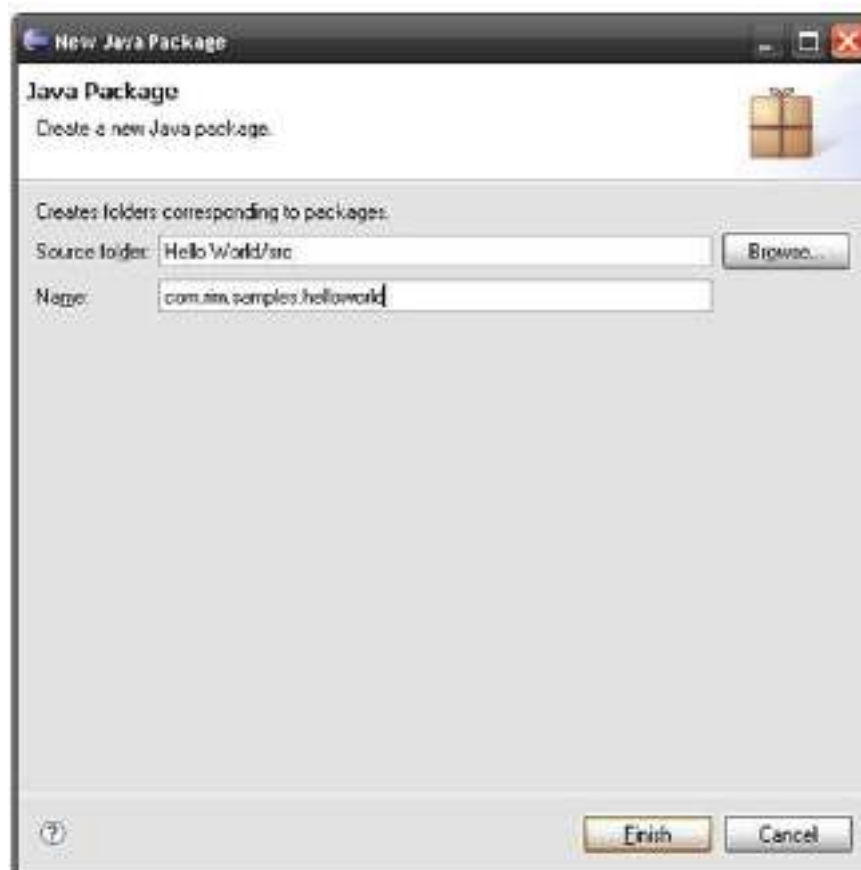
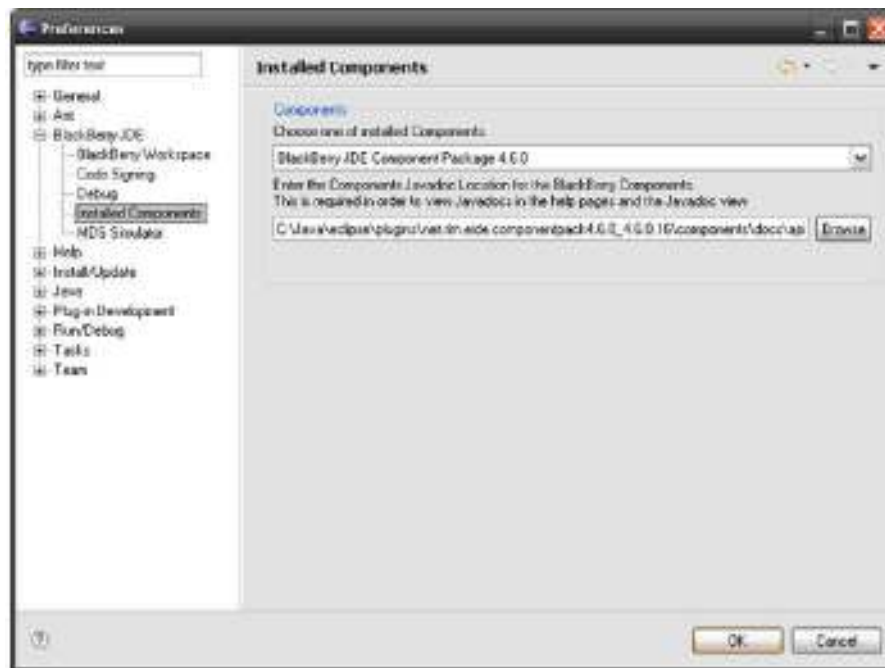
public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

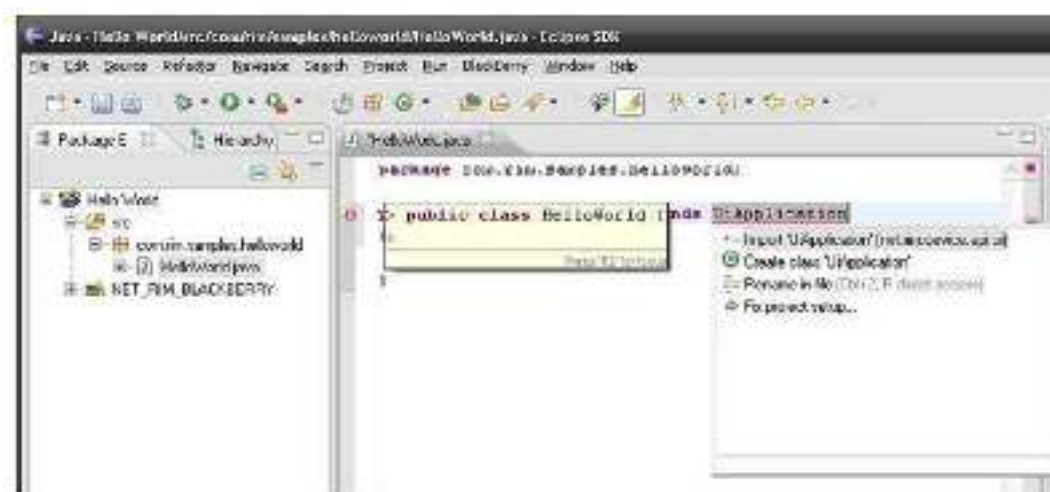
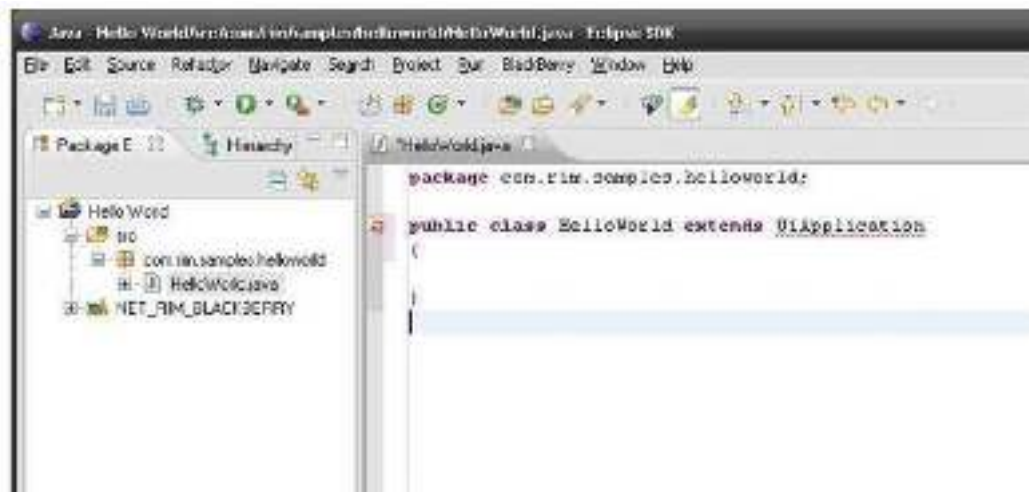


Hello World – BlackBerry Style

- Sun JDK
- Eclipse SDK,
- BlackBerry JDE Plug-in for Eclipse and
- BlackBerry JDE Component Packs 4.3 – 4.7







```
package com.rim.samples.helloworld;
import net.rim.device.api.ui.UiApplication;
public class HelloWorld extends UiApplication
{
    public static void main(String[] args)
    {
        HelloWorld theApp = new HelloWorld();
        theApp.enterEventDispatcher();
    }
    public HelloWorld()
    {
        //display a new screen
        pushScreen(new HelloWorldScreen());
    }
}
```

```

final class HelloWorldScreen extends MainScreen
{
    public HelloWorldScreen()
    {
        super();
        LabelField title = new LabelField("HelloWorld Sample",
        LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Hello World!"));
    }
    public boolean onClose()
    {
        Dialog.alert("Goodbye!");
        System.exit(0);
        return true;
    }
}

```

```

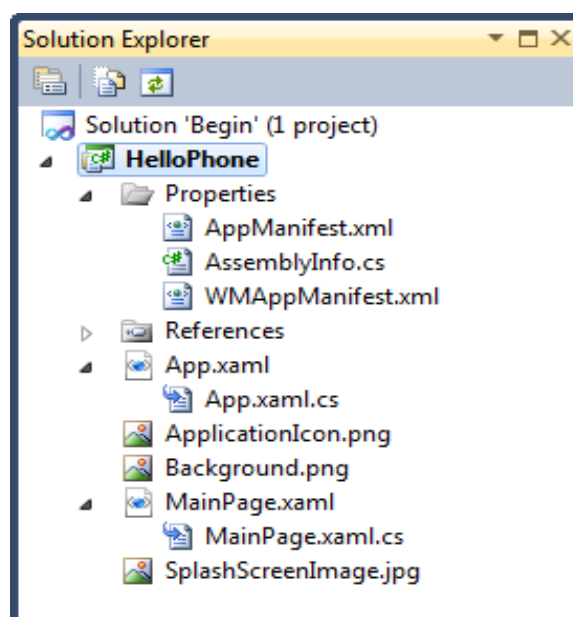
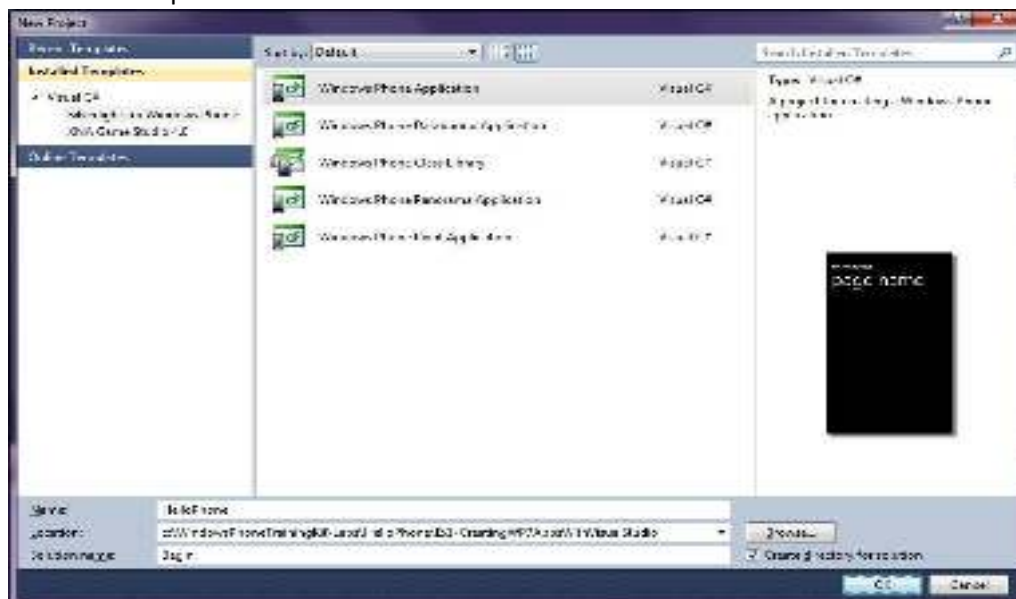
final class HelloWorldScreen extends MainScreen
{
    public HelloWorldScreen()
    {
        super();
        LabelField title = new LabelField("HelloWorld Sample",
        LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Hello World!"));
    }
    public boolean onClose()
    {
        Dialog.alert("Goodbye!");
        System.exit(0);
        return true;
    }
}

```



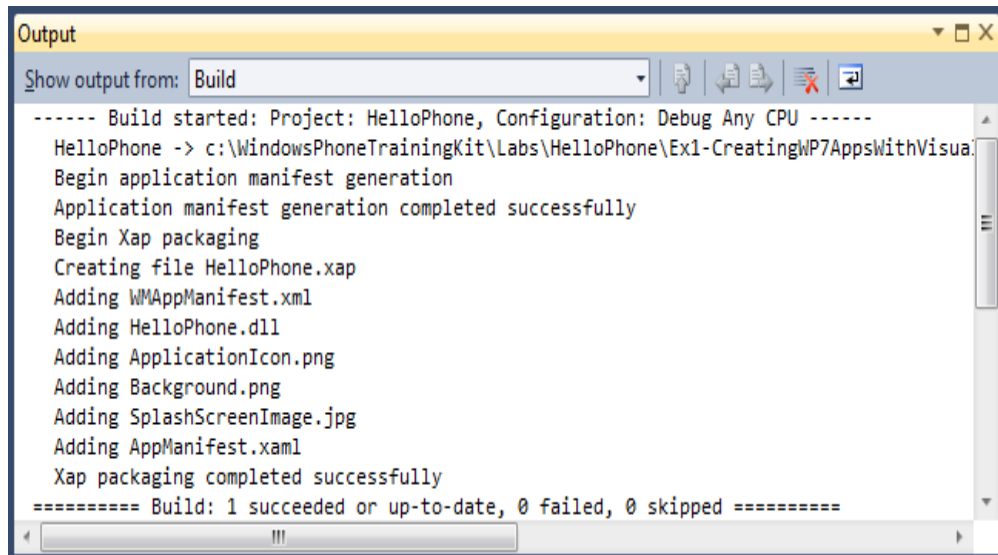
Hello World – WP7 Style

- Microsoft Visual Studio
 - Microsoft Visual Studio 2010 Express for Windows Phone.
- Windows Phone Developer Tools
 - <http://developer.windowsphone.com>
 - Windows Phone SDK 7.1
 - Visual Studio 2010 Express for Windows Phone
 - Windows Phone Emulator Resources
 - Silverlight 4 Tools For Visual Studio
 - XNA Game Studio 4.0
 - Microsoft Expression Blend for Windows Phone









```

Output
Show output from: Build
----- Build started: Project: HelloPhone, Configuration: Debug Any CPU -----
HelloPhone -> c:\WindowsPhoneTrainingKit\Labs\HelloPhone\Ex1-CreatingWP7AppsWithVisua
Begin application manifest generation
Application manifest generation completed successfully
Begin Xap packaging
Creating file HelloPhone.xap
Adding WAppManifest.xml
Adding HelloPhone.dll
Adding ApplicationIcon.png
Adding Background.png
Adding SplashScreenImage.jpg
Adding AppManifest.xaml
Xap packaging completed successfully
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
  
```





XAML

```

...
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*/>
  </Grid.RowDefinitions>
  ...

  </Grid>
</phone:PhoneApplicationPage>

```

```

<phone:PhoneApplicationPage
  x:Class="HelloPhone.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
  xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
  FontFamily="{StaticResource PhoneFontFamilyNormal}"
  FontSize="{StaticResource PhoneFontSizeNormal}"
  Foreground="{StaticResource PhoneForegroundBrush}"
  SupportedOrientations="Portrait" Orientation="Portrait"
  shell:SystemTray.IsVisible="True">

  <!--LayoutRoot is the root grid where all page content is placed-->
  <Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
      <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION" Style="{StaticResource PhoneTextNormal}" />
      <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0" Style="{StaticResource PhoneTextNormal}" />
    </StackPanel>

    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Grid>
  </Grid>

```

XAML

```

...
<Grid x:Name="LayoutRoot" Background="Transparent">
  ...
  <!--ContentPanel - place additional content here-->
  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    <TextBox Grid.Column="0" Name="MessageTextBox" FontSize="{StaticResource PhoneFontSizeExtraLarge}" Margin="20,20,10,20"/>
    <Button Grid.Column="1" Name="ClickMeButton" Content="Click Me"
      HorizontalAlignment="Right" Padding="4" Margin="10,20,20,20" />
  </Grid>

```

XAML

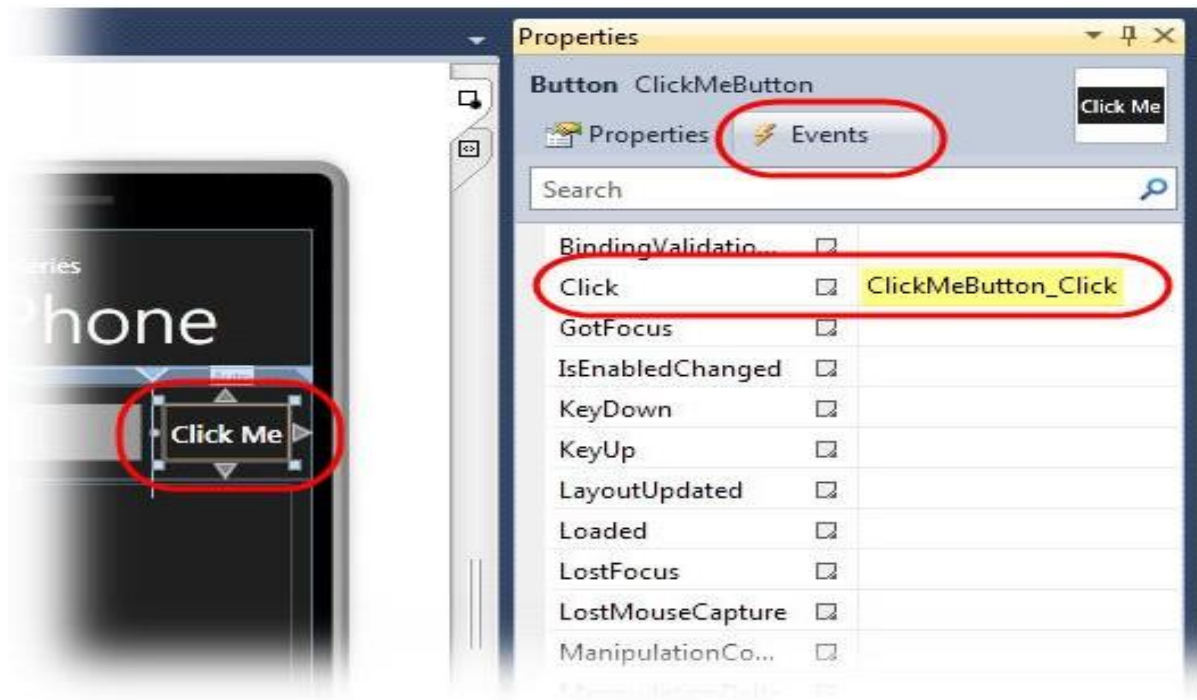
```

...
<Grid x:Name="LayoutRoot" Background="Transparent">
...

<Grid Grid.Row="2">
  <TextBlock Name="BannerTextBlock" Style="{StaticResource PhoneTextExtraLargeStyle}"
    Foreground="#FFFF9A00" HorizontalAlignment="Stretch"
    TextWrapping="Wrap" TextAlignment="Center" FontWeight="Bold" />
</Grid>
</Grid>

```





```

<!--This section is empty. Place new content here Grid.Row="1"-->
<Grid Grid.Row="1">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    <TextBox Grid.Column="0" Name="MessageTextBox" FontSize="{StaticResource PhoneFontSize24}" />
    <Button Grid.Column="1" Name="ClickMeButton" Content="Click Me"
        HorizontalAlignment="Right" Padding="4" Margin="10,20,20,20"
        Click="ClickMeButton_Click" />
</Grid>
<Grid Grid.Row="2">

```

C#

```

private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
    BannerTextBlock.Text = MessageTextBox.Text;
    MessageTextBox.Text = String.Empty;
}

```

```
namespace HelloPhone
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }

        private void ClickMeButton_Click(object sender, RoutedEventArgs e)
        {
            BannerTextBlock.Text = MessageTextBox.Text;
            MessageTextBox.Text = String.Empty;
        }
    }
}
```



Types of Android Applications

- Foreground
- Background
- Intermittent
- Widget

Building Blocks of Android App

- Activities
- Services
- Content Providers
- Intents
- Broadcast Receivers
- Widgets
- Notifications

Application Manifest

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
package="com.my_domain.my_app"
android:versionCode="1"
android:versionName="0.9 Beta">
[ ... manifest nodes ... ]
</manifest>
```

```
<uses-sdk android:minSdkVersion="4"
android:targetSdkVersion="5">
</uses-sdk>
<uses-configuration android:reqTouchScreen=["finger"]
android:reqNavigation=["trackball"]
android:reqHardKeyboard=["true"]
android:reqKeyboardType=["qwerty"/>
<uses-configuration android:reqTouchScreen=["finger"]
android:reqNavigation=["trackball"]
android:reqHardKeyboard=["true"]
```

```
<uses-feature android:glEsVersion=" 0x00010001"
android:name="android.hardware.camera" />
<supports-screens android:smallScreens=["false"]
android:normalScreens=["true"]
android:largeScreens=["true"]
android:anyDensity=["false"] />
```

```

<application android:icon="@drawable/icon"
    android:theme="@style/my_theme"
    android:name="MyApplication"
    android:debuggable="true">
    [ ... application nodes ... ]
</application>
<activity android:name=".MyActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<service android:enabled="true" android:name=".MyService"></service>
<provider android:permission="com.paad.MY_PERMISSION"
    android:name=".MyContentProvider"
    android:enabled="true"
    android:authorities="com.paad.myapp.MyContentProvider">
</provider>
<receiver android:enabled="true"
    android:label="My Intent Receiver"
    android:name=".MyIntentReceiver">
</receiver>

<uses-permission android:name="android.permission.ACCESS_LOCATION"/>

```

Location Based Services

- Your application knows how to say Hello, but it doesn't know where it's located.
- Now is a good time to become familiar with some simple location-based calls to get the GPS coordinates.
- Problem is that emulator does not have GPS sensors!!
- The emulator does not have location sensors, so the first thing you need to do is seed your emulator with GPS coordinates



```
import android.location.Location;
import android.location.LocationManager;
....
public void getLocation() {
    try {
        LocationManager locMgr = (LocationManager)
            getSystemService(LOCATION_SERVICE);
        Location recentLoc = locMgr.
            getLastKnownLocation(LocationManager.GPS_PROVIDER);
        Log.i(DEBUG_TAG, "loc: " + recentLoc.toString());
    }
    catch (Exception e) {
        Log.e(DEBUG_TAG, "Location failed", e);
    }
}
```

- Now try to run the application
- It generates an error

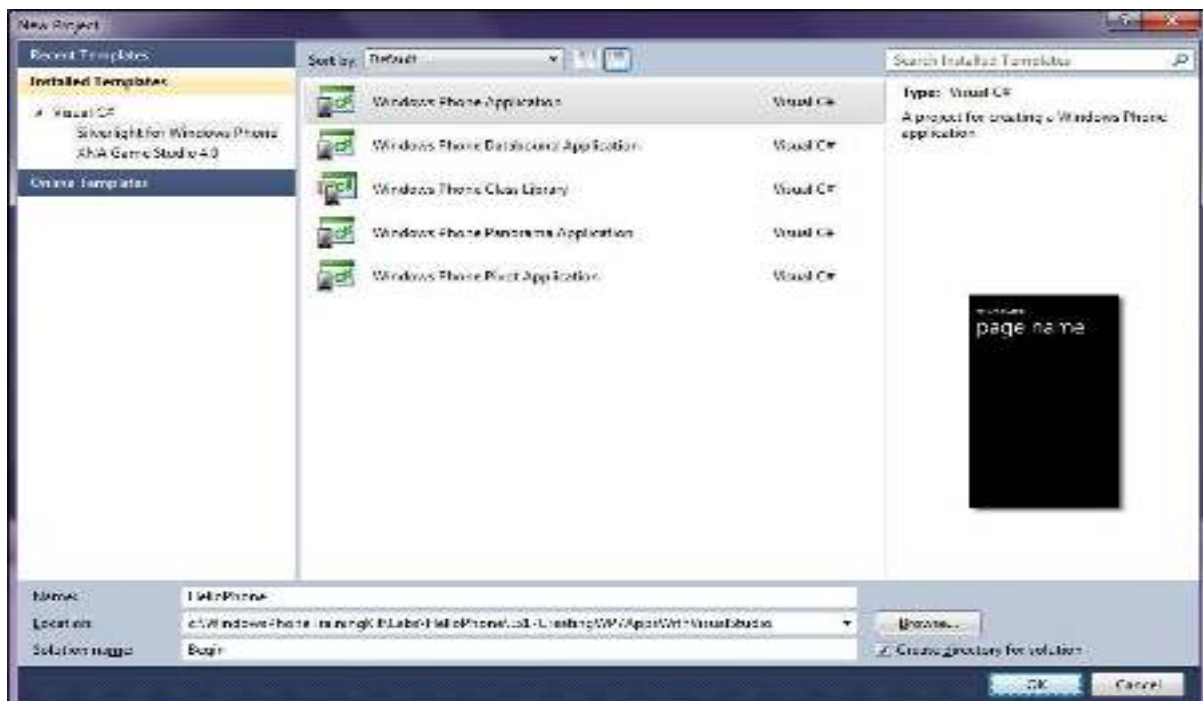
Manifest Permissions

- [ACCESS MOCK LOCATION](#) Allows an application to create mock location providers for testing
- [ACCESS_NETWORK_STATE](#) Allows applications to access information about networks
- [BATTERY_STATS](#) Allows an application to collect battery statistics
- [BLUETOOTH](#) Allows applications to connect to paired bluetooth devices
- [CALL_PHONE](#) Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed.
- [CAMERA](#) Required to be able to access the camera device.
- [INTERNET](#) Allows applications to open network sockets.

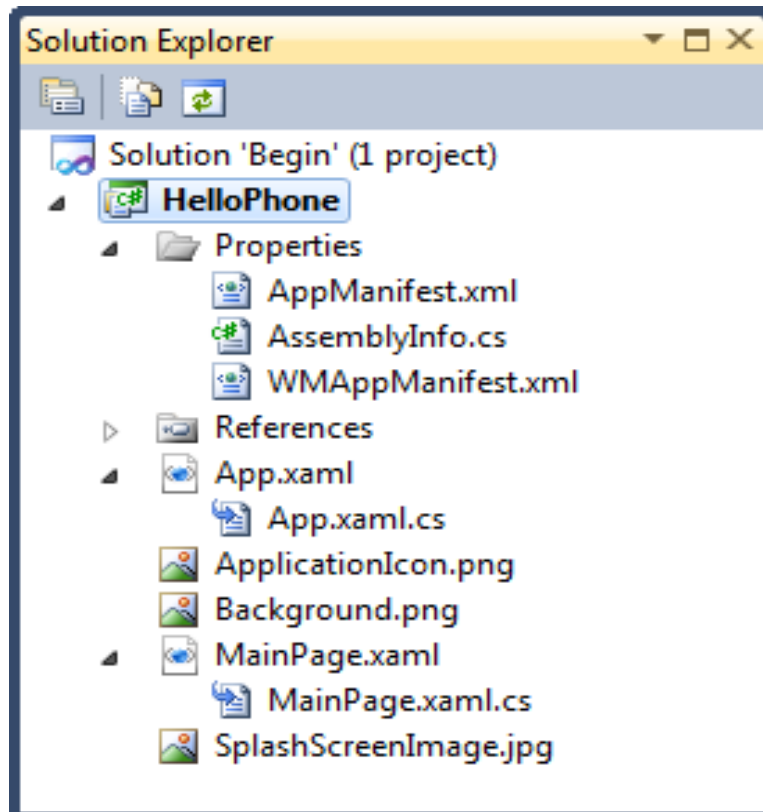
Lecture 37

Hello World – WP7 Style

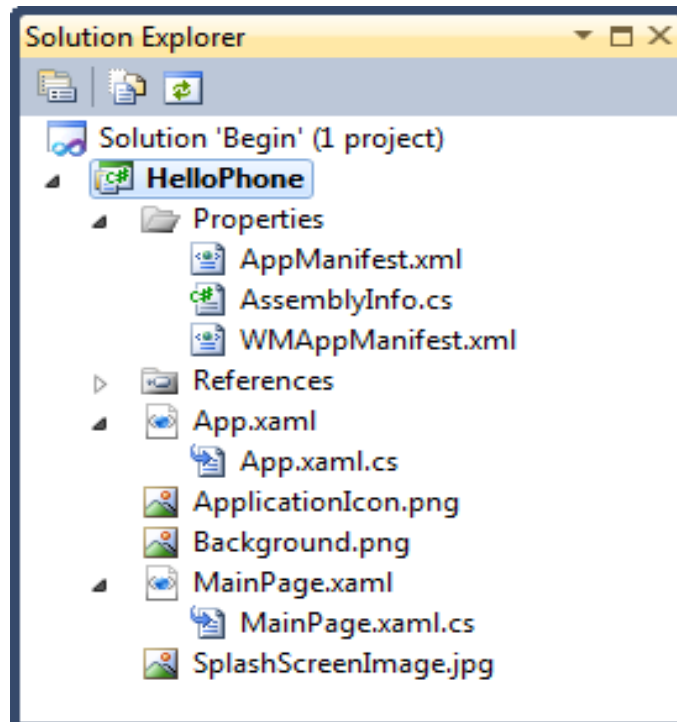
- Microsoft Visual Studio
 - Microsoft Visual Studio 2010 Express for Windows Phone.
- Windows Phone Developer Tools
 - <http://developer.windowsphone.com>
 - Windows Phone SDK 7.1
 - Visual Studio 2010 Express for Windows Phone
 - Windows Phone Emulator Resources
 - Silverlight 4 Tools For Visual Studio
 - XNA Game Studio 4.0
 - Microsoft Expression Blend for Windows Phone



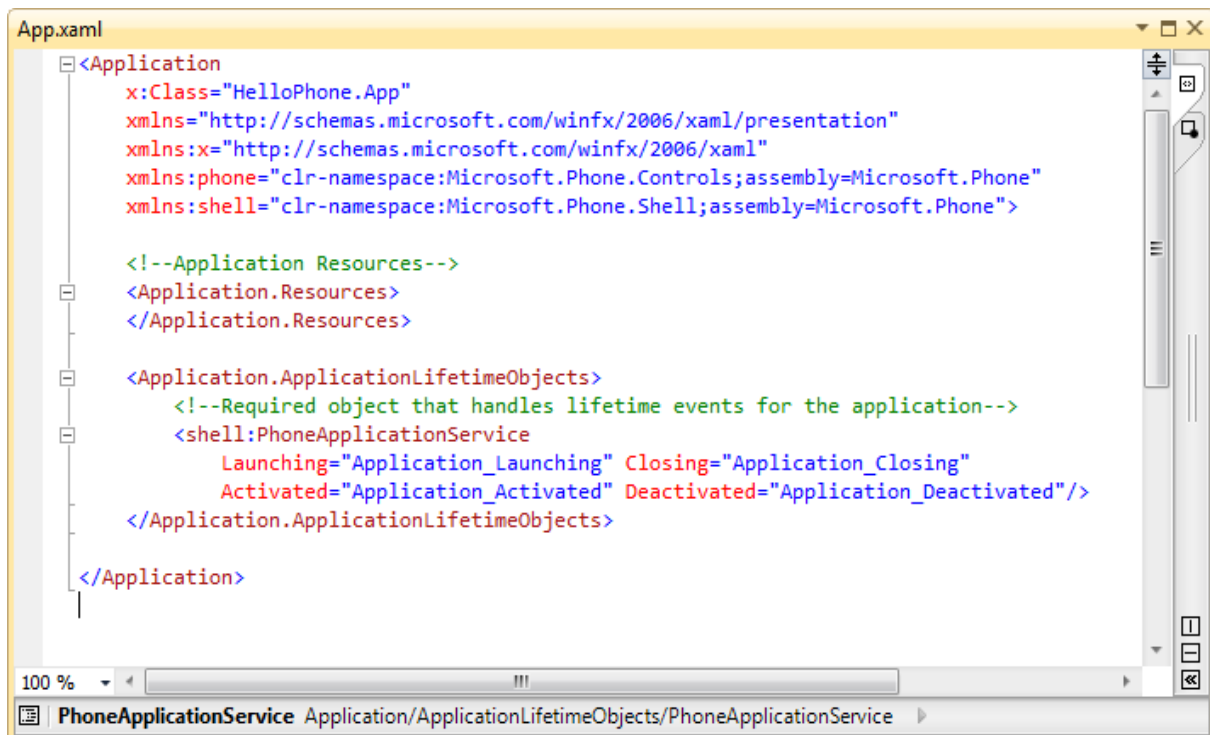
- In the **File** menu, choose **New Project**.
- In the **New Project** dialog, select the **Silverlight for Windows Phone** category in the list of installed templates, and there the Windows Phone Application template. Then set the name to **HelloPhone** and the location to **Ex1-CreatingWP7AppsWithVisualStudio** in the **Source** folder of the lab. Change the solution name to **Begin**, and then click **OK**.



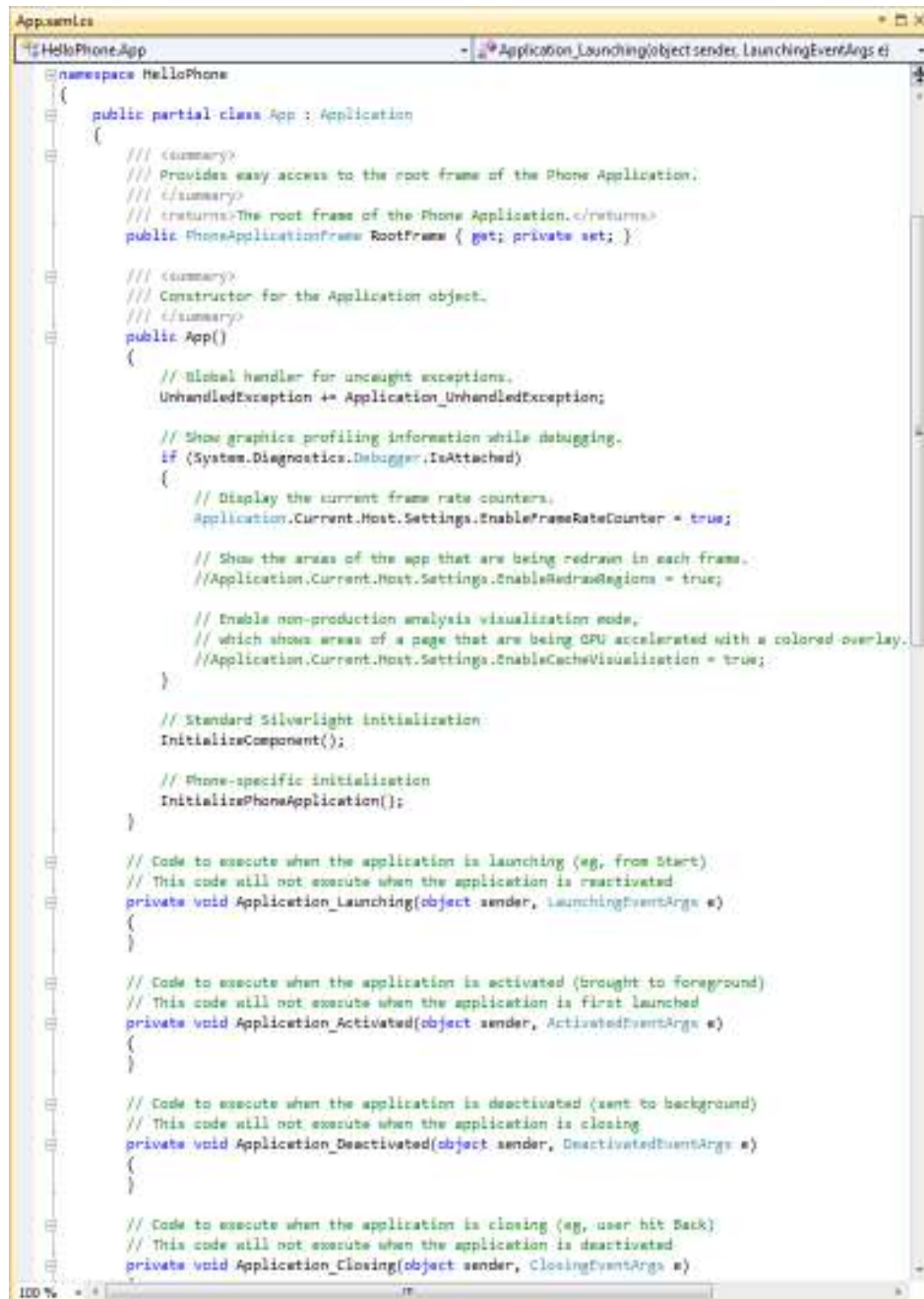
- In **Solution Explorer**, review the structure of the solution generated by the Windows Phone Application template. Any Visual Studio solution is a container for related projects; in this case, it contains a single Silverlight for Windows Phone project named **HelloPhone**.
- **App.xaml / App.xaml.cs** : Defines the entry point of the application, initializes application-scoped resources, and displays the application user interface
- **MainPage.xaml / MainPage.xaml.cs** : Defines a page with the user interface of the application
- **ApplicationIcon.png**: An image file with an icon that represents the application icon in the phone's application list
- **Background.png**: An image file with an icon that represents the application icon in the start screen



- **SplashScreenImage.jpg:** This is the image that will first be displayed when the application launches. The splash screen gives the user immediate feedback that the application is launching and will remain displayed until the navigation to the first page has been completed. Your splash screen can look similar to your first page in order to give the appearance that the application is loading quickly.
- **Properties\AppManifest.xml:** An application manifest file required to generate the application package
- **Properties\AssemblyInfo.cs:** Contains the name and version metadata that is embedded into the generated assembly
- **Properties\WMAppManifest.xml:** A manifest file that includes specific metadata related to a Windows Phone Silverlight application, including specific features available only for Silverlight for Windows Phone
- **References folder:** A list of libraries (assemblies) that provide services and functionality that the application requires to work



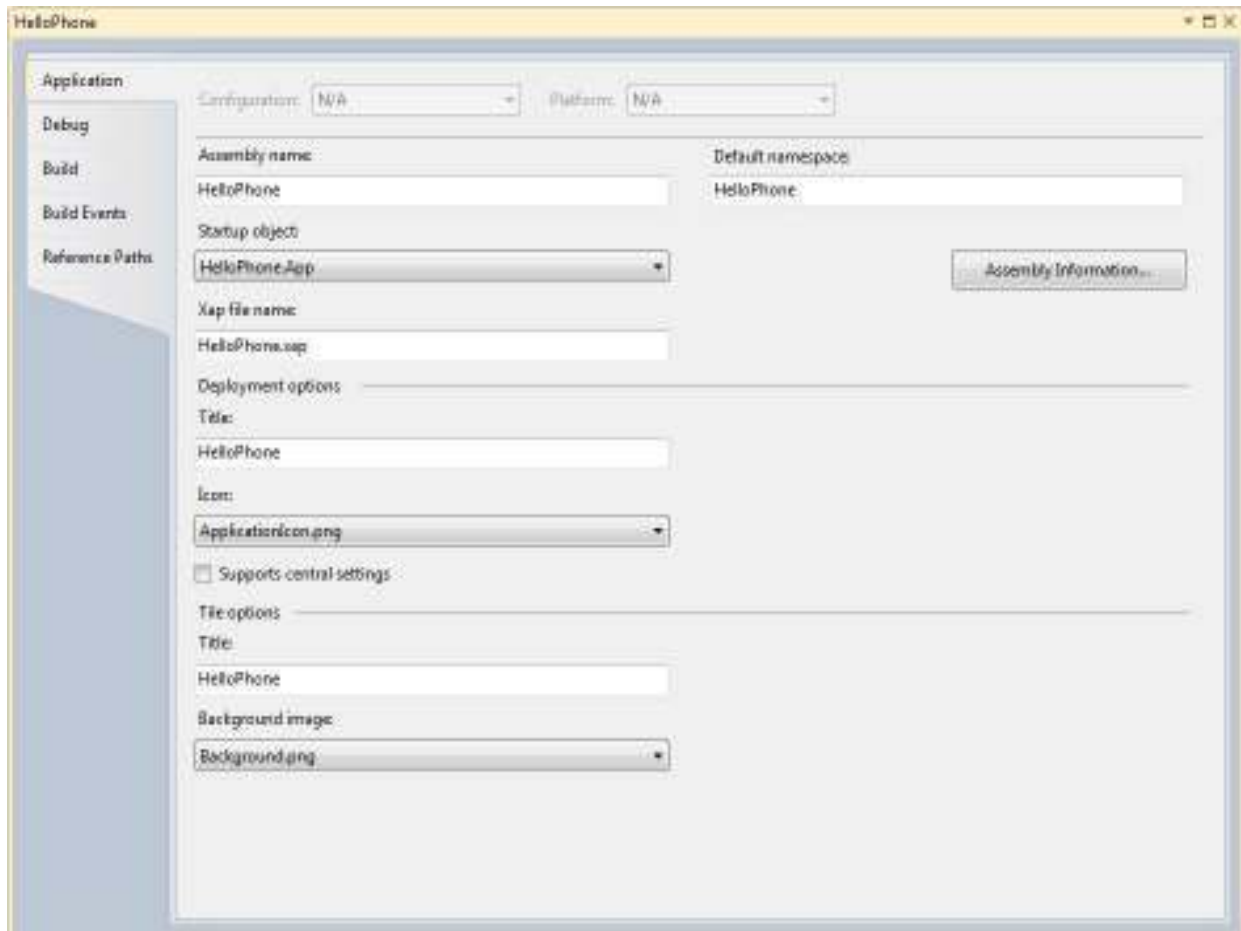
- File contains XAML markup with an **Application** root element and inside it an **Application.Resources** section. Herein you can define application-level resources such as colors, brushes and style objects used throughout the application.
- The XAML code also initializes the **ApplicationLifetimeObjects** property of the **Application** to create a **PhoneApplicationService** object. The **PhoneApplicationService** class provides access to various aspects of the application's lifetime. This includes management of the application's idle behavior and management of the application's state when it becomes active or inactive.



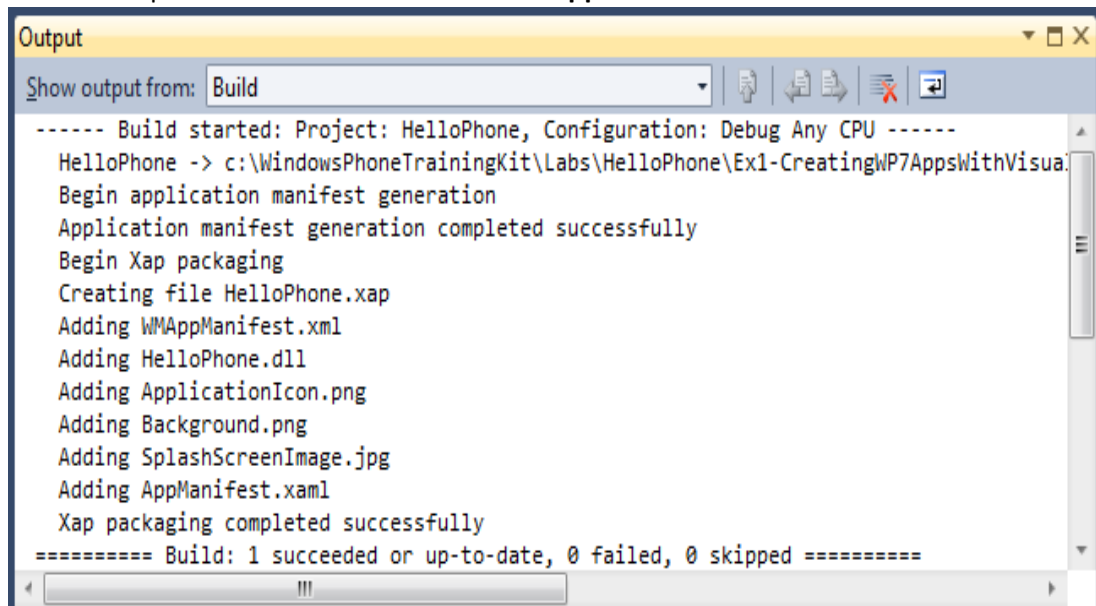
- The **RootFrame** property in the **Application** class identifies the starting page of the application. All Windows Phone applications have a single top-level container element whose data type is **PhoneApplicationFrame**. The frame hosts one or more **PhoneApplicationPage** elements that present content for the application. It also handles navigation between pages.
- *Application code-behind file showing global event handlers*



- The generated project includes a default document that contains XAML markup that defines the main UI of the application, **MainPage.xaml**. The XAML document provides a blank canvas to which you add controls to create your application's user interface. : Extensible Application Markup Language (XAML) is a declarative language.
- The **ApplicationIcon.png** file contains the icon that identifies the application in the quick launch screen of the phone device. You can double-click the item in **Solution Explorer** to open the file in a registered image editing application on your machine, for example, **Paint.exe**.



- The Windows Phone project properties window allows you to modify some phone-specific properties. These properties relate to the deployment and appearance of the application on the device. The parameters are stored in the **WMAppManifest.xml**





- Verify that the target of the deployment is the Windows Phone Emulator. To do this, ensure that Windows Phone 7 Emulator is selected in the **Select Target** drop down next to the **Start Debugging** button on the toolbar.



- Press **F5** to launch the application in the Windows Phone Emulator. Notice that a device emulator window appears and there is a pause while Visual Studio sets up the emulator environment and deploys the application image.
- Once it is ready, the emulator shows the Start page and shortly thereafter, your application appears in the emulator window.



XAML

...

```
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*/>
  </Grid.RowDefinitions>
```

...

```
</Grid>
```

```
</phone:PhoneApplicationPage>
```

- In the XAML markup generated by the default Windows Phone application template, locate the **Grid** container element named **LayoutRoot**. Its purpose is to arrange the elements on the page. Inside its **RowDefinition** property, insert an additional row between the two existing rows and set the value of its **Height** property to **Auto**. This row will soon include a textbox and a button.
- **Grid**: Defines a flexible grid area consisting of columns and rows.

```

<phone:PhoneApplicationPage
  x:Class="HelloPhone.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
  xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
  FontFamily="{StaticResource PhoneFontFamilyNormal}"
  FontSize="{StaticResource PhoneFontSizeNormal}"
  Foreground="{StaticResource PhoneForegroundBrush}"
  SupportedOrientations="Portrait" Orientation="Portrait"
  shell:SystemTray.IsVisible="True">

  <!--LayoutRoot is the root grid where all page content is placed-->
  <Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*/>
    </Grid.RowDefinitions>

    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
      <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION" Style="{StaticResource PhoneTextN
      <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0" Style="{StaticResource Phon
    </StackPanel>

    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Grid>
  </Grid>

```

- Notice that the root **Grid** element contains other nested elements with each one assigned to a different row of the outer grid by defining a **Grid.Row** property. Locate the **Grid** element named *TitlePanel*. Set the **Text** property of the first **TextBlock** element inside the inner Grid to the string "Windows Phone 7". Similarly, set the **Text** property of the second **TextBlock** element to the string "Hello Phone".

Grid.ColumnDefinitions>

```

  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
  <TextBox      Grid.Column="0"      Name="MessageTextBox"      FontSize="{StaticResource
PhoneFontSizeExtraLarge}" Margin="20,20,10,20"/>
  <Button Grid.Column="1" Name="ClickMeButton" Content="Click Me" HorizontalAlignment="Right"
Padding="4" Margin="10,20,20,20" />
</Grid>

</Grid>

```

- Now, locate the **Grid** element named *ContentPanel*, which should currently be empty, and paste the following (blue-highlighted) XAML markup inside this element.

XAML

```

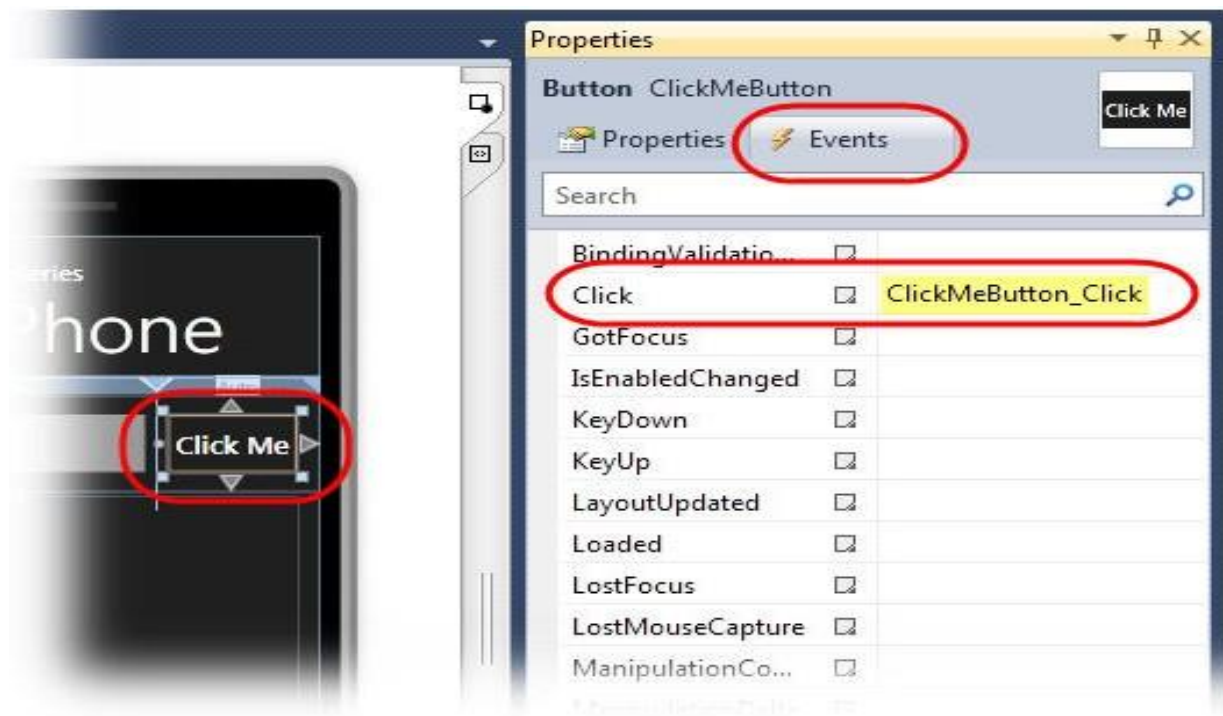
...
<Grid x:Name="LayoutRoot" Background="Transparent">
...

    <Grid Grid.Row="2">
        <TextBlock Name="BannerTextBlock" Style="{StaticResource PhoneTextExtraLargeStyle}"
            Foreground="#FFFF9A00" HorizontalAlignment="Stretch"
            TextWrapping="Wrap" TextAlignment="Center" FontWeight="Bold" />
    </Grid>
</Grid>
...

```

- To complete the design of the page, add a third row to contain the banner with the message entered by the user. To create this row, insert the following (blue-highlighted) XAML markup immediately before the end tag of the outer grid





- Click the button labeled “Click Me” on the designer surface to select it and then press **F4** to open its **Properties** window.
- In the **Properties** panel, click the **Events** tab to display a window with a list of available events. Locate the **Click** event in this list and then type **ClickMeButton_Click** in the text box located next to this event. Press **ENTER** to generate an event handler with this name and open the code-behind file to display the method stub generated by Visual Studio.

```
<!--This section is empty. Place new content here Grid.Row="1"-->
<Grid Grid.Row="1">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    <TextBox Grid.Column="0" Name="MessageTextBox" FontSize="{StaticResource PhoneTextFontSizeNormal}" />
    <Button Grid.Column="1" Name="ClickMeButton" Content="Click Me"
        HorizontalAlignment="Right" Padding="4" Margin="10,20,20,20"
        Click="ClickMeButton_Click" />
</Grid>
<Grid Grid.Row="2">
```

C#

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
    BannerTextBlock.Text = MessageTextBox.Text;
    MessageTextBox.Text = String.Empty;
}
```

- The method implementation (which is an empty method right now) is in the **MainPage.xaml.cs** file. Insert the following code inside the body of the **ClickMeButton_Click** method.

```
namespace HelloPhone
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }

        private void ClickMeButton_Click(object sender, RoutedEventArgs e)
        {
            BannerTextBlock.Text = MessageTextBox.Text;
            MessageTextBox.Text = String.Empty;
        }
    }
}
```



Lecture 38

Application Manifest

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
package="com.my_domain.my_app"
android:versionCode="1"
android:versionName="0.9 Beta">
[ ... manifest nodes ... ]
</manifest>
```

- Each Android project includes a manifest file, AndroidManifest.xml
- The manifest lets you define the structure and metadata of your application, its components, and its requirements.
- It includes nodes for each of the components (Activities, Services, Content Providers, and Broadcast Receivers) that make up your application and,
- Using Intent Filters and Permissions, determines how they interact with each other and with other applications.

```
<uses-sdk android:minSdkVersion="4"
android:targetSdkVersion="5">
</uses-sdk>
<uses-configuration android:reqTouchScreen="finger"
android:reqNavigation=["trackball"]
android:reqHardKeyboard=["true"]
android:reqKeyboardType=["qwerty"]/>
<uses-configuration android:reqTouchScreen="finger"
android:reqNavigation=["trackball"]
android:reqHardKeyboard=["true"]
android:reqKeyboardType=["twelvekey"]/>
```

- uses-sdk This node lets you define a minimum, maximum, and target SDK version that must be available on a device in order for your application to function properly
- uses-configuration Use uses-configuration nodes to specify each combination of input mechanisms supported by your application. a device with a finger touchscreen, a trackball, and either a QUERTY or twelve-key hardware keyboard

```
<uses-feature android:glEsVersion=" 0x00010001"
android:name="android.hardware.camera" />
<supports-screens android:smallScreens=["false"]
android:normalScreens=["true"]
android:largeScreens=["true"]
android:anyDensity=["false"] />
```

- uses-feature nodes to specify each of the hardware features your application requires.
- You can also use the uses-feature node to specify the minimum version of OpenGL required by your application
- supports-screen node lets you specify the screen sizes your application can, and can't, support.

```

<application android:icon="@drawable/icon"
android:theme="@style/my_theme"
android:name="MyApplication"
android:debuggable="true">
[ ... application nodes ... ]
</application>
<activity android:name=".MyActivity" android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

```

- application A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme).
- The <application> node also acts as a container that includes the Activity, Service, Content Provider, and Broadcast Receiver tags used to specify the application components.
- activity An <activity> tag is required for every Activity displayed by your application. Using the android:name attribute to specify the Activity class name. Trying to start an Activity that's not defined in the manifest will throw a runtime exception. Each Activity node supports <intent-filter> child tags that specify which Intents launch the Activity.

```

<service android:enabled="true" android:name=".MyService"></service>
<provider android:permission="com.paad.MY_PERMISSION"
    android:name=".MyContentProvider"
    android:enabled="true"
    android:authorities="com.paad.myapp.MyContentProvider">
</provider>
<receiver android:enabled="true"
    android:label="My Intent Receiver"
    android:name=".MyIntentReceiver">
</receiver>

```

- service As with the activity tag, create a new service tag for each Service class used in your application. Service tags also support <intent-filter> child tags to allow late runtime binding.
- provider Provider tags specify each of your application's Content Providers. Content Providers are used to manage database access and sharing within and between applications
- receiver By adding a receiver tag, you can register a Broadcast Receiver without having to launch your application first. Broadcast Receivers are like global event listeners that, once registered, will execute whenever a matching Intent is broadcast by the system or an application. By registering a Broadcast Receiver in the manifest you can make this process entirely autonomous. If a matching Intent is broadcast, your application will be started automatically and the registered Broadcast Receiver will be run.
- <uses-permission android:name="android.permission.ACCESS_LOCATION"/>
- uses-permission As part of the security model, uses-permission tags declare the permissions you've determined your application needs to operate properly. The permissions you
- include will be presented to the user before installation commences. Permissions are required for many of the native Android services, particularly those with a cost or security implication
- (such as dialing, receiving SMS, or using the location-based services).

Location Based Services

- Your application knows how to say Hello, but it doesn't know where it's located.
- Now is a good time to become familiar with some simple location-based calls to get the GPS coordinates.
- Problem is that emulator does not have GPS sensors!!
- The emulator does not have location sensors, so the first thing you need to do is seed your emulator with GPS coordinates



1. Press the Home key to return to the Home screen.
2. Launch the Maps application from the Application drawer.
3. Click the Menu button.
4. Choose the My Location menu item. (It looks like a target.)
5. Click the DDMS perspective in the top-right corner of Eclipse.
6. You see an Emulator Control pane on the left side of the screen. Scroll down to the Location Control.
7. Manually enter the longitude and latitude of your location. (Note they are in reverse order.)
8. Click Send.

```
import android.location.Location;
import android.location.LocationManager;
....
public void getLocation() {
    try {
        LocationManager locMgr = (LocationManager)
            getSystemService(LOCATION_SERVICE);
        Location recentLoc = locMgr.
            getLastKnownLocation(LocationManager.GPS_PROVIDER);
        Log.i(DEBUG_TAG, "loc: " + recentLoc.toString());
    }
}
```

```

        catch (Exception e) {
            Log.e(DEBUG_TAG, "Location failed", e);
        }
    }

```

- new method called getLocation() in your class and make a call to this method in your onCreate() method. The getLocation() method gets the last known location on the phone and logs it as an informational message. If the operation fails for some reason, the method logs an error.
 - Android logging features are in the Log class of the android.util package.
 - Log.e() Log errors
 - Log.w() Log warnings
 - Log.i() Log informational messages
 - Log.d() Log Debug messages
 - Log.v() Log Verbose messages
-
- Now try to run the application
 - It generates an error

your application requires special permissions to access location-based functionality. You must register this permission in your AndroidManifest.xml file. To add location-based service permissions to your application, perform the following steps:

1. Double-click the AndroidManifest.xml file.
2. Switch to the Permissions tab.
3. Click the Add button and choose Uses Permission.
4. In the right pane, select android.permission.ACCESS_FINE_LOCATION. (Allows an application to access fine (e.g., GPS) location)
5. Save the file.

[ACCESS_COARSE_LOCATION](#) Allows an application to access coarse (e.g., Cell-ID, WiFi) location

Manifest Permissions

- [ACCESS MOCK LOCATION](#) Allows an application to create mock location providers for testing
- [ACCESS_NETWORK_STATE](#) Allows applications to access information about networks
- [BATTERY_STATS](#) Allows an application to collect battery statistics
- [BLUETOOTH](#) Allows applications to connect to paired bluetooth devices
- [CALL_PHONE](#) Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed.
- [CAMERA](#) Required to be able to access the camera device.
- [INTERNET](#) Allows applications to open network sockets.

Types of Android Applications

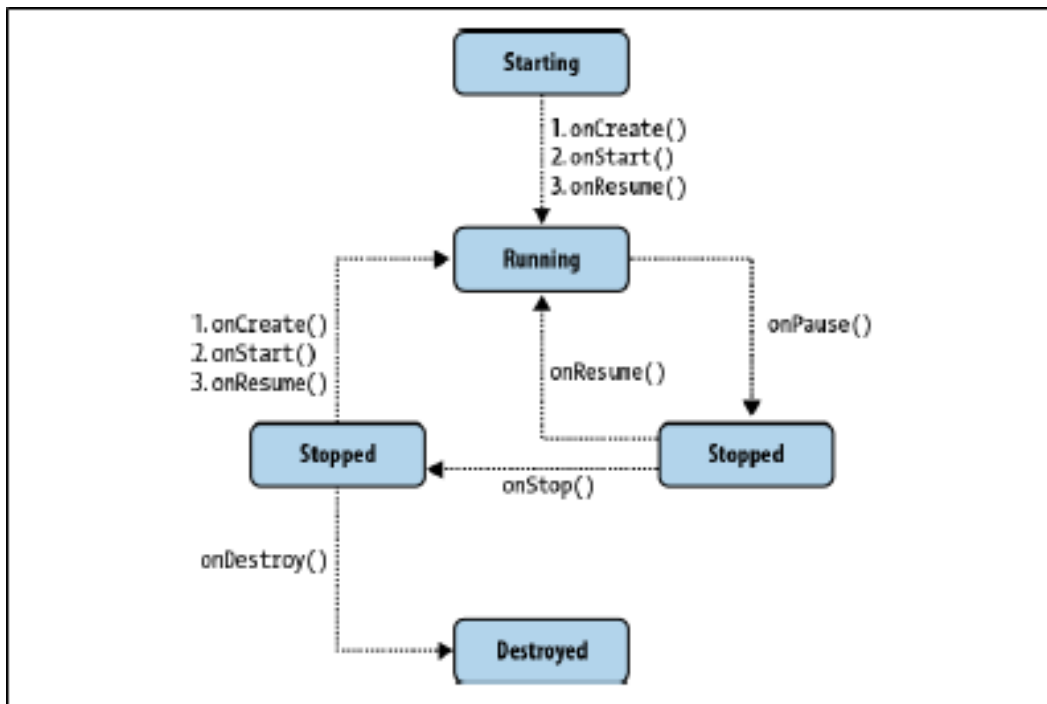
- Foreground
- Background
- Intermittent
- Widget
- **Foreground:** An application that's useful only when it's in the foreground and is effectively suspended when it's not visible. Games and map mashups are common examples.
- **Background:** An application with limited interaction that, apart from when being configured, spends most of its lifetime hidden. Examples include call screening applications and SMS auto-responders.
- **Intermittent:** Expects some interactivity but does most of its work in the background. Often these applications will be set up and then run silently, notifying users when appropriate. A common example would be a media player.
- **Widget:** Some applications are represented only as a home-screen widget.

Building Blocks of Android App

- **Activities:** Your application's presentation layer. Every screen in your application will be an extension of the Activity class. Activities use Views to form graphical user interfaces that display information and respond to user actions. In terms of desktop development, an Activity is equivalent to a Form.
- **Services:** The invisible workers of your application. Service components run in the background, updating your data sources and visible Activities and triggering Notifications. They're used to perform regular processing that needs to continue even when your application's Activities aren't active or visible.
- **Content Providers:** Shareable data stores. Content Providers are used to manage and share application databases. They're the preferred means of sharing data across application boundaries. This means that you can configure your own Content Providers to permit access from other applications and use Content Providers exposed by others to access their stored data. Android devices include several native Content Providers that expose useful databases like the media store and contact details.
- **Intents:** An inter-application message-passing framework. Using Intents you can broadcast messages system-wide or to a target Activity or Service, stating your intention to have an action performed. The system will then determine the target(s) that will perform any actions as appropriate.
- **Broadcast Receivers:** Intent broadcast consumers. If you create and register a Broadcast Receiver, your application can listen for broadcast Intents that match specific filter criteria. Broadcast Receivers will automatically start your application to respond to an incoming Intent, making them perfect for creating event-driven applications.
- **Widgets:** Visual application components that can be added to the home screen. A special variation of a Broadcast Receiver, widgets let you create dynamic, interactive application components for users to embed on their home screens.
- **Notifications:** A user notification framework. Notifications let you signal users without stealing focus or interrupting their current Activities. They're the preferred technique for getting a user's attention from within a Service or Broadcast Receiver. For example, when a device receives a text message or an incoming call, it alerts you by flashing lights, making sounds, displaying icons, or showing messages.

Lecture 39

Activity Life Cycle



- An activity is usually a single screen that the user sees on the device at one time. An application typically has multiple activities, and the user flips back and forth among them. As such, activities are the most visible part of your application. I usually use a website as an analogy for activities. Just like a website consists of multiple pages, so does an Android application consist of multiple activities. Just like a website has a “home page,” an Android app has a “main” activity, usually the one that is shown first when you launch the application. And just like a website has to provide some sort of navigation among various pages, an Android app should do the same. On the Web, you can jump from a page on one website to a page on another. Similarly, in Android, you could be looking at an activity of one application, but shortly after you could start another activity in a completely separate application. For example, if you are in your Contacts app and you choose to text a friend, you’d be launching the activity to compose a text message in the Messaging application.
- **Starting state:** When an activity doesn’t exist in memory, it is in a starting state. While it’s starting up, the activity will go through a whole set of callback methods that you as a developer have an opportunity to fill out. Eventually, the activity will be in a running state. Keep in mind that this transition from starting state to running state is one of the most expensive operations in terms of computing time, and this also directly affects the battery life of the device. This is the exact reason why we don’t automatically destroy activities that are no longer shown. The user might want to come back to them, so we keep them around for a while.

- **Running state:** The activity in a running state is the one that is currently on the screen and interacting with the user. We also say this activity is in focus, meaning that all user interactions—such as typing, touching the screen, and clicking buttons—are handled by this one activity. As such, there is only one running activity at any given time. The running activity is the one that has priority in terms of getting the memory and resources it needs to run as quickly as possible. This is because Android wants to make sure the running activity is zippy and responsive to the user.
- **Paused state:** When an activity is not in focus (i.e., not interacting with the user) but still visible on the screen, we say it's in a paused state. This is not a typical scenario, because the device's screen is usually small, and an activity is either taking up the whole screen or none at all. We often see this case with dialog boxes that come up in front of an activity, causing it to become Paused. All activities go through a paused state en route to being stopped. Paused activities still have high priority in terms of getting memory and other resources. This is because they are visible and cannot be removed from the screen without making it look very strange to the user.
- **Stopped state:** When an activity is not visible, but still in memory, we say it's in a stopped state. Stopped activity could be brought back to the front to become a Running activity again. Or, it could be destroyed and removed from memory. The system keeps activities around in a stopped state because it is likely that the user will still want to get back to those activities sometime soon, and restarting a stopped activity is far cheaper than starting an activity from scratch. That is because we already have all the objects loaded in memory and simply have to bring it all up to the foreground. Stopped activities can be removed from memory at any point.
- **Destroyed state:** A destroyed activity is no longer in memory. The Activity Manager decided that this activity is no longer needed and has removed it. Before the activity is destroyed, it can perform certain actions, such as save any unsaved information. However, there's no guarantee that your activity will be stopped prior to being destroyed. It is possible for a paused activity to be destroyed as well. For that reason, it is better to do important work, such as saving unsaved data, en route to a paused state rather than a destroyed State.

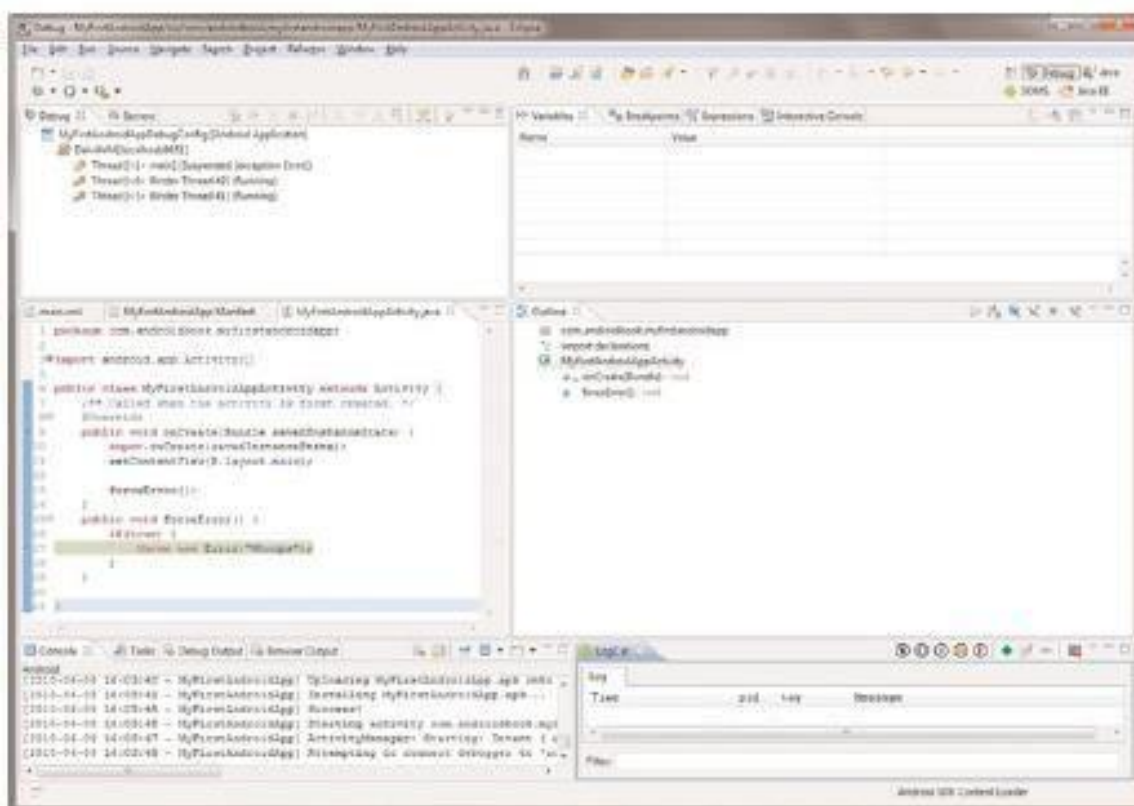
```
package net.learn2develop.Activities;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends Activity {
    String tag = "Events";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(tag, "In the onCreate() event");
    }
}
```

Method	Purpose
<code>Log.e()</code>	Log errors
<code>Log.w()</code>	Log warnings
<code>Log.i()</code>	Log informational messages
<code>Log.d()</code>	Log Debug messages
<code>Log.v()</code>	Log Verbose messages

```

public void onStart() { super.onStart(); Log.d(tag, "In the onStart() event");}
public void onRestart() { super.onRestart(); Log.d(tag, "In the onRestart() event"); }
public void onResume() { super.onResume(); Log.d(tag, "In the onResume() event"); }
public void onPause() { super.onPause(); Log.d(tag, "In the onPause() event"); }
public void onStop() { super.onStop(); Log.d(tag, "In the onStop() event"); }
public void onDestroy() { super.onDestroy(); Log.d(tag, "In the onDestroy() event");}
}

```



Lecture 40

Activity Life Cycle

- Back Button
- Home Button
- Phone Button



When you now press the back button on the Android Emulator, observe that the following is printed:

12-28 13:59:46.266: DEBUG/Events(334): In the onPause() event

12-28 13:59:46.806: DEBUG/Events(334): In the onStop() event

12-28 13:59:46.806: DEBUG/Events(334): In the onDestroy() event

Click the Home button and hold it there. Click the Activities icon and observe the following:

12-28 14:00:54.115: DEBUG/Events(334): In the onCreate() event

12-28 14:00:54.156: DEBUG/Events(334): In the onStart() event

12-28 14:00:54.156: DEBUG/Events(334): In the onResume() event

Press the Phone button on the Android Emulator so that the activity is pushed to the background.

12-28 14:01:16.515: DEBUG/Events(334): In the onPause() event

12-28 14:01:17.135: DEBUG/Events(334): In the onStop() event

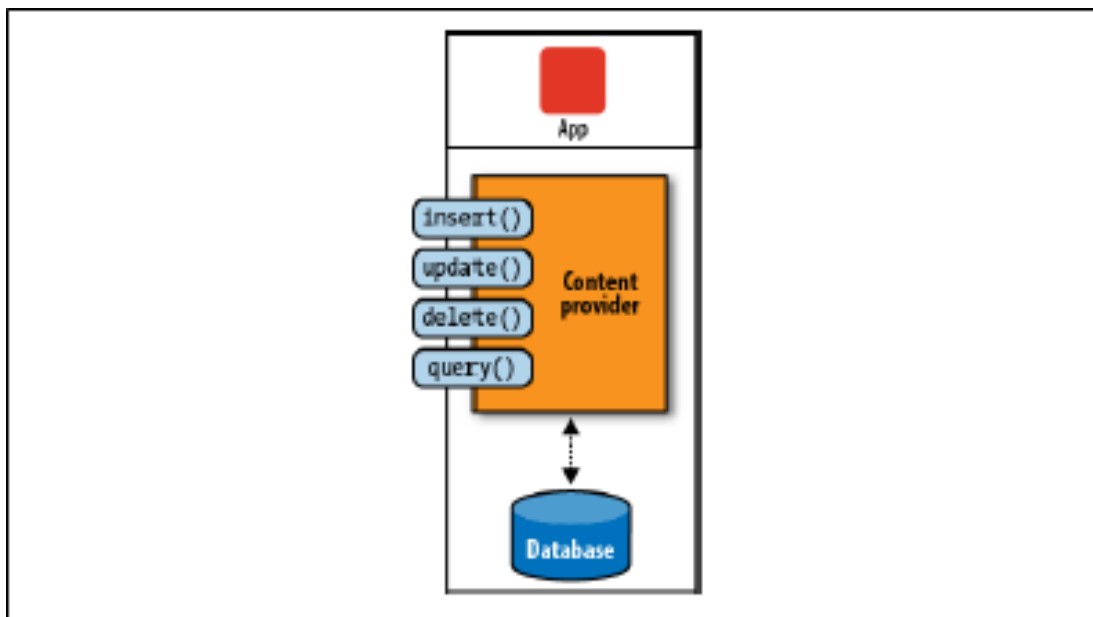
Notice that the onDestroy() event is not called, indicating that the activity is still in memory. Exit the phone dialer by pressing the Back button. The activity is now visible again.

12-28 14:02:17.255: DEBUG/Events(334): In the onRestart() event

12-28 14:02:17.255: DEBUG/Events(334): In the onStart() event

12-28 14:02:17.255: DEBUG/Events(334): In the onResume() event

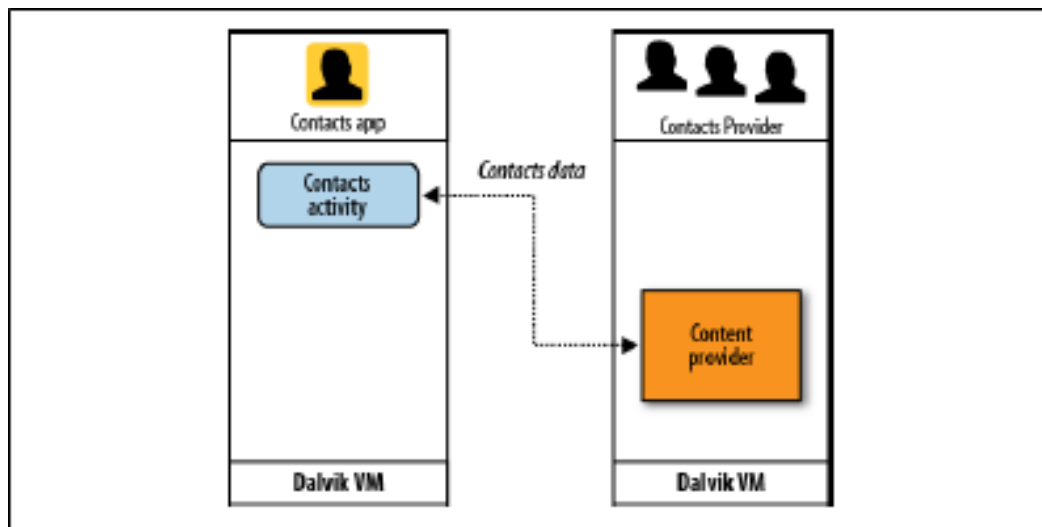
Content Provider



- Content providers are interfaces for sharing data between applications. By default, Android runs each application in its own sandbox so that all data that belongs to an application is totally isolated from other applications on the system. Although small amounts of data can be passed between applications via intents, content providers are much better suited for sharing persistent data between possibly large datasets. The Android system uses this mechanism all the time.
- **Browser** — Stores data such as browser bookmarks, browser history, and so on
- **CallLog** — Stores data such as missed calls, call details, and so on
- **Contacts** — Stores contact details, Contacts Provider is a content provider that exposes all user contact data to various applications
- **MediaStore** — Stores media files such as audio, video and images, responsible for storing and sharing various media, such as photos and music, across various applications.
- **Settings** — Stores the device's settings and preferences, Settings Provider exposes system settings to various applications, including the built-in Settings application.
- Besides the many built-in content providers, you can also create your own content providers.

Lecture 41

Content Provider



- Content providers are relatively simple interfaces, with the standard insert(), update(), delete(), and query() methods. These methods look a lot like standard database methods, so it is relatively easy to implement a content provider as a proxy to the database. Having said that, you are much more likely to use content providers than write your own.
- How the Contacts app uses Contacts Provider, a totally separate application, to retrieve data about users' contacts. The Contacts app itself doesn't have any contacts data, and Contacts Provider doesn't have any user interface. This separation of data storage and the actual user interface application offers greater flexibility to mash up various parts of the system. For example, a user could install an alternative address book application that uses the same data as the default Contacts app. Or, he could install widgets on the Home screen that allow for easy changes in the System Settings, such as turning on or off the WiFi, Bluetooth, or GPS features.
- To query a content provider, you specify the query string in the form of a URI, with an optional specifier for a particular row. The format of the query URI is as follows:
<standard_prefix>://<authority>/<data_path>/<id>

QUERY STRING	DESCRIPTION
content://media/internal/images	Returns a list of all the internal images on the device
content://media/external/images	Returns a list of all the images stored on the external storage (e.g., SD card) on the device
content://call_log/calls	Returns a list of all calls registered in the Call Log
content://browser/bookmarks	Returns a list of bookmarks stored in the browser

The various parts of the URI are as follows:

- The standard prefix for content providers is always content://.
- The authority specifies the name of the content provider. An example would be contacts for the built-in Contacts content provider. For third-party content providers, this could be the fully qualified name, such as com.wrox.provider or net.learn2develop.provider.
- The data path specifies the kind of data requested. For example, if you are getting all the contacts from the Contacts content provider, then the data path would be people, and the URI would look like this: content://contacts/people.
- The id specifies the specific record requested. For example, if you are looking for contact number 2 in the Contacts content provider, the URI would look like this: content://contacts/people/2.

Content Providers – Cursors

- **Cursors**
 - This interface provides random read-write access to the result set returned by a database query.
- **Cursor Adapters**
 - Adapter that exposes data from a Cursor to a ListView Widget
- ListView: A view that shows items in a vertically scrolling list

Content Providers

```
public class MainActivity extends ListActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Uri allContacts = Uri.parse("content://contacts/people");
        Cursor c = managedQuery(allContacts, null, null, null, null);
        ....
        ...
    }
}
```

The managedQuery() method of the Activity class retrieves a managed cursor. A *managed cursor* handles all the work of unloading itself when the application pauses and requerying itself when the application restarts.

The statement

```
Cursor c = managedQuery(allContacts, null, null, null, null);
```

is equivalent to

```
Cursor c = getContentResolver().query(allContacts, null, null, null, null);
```

```
startManagingCursor(c); //---allows the activity to manage the Cursor's // lifecycle based on the activity's lifecycle---
```

The getContentResolver() method returns a ContentResolver object, which helps to resolve a content URI with the appropriate content provider.

```

<?xml version="1.0" encoding="utf-8"?>
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Provider"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
        android:label="@string/app_name">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>
    <uses-sdk android:minSdkVersion="7" />
    <uses-permission android:name="android.permission.READ_CONTACTS">
    </uses-permission>
  </manifest>

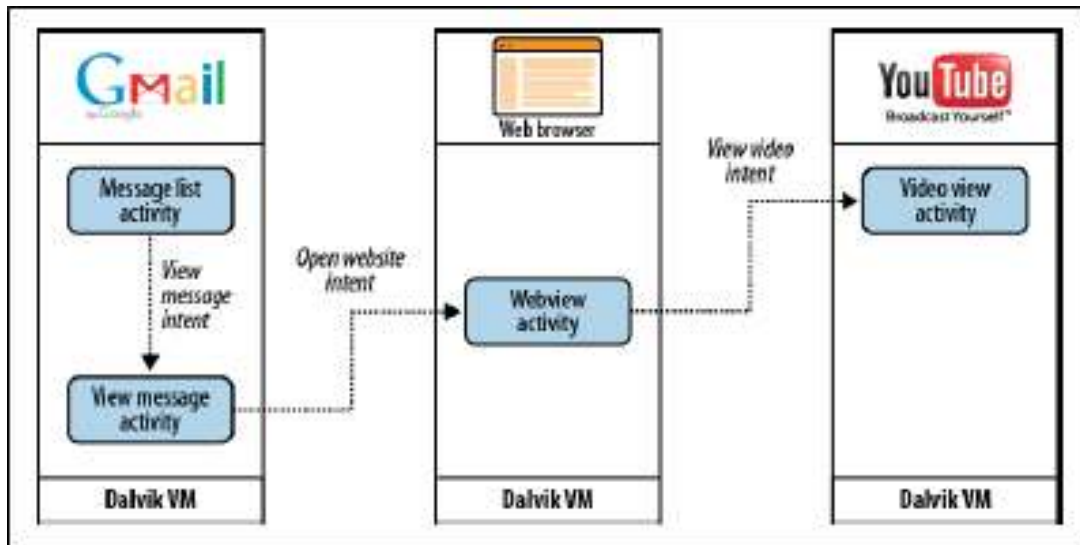
```

Note that in order for your application to access the Contacts application, you need to have the READ_CONTACTS permission in your AndroidManifest.xml file.

- Some examples of predefined query string constants are as follows:
 - Browser.BOOKMARKS_URI
 - Browser.SEARCHES_URI
 - CallLog.CONTENT_URI
 - MediaStore.Images.Media.INTERNAL_CONTENT_URI
 - MediaStore.Images.Media.EXTERNAL_CONTENT_URI
 - Settings.CONTENT_URI

Lecture 42

Intents



- Intents are messages that are sent among the major building blocks. They trigger an activity to start up, tell a service to start or stop, or are simply broadcasts. Intents are asynchronous, meaning the code that sends them doesn't have to wait for them to be completed.
- An intent could be explicit or implicit. In an explicit intent, the sender clearly spells out which specific component should be on the receiving end. In an implicit intent, the sender specifies the type of receiver. For example, your activity could send an intent saying it simply wants someone to open up a web page. In that case, any application that is capable of opening a web page could "compete" to complete this action. When you have competing applications, the system will ask you which one you'd like to use to complete a given action.
- An intent could be explicit or implicit. In an explicit intent, the sender clearly spells out which specific component should be on the receiving end. In an implicit intent, the sender specifies the type of receiver. For example, your activity could send an intent saying it simply wants someone to open up a web page. In that case, any application that is capable of opening a web page could "compete" to complete this action. When you have competing applications, the system will ask you which one you'd like to use to complete a given action. You can also set an app as the default. This mechanism works very similarly to your desktop environment, for example, when you downloaded Firefox or Chrome to replace your default Internet Explorer or Safari web browsers. This type of messaging allows the user to replace any app on the system with a custom one. For example, you might want to download a different SMS application or another browser to replace your existing ones.



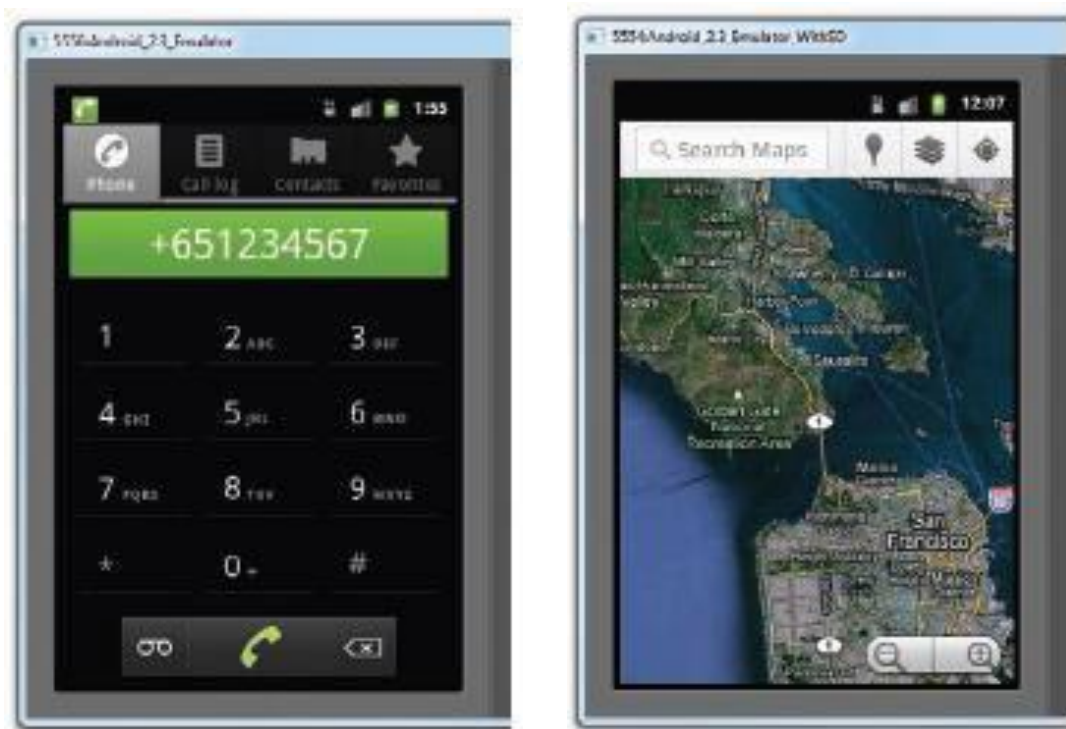
- `Intent i = new Intent(android.content.Intent.ACTION_VIEW, Uri.parse("http://www.amazon.com"));`
`startActivity(i);`
- `Intent i = new Intent(android.content.Intent.ACTION_DIAL, Uri.parse("tel:+651234567"));`
`startActivity(i);`
- `Intent i = new Intent(android.content.Intent.ACTION_VIEW, Uri.parse("geo:37.827500,-122.481670"));`
`startActivity(i);`
- In the first button, you create an Intent object and then pass two arguments to its constructor — the action and the data: The action here is represented by the **`android.content.Intent.ACTION_VIEW`** constant. You use the `parse()` method of the `Uri` class to convert an URL string into an `Uri` object.
- For the second button, you dial a specific number by passing in the telephone number in the data portion. In this case, the dialer will display the number to be called.
- If you simply want to display the dialer without specifying any number, simply omit the data portion, like this: `Intent i = new Intent(android.content.Intent.ACTION_DIAL);`

- The third button displays a map using the ACTION_VIEW constant: Here, instead of using “http” you use the “geo” scheme.
- ```
Intent i = new Intent(android.content.Intent.ACTION_PICK);
i.setType(ContactsContract.Contacts.CONTENT_TYPE);
startActivityForResult(i,request_Code);
```

#### **android.content.Intent.ACTION\_PICK**

- The fourth button invokes the Contacts application to enable the user to pick a contact. Because you are asking the user to select a contact, you need the Contacts application to return a value; in this case, you need to set the type of data to indicate what kind of data needs to be returned:
- Because you are expecting a result from the Contacts application, you invoke it using the `startActivityForResult()` method, passing in the Intent object and a request code. You need to implement the `onActivityResult()` method in order to obtain a result from the activity:





- **Action** : The action portion defines what you want to do
- **Data**: while the data portion contains the data for the target activity to act upon
- **Type**: For some intents, there is no need to specify the data. For example, to select a contact from the Contacts application, you specify the action and then indicate the MIME type using the setType() method
- **Category**: Besides specifying the action, the data, and the type, an Intent object can also specify a category. A category groups activities into logical units so that Android can use it for further filtering.

You can also pass the data to the Intent object using the setData() method. In this example, you indicate that you want to view a web page with the specified URL. The Android OS will look for all activities that are able to satisfy your request. This process is known as *intent resolution*.

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW);
i.setData(Uri.parse("http://www.amazon.com"));
Intent i = new Intent(android.content.Intent.ACTION_PICK);
i.setType(ContactsContract.Contacts.CONTENT_TYPE);
i.setType(ContactsContract.CommonDataKinds.Phone.CONTENT_TYPE);
```

```
<intent-filter>
```

```
<action android:name="net.learn2develop.ACTIVITY2" />
```

```
<category android:name="android.intent.category.DEFAULT" />
```

```
</intent-filter>
```

- This is a very simple example in which one activity calls another using the "net.learn2develop.ACTIVITY2" action.
- An activity can invoke another activity using the Intent object. In order for other activities to invoke your activity, you need to specify the action and category within the <intent-filter> element in the AndroidManifest.xml file

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW, Uri.parse("http://www.amazon.com"));
```

```
i.addCategory("net.learn2develop.Apps");
```

```
startActivity(i);
```

```
<activity android:name=".MyBrowserActivity"
```

```
android:label="@string/app_name">
```

```
<intent-filter>
```

```
<action android:name="android.intent.action.VIEW" />
```

```
<action android:name="net.learn2develop.MyBrowser" />
```

```
<category android:name="android.intent.category.DEFAULT" />
```

```
<category android:name="net.learn2develop.Apps" />
```

```
<data android:scheme="http" />
```

```
</intent-filter>
```

- You add the category to the Intent object using the addCategory() method. If you omit the addCategory() statement, the preceding code will still invoke the MyBrowerActivity activity because it will still match the default category "android.intent.category.DEFAULT".

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW, Uri.parse("http://www.amazon.com"));
i.addCategory("net.learn2develop.OtherApps");
startActivity(i);
<activity android:name=".MyBrowserActivity"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<action android:name="net.learn2develop.MyBrowser" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="net.learn2develop.Apps" />
<category android:name="net.learn2develop.OtherApps" />
<data android:scheme="http" />
</intent-filter>
```

- The preceding category ("net.learn2develop.OtherApps") does not match any in the intent filter, so a run-time exception will be raised.

## Lecture 43

### Intents

- **Action :** The action portion defines what you want to do
- **Data:** while the data portion contains the data for the target activity to act upon
- **Type:** For some intents, there is no need to specify the data. For example, to select a contact from the Contacts application, you specify the action and then indicate the MIME type using the setType() method
- **Category:** Besides specifying the action, the data, and the type, an Intent object can also specify a category. A category groups activities into logical units so that Android can use it for further filtering.
  - You can also pass the data to the Intent object using the setData() method. In this example, you indicate that you want to view a web page with the specified URL. The Android OS will look for all activities that are able to satisfy your request. This process is known as *intent resolution*.
  - **Intent i = new Intent(android.content.Intent.ACTION\_VIEW);**
  - **i.setData(Uri.parse("http://www.amazon.com"));**
  - **Intent i = new Intent(android.content.Intent.ACTION\_PICK);**
  - **i.setType(ContactsContract.Contacts.CONTENT\_TYPE);**
  - **i.setType(ContactsContract.CommonDataKinds.Phone.CONTENT\_TYPE);**

```
<intent-filter . . . >
 <data android:mimeType="video/mpeg" android:scheme="http" . . . />
 <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
 . . .
</intent-filter>
```

- Consider, for example, what the browser application does when the user follows a link on a web page. It first tries to display the data (as it could if the link was to an HTML page). If it can't display the data, it puts together an implicit intent with the scheme and data type and tries to start an activity that can do the job. If there are no takers, it asks the download manager to download the data. That puts it under the control of a content provider, so a potentially larger pool of activities (those with filters that just name a data type) can respond.
- Intents are matched against intent filters not only to discover a target component to activate, but also to discover something about the set of components on the device.
- For example, the Android system populates the application launcher, the top-level screen that shows the applications that are available for the user to launch, by finding all the activities with intent filters that specify the "android.intent.action.MAIN" action and "android.intent.category.LAUNCHER" category.
- It then displays the icons and labels of those activities in the launcher. Similarly, it discovers the home screen by looking for the activity with "android.intent.category.HOME" in its filter.

```
<intent-filter>
<action android:name="net.learn2develop.ACTIVITY2" />
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

This is a very simple example in which one activity calls another using the "net.learn2develop.ACTIVITY2" action.

- An activity can invoke another activity using the Intent object. In order for other activities to invoke your activity, you need to specify the action and category within the <intent-filter> element in the AndroidManifest.xml file

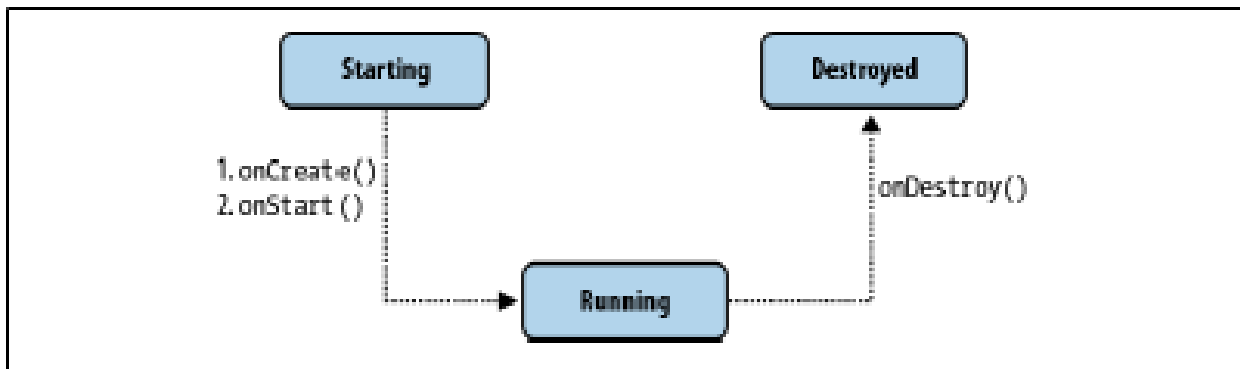
```
Intent i = new Intent(android.content.Intent.ACTION_VIEW, Uri.parse("http://www.amazon.com"));
i.addCategory("net.learn2develop.Apps");
startActivity(i);
<activity android:name=".MyBrowserActivity" android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<action android:name="net.learn2develop.MyBrowser" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="net.learn2develop.Apps" />
<data android:scheme="http" />
</intent-filter>
```

- You add the category to the Intent object using the addCategory() method. If you omit the addCategory() statement, the preceding code will still invoke the MyBrowserActivity activity because it will still match the default category "android.intent.category.DEFAULT".

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW, Uri.parse("http://www.amazon.com"));
i.addCategory("ALTERNATIVE");
startActivity(i);
<intent-filter android:label="@string/resolve_title">
<action android:name="com.android.notepad.action.EDIT_TITLE" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.ALTERNATIVE" />
<category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
<data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

- The preceding category ("net.learn2develop.OtherApps") does not match any in the intent filter, so a run-time exception will be raised.

## Services



- Services run in the background and don't have any user interface components. They can perform the same actions as activities, but without any user interface. Services are useful for actions that we want to perform for a while, regardless of what is on the screen. For example, you might want your music player to play music even as you are flipping between other applications. Services have a much simpler life cycle than activities (see Figure 4-3). You either start a service or stop it. Also, the service life cycle is more or less controlled by the developer, and not so much by the system. Consequently, we as developers have to be mindful to run our services so that they don't consume shared resources unnecessarily, such as the CPU and battery.

```

public class MyService extends Service {
 @Override
 public IBinder onBind(Intent arg0) {
 return null;
 }
 @Override
 public int onStartCommand(Intent intent, int flags, int startId) {
 // We want this service to continue running until it is explicitly
 // stopped, so return sticky.
 Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
 return START_STICKY;
 }
 @Override
 public void onDestroy() {
 super.onDestroy();
 Toast.makeText(this, "Service Destroyed",
 Toast.LENGTH_LONG).show();
 }
}

```

- MyService.java.** Populate it with the following code
- First, you defined a class that extends the Service base class. All services extend the Service class:
- public class MyService extends Service { }

- The `onBind()` method enables you to bind an activity to a service. This in turn enables an activity to directly access members and methods inside a service. For now, you simply return a null for this method.
- `@Override`
- `public IBinder onBind(Intent arg0) {...}`
- The `onStartCommand()` method is called when you start the service explicitly using the `startService()` method (discussed shortly). This method signifies the start of the service, and you code it to do the things you need to do for your service. In this method, you returned the constant `START_STICKY` so that the service will continue to run until it is explicitly stopped.
- `@Override`
- `public int onStartCommand(Intent intent, int flags, int startId) {...}`
- The `onDestroy()` method is called when the service is stopped using the `stopService()` method. This is where you clean up the resources used by your service.
- `@Override`
- `public void onDestroy() {...}`

```
<?xml version="1.0" encoding="utf-8"?>
 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="net.learn2develop.Services"
 android:versionCode="1"
 android:versionName="1.0">
 <application android:icon="@drawable/icon"
 android:label="@string/app_name">
 <activity android:name=".MainActivity"
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 <service android:name=".MyService" />
 </application>
 <uses-sdk android:minSdkVersion="9" />
 </manifest>
```

- All services that you have created must be declared in the `AndroidManifest.xml` file, like this:
- `<service android:name=".MyService" />`
- If you want your service to be available to other applications, you can always add an intent filter with an action name, like this:
- `<service android:name=".MyService">`
- `<intent-filter>`
- `<action android:name="net.learn2develop.MyService" />`
- `</intent-filter>`
- `</service>`



```

public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 Button btnStart = (Button) findViewById(R.id.btnStartService);
 btnStart.setOnClickListener(new View.OnClickListener() {
 public void onClick(View v) {
 startService(new Intent(getApplicationContext(), MyService.class));
 }});
 Button btnStop = (Button) findViewById(R.id.btnStopService);
 btnStop.setOnClickListener(new View.OnClickListener() {
 public void onClick(View v) {
 stopService(new Intent(getApplicationContext(), MyService.class));
 }}
 }

```

- To start a service, you use the startService() method, like this:
- startService(new Intent(getApplicationContext(), MyService.class));
- If you are calling an external service, then the call to the startService() method will look like this:
- startService(new Intent("net.learn2develop.MyService"));
- To stop a service, use the stopService() method, like this:
- stopService(new Intent(getApplicationContext(), MyService.class));

## Intents and Broadcast Receivers

- **Intents:** An inter-application message-passing framework. Using Intents you can broadcast messages system-wide or to a target Activity or Service, stating your intention to have an action performed. The system will then determine the target(s) that will perform any actions as appropriate.
- **Broadcast Receivers:** Intent broadcast consumers. If you create and register a Broadcast Receiver, your application can listen for broadcast Intents that match specific filter criteria. Broadcast Receivers will automatically start your application to respond to an incoming Intent, making them perfect for creating event-driven applications.

## SMS Application

```
<?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"
>

 <Button
 android:id="@+id/btnSendSMS"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Send SMS" />

</LinearLayout>
```

## AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="net.learn2develop.SMS"
 android:versionCode="1"
 android:versionName="1.0">
 <application android:icon="@drawable/icon" android:label="@string/app_name">
 <activity android:name=".MainActivity"
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 </application>
 <uses-sdk android:minSdkVersion="8" />
 <uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
</manifest>
```

- Because sending SMS messages incurs additional costs on the user's end, indicating the SMS permissions in the AndroidManifest.xml file enables users to decide whether to allow the application to install or not.

## MainActivity.java

```
package net.learn2develop.SMS;

import android.app.Activity;
import android.os.Bundle;

import android.app.PendingIntent;
import android.content.Intent;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
 Button btnSendSMS;
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 setContentView(R.layout.main);

 btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
 btnSendSMS.setOnClickListener(new View.OnClickListener()
 {
 public void onClick(View v)
 {
 sendSMS("5556", "Hello my friends!");
 }
 });
 }

 /**---sends an SMS message to another device---
 private void sendSMS(String phoneNumber, String message)
 {
 SmsManager sms = SmsManager.getDefault();
 sms.sendTextMessage(phoneNumber, null, message, null, null);
 }
}
```

- To send an SMS message programmatically, you use the `SmsManager` class. Unlike other classes, you do not directly instantiate this class; instead, you call the `getDefault()` static method to obtain a `SmsManager` object. You then send the SMS message using the `sendTextMessage()` method:

```
private void sendSMS(String phoneNumber, String message)
{
 SmsManager sms = SmsManager.getDefault();
 sms.sendTextMessage(phoneNumber, null, message, null, null);
}
```

Following are the five arguments to the `sendTextMessage()` method:

- destinationAddress** — Phone number of the recipient
- scAddress** — Service center address; use null for default SMSC
- text** — Content of the SMS message
- sentIntent** — Pending intent to invoke when the message is sent
- deliveryIntent** — Pending intent to invoke when the message has been delivered



### **Sending SMS**

```
//---sends an SMS message to another device---
private void sendSMS(String phoneNumber, String message)
{
 String SENT = "SMS_SENT";
 String DELIVERED = "SMS_DELIVERED";
 PendingIntent sentPI = PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
 PendingIntent deliveredPI = PendingIntent.getBroadcast(this, 0, new Intent(DELIVERED), 0);
```

```

//---when the SMS has been sent---
registerReceiver(new BroadcastReceiver(){
 @Override
 public void onReceive(Context arg0, Intent arg1) {
 switch (getResultCode())
 {
 case Activity.RESULT_OK:
 Toast.makeText(getBaseContext(), "SMS sent",
 Toast.LENGTH_SHORT).show();
 break;
 case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
 Toast.makeText(getBaseContext(), "Generic failure",
 Toast.LENGTH_SHORT).show();
 break;
 case SmsManager.RESULT_ERROR_NO_SERVICE:
 Toast.makeText(getBaseContext(), "No service",
 Toast.LENGTH_SHORT).show();
 break;
 case SmsManager.RESULT_ERROR_NULL_PDU:
 Toast.makeText(getBaseContext(), "Null PDU",
 Toast.LENGTH_SHORT).show();
 break;
 case SmsManager.RESULT_ERROR_RADIO_OFF:
 Toast.makeText(getBaseContext(), "Radio off",
 Toast.LENGTH_SHORT).show();
 break;
 }
 }
}, new IntentFilter(SENT));

```

```

//---when the SMS has been delivered---
registerReceiver(new BroadcastReceiver(){
 @Override
 public void onReceive(Context arg0, Intent arg1) {
 switch (getResultCode())
 {
 case Activity.RESULT_OK:
 Toast.makeText(getBaseContext(), "SMS delivered",
 Toast.LENGTH_SHORT).show();
 break;
 case Activity.RESULT_CANCELED:
 Toast.makeText(getBaseContext(), "SMS not delivered",
 Toast.LENGTH_SHORT).show();
 break;
 }
 }
}, new IntentFilter(DELIVERED));

SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
 btnSendSMS.setOnClickListener(new View.OnClickListener()
 {
 public void onClick(View v)
 {
 //sendSMS("5556", "Hello my friends!");
 Intent i = new
 Intent(android.content.Intent.ACTION_VIEW);
 i.putExtra("address", "5556; 5558; 5560");

 i.putExtra("sms_body", "Hello my friends!");
 i.setType("vnd.android-dir/mms-sms");
 startActivity(i);
 }
 });
}

```

- Using the SmsManager class, you can send SMS messages from within your application without the need to involve the built-in Messaging application. However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it do all the work of sending the message.



### Receiving SMS

```
<receiver android:name=".SMSReceiver">
 <intent-filter>
 <action android:name=
 "android.provider.Telephony.SMS_RECEIVED" />
 </intent-filter>
</receiver>
</application>
<uses-sdk android:minSdkVersion="8" />
<uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
<uses-permission android:name="android.permission.RECEIVE_SMS">
</uses-permission>
```

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;
```

```

public class SMSReceiver extends BroadcastReceiver
{
 @Override
 public void onReceive(Context context, Intent intent)
 {
 //---get the SMS message passed in---
 Bundle bundle = intent.getExtras();
 SmsMessage[] msgs = null;
 String str = "";
 if (bundle != null)
 {
 //---retrieve the SMS message received---
 Object[] pdus = (Object[]) bundle.get("pdus");
 msgs = new SmsMessage[pdus.length];
 for (int i=0; i<msgs.length; i++){
 msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
 str += "SMS from " + msgs[i].getOriginatingAddress();

 str += " : ";
 str += msgs[i].getMessageBody().toString();
 str += "\n";
 }
 //---display the new SMS message---
 Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
 }
 }
}

```

- To listen for incoming SMS messages, you create a BroadcastReceiver class. The BroadcastReceiver class enables your application to receive intents sent by other applications using the sendBroadcast() method. Essentially, it enables your application to handle events raised by other applications. When an intent is received, the onReceive() method is called; hence, you need to override this. When an incoming SMS message is received, the onReceive() method is fired.
- The SMS message is contained in the Intent object (intent; the second parameter in the onReceive() method) via a Bundle object. The messages are stored in an Object array in the PDU format. To extract each message, you use the static createFromPdu() method from the SmsMessage class. The SMS message is then displayed using the Toast class. The phone number of the sender is obtained via the getOriginatingAddress() method.
- One interesting characteristic of the BroadcastReceiver is that you can continue to listen for incoming SMS messages even if the application is not running; as long as the application is installed on the device, any incoming SMS messages will be received by the application.



## Lecture 44 & 45

### Notifications

- Status Bar Notifications
  - A status bar notification adds an icon to the system's status bar (with an optional ticker-text message) and an expanded message in the "Notifications" window.
  - When the user selects the expanded message, Android fires an [Intent](#) that is defined by the notification (usually to launch an [Activity](#)).
  - You can also configure the notification to alert the user with a sound, a vibration, and flashing lights on the device.
- A status bar notification should be used for any case in which a background Service needs to alert the user about an event that requires a response.
- A background Service **should never** launch an Activity on its own in order to receive user interaction.
- The Service should instead create a status bar notification that will launch the Activity when selected by the user.



- An [Activity](#) or [Service](#) can initiate a status bar notification. Because an Activity can perform actions only while it is active and in focus, you should create your status bar notifications from a Service.
- This way, the notification can be created from the background, while the user is using another application or while the device is asleep. To create a notification, you must use two classes: [Notification](#) and [NotificationManager](#).
- Use an instance of the [Notification](#) class to define the properties of your status bar notification, such as the status bar icon, the expanded message, and extra settings such as a sound to play.
- The [NotificationManager](#) is an Android system service that executes and manages all Notifications.
- In order to give it your Notification, you must retrieve a reference to the NotificationManager with [getSystemService\(\)](#) and then, when you want to notify the user, pass it your Notification object with [notify\(\)](#).

- Get Reference to Notification Manager

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager) getSystemService(ns);
```

- Instantiate the Notification:

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Hello";
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);
```

- Define the Notification's expanded message and Intent:

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);

notification.setLatestEventInfo(context, contentTitle, contentText, contentIntent);
```

- Pass the Notification to the NotificationManager:

```
private static final int HELLO_ID = 1;

mNotificationManager.notify(HELLO_ID, notification);
```

- Fancy Stuff
  - Adding Sound
  - Add Vibration
  - Adding Flashing Lights

## Widgets

- Widgets are little applications which can be placed on the home screen of your Android device.
- An widget gets its data on a periodic timetable. There are two methods to update a widget, one is based on an XML configuration file and the other is based on AlarmManager.
- Define a layout file for your widget.
- Maintain an XML file (AppWidgetProviderInfo) which describes the properties of the widget, e.g. size or fixed update frequency.
- Create a broadcast receiver which will be used to update the widgets. This receiver extends AppWidgetProvider which provides additional lifecycle hooks for widgets.
- Maintain the App Widget configuration in the "AndroidManifest.xml" file.
- Optional you can also specify a configuration activities which is called once one instance of the widget is added to the homescreen.
- To save power consumption a widget does not have its own process but is part of the homescreen process.
- Therefore the widget UI is created as a RemoteView. A RemoteView can be executed by another process with the same permissions as the original application.
- In the widget configuration file you can specify a fixed update interval. The system will wake up after this time interval and call your broadcast receiver to update the widget. The smallest update interval is 180000 milliseconds (30 minutes).
- `<?xml version="1.0" encoding="utf-8"?>`
- `<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android" android:updatePeriodMillis="180000" android:initialLayout="@layout/widget_layout" android:minHeight="72dp" android:minWidth="146dp"> </appwidget-provider>`
- AppWidget Provider
- Your broadcast receiver extends AppWidgetProvider. The AppWidgetProvider class implements the onReceive() method, extracts the required information and calls the following widget lifecycle methods.

<code>onEnabled()</code>	Called the first time an instance of your widget is added to the <u>homescreen</u>
<code>onDisabled()</code>	Called once the last instance of your widget is removed from the <u>homescreen</u> .
<code>onUpdate()</code>	Called for every update of the widget. Contains the ID's of the widgets currently on the <u>homescreen</u> .
<code>onDeleted()</code>	Widget instance is removed from the <u>homescreen</u>

## Creating Database / Persistence

- Android incorporates SQLite (version 3 to be precise), a server-less, transactional database engine for this purpose.
- Inside the onCreate() method declare and assign an SQLiteDatabase as follows:
- `SQLiteDatabase db;`
- `db = openOrCreateDatabase("my_database.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);`

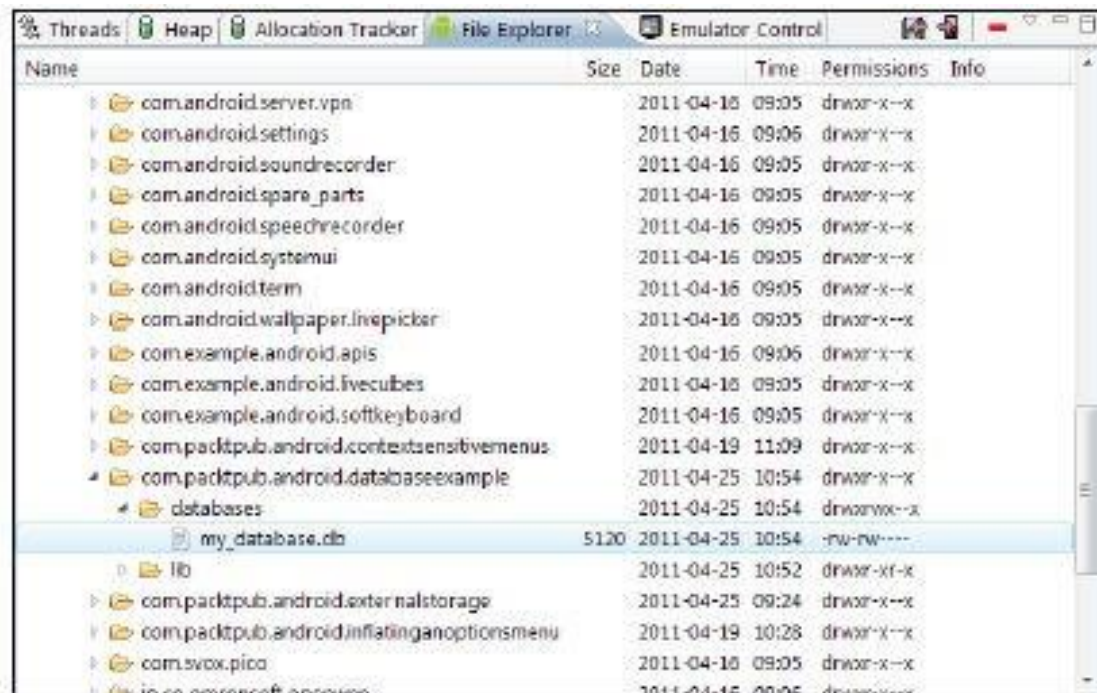
- we can use the SQLiteDatabase method `openOrCreateDatabase()` to produce one from scratch, as we did here. The `openOrCreateDatabase()` function takes a string name, an SQL mode, and a CursorFactory object, which is a public interface for handling Cursor objects. The flag that we used in the SQL mode, `CREATE_IF_NECESSARY`, is just one of several we could have applied—the others being `OPEN_READWRITE`, `OPEN_READONLY`, and `NO_LOCALIZED_COLLATORS`.
- Underneath this, define a table for our database in the following manner:  

```
final String CREATE_TABLE_CITIES = "CREATE TABLE tb_cities (" + "id INTEGER PRIMARY KEY AUTOINCREMENT," + "city_name TEXT);";
```
- Then execute this SQL statement: `db.execSQL(CREATE_TABLE_CITIES);`
- We put together an SQL statement, `CREATE TABLE tb_cities (id INTEGER PRIMARY KEY AUTOINCREMENT,city_name TEXT)`, which we concatenated into a string constant, `CREATE_TABLECITIES`, for clarity. Any SQL statement can be constructed in this way and executed with `execSQL()`.
- Now using a ContentValues object insert some entries:  

```
ContentValues cv = new ContentValues();
cv.put("city_name", "Aberdeen");
db.insert("tb_cities", null, cv);
cv.put("city_name", "Dundee");
db.insert("tb_cities", null, cv);
```
- Inserting data is something we will need to do often with a working database and here we took advantage of the ContentValues class, which allows us to pass values to a ContentResolver that provides us access to the underlying grammar or content model.

Name	Object	Type	Schema
android_metadata	table		CREATE TABLE android_metadata (locale TEXT)
locale	field	TEXT	
tb_cities	table		CREATE TABLE tb_cities (id INTEGER PRIMARY KEY AUTOINCREMENT,city_name TEXT)
id	field	INTEGER PRIMARY KEY	
city_name	field	TEXT	
sqlite_sequence	table		CREATE TABLE sqlite_sequence(name,seq)
name	field		
seq	field		

- And finally close the database:
- `db.close();`



- Making a database thread safe
- To apply locks around our database and therefore making it thread safe, use:
  - `db.setLockingEnabled(true);`
- Versioning a database
- An Android SQLite database can be versioned with:
  - `db.setVersion(2);`

### Detecting User Activity

- Reading a device's orientation
  - Measuring motion with the accelerometer
  - Listing available sensors
  - Recognizing a touch event
  - Detecting multi-touch elements
  - Recognizing gestures
  - Handling multi-touch gestures
  - Controlling on screen keyboards
- 
- If one aspect of modern smartphones makes them stand out from other digital devices, it is surely the large array of sensors that can be found on-board. Android handsets can detect speed, motion, gravitational pull, and even the Earth's magnetic field.
  - Combined with sensitive touch-screens capable of reading complex gestures, these new forms of input device make smartphones among the most versatile and fun devices a programmer can get his or her hands on.
  - Android provides us with many handy tools for detecting user activity of this nature along with an intuitive series of callbacks and listeners.

## Reading Device Orientation

- Open the AndroidManifest.xml file and inside the <activity> node add a configChanges element with the following value:

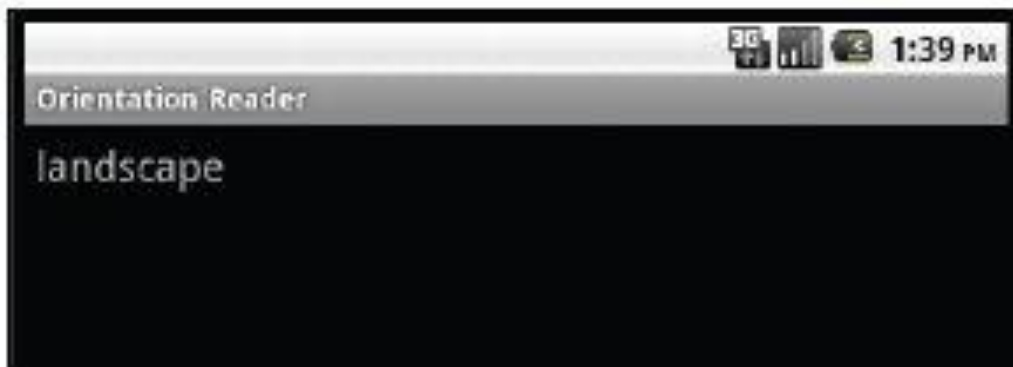
```
<activity
 android:name=".OrientationReader"
 android:configChanges="orientation"
 android:label="@string/app_name">
 ...
</activity>
```

- Open the Java activity window and form an association between a private class member, mTextView, and the TextView that we created in XML:

```
mTextView = (TextView) findViewById(R.id.text_box);
```

Override the onConfigurationChanged() method as follows:

```
@Override
public void onConfigurationChanged(Configuration config) {
 super.onConfigurationChanged(config);
 if (config.orientation == Configuration.ORIENTATION_LANDSCAPE) {
 mTextView.setText("landscape");
 } else if (config.orientation == Configuration.ORIENTATION_PORTRAIT) {
 mTextView.setText("portrait");
 }
}
```



## Measuring Speed with Accelerometer

- There are a wide and growing variety of sensors that can be found on an Android handset, from accelerometers and gyroscopes to light and proximity sensors. Most of these devices can be accessed with the android.hardware.SensorEvent class, although naturally they each produce their own specific data sets.
- Gathering information from sensors is quite straightforward as Android provides a handy interface, android.hardware.SensorEventListener, to facilitate this.
- Inside our main activity's Java class edit the declaration so that it implements SensorEventListener, like so:

```
public class MotionDetector extends Activity implements SensorEventListener {
```

- Just below this, declare a class field of type `SensorManager`:  
`private SensorManager mSensorManager;`
- At the end of the `onCreate()` method add the following `SensorManager` assignment:
- `mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);`
- We need to disable our sensors when we are not using them, so add an `onPause()` and an `onResume()` method, completed as seen here:

`@Override`

```
protected void onResume() {
 super.onResume();
 mSensorManager.registerListener(this,
 mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
 SensorManager.SENSOR_DELAY_UI);
}
```

- Registering the listener, on the other hand, was slightly more complex, requiring a `SensorEventListener`, an `int` type, which here was the default accelerometer, and a delay value. This delay (`int`) value controls how quickly the sensor operates and can have a dramatic affect on power usage. There are four settings:
- `ffSENSOR_DELAY_FASTEST`
- `ffSENSOR_DELAY_GAME`
- `ffSENSOR_DELAY_NORMAL`
- `ffSENSOR_DELAY_UI`

`@Override`

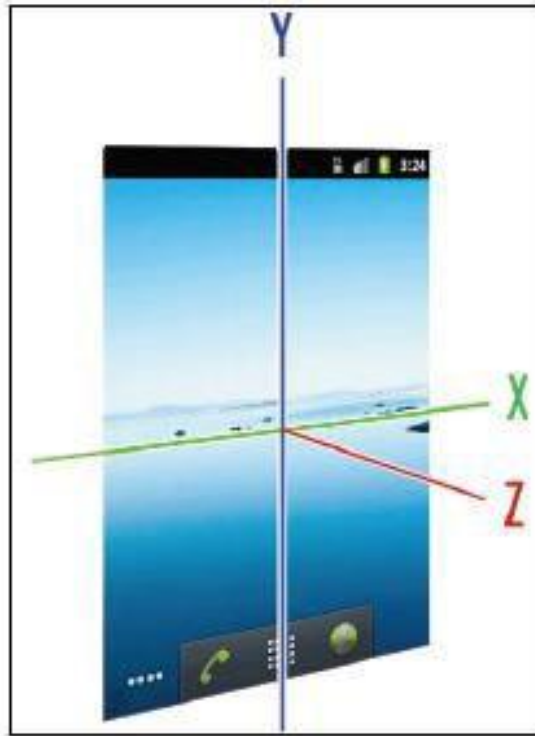
```
protected void onPause() {
 super.onPause();
 mSensorManager.unregisterListener(this);
}
```

- Many sensors, including the accelerometer, are a powerful drain on a device's battery so we need to disable them when not in use. We did so in the `onPause()` callback using `SensorManager's unregisterListener(SensorEventListener)` method.

fill out the body of the `onSensorChanged()` method as follows:

```
public void onSensorChanged(SensorEvent e) {
 synchronized (this) {
 if (e.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
 mTextView.setText("x= " + e.values[0] + "\ny= " + e.values[1] + "\nz= " + e.values[2]);
 }
 }
}
```

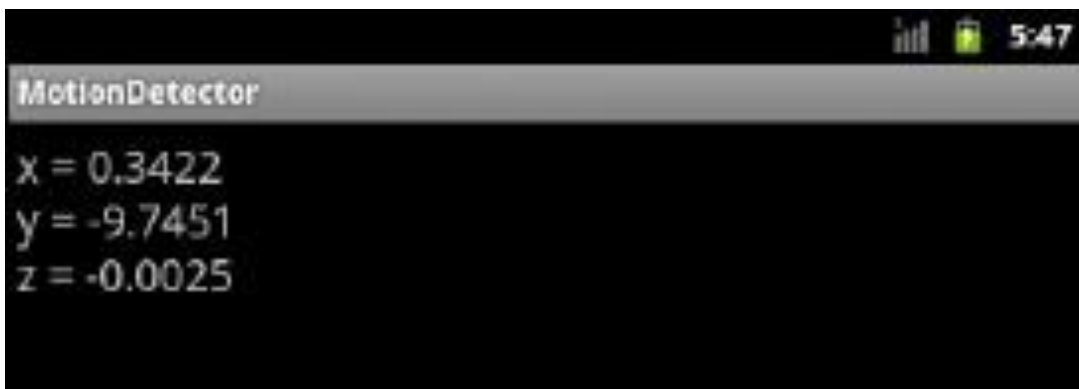
- The actual reading of the accelerometer's values was done in the `onSensorChanged()`. The accelerometer, along with the magnetic field sensor and the gyroscope, returns three values based on a coordinate system. In our example here the three values describe the acceleration of the device along each of the three axes of this coordinate system measured in `m/s2`



The `onAccuracyChanged()` method is not used in this example but must be included anyway—you can leave it like this:

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {
 // TODO Auto-generated method stub
}
```

- Reading a sensor's values requires several components, including the `SensorEventListener` interface that implements two callbacks which we use to respond to changes in sensor values
- or accuracy. The `onAccuracyChanged()` callback is required less often but is nevertheless useful as demands on battery and environmental conditions can cause this setting to change.
- The possible constant int values for this can be:
  - `SENSOR_STATUS_ACCURACY_HIGH`,
  - `SENSOR_STATUS_ACCURACY_MEDIUM`
  - `SENSOR_STATUS_ACCURACY_LOW`
- Provided you have assigned the `TextView` with `findViewById()`, this program can be run on any emulator or handset with an accelerometer:



### Other Types of Sensors

- TYPE\_ACCELEROMETER
- TYPE\_ALL
- TYPE\_GRAVITY
- TYPE\_GYROSCOPE
- TYPE\_LIGHT
- TYPE\_LINEAR\_ACCELERATION
- TYPE\_MAGNETIC\_FIELD
- TYPE\_ORIENTATION
- TYPE\_PRESSURE
- TYPE\_PROXIMITY
- TYPE\_ROTATION\_VECTOR
- TYPE\_TEMPERATURE

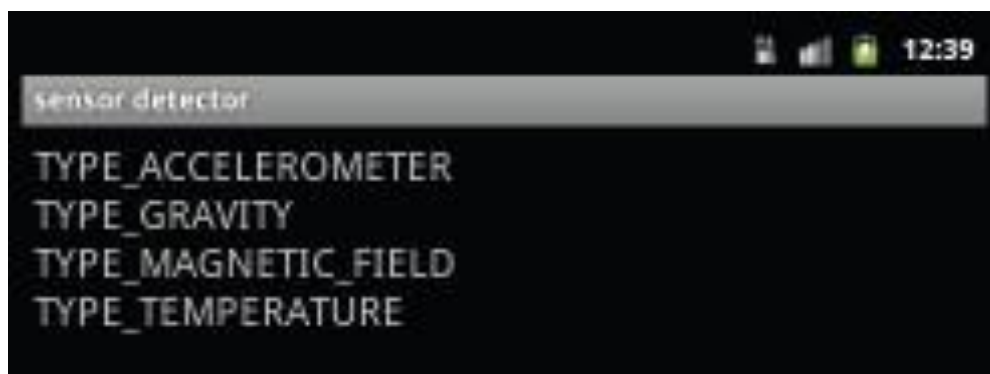
### Listing Available Sensors

- Android handsets come with a wide variety of sensors but which sensors are included is a matter for manufacturers to decide and differs from model to model.
- As developers we need some way to detect which sensors are available to us. In particular we may want to select between sensors that perform similar functions.
- For example it may be preferable to measure motion with a gyroscope if one is available but prepare a function that utilizes the accelerometer when one is not.
- Open up the Java activity file and declare the following private members:  

```
private SensorManager mSensorManager;
private TextView mTextView;
private List mList;
```
- Inside the onCreate() method assign our SensorManager and TextView like this:  

```
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
mTextView = (TextView) findViewById(R.id.text_view);
```
- Finally, add the following statement block just beneath this:  

```
mList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
for (int i = 1; i < mList.size(); i++) {
 mTextView.append("\n" + mList.get(i));
}
```



**Other Functionalities**

- Recognizing a Touch Event
- Detecting Multi-Touch Event
- Recognizing Gestures
- Controlling on-screen Keyboard