**بسم الله الرحمن الرحيم**

**An-Najah University**

**Faculty of Engineering**

**Computer Engineering Department**

****

**Report of project**

**Name of project: Quick Services**

**Supervisor Name: D. Aladdin Masri.**

**Name Student: Wala’ Amjad Faqeh.**

**Table of contents:**

**Name of content…………………………Number of page**

**Abstract ……………………………… 4**

**Introduction …………………………….. 5**

**Chapter 1: Problem Description ………………….. 6**

**1.1 general Description ……………… 6**

**1.2: introduction of android …………6-15**

**1.3: 32feet.net ………………….. 16**

**1. 4:RSS…………………… 18-19**

**Chapter 2:server details ……… 20**

**2.1:how server work …………………. 20-21**

**2.2:web sites that used to fetch RSS ……… 22**

**Chapter 3: client part…………………… 23**

**Chapter 5:Result ………………………… 26**

**Conclusion and future work……………..27**

**References ………………………… 28**

**Appendix …………………………….. 29**

**Table of figures:**

|  |  |  |
| --- | --- | --- |
| **Figure** | **Name of Figure** | **Number of page** |
| Figure 1 | the major components of the Android operating system. | 7 |
| Table 1 | 32feet.net namespaces and description | 16-17 |
| Figure 2 | shape RSS in web site | 22 |
| Figure 3 | interface the client. | 24 |
| Figure 4 | request the Bluetooth in client. | 24 |
| Figure 5 | server –client procedures | 25 |

**Abstract:**

We have developed mobile application . This project must get all information that considers important part in our life such as news, currency and weather. So must supply this information in fast and cheap method. The project contains client and server to offer the goals of project.

This project supply user the services of news, weather, currency, and another services from the server, without pay money, and for every user that around this server. This used for all mobiles (old, new, complex and simple).

**Introduction:**

As mobile devices become more like PC’s they will com to replace object we tend to carry around such as checkbooks, credit card,cameras,planners,mp3 players,etc.In short I have developed mobile application.

We have developed mobile application .This project must get all information that considers important part in our life such as news, currency and weather. This information tell you news of the world. You can know what around you .news of currency is very useful in banks ,in sells and in some business but must come in correct time.

Used mobile application is very important in this modern life. It is very simple and very suitable to every person. The mobile development community is a tipping point .mobile users demand more choice, more opportunities to customize their phones, and more functionality.

Usually, these services are available in many web site. you can choose correct one and open this web site .This required computer ,internet every users, but if you outside the home ,you couldn’t do this and couldn’t reach to information so maybe damage your life.

Instead we go to computer and connect with internet and search web sites ,we developed project for all users around one server to reach services in simple way and short time.

This application is free. I mean, you don’t pay money to fetch this services if used this application. I would to describe this later.

***Chapter1:* Problem Description:**

**1.1: general description**

In this project, we have developed application on mobile for users that can use it to reach to main server that give service that user need it. The project contains client and server to offer the goals of project. The client request and the server must response and fetch information, then send to client, analyze this information to display to user, I describe this later.

The Bluetooth support this project .we used this because Bluetooth available to all old, new, simple, complex mobiles. This makes my application to used from all users. Some mobile don’t contain another technology.

We can control the Bluetooth by library .this can open Bluetooth, lessen to request and response and another advantage that I say it later.

**1.2: android application.**

In mobile application, I used android. Android has emerged as a new mobile development platform, building on past successes and avoiding past failures of other platforms. Android was designed to empower the developer to write innovative applications. The platform is open source licensing so developers have unprecedented access to the handset features when developing applications, with no up-font fees, and developers enjoy many benefits over other competing platforms.

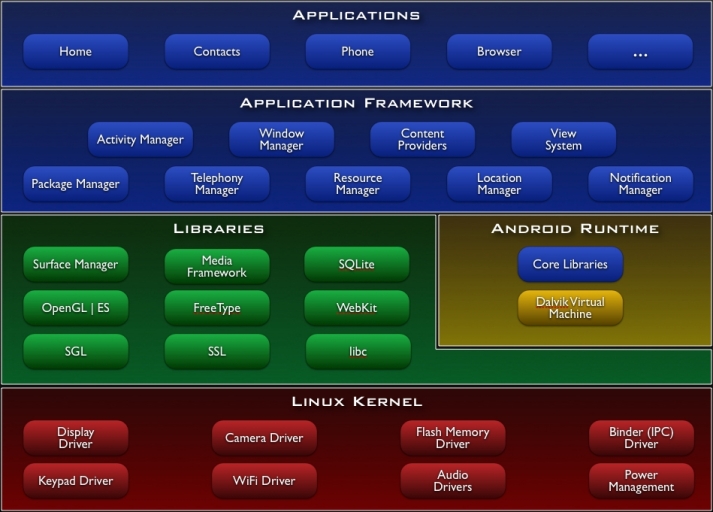
Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The [Android SDK](http://developer.android.com/sdk/index.html)provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

**Features**

* **Application framework** enabling reuse and replacement of components
* **Dalvik virtual machine** optimized for mobile devices
* **Integrated browser** based on the open source [WebKit](http://webkit.org/) engine
* **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
* **SQLite** for structured data storage
* **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
* **GSM Telephony** (hardware dependent)
* **Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
* **Camera, GPS, compass, and accelerometer** (hardware dependent)
* **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Android Architecture

The following diagram shows the major components of the Android operating system. Each section is described in more detail below.



**Figure1:** the major components of the Android operating system.

**Applications**

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

**Application Framework**

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more.

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

Underlying all applications is a set of services and systems, including:

* A rich and extensible set of [Views](http://developer.android.com/resources/tutorials/views/index.html) that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
* [Content Providers](http://developer.android.com/guide/topics/providers/content-providers.html) that enable applications to access data from other applications (such as Contacts), or to share their own data
* A [Resource Manager](http://developer.android.com/guide/topics/resources/resources-i18n.html), providing access to non-code resources such as localized strings, graphics, and layout files
* A [Notification Manager](http://developer.android.com/reference/android/app/NotificationManager.html) that enables all applications to display custom alerts in the status bar
* An [Activity Manager](http://developer.android.com/reference/android/app/Activity.html) that manages the lifecycle of applications and provides a common navigation backstack

For more details and a walkthrough of an application, see the [Notepad Tutorial](http://developer.android.com/resources/tutorials/notepad/index.html).

Libraries of android:

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

* **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
* **Media Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
* **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
* **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
* **SGL** - the underlying 2D graphics engine
* **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
* **FreeType** - bitmap and vector font rendering
* **SQLite** - a powerful and lightweight relational database engine available to all applications

Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

**Application Fundamental:**

Android applications are written in the Java programming language. The Android SDK tools compile the code—along with any data and resource files—into an *Android package*, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

Once installed on a device, each Android application lives in its own security sandbox:

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

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

However, there are ways for an application to share data with other applications and for an application to access system services:

* It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate).
* An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

That covers the basics regarding how an Android application exists within the system. The rest of this document introduces you to:

* The core framework components that define your application.
* The manifest file in which you declare components and required device features for your application.
* Resources that are separate from the application code and allow your application to gracefully optimize its behavior for a variety of device configurations.

**Application Components**

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your application's overall behavior.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of application components:

Activities

An *activity* represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of [Activity](http://developer.android.com/reference/android/app/Activity.html) and you can learn more about it in the [Activities](http://developer.android.com/guide/topics/fundamentals/activities.html)developer guide.

Services

A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of [Service](http://developer.android.com/reference/android/app/Service.html) and you can learn more about it in the [Services](http://developer.android.com/guide/topics/fundamentals/services.html) developer guide.

Content providers

A *content provider* manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as [ContactsContract.Data](http://developer.android.com/reference/android/provider/ContactsContract.Data.html)) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the [Note Pad](http://developer.android.com/resources/samples/NotePad/index.html) sample application uses a content provider to save notes.

A content provider is implemented as a subclass of [ContentProvider](http://developer.android.com/reference/android/content/ContentProvider.html)and must implement a standard set of APIs that enable other applications to perform transactions. For more information, see the [Content Providers](http://developer.android.com/guide/topics/providers/content-providers.html) developer guide.

Broadcast receivers

A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may [create a status bar notification](http://developer.android.com/guide/topics/ui/notifiers/notifications.html)to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of [BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html)and each broadcast is delivered as an [Intent](http://developer.android.com/reference/android/content/Intent.html) object. For more information, see the [BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html) class.

A unique aspect of the Android system design is that any application can start another application’s component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application.

When the system starts a component, it starts the process for that application (if it's not already running) and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process. Therefore, unlike applications on most other systems, Android applications don't have a single entry point (there's no main() function, for example).

Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

**Activating Components**

Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.

An intent is created with an [Intent](http://developer.android.com/reference/android/content/Intent.html) object, which defines a message to activate either a specific component or a specific type of component—an intent can be either explicit or implicit, respectively.

For activities and services, an intent defines the action to perform (for example, to "view" or "send" something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an [Intent](http://developer.android.com/reference/android/content/Intent.html) (for example, you can issue an intent to let the user pick a personal contact and have it returned to you—the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates "battery is low").

The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a [ContentResolver](http://developer.android.com/reference/android/content/ContentResolver.html). The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the [ContentResolver](http://developer.android.com/reference/android/content/ContentResolver.html) object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

There are separate methods for activiting each type of component:

* You can start an activity (or give it something new to do) by passing an [Intent](http://developer.android.com/reference/android/content/Intent.html) to [startActivity()](http://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) or [startActivityForResult()](http://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int))(when you want the activity to return a result).
* You can start a service (or give new instructions to an ongoing service) by passing an [Intent](http://developer.android.com/reference/android/content/Intent.html) to [startService()](http://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). Or you can bind to the service by passing an [Intent](http://developer.android.com/reference/android/content/Intent.html) to[bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)).
* You can initiate a broadcast by passing an [Intent](http://developer.android.com/reference/android/content/Intent.html) to methods like[sendBroadcast()](http://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)), [sendOrderedBroadcast()](http://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String)), or [sendStickyBroadcast()](http://developer.android.com/reference/android/content/Context.html#sendStickyBroadcast(android.content.Intent)).
* You can perform a query to a content provider by calling [query()](http://developer.android.com/reference/android/content/ContentProvider.html#query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String)) on a [ContentResolver](http://developer.android.com/reference/android/content/ContentResolver.html).

For more information about using intents, see the [Intents and Intent Filters](http://developer.android.com/guide/topics/intents/intents-filters.html) document. More information about activating specific components is also provided in the following documents: [Activities](http://developer.android.com/guide/topics/fundamentals/activities.html), [Services](http://developer.android.com/guide/topics/fundamentals/services.html), [BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html) and [Content Providers](http://developer.android.com/guide/topics/providers/content-providers.html).

The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application must declare all its components in this file, which must be at the root of the application project directory.

The manifest does a number of things in addition to declaring the application's components, such as:

* Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.
* Declare the minimum [API Level](http://developer.android.com/guide/appendix/api-levels.html)required by the application, based on which APIs the application uses.
* Declare hardware and software features used or required by the application, such as a camera, bluetooth services, or a multitouch screen.
* API libraries the application needs to be linked against (other than the Android framework APIs), such as the [Google Maps library](http://code.google.com/android/add-ons/google-apis/maps-overview.html).
* And more

Declaring components

The primary task of the manifest is to inform the system about the application's components. For example, a manifest file can declare an activity as follows:

<?xml version="1.0" encoding="utf-8"?>  
<manifest ... >  
 <application android:icon="@drawable/app\_icon.png" ... >  
 <activity android:name="com.example.project.ExampleActivity"  
 android:label="@string/example\_label" ... >  
 </activity>  
 ...  
 </application>  
</manifest>

In the [<application>](http://developer.android.com/guide/topics/manifest/application-element.html)element, the android:icon attribute points to resources for an icon that identifies the application.

In the [<activity>](http://developer.android.com/guide/topics/manifest/activity-element.html) element, the android:name attribute specifies the fully qualified class name of the [Activity](http://developer.android.com/reference/android/app/Activity.html) subclass and the android:label attributes specifies a string to use as the user-visible label for the activity.

You must declare all application components this way:

* [<activity>](http://developer.android.com/guide/topics/manifest/activity-element.html) elements for activities
* [<service>](http://developer.android.com/guide/topics/manifest/service-element.html) elements for services
* [<receiver>](http://developer.android.com/guide/topics/manifest/receiver-element.html) elements for broadcast receivers
* [<provider>](http://developer.android.com/guide/topics/manifest/provider-element.html) elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run. However, broadcast receivers can be either declared in the manifest or created dynamically in code (as[BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html) objects) and registered with the system by calling[registerReceiver()](http://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter)).

For more about how to structure the manifest file for your application, see the [The AndroidManifest.xml File](http://developer.android.com/guide/topics/manifest/manifest-intro.html)documentation.

Declaring component capabilities

As discussed above, in [Activating Components](http://developer.android.com/guide/topics/fundamentals.html#ActivatingComponents), you can use an[Intent](http://developer.android.com/reference/android/content/Intent.html) to start activities, services, and broadcast receivers. You can do so by explicitly naming the target component (using the component class name) in the intent. However, the real power of intents lies in the concept of intent actions. With intent actions, you simply describe the type of action you want to perform (and optionally, the data upon which you’d like to perform the action) and allow the system to find a component on the device that can perform the action and start it. If there are multiple components that can perform the action described by the intent, then the user selects which one to use.

The way the system identifies the components that can respond to an intent is by comparing the intent received to the *intent filters* provided in the manifest file of other applications on the device.

Used Programs:

Eclipse program for Android development.

Microsoft visual studio 2010 (c#) in server side.

**1.3: 32feet.NET**

Bluetooth support requires a device with the Microsoft or Broadcom(Widcomm) Bluetooth stack. Requires .NET Compact Framework v2.0 or above and Windows CE.NET 4.2 or above, or .NET Framework v2.0 for desktop Windows XP SP2, Windows Vista and Windows 7.

32feet.NET is a shared-source project to make personal area networking technologies such as Bluetooth, Infrared (IrDA) and more, easily accessible from .NET code. It supports desktop, mobile or embedded systems. The project currently consists of the following technologies:-

* Bluetooth
* IrDA
* Object Exchange

In this project I used this library to support Bluetooth in server side.

The table below show the namespace that used and som description for this library:

|  |  |
| --- | --- |
| **Namespace** | **Description** |
|  |  |
| **InTheHand.Net** | **The InTheHand.Net namespace contains classes for working with addressing on personal area networks.**  **The code that used:**  **\*** ObexListener d = new ObexListener();  d.Start();  ObexListenerContext ctx = d.GetContext();  ObexListenerRequest req = ctx.Request;  **\***  Uri[] u = new Uri[count];  ObexWebResponse[] response=new ObexWebResponse[count];  ObexWebRequest[] owr=new ObexWebRequest[count];  for (int leng\_uri = 0; leng\_uri < count; leng\_uri++)  {  u[leng\_uri] = null;  response[leng\_uri] = null;  owr[leng\_uri] = null;  } |
| **InTheHand.Net.Bluetooth** | **The InTheHand.Net.Bluetooth namespace contains classes for**  **working with Bluetooth functionality such as Radio hardware.** |
| [**InTheHand.Net.Bluetooth.AttributeIds**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Net_Bluetooth_AttributeIds.htm) | **The InTheHand.Net.Bluetooth.AttributeIds namespace contains definitions of Service Discovery Protocol attributes.** |
| [**InTheHand.Net.Bluetooth.Factory**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Net_Bluetooth_Factory.htm) |  |
| [**InTheHand.Net.Bluetooth.Widcomm**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Net_Bluetooth_Widcomm.htm) |  |
|  |  |
| [**InTheHand.Net.Mime**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Net_Mime.htm) | **The InTheHand.Net.Mime namespace holds types that are used to represent Multipurpose Internet Mail Exchange (MIME) headers** |
| [**InTheHand.Net.Ports**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Net_Ports.htm) | **The InTheHand.Net.Ports namespace contains classes for working with legacy virtual COM ports over Bluetooth.** |
| [**InTheHand.Net.Sockets**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Net_Sockets.htm) | **The InTheHand.Net.Sockets namespace provides added functionality for working with IrDA and Bluetooth Sockets.**  **The code that used is:**  BluetoothClient bc = new InTheHand.Net.Sockets.BluetoothClient();  BluetoothDeviceInfo[] array = bc.DiscoverDevices(); |
| [**InTheHand.Windows.Forms**](mk:@MSITStore:C:\Program%20Files\32feet.NET\Help\ThirtyTwoFeet.chm::/html/N_InTheHand_Windows_Forms.htm) | **The InTheHand.Windows.Forms namespace contains forms related to networking functionality.** |

Table 1 :32feet.net namespaces and description

**1.4 RSS:**

### What is RSS?

RSS is a technology that is being used by millions of web users around the world to keep track of their favorite websites.

In the ‘old days’ of the web to keep track of updates on a website you had to ‘bookmark’ websites in your browser and manually return to them on a regular basis to see what had been added.

### The problems with bookmarking

* You as the web surfer had to do all the work
* It can get complicated when you are trying to track many websites at once
* You miss information when you forget to check your bookmarks
* You end up seeing the same information over and over again on sites that don’t update very often

### RSS Changes Everything

What if you could tell a website to let you know every time that they update? In a sense, this is what RSS does for you.

RSS flips things around a little and is a technology that provides you with a method of getting relevant and up to date information sent to you for you to read in your own time. It saves you time and helps you to get the information you want quickly after it was published.

RSS stands for ‘Really Simple Syndication’. Many people describe it as a ‘news feed’ that you subscribe to.

I find the ‘subscription’ description helpful. It’s like subscribing to a magazine that is delivered to you periodically but instead of it coming in your physical mail box each month when the magazine is published it is delivered to your ‘RSS Reader’ every time your favorite website updates.

How RSS actually technically works is probably a lesson for another day but the key today is for you to understand why it’s good and how to use it.

Let me say right up front that I’m not the most technically savvy guy going around – but even I can use RSS. At first I found it a little strange to make the change from bookmarking to RSS but I found that when I started that I just couldn’t stop.

### How to Use RSS

**Get an RSS Reader** – The first thing you’ll want to do if you’re getting into reading sites via RSS is to hook yourself up with an RSS Feed Reader.

There are many feed readers going around with a variety of approaches and features – however a good place to start is with a couple of free and easy to use web based ones like Google Reader and Bloglines. Either one will do if you’re starting out (I use Google’s Reader) – as I say there are many others to choose from but to get started either of these are fairly easy to use and will help you work out the basics of RSS.

Both of these feed readers work a little like email. As you subscribe to feeds you’ll see that unread entries from the sites you’re tracking will be marked as bold. As you click on them you’ll see the latest update and can read it right there in the feed reader. You are given the option to click through to the actual site or move onto the next unread item – marking the last one as ‘read’.

The best way to learn how to use either Google Reader or Bloglines is to simply subscribe to some feeds and give it a go. Both have helpful help sections to get you up and running.

**Note**: other options to tracking websites that you might already be familiar with include using pages like MyYahoo, MyGoogle and MyMSN.

**Find Some Feeds to Subscribe to** – there are two places to look for a site’s feed:

1. On the Site
2. In Your Browser

**Chapter 2: Server details:**

**2.1How server works**

the server must lessen to the request from client ,if there is lessen the server must know which mobile send this request .every mobile contain information device about itself .The server must know this information about all devices around it.

If mobiles send request, the information about this mobile store in server by class name is device. The information is:

* Authenticated
* Connected
* Device Name
* Last Seen
* Last Used
* Nap
* Sap
* Remembered
* address
* The server accept the request and take the file that send,the server must know.
* The name of file that send depend what user request(later description in client part)this save server from danger .
* The server store the IP address of devise that send the request .
* The name of file determine the server which web site request.
* The server make http request to web site internet.
* Server request suitable RSS.
* We will read data via the response stream.
* Fill the buffer with data.
* Translate from bytes to ASCII text.
* Convert from string to byte to used it for later.
* It make file in bin ->debug and begin to read from web site and writ in this file.
* Write these bytes on the file that exit in bin.
* The server must sure that mobile in range it’s Bluetooth
* So it compare the address of client mobile that store in server when it send request.
* If the client exist and can response to request from server .
* Server prepare file .it’s name depend what client request
* If request currency send file Bluetooth\_curency.If news send Bluetooth\_news ….
* Make socket in server to request to mobile that demand the service.
* If this mobile make response to this socket prepare to send file.
* Specifies that the URI is accessed through the Object Exchange (OBEX) Push protocol.
* This passes the file from server to client.
* Then the server must lessen again to another request from same or other mobiles.

**2.2 web sites that used to fetch RSS.**

* http://www.services.paltop.net/currancy/currancy.xml

this web site used to take the currency in sheql for all countries.

* http://services.paltop.net/weather/weather.xml fo.r wether
* http://www.maannews.net/arb/Rss.aspx?CID=SPO:for sport
* http://www.maannews.net/arb/Rss.aspx?CID=NEW

this web site used to take the news for all world.

****

Figure 2: shape RSS in web site

**Chapter3: client part.**

In this part to write Android application, we must configure our programming environment for java development. The java development kit (JDK), the eclipse development environment, and the Android SDK are available for download on the web at no cost.

The client must contain interface of the application, so user can use this application easy.

This application contains buttons that represent the services for users.

The services different from server to another server, For example, If server in library different if it in company, cafeteria…

**Step1:**

* The client exist around server, because we used Bluetooth
* If user need currency click button currency.
* If need news click button news.
* If need weather click button weather and so on.
* The mobile application that represents as client know that.
* The client create empty file called currency.
* Intent the Bluetooth application in the mobile.
* The client determines the server that would to use.
* The client requests this service from server.
* Client Send file currency through Bluetooth.
* The file transfer between client and server.
* The server response to this client.
* Server takes this file and read name of file.
* Server know the suitable file that send (I explain how this happen in previous part).

After server send the suitable file that taken from web site ,the client

receive this file.

**Step2:**

The users don’t understand the file that reach from server. It contain xml language. This file contains xml.

The android application searches if this file exits. If this occurs display another screen to user. It contains edit text and another button to return to previous directory.

The client begins get the elements of xml by tag name .depend what tag contain.

Each tag display in edit text.

After finish read xml file delete this file. This cause to fetch fresh information . Exactly user get services that happen at this time.

**Step3:**

The interface of user, depend on users. The user can change font, color, and style of activity. I used tabs for this. I used sound on every button. This sound depends on functionality of buttons.

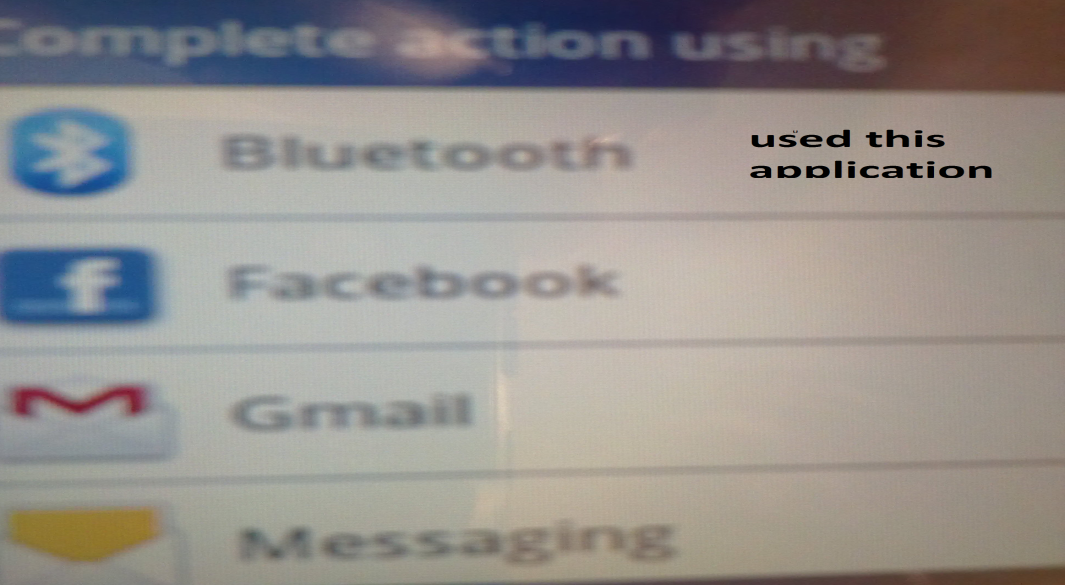
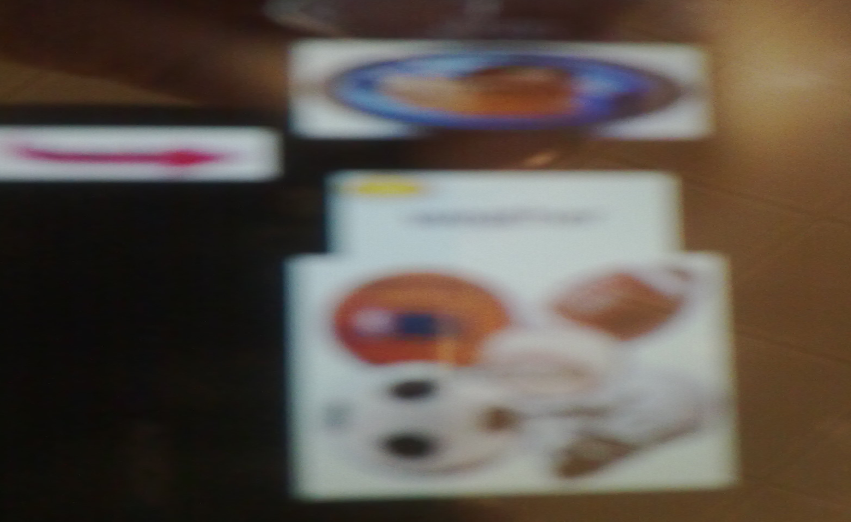


Figure 3:interface the client. Figure 4: request the Bluetooth in client.

Send request for service

Lessen to client

If there request

Store address of client

Analyze what service the client need

Fetch RSS from web page.

Send xml file to client

Response to the server

Take the file XML

Client

Analyze and read file

Server side

Display to users

Figure 5: server –client procedures

**Chapter 5: Result**

This project sends from client to server and vice versa and can success to make the goal of project.

I face some problem in this project. In server side the library 32Feet don’t know which mobile send to server. But I solved this problem.

If more one devices request services at the same time, cause overhead. So one device must wait another .we can solve this used threads that execute parallel.

If the user outside the server can’t reach it.

Same applications are available in Jawwal Company but no free. To fetch the services must pay money .this not used widely.

Advantage of this project:

* Free project.
* Reduce resources.
* Short time.

**Conclusion and Future work:**

This application is free and can be used in university, library, cafeteria and department. It can be used in companies to give features to their employee. It can be used in restaurants to gain customers.

This application can used for all types of android mobiles .we can used this application without need for every user computer and Internet.

**References:**

* http://32feet.codeplex.com.
* http://develope.android.com
* book: Android application development
* book: Pro Android2.
* http://www.alanjmcf.me.uk/comms/bluetooth/32feet.NET%20--%20User%20Guide.html
* http://32feet.net/forums/37.aspx.
* Book: Microsoft visual studio 2010.
* http://Lynda.com.

**Appendix:**

**some code of client and server.**

public Form1()

{

while (true)

{

InitializeComponent();

ObexListener d = new ObexListener();

d.Start();

ObexListenerContext ctx = d.GetContext();

ObexListenerRequest req = ctx.Request;

string jnon= ctx.Request.UserHostAddress;

EndPoint j2 = ctx.Request.RemoteEndPoint;

string j2\_s = j2.ToString();

string[] lines = Regex.Split(j2\_s, ":");

string change\_add = lines[0];

string da = req.Url.UserInfo;

string[] pathSplit = req.RawUrl.Split('/');

string file\_name = pathSplit[pathSplit.Length - 1];

req.WriteFile(file\_name);

string from = file\_name;

d.Close();

byte[] to\_send = web(from);

string read\_name = "b.txt"; //i would to change it later

Stream str = new FileStream("b.txt", FileMode.Truncate);

str.Write(to\_send, 0, to\_send.Length);

str.Close();

bg = new BackgroundWorker();

bg.RunWorkerCompleted += new RunWorkerCompletedEventHandler(bg\_RunWorkerCompleted);

if (!bg.IsBusy)

{

pb.Visible = true;

string change\_addres;

string filename = System.IO.Path.GetFileName(to\_mobile);

BluetoothClient bc = new InTheHand.Net.Sockets.BluetoothClient();

BluetoothDeviceInfo[] array = bc.DiscoverDevices();

int count = array.Length;

Uri[] u = new Uri[count];

ObexWebResponse[] response=new ObexWebResponse[count];

ObexWebRequest[] owr=new ObexWebRequest[count];

for (int leng\_uri = 0; leng\_uri < count; leng\_uri++)

{

u[leng\_uri] = null;

response[leng\_uri] = null;

owr[leng\_uri] = null;

}

for (int i = 0; i < count; i++)

{

Device device = new Device(array[i]);

// MessageBox.Show("" + device.address+""+device.DeviceName);

// devices.Clear();

}

//

// int s = 0;

bool[] yes\_respo = new bool[count];

for (int R = 0; R < count; R++)

{//for for

//s = R - 1;

addr = devices[R].address;

change\_addres = addr.ToString();

if (change\_add != change\_addres) { continue; }

u[R] = new Uri("obex-push://" + addr.ToString() + "/" + filename);

owr[R] = new ObexWebRequest(u[R]);

owr[R].ReadFile(read\_name);//

try

{

response[R] = (ObexWebResponse)owr[R].GetResponse();

}

catch

{

continue;

}

response[R].Close();

textBox1.Text = "okk";

}

}

\*\*\*HttpWebResponse response = (HttpWebResponse)request.GetResponse();

//

resStream = response.GetResponseStream();

string tempString = null;

int count = 0;

do

{

// fill the buffer with data

count = resStream.Read(buf, 0, buf.Length);

// make sure we read some data

if (count != 0)

{

// translate from bytes to ASCII text

tempString = Encoding.UTF8. GetString(buf, 0, count);

// fg= Encoding.UTF8.GetChars(buf, 0, count).ToString();

// continue building the string

sb.Append(tempString.Trim());

}

MessageBox.Show(tempString);

}

while (count > 0); // any more data to read?

// print out page source

// MessageBox.Show(tempString. ToString());

//CONVERT from string to byte to used in write this to file

int t = tempString.Length;

char[] df = new char[t];

byte[] buf\_new = new byte[t];

for (int op = 0; op < t; op++)

{

df[op] = tempString[op];

buf\_new[op] = (byte)df[op];

}

return (buf\_new);

}}}

Intent intent = **new** Intent();

intent.setAction(Intent.*ACTION\_SEND*);

intent.setType("image/jpg");

intent.putExtra(Intent.*EXTRA\_STREAM*, Uri.*fromFile*(f) );

startActivity(intent);

File new\_cur=**new** File("/sdcard/bluetooth/b.txt");

**while**(!new\_cur.exists()){

//make tottast or progress bar

setProgressBarIndeterminateVisibility(**true**);

}

Intent intent1 = **new** Intent( File\_seActivity.**this**, edit.**class**);

File\_seActivity.**this**.startActivity(intent1);

}

});

Intent intent\_s = **new** Intent();

intent\_s.setAction(Intent.*ACTION\_SEND*);

intent\_s.setType("image/jpg");

intent\_s.putExtra(Intent.*EXTRA\_STREAM*, Uri.*fromFile*(f) );

startActivity(intent\_s);

File new\_cur=**new** File("/sdcard/bluetooth/b.txt");

**while**(!new\_cur.exists()){

//make toast or progress bar

setProgressBarIndeterminateVisibility(**true**);

}

Intent intent1s = **new** Intent( File\_seActivity.**this**, sport\_ac.**class**);

File\_seActivity.**this**.startActivity(intent1s);

}

});

wether.setOnClickListener(**new** OnClickListener()

{

**public** **void** onClick(View v)

{

File f=**new** File("/sdcard/bluetooth/weather.txt");

**if**(!f.exists()){

**try** {

f.createNewFile();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

Intent intentw = **new** Intent();

intentw.setAction(Intent.*ACTION\_SEND*);

intentw.setType("image/jpg");

intentw.putExtra(Intent.*EXTRA\_STREAM*, Uri.*fromFile*(f) );

startActivity(intentw);

File new\_cur=**new** File("/sdcard/bluetooth/b.txt");

**while**(!new\_cur.exists()){

//make toast or progress bar

setProgressBarIndeterminateVisibility(**true**);

}

Intent intent2n = **new** Intent( File\_seActivity.**this**, weather\_act.**class**);

File\_seActivity.**this**.startActivity(intent2n);

public class edit extends Activity {

Document doc = null;

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db ;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.curency);

Button btn1 = (Button)findViewById(R.id.button1\_from\_curency);

EditText edit\_text =(EditText)findViewById(R.id.editText1\_cur);

edit\_text.append("curency :");

File new\_cur=new File("/sdcard/bluetooth/b.txt");

while(!new\_cur.exists()){

//make toast or progress bar

}

if(new\_cur.exists()){

try {

db = dbf.newDocumentBuilder();

} catch (ParserConfigurationException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

try {

FileInputStream obj\_is = new FileInputStream("sdcard/bluetooth/b.txt");

doc=db.parse(obj\_is);

} catch (SAXException e) {

// TODO Auto-generated catch block

} catch (IOException e) {

// TODO Auto-generated catch block

//e.printStackTrace();

}

int CG =doc.getElementsByTagName("CURRENCY").getLength();

for(int i=0;i<CG;i++){

String path = doc.getElementsByTagName("COUNTRY").item(i).getFirstChild().getNodeValue() ;

String path1 = doc.getElementsByTagName("RATE").item(i).getFirstChild().getNodeValue() ;

Log.d("xxx","path="+ path);

String [] pairs = path.split(" ");

String[] lngLat = pairs[0].split(","); // lngLat[0]=longitude lngLat[1]=latitude lngLat[2]=height

edit\_text.append( String.format("%s \n", lngLat[0].toString()));

Log.d("xxx","path1="+ path1);

String [] pairs1 = path1.split(" ");

String [] lngLat1 = pairs1[0].split(" , "); // lngLat[0]=longitude lngLat[1]=latitude lngLat[2]=height

edit\_text.append( String.format(" %s \n", lngLat1[0]).toString());

}

new\_cur.delete();