

# Network Backup of Mobile Device Phonebook

**John Paul Costello**

A thesis submitted in part fulfilment of the degree of BSc. (Hons.) in  
Computer Science with the supervision of Dr. John Murphy and  
moderated by Dr. Joe Carthy.



School of Computer Science and Informatics

University College Dublin

30 March 2006

## **Abstract**

This report details research into implementing USSD functionality using J2ME on a mobile device. Reasons are discussed as to the difficulties in realising this task and a different direction for this project is explored. Problems are encountered with this direction and a substantial change in the projects direction is taken.

The alternative project concerns developing a GUI to allow the user to make a backup of the phonebook entries, stored on the mobile devices SIM card, to a network server. Using Web Services over GPRS the SIM card information can be transferred between the mobile device and the server. The GUI is developed in C# using the .NET CF libraries. The operating system used on the mobile device is Microsoft Windows Mobile Edition Pocket PC. Web Services use XML messages and SOAP to marshal data in a RPC fashion, allowing the mobile device to call methods on the server in a similar style to calling methods locally.

## **Acknowledgments**

The author of this report would like to thank O2 for providing a SIM card and an XDA Ili to be used for this project and Robert Burke of Microsoft for all his help in understanding the Visual Studio 2005 platform and its capabilities.

## Table of Contents

1	Introduction .....	5
2	Background Research .....	6
2.1	USSD .....	7
2.2	USSD Dialogue .....	7
2.3	USSD Applications.....	8
2.4	JSR 120 .....	9
2.5	JSR 253 .....	9
2.6	ExTAPI .....	10
3	Network Backup of Mobile Device Phonebook.....	11
3.1	Visual Studio 2005 .....	12
3.2	.NET Compact Framework .....	13
3.3	Web Services .....	14
3.4	SQL Server Express .....	15
4	System Architecture.....	15
4.1	Presentation Tier.....	16
4.2	Business Tier .....	16
4.3	Data Tier .....	17
5	Design and Implementation .....	17
5.1	Client Side.....	17
5.1.1	Uploading SIM Entries.....	17
5.1.2	Downloading SIM Entries .....	18
5.2	Server Side .....	18
5.2.1	Web Site Architecture.....	18
5.2.2	DatabaseWebService.aspx .....	19

5.2.3	DataLayer.cs.....	20
5.3	Extensions.....	20
6	Testing and Evaluation .....	22
7	Conclusion .....	24
8	Reference .....	24

# 1 Introduction

This report details the implementation of a network backup application for a mobile device running the Windows Mobile Pocket PC Edition operating system. This application was not the original project proposed but due to reasons discussed in this report the original project proposal had to be changed. The specification of the original project is given below.

*“USSD (Unstructured Supplementary Service Data) allows data to be sent between a user and a mobile operator's network via the signaling network. As the name suggests, the uses that this can be put to are unspecified. Typical uses of this data include call divert functionality, changing the amount of times the phone rings before switching to voice mail, etc. In some operator's networks this functionality is invoked by a very unintuitive set of key presses on the mobile device (eg 0#7#4#1234567). The objective of this project is to facilitate access to the same functions using a much more natural interface. Access to these functions can be obtained using the API defined in the JSR 120 specification.” [1]*

The mandatory specification of this project entailed sending a USSD message to the network using J2ME. To further this project a J2ME GUI was to be developed to provide access to multiple USSD based functions. An upgraded version of this project would allow the application to be updated with new USSD service codes and add these to the J2ME application dynamically and autonomously of user interaction.

During the research of this project, many articles were read and the Java support for USSD functionality was discovered to be currently unavailable on mobile devices. The impact of this discovery on the project implied a change in the direction for the project. A C++ solution on a Windows Mobile Pocket PC device using ExTAPI was researched but more difficulties were encountered. Further difficulties resulted in a substantial change to the projects direction.

With the projects deadline quickly becoming an issue a new project was ventured. This project involved developing an application on mobile device to allow the user to make a backup of the SIM cards phonebook entries. The specification of this project is as follows

*The objective of this project is to develop a mobile device application which can be used to upload SIM data from the mobile device to a network server. Specifically, this application will enable phonebook information to be uploaded and downloaded from a server, so that if the mobile device is lost, the user can retrieve their phonebook information. The project entails developing an application for the mobile device which can both upload and download the phonebook information, and a server application which can store the information in a file/database. [2]*

The mandatory field for this project is to *“Develop a mobile application which can upload and download phonebook information on user invocation. Develop a server-side system which can store this information in a file/database.”*

The discretionary field for this project is to *“Develop a variant of this application that can determine whether the phonebook information has changed significantly since the last upload and, if not, upload no information but update upload time.”*

To achieve the exceptional field in this project the following goal must be achieved *“Develop a variant of this application that can run in the background and can be used to store the phonebook on a network server periodically - every week, for example.”*

The objective of this project is to allow the user to backup the phonebook entries, on the SIM card, to a network server. GPRS was chosen as the communication medium between the mobile device and the server. An application was designed, using Visual Studio 2005, to provide the user with an interface to execute this operation. A server side system was also developed to receive the entries and store them in an SQL database. Using Web Services the phonebook information was transferred from the mobile device and to the server and vice versa. The extensions to this project included updating the network server periodically and automatically after a specified interval of time had elapsed and a further extension involved designing a mechanism to allow this application to run in the background and detect if any changes to the phonebook were made since the last time the phonebook was uploaded.

The following sections in this report outline the background research of the USSD project and the reasons it could not be realised. Section 3 introduces an alternative project involving Pocket PC, C#, O2's XDA Ili and Microsoft's Visual Studio 2005. The system architecture of the network backup application is detailed in section 4 with the design and implementation examined in section 5. Testing and evaluation of the project is discussed in section 6 and the projects conclusion is reported in section 7. All references used in the project are given in section 8.

## **2 Background Research**

Unstructured Supplementary Service Data refers to operations on mobile devices that require different service codes depending on the network provider. This differs from structured supplementary data in that the service codes defined for structured services are the same for all network providers i.e. the service code for call divert is the same for the O2 and Vodafone network providers. With USSD the service codes for the same operation are unique to each network provider. USSD is a session based service that uses the GSM signalling channels to send and receive USSD-packets. These packets can be sent between the mobile device and the network in a dialogue manner. This implies that USSD does not employ a store and forward methodology as incorporated by other services such as SMS, but instead uses a session-orientated technology. The GSM signalling channels used by USSD to send and receive packets are built directly into the GSM network and are available on all GSM mobile devices.

## 2.1 USSD

To invoke a USSD operation using a mobile device, the user must enter a string of key stroke characters and press 'send' on the mobile device. USSD is a mechanism through which non-standard services are implemented. As the name suggests, these operations are not structured meaning that a string sent to one network provider may have a different result if sent to a different network provider. For example, if an O2 subscriber wishes to check their prepaid balance they would dial \*100# on the device and press send. If the user is, for example, a Vodafone subscriber the same query would require the user to dial \*174#. Using the symbols \* at the start and # at the end specifies that the operation isn't a call but instead a USSD operation. The application retrieves the users balance and returns it as a USSD-packet to the device. The information in the USSD-packet is displayed on the screen of the mobile device in human readable text.

This is a useful mechanism to have on a mobile device. To invoke USSD functions is similar to making a phone call. The user dials a string of characters and then sends the string to their network provider. The disadvantage of USSD is that the user must know the specific 'service code' or string for each USSD operation to be performed, on the users network. This may be awkward and inflexible for the user, as a range of USSD operations are available and can change with different network providers. With the volume of USSD functionality available, users may not be aware of USSD applications that would be useful to them. Remembering many USSD strings can be difficult. Simple human error in dialling the series of characters could result in unwanted or unforeseen actions.

Creating a more natural interface for USSD applications would be of great benefit to the user. The user could choose from a list of available USSD applications and select an appropriate function. This would relieve the user of remembering specific USSD strings. There is also an implementation transparency employed in this approach. Most users are more interested in how to use an application as opposed to how the application actually works and the technology involved. The interface would hide the awkward implementation side of USSD applications from the user with a 'black box' model. This also has the benefit of hiding any changes to the USSD implementation from the user.

## 2.2 USSD Dialogue

The session-based nature of USSD implies that information needs to be exchanged between the mobile device and the network. This is done in a dialogue fashion i.e. one entity sends a request, another entity receives the request and replies with the appropriate response. There are two *phases* of USSD.

- USSD Phase 1
- USSD Phase 2

Phase 1 is mobile initiated allowing a mobile device to send a request to the network and receive a response. The network can either continue the dialogue by processing the operation and sending a request to the mobile device or the network can return a result to the mobile device and end the

dialogue. Both the device and the network can end the dialogue by issuing a release command. A network initiated request is not supported by this phase.

Phase 2 supports a dialogue to be established between the mobile device and network. In this phase the network can send the mobile device a request. The mobile device responds to this request by returning the result of the operation. The network can send two types of operations, a request or a notification. A request requires a response from the mobile device as opposed to a notification, which simply sends the mobile device some information. Multiple operations can be exchanged in sequence. Like mobile initiated dialogue, both the device and the network can end the session by issuing a release command. GSM mobile devices currently use USSD phase 2.

Supplementary Services have many applications on mobile devices. Its functionality ranges from diverting incoming calls to an alternative number (structured) to adding credit to another mobile users account (unstructured). USSD strings typically start with an \* symbol and end with a # symbol. Between these symbols are numbers from 100 – 149.

## 2.3 USSD Applications

The USSD applications that were considered in the USSD project for the user interface were:

- \*100# - prepaid balance query
- \*101# - post paid balance query
- \*103#086xxxxxxx# - call me
- \*104#euros# - mobile top up
- \*108\*086xxxxxxx# - credit me
- \*123#086xxxxxxx# - call back when roaming

The above USSD service codes are specific to the O2 network. Mobile phone number can be added to the USSD format with either a \* or a # separator. The entire USSD string is ended by the # symbol. Prepaid and post paid services simply return the mobile devices account balance upon sending the request. The ‘Call Me’ service provides the user the ability to send a text message to another mobile device (using the same network operator) free of charge.

The ‘Mobile Top Up’ has two formats. The user can first send the request by dialling the service code (\*104#) and pressing send. The network receives the request and sends a request back to the mobile device for the amount the user wishes to top up their account by. The user sends the amount back to the network. The network again queries the user for a pre defined access code. The user sends the access code and the network increases the users account balance by the stated amount. The other format for this service is to allow for all the information to be sent in one string i.e. \*104#30#56448#. This example of USSD shows that, once a connection is established between a mobile device and the network, the session can remain open for multiple exchanges to take place.

‘Credit Me’ provides a user the ability to send a message to another mobile device and requests the user of that device to add credit to the senders’ mobile account. When roaming, the user can

use the 'Call Back when Roaming' USSD function, to send a text message to a mobile device informing the user of that mobile device to ring the roaming user. For this operation the phone number of the device the text message is to be sent to must be specified in the USSD string.

## 2.4 JSR 120

To implement USSD functionality the supporting Java methods are needed. The JSR 120 [3] specification detailed that USSD support for J2ME would be addressed. The USSD definitions that were claimed to be addressed in this JSR are central to the Java implementation of the USSD functionality in the USSD project.

This Java specification request details a wireless messaging API (WMA). The specification defines classes for `javax.wireless.messaging`. The package works with message objects and supplies interfaces for the creating, sending and receiving messages both binary and text.

Two types of message can be created by the `Message` interface: text and binary. Methods are defined for each in relation to message content, timestamps, setting address of recipient and getting address of the sender. A `MessageListener` interface provides a mechanism for detecting incoming messages. An extension to the `javax.microedition.io` package has been added in the form of a `Connector` class. This class creates new connection objects. Instances of this `Connection` class can be used to open a connection through which a message can be sent.

JSR 120 defines interfaces and methods that are concerned with sending and receiving text and binary messages only. Interfaces and methods dealing with sending and receiving USSD messages are not addressed, as claimed, in the specification detail and therefore the Java methods for USSD operations are not defined in this JSR. The specification does not identify any mechanisms for USSD support. Much research was done into how this JSR supported USSD. Many forums, both Java and Nokia, and Internet sites were queried for possible explanations of the JSRs support for USSD, with no positive result. Finally the editor of JSR 120 was contacted. The editor confirmed that JSR 120 did not provide any support for USSD functionality and advised that JSR 253 defined J2ME support for USSD functionality.

The initial suggestion in the projects specification that JSR 120 supported USSD was very confusing and as a result much research was done trying to understand the connection between the JSR and USSD. The high level specification of the JSR implied that support for USSD in J2ME would be addressed but from reading the specification the support is not addressed or defined.

## 2.5 JSR 253

A JSR designed to control voice calls and to use network services that are suitable for J2ME is proposed in JSR 253 [4]. This JSR outlines a Java interface called `UnstructuredService`. The interface contains Java methods for sending service requests to and receiving and responses from the network. An application can register an `UnstructuredServiceListener` interface to listen for incoming messages that are associated with the unstructured service that registers this listener.

JSR 253 supports the interfaces and Java methods that enable USSD operations to be sent from and received by J2ME. JSR 253 has been ratified as of November 2005 but will not be available publicly on mobile devices in time to be used for this project. Devices supporting the

specification may not be available on the market for approximately 6 months after the specification has been ratified.

USSD functionality in Java was not available on mobile devices and was not available in time to be used in the USSD project. Due to this unforeseen circumstance a change in direction for the project was needed.

## 2.6 ExTAPI

An alternative approach that was considered was to use a mobile device running a more sophisticated Operating System. The Microsoft libraries document support for USSD operations using an API. This documentation is found in the C++ Extended Telephony API (ExTAPI) [5]. This API includes such functions as `lineSendUSSD` and `lineGetUSSD` for Pocket PC. These functions support sending a USSD message to the network provider and receiving a USSD response from the network provider respectively. The ExTAPI defines a listener that sends a notification to the device upon receiving a USSD message from the service provider.

O2s XDA Iii (see Figure 1) runs Windows Mobile 2003 Second Edition for Pocket PC phone edition. Windows Visual Studio 2003 was selected as the development platform to develop the application due to its support for mobile device applications. The SDK for Mobile Pocket PCs was downloaded and installed for Visual Studio 2003.



Figure 1. O2 XDA Iii [6]

Further searching of the internet provided little documentation that supported programming applications using the ExtTAPI in C++ on a Pocket PC device. Trying to include the ExtTAPI into a Visual Studio mobile device application also proved difficult and was not be realised. Developing applications on a native Windows CE .NET environment can be difficult. Issues such as releasing memory after the memory is no longer needed are the responsibility of the programmer and applications can be prone to memory leaks. The APIs used by native .NET development are low level and can be difficult to learn and program with. Native Windows CE .NET programming languages are procedural and do not support features such as inheritance and polymorphism that are supported in object orientated languages.

This direction of the project was reconsidered as many aspects of the implementation remained unknown and a definite end product could not be guaranteed to be achieved before the deadline of the project. As the time remaining to complete the project was becoming an issue, a new project was ventured.

### 3 Network Backup of Mobile Device Phonebook

Due to the reasons stated above, a substantial change in the projects direction was required. The new project involves mobile networking, developing a GUI to aid the user with the application and passing data between a mobile device and a network server. The aim of the new project is to develop a network backup of the phonebook on the SIM card of a mobile device. This project has a practical application for mobile devices, as losing ones SIM card or mobile device can result in losing all contact information that was stored on the SIM card. This can be frustrating for the user as recovering all contact information and inputting the information into a new device can be time consuming. A current solution to this problem is to buy a 'SIM CARD Backup Device' similar to the one shown in Figure 2 below.



Figure 2. SIM CARD Backup Device [7]

This project is designed for a mobile device running the Windows Mobile Pocket PC operating system. The XDA Ili meets this requirement. Due to a Microsoft product being used, the Microsoft C sharp (C#) language was chosen to implement the application. A defining factor in this language choice was that C# libraries are already installed on the device making 'deployment' of the application to the device more convenient. C# is an object orientated language which supports the features common to object orientated languages including inheritance, polymorphism, hash tables etc. Garbage collection is used by C# meaning that developers of C# programs do not have to concern themselves with memory management and memory leaks.

Storing the phonebook information on a server would mean that the users' involvement in restoring their phonebook, if lost, would be reduced to simply pressing a button on the application. Similarly for the user to retrieve all their phonebook contacts all that would be required the press of a button and the application would download all the phonebook information to the mobile device.

### 3.1 Visual Studio 2005

The development environment for this project was chosen to be Microsoft Visual Studio 2005. This version of Visual Studio was chosen due to its support for Pocket PC 2003 applications, Web Services and SQL Server Express. These components were each incorporated into the project. The Pocket PC application support gives the programmer an environment and a set of libraries designed specifically for mobile devices. These libraries are contained in the Microsoft .NET Compact Framework (CF). The .NET CF was designed as a subset of the .NET Framework libraries used by all ASP.NET applications. These .NET CF libraries can be used to develop applications for mobile devices.

An emulator is also provided (see Figure 3) which gives the developer the opportunity to test program code at an early stage in development without requiring the code to be deployed to the actual mobile device. This is a very useful feature as deploying code to the mobile device requires a significant amount of time, compared to compiling and running code locally on a computer. Much time can be spent deploying code to the device to test small tweaks to code. Debugging on a mobile device can also be very problematic as applications can crash with little information returned as to the reason for the crash. This can make detecting errors very difficult in large segments of deployed code. Using an emulator can save the developer time in both deploying and debugging. The emulator can also return more detailed and helpful error messages if bugs are encountered.



Figure 3. Visual Studio Pocket PC Emulator [8]

### 3.2 .NET Compact Framework

Resources that are available on a mobile device are limited compared to a desktop PC. For this reason the .NET Compact Framework was specifically designed by Microsoft to aid developers of mobile applications. The functionality that is supported by the .NET Compact Framework is limited compared to the full .NET Framework but the essential and most commonly used libraries are supported. Areas of functionality that are supported on version 2.0 of the .NET Compact Framework include user interface programming, display and layout management, keyboard management, XML, communication (including Web Services) and security [13].

The Common Language Runtime (CLR) is used by all .NET applications and is the runtime environment for the .NET Framework. It handles code execution, resource and memory management of the running applications. The total size the .NET Compact framework libraries consume on a mobile device is 1.5MB [14], which is relatively small for the volume of functionality that is supported.

The structure of the layered .NET Framework is shown in Figure 4 below. If a language is supported by the Common Language Specification (CLS) then that language can access the .NET Framework class libraries via the CLS.

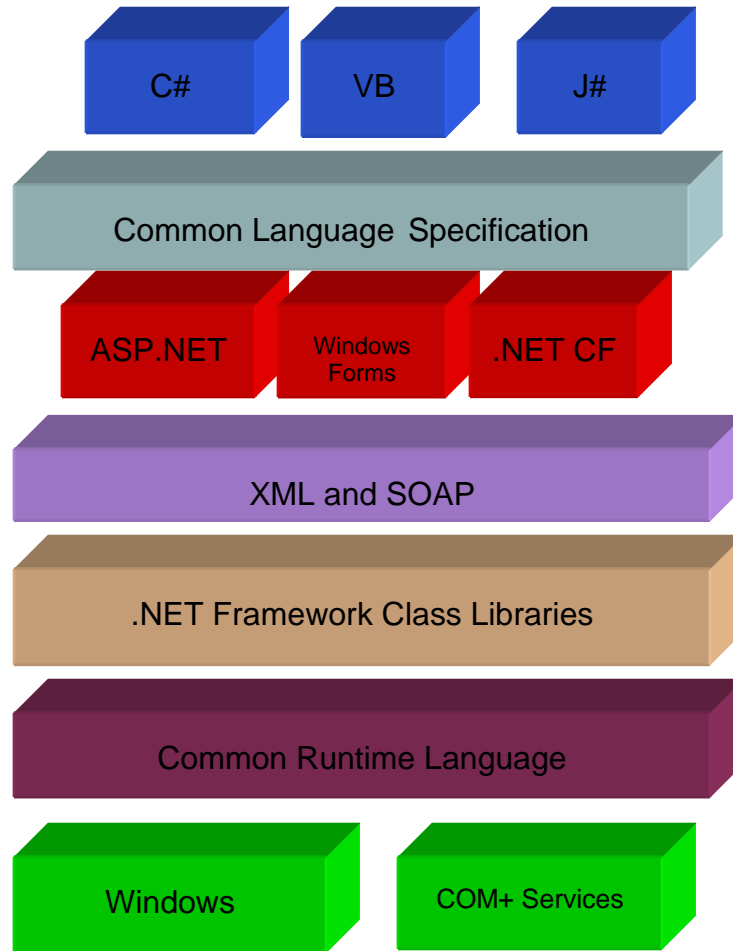


Figure 4. The Layered Architecture of the .NET Framework

### 3.3 Web Services

An important feature of Visual Studio 2005 for this project is the support for Web Services. These Web Services allow the programmer to make connections to the internet using XML (Extensible Markup Language) messages and SOAP (Simple Object Access Protocol) [9]. SOAP allows for methods, written in C#, to be invoked over the internet. SOAP employs a Remote Procedure Call (RPC) style of implementation. The client or user invokes a call on a server. The server receives the call, invokes the desired method and responds to the client with the appropriate result. These calls can be *marshalled* into XML messages. The client marshals the data and sends it to the server. The server *unmarshalls* the data and determines the result. The server then marshals the result and sends the marshalled result back to the client for it to be unmarshalled and used. A mobile device application can call methods, running on a server, using SOAP and XML messages in a similar way to the application calling the methods locally in the application on the device. The Web Services interface on the server is a .asmx file (i.e.

../WebService.aspx). The mobile device can invoke a method by communicating directly with this web page.

### 3.4 SQL Server Express

An SQL database can be used to store the phonebook entries on the server. SQL Server Express [10] allows the developer to program ‘Stored Procedures’ for the database. These Stored Procedures are SQL statements that can be executed on the database. Stored Procedures can be called directly from C# code or can be executed, in Visual Studio, as independent SQL statements on the database. Arguments passed or returned by the Stored Procedures are either passed by the calling C# method or returned to the calling C# method.

An alternative approach to using Stored Procedures to execute SQL statements on the database would be to embed SQL statements in C# code. This can be an awkward approach as C# code must be designed to test the SQL statement returned the correct result. Stored Procedures separate SQL from C# code. To execute a Stored Procedure on a database from C# code the calling C# method passes the name of the Stored Procedure as an argument.

This approach was chosen over implementing SQL queries in the C# code because Stored Procedures are stand alone SQL statements and can be executed without embedding SQL code in C# code. For this reason Stored Procedures can be tested on the database independently of the web interface or C# code. Information in the database can be directly manipulated by designing these Stored Procedures to execute the appropriate commands in SQL.

## 4 System Architecture

This project employs a 3 tier architecture structure. These 3 tiers consists of a presentation tier (on the mobile device), a business tier (the web site) and a data tier (database on server). This 3 tier architecture is graphically represented in Figure 5. The presentation tier is the GUI the user interacts with. From this GUI the user is able to manually upload and download the phonebook entries from the SIM card of the mobile device. The business tier provides a communication point between the mobile device and the database. The business tier hosts a website that the user can interact with and view details about the phonebook entries that are uploaded and also delete their entries stored in the database. This tier also hosts an interface web page that the mobile device can communicate directly with. This interface web site contains C# *Web Methods* that the application on the mobile device can invoke. The data tier consists of the SQL Server Express database. This database is written in SQL and stores name, phone number pairs.

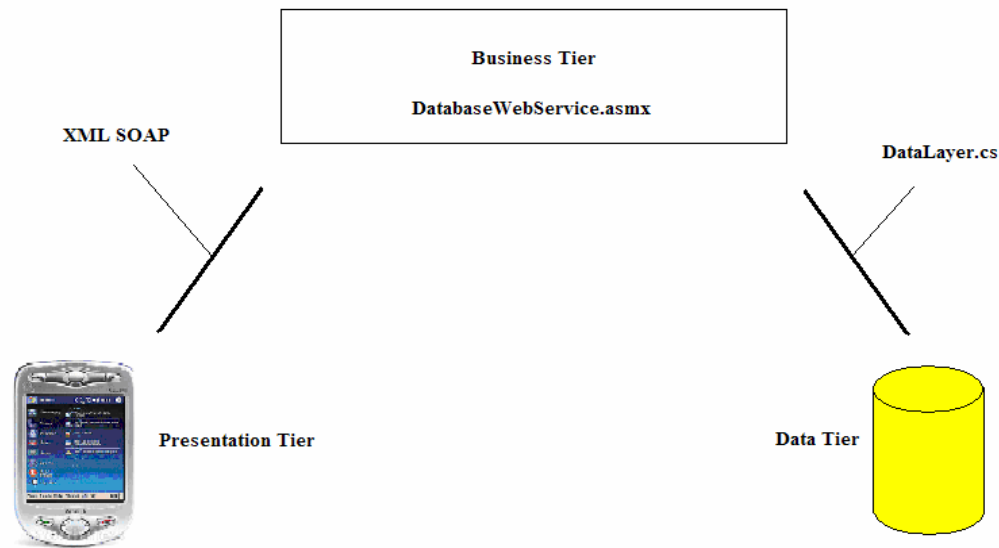


Figure 5. 3 Tier Architecture

## 4.1 Presentation Tier

The Presentation Tier is the user interface displayed on the mobile device. Buttons, displayed on the interface, allow the user to manually upload and download the SIM cards phonebook. The application connects to the server via GPRS (General Packet Radio Service). Data is passed via GPRS to the server using XML messages and SOAP. The Presentation Tier hides all implementation details from the user.

## 4.2 Business Tier

The Business Tier is hosted by Microsoft's IIS (Internet Information Services) Server. IIS comes as standard with Windows Server 2003 and as an optional add-on with Windows XP Professional Edition. IIS incorporates Web Services and facilitates SQL Server Express. Web Services are used in this project by making web methods on the server available to be invoked by the application on the mobile device. The web methods that can be called by the mobile device are displayed on a web page with a .asmx extension (i.e. WebService.asmx) on the server. This .asmx web page displays web methods that can be called by the application on the mobile device to implement an operation (i.e. DownloadPhonebook()) on the .asmx web page is a web method that can be called by the mobile device).

The C# implementation of these web methods is stored in a C# file that is 'hidden' behind the .asmx web interface (see figure 6). This hidden C# class stores the implementation code for the web methods available on the .asmx web page to the mobile device.

For the business tier to communicate with the database in the Data tier a C# connecting file is needed. This connecting file will bridge the gap between the web methods, written in C#, on the web site and the database, written in SQL, stored in the Data tier.

### 4.3 Data Tier

The Data Tier, which comprises of the database, is stored on the server. The primary function of this database is to store and return the name, number pairs upon request. SQL Server Express databases can hold up to 4 GBs of data. This will be sufficient as most modern SIM cards can only store a maximum of 250 name, number pairs. The connecting C# class can invoke Stored Procedures on the database and receive data returned, if any, from the database. The connecting class specifies the appropriate Stored Procedure to be executed on the database and also handles any arguments that are passed or returned.

The 3 tier architecture implemented in this project is explained above. The implementation of each tier is explained in the next section.

## 5 Design and Implementation

There are 3 tiers in this project comprising of presentation, business and data tiers. These three main components are discussed in the System Architecture. The design and implementation of these components is encapsulated in the client side and the server side aspects. These aspects are discussed in this section.

### 5.1 Client Side

The client side involves the presentation of the GUI to the user and how the user should interact with the GUI. All implementation aspects of the application are hidden from the user. The application allows the user to select either an upload or a download function from buttons displayed on the user interface. A label is also included to inform the user, in human readable text, the status of the application and to display messages upon the completion of operations. The label displays the date and time of the last completed operation the application performed.

#### 5.1.1 Uploading SIM Entries

To access the SIM card information the OpenNETCF [11] libraries are used. A SIM object is created and using this SIM object the phone book entries are accessed. Using the `OpenNETCF.Phone.Sim.PhonebookEntry()` class, individual phonebook entries can be read. To read all phonebook entries a new `OpenNETCF.Phone.Sim.PhonebookEntry()` object is created for each entry. Due to the indexing of entries on the SIM card the capacity of the SIM card must be checked for entries. This is an inefficient approach to the search for occupied entries but it is the only reliable approach. Simply using the number of entries that are occupied on the SIM card will not guarantee that all the occupied entries will be retrieved. If there are, for example, 5 occupied entries on the SIM card then the count (the number of occupied entries on the SIM card) returned is 4 (0-4). This does not have any relation to the index at which these 5 entries are located. An array of phonebook entries is used to store all the occupied entries on the SIM card. The length of the array is equal to the number of occupied entries on the SIM card i.e. if there are

5 occupied entries on the SIM card then the arrays length will be 5. This array is marshalled into an XML message and sent to the server using a Web Service.

### 5.1.2 Downloading SIM Entries

Downloading the SIM entries from the server is done using a Web Service. The application on the mobile device requests the server to retrieve the list of phonebook entries from the database. This list is returned to the mobile device as an array. The OpenNETCF libraries do not appear to support writing entries to the SIM card. For this reason the retrieved entries are written to a text file that is stored in the applications directory. Each entry in the array is individually written to the file, called Phonebook.txt, using a `StreamWriter`. The Phonebook.txt file is stored in the same directory as the application. The entries are successfully retrieved and downloaded from the server to the mobile device but cannot be stored directly onto the SIM card via the application. To write the downloaded phonebook entries to the SIM card requires more research.

## 5.2 Server Side

Microsoft's Windows Server 2003 was setup and installed on a machine to host the web site needed for this project. Visual Studio 2005 was also installed on this machine so that the C# and SQL libraries were installed correctly. The URL for this machine is <http://193.1.132.40/pjcostello/Default.aspx>. From this web site the user can retrieve and delete their phonebook entries from the database.

### 5.2.1 Web Site Architecture

The web site architecture is designed for two types of user: a mobile device and a web user. For this reason two separate and different interfaces exist: *DatabaseWebService.asmx* for the mobile device and *Default.aspx* for the web user. Both interfaces have a C# class 'behind' them to implement the functionality of their respectful interfaces.

The C# implementation of *DatabaseWebService.asmx* is contained in the class *DatabaseWebService.cs*. The implementation of *Default.aspx* is contained in the *Default.aspx.cs* class.

To connect to the SQL database the C# implementation classes require a class that can communicate to the database. This class is called *DataLayer.cs* and contains C# code that can pass and receive arguments to and from the database.

The *DataLayer.cs* class can execute Stored Procedures on the database. Stored Procedures are used instead of embedding SQL statements in C# code for reasons explained in section 3.2.3. The Stored Procedures can directly manipulate information stored in the database.

A diagram representing the web architecture is given in Figure 6. This figure highlights the relationship between the interface classes and the implementation classes and also how the implementation classes relate to the database.

The web user interface (*Default.aspx*) is a simple interface that allows the user to view their uploaded phonebook entries and also delete all their uploaded phonebook entries.

The mobile users interface and the *DataLayer.cs* class are more complex than the web users' interface and are explained in depth in sections 5.2.2 and 5.2.3.

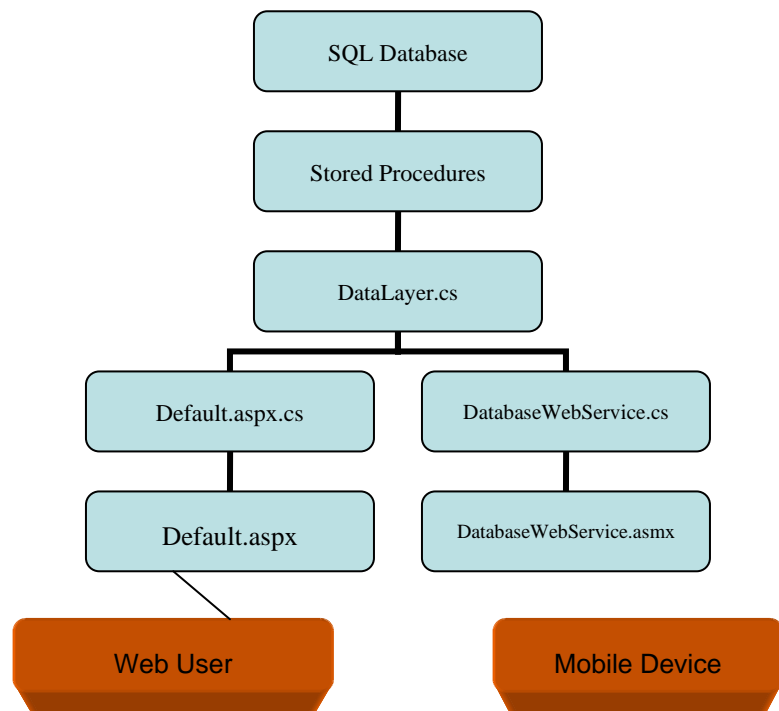


Figure 6. Web Architecture

### 5.2.2 DatabaseWebService.asmx

The mobile device can call Web Methods using the *DatabaseWebServices.asmx* interface hosted by IIS. This web page contains the header declarations for the C# methods that are available to the mobile device to invoke. The headers contained on this web page include:

- `DownloadPhonebook()`
- `UploadPhonebook(PhonebookEntry[] allSimEntries)`
- `DeleteAllPhonebookEntries()`

The `DownloadPhonebook()` method returns a list of the phonebook entries stored in the database to the mobile device.

`UploadPhonebook(PhonebookEntry[] allSimEntries)` takes as an argument an array of phonebook entries that contain all the SIM card entries from the mobile device.

`DeleteAllPhonebookEntries()` removes all the phonebook entries in the database.

The implementation of these methods is located in *DatabaseWebServices.cs*. The Web Method attribute allows the method to be called externally i.e. from a mobile device. This Web Method attribute is associated with every method in *DatabaseWebService.cs* that is available to be called by the application on the mobile device. Methods that are not associated with a Web Method attribute in the *DatabaseWebService.cs* class are not available to be called by the application on the mobile device. For internal methods the Web Method attribute is not used and therefore that method cannot be called externally i.e. it can only be used by other methods in the *DatabaseWebService.cs* class.

### 5.2.3 DataLayer.cs

For these Web Methods to communicate with the SQL database a 'link' class is needed. The *DataLayer.cs* class is used to link the Web Methods, in *DatabaseWebService.cs*, to the SQL database. The *DataLayer.cs* class contains the following methods:

- `DownloadPhonebook()`
- `InsertPhonebookEntry(PhonebookEntry entry)`
- `DeleteAllPhonebookEntries()`
- `UploadPhonebook(PhonebookEntry[] simEntries)`

`DownloadPhonebook()` reads all the entries in the database and adds each entry to a List of `PhonebookEntry`. The List is returned after all entries in the database have been read and added.

`InsertPhonebookEntry(PhonebookEntry entry)` takes a phonebook entry as argument and inserts that entry into the database.

`DeleteAllPhonebookEntries()` deletes all the entries in the database.

`UploadPhonebook(PhonebookEntry[] simEntries)` iterates through the array of phonebook entries and stores each entry individually in the database using the `InsertPhonebookEntry(PhonebookEntry entry)` method.

To execute operations on the SQL database the Stored Procedures are used. The implementations of the methods in the *DataLayer.cs* class are able to call the Stored Procedures of the database. The calling method in *DataLayer.cs* invokes the Stored Procedure by passing the name of Stored Procedure as a `SqlCommand` argument with the connection string for the database. The calling

method can pass arguments that maybe required by the Stored Procedure and also handle any values returned after the execution of the Stored Procedure.

### 5.3 Extensions

Some extensions were added to the basic design of the application to add extra and helpful functionality. These extensions included an automatic update that can upload the phonebook after a specific interval of time has elapsed and also mechanism in the application was developed to determine whether the phonebook on the SIM card had been modified since a previous upload. If the phonebook had not been modified since the last upload then an upload would not be executed. A smart check was also designed for manual uploads to inform the user if the phonebook has not been modified since the last upload. A ReadMe.html file was also written and is stored in applications directory and also in the Help option on the Start menu of the mobile device.

Using the Timer class in C# the application can be scheduled to upload the phonebook after a specified interval of time has passed. The user, using Radio Buttons, on the GUI can set the time interval. The time interval can be set by the user to check the phonebook daily, weekly and bi-weekly for changes. Once the time has elapsed the phonebook is automatically updated. That user can also disable automatic updates. The application is set to run continuously in the background once activated to allow for automatic updates without any intervention from the user. To stop the application the user must explicitly terminated it. If the application is terminated automatic updates will not be active.

OpenNETCF provides a `getHashCode()` method that can be called on the name and number fields of each phonebook entry. Hash codes are calculated for each field in every entry and summed together to result in one hash code. The summation of all hash codes is needed to ensure that any change in the phonebook will be detected. Although there is a possibility of the same hash code been generated after a modification to the phonebook has happened, it is not likely to occur as the hash code generated is an 11 digit number. The possibility of getting two 11 digit number generated after a change to the phonebook is very unlikely. These hash codes can be summed and stored as an indicator to determine if any changes to the phonebook have taken place. The `getCurrentHashCode()` method is used to calculate the current value of the hash code. This value can then be compared against the previous hash code value that is stored in a global variable. If the two values differ, then the phonebook is uploaded else the phonebook has not changed and is not uploaded. This is a useful extension as the application can determine if the phonebook needs to be uploaded instead of 'blindly' uploading the entire phonebook when no modifications have occurred since the last upload. The primary purpose of this mechanism is to save the user money by not uploading the phonebook unnecessarily. It would not be efficient or desirable for the application to blindly upload the phonebook contents after the specified interval of time has passed but instead only if the contents of the phonebook have been modified since the last upload.

For manual uploads a smart check is used. This smart check is a Boolean function, `isChanged()`, and is called when the user wishes to manually upload the phonebook. The smart check returns true if the phonebook has changed since the last upload and allows the user to upload the phonebook else it returns false. If false is returned (the phonebook has not changed since the last upload) then the user is informed that the phonebook has not been modified since the last upload and queries the user to continue with the upload. The purpose of this smart check feature is to

save the user both time and money on manual uploads by not uploading the phonebook unnecessarily. If the phonebook has not been modified since the last upload there is no reason to upload the same phonebook contents again. The user is given the option to upload the phonebook even if it has not been modified.

A readme file for the application was written in html to inform the user about what the application does, how it works and the features of the application. This ReadMe.html can be accessed from the applications directory or by using the Help option in the Start menu and selecting 'SimBackup'.

## 6 Testing and Evaluation

The application was tested at various stages during development. Tests included:

- checking connectivity and communication between the mobile device and the server
- ensuring all phonebook entries were retrieved for the both upload and download functionality
- testing the Stored Procedures performed the correct operations on the database
- monitoring the hash codes for differences and ensuring that any change to the phonebook resulted in a change to the hash value and the modified phonebook was uploaded
- different values were used for the timers interval value to ensure updates were scheduled at the correct times
- uploading and downloading the phonebook with the maximum number of entries stored in the phonebook i.e. full capacity
- measuring the cost and time required to upload and download the phonebook at full capacity

Another extension that was considered and researched was to allow for multiple devices to upload their phonebook entries on the same server and manage each mobile devices phonebook individually. The database would need to be restructured from storing name, number pairs to storing number, character array pairs. The number would be the SIM cards MSISDN (Mobile Subscriber Integrated Services Digital Network) [12] number and the character array would contain all the phonebook information. The devices MSISDN would be the primary key for each entry in the database and would be used as the look up for retrieving each devices phonebook.

Designing the web site to manage SIM phonebooks from multiple users would be of great benefit to this application. The phonebooks could be stored in the database and retrieved using the mobile

devices MSISDN. If this extension to the database was realised, the web site could also be extended to allow users to register accounts for uploading their phonebook contents. The web interface could be modified to allow these registered users to logon to their accounts using their MSISDN and a personal identification number (PIN). From the users account a range of functionality could be implemented including viewing, editing and downloading entries and other media (ring tones, pictures, etc). More research would be required to realise this feature.

To retrieve the MSISDN number from the SIM card proved to be very difficult and was not achieved. OpenNETCF provides a method in its `OpenNETCF.Phone.Sms` class that retrieves the devices phone number. It was discovered, during testing, of the `Sms` class, that the device phone number could not be retrieved. A `Win32 Exception` is thrown with the message "Error trying to retrieve phone number". From searching forums on the internet it was mentioned that some network providers do not make the MSISDN available to all applications, this might explain why the phone number could not be retrieved.

The objective of this project was to develop an application to upload and download the SIM cards phonebook entries from the mobile device to a network server. Using the application all phonebook entries on the SIM card are retrieved and uploaded to the network server. The uploaded phonebook is stored in a database on the server. This functionality was tested using a SIM card with 250 phonebook entries (the SIM cards capacity). The time required by the application to successfully retrieve all phonebook entries, upload the entries to the server and store them in the database was timed to be approximately 24 seconds. The upload was repeated over 15 times and the average time for uploads was 24 seconds.

Reasons for the upload taking this amount of time include searching the entire phonebook for occupied entries and the slow transfer rate of data via GPRS. The algorithm for searching the SIM card for occupied phonebook entries must search the capacity of the SIM cards phonebook as opposed to the number of occupied entries to ensure all occupied entries are retrieved. This is a necessary approach to this problem, as the indexing of entries on the SIM card is not guaranteed to be in a sequential order. If the phonebook count (number of occupied entries) was used then entries with an index higher than the count would not be retrieved. To guarantee all occupied entries are retrieved the capacity of the SIM cards phonebook is checked for occupied entries.

Transferring data via GPRS is a slow medium compared to transferring data over a LAN. The mobile device can be connected to a computer using USB. If that computer is connected to the Internet, the connection settings of the mobile device can be configured by the user to connect to the Internet via the USB connection to the computer. In this instance the transfer rate is increased as GPRS is not used and also there is no charge to the user of the mobile device for the upload/download.

The recorded cost of uploading 250 phonebook entries using GPRS on the O2 network was recorded to be €0.41 for 56KB of data. These figures are taken from the O2 bill of the SIM card used in the mobile device for the upload. Costs of uploading and downloading the phonebook can be affected by the network providers GPRS rates for that client. The time and cost of this upload function could be decreased if the phonebook data was compressed before sending the information to the server via GPRS. To compress the data before sending it to the server would require more research but should be possible as compressing data on a mobile device would be common. Compressing data before sending it to a server or another mobile device would save battery power and cost. It is possible that compression libraries are available for Pocket PCs. If

possible this compression extension to the application would have practical benefits for the user in respect to time and money.

The application can successfully download all the phonebook entries stored on the server. To download 250 phonebook entries, from the server to the mobile device, took approximately 9 seconds and cost €0.46 for 63.0KB. These figures are taken from the O2 bill of the SIM card used in the mobile device for the upload. Saving the downloaded phonebook entries to the SIM card was not achieved in time for the deadline of this project and requires further research. It is unclear if the OpenNETCF libraries support this functionality and many forums were search for a solution but with no positive result. The downloaded phonebook is stored in text file located in the applications directory.

## 7 Conclusion

Research into implementing USSD functionality using J2ME proved to be unrealistic for the deadline of this project. The project was altered slightly to implement USSD functionality on a more sophisticated operating system and using the C++ programming language. Further difficulties were encountered with this approach. An alternative project, using mobile networking, was considered and a specification for that project was designed. This new project was concerned with uploading and downloading the phonebook entries on a SIM card. The application can successfully execute these tasks. Extensions that were added to the application include an automatic upload timer that uploads the phonebook after an interval of time has passed, a mechanism to determine if any changes to the phonebook have occurred since the last upload of the phonebook, a smart check for manual uploads and a ReadMe document was written to aid the user in using the application.

The application and web interface could be extended to include more functionality if the MSISDN was retrieved from the device. Retrieving the MSISDN would allow the web interface a means to possibly locate and send information to the mobile device. The database could be extended to manage and store various types of data. Having the ability to upload and retrieve data stored on a server is a very useful application for a mobile device. The limited storage space on a mobile device becomes less of an issue with this application. The technology used in the implementation to this project provides a foretaste into the potential that is possible.

## 8 Reference

- [1] [http://www.cs.ucd.ie/courses/undergrad/bsc/FourthYear/projects/Project\\_40/default.htm](http://www.cs.ucd.ie/courses/undergrad/bsc/FourthYear/projects/Project_40/default.htm)

USSD Project Specification.

- [2] [http://www.cs.ucd.ie/courses/undergrad/bsc/FourthYear/projects/Project\\_20/default.htm](http://www.cs.ucd.ie/courses/undergrad/bsc/FourthYear/projects/Project_20/default.htm)

Network backup Project Specification.

[3] <http://www.jcp.org/en/jsr/detail?id=120>

Java Community Process, JSR 120.

[4] <http://jcp.org/aboutJava/communityprocess/final/jsr253/index.html>

Java Community Process, JSR 253.

[5] [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/apippc/html/ppc\\_extapi\\_functions.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/apippc/html/ppc_extapi_functions.asp)

MSDN, ExTAPI functions.

[6] <http://www.dynamoo.com/moobiles/o2-xda-iii.shtml>

XDA image.

[7] <http://www.solware.co.uk/mobile-phone-sim-card-backup/index.shtml>

SIM card backup device image.

[8] <http://msdn.microsoft.com/library/en-us/dnppcgen/html/nativedevicedev15.gif>

Visual Studio 2005 Emulator image.

[9] <http://xml.coverpages.org/soap.html>

XML Technology Report, SOAP.

[10] <http://www.microsoft.com/sql/editions/express/default.mspx>

Microsoft SQL Server Express Edition.

[11] <http://www.opennetcf.org/CategoryView.aspx?category=Home>

OpenNETCF.

[12] <http://en.wikipedia.org/wiki/MSISDN>

Wikipedia, Topic: MSISDN.

[13]

[http://msdn.microsoft.com/netframework/programming/netcf/default.aspx?pull=/library/en-us/dnnetcomp/html/whats\\_new\\_netcf2.asp](http://msdn.microsoft.com/netframework/programming/netcf/default.aspx?pull=/library/en-us/dnnetcomp/html/whats_new_netcf2.asp)

MSDN, .NET Compact Framework 2.0.

[14]

[http://msdn.microsoft.com/netframework/programming/netcf/gettingstarted/default.aspx?pull=/library/en-us/dnnetcomp/html/net\\_vs\\_netcf.asp](http://msdn.microsoft.com/netframework/programming/netcf/gettingstarted/default.aspx?pull=/library/en-us/dnnetcomp/html/net_vs_netcf.asp)

MSDN, Fundamentals of .NET CF.