

Final Year Project Report

RSS Reader for a Mobile Device

Kevin Hall

A thesis submitted in part fulfilment of the degree of

BA/BSc (hons) in Computer Science

Supervisor: Dr. John Murphy

Moderator: Dr. Chris Bleakley



UCD School of Computer Science and Informatics
College of Engineering Mathematical and Physical Sciences
University College Dublin

March 23, 2006

Table of Contents

Abstract	2
1 Introduction	4
1.1 Project Specification	4
1.2 RSS	4
1.3 J2ME	5
1.4 Report Summary	5
2 Background Reading	6
2.1 RSS	6
2.2 Choosing the Development Platform	8
2.3 J2ME	10
3 Design	14
3.1 Conclusions derived from background reading	14
3.2 Proof of Concepts	15
3.3 Goals and Milestones	15
3.4 Program Structure; Classes and key Methods	15
4 Implementation	17
4.1 Development Environment	17
4.2 RSS Reader; detailed description	17
4.3 Verification	21
4.4 Packaging & Deployment	21
5 Testing	23
5.1 Functional Testing	23
6 Conclusion	24
6.1 Project Evaluation	24
6.2 Description specifics and implementation	25
6.3 Future Work	25

Abstract

RSS is an XML file format used for convenient syndication of information and data. This report details the research made into RSS and J2ME, the development platform used. Also discussed is how the project was designed and a detailed description of the implementation which includes class descriptions and operations, steps taken in packaging the application, deployment of the package and how the application was tested. The report concludes with an evaluation of the project specification and whether the goals were met.

Acknowledgments

Dr. John Murphy and Dr. Seán Murphy of the School of Computer Science and Informatics and Performance Engineering Laboratory UCD provided invaluable guidance and feedback during the development of the project.

This project uses the kXML2 Open Source XML parser[1] in order to parse the RSS feeds for display. This API was an indispensable resource to this project.

The test device, a Nokia 6630, a series 60 mobile phone was provided by O2 Ireland and proved to be very useful.

Chapter 1: Introduction

1.1 Project Specification

1.1.1 Description

This project will develop a small application in which a user registers a number of RSS feeds. The device/application checks every so often to see if there is anything new on these feeds. If there is something new, then this gets added to some kind of last updates section of the application. The application would enable the user to register RSS feeds, to delete them, to check the latest information from the RSS feeds and, hopefully, to have some background mode of operation by which the application could check the RSS feeds periodically and alert the user if there were any updates. While it is likely that J2ME will be easier to get the RSS working, there might be difficulties with Audio content. Therefore it might be required to investigate Symbian as well.

1.1.2 Mandatory

To develop an application that can read one RSS format and can store the relevant information on a mobile device for perusal by a user.

1.1.3 Discretionary

To add functionality to this to enable the user to modify (add and delete) subscriptions to RSS feeds.

1.1.4 Exceptional

To provide functionality to enable user to determine via RSS if audio content has been added to a particular RSS feed and to download to mobile device so it can be played.

1.2 RSS

RSS, standing for Really Simple Syndication as of RSS 2.0, is an XML based method of publishing data for public consumption efficiently and with as little effort as possible. An RSS feed contains a channel, which has a title, a link to the content, a short description, and a series of items which in turn also have a title, link and a description. RSS is in wide use on the Internet, particularly in the areas of:

- News Websites; headline feeds.
- Web logging, or blogs; entries and often comments are presented with feeds.
- E-commerce; many on-line shopping sites provide RSS feeds of their latest special offers.
- Media/Content distribution; audio/video content and also application data such as torrent files can be embedded in an RSS file to allow for automated identification and downloading of content.

For the project it was necessary to research the history of RSS so as to learn about the various formats and the reasons that they are all for the most part still in use today. It was also necessary to research how the feeds are presented in XML format to parse the feeds for display on a mobile device.

1.3 J2ME

J2ME is a version of the Java platform that is optimised for resource limited devices. As the project specification requires the application to be developed for mobile devices, J2ME was chosen for reasons detailed later in this report.

1.4 Report Summary

The main chapters in the report are:

- Background Reading; this chapter describes all the research undertaken before development began on the application. Topics include RSS, SymbianOS and J2ME.
- Design; this chapter describes the conclusions that were made based on the research and how they would affect the project.
- Implementation; this chapter describes in detail the main classes and methods that make up the program. It also describes the steps necessary to deploy the application.
- Testing; Describes the tests that were performed on the application.
- Conclusion; examines the project specification and whether or not the completed project fulfills the criteria. Any possible future work is discussed in this chapter also.

Chapter 2: Background Reading

2.1 RSS

RSS is an acronym used as an umbrella to describe a family of XML file formats commonly used to facilitate the syndication of data available on a web site for the use of RSS reader programs or other websites displaying that data for their visitors.[2] Although the goal of RSS is to make the publishing of web based information like web logs (blogs) and news headlines extremely easy and hassle free, the large amounts and largely differing standards can make the job of choosing the right feed more difficult than actually using them for publishing.

For the purposes of this project, the result of this incoherence is that the RSS Reader needs to be able to take into account all the feeds currently in use including but not limited to the two separate branches that the original specification forked into. There have been three standards of RSS that have been cleared for use and these are:

2.1.1 RSS 0.91; Rich Site Summary

RSS 0.91, or Rich Site Summary was actually an update to the 0.9 specification detailed below. Work on 0.91 was initially started by Netscape, who then dropped development of it when they got out of the web portal business. Another vendor, Userland Software, picked up development of the project to use in its web logging and other web publishing software. The aim of 0.91 was to be simpler to use than the original 0.9.

RSS 0.91 is an incredibly simple and easy to use format for syndication. In 0.91 a feed consists of a channel which is the highest level of the RSS hierarchy, having a title, a link and a description. Within each channel is a series of items, each having their own title, link and description. Figure 1 shows an example version 0.91 feed in which you can see the clearly defined structure and how human readable the simplified code is.

```
<rss version="0.91">
  <channel>
    <title>Example RSS Feed</title>
    <link>http://www.rss.com</link>
    <description>An example of the format that an RSS feed version 0.91
      is presented in</description>
    <item>
      <title>Example Item</title>
      <link>http://www.rss.com/item1.html</link>
      <description>An example item</description>
    </item>
  </channel>
</rss>
```

2.1.2 RSS 0.9 & 1.0; RDF Site Summary

The original 0.9 specification was developed by Netscape as a way to easily populate its web portal with headlines from mainstream news sites. When Netscape dropped development of its own update, 0.91 development branched into two separate trees. The first is detailed in 2.1.1. The second was a non-commercial group who felt that the 0.91 standard was straying too far from the guidelines and standards originally set out in version 0.9.

This group developed a new format based on the Resource Description Framework (RDF) and is called RSS 1.0. This format is quite a bit more verbose than 0.91 and can be particularly useful for RDF based applications or anything that requires RDF specific modules. As can be seen in Figure 2, RSS version 1.0 separates the items from the channel in the feed. Instead, within the channel, an `<items>` and RDF object called `Seq` lists the resources that the items use later on in the feed. Not shown here, is the fact that item level authors and publishing dates can be included which is not supported by version 0.91.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
>
  <channel rdf:about="http://www.rss.com/about.html">
    <title>Example RSS Feed</title>
    <link>http://www.rss.com</link>
    <description>An example of the format that an RSS feed version 1.0
      is presented in</description>
    <items>
      <rdf:Seq>
        <rdf:li rdf:resource=http://www.rss.com/item1.html/>
      </rdf:Seq>
    </items>
  </channel>
  <item>
    <title>Example Item</title>
    <link>http://www.rss.com/item1.html</link>
    <description>An example item</description>
  </item>
</rdf:RDF>
```

2.1.3 RSS 2.0; Really Simple Syndication

RSS 2.0 is the current culmination of the work that Userland Software began with 0.91. Using a hybrid of 0.91, having items inside the channel as well as having no default name space or RDF, and 1.0 in that it has the ability to use name spaces if and when they are needed.

For example, to create a feed using version 2.0, it is possible to create it as though you were creating a feed using 0.91, however if you decided that you would like to include the authors of the items you are publishing, it isn't possible to have an author field in 0.91. Using 2.0 however, it is possible to include the name space for Dublin-Core, which describes item elements for metadata like authors and dates and thus you get the functionality of 1.0 without having to use RDF objects.

```

<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel>
    <title></title>
    <link></link>
    <description></description>
    <item>
      <title></title>
      <link></link>
      <description></description>
      <dc:creator></dc:creator>
      <dc:date></dc:date>
    </item>
  </channel>
</rss>

```

2.2 Choosing the Development Platform

Initial research in the project focused on identifying which platform was to be used in developing the application. It was necessary to identify and define the key goals of the project and then choose the platform that would enable those goals to be achieved fully.

The project specification was very broad in its scope of the target platform the application should be developed for, using the term "Mobile Device" in place of a stricter definition. This was taken to mean that a high degree of portability was a priority of development. This weighed heavily on deciding what was going to be used to develop the application. The issue of portability however was not the only issue that needed to be considered. Sacrificing the functionality that would be lost by making the application truly portable was not something that could be accepted lightly as a large amount of the discretionary and exceptional goals would rely on APIs that are not ubiquitous across all devices. The following sub-sections summarise the research undertaken in the initial stages to pin down exactly what would be used to develop the application.

2.2.1 SymbianOS

There are two primary means of developing applications for the Symbian operating system. The first and most prevalent is Java which is detailed in section 2.3 of this report. The second is using a modified version of the C++ programming language that was optimised for the resources available on the mobile devices much like the J2ME platform.

As with J2ME, phone manufacturers that produce phones using Symbian provide an SDK that includes the APIs for development and an emulator for testing software.

In developing the application for Symbian, however, it would be limiting the range of devices that the application would be usable on as Symbian is by no means as prevalent as J2ME is.

C++ on Symbian OS is highly customised for use on mobile devices. The need to run with only limited resources including memory and battery power means that Symbian's C++ variant takes a much more proactive role in the area of memory management, with use of descriptors and the "cleanup stack", which is how exceptions are handled. Any objects which

could be orphaned on an exception in a normal C++ program, leading to a memory leak, is tracked on the heap using this cleanup stack and is removed if an exception (or "leave" in Symbian C++) occurs.[3]

SymbianOS is an event driven operating system. The CPU on a mobile device is switched off unless an event is being handled by any of the applications that may be resident. This allows the operating system to keep power hungry resources from being overused and greatly extends the battery life.

Through researching SymbianOS it was clear that the application, if developed solely for Symbian, would work well on devices that use that Operating System. Portability from that operating system would be severely curtailed and further retarded by the fact that the C++ implementation is just different enough to make it more problematic in even porting it to other devices that have a C variant such as smart-phones and PDAs.

The complexity of developing for the Symbian platform also had to be taken into account. C++ in itself is quite a low level language. Many development tasks such as handling key presses and developing a UI can be a very tedious task, whereas with Java, these are very highly abstracted with rich APIs that hide the complexity of the hardware from the developer, while still maintaining the functionality.[4]

2.2.2 J2ME

As J2ME is essentially a reduced feature set version of Java Standard Edition, many of the ideologies that drove Java development hold true for the Mobile variant. J2ME is an attempt at making as high level as possible abstraction of the hardware upon which the framework lies. It is tailored not for any specific device, but for an environment. That environment being a heavily resource limited one, with processing power, RAM and static storage typically being a lot more restricted than in a desktop computing environment.

As a mobile development platform, J2ME has matured significantly since it's initial release, under the Java Specification Request (JSR) #68.[5] Since this initial release, all changes to the base J2ME specification as well as packages and APIs implemented by manufacturers on an optional basis are all submitted and approved/disapproved by The Java Community Process. This process allows individuals and organisations with a vested interest in the platform, to submit new JSRs and take part in the expert panels which review submitted JSRs. This has led to the platform growing in features and functionality tailored specifically to the mobile environment.

J2ME offers portability across a wide range of platforms including mobile phones, PDAs and smart-phones. It also offers a wide range of APIs that allow the application to fulfill the mandatory, discretionary and, on a subset of mobile devices, the exceptional requirements. The lack of an API which allows for scheduling access is the only thing that went against using J2ME. However the benefits of portability and the ease of developing in an already familiar language was deemed more of an advantage than developing in C++ for SymbianOS.

The specifics of what J2ME offers developers in terms of functionality are detailed more thoroughly in section 3 of this chapter.

2.3 J2ME

2.3.1 Introduction to J2ME

The traditional Java 2 platform, or J2SE, has grown so large over the years, with APIs like Swing, Java2d, Java3d etc that when the advantages of Java on small consumer devices was seen it was obvious that the platform as it was, was untenable as a development platform.

The answer to this was to pare down J2SE. By removing the bulk that allowed Java to be as portable as it is, and also remove APIs that would see little or no use on these mobile platforms, a platform began to take shape that was minimal, maintained it's portability and had very small resource consumption.

J2ME, or Java 2, Micro Edition is a version of the Java programming language that has a collection of APIs (application programming interface) that are tailored specifically to small portable consumer devices such as mobile phones and PDAs. For example, the APIs that are used in developing GUIs for programs in J2ME are designed to be used with LCD displays with a limited resolution, also allowing for alerts and full screen applications.

2.3.2 Architecture

In order to facilitate the deployment of Java across devices with a vast array of features and capabilities it was necessary to change the architecture significantly from the all-in-one nature of J2SE and J2EE. As a result J2ME is not simply a JVM with APIs. J2ME is a framework on which a J2ME "configuration" targets a range of devices with a specific set of capabilities e.g. Mobile phones. On top of this, a "profile" selects a configuration and a range of APIs that facilitate the development of a range of applications targeted at a specific device or range of similar devices, depending on the configuration specified. Essentially, there are two kinds of configurations available to J2ME.

The first is for devices that are resource limited, typically Mobile phones. This configuration is called Connected Limited Device Configuration (CLDC). It consists of a Virtual Machine and basic set of APIs for use with devices that typically have 128K - 512K memory available, limited power available, a simplified user interface and network connectivity of some kind such as GPRS, Wi-Fi or Bluetooth.

The second configuration is the Connected Device Configuration (CDC) which is the framework to build applications that can be shared across consumer and embedded devices such as PDAs and set top boxes. This configuration is targeted at those devices that have >512K memory available and are connected to a network resource.

The configurations are nestable. An application made using the less capable configuration (CLDC) should be able to be executed on the more capable configuration with little or no changes made. Configurations are at the lowest level of the J2ME architecture. Sitting on top of these are profiles.

A profile is a specification that describes the Java APIs built upon and using the APIs that are described in the configuration the profile sits on top of, that are necessary to build and execute applications for a specific kind of device.

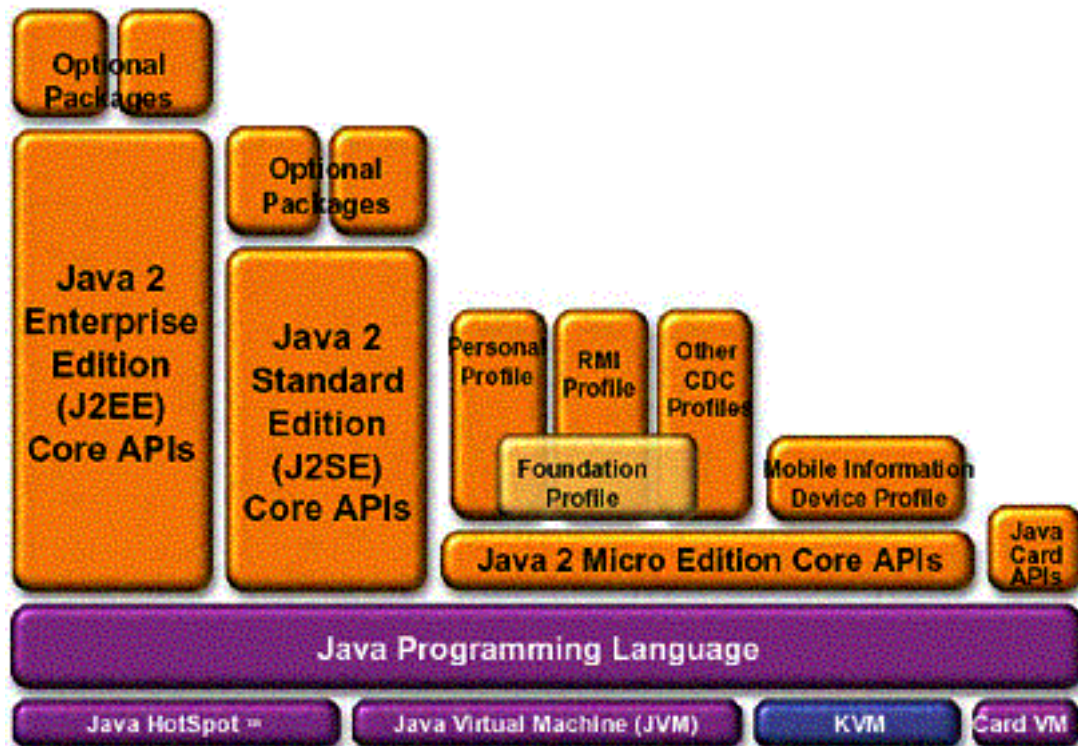


Figure 2.1: Graphical representation of the J2ME architecture

2.3.3 Mobile Information Device Profile (MIDP)

When used in conjunction with the Connected Limited Device Configuration, MIDP provides a standard run time environment for the most popular Java enabled portable information devices such as mobile phones. The specification for MIDP was designed by leading device manufacturers, wireless carriers and software vendors.

The goal of the MIDP profile specification was to provide a complete set of APIs and a properly configured Virtual Machine so that applications written using this specification would have the greatest compatibility possible with as many devices in the mobile market as is possible. Essentially, if an application was built and runs to MIDP specification it should run on any phone certified for J2ME and MIDP 2.0 compliant.

This does not mean however, that a phone's available APIs cannot be greater than those specified in the MIDP profile. Manufacturers often do provide APIs in addition to those provided by MIDP for features that are not covered or that are not available from other manufacturers, there are many JSRs specified that provide optional functionality. Such as the FileConnection API defined in JSR 075. For this reason, phone manufacturers can provide their own profiles specific to a model or series of models for use with the CLDC configuration.

2.3.4 Emulation & Signing

With J2ME development, it isn't practical to develop on the machines that you are developing for, given the nature of the resources available. To accommodate this, Sun has created a MIDP emulator, which emulates a device that is capable of meeting the specifications needed to run an application that is developed for the MIDP specification.

Emulating a mobile device that implements MIDP1/2 is not an entirely simple affair as device

manufacturers have varying levels of security for untrusted/signed 3rd party applications. For this reason, manufacturers often provide their own emulators that either emulate a specific device, or series of devices. These often have configurable options that allow the developer to emulate the security of a specific device with regards to untrusted, unsigned programs as well as trusted, signed 3rd party applications.

Restricted APIs like the Connector API and the FileConnection API are not available to untrusted applications. Applications under development can be signed for a limited amount of time using the wireless toolkit, this is solely to allow applications to be tested on the devices they are designed to be run on.

2.3.5 Record Management Store (RMS)

J2ME is a true sandbox platform; a MIDlet developed using MIDP1.0 will never have access to the file system of the device it is running on. In order to facilitate the persistence of data in Java applications, the Record Management System (RMS) was created. The purpose of this system is to allow applications to save data in the form of byte arrays which can be accessed at a later time by that MIDlet or a MIDlet that is in the same suite.

The device on which J2ME is located is responsible for maintaining the integrity of the RMS across reboots, system crashes and battery removal/drain. It is up to the developer of the MIDlet to ensure the integrity of the data that it is accessing or saving. The RMS does not care about multi-threaded MIDlets accessing the record store and it is the responsibility of the MIDlet to coordinate the access.

Records are stored as an array of bytes paired with a unique identifier. This identifier is an integer of increasing value starting at 1. Records can be added, retrieved, updated and deleted as seen fit by the application. The Record Store does a very good job of allowing very small amounts of data to be persistent between instances of applications. However, since the publication of the MIDP2 spec, it has become clear that a more powerful means of data persistence was needed.

2.3.6 File Connection

JSR-075 was a Java Specification Request that was drafted and proposed by Tom Chavez of Palmsources Inc. and Ken Waker of IBM. [6] It was drafted under a need for J2ME applications to be able to access removable storage in mobile devices such as Memory StickTM and Multimedia Memory Cards (MMCs). These storage devices typically are not home to sensitive system files, so limited access to these devices should not be counter to the sandbox design of J2ME.

Security is further heightened through the use of file and folder access restrictions. Access to files are restricted in such a way as to protect any possible private data and file that are integral to the operation of the mobile device. Folders in the device need to be set to allow the FileConnection API to read and write to those folders.[7] Many devices have folders that are used by default to hold images, video and audio. Nokia for example, use system properties that can be passed directly to the Connector object using the open(URL) method, such as "fileconn.dir.tones" for audio files and ringtones. [8]

If an application has need to use the FileConnection API then on deployment, the JAD file accompanying the JAR package needs to request the read and write permissions using

the MIDlet-Permission property. Without requesting this, depending on the default access restrictions of the device, the application may not be able to read files, and more likely than not, will be unable to write to disk either.

When developing applications that make use of this API it is important to make sure that other functionality in the application will still be accessible if the FileConnection API is not available. To this end, the system property "microedition.io.file.FileConnection.version" has been added which will contain the current version of the API that is implemented by the device. If this is missing on a device then an attempt to access it will throw an exception which should be handled and then disable any functionality that relies on this API. It will return a string if present, allowing the application to continue normally.

The use of FileConnection is very straightforward. As it is just an implementation of the Connector object, passing an URL with the prefix "file://" will cause the Connector object to instantiate the FileConnection automatically. If access to the file system is needed in order to create, rename, delete or otherwise manipulate files then the FileConnection object can be created and methods invoked to carry out those commands.

```
String fileUrl = "file://**Directory to save to**/testFile.txt";
FileConnection fileConn;
try {
    fileConn = (FileConnection) Connector.open( fileUrl,
        Connector.WRITE_ONLY );
    if( fileConn.create() ){ // create the file
//implement code to write to file
    }
    fileConn.close();
}
catch( IOException e ){
    //handle exception
}
catch( SecurityException e ){
    // if a SecurityException is thrown then there is no permission
    // to write to the directory. User should be notified to try another folder.
}
}
```

The above code is an example of using the FileConnection object "fileConn" to create a file called "testFile.txt," if creation is successful then code to write to the file would be executed if implemented.

Chapter 3: Design

The background reading that was undertaken, first in determining the platform to develop on, and then the more detailed research into J2ME was crucial in determining how to approach the project. It was necessary to identify key goals and milestones in the project that were very strongly tied to the three sections of the project specification. This chapter will outline and describe the methods of designing the project and the conclusions that were drawn from the research that had been carried out.

3.1 Conclusions derived from background reading

It was initially very important to take what had been learned from the time spent researching the project and work out how to begin to develop the application. Each area of the project's scope has implications for how development was to proceed.

3.1.1 RSS Compatibility

As the research detailed in section 2.1 suggests, there is no clear upgrade path with regards to the various RSS feeds. There is no motivation for someone publishing a feed using RSS 1.0 to change their feed to RSS 2.0. The primary reason being that 2.0 does not support the advanced RDF functionality that RSS 1.0 does. Similarly, a feed using 0.91 has no reason to upgrade to 2.0 unless they feel that they have a use for the namespace functionality that it offers.

For this reason, the RSS reader would be required to implement a parser capable of parsing any number of RSS feeds without having to worry about which version it would be parsing. It was not a simple case of being able to say that one standard above the others was being used in the majority of feeds. That is not the case, all the standards are still in use and all need to be accommodated.

3.1.2 Implementation and use of the FileConnection API

After researching the FileConnection API and what functionality it is able to add on devices that support it, it was decided that it would be used to fulfill the exceptional goals of the project. It was still clear however that since the API is an optional package and not implemented by every mobile device manufacturer, the Record Management Store would still need to be used in order to have the user inputted feeds remain persistent even when the application is closed by the user.

It was therefore decided to be very important that a device that does not implement the API should still be able to access the functionality that does not rely on the FileConnection.

3.2 Proof of Concepts

Rather than begin to develop the application immediately. The decision was made to write a few very small MIDlets that would explore some of the functionality that would later be employed to create the RSS reader.

These applications were essentially proof of concepts for elements such as the GUI and the Record Management Store, in order to better understand how they are implemented in J2ME. One such application was developed with the aid of GUITests[9], a small MIDlet that serves as a sample implementation of an interface using the lcdgui API. The other application was a small MIDlet that stored a simple string in the RMS. The second such application was a small MIDlet that implemented an interface for the RMS, which just stored a simple string that could be retrieved again once the application had closed and started again.

These served as valuable tools in deciding how to approach the development of the project.

3.3 Goals and Milestones

With the issue of what direction the project was to take decided following the conclusion of the research, it was necessary to lay out a road map of key milestones and goals that would need to be met along the development of the application. These were laid out as follows:

- A method of parsing an RSS feed was needed that would create a data structure containing all the data contained in a feed in an easily retrievable form.
- Writing a GUI that would interface with this parser to download a hard coded feed and display the contents of the feed.
- Implement an interface for the RMS so the user could subscribe to feeds.
- Further develop this interface to allow the user to edit subscriptions and delete them if necessary.
- Implement a means of identifying audio in a feed and downloading the audio for playback.
- Utilise the FileConnection API to allow the user to download and save audio to the device.

With a clear path of objectives to meet in developing the application, work began on the class structure the final program should implement.

3.4 Program Structure; Classes and key Methods

In designing the structure of the program; the classes and methods that would be used, it was necessary to determine at a high level what design features the application would require. These were identified as:

- The main class, or MIDlet.
- A GUI, to be implemented in the main class.
- Two classes defining datastructures representing each individual RSS feed and also items.
- A class implementing an RSS parser
- The Record Management Store would be implemented as either a separate class or a series of methods in the main class.

With this information laid out, it was becoming clear what structure the final application should look like. The next step was to take this data and create a formal class specification. The class names and a brief description follows:

- RSSReader; main class, implements MIDlet
- RSSParser; implements a parser for the RSS feeds, creates RSSFeed class with necessary number of items
- RSSFeed; simple datastructure to hold feed information, implementing some sort of datastructure for holding, or pointing to, the constituent items.
- RSSFeedItem; simple datastructure to hold the attributes of an item in a feed.
- RSSRecordStore; RMS interface class, provides methods that allow data to be loaded from and saved to the RMS.

Chapter 4: Implementation

This chapter will detail how the application was implemented once the basic structure of the program was designed, the development environment, and how the research carried out before development helped in the development of the application. It will also deal with any problems that were encountered during development and how they were resolved. Any problems that were not able to be resolved will be discussed and what was done to work around them.

4.1 Development Environment

Before development began it was necessary to choose an Integrated Development Environment (IDE) with which the application would be developed. The IDE chosen was Eclipse, an open source project that was developed in order to provide a vendor neutral IDE for developing software. This ran on top of the JavaSDK1.5, with compile options set to Java 5.0 compliance.

Eclipse provides support for a variety of plug-ins which allow the IDE to be used for much more than just Java development. One of these plug ins which was used in the development of the project was the EclipseME.org J2ME plug-in, which implements functionality that aids in the development of J2ME applications. This functionality includes fully integrating the Java wireless toolkit, which provides configurations and profiles for mobile development. The plug-in also integrates the MIDP2 emulator that is provided with the wireless toolkit which can then be launched from within Eclipse.

The second plug-in used in developing the application was the Nokia Development Suite version 2.0. The device used as the test platform was a Nokia 6630 mobile phone, which is a series 60 phone. This plug-in provided support for emulating the series 60 model phones which run the Symbian operating system. It also provided tools for deployment of packages, automatic JAR and JAD file creation and the ability to sign the application for a short period of time for testing purposes.

The Eclipse IDE along with the two plug ins described above formed the Development Environment under which the application was developed. With Eclipse providing error checking and compilation management, the J2ME plug-in providing the CLDC and MIDP2 APIs, and the Nokia Developer Suite2.0 providing emulation and deployment packages, the development of the RSS Reader could begin.

4.2 RSS Reader; detailed description

This section will describe the classes and methods of note that were developed for the application.

4.2.1 MIDlet class; RSSReader.java

The executable class of a J2ME application is the class that implements the MIDlet super class. It is necessary to do this as only a MIDlet implemented class can be executed on a MIDP2 device. In the RSS Reader application, this class is RSSReader. The main purpose of this class is to provide the GUI of the application. There are five main components of the GUI:

Note: all images are of the application running on the Nokia series 60 emulator as provided by the Nokia Developers Suite version 2.0

- *Feed List*; This is the main screen. Contains a list of all saved bookmarks. Through menu, allows user to add a new bookmark, edit an existing one, download and view a feed or save the feeds and exit.

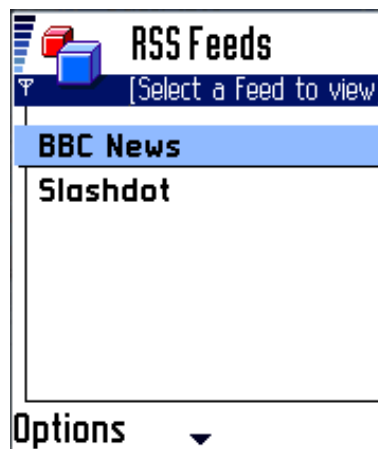


Figure 4.1: Image of the feed list from the RSS reader.

- *Bookmark Form*; When a user chooses to add or edit a bookmark this screen is shown. Contains a form with two entries, one for the name of the feed and the second with the feed URL.

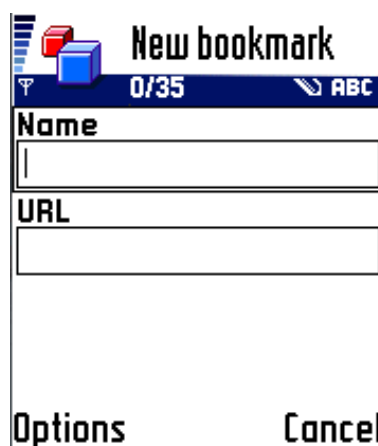


Figure 4.2: Image of the bookmark form from the RSS reader.

- *Loading Screen*; This is an interim screen shown between selecting a feed and displaying its contents. Its purpose is to assure the user that the application is working, this is further reinforced by a ticker scrolling across the screen reassuring the user that the application has not frozen if the feed is taking a long time to download.

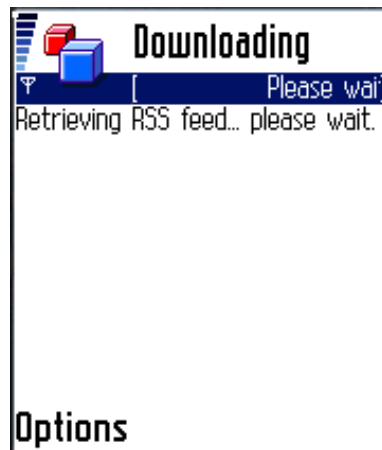


Figure 4.3: Image of the loading screen from the RSS reader.

- *Item Browser*; Like the Feed list, this screen contains a list of all the items contained in the feed. The menu options allow the user to open an item, update the feed or return to the feed list.



Figure 4.4: Image of the item browser from the RSS reader.

- *Item Details*; A screen containing a text box containing a description of the item and the link contained in the item, if available.

This class also handles all the commands received from the user. The method `commandAction(Command cmd, Displayable dis)` is called whenever the user chooses a menu command or presses a key, and the method executes the proper response.

4.2.2 Parsing the RSS feeds; kxml2 & rssParser.java

It became clear during the initial stages of development that developing an RSS parser would be a very impractical use of time that would be better spent on implementing more features into the application. Unfortunately, there is no API provided in CLDC, MIDP2 or in optional JSRs that would help in parsing RSS feeds. However in looking for one, an Open Source project was discovered that aims to provide functionality to allow XML parsing.

kXML2 is an implementation of the XmlPull API specifically designed for use on heavily resource limited devices such as those that MIDP2 is designed for. It works at a higher level

than RSS, instead parsing all forms of XML. The only constraint being that the XML code is well formed, the small footprint needed so that it can be deployed to mobile devices means that it can not check for and handle errors that more robust parser would be able to recognise and handle.

The `rssParser` class, has a single constructor which takes in an `RSSFeed` object as an argument, assigning that feed to the class variable `rssFeed`. The primary method of note in the class is `parseRssFeed()`. This method downloads the RSS feed using the URL contained in the `RSSFeed` object that was passed in the constructor. Using the `HttpConnector`, the application attempts to connect to the URL, and if successful, creates an input stream from the `HttpConnector`.

This input stream is then passed into the method `parseRssFeedXml(InputStream is)`. At this point, the `kXML2` parser is used. A `kXmlParser` object is created, and it's input is set to the input stream passed into the method. The first task, is to check the input stream to ensure that it isn't an empty RSS feed, or that it isn't an RSS feed at all, this is done through the use of an iterative loop, if the end of the page is reached and no `<item>` tag is found, then an exception is thrown. Otherwise, the feed stepped through and for each `<item>` tag found, a new `Item` object is created with the relevant title, description and link. The `Vector` of `RSSFeedItems` that is contained in the `RSSFeed` object is retrieved with the `getItems()` method and the new `RSSFeedItem` is added with the `Vector<RSSFeedItem>.addElement(RSSFeedItem item)` method.

Audio data is embedded in RSS feed items through the use of the `<enclosure>` tag. The enclosure tag can have several attributes, among them the URL and type attributes. If the parser finds an enclosure tag, it then looks at it's attribute, if it finds an URL and that the type tag contains the substring "audio/" then it adds the URL to the item, flagging it as containing audio data. The `MIDlet` then applies the correct menu items to that item when viewed with the application.

4.2.3 Datastructure classes; `RSSFeed.java` & `RSSFeedItem.java`

Two objects were required which could store the data contained in an RSS Feed. The first of these was the `RSSFeedItem` object.

The `RSSFeedItem` object contains a constructor which takes in four parameters, all `Strings`. These strings represent the title of the item, the description, the URL it contains and an URL to audio data. Each one is accessible via getter methods. If the `audioUrl` is an empty string, then this indicates that there is no audio content. If it is not empty then there is audio content then the boolean `hasAudioContent` is set to true. There is also a getter method called `hasAudioContent()` that returns that boolean.

The `RSSFeed` object contains two constructors, the first, is used when a user adds a new bookmark and takes in two strings. These are the name of the feed and the URL of the feed. The second constructor is called at application start up when the saved feeds are retrieved from the `Record Store`. This takes in a concatenated string composed of the name, the break symbol `'|'` and the URL. The constructor breaks the string up using substrings based on the index of the `'|'` character and assigns them to the relevant class variables. The object also contains a method called `getStoreString()` which returns the name and URL of the object concatenated, for saving to the `RMS`. The last method `getItems()` returns the `Vector` which contains the `RSSFeedItem` objects for display or to add a new one.

4.2.4 Record Management Store(RMS) interface; RSSRecordStoreInterface.java

Creating a class to interface with the Record Store was accomplished very much with the help of researching a sample application provided by IBM[10].

The RSRecordStoreInterface class provides methods that allow the application to save and load data which will remain persistent across instances of the application. The main task of this class, is to take the HashTable rssData in the `save()` and `load()` methods. In the case of the save method, the HashTable is converted into a byte array and added to the record store, and in the case of the load function, a byte array is retrieved from the record store and converted into a HashTable. The `getStringProperty(String, String)` is called by the RSSReader class to retrieve the saved feeds.

4.3 Verification

Before deployment of an application, all classes need to be pre-verified by the wireless toolkit verification tool. This does a part verification of the classes, splitting the entire verification process between the build part of the development process and run-time on the device. The reason for this is due to the limited resources of MIDP2 devices they cannot verify that every class will not pose a security risk, the standard JVM verifier is too large and resource consuming to run on mobile devices necessitating this split. Classes will not pass this strict verification if any warnings are given by the java compiler.

4.4 Packaging & Deployment

Assuming the application and any additional classes have been verified, the classes can now be packaged for deployment on a mobile device.

4.4.1 Packaging; JAR and JAD files

Packaging the application was largely automated with the use of the Nokia Developer Suite and Eclipse. "Generate Package" is an option in the tools menu that once selected.

The JAR(Java Archive) file, is an archive that contains all the application classes, any extra resources that the application requires, and a manifest file which contains information on the MIDlet suite that the device will read before installing the application. This information must include the attributes MIDlet-Name, MIDlet-Version, MIDlet-Vendor, MicroEdition-Profile, MicroEdition-Configuration, and the MIDlet name for each MIDlet in the suite.

The JAD (Java Application Descriptor) file is an optional file which gives the developer more control over the environment that the application is going to be run under. It also allows the application manager of the device to determine if the device meets the minimum requirements needed to run the application. If the application is signed, then the encrypted key is found

inside the JAD file.

The JAD file for this project requests permission to access the FileConnection API as well as providing the public key for the JAR file so the application runs as a trusted application.

4.4.2 Deployment

The application was deployed to a mobile device by two means. When available, JAR and JAD files were transferred directly onto the memory card of the test device and using the file manager of the device, the JAD file was executed directly.

Where that means was unavailable, the JAR and JAD files were published to a website and installed via the Internet browser on the mobile device.

Once installed, the application is available in the application manager of the device.

Chapter 5: **Testing**

Testing of the application was a largely straightforward affair. Short of downloading a feed, displaying it on the screen, and checking that the entire feed has been retrieved and displayed correctly, there is no empirical results to speak of. That said, a good deal of Functional Testing was carried out, with particular attention paid to correctness of execution and looking for unhandled exceptions.

5.1 Functional Testing

The functional testing of the application consisted entirely of attempting to break the application by trying as many combinations of key presses and menu choices as possible. The number of options is quite limited so when coding the application, complexity traps that lead to functional errors were few, and limited only to simple logic errors that were quickly found and fixed.

An example of a pattern used in testing the application:

- Start the application.
- Add a new bookmark.
- Edit an existing bookmark.
- Choose a feed to download.
- Browse the feed and view some items.
- Return to feed list and choose a new feed.
- View an item in that feed, quit application.
- Result: Test passed.

Such tests were performed every time a new function was added.

Chapter 6: Conclusion

This chapter will evaluate whether or not the project specification was met, any weaknesses in the approach taken and possible directions the project could be taken in the future.

6.1 Project Evaluation

The project specification is included in full in the introduction of this report. In this section each goal will be examined and whether or not it was implemented in a way that satisfies the requirements laid out will be discussed. Any shortcomings with the implementation will be highlighted and discussed.

6.1.1 Mandatory

The mandatory section outlined a requirement to have an application that could read a feed in any one RSS format, parse the feed and handle the information contained in that feed for viewing by the user.

The application is fully compliant with this specification. The reader can, demonstrably, download a specified feed from an URL contained in the program, parse the feed into a Vector of items containing all the necessary information for each item; the title of the item, the description, and the accompanying link.

6.1.2 Discretionary

The discretionary section requires that the user be able to input the URL of a feed and have the application save this URL. The user should also be able to delete feeds that have previously been subscribed to.

The application uses the record store to create a HashTable using the concatenated strings of all the combined feed names and URLs. This allows the feed subscriptions to remain persistent across active instances of the program. Upon start up, the application retrieves the HashTable from the record store and splits the concatenated string back into each feeds name and URL. The user can then choose the delete option to delete the currently selected feed, or choose edit and change the name and/or URL of the feed. The HashTable is recreated and stored in the record management store if the option 'Save and Exit' is chosen in the options menu of the feed selection screen.

As a consequence of this, the discretionary requirement is fulfilled.

6.1.3 Exceptional

The exceptional section of the project specification requires that the application be able to determine whether or not a feed contains audio data, and to download this data to the device for playback.

When the RSS feed is parsed, the parser loops through every tag in the feed. For each tag, it checks to see whether it is a tag of significance. This allows the parser to build each item. RSS feeds embed links to media files and often application files using the <enclosure> tag as previously discussed in chapter 4, section 2.2 of this report. If the parser finds such a tag, it then loops over it's attributes to locate the URL attribute, and the type attribute. It then checks if the type attribute contains the substring "audio" and if found, sets a flag in the item in question that audio data is present.

When the user selects an item that contains audio data, one of two things will happen depending on whether or not the FileConnection API is present. If the API is not present, then one options will be available to the user, "Download and Play" will essentially stream the audio file to the device and playback the file without saving the audio file to the device.

If the API is present, then a second option will be available to the user, "Download and Save" will retrieve the audio file from the URL, and prompt the user for a location to save the file.

While it would be preferable to give the user the option to save the file under all circumstances, the FileConnection API being optional means that that can never be the case. Giving the user the option to stream the file was decided to be an adequate compromise. As a result, the exceptional requirements have been met, as the audio can be downloaded and played in both scenarios.

6.2 Description specifics and implementation

Although it was never specified in the mandatory, discretionary and exceptional requirements, the description mentions that the application should be able to download feeds automatically at regular intervals determined by the user. The application should also have been able to determine if anything new had been posted to the feed and if so, download the feed again.

J2ME as of the time of application development, does not have an API that allows for scheduled execution of an application. As a result of this, the choice to develop the application on J2ME meant that the RSS reader would not be able to have scheduled updates.

6.3 Future Work

In this chapter, some consideration will be given to some ideas on where any future work on the application could go. This are composed mostly of things that were not implemented due to time constraints, or ideas that could not be implemented due to the lack of an acceptable API.

6.3.1 Scheduling

The project specification made mention of the ability for the program to automatically check for updates to an RSS feed, and if found, download the feed for later perusal by the user. Unfortunately, at the time of development, there were no J2ME APIs available to facilitate scheduling the application to run.

However, MIDP 3.0 is currently making it's way through the Java Community Process and this specification will allow for background MIDlets and scheduled execution[?]. It would be recommended that when this becomes available, the application be updated to make use of this added functionality.

6.3.2 Following the included link

The nature of RSS is that each item in a feed is a small description of an article that the item refers to. The feed should also contain a link to the full article that the item describes. Unfortunately, there was not enough development time to research and implement a way for the application to open an external program, like a web browser, that would follow the link to the web page it to which it belonged.

Should any further work be conducted on the application, this is functionality that should be looked at being implemented.

Bibliography

- [1] kXML Project. *kXML2*. <http://kxml.sourceforge.net/> 2005
- [2] Mark Pilgrim. *What is RSS?*. <http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html> 2002
- [3] Richard Barker & Leigh Edwards *Getting Started With C++ SDKs and Symbian Development* <http://www.awprofessional.com/articles/article.asp?p=176459&seqNum=2&rl=1> 2004
- [4] Nokia *Symbian OS Basics* Nokia Developer Training; Course Pack 2006
- [5] John Courtney, Sun Microsystems, Inc. *J2ME™ Platform Specification*. <http://www.jcp.org/en/jsr/detail?id=68> 2000-2002
- [6] Tom Chavez & Ken Walker. *JSR 75: PDA Optional Packages for the J2ME™ Platform*. <http://jcp.org/en/jsr/detail?id=75> 2000-2004
- [7] Eric Giguere. *An Overview of the File Connection Optional Package*. <http://developers.sun.com/techttopics/mobility/apis/ttips/fileconnection/>, 2003
- [8] Nokia. *Introduction to the FileConnection API v1.1*, 2003 , http://www.forum.nokia.com/info/sw.nokia.com/id/82644083-2f4b-4775-a292-c02d6bf5be57/Introduction_To_The_FileConnection_API_v1.1.zip.html
- [9] Qusay H. Mahmoud. *MIDP GUI Programming: Programming the Phone Interface* <http://developers.sun.com/techttopics/mobility/midp/articles/ui/> 2000
- [10] Soma Ghosh. *J2ME record management store*. <http://www-128.ibm.com/developerworks/wireless/library/wi-rms/> 2002
- [11] Mike Milikich & Jim Van Peurse *JSR 271: Mobile Information Device Profile 3* <http://www.jcp.org/en/jsr/detail?id=271> 2005