FORUM NOKIA

J2ME™ RSS News Reader

Version 1.0; November 24, 2003

Java™

NOKIA

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1 Introduction 6

1.1 Topics 6

1.2 Software Requirements 6

2 RSS Basics 7

2.1 Origination of RSS 7

2.2 What Is RSS? 7

2.3 Aggregators 7

3 Inside an RSS File 8

3.1 Inside an RSS File 8

3.1.1 Channel tag 9

3.1.2 Item tag 9

3.1.3 Additional tags 9

4 Creating a J2ME News Reader 11

4.1 Application Screen Shots 11

4.2 Application Design 13

4.3 Class Files 14

4.3.1 NewsReader 14

4.3.2 NewsSourceEntry 15

4.3.3 HeadlineList 15

4.3.4 NewsParser 15

4.3.5 ArticleDisplayForm 16

4.3.6 AddForm 17

4.3.7 EditForm 18

4.4 Parsing an XML File 20

4.4.1 Types of XML parsers 20

4.4.2 KXML parser 21

4.4.3 Parsing a news source 21

4.5 Deleting News Sources 22

4.6 Writing News Sources to Persistent Storage 23

4.7 Displaying News Sources 24

4.8 Displaying Headlines 25

4.9 Displaying Descriptions 27

5 Source Code 28

5.1 NewsReader.java 28

5.2 HeadlineList.java 35
5.3 NewsParser.java 37
5.4 ArticleDisplayForm.java 40
5.5 AddForm.java 41
5.6 EditForm.java 42
5.7 InfoForm 44
5.8 NewsSourceEntry 45
6 Summary 47
7 Terms and Abbreviations 48

Change History

November 24, 2003	V1.0	Initial document release

1 Introduction

Much of the information flow from large organizations can be treated by an application as a news feed, and moving that information flow to mobile devices can be a major productivity step for the enterprise. An ever-increasing percentage of the internal workforce is working on the road and needs access to the company data. In addition to this very important factor, the mobile worker also needs to stay informed by using other information sources. Today critical business information sometimes appears first online. A connected mobile device could fit into the overall information infrastructure.

One technical option available to the programmer is to connect a mobile handset to the enterprise backbone using Web Services, proprietary protocols, WML, XHTML, and Short Message Service (SMS)/Multimedia Messaging Service (MMS). Another option is to use a method that includes Rich Site Summary (RSS), a data format based on XML for distributing content over the Internet. An RSS feed could be used by a mobile device to pull public information (news, announcements, contact data, product data, and so forth) from the enterprise server or to access other RSS news sources from third parties.

This document introduces developers to a method for accessing RSS documents from within a Java™ 2 Platform, Micro Edition (J2ME™) application. Thousands of Web sites already publish pointers to content using RSS, which allows them to syndicate content more easily. Our example is using open RSS feeds as an example. Usually RSS is used to provide public information for everyone. While it is possible to transfer account data to control any access to confidential information, a more secure technology should be used instead for this specific purpose.

RSS, using only the most important information to specify a document, and thus reducing the size of such an entry to the bare minimum, is a perfect fit to build up and use news feeds or any other content feeds that could also reach out to mobile devices. We'll look at all the specifics to get up and running with RSS. This will include a brief introduction to the history of RSS, the format and parsing of RSS files, as well as creating a complete sample Mobile Information Device Profile (MIDP) application to read and display RSS headlines from various sources.

1.1 Topics

In the first section, we'll introduce the basics of RSS: where it originated, understanding the underlying format of an RSS document, and concluding with a brief overview of content aggregators.

Once the foundation is in place, we'll describe the format of RSS documents. This will encompass a close look at the XML documents that make up news feeds, exploring and explaining all relevant markup tags.

The bulk of this article is in the final section, creating a MIDlet to display RSS news sources and their accompanying headlines.

1.2 Software Requirements

For the development of the RSS news reader, you will need to install the Java Development Kit (JDK™), the J2ME Wireless Toolkit, or the Nokia Developer's Suite for the Java™ 2 Platform, Micro Edition, and one or more Nokia emulators. Visit the Tools and SDKs section of Forum Nokia for an exhaustive list of Java-related tools for mobile development.

2 RSS Basics

2.1 Origination of RSS

RSS was originally developed to allow users of the My Netscape portal access to external news feeds. Although news feeds are no longer available on the portal, RSS lives on.

Based on XML, RSS provides a standard means of describing and syndicating Web-based content. In March 1999, Netscape introduced Version 0.9 of the RSS specification. This was quickly followed by version 0.91, released in July of that same year. One unique characteristic of these particular RSS versions is their holding power — it is estimated that more than 95 percent of RSS feeds provide content in the format specified by 0.90 or 0.91.

2.2 What Is RSS?

As the name implies, RSS is summary of content on a Web site. At the lowest level, an RSS document is nothing more than an XML file. With the inherent simplicity of RSS, document summaries can be created and edited with ease. In the sections to follow, we'll explore just how easy it is to read and parse RSS feeds.

2.3 Aggregators

Before getting into the heart of RSS, it's important to touch upon where one can find RSS news feeds. An aggregator, as the name implies, is a Web site that contains links to various RSS feeds. In a typical enterprise environment, the RSS feed could be the public company Web server or the Intranet server (with additional security steps like mandatory account information in the request header and/or XML encryption).

Aggregators typically operate in one of two ways: They either search for feeds or require users to register their news sources. In either case, news feeds are most often sorted and presented in categories to simplify locating topics of interest. There are many popular aggregators; here are few sources to get you started:

NewsForSites

Moreover

Meerkat

NewsIsFree

Syndic8

NewsForge

3 Inside an RSS File

3.1 Inside an RSS File

Sample XML File

Let's begin by dissecting the contents of an RSS document. The following is a partial listing from E-Commerce Times, a news feed located at:

http://www.ecommercetimes.com/perl/syndication/rssfull.pl

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE rss (View Source for full doctype...)>

<rss version="0.91">
<channel>
 <title>E-Commerce Times</title>
  <link>http://www.ecommercetimes.com</link>
  <description>E-Commerce Times: E-Business and Technology Super
Site</description>
  <language>en-us</language>

  <item>
    <title>AMD's Next Move</title>
    <link>http://www.ecommercetimes.com/perl/story/21073.html</link>
    <description>AMD entered 2003 with more to lose than ever before --
including a
     healthy market share in the desktop and notebook chip markets. Now,
the company
     plans a slew of new product releases that could sway its fortunes for
good or
     ill. Will AMD rise above the challenges that face it to emerge
stronger than
     ever, or will it stumble and fall behind?</description>
   </item>

  <item>
    <title>Court Ruling Could Help Fight Spam</title>
    <link>http://www.ecommercetimes.com/perl/story/21077.html</link>
    <description>In a legal action that may help combat spam, a federal
appeals
    court has ruled that a law restricting junk faxes is constitutional.
Anti-spam
    experts hailed the decision, noting that it creates a legal precedent
that could
    help create similar legislation banning junk e-mail.</description>
  </item>

   ...

</channel>
</rss>
```

After the requisite version information, notice that a <channel> tag is defined, followed by a number of <item> tags. The <item> tag will form the foundation of the content for the news reader created within this document.

3.1.1 Channel tag

The <channel> tag contains the basic information about the document: who created it, a link to the creator, a description of the channel, and the language of the channel contents. Beyond this, we don't need to be concerned with the <channel> tag, as it will be ignored when parsing a news feed.

3.1.2 Item tag

An <item> represents a page, or content within a page on a Web site. The <title> and <link> tags are required, whereas <description> is an optional sub-element. As we'll see, the news reader MIDlet will look for this tag in order to build up a list of headlines and their associated descriptions.

In the 0.91 specification, there are a maximum of 15 <item> tags per channel.

3.1.3 Additional tags

For many RSS news feeds, <item> is the primary tag of interest. However, for completeness, it's important to point out there are many more sub-elements available. For example, <lastBuildDate> specifies the last time the channel was updated:

```
<lastBuildDate>Mon, 24 Mar 2003 16:17:27 GMT</lastBuildDate>
```

Following are the required and optional sub-elements of the channel tag taken directly from the 0.91 specification:

Required:

- description
- language
- link
- title

Optional:

- copyright
- docs
- image
- item
- lastBuildDate
- managingEditor
- pubDate
- rating
- skipDays

skipHours

textinput

webMaster

4 Creating a J2ME News Reader

4.1 Application Screen Shots

The following figures show the NewsReader MIDlet, including the headlines and story descriptions retrieved from two different RSS news feeds. The leftmost screen shot is a list of news sources; the middle screen displays a list of headlines at the selected source; the rightmost screen is the description of the associated headline. Depending on the device simulator used, the actual look and feel might be different and additional user interaction might be required, depending on the device security options — for example, when using a MIDP 2.0 simulator.

Figure 1: Humorous Quotes Web site with headlines and story description

Figure 2: Slashdot.org Web site with headlines and story description

The following figures walk through the process of adding a news source. Figure 3 shows the original news source list, a menu of application options, and a Form with two TextFields to input a news source description and link.

Figure 3: Adding a news source

Once the add Form is complete, the options available are to save the changes or return to the main list. In Figure 4, notice that the new entry appears at the bottom of the display.

Figure 4: Save and display the news source

The final option within this MIDlet is to edit an existing description and/or link of a news source. Figure 5 shows the steps to display the edit Form, which includes two TextFields: one for the description and one for the link.

Figure 5: Editing a news source

In Figure 6, the news source "Slashdot.org" has been changed to "Slashdot." Clicking the "Options" soft key and selecting "Save" returns you to the main display, with the updated name now visible.

Figure 6: Saving an edited news source. Removing the ".org" from the description

4.2 Application Design

Figure 7 shows the big picture view of the news reader MIDlet. `NewsReader` class is the workhorse of the application; it contains the core functionality:

Displaying a list of available RSS news sources.

Internal management of the news sources (names and HTTP links).

Handling of events: displaying add/edit forms and dispatching the XML parser to read headlines and articles.

Reading and writing news sources (names and links) to/from persistent storage (record store).

Displaying status information while downloading and parsing content.

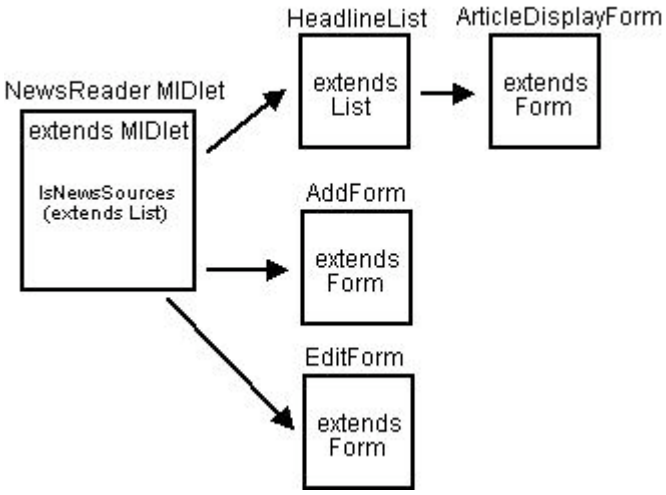

Figure 7: NewsReader

4.3 Class Files

The following sections include a quick summary of each class in our MIDlet. As mentioned previously, `NewsReader` wraps the entire application together, calling other classes to carry out various operations, such as parsing an XML document or displaying an add/edit form.

4.3.1 NewsReader

Using a `List` component (`lsNewsSources`), this class is the primary interface between the application and the user. Along with managing the display, this class must also keep track of news-source names and their associated HTTP links. The latter task is accomplished using a vector that contains objects of type `NewsSourceEntry`. Following are the key declarations inside this class:

```
public class NewsReader extends MIDlet implements CommandListener
{
  ...
  protected List lsNewsSources;       // Main list of rss sources
  protected Vector vecNewsSources;    // Vector of NewsSourceEntry objects
  private AddForm fmAddNewsSource;    // Form to add news source
  private EditForm fmEditNewsSource;  // Form to edit news source
  private HeadlineList lsShowHeadlines = null; // List of headlines from
rss
  private RecordStore rsNewsSources;  // Reference to record store
  ...
}
```

4.3.2 NewsSourceEntry

Each news source has a name (E-Commerce Times) and a link to an RSS news feed (http://www.ecommercetimes.com/perl/syndication/rssfull.pl). There is one instance of `NewsSourceEntry` for each news source.

```
public class NewsSourceEntry
{
  private String description, link;
  ...
}
```

These objects are stored in the vector `vecNewsSources`, which is defined in `NewsReader`.

4.3.3 HeadlineList

The `HeadlineList` class extends the List component and is invoked from within `NewsReader` to build a list of titles (headlines) and associated descriptions from a news source. The primary methods of this class are shown here:

```
public class HeadlineList extends List implements CommandListener
{
  private Vector vecTitle, vecDescription;
  ...

 protected void getHeadlines(String source)
  {
    ...
    /* Create and start a NewsParser thread */
    NewsParser tmp = new NewsParser(source, this);
    tmp.start();
  }

  public void newHeadline(String title, String description)
  {
    // Add the title and description to vector
    vecTitle.addElement(title);
    vecDescription.addElement(description);

    // Add the current title to the list component
    this.append(title, null);
  }
}
```

`getHeadlines()` invokes the XML parser as a separate thread to read content from the requested news source. For each new title and description, `newHeadline()` is called to add the information to the vectors.

4.3.4 NewsParser

Once a news source is selected from the main display (see Figures 1 and 2), this class is responsible for opening a connection to the news source via HTTP, parsing the source for `<item>` tags, and sending each item's title and description to the `HeadlineList` class shown in the previous section.

```
public class NewsParser implements Runnable
{
  ...
  private String source;   // News source to read from
  private HeadlineList listComponent; // List component of headlines

  public NewsParser(String source, HeadlineList listComponent)
  {
    this.source = source;
    this.listComponent = listComponent;
  }

  public void run()
  {
    readNews();
  }

  public void start()
  {
    Thread thread = new Thread(this);
    ...
  }

  private void readNews()
  {
   ...
   headlineTitle = parser.read().getText();
   headlineDescription = parser.read().getText();
   ...
   listComponent.newHeadline(headlineTitle, headlineDescription);
   ...
  }
}
```

Notice the reference to `HeadlineList`, which allows this class a means to call `newHeadline()` to add a title and description to the vector contained within the `HeadlineList` class.

We'll take a closer look at the details of parsing content from a news source in Section 4.4.

Note: An alternative to the implementation shown here would be to use a *callback* to invoke the newHeadline() method.

4.3.5 ArticleDisplayForm

The sole purpose of the `ArticleDisplayForm` class is to display an article headline and description. Derived from the Form component, the title is set to the article title and a StringItem is used to display the description. See the rightmost screen shot in Figure 8.

Figure 8: Displaying headlines and an article description

```
public class ArticleDisplayForm extends Form implements CommandListener
{
  private StringItem siArticle;

  ...

  /*-------------------------------------------------
  * Set the StringItem to the article description
  *-------------------------------------------------*/
  protected void setArticle(String str)
  {
    siArticle.setText(str);
  }

  /*-------------------------------------------------
  * Set title of Form to article title
  *-------------------------------------------------*/
  protected void setArticleTitle(String str)
  {
    setTitle(str);
  }
  ...
}
```

4.3.6 AddForm

In addition to the news sources hard-coded into the MIDlet, you can insert additional sources using the AddForm class.

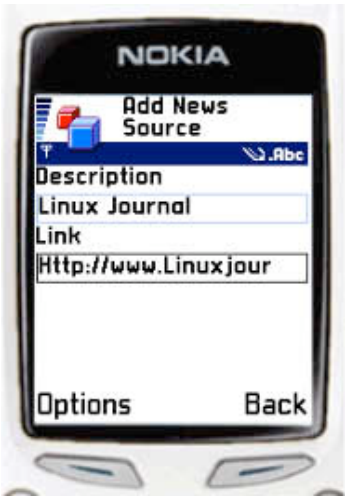

Figure 9: Add form and two TextFields

```
public class AddForm extends Form implements CommandListener
{
  private Command cmBack,
                  cmSave;
  protected TextField tfDescription;
  protected TextField tfLink;

  ...
  midlet.add2Vector(tfDescription.getString(), tfLink.getString());
  midlet.lsNewsSources.append(tfDescription.getString(), null);
  ...
}
```

Upon selecting the command `cmSave`, the description and link are stored in vector `vecNewsSources` and are also appended to the List component on the main display, `lsNewsSources`, both of which are declared in `NewsReader`.

4.3.7 EditForm

Editing of existing sources is accomplished through the EditForm. In Figure 10, notice that the two TextFields in the rightmost screen shot are filled with the current values of the selected entry: the description and link.

Figure 10: Edit form and two TextFields

`EditForm` is populated inside `NewsReader` with the current description and length through calls to `setDescription()` and `setLink()`.

```
public class EditForm extends Form implements CommandListener
{
  private Command cmBack,
                  cmSave;
  private TextField tfDescription;
  private TextField tfLink;

  protected void setDescription(String description)
  {
    tfDescription.setString(description);
  }

  protected void setLink(String link)
  {
    tfLink.setString(link);
  }
  ...

  public void commandAction(Command c, Displayable s)
  {
    if (c == cmSave)
    {
      ...
      // Create new news source entry (with changes) to replace the
previous
      NewsSourceEntry item  = new
NewsSourceEntry(tfDescription.getString(),
                                                tfLink.getString());

      // Update vector using the previously saved index
      midlet.vecNewsSources.setElementAt(item, index);

      // Update the description in the main news list component
      midlet.lsNewsSources.set(index, tfDescription.getString(), null);
    }
```

```
    ...
  }
}
```

Once the edit form is displayed and the `cmSave` command is chosen, a new `NewsSourceEntry` object is created, using the values from the form fields. This new object is then written into the vector `vecNewsSources`, replacing the previous entry at the same location. In a similar way, `lsNewsSources`, which is the List component inside `NewsReader`, is updated.

4.4 Parsing an XML File

To better understand what we need to extract from a news feed, let's quickly review the RSS feed presented earlier.

```
<rss version="0.91">
<channel>
 <title>E-Commerce Times</title>
  <link>http://www.ecommercetimes.com</link>
  <description>E-Commerce Times: E-Business and Technology Super
Site</description>
  <language>en-us</language>

  <item>
    <title>AMD's Next Move</title>
    <link>http://www.ecommercetimes.com/perl/story/21073.html</link>
    <description>AMD entered 2003 with more to lose than ever before --
including a
     healthy market share in the desktop and notebook chip markets. Now,
the company
     plans a slew of new product releases that could sway its fortunes for
good or
     ill. Will AMD rise above the challenges that face it to emerge
stronger than
     ever, or will it stumble and fall behind?</description>
   </item>

   <item>
   ...
   </item>

   ...

</channel>
</rss>
```

Our concern lies with locating each `<item>` and `</item>` tag and reading the title and description enclosed within. Before looking at how this is accomplished, let's quickly review XML parsing options with J2ME.

4.4.1 Types of XML parsers

The types of parsers available for XML documents fall into one of three categories:

With a ***Pull parser***, the application is in control of the flow of data. Content is retrieved in "chunks" as requested by the application.

With a ***Push parser***, a listener object is registered with the parser and, as the document is parsed, the application is "pushed" data through the interface. One drawback to this approach is that, in most cases, the entire document is read, and parsed content must be stored in the application.

An ***object model parser*** builds a tree-like representation of an XML document. As with a Push parser, the entire document is read.

4.4.2 KXML parser

Given the memory limitations typical of most MIDP devices, a Pull parser is a good option, as data is read only as needed. We'll use KXML 1.2 in our MIDlet to read XML news sources.

Once downloaded, simply extract the Java Application Descriptor(JAR) file (kxml-min.jar) and save to a location in the classpath. It's no more difficult than that to get started with KXML.

4.4.3 Parsing a news source

Within the class NewsParser, parsing an XML document is as simple as opening a network connection, creating an instance of the KXML parser, and calling the `read()` method to begin parsing. As you can see in Chapter 5, we added some messages to inform the user of the actual status between the central steps while setting up the connection to the data feed and then parsing the XML data. This is very important since the task to connect to external networked data sources plus the overhead of parsing potential large data sets may take longer than expected.

```
private void readNews()
{
  ContentConnection conn = null;
  try
  {
    conn = (ContentConnection) Connector.open(source);

    Reader reader = new InputStreamReader(conn.openInputStream());
    XmlParser parser = new XmlParser(reader);
    ParseEvent pe = null;
    ParseEvent event = parser.read();
    String headlineTitle = null, headlineDescription = null;

    while (true)
    {
      event = parser.read();
      if (event.getType() == Xml.START_TAG &&
event.getName().equals("item"))
      {
        // Get more content
        event = parser.read();

        // Loop through the "item", obtaining the title and description
        while (true)
        {
          if (event.getType() == Xml.START_TAG &&
event.getName().equals("title"))
             headlineTitle = parser.read().getText();
          else if (event.getType() == Xml.START_TAG &&
                   event.getName().equals("description"))
             headlineDescription = parser.read().getText();
```

```
            event = parser.read();

            if (event.getType() == Xml.END_TAG &&
               event.getName().equals("item") == true)
            {
              // Send data to Headline list component
              listComponent.newHeadline(headlineTitle, headlineDescription);
              break;
            }
          }
        }

        if(event.getType() == Xml.END_TAG && event.getName().equals("rss"))
          break;
      }
    }
  ...
}
```

Looking over the code, it should be clear there are only a few tags of interest when reading the XML file. First, look for a ***START_TAG*** that has the value of ***item***. Once this tag is found, request data from the parser, looking for both ***title*** and ***description*** tags. Once the title and description are found, send both fields to the List component ***Headline***, where it will be added to the list of titles (headlines) and associated descriptions. This process is repeated until the ***END_TAG,*** where the value of ***rss*** is encountered.

4.5 Deleting News Sources

In addition to adding and editing new sources, the MIDlet wouldn't be complete without an option to delete news sources. Choose the **Delete** option from the menu. There is no confirmation before deleting, so proceed carefully.

Figure 11: Deleting a news source

As shown in the code, and before deleting an entry, a check is made to verify that there is more than one news source. Next, a reference to the current selected item in the list is obtained. Using the `delete()` method of the List component, the entry is removed. The final step is to delete the same entry from the vector that stores `NewsSourceEntry` objects.

```
public void commandAction(Command c, Displayable s)
{
  ...
  else if (c == cmDelete)
  {
    // Can only delete if more than one entry
    if (lsNewsSources.size() > 1)
    {
      // Selected entry to delete
      int i = lsNewsSources.getSelectedIndex();

      // Delete from the list component
      lsNewsSources.delete(i);

      // Delete from the vector that holds NewsSourceEntries objects
      vecNewsSources.removeElementAt(i);
    }
    else
      System.out.println("Unable to delete...");
  }
  ...
}
```

4.6 Writing News Sources to Persistent Storage

Other than hard-coding news-source descriptions and links in the MIDlet, the only option to save news sources is using persistent storage. Inside `NewsReader`, the method `writeNewsList()` is responsible for this task.

```
private void writeNewsList()
{
  try
  {
    // Write data into an internal byte array
    ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();

    // Write Java data types into the above byte array
    DataOutputStream strmDataType = new DataOutputStream(strmBytes);

    byte[] record;
    NewsSourceEntry item;
    String description;
    String link;

    for (int i = 0; i < vecNewsSources.size(); i++)
    {
      // Get entry from vector
      item = (NewsSourceEntry) vecNewsSources.elementAt(i);

      // Build a string with the description, separator and link
      String tmp = item.getDescription() + SEPARATOR_CHAR +
item.getLink();

      // Write the string to rms
      strmDataType.writeUTF(tmp);

      // Clear any buffered data
```

```
            strmDataType.flush();

            // Get stream data into byte array and write record
            record = strmBytes.toByteArray();

            rsNewsSources.addRecord(record, 0, record.length);

            // Toss any data in the internal array so writes
            // starts at beginning (of the internal array)
            strmBytes.reset();
          }

          strmBytes.close();
          strmDataType.close();
        }
        catch (Exception e)
        {
          db(e.toString());
        }
      }
```

Looping through each entry in the vector `vecNewsSources` (Section 4.3.1), acquire a reference to a `NewsSourceObject`. From this object extract the strings that represent description and link; write both strings, separated by a "-" to the record store.

Note: The reason for writing a "-" between the description and link is to allow the fields to be stored in the same record in persistent storage. The "-" is necessary to distinguish between the fields when reading the record from storage at application startup.

4.7 Displaying News Sources

Figure 12 shows the MIDlet main screen with a list of news sources.

Figure 12: List component showing news sources

At this point in the discussion, all the pieces are in place to display the list shown in Figure 12. The only additional point to address is the creation of the Commands and requesting the list to be set as the current displayable. The following shows two small code blocks from `NewsReader`, covering each of these last two steps.

```
public class NewsReader extends MIDlet implements CommandListener
{
   protected List lsNewsSources; // Main list of rss sources
   ...

    // Add commands the main list
    lsNewsSources.addCommand(cmExit);
    lsNewsSources.addCommand(cmAdd);
    lsNewsSources.addCommand(cmEdit);
    lsNewsSources.addCommand(cmDelete);
    lsNewsSources.setCommandListener(this);

   ...

   public void startApp()
  {
    display.setCurrent(lsNewsSources);
  }

  ...
}
```

Note that the commands are not visible on the display. All are shown in a window by pressing the soft key beneath "Options," as shown in Figure 12. Then the available options will be shown:

Figure 13: Commands inside a menu

4.8 Displaying Headlines

Once a news source is selected, the MIDlet will display a list of titles (headlines) available at the news source. Figure 14 shows how this may look. Refer to Figure 7 to see the component for managing and displaying `HeadlineList`.

Figure 14: Headlines from the news source "Humorous Quotes"

The following code shows how to build a list of headlines from a selected news source.

```
public void commandAction(Command c, Displayable s)
{
  // News topic from main list selected
  if (s == lsNewsSources && c == List.SELECT_COMMAND)
  {
    // Create list that will store headline
    // titles and descriptions
    if (lsShowHeadlines == null)
      lsShowHeadlines = new HeadlineList(this);

    // Get a reference to the current (selected) entry in the vector.
    // (the vector contains NewsSourceObjects that hold descriptions
    // and http links to the actual news source)
    NewsSourceEntry item =
      (NewsSourceEntry)
vecNewsSources.elementAt(lsNewsSources.getSelectedIndex());

    // Using the http link, create the list of
    // headlines from the selected rss news source
    lsShowHeadlines.getHeadlines(item.getLink());

    // Show the List of headlines available at the news source
    display.setCurrent(lsShowHeadlines);
  }
...
}
```

The method ***commandAction()*** is the central dispatch for event processing. Once we know an entry on the list was selected, we allocate an instance of ***HeadlineList***, if it does not already exist (Section 4.3.3).

The next step is to get a reference to the current entry selected. Once this is in place, we can get the HTTP link of the news source and begin reading content. If you recall from Section 4.4.3, "Parsing a news source," this method creates a new parser object and starts a new thread to read from the news source.

Once the content has been read, the current displayable is set to show the new list of headlines available. This final step corresponds to the directional arrow shown in Figure 7 going from `NewsReader` to `HeadlineList`.

4.9 Displaying Descriptions

Now that a list of headlines is available, the last step is to show the description of any selected headline along with its title. You can see the entire sequence of events in Figure 7, ending here, at the form `ArticleDisplayForm`.

Figure 15: Description of a selected headline

The following is the event processing code inside `HeadlineList`:

```
public void commandAction(Command c, Displayable s)
{
  // Was 'select' key pressed
  if (c == List.SELECT_COMMAND)
  {
    // Set the title on the form
    String item = (String) vecTitle.elementAt(getSelectedIndex());
    midlet.fmArticle.setArticleTitle(item);

    // Set the description (article) on the form
    item = (String) vecDescription.elementAt(getSelectedIndex());
    midlet.fmArticle.setArticle(item);

    // Display the form with article text
    midlet.display.setCurrent(midlet.fmArticle);
  }
  ...
}
```

There are two key operations here: Populate the title and description of the form (Section 4.3.5), which will display the article, and change the current displayable to the form.

5 Source Code

5.1 NewsReader.java

```
/*--------------------------------------------------
 * NewsReader.java
 *
 * Main MIDlet class
 *--------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import java.util.*;
import javax.microedition.rms.*;

public class NewsReader
  extends MIDlet
  implements CommandListener
{
  protected Display display; // Reference to Display object

  // Main newsreader list and commands
  protected List lsNewsSources; // Main list of rss sources
  private Command cmExit; // Exit
  private Command cmAdd; // Add news source
  private Command cmEdit; // Edit news source
  private Command cmDelete; // Delete news source

  protected static final int MAX_NEWS_SOURCE = 15;
  protected static final int MAX_NEWS_LINK = 75;

  protected static final String[] DEFAULT_NEWS_DESCRIPTIONS =
    {
    "ECommerce Times", "Humorous Quotes",
    "Book Reviews", "Slashdot.org", "Web Developer"};

  protected static final String[] DEFAULT_NEWS_LINKS =
    {
    "http://www.ecommercetimes.com/perl/syndication/rssfull.pl",
    "http://www.quotationspage.com/data/qotd.rss",
    "http://p.moreover.com/cgi-local/page?index_bookreviews+rss",
    "http://slashdot.org/slashdot.rss",
    "http://headlines.internet.com/internetnews/wd-news/news.rss", };

  // The master list of news sources, containing 'NewsSource' objects
  protected Vector vecNewsSources;

  private RecordStore rsNewsSources; // Reference to record store
  private static final String REC_STORE_NEWS = "NewsSources"; // Record
store name

  // What separates the description and link in the record store
  protected static final char SEPARATOR_CHAR = '-';
```

```
  // Each entry in record store contains the source (description),
separator and http link
  byte[] rsData = new byte[MAX_NEWS_SOURCE + 2 + MAX_NEWS_LINK];

  private AddForm fmAddNewsSource; // Form to add news source
  private EditForm fmEditNewsSource; // Form to edit news source
  protected HeadlineList lsShowHeadlines = null; // List of headlines from
rss
  protected ArticleDisplayForm fmArticle; // Form to display selected
headline
  protected InfoForm fmInfo; // Form to show messages

  public NewsReader()
  {

    display = Display.getDisplay(this);

    // Create the commands for main news list
    cmExit = new Command("Exit", Command.EXIT, 1);
    cmAdd = new Command("Add", Command.SCREEN, 2);
    cmEdit = new Command("Edit", Command.SCREEN, 3);
    cmDelete = new Command("Delete", Command.SCREEN, 4);

    // Create main news list and commands........
    vecNewsSources = new Vector();
    // Open/create the record stores
    rsNewsSources = openRecStore(REC_STORE_NEWS);

    // Read rms news sources in the vector
    readNewsList();

    // Close the record store
    closeRecStore(rsNewsSources);

    // Create news source list component
    lsNewsSources = new List("News on the Go", List.IMPLICIT);

    // Add entries to the news source list component (from vector)
    buildNewsList();

    // Add commands the main list
    lsNewsSources.addCommand(cmExit);
    lsNewsSources.addCommand(cmAdd);
    lsNewsSources.addCommand(cmEdit);
    lsNewsSources.addCommand(cmDelete);
    lsNewsSources.setCommandListener(this);

    // Create form to add a news source
    fmAddNewsSource = new AddForm(this);

    // Create form to edit a news source
    fmEditNewsSource = new EditForm(this);

    // Create canvas that will display articles
    fmArticle = new ArticleDisplayForm(this);

    // Create the InfoForm instance
    fmInfo = new InfoForm(this);
```

```
}

/*--------------------------------------------------
 * Startup
 *--------------------------------------------------*/
public void startApp()
{
  display.setCurrent(lsNewsSources);
}

/*--------------------------------------------------
 * Pause MIDlet
 *--------------------------------------------------*/
public void pauseApp()
{
}

/*--------------------------------------------------
 * Exit MIDlet
 *--------------------------------------------------*/
public void destroyApp(boolean unconditional)
{
  // Delete the old rms
  deleteRecStore(REC_STORE_NEWS);

  // Open the rms
  rsNewsSources = openRecStore(REC_STORE_NEWS);

  // Write the vector to rms and close
  writeNewsList();
  closeRecStore(rsNewsSources);
}

/*--------------------------------------------------
 * Event processing
 *--------------------------------------------------*/
public void commandAction(Command c, Displayable s)
{
  // News topic from main list selected
  if (s == lsNewsSources && c == List.SELECT_COMMAND)
  {
    // Create list that will store headline
    // titles and descriptions
    if (lsShowHeadlines == null)
    {
 lsShowHeadlines = new HeadlineList(this);

 // Get a reference to the current (selected) entry in the vector.
 // (the vector contains NewsSourceObjects that hold descriptions
 // and http links to the actual news source)
    }
    NewsSourceEntry item = (NewsSourceEntry) vecNewsSources.elementAt(
 lsNewsSources.getSelectedIndex());

    // Using the http link, create the list of
    // headlines from the selected rss news source
    lsShowHeadlines.getHeadlines(item.getLink());
```

```
    // Update the title on the headline display
    // to match the selected entry
    lsShowHeadlines.setTitle(item.getDescription());

    // Show the List of headlines available at the news source
    display.setCurrent(lsShowHeadlines);
  }
  else if (c == cmAdd)
  {
    // Show the "add news source form"
    display.setCurrent(fmAddNewsSource);
  }
  else if (c == cmEdit)
  {
    // Edit the current entry by showing the
    // 'edit new source form'

    // Get the description and link from the current
    // list entry. This is stored in the vector which
    // holds objects of the type 'newsSourceEntry'
    if (lsNewsSources.size() > 0)
    {
int i = lsNewsSources.getSelectedIndex();
NewsSourceEntry tmp = (NewsSourceEntry) vecNewsSources.elementAt(i);
String description = tmp.getDescription();
String link = tmp.getLink();

// Set the text fields on the form
fmEditNewsSource.setDescription(description);
fmEditNewsSource.setLink(link);
fmEditNewsSource.setIndex(lsNewsSources.getSelectedIndex());

// Show the "edit news source form"
display.setCurrent(fmEditNewsSource);
    }
  }
  else if (c == cmDelete)
  {
    // Can only delete if more than one entry
    if (lsNewsSources.size() > 1)
    {
// Selected entry to delete
int i = lsNewsSources.getSelectedIndex();

// Delete from the list component
lsNewsSources.delete(i);

// Delete from the vector that holds NewsSourceEntries objects
vecNewsSources.removeElementAt(i);
    }
    else
    {
fmInfo.setMessage("Unable to delete...");   // @to do: error screen
    }
  }
  else if (c == cmExit)
  {
```

```
    destroyApp(false);
    notifyDestroyed();
  }
}

/*-------------------------------------------------
 * Open a record store
 *-------------------------------------------------*/
private RecordStore openRecStore(String name)
{
  try
  {
    // Open the Record Store, creating it if necessary
    return RecordStore.openRecordStore(name, true);
  }
  catch (Exception e)
  {
    db(e.toString());
    return null;
  }
}

/*-------------------------------------------------
 * Close a record store
 *-------------------------------------------------*/
private void closeRecStore(RecordStore rs)
{
  try
  {
    rs.closeRecordStore();
  }
  catch (Exception e)
  {
    db(e.toString());
  }
}

/*-------------------------------------------------
 * Delete a record store
 *-------------------------------------------------*/
private void deleteRecStore(String name)
{
  try
  {
    RecordStore.deleteRecordStore(name);
  }
  catch (Exception e)
  {
    db(e.toString());
  }
}

/*-------------------------------------------------------
 * Create news source list (vector) by reading entries
 * from rms and storing as 'NewsSourceEntries' in vector
 *-------------------------------------------------------*/
private void readNewsList()
{
```

```
 try
 {
   // Add a default to the vector, if there are no entries...
   if (rsNewsSources.getNumRecords() < 1)
   {
int i = DEFAULT_NEWS_DESCRIPTIONS.length;
for (int j = 0; j < i; j++)
{
  add2Vector(DEFAULT_NEWS_DESCRIPTIONS[j], DEFAULT_NEWS_LINKS[j]);
}
return;
   }

   // Read from the specified byte array
   ByteArrayInputStream strmBytes = new ByteArrayInputStream(rsData);

   // Read Java data types from the above byte array
   DataInputStream strmDataType = new DataInputStream(strmBytes);

   NewsSourceEntry item;
   String tmp_rms, tmp_parse;
   int offset;
   for (int i = 1; i <= rsNewsSources.getNumRecords(); i++)
   {
// Get data into byte array
rsNewsSources.getRecord(i, rsData, 0);

tmp_rms = strmDataType.readUTF();

// Parse out the description and link from the string,
// based on SEPARATOR_CHAR between them
offset = tmp_rms.indexOf(SEPARATOR_CHAR); // Offset of the separator

// Add the description and link knowing the location of separator
item = new NewsSourceEntry(tmp_rms.substring(0, offset),
            tmp_rms.substring(offset + 1, tmp_rms.length()));

// Add entry to the vector
vecNewsSources.addElement(item);

// Reset so read starts at beginning of array
strmBytes.reset();
   }

   strmBytes.close();
   strmDataType.close();

 }
 catch (Exception e)
 {
   db(e.toString());
 }
}

/*------------------------------------------------------
 * Add an entry to the vector
 *------------------------------------------------------*/
protected void add2Vector(String description, String link)
```

```
  {
    NewsSourceEntry item = new NewsSourceEntry(description,
link.toLowerCase());

    // Insert into vector at the end
    vecNewsSources.insertElementAt(item, vecNewsSources.size());
  }

  /*-------------------------------------------------------
   * Create the List component for the main display.
   * This is built from the news source vector
   *-------------------------------------------------------*/
  private void buildNewsList()
  {
    NewsSourceEntry item;
    for (int i = 0; i < vecNewsSources.size(); i++)
    {
      // Get entry from vector
      item = (NewsSourceEntry) vecNewsSources.elementAt(i);

      // Add only the description to the list
      lsNewsSources.append(item.getDescription(), null);
    }
  }

  /*-------------------------------------------------------
   * Write the news source list (vector) into rms
   *-------------------------------------------------------*/
  private void writeNewsList()
  {
    try
    {
      // Write data into an internal byte array
      ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();

      // Write Java data types into the above byte array
      DataOutputStream strmDataType = new DataOutputStream(strmBytes);

      byte[] record;
      NewsSourceEntry item;
      String description;
      String link;

      for (int i = 0; i < vecNewsSources.size(); i++)
      {
   // Get entry from vector
   item = (NewsSourceEntry) vecNewsSources.elementAt(i);

   // Build a string with the description, separator and link
   String tmp = item.getDescription() + SEPARATOR_CHAR + item.getLink();

   // Write the string to rms
   strmDataType.writeUTF(tmp);

   // Clear any buffered data
   strmDataType.flush();

   // Get stream data into byte array and write record
```

```
            record = strmBytes.toByteArray();
            rsNewsSources.addRecord(record, 0, record.length);

            // Toss any data in the internal array so writes
            // starts at beginning (of the internal array)
            strmBytes.reset();
               }
               strmBytes.close();
               strmDataType.close();
             }
             catch (Exception e)
             {
               db(e.toString());
             }
          }

          /*------------------------------------------------
           * Message to console for debug/errors
           *------------------------------------------------*/
          private void db(String str)
          {
            fmInfo.setMessage("Msg: " + str);    // @to do: error screen
          }
        }
```

5.2 HeadlineList.java

```
        /*------------------------------------------------
         * HeadlineList.java
         *
         * Extend List component to store title and
         * description for all articles from news source.
         *
         * There are two vectors used - One to store the
         * article title, one for the article description.
         *------------------------------------------------*/
        import java.util.*;
        import javax.microedition.lcdui.*;

        public class HeadlineList
          extends List
          implements CommandListener
        {
          private Command cmBack;
          private NewsReader midlet;
          private Vector vecTitle, vecDescription;

          public HeadlineList(NewsReader midlet)
          {
            // Call the List constructor
            super("Headlines", List.IMPLICIT);

            // Save reference to MIDlet so we can access display manager
            this.midlet = midlet;

            // Go back
            cmBack = new Command("Back", Command.BACK, 1);
```

```
    // Add to form and listen for events
    addCommand(cmBack);
    setCommandListener(this);

    vecTitle = new Vector();
    vecDescription = new Vector();
  }

  /*------------------------------------------------
    * Deletes all entries in vecTitle and vecDescription
    * and also any current content being shown.
    *------------------------------------------------*/

  protected void clearHeadline()
  {
    // clear out any old contents
    vecTitle.removeAllElements() ;
    vecDescription.removeAllElements();

    // clear the list if it is not empty
    while (this.size() > 0)
    this.delete(0);
  }

  /*------------------------------------------------
   * Get the headlines from the news source passed in.
   * Accomplished by starting thread in parser.
   *------------------------------------------------*/
  protected void getHeadlines(String source)
  {
    // Clear out any old contents
    vecTitle.removeAllElements();
    vecDescription.removeAllElements();

    // Clear out the list if not empty
    while (this.size() > 0)
    {
      this.delete(0);
    }

    /* Create and start a NewsParser thread */
    NewsParser tmp = new NewsParser(midlet, source, midlet.fmInfo, this);
    tmp.start();
  }

  /*------------------------------------------------
   * Called inside NewsParser class with title
   * and description from the rss new source
   *------------------------------------------------*/
  public void newHeadline(String title, String description)
  {
    // Add the title and description to vector
    vecTitle.addElement(title);
    vecDescription.addElement(description);

    // Add the current title to the list component
    this.append(title, null);
```

```
  }

  /*------------------------------------------------
   * Event processing
   *------------------------------------------------*/
  public void commandAction(Command c, Displayable s)
  {
    // Was 'select' key pressed
    if (c == List.SELECT_COMMAND)
    {
      // Set the title on the form
      String item = (String) vecTitle.elementAt(getSelectedIndex());
      midlet.fmArticle.setArticleTitle(item);

      // Set the description (article) on the form
      item = (String) vecDescription.elementAt(getSelectedIndex());
      midlet.fmArticle.setArticle(item);

      // Display the form with article text
      midlet.display.setCurrent(midlet.fmArticle);
    }
    else
    {

      // Any other command, go back to the main news source list...
      midlet.display.setCurrent(midlet.lsNewsSources);
    }
  }
}
```

5.3 NewsParser.java

```
/*------------------------------------------------
 * NewsParser.java
 *
 * Parse XML documents
 *------------------------------------------------*/
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import org.kxml.*;
import org.kxml.parser.*;
import java.io.*;

public class NewsParser
  implements Runnable
{
  private String source; // News source to read from
  private InfoForm fmInfo; // List component of headlines
  private HeadlineList listComponent; // List component of headlines
  private NewsReader midlet;

  public NewsParser(NewsReader midlet, String source, InfoForm fmInfo,
HeadlineList listComponent)
  {
    this.midlet = midlet;
    this.source = source;
```

```
    this.fmInfo = fmInfo;
    this.listComponent = listComponent;
  }

  /*------------------------------------------------
   * Start the parser routine
   *------------------------------------------------*/
  public void run()
  {
    parseNewsFeed();
  }

  /*------------------------------------------------
   * Start a thread
   *------------------------------------------------*/
  public void start()
  {
    Thread thread = new Thread(this);

    try
    {
      thread.start();
    }
    catch (Exception e)
    {
      fmInfo.setMessage("Fatal: Unable to start the NewsReader.");
    }
  }

  /*------------------------------------------------
   * Parse newsfeed (XML document)
   * This parser needs to be adapted for different versions
   * of RSS/RDF.
   *------------------------------------------------*/
  private void parseNewsFeed()
  {
    ContentConnection conn = null;

    try
    {
      fmInfo.setMessage("trying to connect ...");

      conn = (ContentConnection) Connector.open(source);

      if (conn == null)
      {
        fmInfo.setMessage("no connection to \n" + source);
        return;
      }
      else
      {
        fmInfo.setMessage("reading ...");
      }

      Reader reader = new InputStreamReader(conn.openInputStream());
      fmInfo.setMessage("parsing ...");

      XmlParser parser = new XmlParser(reader);
```

```
      fmInfo.setMessage("done parsing ...");

      ParseEvent pe = null;
      ParseEvent event = parser.read();
      String headlineTitle = null, headlineDescription = null;
      fmInfo.setMessage("building news list ...");

      midlet.display.setCurrent(listComponent);

      while (true)
      {
   event = parser.read();
         if (event.getType() == Xml.START_TAG &&
event.getName().equals("item"))
   {
           // get more content
           event = parser.read();

           // loop through the "item", obtaining the title, link and
description
          while (true)
          {
            if (event.getType() == Xml.START_TAG &&
                        event.getName().equals("title"))
           {
             headlineTitle = parser.read().getText();
           }
           else if (event.getType() == Xml.START_TAG &&
          event.getName().equals("description"))
         {
           headlineDescription = parser.read().getText();
         }
         event = parser.read();

         if (event.getType() == Xml.END_TAG &&
             event.getName().equals("item") == true)
         {
           // Send data to Headline list component
           listComponent.newHeadline(headlineTitle, headlineDescription);
           break;
         }
       }
     }

     if (event.getType() == Xml.END_TAG && event.getName().equals("rss"))
     {
       break;
     }
  }
 }
 catch (Exception e)
 {
   fmInfo.setMessage("Error parsing XML!" + e.toString());
 }
 finally
 {
    try
      {
```

```
            if (conn != null)
            {
              conn.close();
            }
        }
        catch (IOException ignored)
        {
          fmInfo.setMessage("IO exception" + ignored.toString());
        }
      }
    }
  }
```

5.4 ArticleDisplayForm.java

```
/*-------------------------------------------------
 * ArticleDisplayForm.java
 *
 * Form to display articles (descriptions)
 *-------------------------------------------------*/
import javax.microedition.lcdui.*;

public class ArticleDisplayForm
  extends Form
  implements CommandListener
{
  private NewsReader midlet;
  private Command cmBack;
  private StringItem siArticle; // Component to display article

  public ArticleDisplayForm(NewsReader midlet)
  {
    // Call the Form constructor
    super("");

    // Save reference to MIDlet so we can access display manager
    this.midlet = midlet;

    // Add to form and listen for events
    cmBack = new Command("Back", Command.BACK, 1);
    addCommand(cmBack);
    setCommandListener(this);

    // The stringitem is the article to display
    siArticle = new StringItem("", "");
    append(siArticle);
  }

  /*-------------------------------------------------
   * Set the StringItem to the article description
   *-------------------------------------------------*/
  protected void setArticle(String str)
  {
    siArticle.setText(str);
  }

  /*-------------------------------------------------
```

```
     * Set title of form to article title (description)
     *------------------------------------------------*/
    protected void setArticleTitle(String str)
    {
      setTitle(str);
    }

    /*------------------------------------------------
     * Event processing
     *------------------------------------------------*/
    public void commandAction(Command c, Displayable s)
    {
      // Go back to the headline list of the current news source
      midlet.display.setCurrent(midlet.lsShowHeadlines);
    }
  }
```

5.5 AddForm.java

```
/*------------------------------------------------
 * AddForm.java
 *
 * Extend Form component to add news sources
 *------------------------------------------------*/
import javax.microedition.lcdui.*;

public class AddForm
  extends Form
  implements CommandListener
{
  private Command cmBack, cmSave;
  protected TextField tfDescription;
  protected TextField tfLink;
  private NewsReader midlet;

  public AddForm(NewsReader midlet)
  {
    // Call the Form constructor
    super("Add News Source");

    // Save reference to MIDlet so we can access display manager
    this.midlet = midlet;

    // Commands
    cmSave = new Command("Save", Command.SCREEN, 1);
    cmBack = new Command("Back", Command.BACK, 2);

    // Textfields for news source description and link
    tfDescription = new TextField("Description", null,
midlet.MAX_NEWS_SOURCE,
              TextField.ANY);
    tfLink = new TextField("Link", null, midlet.MAX_NEWS_LINK,
TextField.ANY);

    // Add stuff to form and listen for events
    addCommand(cmSave);
    addCommand(cmBack);
```

```
      append(tfDescription);
      append(tfLink);
      setCommandListener(this);
    }

    /*------------------------------------------------
     * Event processing
     *------------------------------------------------*/
    public void commandAction(Command c, Displayable s)
    {
      if (c == cmSave)
      {
        // Replace any occurrence of the separator character with an empty
space
        // This is necessary so we can properly parse entries stored in the
rms
        tfDescription.setString(tfDescription.getString().replace(midlet.
   SEPARATOR_CHAR, ' '));

        midlet.add2Vector(tfDescription.getString(), tfLink.getString());
        midlet.lsNewsSources.append(tfDescription.getString(), null);
      }

      // Display the main news source list...
      midlet.display.setCurrent(midlet.lsNewsSources);
    }
  }
```

5.6 EditForm.java

```
/*------------------------------------------------
 * EditForm.java
 *
 * Extend Form component to edit news sources
 *------------------------------------------------*/
import javax.microedition.lcdui.*;

public class EditForm
  extends Form
  implements CommandListener
{
  private Command cmBack,
    cmSave;
  private TextField tfDescription;
  private TextField tfLink;
  private NewsReader midlet;

  // When editing an entry, the midlet makes note of which
  // entry in the list (and vector) are being changed
  private int index;

  public EditForm(NewsReader midlet)
  {
    // Call the Form constructor
    super("Edit News Source");

    // Save reference to MIDlet so we can access display manager
```

```
    this.midlet = midlet;

    // Commands
    cmSave = new Command("Save", Command.SCREEN, 1);
    cmBack = new Command("Back", Command.BACK, 2);

    // Textfields for news source description and link
    tfDescription = new TextField(null, null, midlet.MAX_NEWS_SOURCE,
              TextField.ANY);
    tfLink = new TextField(null, null, midlet.MAX_NEWS_LINK,
TextField.ANY);

    // Add stuff to form and listen for events
    addCommand(cmSave);
    addCommand(cmBack);
    append(tfDescription);
    append(tfLink);
    setCommandListener(this);
  }

  /*------------------------------------------------
   * Set the description field
   *------------------------------------------------*/
  protected void setDescription(String description)
  {
    tfDescription.setString(description);
  }

  /*------------------------------------------------
   * Set the link field
   *------------------------------------------------*/
  protected void setLink(String link)
  {
    tfLink.setString(link);
  }

  /*------------------------------------------------
   * Set the index field of the entry being edited
   *------------------------------------------------*/
  protected void setIndex(int index)
  {
    this.index = index;
  }

  /*------------------------------------------------
   * Event processing
   *------------------------------------------------*/
  public void commandAction(Command c, Displayable s)
  {
    if (c == cmSave)
    {
      // Replace any occurrence of the separator character with an empty
space
      // This is necessary so we can properly parse entries stored in the
rms
      tfDescription.setString(tfDescription.getString().replace(midlet.
   SEPARATOR_CHAR, ' '));
```

```
      // Create new news source entry (with changes) to replace the
previous
      NewsSourceEntry item = new NewsSourceEntry(tfDescription.getString(),
                  tfLink.getString());

      // Update vector using the previously saved index
      midlet.vecNewsSources.setElementAt(item, index);

      // Update the description in the main news list component
      midlet.lsNewsSources.set(index, tfDescription.getString(), null);
    }

    // Display the main news source list...
    midlet.display.setCurrent(midlet.lsNewsSources);
  }
}
```

5.7 InfoForm

```
/*-------------------------------------------------
 * InfoForm.java
 *
 * Extend Form component to show system/info messages
 *-------------------------------------------------*/
import javax.microedition.lcdui.*;

public class InfoForm
  extends Form
  implements CommandListener
{
  private Command cmBack;
  private StringItem Message;
  private NewsReader midlet;

  public InfoForm(NewsReader midlet)
  {
    // Call the Form constructor
    super("Information");

    // Save reference to MIDlet so we can access display manager
    this.midlet = midlet;

    // Commands
    cmBack = new Command("Back", Command.BACK, 2);

    // Textfields for news source description and link
    Message = new StringItem ("Please wait", "");

    // Add stuff to form and listen for events
    addCommand(cmBack);
    append(Message);

    setCommandListener(this);
  }

  /*-------------------------------------------------
   * Event processing
```

```
     *-------------------------------------------------*/
    public void commandAction(Command c, Displayable s)
    {
      if (c == cmBack)
      {
          // do some cleanup if needed ...
      }

      // Display the main news source list...
      midlet.display.setCurrent(midlet.lsNewsSources);
    }

    /*-------------------------------------------------
     * setMessage
     * Shows a new message in the Form
     *-------------------------------------------------*/
    public void setMessage(String Message)
    {
      this.Message.setText(Message);  // set the message text
      midlet.display.setCurrent(this);  // activate this Form regardless
    }
}
```

5.8 NewsSourceEntry

```
/*-------------------------------------------------
 * NewsSourceEntry.java
 *
 * Each news source has a description and http link
 *-------------------------------------------------*/
public class NewsSourceEntry
{
  private String description, link;

  public NewsSourceEntry(String description, String link)
  {
    this.description = description;
    this.link = link;
  }

  /*-------------------------------------------------
   * Get description field
   *-------------------------------------------------*/
  public String getDescription()
  {
    return description;
  }

  /*-------------------------------------------------
   * Set description field
   *-------------------------------------------------*/
  public void setDescription(String description)
  {
    this.description = description;
  }

  /*-------------------------------------------------
```

```
     * Get linkfield
     *------------------------------------------------*/
    public String getLink()
    {
      return link;
    }

    /*------------------------------------------------
     * Set description field
     *------------------------------------------------*/
    public void setLink(String link)
    {
      this.link = link;
    }
  }
```

6 Summary

This document provided an in-depth look at RSS news feeds. Given the relative simplicity of a news feed XML file, it is quite simple to create an application to take advantage of RSS. This article demonstrated one approach to parse and display news feed content from within a MIDlet.

7 Terms and Abbreviations

Term or abbreviation	Meaning
RDF	Resource Description Framework, which is used to define RSS data.
RSS	Rich Site Summary.
Tag	Single XML element containing information.