Article Management System
using
Objective-C on the NeXT computer

by
Kris Kamisetty
Arizona State University

1. Introduction

It is attempted to describe in this report, the development cycle that has been gone through for developing a fully functional "Article Management System". This is a system envisaged to aid a typical researcher in organizing the personal collection of research papers/articles that he/she has accumulated over time. It often becomes a tedious task to rummage through the entire large collection for choosing the papers of interest, for example, dealing with a particular subject area or written by the same author(s). A system with a good user interface to take care of all the organization and search procedures can come in very handy. This project has been particularly chosen by this author as he finds himself exactly in the position as described above.

2. Design Process

The design process adopted for developing the "Article Management System" (AMS) is quite typical of applications designed on NeXT using Interface Builder.

The very first step has been to create a user interface for AMS and identify the objects i.e. windows, panels, buttons, menu items etc. that will exist in the application. Then the data that needs to be maintained for keeping track of the various research articles/papers has been enumerated.

Then an appropriate class (**ArticleFile**) has been created in the class hierarchy that will manage this data through the execution of appropriate methods. The methods have been determined next and the appropriate connections that can be defined at the time of compiling the application itself are made between the various user interface objects existing in the

system at the start-up time of the application. Once the connections have been made, what is left to be done is the development of objective-C code to "encapsulate" intelligence into the buttons and the windows of the AMS user interface.

The application is then tested for the functionality and elegance and it is fine-tuned again until it meets the programming guidelines and user interface styles normally adopted by NeXT programmers.

It has been this student's attempt to make the class developed for this application (**ArticleFile** class) as method rich and job-independent as possible so that another developer can safely subclass it and customize it to suit his/her needs. In fact, it so turns out that the interested developer can subclass **ArticleFile** and override just one method (i.e. **collectInfo:sender**) to be able to customize it according to his/her needs. There is very little extra code required of the potential user of this class.

The manner in which various user input windows are sequenced is very flexible. The developer can specify the order through **collectInfo:sender** method.

AMS also implements a facility to input research papers with multiple authors and multiple key words. One restriction with using AMS is that the user should make sure that he/she assigns unique numbers to his/her articles. The application does not pretend to make any efforts to take care of that. Caution should be exercised in this regard because entering an already existing article number can overwrite and hence destroy the existing article information.

3. Implementation

The main implementation strategy is to provide windows to receive information from the user regarding the research articles. As this information is received, it is stored in a file, in the directory specified by the user. Once the user completes the input process, the file creation process is complete. Thus, a number of articles can be created in the user-specified directory. Then, the user can choose from the menu to index all the files in the directory. The indexing utility is provided by NeXT as a command line utility. This is exactly the same utility that the Librarian application on NeXT uses among other C functions and command line utilities.

Once an index is created for the directory specified by the user, the user can then choose "Search..." from the menu. A panel asks for the keyword to be searched and passes on this string to another application called Librarian on NeXT through a UNIX **remote procedure call.** This is accomplished on NeXT through utilizing the Speaker and Listener classes for the AMS application.

The indexing mechanism and the remote messaging process have been so critical to the success of this application that they are described next in two subsections followed by a subsection on user interface. Later, a free-hand diagram attempts to put the entire process in perspective.

Indexing Mechanism

The NeXT text indexing philosophy relies heavily on how the salient words are extracted from a document. When a document is indexed, its type is identified by an identification algorithm. A keyword extraction program specific to the document type then reads the document file, generating a weighted list of keywords extracted from the document text. Weighted references are then generated from the list, associating the name of the document with each of the keywords through the index structures. The keyword extraction used on NeXT weights terms according to the frequency of use in the source text, **relative to their frequency of use in some base domain**.

Indexing can be accomplished by calling a command line utility on NeXT called **ixBuild**. ixBuild visits all of the files in the directory tree rooted at a master directory, extracting keywords and a description from each file visited. With this information, it creates or updates an index located in a subdirectory called **.index,** which resides in the master directory. This utility has a number of options for controlling the way in which the index is built.

The NeXT text indexing facilities impose the restriction that all of the files to be indexed in a common index structure must reside in a single directory tree originating at some master directory that can be written to, at the time of the index construction. This is because of NeXT's close ties with the UNIX file system.

The indexing process creates the following five files: index.L, index.D, index, index.Registry.L and index.Registry.D. The first two are leaf and directory files respectively for a hashed database containing 'keyword to

file name' associations. The database associates each keyword in the index with a list of pointers. A pointer consists of a **weight** and an **identifier**.

The weight is an arbitrary normalized number that indicates the relative importance of the keyword to the file referenced by the pointer. The identifier is an offset into the .index file and points to a block of information called a FileCell that describes the referenced file. The index.Registry.L and index.Registry.D files keep track of the date of creation of the FileCell associated with each file.

Remote Messaging of the Librarian Application

The Appkit supplied by NeXT provides programmers with a pair of wonderful classes that let the applications effortlessly communicate with each other: Speaker and Listener. Since all the NeXTStep applications can talk to each other, each application can concentrate on doing one class of things very well, and depend on other applications to provide other necessary services. This is exactly what this student has attempted to do for developing AMS application.

Librarian is a bundled application with the NeXT. It uses the indexing principle as described above. It has the user interface implemented to display the references that it finds for the particular keyword. Thus, AMS application messages its own Speaker (which is created by default at the start-up time) to send a method that tells the Librarian to search for the keyword (supplied as an argument to the method) to the Listener of Librarian. This process of remote messaging is facilitated by the fact that potentially **all** applications designed on NeXT using Interface Builder **have** Speaker and Listener instantiated for them by their **new** methods.

Some of the benefits of remote messaging as found on NeXT are as listed below:

1. Users write their own front ends, with their own customized code, and hook it into any other application e.g. AMS without any difficulty.

2. No re-compilation is required by this scheme.

3. The programmer does not have to give the source code away.

4. Applications can enforce rigorous validity checking on the data coming

from the user's front end.

5. By providing a public Listener, and documenting the messages it expects, the programmer can help extend the usefulness, and ultimately the indispensability of the application.

User Interface

In all, AMS application uses 7 windows and countless message and warning panels. Among these windows, 4 have been used to elicit the information regarding the article. Two windows have been used to get the information regarding the article number and directory and one another is used to get the search expression from the user for his/her search needs.

Many warning and messages panels have been designed to appear when needed. It has been attempted to reduce the amount of information that appears on any given window to a bare minimum so that the user can feel comfortable with what is presented.

Operation

When the user double-clicks the AMS application, the user is presented with a main menu. The main menu consists of 5 basic features. They are:

1. Add Article
2. Update Article
3. Index Articles
4. Search
5. Delete Article

Panels and windows that appear as a result of the choice made by the user guide the user through the process of the appropriate operation. A file of the same name as the article number is created when the user finishes feeding all the information. The directory in which the article is stored depends upon the user input. Once the article is stored, it can be updated or deleted any time. When the user selects "Update Article", Edit application will open up the article selected by the user. Similarly, the user can delete any article in any directory by choosing "Delete Article" and giving the appropriate input. "Index Articles" can be chosen to create or update all the articles in any master directory.

At the same time, the user can choose not to index for the time being

and search for the words with the help of a previously created index.

"Search" presents a window wherein the user can specify the word(s) to be searched for. Then the AMS application reads this string and passes it over to Librarian application bundled with NeXT. Then, Librarian launches itself and displays all the files that reference the specified word(s).

4. Future Potential

There are a few things that could help AMS application serve the needs of the users even better. Some of these are indicated below:

1. ScrollView can be used to let the user type in the abstract for the research paper/article in question. This will be added on to the information already stored under an appropriate heading.

2. A "Pull Down List" can be implemented instead of a column of buttons in the Directory panel for a more elegant interface.

3. A radio button panel can also substitute the column of buttons in the directory panels because the choices are mutually exclusive anyway.

4. More flexibility with managing the various articles can be achieved by using the StreamTable class or List class to store the articles and update them as needed.

5. AMS could serve as a model front end for ASU campus libraries' catalog search facility.

6. The numbers associated with the deleted articles could be reclaimed and reused.

5. Conclusion

AMS application has been developed in a short time because of the powerful interface capabilities and the open, powerful operating system environment. The seed for this project is the "Article Tracking System"

developed by Dr. Beaumariage in SmallTalk environment. AMS has been developed by this author because he finds himself at the threshold of full time research and it already has become impossible to keep track of all the articles that have been collected. This author hopes to use AMS extensively and better it over time.

This effort has been a really an enjoyable and a useful one. This student is truly grateful to Dr. Beaumariage for giving a 'go-ahead' to use Objective-C on NeXT for this project while the rest of the class is using SmallTalk for their projects.

Appendix A
Class Specifications

ArticleFile

INHERITS FROM	Responder : Object
REQUIRES HEADER FILES	appkit.h,stdio.h, strings.h,ArticleFile.h
DEFINED IN	AMS Tech. Docs, Version 1.0

CLASS DESCRIPTION

The ArticleFile class helps in organizing the any kind of documents produced using simple ASCII editors though this class is primarily designed to help keep track of any personal collection of research papers/articles.

ArticleFile provides various methods to collect information from the user regarding the research papers/articles. A later version of this class will have the facility for the user to type in the abstract in a ScrollView. There are methods to add author, keyword, other article information and abstract information. There are methods implemented to help the user specify the order in which the windows have to be sequenced. There are methods to update, index and delete the articles. There is also a method that gets the search expression from the user and messages Librarian to search for it.

There are some other miscellaneous messages to warn the user about the features not implemented etc. Only the significant instance methods created by the developer of this class are described here. The rest of them are outlet initialization methods generated by Interface Builder.

A developer who would like to construct his/her own application based on ArticleFile class will just have to override collectInfo:sender method.

INSTANCE VARIABLES

Inherited from Object Class isa;

Inherited from Responder id nextResponder;

Declared in ArticleFile id place;
id pages;
id date;
id title;

id keyWord;
id dueOn;
id country;
id num;
id comments;
id vol;
id lname;
id journal;
id chkout;
id fname;
id wordField;
id absWindow;
id authWindow;
id articleWindow;
id keyWindow;
id numberWindow;
id directoryWindow;
id articleNumber;
id directory;

CLASS METHODS

NONE

INSTANCE METHODS

- addKeyWordAndClose:sender

This method reads the string values from the Form field pointed to by keyWord outlet, stores the value in a file specified by the user in

collectInfo:sender method and then closes the window pointed to by keyWindow outlet. It also stops the modal session initiated by collectInfo:sender method before closing the window.

- addMoreKeyWords:sender

This method does not close the window pointed to by keyWindow after reading the string value from the Form field pointed to by keyWord outlet and writing this value to a file specified by the user in the collectInfo:sender method. Instead, it selects the text in the Form field and waits for further user events.

- addArticleInfoAndClose:sender

It reads the string values from the form fields pointed to by the following outlets: title, journal, vol, num, date, pages, place, country, comments, chkout, dueOn. It writes these strings to a file specified by the user in the collectInfo:sender method. It stops the modal session initiated by the collectInfo:sender method. It then closes the window pointed to by articleWindow outlet.

- addAbstractAndClose:sender

At this time, this method is not implemented. So, it just stops the modal session initiated by collectInfo:sender method and closes the window pointed to by absWindow outlet.

- addAuthorAndClose:sender

This method reads the string values from the Form fields pointed to by lname and fname outlets, writes them to a file specified by the user in the collectInfo:sender method and then closes the window pointed to by authWindow.

- addMoreAuthors:sender

This method is just like addAuthorAndClose: method except that it does not close the window pointed to by authWindow after reading the string values and storing them in a file specified by the user in collectInfo:sender method.

- warnUser:sender

This method just puts up an alert panel informing that the user-selected feature is not available at this time.

- skipAbstract:sender

This method stops the modal session for the window pointed to by absWindow, closes it and then puts up an alert panel indicating that the abstract portion has been skipped.

- fileInfoOpen:(int)item

This method is provided basically to help the user get the flexibility to choose the sequence in which the windows need to be sequenced. This is done so as to avoid hardcoding the sequence in which the windows associated with article information input need to be run. There are 4 kinds of windows possible as specified by AUTHORS, KEYWORDS, ARTICLE and ABSTRACT.

- clearAllFields

This clears all the Form fields pointed to by the following outlets: articleNumber, lname, fname, title, journal, vol, num, date, pages, place, country, comments, chkout, dueOn, keyWord.

- collectInfo:sender

This is the main method where the programmer can define the way in which all the methods should be fired. It basically reads the string value from the Form field pointed to by directory and stores it and closes the window pointed to by directoryWindow. It then opens the file with the name same as the article number specified by the user and closes the window pointed to by numberWindow. Later, it messages NXApp (the main Application object) to run modal sessions for the four windows. Then, it closes the file and clears all the fields.

- deleteInfo:sender

This method asks for the directory and the article number and then deletes the associated file from the directory.

- indexInfo:sender

This method indexes the articles in the directory specified by calling a command line utility called ixBuild and comes back with a message when it is done.

- updateInfo:sender

This method gets the article number and the directory from the user, closes the respective windows and then launches Edit application on NeXT and loads the file associated with the article number specified.

- searchWords:sender

This method reads the string value from a Form field pointed to by wordField, messages Librarian on NeXT to launch itself and passes the search expression to Librarian so that Librarian can display the appropriate references. In case the messaging fails, alert panels will come up explaining the situation.

Appendix B
Objective-C Program code

INTERFACE FILE (ArticleFile.h)

```
/* Programmed by Krishnaprasad Kamisetty */

#import <appkit/Responder.h>
#import <stdio.h>
#import <appkit/appkit.h>
#import <strings.h>

#define AUTHORS         1
#define ARTICLE           2
#define KEYWORDS    3
#define ABSTRACT     4
#define Y   1
#define N   0

    FILE   *fopen(), *output;


@interface ArticleFile:Responder
{
     id             place;
     id             pages;
     id             date;
     id             title;
     id             keyWord;
     id             dueOn;
     id             country;
     id             num;
     id             comments;
     id             vol;
     id             lname;
     id             journal;
     id             chkout;
     id             fname;
     id                     wordField;
     id             absWindow;
     id             authWindow;
     id             articleWindow;
     id             keyWindow;
```

```
    id          numberWindow;
    id          directoryWindow;
    id          articleNumber;
    id          directory;

}

- setPlace:anObject;
- setPages:anObject;
- setDate:anObject;
- setTitle:anObject;
- setKeyWord:anObject;
- setDueOn:anObject;
- setCountry:anObject;
- setNum:anObject;
- setComments:anObject;
- setVol:anObject;
- setLname:anObject;
- setJournal:anObject;
- setChkout:anObject;
- setFname:anObject;
- setArticleNumber:anObject;
- setDirectory:anObject;
- setWordField:anObject;
- setAuthWindow:anObject;
- setKeyWindow:anObject;
- setArticleWindow:anObject;
- setAbsWindow:anObject;
- setNumberWindow:anObject;
- setDirectoryWindow:anObject;

- addKeyWordAndClose:sender;
- addMoreKeyWords:sender;
- addArticleInfoAndClose:sender;
- addAbstractAndClose:sender;
- addAuthorAndClose:sender;
- addMoreAuthors:sender;
- warnUser:sender;
- skipAbstract:sender;
- fileInfoOpen:(int)item;
- clearAllFields;
- collectInfo:sender;
- deleteInfo:sender;
- indexInfo:sender;
```

```
- updateInfo:sender;
- searchWords:sender;

@end
```

IMPLEMENTATION FILE (ArticleFile.m)

```
/* Programmed    by Krishnaprasad Kamisetty */

#import "ArticleFile.h"

@implementation ArticleFile

/* Initialization Methods */

- setPlace:anObject
{
      place = anObject;
      return self;
}

- setPages:anObject
{
      pages = anObject;
      return self;
}

- setDate:anObject
{
      date = anObject;
      return self;
}

- setTitle:anObject
{
      title = anObject;
      return self;
}


- setKeyWord:anObject
{
      keyWord = anObject;
```

```
    return self;
}

- setDueOn:anObject
{
    dueOn = anObject;
    return self;
}

- setCountry:anObject
{
    country = anObject;
    return self;
}

- setNum:anObject
{
    num = anObject;
    return self;
}

- setComments:anObject
{
    comments = anObject;
    return self;
}

- setVol:anObject
{
    vol = anObject;
    return self;
}

- setLname:anObject
{
    lname = anObject;
    return self;
}



- setJournal:anObject
{
    journal = anObject;
```

```
    return self;
}

- setChkout:anObject
{
    chkout = anObject;
    return self;
}

- setFname:anObject
{
    fname = anObject;
    return self;
}

- setDirectory:anObject
{
    directory = anObject;
    [directory setStringValue:"Articles" at:0];
    [directory selectTextAt:0];
    return self;
}

- setWordField:anObject
{
   wordField = anObject;
   return self;
}

- setAbsWindow:anObject
{
    absWindow = anObject;
    [absWindow setFreeWhenClosed:NO];
    [absWindow close];
    return self;
}




- setAuthWindow:anObject
{
   authWindow = anObject;
  /*   [authWindow setFreeWhenClosed:NO];
```

```
    [authWindow close];   */
    return self;
}

- setArticleWindow:anObject
{
    articleWindow = anObject;
    [articleWindow setFreeWhenClosed:NO];
    [articleWindow close];
    return self;
}

- setKeyWindow:anObject
{
    keyWindow = anObject;
    [keyWindow setFreeWhenClosed:NO];
    [keyWindow close];
    return self;
}

- setNumberWindow:anObject
{
     numberWindow = anObject;
     [numberWindow setFreeWhenClosed:NO];
     [numberWindow close];
     return self;
}

- setDirectoryWindow:anObject
{
    directoryWindow = anObject;
    [directory setFreeWhenClosed:NO];
    [directory close];
    return self;
}




- setArticleNumber:anObject
{
     articleNumber = anObject;
     return self;
}
```

```
/* The actual Instance methods */

- addKeyWordAndClose:sender
{
        const char *temp1;
        char     temp[20];

      temp1 = [keyWord stringValueAt:0];
      sprintf(temp,"%s",temp1);
      fprintf(output,"%s.",temp);
      [NXApp stopModal];
      [keyWindow close];
      return self;
}

- addMoreKeyWords:sender
{
        const char *temp1;
        char temp[20];

      temp1 = [keyWord stringValueAt:0];
      sprintf(temp,"%s",temp1);
      fprintf(output," %s,",temp);
      [keyWord selectTextAt:0];
      return self;
}

- addArticleInfoAndClose:sender
{
        const char *temp1;
        char temp[75];

      temp1 = [title stringValueAt:0];
      sprintf(temp,"%s",temp1);
      fprintf(output, "\"%s\",",temp);
      temp1 = [journal stringValueAt:0];
      sprintf(temp,"%s",temp1);
      fprintf(output," %s,",temp);
      temp1 = [vol stringValueAt:0];
      sprintf(temp, "%s",temp1);
      fprintf(output," Vol. %s,",temp);
      temp1 = [num stringValueAt:0];
      sprintf(temp, "%s",temp1);
```

```
    fprintf(output," No. %s,",temp);
    temp1 = [date stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output," %s,",temp);
    temp1 = [pages stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output," pp %s,",temp);
    temp1 = [place stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output," %s,",temp);
    temp1 = [country stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output," %s.\n\n",temp);
    temp1 = [comments stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output,"Comments: %s\n\n",temp);
    temp1 = [chkout stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output,"Checked Out By: %s\n",temp);
    temp1 = [dueOn stringValueAt:0];
    sprintf(temp, "%s",temp1);
    fprintf(output,"Due On: %s",temp);

    [NXApp stopModal];
    [articleWindow close];
    return self;
}

- addAbstractAndClose:sender
{
    [NXApp stopModal];
    [absWindow close];
    return self;
}



- addAuthorAndClose:sender
{
     const char *temp1;
     char    temp[40];

    temp1 = [lname stringValueAt:0];
    sprintf(temp,"%s",temp1);
```

```
	fprintf(output," %s,",temp);
	temp1 = [fname stringValueAt:0];
	sprintf(temp,"%s",temp1);
	fprintf(output,"%s.,",temp);
	[NXApp stopModal];
	[authWindow close];
	return self;
}

- addMoreAuthors:sender
{
	 const char *temp1;
	 char temp[20];

	temp1 = [lname stringValueAt:0];
	sprintf(temp,"%s",temp1);
	fprintf(output," %s,",temp);
	temp1 = [fname stringValueAt:0];
	sprintf(temp,"%s",temp1);
	fprintf(output," %s,",temp);
	[lname selectTextAt:0];
	return self;
}

- warnUser:sender
{
	 NXRunAlertPanel("WARNING","The feature that you selected has not
yet been implemented","FINE!", NULL,NULL);
	return self;
}




- skipAbstract:sender
{
	[NXApp stopModal];
	[absWindow close];
		NXRunAlertPanel("Message","Abstract	is	Skipped","OK!!",
NULL,NULL);
	return self;
}
```

```
- fileInfoOpen:(int)item
{
    switch (item)
    {
        case AUTHORS:
            [authWindow makeKeyAndOrderFront:self];
            break;
        case ARTICLE:
            [articleWindow makeKeyAndOrderFront:self];
            break;
        case KEYWORDS:
            [keyWindow makeKeyAndOrderFront:self];
            break;
        case ABSTRACT:
                                                [absWindow
makeKeyAndOrderFront:self];
            break;
    default:
            [authWindow makeKeyAndOrderFront:self];
            break;
     }
    return self;
}

- clearAllFields
{
      [articleNumber setStringValue:"" at:0];
      [lname setStringValue:"" at:0];
      [fname setStringValue:"" at:0];
      [title setStringValue:"" at:0];
      [journal setStringValue:"" at:0];
      [vol setStringValue:"" at:0];
      [num setStringValue:"" at:0];
      [date setStringValue:"" at:0];
      [pages setStringValue:"" at:0];
      [place setStringValue:"" at:0];
      [country setStringValue:"" at:0];
      [comments setStringValue:"" at:0];
      [chkout setStringValue:"" at:0];
      [dueOn setStringValue:"" at:0];
      [keyWord setStringValue:"" at:0];
      return self;
}
```

```
- collectInfo:sender
{
        char cpCommand[35], rmCommand[35];
        char dir[15];

      sprintf(dir,"~/%s",[directory stringValueAt:0]);
      [directory selectTextAt:0];
      [directoryWindow close];
                        sprintf(cpCommand,"cp     ~/%s     %s/.",[articleNumber
stringValueAt:0],dir);
      sprintf(rmCommand,"rm ~/%s",[articleNumber stringValueAt:0]);
      output = fopen([articleNumber stringValueAt:0], "w");
      [numberWindow close];
      [self   fileInfoOpen:AUTHORS];
      [NXApp runModalFor:authWindow];
      [self fileInfoOpen:ARTICLE];
      [NXApp runModalFor:articleWindow];
      [self fileInfoOpen:KEYWORDS];
      fprintf(output,"\n\nKEYWORD(S):\n\n");
      [NXApp runModalFor:keyWindow];
      [self fileInfoOpen:ABSTRACT];
      [NXApp runModalFor:absWindow];
      fclose(output);
      system(cpCommand);
      system(rmCommand);
        NXRunAlertPanel("Add Article","Your article %s has been added to
%s directory","OK",
                        NULL,NULL,[articleNumber     stringValueAt:0],[directory
stringValueAt:0]);
      [self clearAllFields];
      return self;
}



- deleteInfo:sender
{
      char deleteCommand[20];
      const char   *dir, *artnum;

      dir = [directory stringValueAt:0];
      artnum = [articleNumber stringValueAt:0];
```

```
        sprintf(deleteCommand,"rm ~/%s/%s",dir, artnum);
        [directoryWindow close];
        [numberWindow close];
        system(deleteCommand);
         NXRunAlertPanel("Delete Article","Your article %s has been deleted
from %s directory","OK",NULL,NULL,artnum,dir);
              return self;
}

- indexInfo:sender
{
      char indexCommand[25];
      id info;
      NXModalSession *nextModal;

      sprintf(indexCommand,"ixBuild %s",[directory stringValueAt:0]);
                info = NXGetAlertPanel("Indexing Status","Indexing %s
directory...Patience!",NULL,NULL,NULL,[directory stringValueAt:0]);
      nextModal = [NXApp beginModalSession:NULL for:info];
      system(indexCommand);
      [NXApp endModalSession:nextModal];
      NXFreeAlertPanel(info);
       NXRunAlertPanel("Index Completion Message","Indexing %s directory
completed","Good!",NULL,NULL,[directory stringValueAt:0]);
      return self;
}

- updateInfo:sender
{
    char updateCommand[30];
    id info;
    NXModalSession *nextModal;

    sprintf(updateCommand,"Edit ~/%s/%s",[directory stringValueAt:0],
[articleNumber stringValueAt:0]);


 info = NXGetAlertPanel("Update"," Edit will open to let you update article
%s in %s directory",NULL,NULL,NULL,[articleNumber   stringValueAt:0],
[directory stringValueAt:0]);
    [articleNumber setStringValue:"" at:0];
    [numberWindow close];
    [directory selectTextAt:0];
    [directoryWindow close];
```

```
    nextModal = [NXApp beginModalSession:NULL for:info];
    [NXApp endModalSession:nextModal];
    NXFreeAlertPanel(info);
    system(updateCommand);
    return self;
}

- searchWords:sender
{
    int speakerResult, ignored;
    port_t   appPort;

    /* look up the public port for app's Listener on local host */
    appPort   = NXPortFromName("Librarian",NULL);
    if (appPort == PORT_NULL)
        NXRunAlertPanel("Failure","Couldn't find the Public port for
Librarian","Too Bad!",NULL,NULL);

    /* connect   the port to the Application's Speaker */

   [ [NXApp appSpeaker] setSendPort:appPort];

     /* use the openFile:ok:method (Librarian uses this for lookup requests.
The others use it for file-opening requests */

speakerResult = [[NXApp appSpeaker] openFile:[wordField
stringValueAt:0] ok:&ignored];
    if (speakerResult != 0 )
        NXRunAlertPanel("Fiasco","Couldn't pass the message to the
Librarian","Hummm",NULL,NULL);
    return self;
}

@end
```

Table of Contents