

12-2007

RSS Management: An RSS Reader to Manage RSS Feeds That Efficiently and Effectively Pulls and Filters Feeds With Minimal Bandwidth Consumption

Brian Cooper
Columbus State University

Follow this and additional works at: https://csuepress.columbusstate.edu/theses_dissertations



Part of the [Computer Sciences Commons](#)


Recommended Citation

Cooper, Brian, "RSS Management: An RSS Reader to Manage RSS Feeds That Efficiently and Effectively Pulls and Filters Feeds With Minimal Bandwidth Consumption" (2007). *Theses and Dissertations*. 80.
https://csuepress.columbusstate.edu/theses_dissertations/80

This Thesis is brought to you for free and open access by the Student Publications at CSU ePress. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of CSU ePress.

RSS MANAGEMENT
AN RSS READER TO MANAGE RSS FEEDS THAT EFFICIENTLY AND
EFFECTIVELY PULLS AND FILTERS FEEDS WITH MINIMAL BANDWIDTH
CONSUMPTION

Brian Cooper



Digitized by the Internet Archive
in 2012 with funding from
LYRASIS Members and Sloan Foundation

<http://archive.org/details/rssmanagementrss00coop>

Columbus State University

The College of Science

The Graduate Program in Applied Computer Science

RSS Management
An RSS Reader to Manage RSS Feeds that Efficiently and Effectively Pulls and Filters Feeds with Minimal Bandwidth Consumption

A Thesis in

Applied Computer Science

by

Brian Cooper

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2007

© 2007 by Brian Cooper

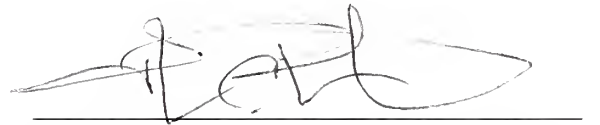
I have submitted this thesis in partial fulfillment of the requirements for the degree of Master of Science.

5/1/07
Date

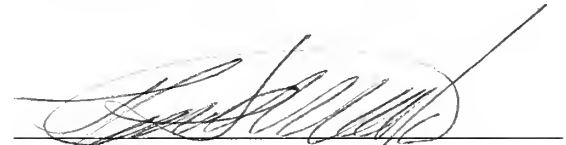

Brian Cooper

We approve the thesis of Brian Cooper as presented here.

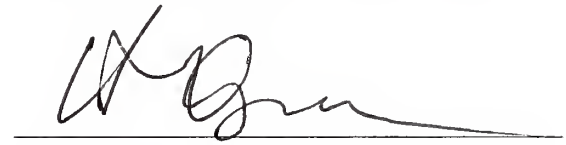
5/1/07
Date


Prof. Chris Whitehead, Assistant Professor
of Computer Science, Thesis Advisor

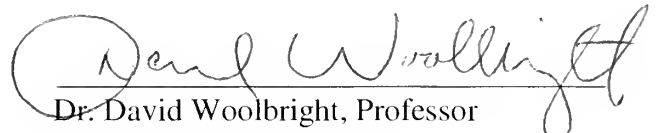
5/1/07
Date


Dr. Rodrigo Obando, Assistant Professor
of Computer Science, Assistant Chair

5-1-7
Date


Dr. Patrick Hogan, Computer Information
Systems Management Professor

5/1/07
Date


Dr. David Woolbright, Professor
of Computer Science

Abstract

In the early 2000s, RSS (Really Simple Syndication) was launched into cyber space and rapidly gained fame by existing as the underlying technology that fueled millions of web logs (blogs). Soon RSS feeds appeared for news, multimedia podcasting, and many other types of information on the Internet. RSS introduced a new way to syndicate information that allowed anyone interested to subscribe to published content and pull the information to an aggregator, (RSS reader application), at their discretion. RSS made it simple for people to keep up with online content without having to continuously check websites for new content. This new technology quickly had its shortcomings though. Aggregators were set to periodically check a feed for new content and if the new content did exist then the whole feed may be downloaded again and content filtering was either completely absent or filtering was performed once the file was already downloaded. Users who may have only occasionally checked a site for new content were now equipped with the ability to subscribe to content all over the web and have an aggregator poll the sites periodically for new content. However this presented a serious scalability problem in terms of bandwidth utilization. The same users that were checking a site once a day for new content were now checking the sites with the aggregator on a specific interval such as every hour. Bandwidth utilization increased dramatically where RSS was involved. The aim of this thesis is to design a better RSS aggregator that effectively and efficiently polls, downloads and filters RSS content while using a minimal amount of bandwidth and resources.

To meet these needs, an RSS aggregator named FeedWorks has been developed that allows for users to create subscriptions to content and set a interval to poll that subscription for newly published material. The aggregator uses specific HTTP

(hypertext transfer protocol) header information to check for new content before it downloads a file and if new content is found, downloads the file but filters it based on user-created filter criteria before it writes the information to disk. Filtering and searching algorithms have been researched to tune the performance and limit the strain on the processor. Caching mechanisms have also been used to enhance the performance of the application. The aggregator contains content management functionality to allow users to create subscriptions and subscription groups and to apply filters to a specific subscription or groups of subscriptions.

This thesis compares the aggregator with other currently available products and services. It provides detailed information regarding the end user's interface and the content management functionality it provides. Descriptive information is also presented that explains the content filtering and feed polling functionality and their respective algorithms.

7.8 Managing Blog Options	46
7.9 Managing Feeds	48
7.10 Refresh All/This Feed(s)	49
8. Polling and Filtering.....	50
8.1 Polling Schedules	50
8.2 Check for new content with HTTP headers	51
8.3 Filtering Feeds.....	52
8.4 Creating and Managing Keyword Search Filters	53
8.5 Applying Filters to Subscriptions.....	54
8.6 Filtering by Including Keywords	54
8.7 Search/Filter Algorithm.....	54
9. Knowledge Bases	57
9.1 Creating a Knowledge Base	57
9.2 Searching Knowledge Bases	58
9.3 Future Functionality of RSS Knowledge Bases.....	59
10. Installing and Uninstalling FeedWorks.....	61
11. Testing.....	62
12. Conclusion.....	63
12.1 Future Work	64
12.2 Future Testing	65
Appendix A: Installation Guide for FeedWorks	67
References	68

List of Figures

Figure 1. Feed Demon.....	18
Figure 2. BlogLines.....	19
Figure 3. Feed Reader.....	20
Figure 4. Google Reader.....	21
Figure 5. Sharp Reader.....	22
Figure 6. Database Schema.....	37
Figure 7. FeedWorks Main Screen.....	40
Figure 8. FeedWorks Add New Feed Screen.....	41
Figure 9. FeedWorks Add Feeds with OPML Screen.....	43
Figure 10. Export OPML Save As Dialog Screen.....	44
Figure 11. Feed Right-Click Context Menu.....	45
Figure 12. Search Functionality Example.....	45
Figure 13. Export to Blog Example.....	46
Figure 14. Managing Blog Options Screen.....	47
Figure 15. Managing Feeds Screen.....	48
Figure 16. Managing the Schedule Screen.....	51
Figure 17. Mange Filters Screen.....	54
Figure 18. Mange Filters Screen.....	56
Figure 19. Mange Filters Screen.....	56
Figure 20. Manage Knowledge Bases Screen.....	58
Figure 21. Searching Knowledge Bases.....	59
Figure 22. Progress Bar.....	60

List of Tables

Table 1. Competitors	16
Table 2. Comparison of Features of readers	24
Table 3. Field descriptions of tblSubscriptions	27
Table 4. Field descriptions of tblFeeds	28
Table 5. Field descriptions of tblArchives	29
Table 6. Field descriptions of tblFilters	30
Table 7. Field descriptions of tblKeys	31
Table 8. Field descriptions of tblFilterKeys.....	31
Table 9. Field descriptions of tblschedule.....	32
Table 10. Field descriptions of tblKnowledgeBase	32
Table 11. Field descriptions of tblKnowledgeBase_Content.....	33
Table 12. Field descriptions of tblBlog_Info	34
Table 13. Field descriptions of tblFilterSubscriptions	35
Table 14. Field descriptions of tblFilterSubscriptions	35
Table 15. Field descriptions of tblFilter_Subscriptions	35
Table 16. Field descriptions of tblFilter_Subscriptions	36

Acknowledgements

I would like to express my sincere gratitude to my thesis advisor Professor Chris Whitehead for helping me through the project by providing guidance, wisdom and patience. I greatly appreciate the mentoring and advice during the research project and the opportunity to be your student during graduate school. I also would like express my thanks to all my thesis committee Dr. Rodrigo Obando, Dr. Patrick Hogan, Dr. David Woolbright and all of my Professors at Columbus State University.

I would like to thank my family for the support during the long hours spent during the course of the research and especially my wife for supporting me through school and all of the time spent achieving this goal. I owe her more than I could ever repay.

I also would like to express my appreciation to my friends and colleagues at Ingram Barge for supporting me and helping me to make all of this possible. Without their unconditional support, I would not have had the ability to pursue this opportunity. Thank you, Dale Heller, Susan Levy, J Formosa, and Craig Philip.

Lastly I would like to thank Chris Bunch for encouragement, collaboration and healthy competition as we both worked so hard to achieve our educational goals. Thank you for being a true friend.

1. Introduction

The information age stands before us. The new frontier of cyber space has had its gold rush and now in its aftermath, humanity is left to reorganize, rethink, and organize the vast amounts of information available that continues to grow at a rapid pace. Now trends are changing and the web is no longer new. Instead, it is a tool; a place to find information from catalogs, conversations, and communities. The web has plugged users into a collective consciousness and new problems have arisen that deal with information organization, findability, and security. The Internet community has shown that it is no longer happy surfing the web and spending massive amounts of time searching for information. Now users want to jump on a site and find out the information they need and leave the site [1]. Searching and surfing the web for information is very time consuming. Many web users now check a collection of continuously updated sites every day. As the web grows, so does the desire to consume more and more content, but having to physically surf to sites and search through those sites to read new content is time consuming and creates a physical bottleneck that severely limits the amount of information that is able to be consumed.

Different types of information delivery have been tried throughout the course of the web's rise to power. Feeds on websites brought live news, but users had very little control over that content and the technology was usually expensive and limited to only a handful of providers. Next, the Internet brought us email subscription content which turned into the giant industry of unsolicited advertising or as it is more affectionately known "spam." All of this technology was push-type technology. Once the subscription was created, the provider pushed the information to users who had very little control as

to what information was actually sent over the wire [2]. Creating a subscription with an email account could lead to massive amounts of spam swallowing the good information in a sea of useless unsolicited advertisements. The subscription service solutions were also not meant for the common Internet user. Subscribing to content was easy but creating syndicated content was not feasible for the normal Internet user.

A simple solution was necessary that would provide a standard means of syndicating content that would be easy to create and consume. That simple solution came in the 90s as an XML derived RSS (Really Simple Syndication) [3].

2. Motivation

RSS was everything that people wanted in a new syndication technology. The technology sparked the incredible growth of blogs, Internet web logs or dairies that spanned anything from syndicated personal life entries to business articles [3]. This growth boom created a firm place on the Internet for RSS. Syndicated content with RSS began to appear everywhere. News feeds from all major news providers, technical journals, video podcasts or radio programs and even prime time national television providers. RSS feeds also began to appear for alerting of events, forum articles, announcements, internal corporate communications, magazine articles, job sites, etc. Almost everything that was a list of discrete items seemed to be created as online content with a syndication option to subscribe as an RSS feed.

RSS reader applications, or as they are formally known as, aggregators, were launched to help people organize their RSS subscriptions. The aggregators allowed for the creation of channels or groups of types of content and then multitudes of RSS subscriptions within those channels. The aggregators could be set up to poll all RSS subscriptions at specific intervals to check for new content and then download the information into the aggregator for reading online or offline at the subscriber's convenience. The aggregators helped RSS consumers organize their subscriptions and the aggregators provided the ability to have newly published content shipped directly to readers without having to spend time surfing through sites looking to see if new content was available.

However, with so many advances came new problems. The first problem to arise was bandwidth [4]. The aggregators were being set to download anywhere from a

couple to hundreds of actual documents over the Internet and anywhere from once a day to every five minutes. RSS files are generally the size of an average Microsoft Word or web page file created with the web page language, HTML (Hypertext Markup Language); only being in the two to three digit kilobyte numbers. However, enable hundreds or thousands of users in a large corporation to use frequently polling RSS aggregators, and bandwidth quickly becomes a concern even with the small file sizes of RSS feeds. The main issue though was not the size of the files or the amount of polling the aggregator performed; instead, it was the fact that most readers downloaded all RSS files upon every poll regardless of the existence of new content or not. The intent of this paper is to research and develop ways to properly check for content updates without downloading the RSS file as well as provide a means of efficiently and effectively filtering content before writing it to disk.

The second problem to arise with RSS was that no filtering processes were present in aggregators [6]. This meant that a user may create a channel for a specific widget and subscribe to thirty blogs about that widget, but may notice that only a portion of the content from the sites are actually about the widget and the rest of the content may be of a personal nature or an off-topic nature. This not only can create frustration for the reader, but it can also undermine the organizational capabilities of the aggregators and increase the amount of bandwidth use causing further waste. This paper will present an efficient algorithm solution to search through, filter and include or exclude content based on keywords.

Lastly, as aggregators began to increase in popularity, functionality was added to archive information in files for later recall. For many of these aggregators,

functionality was provided to search the blog archives. This provided some needed retention of content that a reader may want to keep, but more was needed in this area to help organize the vast amounts of content on the web that may be on similar subjects but completely isolated from one another. To help with this issue, many Internet-savvy users created intermediary blogs that kept an import from a collection of similar blogs and provided that content all aggregated into one RSS feed. However, this is not enough to adequately organize the information because though the content may be similar, it still may not be filtered. It is possible to group forty blogs about a widget, the widgets may be of different categories such a repairing the widget, creating the widget and selling the widget. A user may only want to collect content on repairing the widget into a widget repair category. Many web sites will offer RSS feeds of a specific category such as widget repair. However, again, even this is not enough to adequately organize the information because though the content may be similar, it still may not be completely filtered to ensure the information is exactly what the consumer desires. So, though it is possible to group forty blogs about a widget, the blogs may have many different categories regarding widgets and that content may also not be completely filtered. Blog readers may also want to collect widget information and store it into an archived category for widget repair. This paper proposes to take this process a step further to create specific organizational structures to store feed content that has been filtered and archived to a common location; a knowledge base of archived feed content .

2.1 Desired Features

The desired features of the aggregator will first be normal aggregator functionality to create subscriptions to RSS feeds, set and manage schedules to import those feeds and the ability to browse through feeds. The aggregator will also have functionality to refresh feeds, mark feeds as read or unread, archive, delete, and blog feeds to a personal blog. The aggregator will also support the importing and exporting of Outline Processor Markup Language (OPML) files which is an XML derived standard that allows the sharing of a complete outline list of all subscribed blogs with other people.

Filtering functionality will be present to set keywords as inclusions or exclusions and apply those keywords as a filter object that may be applied to directly to specific RSS feeds.

Other management functions will exist to manage blog subscriptions by marking the subscriptions as active or inactive, or permanently deleting the subscriptions. Functionality will exist for scheduling of automatic new content polling and filtering of content as it is downloaded. Additional functionality will exist to create knowledge base collections of archived articles that may be searched within the aggregator.

During a development project such as this, market analysis of the competition is important to ensure that the demand for the product exists and that there is enough room for competition within the market.

3. Competitive Analysis

Competitive analysis was completed as a part of the thesis research to determine major players in the market place and to determine the strengths and weaknesses of their products. Major players were determined by popularity ranking in web search engine queries, company information, lists from various websites such as, <http://blogspace.com/rss/readers> and cross referencing findings with popular web culture sites such as Wikipedia < http://en.wikipedia.org/wiki/News_aggregator>. Once major players in the industry were discovered, an analysis was completed on each product to determine the extent of functionality and its strengths and weaknesses implementing that functionality (See [Table 1](#)). Each product analyzed was installed and used extensively to discover how the products worked and performed.

Table 1. Competitors

<i>Name</i>	<i>Company</i>	<i>Address</i>
FeedDemon	NewsGator	http://www.newsgator.com/
BlogLines	IAC Search & Media	http://www.bloglines.com/
Feed Reader	i-Systems Inc.	http://www.feedreader.com/
Google Reader	Google	http://www.google.com/reader/view/
Sharp Reader	Luke Hutteman	http://www.sharpreader.net/

As shown in [Table 1](#), the major players identified with applications similar to FeedWorks are FeedDemon, BlogLines, Feed Reeder, Google Reader and Sharp Reader.”

3.1 FeedDemon

FeedDemon (see Figure 1) is a RSS reader developed by NewsGator Technologies, one of the leading RSS platform companies. NewsGator is headquartered in Denver, Colorado, and can be found on the web at <http://www.newsgator.com/>. NewsGator's RSS reader, FeedDemon, is one of the more robust RSS readers. The reader provides common RSS reader functionality that shall be considered from this point forward as: subscribing to feeds, managing feeds, deleting feeds, marking as read/not read, manually importing items, scheduling imports, as well as browser type functionality for reading blogs.

FeedDemon also provides good advanced functionality such as creating filters called watches. These watches can watch for multiple keywords located within the title of description fields of a feed. The reader downloads all of its subscribed feeds into their own respective flat files. It only downloads content if it is new which saves on bandwidth, but it reruns the watch each time and takes out all content that meets the watch's criteria. The reader then places the copied content in a separate flat file duplicating the information. This means that a reader may subscribe to a feed and only want content from that feed if it matches a certain criteria, but FeedDemon will still download all of the content regardless of the watch. FeedDemon will first download everything and then run the filter and create a new area for displaying the filtered content. This is inefficient because the content could be filtered before downloading it and writing the unnecessary content to file.

FeedDemon provides other advanced functionality such as archiving content and creating knowledge base areas called "News Bins." These areas allow the saving of

content to an area marked off for a specific type of content. For instance, a news bin may be created for all content about Data Mining to be used as a knowledge base for Data Mining information.

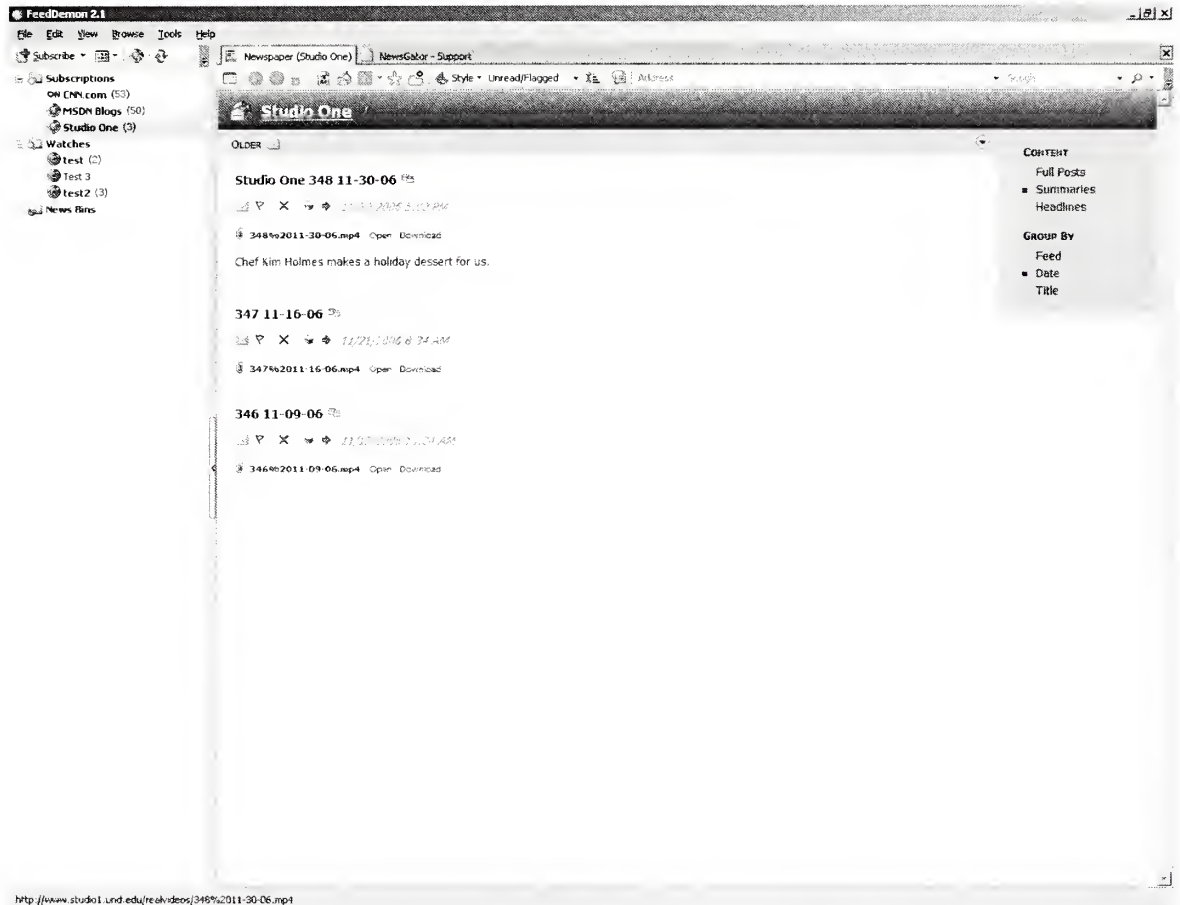


Figure 1. Feed Demon

3.2 Blog Lines

Blog Lines ([see Figure 2](http://www.bloglines.com/)) is an online RSS reader and can be found at <http://www.bloglines.com/>. The online reader is convenient since it can be accessed through a browser on any computer with Internet access and it stores all blog

subscriptions on its servers. Since this application is hosted on the Internet, much of the functionality is not configurable such as the poll interval to check for new content. Blog Lines does provide common reader functionality, but does not provide any advanced functionality such as filtering, archiving or a knowledge base. It is also unknown how efficient the parsing of content is, since it happens on their servers.

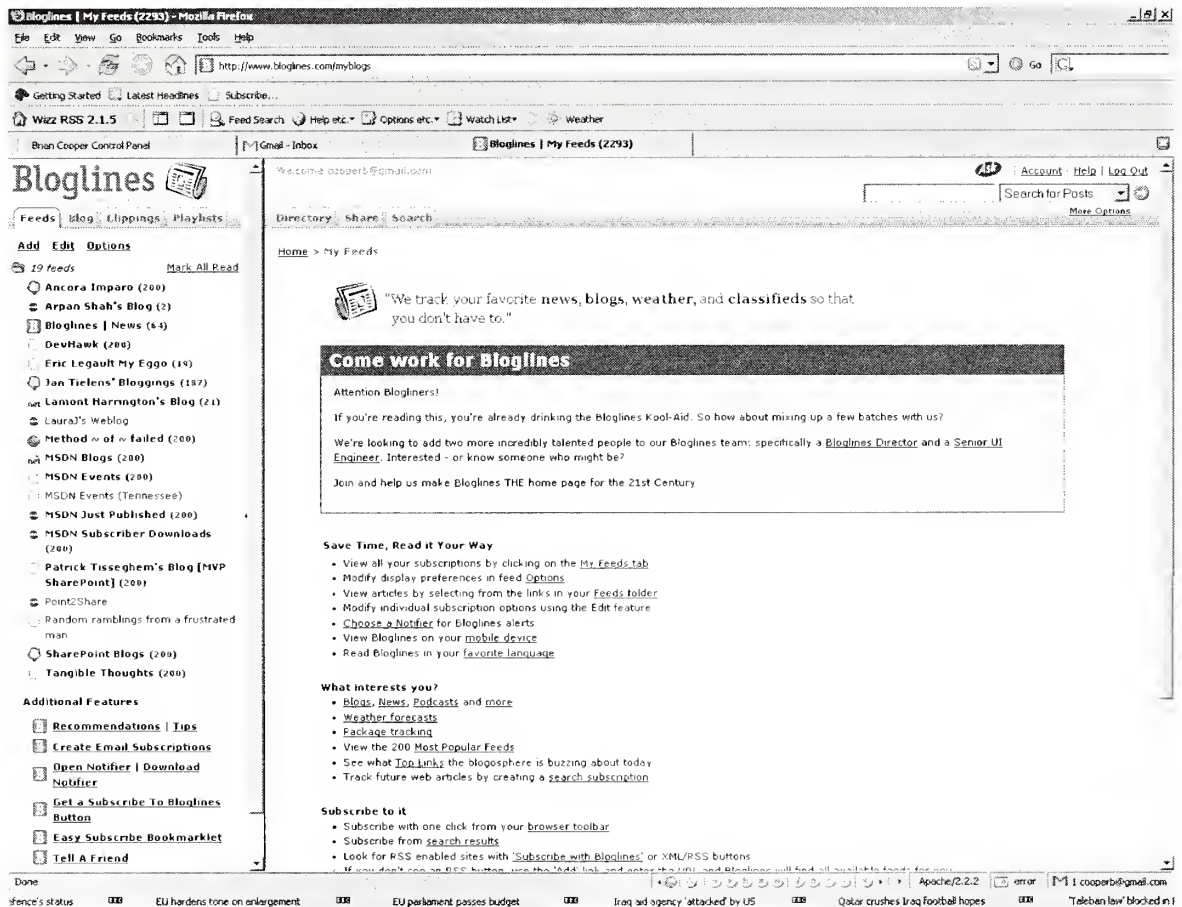


Figure 2. BlogLines

3.3 Feed Reader

Feed Reader (see Figure 3) is another client-based RSS reader that can be located at <http://www.feedreader.com/>. Feed Reader provides all common functionality.

The reader does provide some advanced functionality such as filters that may be applied that create what Feed Reader calls “smart feeds.” The smart feeds create a category of filtered content but the application still downloads all content from a feed into its proprietary database and then filters the content once it is downloaded. This is an inefficient approach since the real goal is to not waste bandwidth and disk IO (input/output) while downloading and filtering all content to disk. Feed Reader does not provide knowledge base functionality.

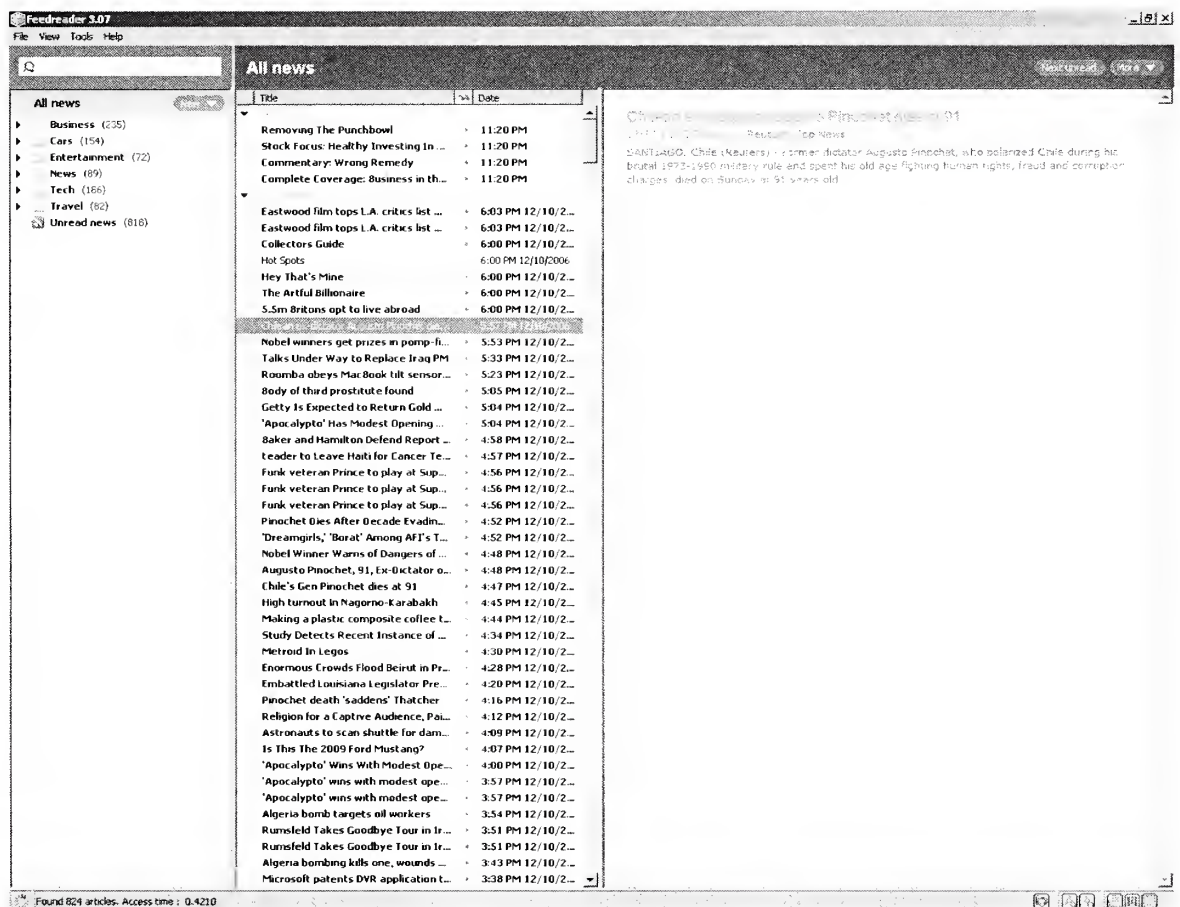


Figure 3. Feed Reader

3.4 Google Reader

Google Reader (see Figure 4) is an RSS reader developed by Google that can be found at <http://www.google.com/reader/view/>. The reader follows the same look and feel as most Google applications but provides only basic functionality of most RSS Readers. No advanced features are present such as filtering and knowledge base type technology. The reader is web based so efficiency of downloading content is based on Google's servers and is not configurable. The reader does plug directly into Blogger allowing direct posting to Google's own blog tool.

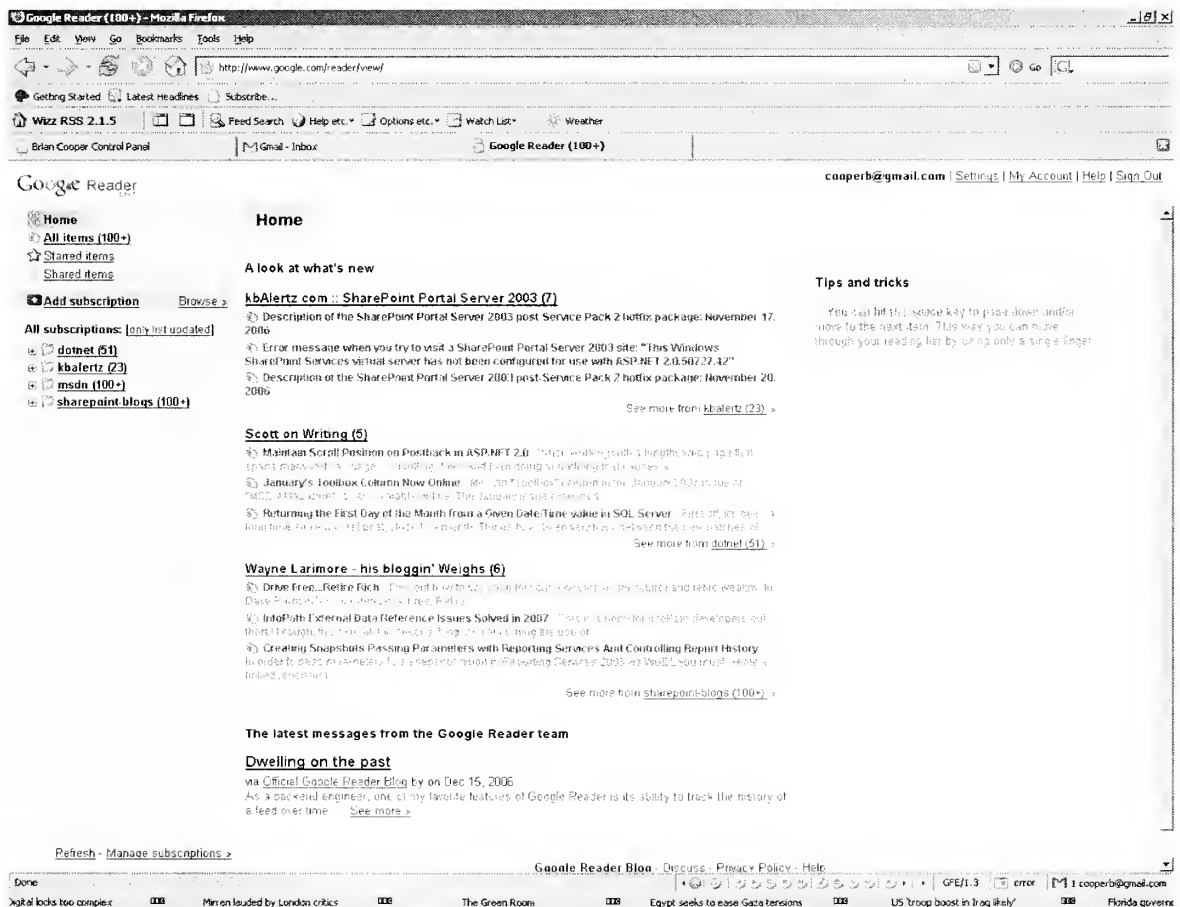


Figure 4. Google Reader

3.5 Sharp Reader

Sharp Reader (see Figure 5) is another client-based RSS reader that can be found at <http://www.sharpreader.net/>. The reader provides basic functionality common to most readers. It does provide some advanced functionality such as filtering. The filtering though, applies to every single subscribed feed instead of just one feed or a group of feeds. This means that a filter applied to a programming feed will also apply to a news feed from CNN. The filtering process also takes place once all feeds are downloaded to disk in their respective flat files. The reader provides no functionality for knowledge base type technology or automatically posting to blogs.

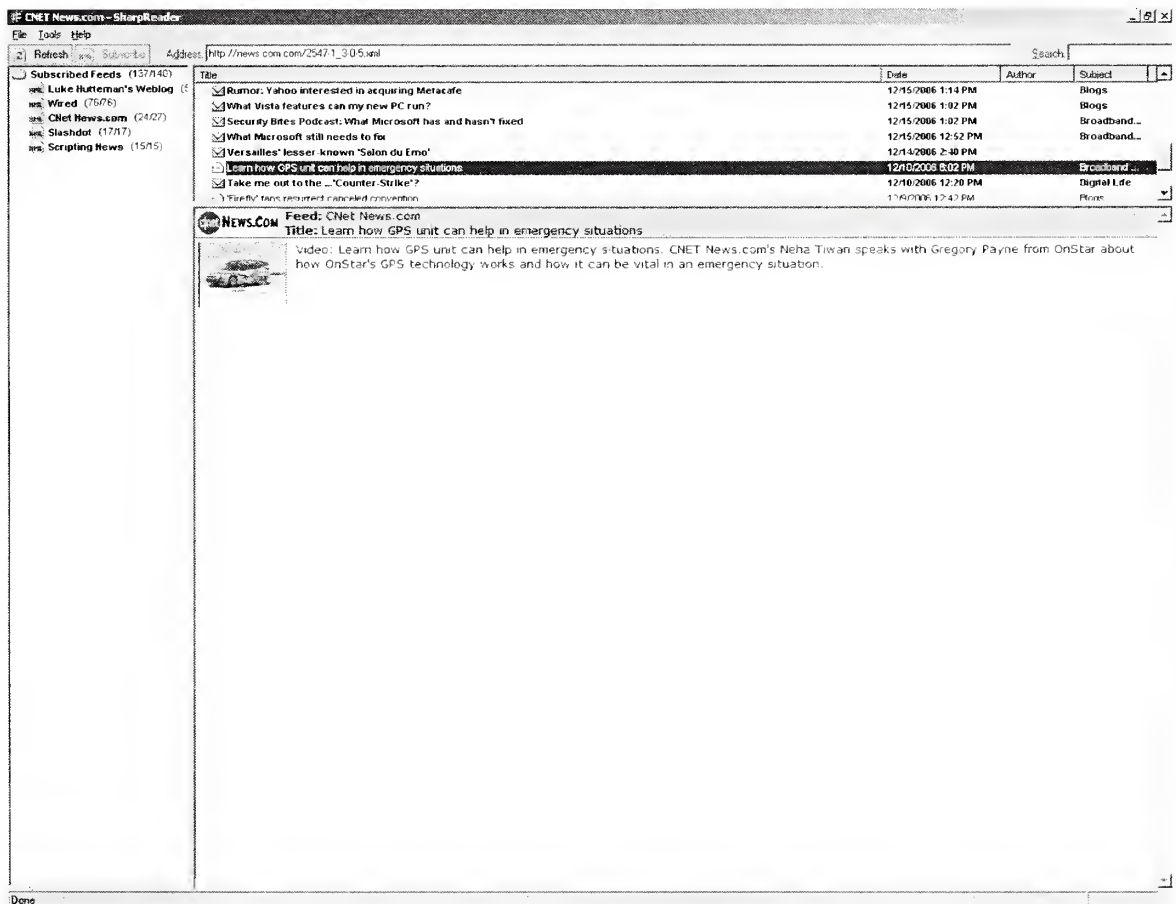


Figure 5. Sharp Reader

4. Survey of Proposed Product, FeedWorks

The product developed as part of this thesis is named FeedWorks. FeedWorks is a newsfeed aggregator which provides all basic functionality of a feed reader. FeedWorks also provides a more efficient method to filter subscription content and application tuning to reduce the bandwidth consumption while downloading subscription based content. In addition, FeedWorks provides archiving of content into general archives or into knowledge bases specifically geared to help organize large amounts of similar, archived content.

4.1 Comparison of Products

Competitors to FeedWorks were analyzed against specific criteria. This criterion was broken into five categories (See Table 2). A ‘Yes’ denotes that the feature is present and a ‘No’ denotes that the feature is not present at all. A ‘NA’ means that the feature is not applicable to the application probably because of platform constraints.

1. Common RSS reader functionality
 - a. Common RSS reader functionality is described in this research project as: subscribing to feeds, managing feeds, deleting feeds, marking as read/not read, manually importing items, scheduling imports, as well as browser type functionality for reading blogs.
2. Filtering process before File Input/Output
 - a. The filtering process that parses feeds being imported and applies custom filters either including or excluding content before it is written to disk.

3. Bandwidth reduction techniques

- a. Techniques to check for new or updated content before an RSS reader application utilizes bandwidth to download a RSS feed to memory.

4. Knowledge base type technology

- a. Knowledge based technology is the ability for a RSS reader application to group feed articles into a central location for user consumption as an aggregation of knowledge on a specifically defined category.

5. Blog Posting

- a. Blog posting is the ability for a RSS reader application to post a feed article to a user defined blog with the correct citations.

Table 2. Comparison of Features of readers

	Feed Demon	Blog Lines	Feed Reader	Google Reader	Sharp Reader	Feed Works
Common RSS Reader Functionality	Yes	Yes	Yes	Yes	Yes	Yes
Filtering Process Before File I/O	No	No	No	No	No	Yes
Bandwidth Reduction Techniques	Yes	NA	Yes	NA	Yes	Yes
Knowledge Base Type Technology	Yes	No	No	No	No	Yes
Blog Posting	Yes	Yes	No	Yes	No	Yes

5. Data Representation of RSS Feed Content

All data inside of FeedWorks is stored inside of a SQL Server database. A small version of SQL Server named SQL Server Express will be installed on the client's machine during the installation of FeedWorks. FeedWorks employs database technology instead of flat files to take advantage of indexing, stored procedures, views and easy data access to increase performance of the application.

5.1 Organizational structure of subscriptions and related content

FeedWorks stores all RSS Feed subscriptions in one table and uses this table to find the subscribed feeds and download their content. Once the content is downloaded, all content goes into one table where it resides until it is deleted, archived or moved into a knowledge base. Once any of the aforementioned actions takes place, the content is moved into a new table representing the content's new status. All content tables are optimized with indexing for fast lookups and views of the data are provided for the application to access the data quickly. All queries performed against the data stored within the database are done with precompiled stored procedures located on the SQL Server.

5.2 Content Archives

Archived content in FeedWorks is located in one archive table regardless of if that content is attached to a knowledge base. Archived content and knowledge bases may be viewed within FeedWorks by changing the view of the application from the

standard view to an archive or knowledge base view of the feed content. Archives are kept indefinitely and clean-up of the archives and knowledge bases is the user's discretion.

5.3 Database structure

RSS reader's aggregate content from many different sources into one place. FeedWorks takes this a step further and not only downloads content from subscribed feed sources, but it also provides intelligent features to filter and organize content. FeedWorks utilizes a relational database to effectively accomplish the filtering and organization of content. Below is a list of all the tables found within the project database, as well as a summary of each field used per table.

5.3.1 tblSubscriptions

tblSubscriptions (see Table 3) holds information on all subscribed feeds. This information is used to locate and download information from all subscribed feeds.

Table 3. Field descriptions of tblSubscriptions

<i>Field Name</i>	<i>Data Type</i>	<i>Key</i>	<i>Required</i>	<i>Description</i>
Sub_ID	AutoNumber (Integer)	Primary	Yes	Auto generated number for each subscription entered. Guarantees that each subscription will be uniquely identified.
Sub_URL	Text (nvarchar(2000))	None	Yes	URL of the subscription used to download the subscribed content.
Sub_Name	Text (nvarchar(500))	None	Yes	Title of the subscription resource.
Sub_Desc	Text (nvarchar(4000))	None	No	A short description of the subscription.
Sub_CreatedO n	DateTime (DateTime)	None	Yes	Date and Time the subscription was created.
Sub_Active	Bit	None	No	Determines if the subscriptions are active or inactive. By default all subscriptions are active

5.3.2 tblFeeds

tblFeeds (see Table 4) is the main table that stores all content that has been downloaded from subscribed feed sources.

Table 4. Field descriptions of tblFeeds

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Feeds_ID	AutoNumber - Identity (Integer)	Primary	Yes	Auto generated number for each feed inserted. Guarantees each feed will be uniquely identified
Feeds_Title	Text (nvarchar(MAX))	None	No	Title inside the feed content
Feeds_Author	Text (nvarchar(50))	None	No	Author that is located inside the feed data.
Feeds_PubDate	Text (nvarchar(50))	None	No	Date that the article was published. This date is located within the feeds data.
Feeds_Desc	Text (ntext))	None	No	Main body of content within the Feed.
Feeds_Link	Text (nvarchar(250))	None	No	Link back to the actual article. This information is located with the feed data.
Feeds_Content Title	Text (nvarchar(250))	None	No	Title of the Site that the feed data was downloaded.
Feeds_Read	Bit	None	Yes	Flag to mark the content as read or not read.
Sub_ID	Integer	Foreign Key	Yes	Foreign Key back to the actual subscription.
Feeds_DateSubmitted	DateTime	None	Yes	Date feed was written to the database

5.3.3 tblArchives

tblArchives (see Table 5) is almost exactly the same table as tblFeeds, the primary feed content table, except its purpose is to store content that has been marked for archiving or as knowledge base content.

Table 5. Field descriptions of tblArchives

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Archive_ID	AutoNumber - Identity (Integer)	Primary	Yes	Auto generated number for each feed inserted. Guarantees each archived feed will be uniquely identified
Archive_Author	Text (nvarchar(50))	None	No	Author that is located inside the archived feed data.
Archive_PubDate	Text (nvarchar(50))	None	No	Date that the article was published. This date is located within the archived feed's data.
Archive_Desc	Text (ntext))	None	No	Main body of content within the archived feed.
Archive_Link	Text (nvarchar(250))	None	No	Link back to the actual article. This information is located with the feed data.
Archive_ContentTitle	Text (nvarchar(250))	None	No	Title of the Site that the archived feed data was downloaded.
Archive_DateArchived	DateTime	None	Yes	Date the content was archived.
Sub_ID	Integer	Foreign Key	Yes	Foreign Key back to the actual subscription.

5.3.4 tblFilters

tblFilters (see Table 6) stores data on actual filter objects. Since more than one filter may be created and each filter that is created may have more than one filter key, three tables are required to store information regarding filters. A table must store the filter objects, a table must store the filter keys and a table must exist to link which filter keys map to which actual filter objects.

Table 6. Field descriptions of tblFilters

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Filter_ID	Number (Integer)	Primary	Yes	Auto generated number for each filter inserted. Guarantees each filter will be uniquely identified
Filter_Name	Text (nvarchar(50))	None	Yes	Name of the Filter.
Exclude	Bit	None	Yes	Determines if the filter is an exclusion filter or inclusion filter

5.3.5 tblKeys

tblKeys (see Table 7) stores all filter keys that are used to filter feed content. The keys stored in this table are the keys that are used in a filter to search for within feed data. Keys may be rules for excluding or including content that is found to possess key data.

Table 7. Field descriptions of tblKeys

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Key_ID	Number (Integer)	Primary & Foreign	Yes	Auto generated number for each Key inserted. Guarantees each Key will be uniquely identified
[Key]	Text (nvarchar(150))	None	Yes	The specific key data for which will be used to filter content.

5.3.6 tblFilterKeys

tblFilterKeys (see Table 8) is a table that maps keys to filters. From this table the application may discern all keys that make up a specific filter.

Table 8. Field descriptions of tblFilterKeys

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Filter_ID	Number (Integer)	Foreign Key back to tblFilter	Yes	Points back to the filter in which this Key belongs.
Key_ID	Number (Integer)	Foreign Key back to tblKeys	Yes	Points back to the Key which is part of this filter.

5.3.7 tblSchedule

tblSchedule (see Table 9) stores the interval in which the application will poll for new feed content against all subscriptions. The scheduler also has the ability to be turned off so that a user of the application may only poll manually for new content.

Table 9. Field descriptions of tblschedule

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Schedule_ID	Number (Integer)	Primary & Foreign	Yes	Auto generated number for each schedule inserted. Guarantees each schedule will be uniquely identified
Schedule_Time	Number (Integer)	None	Yes	The interval in which the aggregator will poll for new content.
Schedule_Enabled	Bit	None	Yes	Determines if the scheduler is enabled or not.

5.3.8 tblKnowledgeBase

tblKnowledgeBase (see Table 10) stores the knowledge base objects that help to organize archived content into knowledge base categories.

Table 10. Field descriptions of tblKnowledgeBase

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Know_ID	Number (Integer)	Primary	Yes	Auto generated number for each knowledge base inserted. Guarantees each knowledge base will be uniquely identified
Know_Name	Text (nvarchar(150))	None	Yes	Name of the knowledge base.
Schedule_CreatedOn	DateTime	None	Yes	Date the knowledge base was created.

5.3.9 tblKnowledgeBase_Content

tblKnowledgeBase_Content (see Table 11) maps the archived content to a one or more knowledge bases.

Table 11. Field descriptions of tblKnowledgeBase_Content

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Know_ID	Number (Integer)	Foreign Key	Yes	Refers back to a specific knowledge base.
Archive_ID	Number (Integer)	Foreign Key	Yes	Refers back to a specific article in the archives.

5.3.10 tblBlog_Info

tblBlog_Info (see Table 12) stores information that the application may use to automatically submit content to a blog. Many blog applications have interfaces that allow other programs to submit content. This table stores such information to allow users of FeedWorks to be able to submit content directly to their own blog.

Table 12. Field descriptions of tblBlog_Info

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Blog_ID	Number (Integer)	Primary	Yes	Auto generated number for each Blog inserted. Guarantees each Blog will be uniquely identified
Blog	(nvarchar(150))	None	Yes	Name of Blog.
Blog_Address	(nvarchar(150))	None	Yes	URL that points to Blog.
Blog_Username	(nvarchar(50))	None	No	Username to access API of Blog.
Blog_Password	(nvarchar(50))	None	No	Password to access API of Blog.

5.3.11 tblFilterSubscriptions

tblFilterSubscriptions (see Table 13) maps filters to a one or more subscriptions.

Table 13. Field descriptions of tblFilterSubscriptions

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Filter_ID	Number (Integer)	Foreign Key	Yes	Refers back to a specific Filter.
Sub_ID	Number (Integer)	Foreign Key	Yes	Refers back to a specific Subscription.

5.3.12 tblFolders

tblFilterSubscriptions (see Table 14) stores the name of Folders used as organizational objects that categorize feed subscriptions.

Table 14. Field descriptions of tblFilterSubscriptions

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Folder_ID	Number (Integer)	Primary Key	Yes	Auto generated number for each Folder inserted. Guarantees each Folder will be uniquely identified
FolderName	(nvarchar(50))	None	Yes	Name of Folder.

5.3.13 tblFolderSubscriptions

tblFilter_Subscriptions (see Table 15) maps subscriptions to folders.

Table 15. Field descriptions of tblFilter_Subscriptions

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Folder_ID	Number (Integer)	Foreign Key	Yes	Refers back to a specific Folder.
Sub_ID	Number (Integer)	Foreign Key	Yes	Refers back to a specific Subscription.

5.3.14 tblScheduleIntervals

tblFilter_Subscriptions (see Table 16) store the time intervals that may be used by the schedule to automatically check for new feed content.

Table 16. Field descriptions of tblFilter_Subscriptions

<i>Field Name</i>	<i>Data Type</i>	<i>Primary/ Foreign Key?</i>	<i>Required?</i>	<i>Description</i>
Schedule_Times_ID	Number (Integer)	Primary Key	Yes	Auto generated number for each Folder inserted. Guarantees each Folder will be uniquely identified
Schedules_Times	(Nvarchar(50))	None	No	Time intervals identified in 24 hour clock times.
Schedule_Time_r_Times	Number (Integer)	None	No	Time intervals identified in milliseconds.

Figure 6 is an entity-relationship diagram to demonstrate how primary and foreign keys are used in this database structure.



Figure 6. Database Schema

6. Implementation of FeedWorks

6.1 Language of Choice

C#.NET was chosen for this product primarily because it uses the .NET framework. The .NET framework is commonly installed on Windows operating systems which has the majority of the operating system market place [9]. Windows operating system exposure presents the ability for FeedWorks to reach large audience and it also provides the possibility for FeedWorks to interface with other standard Windows applications such as Microsoft Office. C# provides an object-oriented programming language that can be developed using Microsoft's Visual Studio.NET that presents programmers with a rich development interface and excellent debugging functionality. C# also supports inheritance, polymorphism, and encapsulation [5]. Another reason for the language choice is the large, active development community that fosters an excellent coding support environment full of help forums and how-to articles.

7. End User Interface

7.1 Working with content

FeedWorks' primary purpose is to aggregate feed content and present it for end user consumption. The presentation of content is extremely important since FeedWorks aggregates content from many sources and must organize and display that content inside a user interface in a manner that is easy to use and understand by average end users.

To accomplish this goal, FeedWorks mimics interfaces that many users are already familiar with and use to digest similar content. FeedWorks intentionally looks and feels very similar to Microsoft Outlook and Microsoft Explorer to try and keep users in a familiar environment since reading feed articles is very similar to reading emails and navigating to blog sites from feed articles actually steps into the realm of the internet browser. For the web surfing functionality FeedWorks actually employs the browser objects within .NET 2.0, to give the application a very similar experience of using Microsoft Internet Explorer. FeedWorks actually uses icons that represent browsers functionality such as back, forward, stop, and refresh.

FeedWorks also is developed using .NET 2.0 and keeps the Microsoft Windows look and feel to the application. This strategy gives many users familiarity with the application by just being familiar with the Windows platform. This is very important to the usability of the application since many of the fundamental functions of FeedWorks are actually driven by familiar Microsoft Windows forms functionality. This functionality includes minimizing, maximizing, closing, and resizing of the application window. The behavior of menus, right-click context menus, minimizing to the system tray, etc. are all items that are found commonly in Windows applications. Using these

controls that are familiar to many Windows users helps gain a level of inherent knowledge to the usability of FeedWorks for new users.

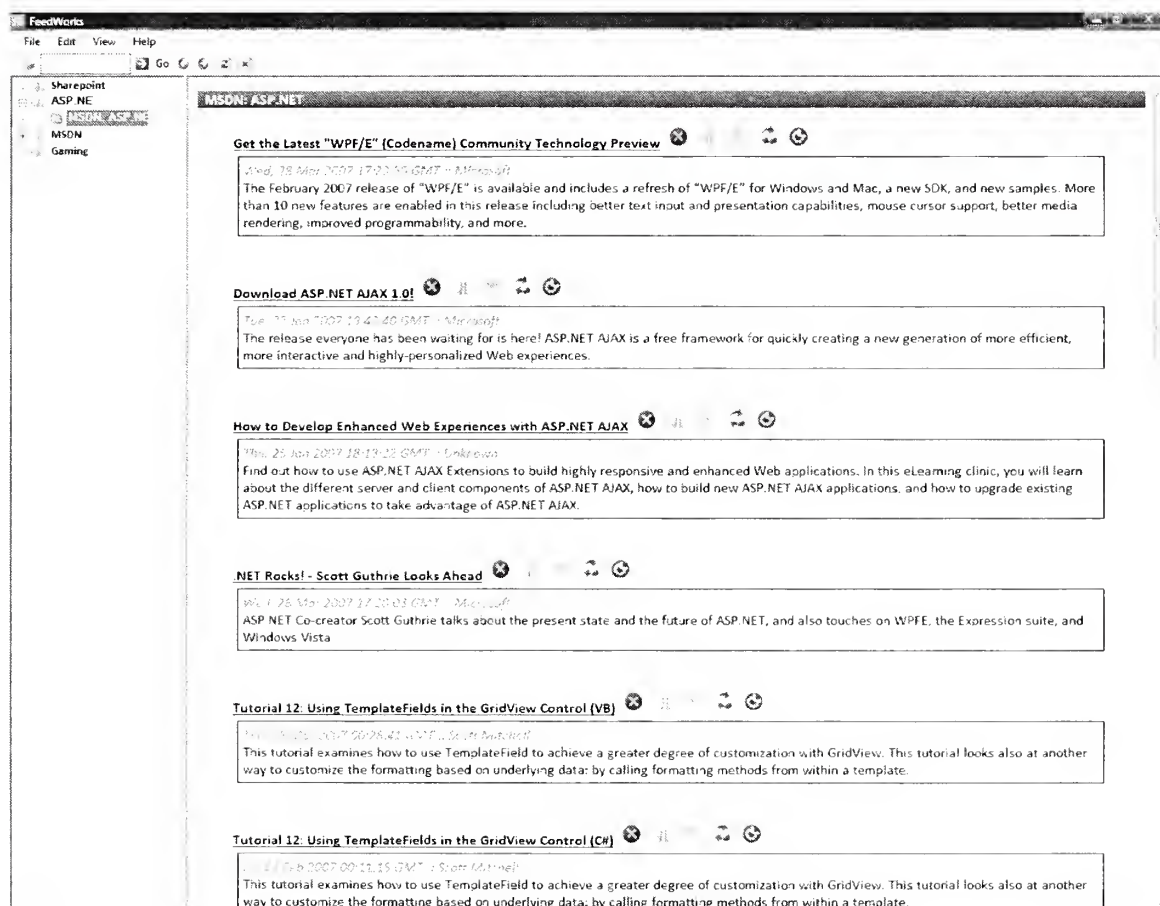
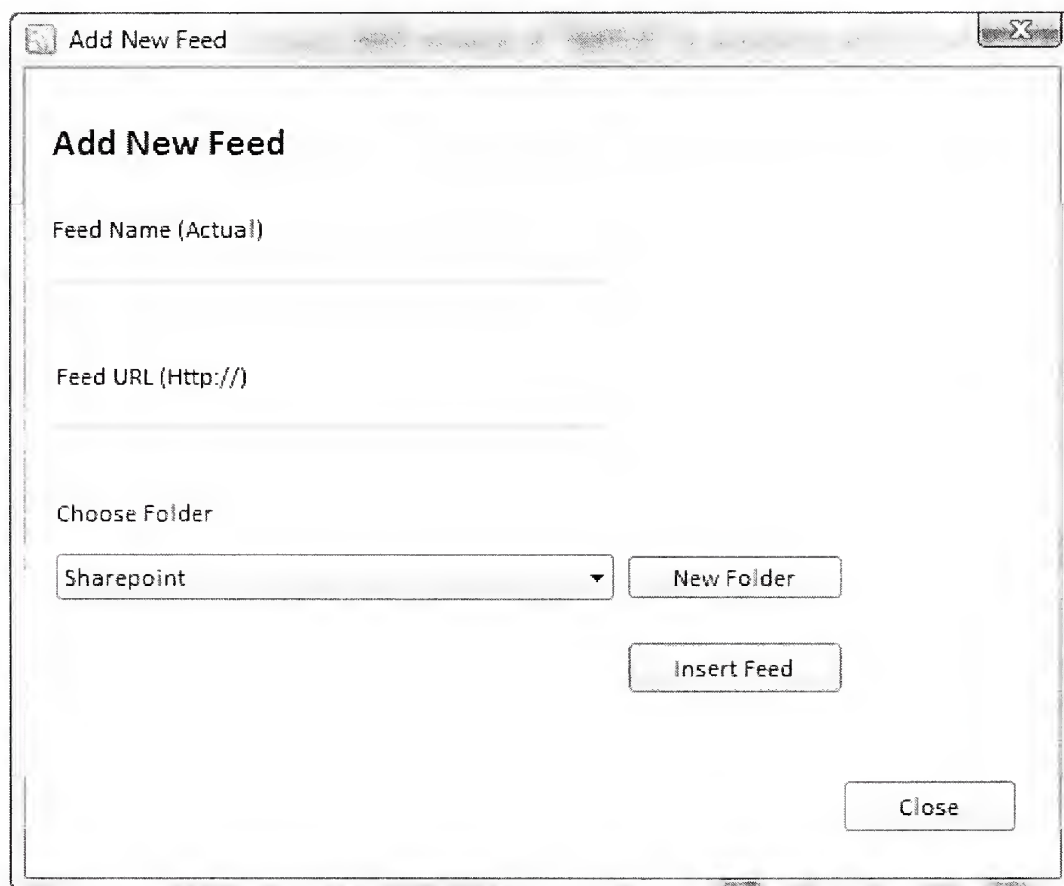


Figure 7. FeedWorks Main Screen

7.2 Subscribing to Feeds

One of the most fundamental pieces of functionality for FeedWorks' user interface is the ability to add new feeds. New feeds may be added to the application by using the "New Feed" option under "File" on the menu. Selecting this option displays

the dialog box shown in Figure 8. Adding a new feed adds the feed to the database and immediately validates the feed URL and imports the feed's content into the reader. The feed may also be given a meaningful name that is separate from the literal URL. As stated above, FeedWorks does validate the new feed by attempting to access it at the point of new feed creation. If the feed is invalid an error is returned to the user and no feed is added to the database. A folder may also be chosen or created from this form as feeds are grouped into folders for organization within FeedWorks.



The image shows a screenshot of a software dialog box titled "Add New Feed". The dialog box has a standard Windows-style title bar with a close button (an 'X' icon) in the top right corner. The main content area of the dialog is titled "Add New Feed" in a bold font. Below the title, there are three input fields: "Feed Name (Actual)" with a text input box, "Feed URL (Http://)" with a text input box, and "Choose Folder" with a dropdown menu. The dropdown menu currently shows "Sharepoint" and has a small downward arrow on its right side. To the right of the dropdown menu is a button labeled "New Folder". Below the "New Folder" button is another button labeled "Insert Feed". At the bottom right of the dialog box is a button labeled "Close".

Figure 8. FeedWorks Add New Feed Screen

7.3 Importing OPML

FeedWorks also allows feeds to be created using an OPML file (Outline Processor Markup Language). OPML files allow an easy way to share many feeds between feed consumers without having to manually enter each feed entry. Most online and offline feed readers and even many blogging tools allow for the export of feeds to an OPML file. FeedWorks may use these files to import groups of feeds as seen in [Figure 9](#). The OPML must be a physical file located on the user's desktop that may be located and uploaded. Each feed in the OPML file is treated like a separately inserted feed and is placed in the database and uploaded to FeedWorks. FeedWorks parses the OPML file before importing the feeds to validate that the OPML is well-formed. FeedWorks validates the feeds from the OPML before each feed is inserted because the application will skip bad feeds but continue to import those feeds that are valid. A message box is displayed after an OPML import stating any feeds that were not valid or had errors.

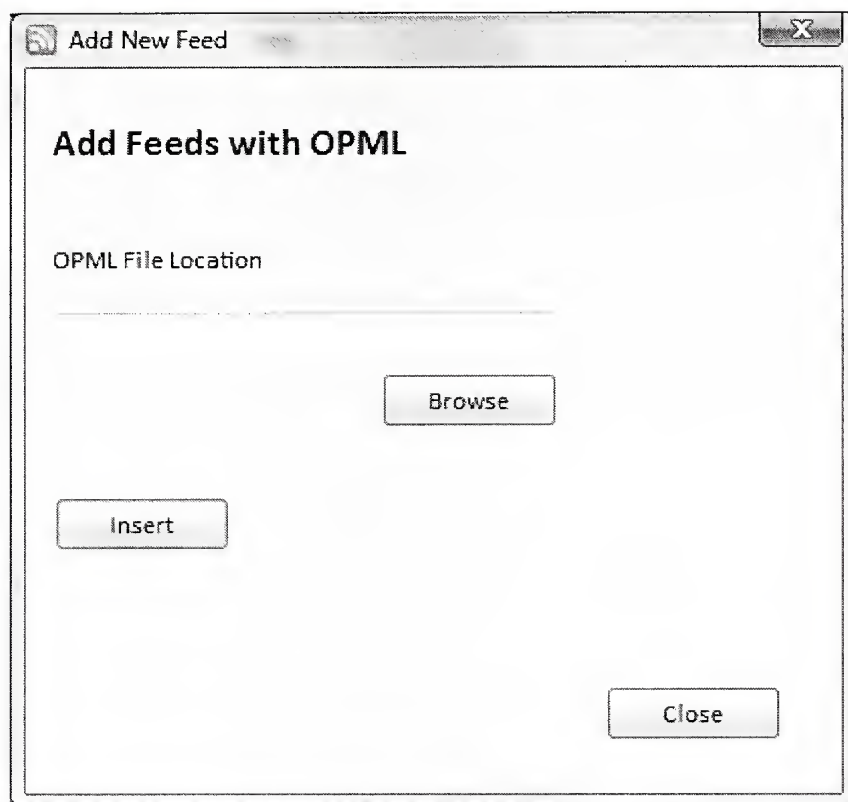


Figure 9. FeedWorks Add Feeds with OPML Screen

7.4 Exporting OPML

FeedWorks allows for the exporting of all feeds located within the FeedWorks database to an OPML flat file for easy sharing between different applications and people. Exports may be accomplished from the menu by click on File -> Export OPML as seen in [Figure 10](#). The flat file gets dynamically created and exported to a location on the user's desktop.

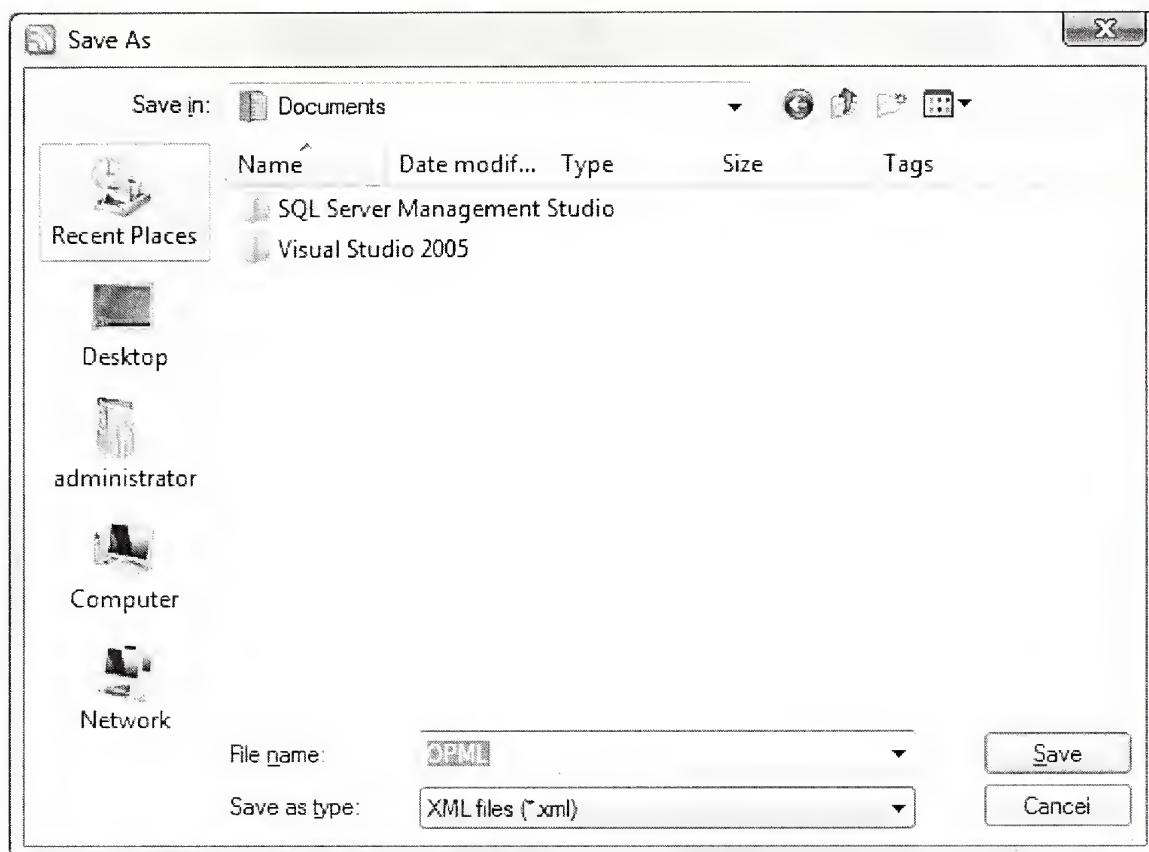


Figure 10. Export OPML Save As Dialog Screen

7.5 Marking as Read/Unread, Archiving, and Deleting

FeedWorks has functionality very similar to email applications to allow content to be marked as read, unread and even archive content that needs to be kept for later as seen in [Figure 11](#). Marking content as read or as unread allows the application users to manage the read state of content in case the content needs to be read again or doesn't interest the user and may need to be marked to not be read. Archiving allows FeedWorks users to keep content for later consumption. Application users may also permanently delete articles from this menu. All of the above functionality is located on the page of displayed feed content. Icons are displayed to represent actions for deleting and article, marking an article as read/unread and archiving an article.

Get the Latest "WPF/E" (Codename) Community Technology Preview



Wed, 28 Mar 2007 17:23:50 GMT :: Microsoft

The February 2007 release of "WPF/E" is available and includes a refresh of "WPF/E" for Windows and Mac. More than 10 new features are enabled in this release including better text input and presentation capabilities, improved rendering, improved programmability, and more.

Figure 11. Feed Right-Click Context Menu

7.6 Searching Content Stores

FeedWorks provides search functionality to search active feeds, archived feeds and knowledge bases to easily find content without having to dig through all feed content. This allows quick searching of content to find specific articles or even filtering to find all articles that contain specific search criteria. Search results are displayed within the browser window (see [Figure 12](#)) where they may be interactively clicked to view more detail about the article result.

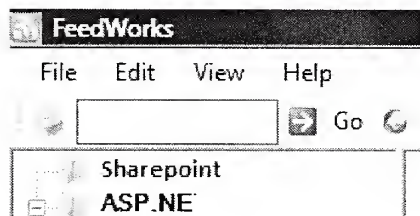


Figure 12. Search Functionality Example

7.7 Blogging Content

FeedWorks provides functionality for users that own their own blog to automatically post content to their blog from FeedWorks as seen in [Figure 13](#). The FeedWorks user must first setup their blog in the application options. Once their blog is successfully added to the application, the user may then click on any article to directly post it to their blog with automatic citations to the source of the content. FeedWorks works with the MetaBlog API to interface with blog applications. The MetaWeblog API (MWA) is a programming interface that allows external programs to get and set the text and attributes of weblog posts [6].

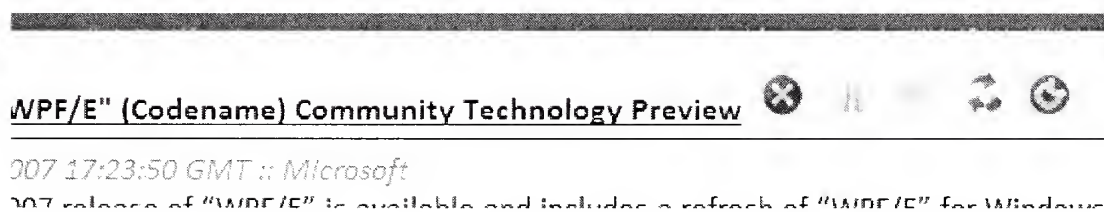
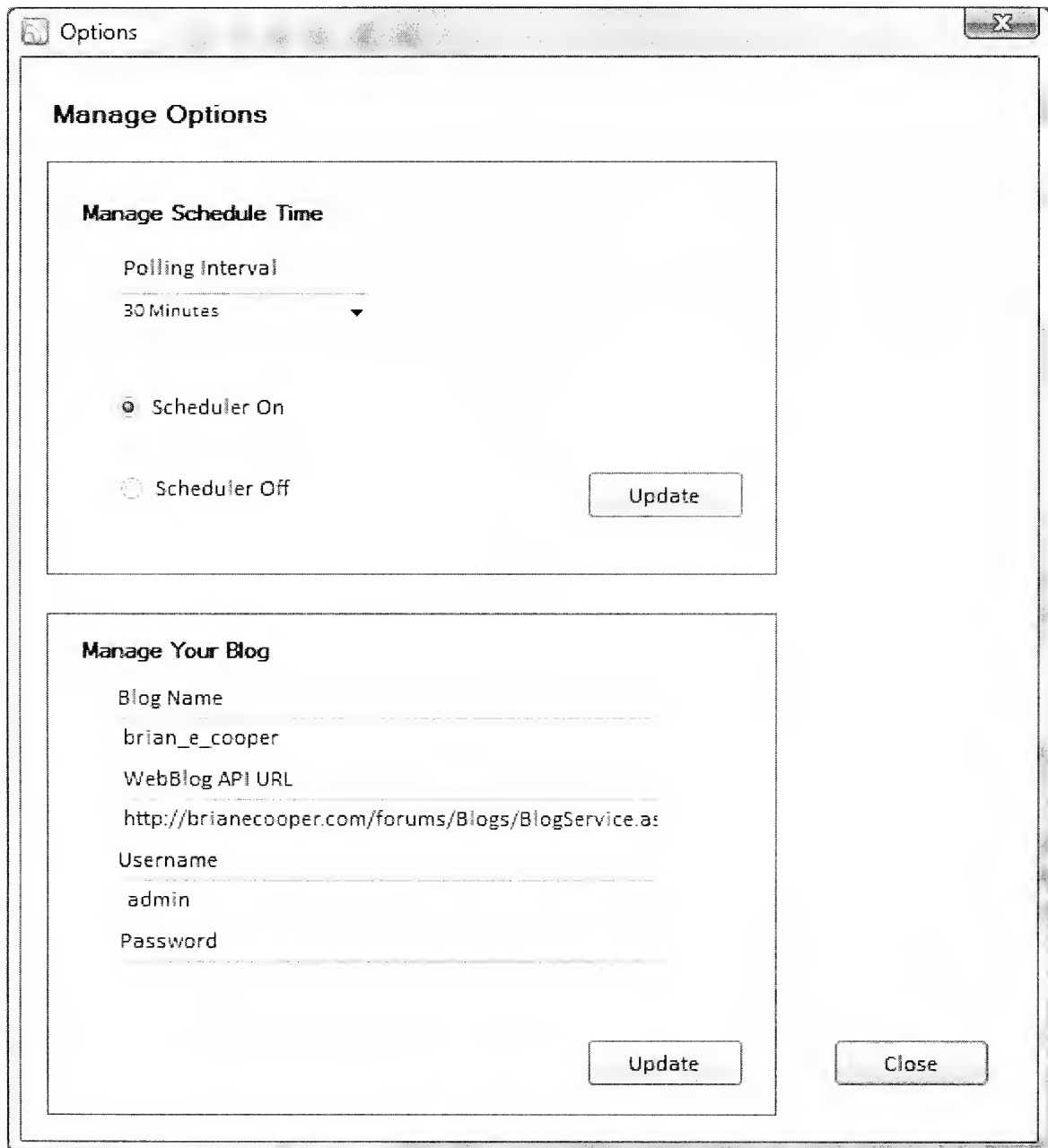


Figure 13. Export to Blog Example

7.8 Managing Blog Options

Users of FeedWorks may set a blog in the options as the default blog to use when exporting feed content to a blog (see [Figure 14](#)). Only the name of the user's blog and the direct URL to the MetaWeblog API which should be stated within the blog's documentation, and user's blog username and password need to be entered for the export to work correctly. FeedWorks uses the MetaWeblog API to interface with the

blog and passes it the information from the feed article as well as the username and password set by the user to authenticate and post the content to the stated blog. If an error occurs, the error is stated to the user within a pop-up message box as to the nature of the error so that the user may troubleshoot such items as firewall access.



The screenshot shows a window titled "Options" with a close button in the top right corner. The window is divided into two main sections:

- Manage Options**: This section contains a "Polling Interval" dropdown menu set to "30 Minutes". Below it are two radio buttons: "Scheduler On" (which is selected) and "Scheduler Off". An "Update" button is located to the right of these options.
- Manage Your Blog**: This section contains four text input fields:
 - Blog Name: brian_e_cooper
 - WebBlog API URL: http://briane.cooper.com/forums/Blogs/BlogService.as
 - Username: admin
 - Password: (empty)An "Update" button is located below the input fields, and a "Close" button is located to its right.

Figure 14. Managing Blog Options Screen

7.9 Managing Feeds

FeedWorks provides the ability to manage current feeds through the “Manage Feeds” screen under “options” within the main screen menu as seen in [Figure 15](#). Feeds may be marked as active, inactive or permanently deleted. Inactive blogs keeps the blog for the possibility of marking it as active later, but deleting a blog marks it as permanently deleted. Permanently deleted blogs still reside in the database for reference of any archive or knowledge base content but do not show in the active or inactive list.

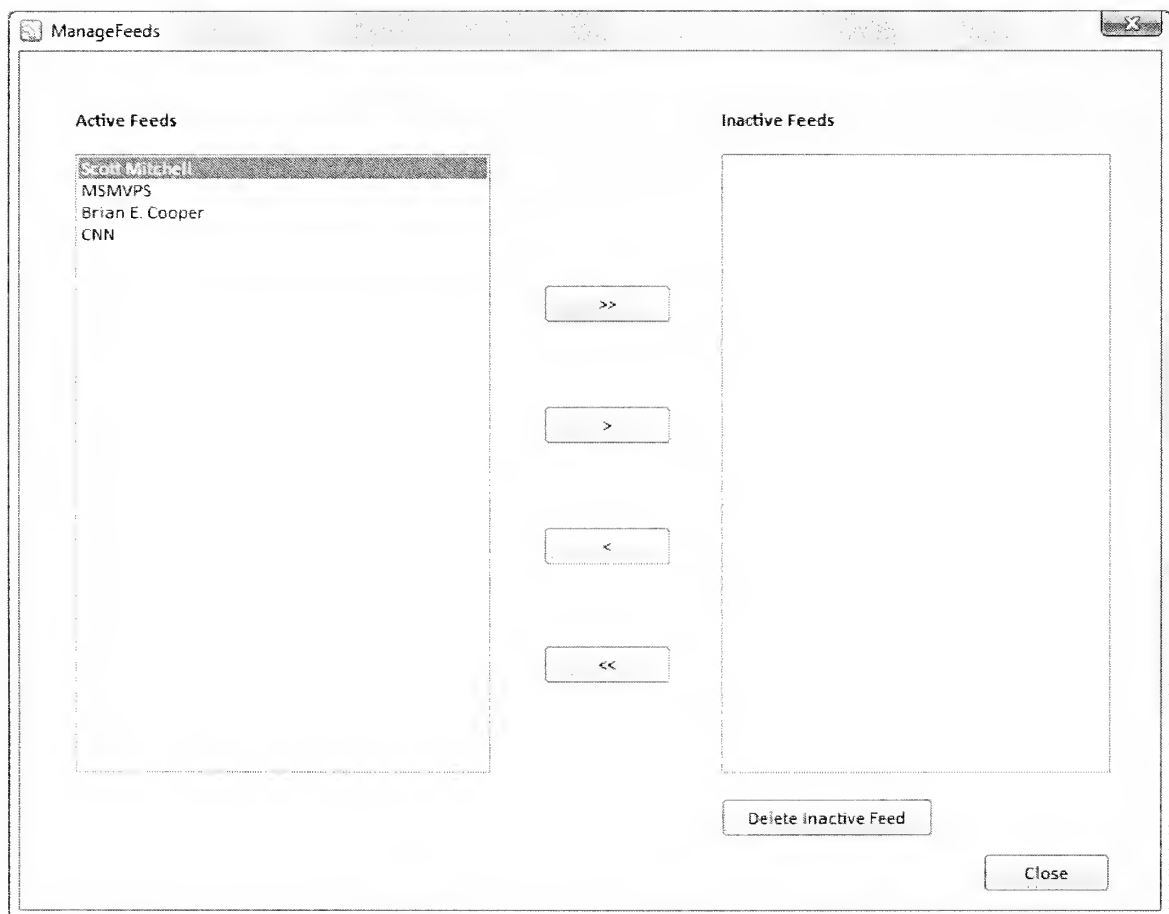


Figure 15. Managing Feeds Screen

7.10 Refresh All/This Feed(s)

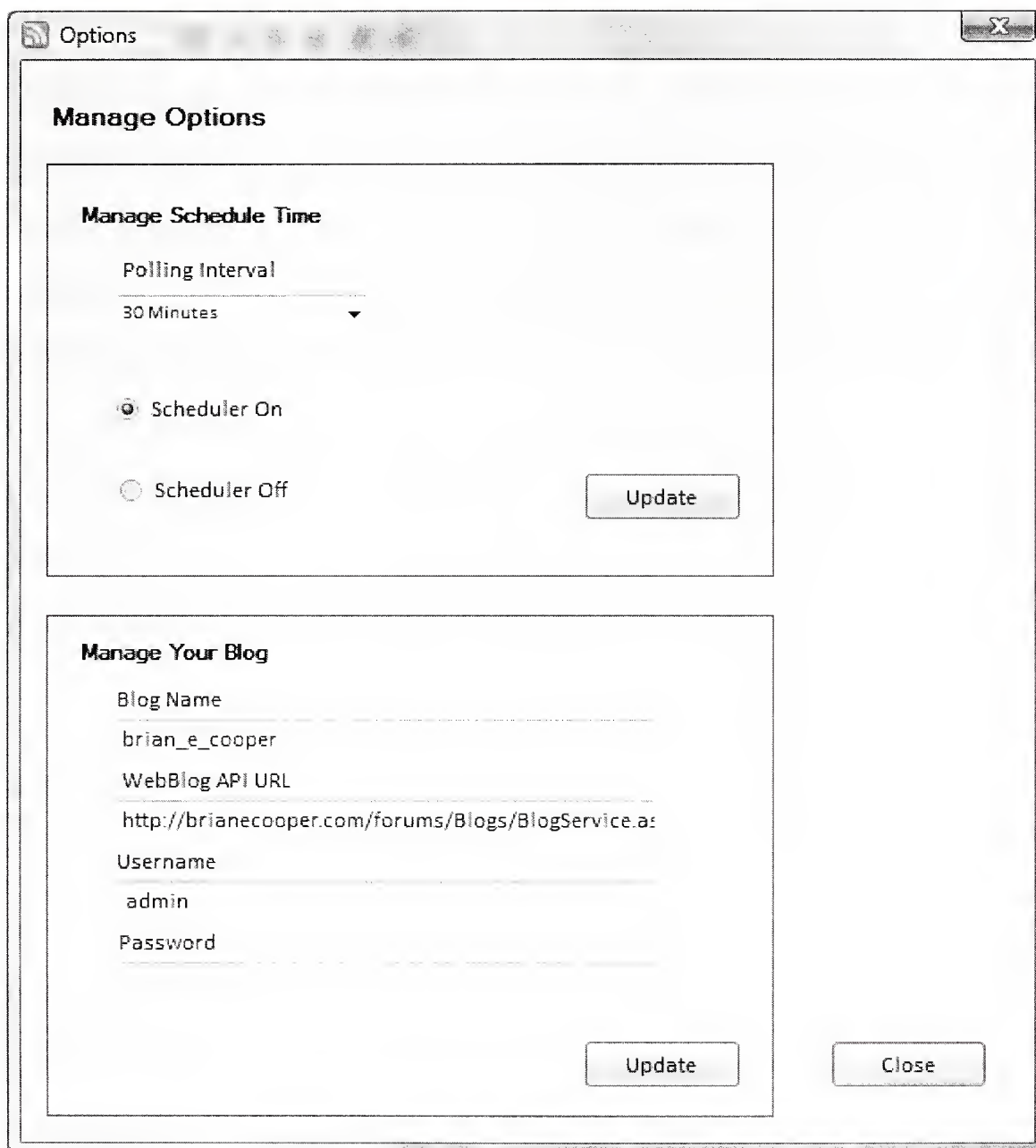
Feeds get imported by default in FeedWorks from a set schedule, but a user may also import feeds through manually choosing to refresh all or just certain feeds. All feeds may be refreshed by clicking the “Refresh All Feeds” under “File” in the main menu or by right-clicking a root node in the feed hierarchy and choosing “Refresh All Feeds.” Refreshing all feeds starts the import process for all feeds and will check for new content and filter imported content. A user may also just refresh a specific feed by right-clicking the child node in the feed hierarchy and choosing “Refresh This Feed.” Choosing to just import a specific feed will begin the import process of just that specific feed, but it will not update any other active feed.

8. Polling and Filtering

Polling and Filtering are two of the primary functionality pieces of FeedWorks. Polling occurs on a set schedule determined by the user but set to every 30 minutes by default. The scheduled time reflects the time that the application checks for new content on all of subscribed feeds. Filtering is the process of creating a filter and attaching it to a feed or groups of feeds. Feeds that have a filter attached then get parsed for keywords and depending on the rules of the filter to either include or exclude the content that posses the keyword. Once the content is filtered, all content matching the rules of the filter then gets imported into the database.

8.1 Polling Schedules

FeedWorks imports feed content on a specified schedule. The application polls on an interval that is set within the options screen of the application. The polling scheduler may also be turned off if the user wishes to not poll for new content. The polling interval has many different options but it may be set up to its fastest interval of every fifteen minutes or even up to its slowest interval of once a day (see [Figure 16](#)). An internal timer checks every fifteen minutes to see if the elapsed time since the last import meets the schedules polling interval. If the condition is met, then the import process fires and if not the timer continues to check until the condition is successfully met.



The screenshot shows a window titled "Options" with a close button in the top right corner. The window is divided into two main sections:

- Manage Options:** This section contains a "Polling Interval" dropdown menu set to "30 Minutes", two radio buttons for "Scheduler On" (selected) and "Scheduler Off", and an "Update" button.
- Manage Your Blog:** This section contains five text input fields: "Blog Name" (brian_e_cooper), "WebBlog API URL" (http://briane.cooper.com/forums/Blogs/BlogService.as), "Username" (admin), and "Password" (empty). It includes "Update" and "Close" buttons at the bottom right.

Figure 16. Managing the Schedule Screen

8.2 Check for new content with HTTP headers

FeedWorks checks for new content from all subscribed feeds each time the polling interval is reached. When FeedWorks polls for new content, it checks to make sure new content exists instead of downloading all content regardless of if it is new or

old. To accomplish this checking for new content, FeedWorks enlists help from the HTTP protocol by using its last-modified property [7]. When FeedWorks first downloads an RSS file, it stores the date that it downloads the content. Once FeedWorks polls the feed for more content, it sends a request for the last-modified information within an HTTP Request. If the content has changed since the last download of the RSS file, then the source server will return the new RSS File. If the content has not changed since the last download , then FeedWorks does not download the content at all and moves on to checking the next subscribed RSS feed or quits the schedule polling [8]. This method helps save bandwidth and resources by checking first if new content even exists before downloading it.

8.3 Filtering Feeds

Feeds are filtered in FeedWorks as they are imported into the application. Once the content source returns the RSS file to the client, the filter process begins to parse through each feed article to search for keywords. As keywords are matched with the content, the content is either written to the database or discarded based on the specific rules of the filter. A filter may have many different keywords that it searches for and a feed may have many different filters. Filters may be applied directly to a feed or to a group of feeds. Grouping feeds allows many different feeds to be categorically grouped into a logical container. The competitive analysis completed during research for this project presented the fact that most feed readers download all content and then just filter it to present the user a view of the content but never actually delete the content from disk. FeedWorks filters the feeds before writing them to disk to allow the application to

save on disk space and resources since content that is not wanted is filtered before writing the content to disk. Users may use FeedWorks to scour the Internet and import multitudes of feeds into FeedWorks to have the application filter out all content that is not wanted and group the content that is wanted into effective categories.

8.4 Creating and Managing Keyword Search Filters

Filters are created in FeedWorks using the Filter Manager Screen (see [Figure 17](#)). Filters may be created by naming the filter and clicking the create button. Once a feed is created, keywords may be added to a keyword list. The keyword list may be used for inclusion or exclusion which will determine if the feed looks for feed content containing the keywords or for feed content that does not contain the keywords. The Filter Manager also displays all feeds that are attached to the chosen filter. Feeds may be deleted from a filter from the Filter Manager but are added by right-clicking a feed from the main screen.

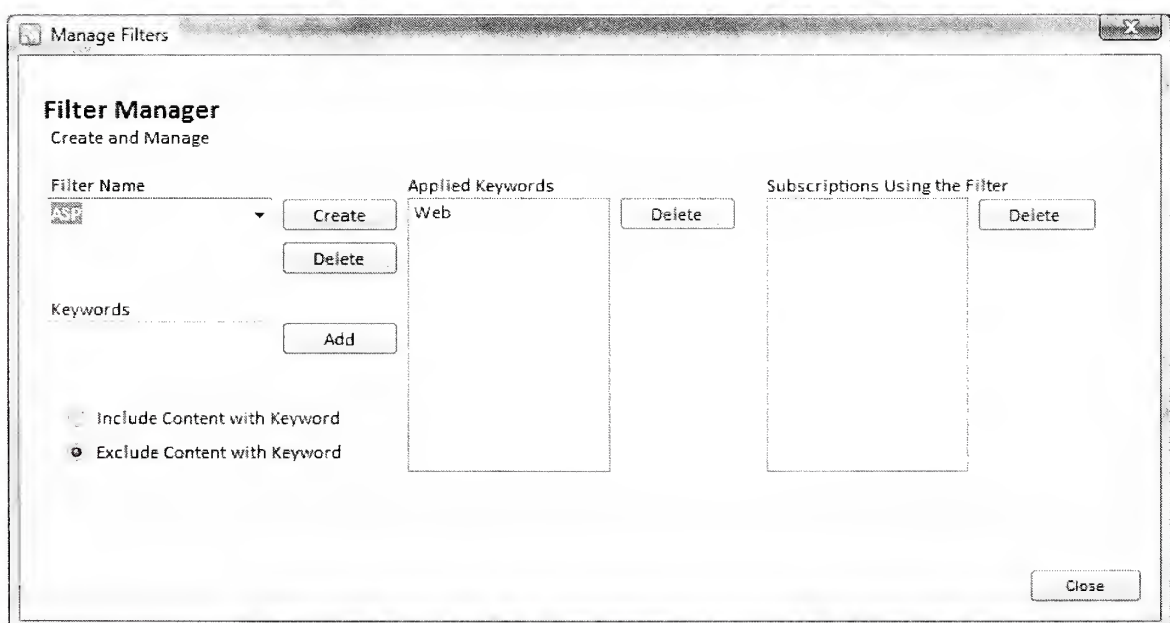


Figure 17. Mange Filters Screen

8.5 Applying Filters to Subscriptions

Active feeds may be applied or deleted from a filter. Filters may contain any number of feeds. Filters are applied by right-clicking a feed from the main screen's tree view and choosing "Apply Filter". Filters are deleted from feeds from the Filter Manager screen by clicking "Edit" then "Filters" from the main menu.

8.6 Filtering by Including Keywords

FeedWorks filtering process searches for keywords within each article that is imported during the scheduled polling for new content. The keyword parser searches the title and body of each article for non-case sensitive matches. Once a match is found, the parser checks if the filter has been designated an inclusion or exclusion filter. If the filter is an inclusion filter, then the keyword match is an article that it wants to keep since an inclusion filter is a filter that is looking for articles that contain the aforementioned keyword. Likewise, a match that is an exclusion filter will not import the current article because an exclusion filter looks for keywords that the application does not want to import. All articles that are not imported during the filter process are discarded and never written to disk.

8.7 Search/Filter Algorithm

FeedWorks applies filters and parses feed articles before the content is written to the database. To achieve more efficient searching of text for keywords, Feedworks

employs the Boyer-Moore string searching algorithm. The Boyer-Moore string searching algorithm was developed by Bob Boyer and J Strother Moore in 1977 [18]. The algorithm is considered to be the most efficient string searching algorithm in usual applications [16]. The Boyer-Moore algorithm preprocesses the keys that are being searched for in a given text and searches for those keys from right to left.

Arthur Gittleman in his research paper, “Predicting String Search Speed”, best explains the Boyer-Moore algorithm:

“The Boyer-Moore algorithm also uses a match rule, but vastly improves performance by matching the pattern right-to-left and using an occurrence rule. To understand the occurrence rule, suppose the text character t fails to match the pattern character p_i . The smallest shift that could possibly lead to a match aligns t with its rightmost occurrence in the first $i-1$ pattern characters. The Boyer-Moore occurrence rule finds the rightmost occurrence, p , in the entire pattern, of t . If t does not occur then we can shift the pattern to the right of t . If p is to the left of t in the current alignment, then the rule requires a shift to align p and t . If p is to the right of t , we do not know the next leftward occurrence of t in the pattern so we simply shift one position to the right. The algorithm thus simply shifts by the larger of the two shifts required by these rules.”[17].

Tests were performed using data from Feedworks against many different search algorithms including Boyer-Moore and the common .NET framework string search method `String.IndexOf` (see [Figure 18](#)) [19]. The first of these tests showed that the

Boyer-Moore was much more efficient than the `indexOf` but the number of matches were very different. The matches were different because Boyer-Moore is case sensitive where `indexOf` is not.

```
C:\>BoyerMoore_Demo.exe c:\xmlfile2.xml Blog
Finding all occurrences of Blog in c:\xmlfile2.xml (55475 characters in length)
String.indexOf found 434 matches in 0.0016642 seconds
Horspool found 35 matches in 0.0003076 seconds
Boyer-Moore found 138 matches in 0.0003435 seconds
Turbo Boyer-Moore found 138 matches in 0.0006669 seconds
Apostolico-GiancarloMatch found 138 matches in 0.0008714 seconds
```

Figure 18. Mange Filters Screen

An additional test was created to test all algorithms with all content being changed to upper case (see Figure 19). The Horspool algorithm was the most efficient but its matches were considerably less. The Boyer-Moore now found the same number of matches as the .NET `indexOf` but in considerably less time.

```
Finding all occurrences of Blog in c:\xmlfile2.xml (55475 characters in length)
String.indexOf found 434 matches in 0.0020396 seconds
Horspool found 107 matches in 0.0006054 seconds
Boyer-Moore found 434 matches in 0.0007044 seconds
Turbo Boyer-Moore found 434 matches in 0.0009741 seconds
Apostolico-GiancarloMatch found 434 matches in 0.0011903 seconds
C:\>
```

Figure 19. Mange Filters Screen

9. Knowledge Bases

Knowledge bases in FeedWorks are used to group archived content into logical archived categories. The functionality is much like creating a folder for archived emails pertaining to one particular subject. This functionality allows a FeedWorks user to have content imported from many different feed sources, then filtered based on keywords and finally archived into knowledge bases that might even reflect the keywords that filter the content into the aggregator. For instance, a FeedWorks user may have five hundred different feeds that related to all things Java. The user may want to filter those feeds for just information regarding Java Server Pages. Once the feeds are imported into FeedWorks, the user may then archive the feed content to a knowledge base entitled “Java Server Pages.” This information now has been taken offline and stored in a repository that is indexed and may be searched or browsed as a knowledge base of content deemed worthy of saving by the user. Knowledge bases in FeedWorks represent a way to bring specific subject matter from the Internet, parse it for keywords to filter out unwanted content and then build a knowledge base of content that is useful to the content consumer.

9.1 Creating a Knowledge Base

Knowledge bases may be created in FeedWorks by clicking on “Knowledge Bases” within the “Edit” option of the main menu. Knowledge bases may be named and created from this screen or current knowledge bases may be chosen from a drop-down menu and deleted ([see Figure 20](#)).

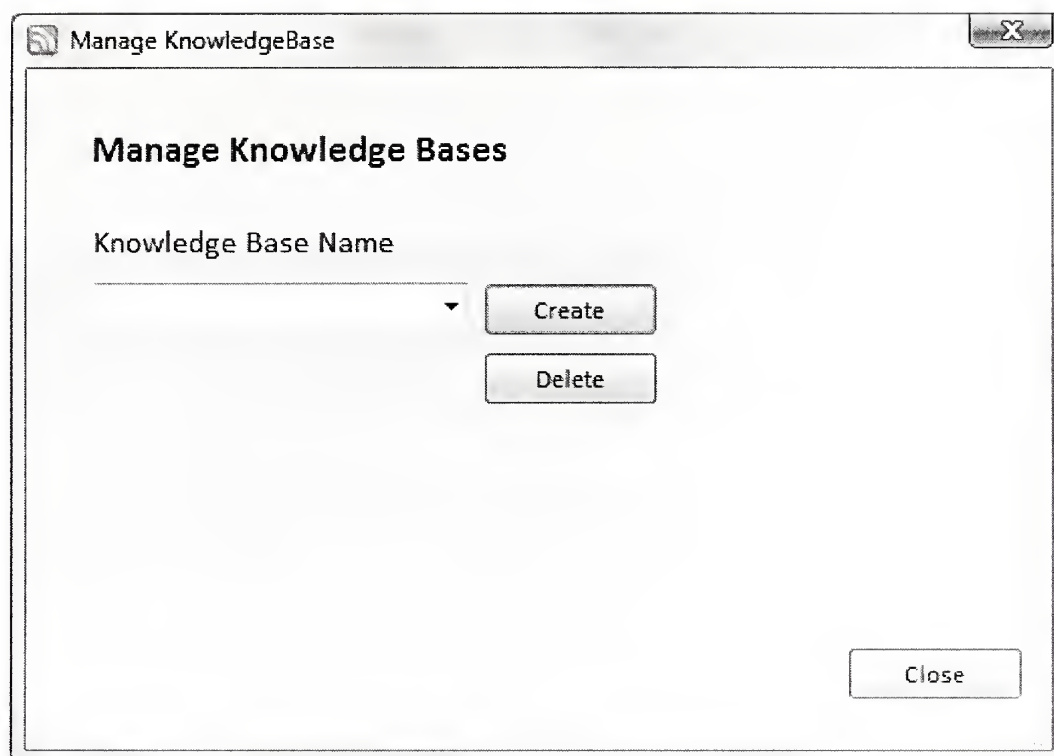


Figure 20. Manage Knowledge Bases Screen

9.2 Searching Knowledge Bases

FeedWorks allows searching of all feed content regardless of if that content is located within the active feeds, archives or within a knowledge base. Searching within any of the different content stores such as the active feeds, archives, or knowledge bases only requires that the view of the main screen be changed to the corresponding view that the search needs to be queried against as seen in [Figure 21](#). Once the view is changed from the menu, then all subsequent search queries will be applied to only content within that view. Changing the view from active to archived content helps to eliminate the clutter of possible large amounts of content and searches that may become less useful when searching within large data stores that could span multiple years. The

most useful organization method of archived content is to keep archived content categorically organized in knowledge bases.

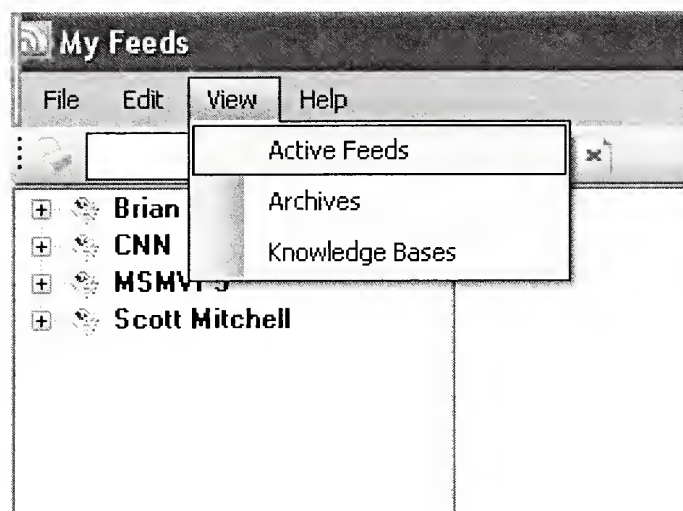


Figure 21. Searching Knowledge Bases

9.3 Future Functionality of RSS Knowledge Bases

FeedWorks' knowledge bases can be extended to create an even more robust and useful solution by combining the knowledge base functionality and the automatic posting to the user's blog functionality. Combining these two pieces will allow a solution for posting knowledge bases to the web through a user's blog. The user's blog must first have a category set up which is common to most web blogs. This category may be named something that identifies it as a container of articles about a specific subject (e.g. "ASP.NET Knowledge Base"). Once the category is established, the user must use the blog configurations options within FeedWorks to set up his or her blog so that FeedWorks may interface with the blog's MetaBlog API. This configuration will set up FeedWorks to be able post feed articles from the application to the user's blog.

After the user's blog and FeedWorks are both correctly configured, the user may go to a specific knowledge base and right-click the root node and click "Export Knowledge Base to Blog." This will initiate a screen that will allow the user to choose a category that is presented by the MetaBlog API from his or her blog. Once a category screen is chosen, the user clicks a button to begin the import process. A progress bar screen (see [Figure 22](#)) will then appear that will send feedback to the user as all articles are submitted to the user's blog under the chosen category.

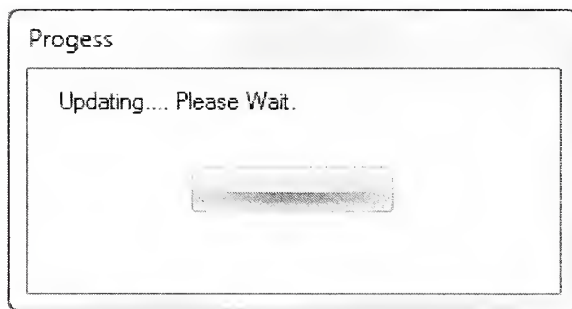


Figure 22. Progress Bar

10. Installing and Uninstalling FeedWorks

FeedWorks may be installed on Windows based client machines using a Microsoft Installer package that will install the complete application including SQL Server databases. The installer checks for all prerequisites and prompts the user for information regarding their specific installation. Once all necessary information is gathered, the installer completes the installation and the application is ready for use. A detailed installation guide is available within the appendix of this document.

Uninstalling FeedWorks is accomplished by removing the application from the “Add/Remove Programs” within the system control panel. Uninstalling the application removes all files/folders, data and databases provisioned by FeedWorks.

11. Testing

Testing of FeedWorks was performed using test cases to accomplish functional testing. Functional testing tests the product according to the programmable work [14]. Functional testing selects valid and invalid input and determines if the generated output is correct [15]. A test case is a software testing document, which consists of event, action, input, output, expected result and actual result [14]. Test cases were used on FeedWorks to determine a normal course of user action to accomplish a goal and if that goal was achieved successfully with either valid or invalid inputs. Test cases included adding feeds, managing feeds, deleting feeds among many others.

Parsing algorithm testing was also performed to test the efficiency of importing and parsing of RSS feeds. Each algorithm evaluated was tested based on time to complete a large feed and its ability to scale testing an enormous amount of feed data.

12. Conclusion

FeedWorks has been designed to offer a better RSS aggregator that effectively and efficiently polls, downloads and filters RSS content while using a minimal amount of bandwidth and resources. The application accomplishes these goals by checking for subscribed feed content at specific intervals that are determined by the application user. The application polls for new content by checking the HTTP response header information generated by the remote server hosting the RSS feed content. By checking the header information, FeedWorks saves bandwidth by asking for the feed content only if the content is newer than the last time FeedWorks downloaded that specific feed's content.

In the case that the feed content is not newer than the previously downloaded content, FeedWorks does not download content at all. If the content is newer than the previously downloaded content, FeedWorks downloads the content and begins to parse the information. FeedWorks utilizes a specific search algorithm to parse the content as efficiently as possible as it applies all filters created by the application user. During the parsing of the new content, FeedWorks either keeps or discards the content based on the filter rules. All content that passes the filter rules is written to disk into a SQL Server database to await consumption by the application user.

FeedWorks filters content before writing it to disk, as opposed to many other aggregators researched in this thesis, to save on system resources in the form of IO and disk storage space. Content that is discarded by FeedWorks does not get written to disk and is discarded within system memory before it takes up IO time and disk resource by being stored within the database even though the content has been denoted as unwanted.

FeedWorks employs these methods such as checking HTTP header information, efficient parsing algorithm, and filtering out unwanted content before writing to disk to save on bandwidth and system resources. Saving bandwidth and resource consumption within the FeedWorks application creates a more scalable environment, as the amount of RSS feeds and consumers of RSS content grow within the private and public sectors.

12.1 Future Work

One problem with using a client application for FeedWorks is that RSS feeds are local to the machine in which FeedWorks is installed. One way to get around this is to export the OPML file and import it on another machine in which FeedWorks is installed, but this doesn't provide true synchronization. Some RSS companies such as NewsGator provide an online aggregator that their client products such as Feed Demon can use to keep users' RSS feeds synchronized between multiple clients. This functionality is nice but would require a network infrastructure to be able to synchronize clients. There are many free options that could be used to accomplish the same goal of RSS feed synchronization.

Del.icio.us is a social bookmarking site that offers a service to store, tag and share favorite bookmarks with other people [11]. FeedWorks could be altered to use del.icio.us to store RSS feeds to the del.icio.us site as web storage solution to store links, synchronize with other clients, and share with friends. All that would be required is a del.icio.us account and as RSS feeds are added to FeedWorks they would also log into the API of del.icio.us and store the RSS feeds under a specific RSS tag and either set the link to share openly or private. FeedWorks could then be configured to use the

RSS feed that del.icio.us exposes of the bookmarked RSS feeds. Integration with del.icio.us would provide synchronization without the overhead of a network infrastructure to provide the same technology that is already provided by solutions such as del.icio.us.

FeedWorks also does not currently provided functionality for handling podcasts, which are defined by Wikipedia as, digital media files that are distributed over the Internet using syndication feeds, “for playback on portable media players and personal computers” [12]. In a future release, it is planned to have FeedWorks handle enclosures inside of RSS feeds. FeedWorks would have the ability to download enclosed content such as podcasts for consumption by the application user.

Further consideration should also be given to methods to save bandwidth consumption such as checking feeds for the use of data compression methods and provide means in the application to decompress the feed after transport. These methods could further save bandwidth but possibly create overhead within the application when decompressing data.

12.2 Future Testing

Testing should continue for FeedWorks as new functionality is added to the application. As possible future functionality is added such as the RSS feed synchronization the application should be tested to determine its efficiency synchronizing the RSS feeds as bookmarks. Testing should be performed to determine the speed of the synchronization with a low amount of RSS feeds as well as how scalable the solution is once the application needs to synchronize hundreds or thousands

of RSS feeds. The metrics gathered on these tests should also be compared to such RSS aggregators that also provide RSS feed synchronization such as FeedDemon (<http://www.newsgator.com>).

Once podcasts and other enclosures are added into the application, testing should be performed on the efficiency of downloading attached files to the desktop. Since enclosures could slow down the application's rate of refreshing feeds, the enclosures may need to be downloaded in a separate process outside of the actual RSS feed. Testing of different methods and approaches would render a best approach to downloading enclosures without losing application efficiency or creating a scalability issue. Again the testing done in this area should be weighed against various competitor applications to determine how the application stands up on a side-by-side performance test.

Appendix A: Installation Guide for FeedWorks

Installation for FeedWorks is performed by a Windows Installer package. Windows Installer is defined by Wikipedia as, “The Windows Installer (previously known as Microsoft Installer, codename Darwin) is an engine for the installation, maintenance, and removal of software on modern Microsoft Windows systems” [10].

A user installing FeedWorks should use the install.msi that is the package created with Windows Installer and is included with the FeedWorks software. Clicking on the provided setup.msi will begin the installation process. Next the installation will prompt to enter the name of a SQL Server as well as the name and password of an account that has rights to create databases on the denoted SQL Server. SQL Server is currently required for installation of FeedWorks.

Once the SQL Server information is entered and verified, the installer will prompt the user to designate FeedWorks’ installation location. The install location represents the physical location where the application and all related files will be installed except for the FeedWorks database which is installed in the default location for all SQL Server databases which is “<Program Files>\Microsoft SQL Server\MSSQL.1\MSSQL\ “ [13]. After the install location is chosen, the installation will copy all necessary files and apply all applications configurations which will mark the completion of the installation.

FeedWorks may be uninstalled at any time by using the add/remove functionality located within the Microsoft Windows Control Panel. A removal of the application will delete all application files, folders and it will drop the SQL Server database.

References

1. O'Reilly, Dennis. "Web-User Satisfaction on the Upswing". 2004. PC World. 2 February 2007. <<http://www.pcworld.com/article/id,116060-page,1/article.html>>.
2. King, Andrew. "The Evolution of RSS". WebReference.com. 5 February 2007. <<http://www.webreference.com/authoring/languages/xml/rss/1/index.html>>.
3. Miller, Rich. "RSS Focus Shifts to Bandwidth Management". 24 October 2004. NetCraft. 5 February 2007. <http://news.netcraft.com/archives/2004/10/20/rss_focus_shifts_to_bandwidth_management.html>.
4. Shucha, Bonnie. "Too Much Information: Filtering RSS Feeds". December 2006. Connecting. Volume 7, Issue 1. Page 15. 5 February 2007. <http://www.aallnet.org/sis/cssis/newsletter/2006/2006_December.pdf#page=1>.
5. Murach, Joel. Murach's C# 2005. Mike Murach and Associates INC. 2006.
6. Winer, Dave. "RFC: MetaWeblog API". 2002. XML-RPC. 1 April 2006. <<http://www.xmlrpc.com/metaWeblogApi>>.
7. Connolly, Dan. "Hypertext Transfer Protocol -- HTTP/1.1". 1 September 2004. World Wide Web Consortium. 1 April 2007. <<http://www.w3.org/Protocols/rfc2616/rfc2616.html>>.
8. Miller, Charles. "HTTP Conditional Get for RSS Hackers". 21 October 2002. The Fish Bowl. 1 April 2007. <http://fishbowl.pastiche.org/2002/10/21/http_conditional_get_for_rss_hackers>

9. "Microsoft's Windows dominates the OS market on the web according to OneStat.com". 14 August 2006. Onestat.com. 1 April 2007. <
http://www.onestat.com/html/aboutus_pressbox46-operating-systems-market-share.html>.
10. "Windows Installer". 5 April 2007. [Wikipedia: The Free Encyclopedia](#). 7 April 2007.
<http://en.wikipedia.org/w/index.php?title=Windows_Installer&oldid=120523515>.
11. del.icio.us. 7 April 2007. <<http://del.icio.us/about/>>
12. "Podcast". 7 April 2007. [Wikipedia: The Free Encyclopedia](#). 8 April 2007.
<<http://en.wikipedia.org/w/index.php?title=Podcast&oldid=121018327>>.
13. "File Locations for Default and Named Instances of SQL Server 2005 ". 5 December 2005. [SQL Server 2005 Books Online](#). 15 April 2007.
<<http://msdn2.microsoft.com/en-us/library/ms143547.aspx>>.
14. "Software testing". 16 April 2007. [Wikipedia: The Free Encyclopedia](#). 16 April 2007.
<http://en.wikipedia.org/w/index.php?title=Software_testing&oldid=123251989>
15. "Black box testing". 13 April 2007. [Wikipedia: The Free Encyclopedia](#). 16 April 2007.
<http://en.wikipedia.org/w/index.php?title=Black_box_testing&oldid=12247635>

16. Christian.Charras, Thierry.Lecroq . “Boyer-Moore algorithm”. 14 Jan 1997.
Instut d'elxtronique et d'informatique - Gaspard-Monge. 21 April 2007.
<<http://www-igm.univ-mlv.fr/~lecroq/string/node14.html>>
17. Gittleman, Arthur. "Predicting String Search Speed". 10 Jun 1996. Association
for Computing Machinery. 21 April 2007.
<<http://www.acm.org/jea/ARTICLES/Vol1Nbr2/index.html>>
18. "Boyer–Moore string search algorithm”. 30 March 2007. Wikipedia: The Free
Encyclopedia. 21 April 2007.
<http://en.wikipedia.org/w/index.php?title=Boyer%E2%80%93Moore_string_search_algorithm&oldid=119109024>.
19. Daniels, Carl. “Boyer-Moore and related exact string matching algorithms”. 17
Jan. 2006. The Code Project. 30 March 2007.
<<http://www.codeproject.com/cs/algorithms/BoyerMooreSearch.asp>>

