

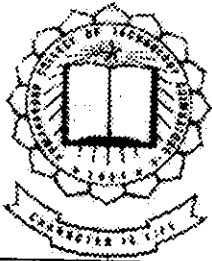
# WEB CRAWLER

SRM SYSTEMS AND SOFTWARE LIMITED

P-1068

## PROJECT REPORT

Submitted In Partial Fulfillment Of The Requirements For The Award Of The Degree Of  
Master Of Science In Applied Science - Software Engineering  
of Bharathiar University, Coimbatore.



Submitted By

Mr. N. Dev Anand

Reg.No: 0037S0087

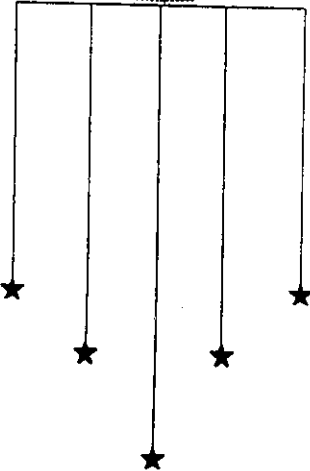
Guided By

Mrs. A. Muthukumar. M.C.A., M.Phil.,  
Asst. Professor, Kumaraguru college of Technology,  
Coimbatore - 641 006.

Ms. R. Sowmya

Software engineer

SRM Systems And Software Limited  
Chennai.



Department of computer Science and Engineering  
Kumaraguru College Of Technology  
Coimbatore - 641 006.

# CERTIFICATE



**Kumaraguru College Of Technology**

Department of Computer Science and Engineering  
Coimbatore - 641 006.

This is to certify that the project work entitled  
" Web Crawler "

Done By

Mr. N. Dev Anand

Reg.No: 0037S0087

Submitted to the partial fulfillment of the requirements for the award of the  
Degree of Master of Science in Applied Science - Software Engineering of  
Bharathiar University, Coimbatore.  
During the Academic Year 2003 - 2004

Signature of Guide

Head of the Department

Certified that the candidate was examined by us in the project in the Project  
Work Viva Voce Examination held on 29/9/03 and the University  
Register Number was 0037S0087.

Internal Examiner

External Examiner

# SRM SYSTEMS AND SOFTWARE LIMITED

24, G.N. Chetty Road, T.Nagar, Chennai - 600 017.  
© : 91 - 44 - 8250771, 8258757, 8269471 Fax : 91 - 44 - 8283359  
E-mail : srm@srmsoft.co.in Web Site : <http://www.srmsoft.com>  
Regd. Off : 2, Veerasamy St., West Mambalam, Chennai - 600 033.



23.09.2003

## CERTIFICATE

This is to certify that the project work entitled "WEB CRAWLER" was Analyzed, Designed and Developed by Mr. N.DEVANAND of KUMARAGURU COLLEGE OF TECHNOLOGY (CBE), submitted in partial fulfillment of the requirements of degree of 4<sup>th</sup> Year M.Sc (S.E) has been carried out in our organization from June 2003 to Sep 2003. This project has been developed using VC++.

We wish him success in all his future endeavors.

**For SRM Systems And Software Limited**

A handwritten signature in black ink, appearing to read "Saveri".

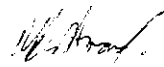
**Manager-Projects**

## DECLARATION

I hereby declare that the project entitled "WEB CRAWLER" submitted to Bharathiar University, Coimbatore, as the project work of Master of Science in Applied Science - Software Engineering, is a record of original work done by me under the supervision and guidance of Ms. Sowmya, Software Engineer, SRM Systems and Software LTD and Mr. A. Muthukumar, M.C.A., M.Phil., Asst. Professor, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore. And this project work has not found the basis for the award of any Degree/ Diploma/ Associate ship/ Fellowship or similar title to any candidate of any University.

Place: COIMBATORE

Date: 26-9-03



Signature of the Student

Countersigned By

Project Guide  
(Mr. A. MuthuKuamr)

## ACKNOWLEDGEMENT

I would like to begin with a special note of gratitude to my sincere and heart felt thanks to our Principal Dr. K.K.Padmanaban B.Sc (Engg), M.Tech, PhD and our HOD Prof. Dr.S.Thangasamy PhD for giving me the needed encouragement for this project and carrying out successfully.

I also take an immense pleasure especially in thanking our class coordinator Mrs. S. Devaki B.E., MS., Asst. Professor, CSE department and my internal project guide Mr. A. Muthu Kumar MCA., M.Phil., Asst. Professor, CSE Department who have taken up keen interest in the success of this project.

My grateful thanks goes to Ms. Sowmya, Software Engineer, SRM Systems and Software Ltd for rendering her excellent guidance for the successful and timely completion of the project.

I also externally indebted to my parents for their unending support, love and understanding throughout the endeavor.

Above all, I owe my gratitude to God Almighty, for showering abundant blessings one me.

## SYNOPSIS

The project entitled " Web Crawler" is helps a site manager or web designer to maintain his site in an effective manner. This is user-friendly software so that any end user can use it. If we want to down load any contents from the website or download a full website easily by using this software. Before download the website the software will generate an indexed file. This also generates the table of contents. The table contents generated in this software will give us separate list of the different kinds of files. For example, it separates the multimedia files, HTML files, text files etc., in a table or tree structure format.

# CONTENTS

	Page No
1. INTRODUCTION	
1.1 Existing System and Its limitations	1
1.2 Proposed system and its advantages	1
2. SYSTEM STUDY & ANALYSIS	3
3. PROGRAMMING ENVIRONMENT	4
3.1 Hardware & Software Configuration	4
3.2 Description Of Software's & Tools Used	5
3.3 Development Environment	6
3.4 Microsoft Foundation class	10
3.5 Dynamic Link Libraries	12
4. SYSTEM DESIGN AND DEVELOPMENT	
4.1 Input Design	14
4.2 Miscellaneous Notes	15
4.3 Output Design	18
5. SYSTEM IMPLEMENTATION AND TESTING	22
5.1 System Implementation	25
5.2 Operational Documentation	25
5.3 System Testing	25
5.4 Scope and Future Works	26
6. CONCLUSION	28
7. BIBLIOGRAPHY	30
8. APPENDIX	31
A. SAMPLE SCREENS	
B. SAMPLE CODE	32
	37

# 1. INTRODUCTION

## 1.1 Existing System and Its limitations

This project involves in the Internet downloading purpose. The web crawler is mainly used in the Internet. This web crawler gets the input (URLs) from the user and search the particular website on the web server. So this software first acts as a search engine. This will produce output like in the form of word index. Web crawlers do a depth-first or breadth-first search of all of the web pages that are directly or indirectly linked to some starting page. In general the function is to, select a page that has not yet been searched. If the page is not an HTML page then discard it and go on to the next page.

## 1.2 Proposed system and its advantages

This project is mainly used for download a large amount information's or data form the website very easily. This project also helps the web site manager to maintain his website in proficient manner. The main purpose of this software is, this will generate a word index for a World Wide Web Site. The word index contains all the contents in a web site. For example, this will separates the contents into multimedia files, text files, html files, word files, script files etc., This will display the word index in a tree structure format.

First we give a project name for which data or information to be downloaded through this software. This automatically creates an area or a



directory as in the name of the project we entered. This area is used for storing the data or web site contents.

Being an excellent system side development tool VC++ was used in order to delivered improved support for reuse, registry handling tool interoperability and resource management.

Once we select a particular web site through this software, it connects this software and the web server. Once we connect this software into a web site, this display the tree structure in a flash. After that we select the particular contents or full website from the tree structure and download easily.

## 2. System Study and Analysis

Web crawler is used to download a full website. The other software's are also available in markets, they have a limitation to download up to some time or up to some links. So, if the website has many web pages it cannot be downloaded fully. The web crawler is used to over come this problem. It will download a full website. It will download the web pages and pictures, multimedia files, text file etc., separately. It can be viewed in a Tree structure. The other software's download the links from outside the given domain.

### 3. Programming Environment

#### 3.1 Hardware & Software Configuration

##### SOFTWARE REQUIREMENTS

Language	:	VISUAL C++ 6.0
Operating System	:	Windows 9x, NT

##### HARDWARE REQUIREMENTS

Processor	:	Pentium IV
CPU Speed	:	1.7 GHZ
RAM	:	128 MB
CD ROM	:	56X
Hard Disk	:	40 GB
Keyboard	:	108 keys.

##### ADDITIONAL REQUIREMENTS

Internet Networking.

## 3.2 Description Of Software's & Tools Used

### Visual C++ 6.0

The major purpose of Visual C++ is to support the newest Microsoft technologies, such as the New Active Desktop in Internet Explorer 4.0. With Visual C++ 6.0 we can develop for Internet Explorer 4.0 by accessing the power and flexibility of Dynamic HTML and the new common controls. Internet Explorer 4.0 support enables us to integrate the web directly into Microsoft Foundation Class (MFC) applications.

Visual C++ has supported writing ActiveX documents for some time. Now it supports writing ActiveX document containers, including support for printing, saving, and loading. We can seamlessly get all the functionality of programs such as Excel or Word in our MFC applications with DocObject Containment. We can easily put full - featured charts, graphs, or even Web browsers right in our applications, with full menu merge.

The Visual C++ Integrated Development Environment, or IDE is organized into four distinct areas. They are, Menu and Toolbars, Project view window, code editor, and debug window. The menu items enable we to access different options not only for the IDE but also for our overall project. Using different menu options, we can control everything from the compiler's behavior of the code editor. All the toolbars are fully dockable.

This means that they can be docked to any of the four sides of the IDE's main window. We can also "float" the toolbars anywhere on our desktop.

The debug window displays important during the project building process and while our project is executing within the Visual Studio debugger. Typically, the debug window will display error messages that occur when we compile, or build, our project. We can also write messages directly to the debug window from within our project by using the Trace Macro.

### **3.3 DEVELOPMENT ENVIRONMENT**

VISUAL C++ 6.0 includes the Microsoft developer studio Integrated Development Environment (IDE). This environment is the centerpiece of most any interaction to create visual c++ projects, including source file creation, resource editing, compiling, linking, debugging and many other useful features that will make the development tasks much simpler. Visual C++ is the first and foremost C++ compiler, made up of many components, which paves the way for efficient programming.

#### **PROJECT WORKSPACE:**

Working with developer studio is working with project workspaces. These workspace represent a particular set of projects, which represent anything form single application, to a function library, or to an entire suite of applications. Each workspace

projects that we want to group into a single workspace so that we can work closely with each separate project at the same time. The project workspace (.dsw) file is responsible for maintaining all of the information that defines our workspace and the projects that we have included in it.

## **REGISTRY:**

The Registry is a system-defined database that applications and system components use to store and retrieve configuration data.

The Registry is a hierarchically organized store of information. Each entry in the tree-like information structure is called a key. A key may contain any number of sub keys, it can also contain data entries called values. In this form, the registry stores information about the system, its configuration, hardware devices, and software applications. A registry key is identified by this name. Key names consist of printable ASCII characters except the backslash (\), space and wildcard (\* or ?) characters. The use of key names that begin with a period (.) is reserved. Key names are not case sensitive. A value in the registry is identified by its name. Value names consist of the same characters as key names. The values itself can be a string, binary data or a 32-bit unsigned value.

## PREDIFINED REGISTRY KEYS:

The registry contains several predefined keys.

The HKEY\_LOCAL\_MACHINE key contains entries that describe the computer and its configuration. This includes information about the processor system board, memory and installed hardware and software.

The HKEY\_USERS key serves as the root key for the default user preferences setting as well as individual user preferences.

The HKEY\_CLASSES\_USER key is the root key for information relating to the preferences of the current (logged in) user.

Under Windows 95, there are two additional predefined keys. The HKEY\_CURRENT\_CONFIG key contains information about the current system configuration settings. This key is equivalent to a sub key (such as 0001) of the key HKEY\_LOCAL\_MACHINE\Config.

The HKEY\_DYN\_DATA key provides access to dynamic status information, such as information about plug and play devices.

## **DIALOG BOXES:**

The user gives the input to this product through edit controls with the help of three main dialog boxes, they are Parameter dialog box, Target dialog box and Segment dialog box. Depending on the number of targets in the parameter dialog box the target dialog boxes appear. Depending on the number of segments in the target dialog class the Segment dialog boxes appear. While calling the dialog boxes for loop and Message handlers play a view important role.

## **Exception Condition:**

At the time of execution there is a chance of having some exception conditions like displaying the particular dialog box for more number of times or less number of items.

## **Exception Handling:**

In order to overcome these difficulties we go for executing the program in the Debug mode, where the programs are executed om the step mode by fixing some debug points in the suspected statements. In debug mode the loop flows are very easily traced and the values of the particular member variables is also seen in the watch mode. After tracing the values if there is any error in the logic or if there is any error in the variable declaration they are handled.



### 3.4 Microsoft Foundation Class (MFC)

MFC is the abbreviation for a collection of C++ classes produced by Microsoft that are called the Microsoft Foundation Classes. MFC provides an object - oriented framework those application developers can use to create Windows applications. MFC is organized as a hierarchy of C++ classes. Several high - level classes provide general functionality while the low - level classes implement more specific behaviors. Each of the low - level classes is derived from a high - level class and, thus, inherits the behaviors of the high - level class.

MFC handles many common Windows - related tasks, such as message handling and routing, in the background. Instead of having to write to the same message-handling loop in every Windows application we develop, MFC implements the message loop for us and provides easy to understand and use member functions, like `OnPaint()`, which enables us to insert code to handle the window message.

In addition to the class hierarchy, MFC also provides an application development model. This model is called the Document/View model. Document/View, or Doc/View is method of designing an application so that the application's data is separated fro the user interface elements. This allows the two parts of the application to stand on their own, enabling the programmer to make changes to one without having to make drastic changes to the other.

## Browser ActiveX Control

To present the search results in a well defined format as well as provide the user the capability of browsing the results easily. With this goal in mind, we start to think of using the web browser ActiveX Controls. By incorporating it to our application, the user will have all capabilities with the typical web browser. The search results can be shown in the HTML format. When the user browsing the search results and clicking on a link, the document will be downloaded from the Internet directly and presented to user neatly.

In the final version of the project, the view window contains two Web Browser controls that are sized to occupy the entire client area. When the user clicks an item in the display (left-hand) control, the program intercepts the command and routes it to the target (right-hand) control. To create the ActiveX Controls at run time, we mapped the WM\_CREATE message for the view window. The message handler function will call the embedded control class's Create member function that indirectly displays the new control in the view window.

In order to get access to the Web Browser controls, we need to add data members, which are objects of the wrapper class for the ActiveX control, to the view class. Every ActiveX Control has some predefined events that can be invoked when needed. The event mapping is done in the same

way as mapping window messages and command messages from controls. More specific, an event sink map is maintained to connect the mapped events to their handler functions. Two event handler functions are used: `OnBeforeNavigate2 ()` and `OnTitleChange ()`. Because the prototypes of these functions in VC++ 6.0 are different with that in VC++ 6.0, the data type conversion from the VARIANT type to CString is necessary.

### 3.5 Dynamic Link Libraries

A Dynamic Link library (DLL) is an executable file that acts as a shared library of functions. Dynamic linking provides a way for a process to call a function that is not part of its executable code. The executable code for the function is located in a DLL, which contains one or more functions that are compiled, linked, and stored separately from the processes that use them. DLLs also facilitate the sharing of data and resources.

Multiple applications can simultaneously access the contents of a single copy of a DLL in memory. Many applications can benefit by using split into a series of main programs and DLLs. There are several linkage options. An MFC library DLL can accommodate entire C++ classes. These DLL represent classes can be used the same way that statically linked classes are used.

Dynamic linked differs from static linking in that it allows an executable module (either a .DLL or .EXE file) to include only the information needed to

function. DLLs save memory, reduce swapping, save disk space, upgrade easier, provide after - market support, provide a mechanism to extend the MFC library classes, support multi language programs, and ease the creation of international versions.

DLLs also facilitate the sharing of data and resources. Multiple applications can simultaneously access the contents of a single copy of a DLL in memory.

## 4. SYSTEM DESIGN AND DEVELOPMENT

This project is to implement a web crawler that generates a word index for a WWW site or portion of a site. The web crawler should take a starting URL (Uniform Resource Locator) and index all of the words on the HTML page. It should then proceed to the other HTML pages that are linked to by the starting URL and index those pages and so on.

Our program must accept a URL that specifies the start of our search for web pages, the name of a directory where the pages generated by our program should be placed, and a file of stop words. The form of the command line for our program should be:

```
Crawler startURL output Directory stopWordFile
```

The startURL is any valid URL. See the part URL Description for a description of valid URLs.

The limit to the set of pages that is indexed by our web crawler is the set of pages that are stored within the prefix specified by the startURL. We should ignore any links on pages that move outside the domain of the startURL. The prefix of the startURL is everything in the URL before the page name. For example, for the startURL <http://topics.com/cs/default.htm>, the prefix is <http://topics.com/cs240/>.

We will need to ensure that our program deals with cycles in the links so that it does not get into an infinite loop.

The output Directory is any valid UNIX directory name where the generated files can be placed. The home page for our generated index should be `outputDirectory/index.html`. All other file names that we generate can be anything we want as long as those files are found in output Directory.

#### 4.1 INPUT DESIGN

Web crawlers do a depth-first or breadth-first search of all of the web pages that are directly or indirectly linked to some starting page. In general the function is to:

Select a page that has not yet been searched

If the page is not an HTML page then discard it and go on to the next page.

Read in the selected page

For all text areas in the page, parse out all of the words.

For each word located that is not a stop word, place that word in the index with a reference to the page currently being indexed. Words are **case insensitive** and must be sorted in **ascending order**.

For all anchor tags with an href attribute (see the HTML definition) and for all frame tags with a src attribute, check the href or src value to see if it is a relative URL or an absolute URL. For details of how to deal with each type of link, Save the URL and any summary information about the page

repeat until there are no pages left to search. Generate the HTML pages for the index.

### **Friendly Web Crawling**

While testing our web crawler do not repeatedly hammer a particular site. For simple testing we should create a small set of web pages in a local directory and use them to work with, so that we do not slow down a real site with excessive traffic.

Many sites such as Amazon.com do not actually store many web pages. Most of the pages are generated on the fly from a database. Please do not point our web crawler at any such commercial sites. They cause a huge number of hits on such sites and will overflow the capacities of our machines with the results.

### **HTML**

One of the easiest ways to learn HTML is to look at the source for web pages that we commonly use. In Netscape we can use the Page Source item found in the View menu. There is a similar feature in Internet Explorer. One can also read about HTML. The key tags that we should understand are `<a>`, `<frame>`, `<title>` and the heading tags `<h1>`, `<h2>`,...

### **Valid URLs**

## Absolute URLs

These are the addresses that we would type into a web browser in order to visit a web page. They always have the following format:

`http://address/path/pagename`

`file: path/pagename`

The way to identify an absolute URL is that it starts with either `http://` or `file:`. If a link does not start with `http://` or `file:`, then it is treated as a relative URL. In order to be a valid URL, all absolute URLs must have the same prefix as the URL given to the program on the command line as the `startURL`. The prefix of the `startURL` is everything in the URL before the page name. For example, if the `startURL` is:

`http://cs-online.cs.byu.edu/cs240/default.htm`

then the prefix of the `startURL` is

`http://cs-online.cs.byu.edu/cs240/`

If the first characters of the link's address do not exactly match the prefix of the `startURL`, then it is not a valid link.

Also, all absolute URLs must end with either `.htm` or `.html`. If these are not the last characters of the link, then it is not a valid URL.



## Relative URLs

Relative URLs are all the links that do not begin with either `http://` or `file:`. All relative URLs must end with either `.htm` or `.html`. If these are not the last characters of the link, then it is not a valid URL. Also, a relative link is not valid if it starts with the characters `..` or `./`. If `..` are the first two characters of the relative link, or `.` is the first character of the relative link, and then the link is not valid.

In order to calculate the full address of a relative link, we should take the prefix obtained from the `startURL` and add the relative URL to the end of it. However, we must also be aware that a relative link may take us into a subdirectory. For example, if the `startURL` is `http://we.com/index.htm` and the `index.htm` contains the link: `<a href="dir/a.htm">`, then the correct address to reach that page would be the prefix plus the relative URL, or in other words: `http://we.com/dir/a.htm`. However, if the page `a.htm` contained the link: `<a href="b.htm">`, then the correct address to reach that page is the prefix, plus the subdirectory, plus the page name, or in other words: `http://we.com/dir/b.htm`

## 4.2 Miscellaneous Notes

A word is any contiguous sequence of alphabetic letters, numbers, underscore `_` or dash `-` that begins with a letter, and does not appear inside a tag. White space, punctuation or any other character that is not a letter, number, underscores or dash terminates words. Examples: `123Bob`

word because it is a contiguous sequence of alphabetic letters, numbers, underscore \_ or dash - that does not begin with a letter. Bob & Jim gives the two words: Bob and Jim. Bob's house gives three words: Bob, s, and house, since Bob and s are separated by punctuation. Bob<b>by</b>Jim gives the three words: Bob, by, and Jim, since the HTML tags count as punctuation to separate the words.

The correct definition of an HTML tag is a sequence of characters that starts with < sign followed by an alphabetic character (A-Z or a-z), without any white space between the < and the alphabetic character. All properly formatted HTML tags will end with >. If there is a < in the file that is not directly followed by an alphabetic character, then it should be treated as punctuation. HTML tags can appear anywhere inside an HTML document. All HTML tags should be ignored, except for title tags, header tags, and links.

We need only consider links that end in ".html" or ".htm". We may assume that all other links are not HTML (we know this is not true, but it simplifies the program). We may assume that all links that end in ".html" or ".htm" will lead to a page containing valid, correctly formatted HTML. Some web servers are case sensitive; so don't change the case of the links.

We do not have to worry about generating a default file name when one is not specified in a URL. Ignore all links that assume a default page. A

link assumes a default page if it ends in "/" or does not end in ".html" or ".htm".

We don't have to parse any text or HTML tags that appear outside of the `<HTML> ... </HTML>` tags in a file. We should only index words that appear inside either the title tags or the body tags of the page. Any words outside the `<BODY></BODY>` pair should not be indexed (unless they are in the page's `<TITLE></TITLE>` pair).

We should also ignore any special characters - there are two types of HTML special characters:

The first type of special character always starts with `&` (an ampersand) and ends with `;` (a semi-colon) and never has any white space in between the `&` and the `;`. For example:

`&gt; &nbsp;`

The second type of special character always starts with `&` (an ampersand) followed by a `#` (a pound sign) followed by three digits (with no white space in between) followed by a `;` (a semi-colon). For example:

`&#123;`

`&#060;`

Our project must be scalable to any size of web site.

The first step in developing a larger program like the Web Crawler is to spend some time understanding the problem that is to be solved. Once we understand the problem, start to design the classes that we will need by visualizing the operation of the program in our mind, and creating classes that perform each of the functions required by the program. For each class that we create, we should document what responsibilities the class has, and how it interacts with other classes to perform its responsibilities. This exercise will help we figure out what classes we need to write, and how those classes work together to produce a working program. Doing this type of design work before we start coding will save we time because it will help we avoid spending effort on dead-end ideas that don't work.

Once we've thought through our design the best that we can without writing any code, we should make a first attempt at implementing our design. As our implementation progresses, we will probably find that our design was incomplete or faulty in some respects. This is to be expected because some insights only come from actually implementing a design. As we proceed, make necessary changes to our design, and incorporate them into our code.

To encourage we to follow this type of design process, about halfway through the Web Crawler project we will be asked to document and turn in

the Use Cases and CRC Cards that we have developed for our program. We will be graded on the quality and completeness of our design.

### 4.3 OUTPUT DESIGN

The home page for our generated output "index.html" should contain some kind of a welcoming header and a list of all letters in the alphabet. The welcoming header should also indicate the startURL for which this is an index. Each letter should be a link to a letter page for that letter.

A letter page should contain a header, which indicates letter that this page is an index for and a link back to the home page. Each such letter page should contain all of the index words that begin with that page's letter. Remember that an index word is any word that appears in the text portion (not inside of the tags) of any HTML page and excluding any words found in the stop word file. Index words are also case insensitive. Each index word should only appear once on a letter page and should be a hyperlink to a word page for that word. The index words should appear in alphabetical order.

A word page should contain a header that indicates the word for this page as well as links to the home page and to the letter page for this word. On a word page each located page that contains that word should be listed. A located page is listed using its summary. A page's summary is the contents of the <title> tag (if there is one). If there is no <title> tag, the

it should be the text contents of the first header tag (<h1>, <h2>, <h3>, ...) in the file. If there are no <title> or header tags then the summary should be the first 100 characters of whatever text is not found inside of a tag. The page summary information should be hyper linked to the actual page itself using the URL that we stored during the web crawling phase.

AppWizard has created this SiteSnag application for we. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing our application. This file contains a summary of what we will find in each of the files that make up our SiteSnag application.

### **SiteSnag.h**

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CSnaggerApp application class.

### **SiteSnag.cpp**

This is the main application source file that contains the application class CSnaggerApp.

### **SiteSnag.rc**

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

### **res\SiteSnag.ico**

This is an icon file, which is used as the application's icon. The main resource file SiteSnag.rc includes this icon.

### **res\SiteSnag.rc2**

This file contains resources that are not edited by Microsoft Developer Studio. We should place all resources not editable by the resource editor in this file.

### **SiteSnag.clw**

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.





generated after the encryption for decrypt the data. The documentation is prepared keeping in mind users who have little or no knowledge of computers.

The operational documentation includes a document describing the overall architecture, a maintenance guide, a user manual for operations like how to data encrypted and decrypted, and how to use the key and maintain the key. The purpose of input controls and the validations for the same are explained diagrammatically. A clear picture of the system and its functionalities are thus provided.

### **5.3 System Testing**

Periodical tests were conducted during the design and implementation phases of development. Tests were conducted as per test plans, which were scheduled according to the company's policies. A detailed report on various tests conducted is given below. A Bottom - up testing methodology was adapted to test the system developed. A bottom - up test strategy starts with the fundamental components and works upwards.

While conceiving the Architectural Design of the design phase in development decomposing of the entire project into modules, the relationship between the normal text and the data they encrypted into cipher text thoroughly analyzed. In 6

design phase an analysis was conducted on the algorithm specification to implement functions, decision on data structures to represent data, and the decision of design techniques to be followed.

In the implementation phase tests were conducted according to the most widely used two stages testing process. The system tests involved Unit Testing and Module testing.

### **Unit Testing**

Unit testing was used to test individual units (i.e., Functions) in the system and ensure that they operate correctly. Alternate logic analysis and screen validations were tested in this phase to ensure optimum efficiency in the system. The procedures and functions used and their association with data were tested.

### **Module Testing**

Module testing was used to ensure that the dependable components in a module work in coordination with one another. Functional testing, performance testing and stress tests were conducted on modules independently to ensure robustness in the system developed. The various functions and their validations in module were analyzed and tested. The procedures and functions common to a module were also tested during module testing.

## 5.4 Scope and Feature Works

Because there are many commercial crawlers running over Internet, it's very easy to find the limitation of our simple crawler. For example,

### **Processing badly formatted HTML page.**

This crawler can get better result with well-formatted HTML page, especially computer-generated HTML page. Whenever encountered manually created HTML page, the crawler will miss many useful information or links. Ideally, a crawler shall have same fault-tolerant capacity like Netscape or IE, find human error and bypass it.

### **System efficiency**

Although we have a simple moniker schedule algorithm, the efficiency of this algorithm is not clear. We did not do any benchmark work. Actually, web crawler efficiency is difficult to evaluate because of involvement of server, client and network connection factors.

### **Advanced search**

Currently this crawler just supports combination of URL/title/content search. It's possible to support more advanced search criteria. Especially, if classification or other artificial intelligence is used in collected data, more satisfactory result is possible.

## Scalability

Microsoft Access Database may be enough to support an end user, we already tested database with thousands of entries and did not find obvious system degrade. However, if we want to make this database available to general web user, Oracle or some specific search engine may be necessary to support frequent accesses.

## 6. CONCLUSION

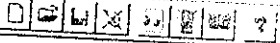
The Web crawler is the application used to download the entire website or a part of a website. These have no time limitations to download websites. For each and every project separate folder will be created in that, the files will be downloaded into that folder according to that files types the files will be saved in separate folders. By introducing artificial intelligence the search can be made more efficient and also can make advance search. Microsoft Access Database may be enough to support an end user, we already tested database with thousands of entries and did not find obvious system degrade. However, if we want to make this database available to general web user, Oracle or some specific search engine may be necessary to support frequent accesses. It will not download some links created manually so this can be rectified in the future development.

(No Project) - SiteSnagger

Project View Help



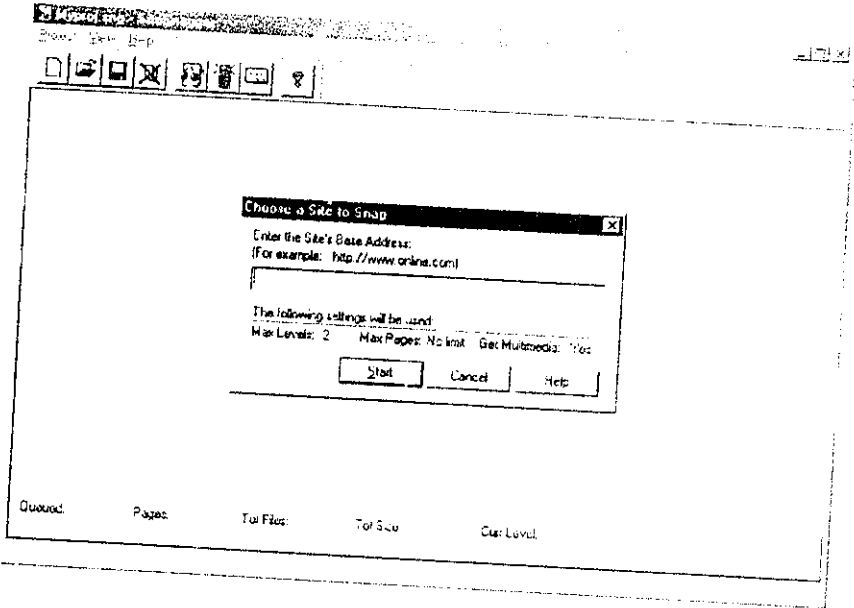
Queues	Pages	Total Pages	Total Size	Current Level
--------	-------	-------------	------------	---------------



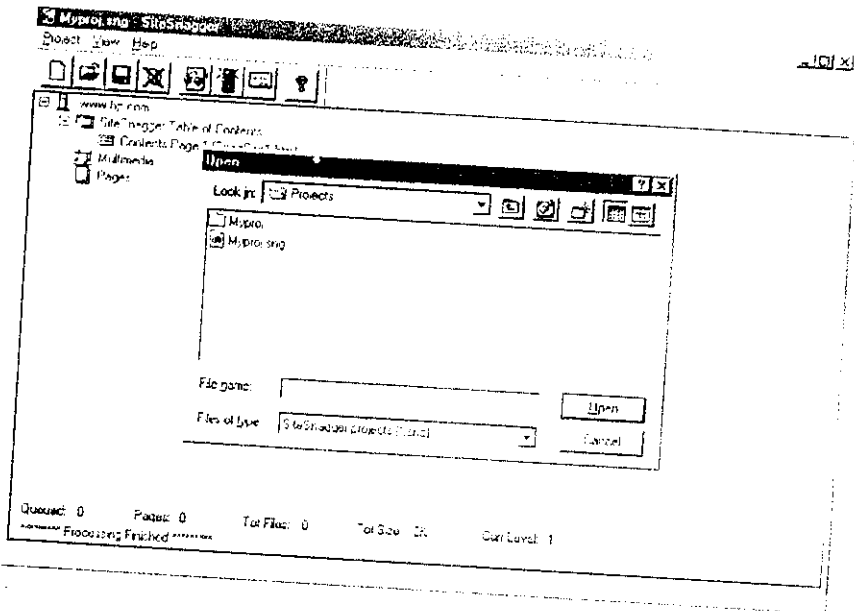
**New Project**

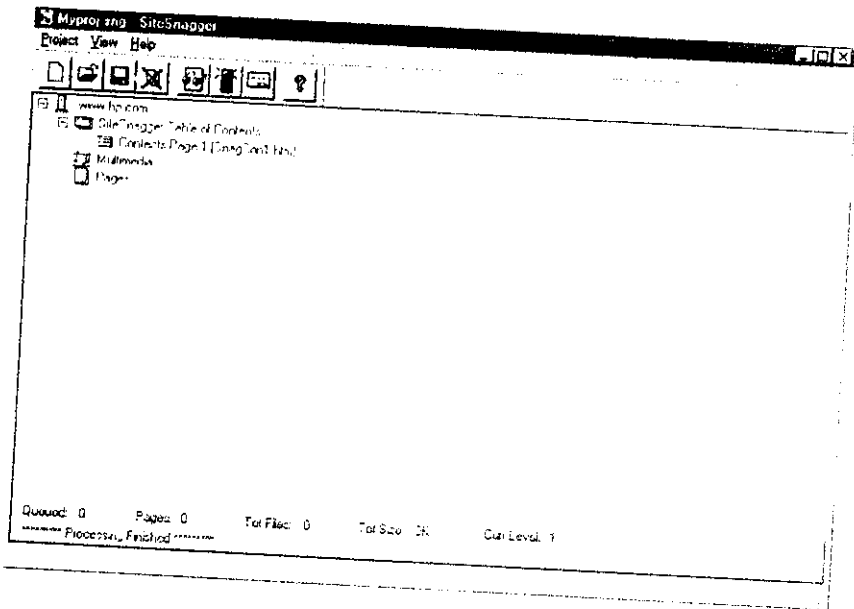
Enter a name for your new project  
(for example - MyArch)

Queued      Pages      Total Pages      Total Size      Current Level









## Coding:

```
/* SiteSnag.cpp : implementation of the CSnaggerApp class
   Implements the main application class, derived from CWinApp.
*/

#include "stdafx.h"
#include <direct.h>
#include <io.h>
#include "SiteSnag.h"
#include "Frame.h"
#include "Document.h"
#include "View.h"
#include "inet.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CSnaggerApp
BEGIN_MESSAGE_MAP(CSnaggerApp, CWinApp)
    //{{AFX_MSG_MAP(CSnaggerApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_HELP_CONTENTS, OnHelpContents)
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW,
```

```
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()
```

```
// Determine if we are running the new shell (i.e. Windows 95 or Windows
NT 4.0
```

```
// or later)
```

```
BOOL CSnaggerApp::HasNewShell()
```

```
{
```

```
    // Get the Windows Version
```

```
    DWORD dwVersion = GetVersion();
```

```
    // Windows NT?
```

```
    if(dwVersion < 0x80000000)
```

```
    {
```

```
        // Yes, Windows NT 4.0 or greater?
```

```
        if(LOBYTE(LOWORD(dwVersion)) >= 4)
```

```
            return TRUE;
```

```
    } // Windows 95?
```

```
    else if(LOBYTE(LOWORD(dwVersion)) >= 4)
```

```
        return TRUE;
```

```
    return FALSE; // Win16, 32s or NT3.x.
```

```
}
```

```
// Determine if SiteSnager is already running by using a Mutex object --
returns
```

```
// TRUE if already running, FALSE otherwise. Note that a mutex is created  
// with the name of the application
```

```
BOOL CSnaggerApp::AlreadyRunning()
```

```
{
```

```
    BOOL bFound = FALSE;
```

```
    // Try to create a mutex with the app's name
```

```
    HANDLE
```

```
        hMutexOneInstance
```

```
=
```

```
    ::CreateMutex(NULL, TRUE, _T(AfxGetAppName()));
```

```
    // Already there...means that we are already running an instance
```

```
    if(::GetLastError() == ERROR_ALREADY_EXISTS)
```

```
        bFound = TRUE;
```

```
    // Release the mutex
```

```
    if(hMutexOneInstance)
```

```
        ::ReleaseMutex(hMutexOneInstance);
```

```
    return(bFound);
```

```
}
```

```
// CSnaggerApp construction
```

```
CSnaggerApp::CSnaggerApp()
```

```
{
```

```
}
```

```
////////////////////////////////////  
////////////////////////////////////
```

```

// The one and only CSnaggerApp object
CSnaggerApp theApp;
// CSnaggerApp initialization
// Initialize the app -- check to see if is already running, make sure we're
// running the right OS
BOOL CSnaggerApp::InitInstance()
{
    // Only allow one instance of our application to execute
    if(AlreadyRunning())
    {
        AfxMessageBox(IDS_ALREADY_RUNNING,MB_OK|MB_ICONWARNING);
        return(FALSE);
    }

    // Only run with the new shell --- we need a system tray
    if(!HasNewShell())
    {
        AfxMessageBox(IDS_WRONG_SHELL,MB_OK|MB_ICONSTOP);
        return(FALSE);
    }

    AfxEnableControlContainer();
    // Standard initialization
    // If we are not using these features and wish to reduce the size
    // of our application

```

```

// the specific initialization routines we do not need.
#ifdef _AFXDLL
    Enable3dControls(); // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

int nSaveCmdShow = m_nCmdShow;
m_nCmdShow &= -SW_SHOW;
// Setup the registry key, we only use this to store the MRU list
SetRegistryKey(_T("PC Magazine\\SiteSnagger"));
LoadStdProfileSettings(); // Load standard INI file options (including
MRU)

// Register the application's document templates. Document
templates

// serve as the connection between documents, frame windows and
views.

CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CSnaggerDoc),
    RUNTIME_CLASS(CSnaggerFrame), // main SDI frame
    RUNTIME_CLASS(CSnaggerView));
window
AddDocTemplate(pDocTemplate);

```

```

// Parse the command line
if(!ProcessCommandLine())
    return FALSE;

// The one and only window has been initialized, so show and update
it.

m_pMainWnd->SetWindowPos(NULL,-1,-1,620,360,SWP_NOMOVE);
m_pMainWnd->CenterWindow();
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
return TRUE;
}

// Allows access to the applications Most Recently Used (MRU) file list
void CSnaggerApp::UpdateMRU(CCmdUI *pCmdUI)
{
    CWinApp::OnUpdateRecentFileMenu(pCmdUI);
}

// Process the command line arguments passed in
//
// The format is: sitesnag project url [options]
//
BOOL CSnaggerApp::ProcessCommandLine()
{
    CString strURL;
    CSnagOptions Options;

```



```
// Create the projects directory (if not in command line mode)
if(__argc < 2)
    mkdir("Projects");

// Create an empty document -- this ensures that the tree control
will
// always exist
OnFileNew();

// Was a document name specified?
if(__argc > 1)
{
    // Make sure that the Projects subdirectory exists, if not, then
    // we're running from the wrong directory or there is no
project
    if(access("Projects",0) != 0)
    {
        AfxMessageBox("The working directory must be the same
as the SiteSnagger program directory.",
        MB_ICONERROR|MB_OK);
        return(FALSE);
    }
}
```

```
// Add the "Projects" subdirectory if the user didn't specify a
// subdirectory location
CString strFileName;
CString strPath = CInet::SplitFileName(__argv[1],CInet::PATH);
if(strPath.IsEmpty())
    strFileName = CString("Projects\\");
strFileName += __argv[1];

// Try to open the document that the user specified on the
command line
CSnaggerDoc *pDoc = (CSnaggerDoc *) m_pDocManager-
>OpenDocumentFile(strFileName);

// Didn't work, get out
if(pDoc == NULL)
    return(FALSE);

// Do we have a URL???
if(__argc > 2)
{
    // Yes, this means that it is automatic mode
    strURL = __argv[2];

    // Set some default options (these will override the
current
```

```

// project options)
Options.nMaxDepth = 2;
Options.nMaxPages = 0;
Options.bOffsiteLinks = FALSE;
Options.bMultimedia = TRUE;
Options.bFixupLinks = TRUE;
Options.bContents = TRUE;

for(int i = 3; i < __argc; i++)
{
    LPCTSTR lpszParam = __argv[i];
    if(*lpszParam == '-' || *lpszParam == '/')
    {
        // Skip the flag
        ++lpszParam;

        switch(tolower(*lpszParam))
        {
            // Levels
            case 'l':
                Options.nMaxDepth =
atoi(lpszParam+1)%(MAX_LEVELS+1);
                if(Options.nMaxDepth <= 0)
                    Options.nMaxDepth =

```

```
break;
```

```
// Pages
```

```
case 'p':
```

```
Options.nMaxPages =
```

```
atoi(lpszParam+1);
```

```
if(Options.nMaxPages < 0)
```

```
Options.nMaxPages = 0;
```

```
break;
```

```
// Offsite links
```

```
case 'o':
```

```
Options.bOffsiteLinks =
```

```
*(lpszParam+1) == '+';
```

```
break;
```

```
// Multimedia
```

```
case 'm':
```

```
Options.bMultimedia =
```

```
*(lpszParam+1) == '+';
```

```
break;
```

```
// Fixup for browsing
```

```

Options.bFixupLinks = *(lpszParam+1) == '+';
break;
//Generate table of contents
case 'c':
Options.bContents = *(lpszParam+1) == '+';
                                break;
                                }
                                }
                                }

// Setup the document and view for snagging
POSITION pos = pDoc->GetFirstViewPosition();
CSnaggerView* pView = (CSnaggerView *) pDoc-
>GetNextView(pos);

pDoc->SetAutoMode(TRUE);
pDoc->SetOptions(Options);
CString strPath = CInet::SplitFileName(pDoc-
>GetPathName(),

CInet::DRIVE|CInet::PATH|CInet::FNAME)+"\\";
pView->ClearProject(strPath);
pView->InitTree(strURL);
pView->SetSnagging(TRUE);

```

```
        // Start the snagging operation
        pDoc->RecursiveDownload(strURL);
        pDoc->SetModifiedFlag(TRUE);
    }
}
```

```
    return(TRUE);
}
```

```
// CAboutDlg dialog used for App About
```

```
class CAboutDlg : public CDialog
```

```
{
```

```
public:
```

```
    CAboutDlg();
```

```
// Dialog Data
```

```
//{{AFX_DATA(CAboutDlg)
```

```
enum { IDD = IDD_ABOUTBOX };
```

```
//}}AFX_DATA
```

```
// ClassWizard generated virtual function overrides
```

```
//{{AFX_VIRTUAL(CAboutDlg)
```

```
protected:
```

```
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
```

```
support
```

```
//}}AFX_VIRTUAL
```

```
// Implementation
```

```
protected:
```

```
    //{{AFX_MSG(CAboutDlg)
```

```
        // No message handlers
```

```
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
```

```
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
```

```
{
```

```
    //{{AFX_DATA_INIT(CAboutDlg)
```

```
    //}}AFX_DATA_INIT
```

```
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
```

```
    CDialog::DoDataExchange(pDX);
```

```
    //{{AFX_DATA_MAP(CAboutDlg)
```

```
    //}}AFX_DATA_MAP
```

```
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
    //{{AFX_MSG_MAP(CAboutDlg)
```

```
        // No message handlers
```

```
//}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
// App command to run the dialog
```

```
void CSnaggerApp::OnAppAbout()
```

```
{
```

```
    CAboutDlg aboutDlg;
```

```
    aboutDlg.DoModal();
```

```
}
```

```
// CSnaggerApp commands
```

```
void CSnaggerApp::OnHelpContents()
```

```
{
```

```
    WinHelp(0,HELP_FINDER);
```

```
}
```

```
/*
```

```
Project.cpp : implementation of the CProjectDlg class
```

```
Implements the project creation dialog.
```

```
*/
```

```
#include "stdafx.h"
```



```

#include <ctype.h>

#include "SiteSnag.h"

#include "Project.h"

#ifdef _DEBUG

#define new DEBUG_NEW

#undef THIS_FILE

static char THIS_FILE[] = __FILE__;

#endif

// CProjectDlg dialog

// Constructor

CProjectDlg::CProjectDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CProjectDlg::IDD, pParent)
{

   //{{AFX_DATA_INIT(CProjectDlg)

    //}}AFX_DATA_INIT

}

// Data field binding for MFC

void CProjectDlg::DoDataExchange(CDataExchange* pDX)
{

    CDialog::DoDataExchange(pDX);

   //{{AFX_DATA_MAP(CProjectDlg)

    //}}AFX_DATA_MAP

}

```

```

BEGIN_MESSAGE_MAP(CProjectDlg, CDialog)
   //{{AFX_MSG_MAP(CProjectDlg)
        ON_BN_CLICKED(IDC_HELPBTN, OnHelpbtn)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CProjectDlg message handlers
// Handles OK button processing
void CProjectDlg::OnOK()
{
    m_ProjectEdit.GetWindowText(m_strProjectName);
    CDialog::OnOK();
}

// Initialization -- handles the WM_INITDIALOG message
BOOL CProjectDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Subclass the project name so we can validate it
    m_ProjectEdit.SubclassDlgItem(IDC_PROJECT_NAME, this);

    // Set the title of the dialog -- this allows us to use the same dialog
    // to rename a project as well.
    if(!m_strTitle.IsEmpty())
        SetWindowText(m_strTitle);

    return TRUE; // return TRUE unless we set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

// CProjectEdit -- Subclass for the project name field

// Constructor
CProjectEdit::CProjectEdit()
{
}

// Destructor
CProjectEdit::~CProjectEdit()
{
}

BEGIN_MESSAGE_MAP(CProjectEdit, CEdit)
   //{{AFX_MSG_MAP(CProjectEdit)
    ON_WM_CHAR()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
////////////////////////////////////

// CProjectEdit message handlers

// Filter out all of the characters but alpha-numeric, space and backspace
void CProjectEdit::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if(isalnum(nChar) || isspace(nChar) || nChar == '\b')

```

```
        else ::MessageBeep(0);
    }

    // Call online help
void CProjectDlg::OnHelpbtn()
{
    if(m_strTitle.IsEmpty())
        AfxGetApp()->WinHelp(1001,HELP_CONTEXT);
    else AfxGetApp()->WinHelp(1005,HELP_CONTEXT);
}
```