A WEB SITE ANALYZER USING JAVA APPLICATION TECHNOLOGY

LAN JIN

A MAJOR REPORT

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

AUGUST 2003

Your file Votre référence
ISBN: 0-612-83906-0
Our file Notre référence
ISBN: 0-612-83906-0

Acknowledgements

I would like to thank all those who made the final realization of this dissertation possible. It is not possible to mention all their names, however I would like to express my special gratitude to the following contributors.

I am greatly indebted to Professor Peter Grogono, who would like to be my supervisor and encouraged my interest in the project of Web Site Analyzer, for his technical advice, generous attention, constant help and critical remarks throughout my Major Report. I am pleased to express my gratitude to Prof. W.M. Jaworski, an examiner for my work, for his valuable comments and evaluations on this report. My thanks are extended to Halina Monkiewicz, the Graduate Program Secretary, for her support and collaboration.

My sincere thanks are due to the support given by my company, Motorola Canada Software Center, Montreal, Quebec, Canada. My special thanks to Lisa Li, Operation Manager, and Josée Despatie, Human Resources, for their support and encouragement in many ways during my master program study.

Finally, I would like to express my deep gratitude to my husband, Litian Zhou, for his all kind of assistants and supports.

ABSTRACT

A WEB SITE ANALYZER USING JAVA APPLICATION TECHNOLOGY

Lan Jin

Over 5% of the links of a typical web site are broken. Orphaned files are another problem: they form a kind of garbage in the server file system of a web site. To maintain a web site — especially a large web site — is difficult without an assistant tool. In this report, I describe the development of a web site analyzer tool intended to help webmasters to maintain their web sites efficiently. The tool can perform link checking in both remote and local web sites and can search the orphaned files in the local server file system. The project is designed with object-oriented design methodology and implemented using Java application technology.

Table of Contents

List of Figures

1 Introduction

With the rapid growth of the Internet since the mid 1990s, maintaining web sites — especially large web sites — becomes ever more important and difficult. Over 5% of links on a typical web site are broken because of bad tag usage, HTML code syntax errors, and spelling errors. The errors make it hard for browsers to process the HTML and difficult for visitors to read the web site [1].

Every link and connection within a web site must work together flawlessly to ensure effective visitor experiences. Since the mid 1990s, several tools have been created to act as "Link Checkers": they check broken (often called "bad") links and orphaned files (in other words, files that cannot be reached via one or more internal links from the web site's Home Page). The goals of tools of this kind include improving web site efficiency, protecting the web site owner's integrity, and increasing web team productivity for webmasters who wish to check internal and external links within their Internet site [1] [2] [3] [4].

Since the purpose of this kind of tool is to check the validity of each link on any Web Page on the web site, it should be an accurate, fast, scalable product that is easily available and highly customizable. It should be also have a simple and friendly interface: it should be necessary only to enter a URL into the tool and let the tool check everything, reporting all good and dead links on the site. As well, the report of the result should be comprehensive and detailed [2] [3] [4].

The Web Site Analyzer described in this report was developed using JAVA application technology to implement link checking and web site management for both Unix and Microsoft Windows operating systems. It can support both remote and local link checking. It will check URL links, image links, frame links, ftp links, mailto links and so on. It will also perform duplicate checking, dead loop checking, and orphaned file checking. After finish the link checking, a tree structure is displayed in the user interface to show the user relationships between the links in a clear manner. The report is generated in HTML format and so may be easily viewed in all industry standard browsers.

The rest of the report is organized as follows. Chapter 2 provides the background of why and how this kind of tool needs to be developed. Chapter 3 outlines the functional requirement specifications. Chapter 4 is the software design document including high level design and detailed design description. Chapter 5 shows the testing plan and result. Chapter 6 gives the conclusion and the future work. There is a list of the references used in this project. Finally, the appendix provides sample implementation codes and illustrates the format of the web checking report.

2 Background

2.1Common Problems with Web Sites

2.1.1 Broken Links

A *broken link* (also called “bad link”) is any link to HTML or text file that the Web Site Analyzer (WSA) was unable to follow. It is a common occurrence for users to experience broken links when they browse web sites. Links may be broken for a variety of reasons. These include [5]:

1. the destination document has been moved or deleted;
2. the source document has been moved (in the case of relative links);
3. the URL is poorly formed (for example, it may have un-escaped special characters such as “<” and “>”);
4. the HTML is poorly coded (for example, there may be incorrect syntax);
5. the server configuration has changed;
6. the indexer has become confused.

Maintaining a web site — especially a large web site — is difficult without an assistant tool of some kind. Webmasters must check all the links in the web site regularly to keep the site correct and efficient. It is almost impossible to do the link checking manually for a large web site. With the wide use of web sites, WSA is the ideal tool to perform this task for the web site maintenance.

2.1.2 Orphaned Files

Orphaned files are files that were found on the server file system but had no hyperlinks pointing at them. Orphaned files are the main other problem (apart from missing links) that occur during web sites development and maintenance. They are a form of "garbage" in the server file system. They may be caused by:

1. a hyperlink has been changed or deleted in the web pages;
2. the source documents lost connection with the hyperlinks since the migration of a source file;
3. hyperlink editing errors;
4. incorrect HTML syntax;
5. the server configuration has changed.

Finding orphaned files in the server file system and cleaning them up is the other principal task in web site maintenance. Most link checking tools do not have a feature to find orphaned files. But if the orphaned files become more and more frequent in the server file system, they will occupy more and more space, thereby impacting the efficiency of the web site performance.

Using the orphaned file report can identify the files that are no longer used in the web site. In some cases the webmaster will want to archive or delete the orphaned files, and in others he/she may need to restore some navigational links so that the orphaned files may be found by users [4]. The orphaned file report is very useful to repair the web site and clean up the disk space that it occupies.

2.2 Related Algorithms

There are different algorithms that could be used to implement the link checking tool. But to design a good algorithm is worthwhile because it proves a more efficient implementation of the tool. The following pseudo code indicates one way in which the Web Site Analyzer might work.

```
1. Create an empty “master file list” and an empty “master edge list”.
2. Create an empty queue, $Q$.
3. Insert the names of all of the files in the given directory into $Q$.
4. While $Q$ is not empty:
    (a) Remove the first file, $F$, from $Q$.
    (b) Insert $F$ into the master file list.
    (c) If $F$ is a local HTML file, process it as follows.
    (d) For each tag in $F$ of the form <a href=>. . . :
        i. Let the name of the linked file be $G$.
        ii. Insert $G$ into the master file list.
        iii. Add the edge $F \rightarrow G$ to the master edge list.
5. Generate a report showing the information in the master lists.
```

3 Functional Requirement Specification

3.1 Purpose

Maintaining a large web site is a significant task. There are tools available to help but they are slow, expensive, or cumbersome to use. The objective of this project is to design and implement a Web Site Analyzer that is simple to use and provides useful results quickly for both remote and local web sites.

3.2 Product Perspective

The Web Site Analyzer (WSA) will be easily installed in to a personal computer with operating system Windows or Unix/Linux. When running WSA, a web browser, such as Internet Explorer 5.0, or Netscape 4.7, will be needed for view the analyzed report. If analyzing a remote web site, an Internet connection through a local Internet Service Provider is also needed.

3.3 Functional Requirements

There are two main functionalities in WSA: Link Checker and Analyzer Report. The user can start to run the Web Site Analyzer by giving it the address of a web site or the name of a directory on the local disk. Then the tool will perform link checking. The user also can ask to view the Analysis Report after link checking has finished.

3.3.1 Two Domain Applications

The web site analyzer will allow the users to perform the link checking of both remote and local web sites. The user can import a web site home address to perform a remote web site check through the Internet, or import a local file path for local link checking.

3.3.2 Link Checker

3.3.2.1 Page Parser

The Link Checker will find all linked files starting from the home address or local directory. The Page Parser will scan all the linked files in the site and show them to the user in the user interface.

3.3.2.2 Tree Structure

After scanning the web site or local directory, the tool will build a Tree Structure and display it in the user interface. The nodes of the tree are files. The edges of the tree are links between files. The Tree Structure will make it easy to reach all the files in the site from those files by traversing links.

3.3.2.3 Internal Link List

The Link Checker will build an Internal Link List to list all the links which contain some child links. The home address is listed in the Internal Link List. All the Internal Linked files will be parsed by the Page Parser.

3.3.2.4 External Link List

The Link Checker will build an External Link List to list all the links which connect to outside of the site. The Link Checker will stop checking at External Linked files. (Otherwise the Web Site Analyzer will return the entire World Wide Web as its result.)

The External Links are treated as Leaf Links. The External Linked files are also called "foreign" files.

3.3.2.5 Leaf Link List

The Link Checker will build a Leaf Link List to list all the links which do not contain any outgoing links. Such a file might be an image, an executable, or a simple text file.

3.3.2.6 Bad Link List

The Link Checker will build a Bad Link List to list all the links which can not link to the target files. The Link Checker will connect to all the links to test whether the target files can be opened.

3.3.2.7 Permission Denied Link List

The Link Checker will build a Permission Denied Link List to list all the links which are forbidden to access the target pages. The Permission Denied Links are treated as the Leaf Links.

3.3.2.8 Orphaned File List

The Link Checker will build an Orphaned file List to list all the files which cannot be reached via one or more internal links from the Home Page. This kind of search will only be done when analyzing a local web site.

3.3.2.9 Reference Info List

The Link Checker will build a Reference Info List for each checked link. The Reference Info List will list all the files which called the current link. That is used to trace back all the parent files of this link.

3.3.3 Analyzer Report

The user can ask to generate the Analyzer Report from the user interface after finishing the Link Checking. The Report will be generated in an HTML file and opened in a web browser (Internet Explorer is as the default web browser).

The Report will include:

- The home address which the user imported
- A table of contents
- The Internal Link List with the children links for each internal link
- The Reference Info List with the parent links for each link
- The External Link List
- The Leaf Link List
- The Bad Link List
- The Permission Denied Link List
- The Orphaned File List for local web site
- The Summary of Web Analysis with the number for each checked list

3.4 User Interface Requirements

A user friendly GUI (Graphic User Interface) is required for users to implement the tool. The user interface will be simple, clear and easy to run. No special skills are required to use the system. The main requirements for the user interface are list as following:

- A text field is required for the user to import the web site URL or local file path.
- A menu bar is required for the user to select the different functionality such as "Link Checker" or "Generate Report".

- A button "Enter" is required for the user to start running the program.
- An output window is required to allow the user to monitor the processing of checking in the run time and to display the Tree Structure after finish the checking.
- A status line is required to show some messages to indicate the user.
- A button "Quit" is required for the user to exit the system at any time.

3.5 System Requirements

3.5.1 Platform

Windows 95/98/2000/NT or Unix/Linux will be the platform for this project. The minimum requirements for PC are: CPU 486 or higher, 32 MB memory or higher, and 5 GB hard disk or higher.

3.5.2 Java Environment

JBuilder 7 or above is for the developers writing, deploying, and running applications in the Java programming language.

3.5.3 Internet Connection

A reliable Internet connection is required for running this web site analyzer. DSL high-speed connection or cable connection is recommended, because the system running time depends on the Internet connection speed for a same web site checking. Though dial-up connection with at least 56k bps modem can also be considered.

4 Design

4.1 Design Rationale

Due to the nature of the software development life cycle, constant change is encountered in both functional requirements and non-functional requirements. The key principle of a good software design is the concept of “design for change”. The software development process should have the potential to adapt as many changes as possible during the course of development and even after the system is already in use. So the software design goals should include high cohesion of functions within a module, loose coupling of functions between modules, and, as well, as design should be carefully abstracted and easy to understand.

According to the software requirement specification, this project is an Internet application tool. The system was implemented using JAVA application technology. All users can run the tool by installing it into their personal computer with JAVA environment, a web browser and an Internet connection.

The design uses object-oriented design methodology and meets the concept of separation of concerns by using the Model/Controller/View structure. In the architecture design, the system capabilities will be associated with the system components that implement them. All components will be usually modularized and the architecture also describes the interconnections among them. The goal of the design is to achieve a quality of product with ease of understanding, ease of implementation, ease of testing, ease of modification and correct translation from the requirement specification.

4.2 Technology and Development Tool Selection

Java is a powerful language for developing Internet application. Java and J2EE technology provide the enterprise solution for building a modular system. The Internet application requires to be built with standardized, modular components, with high reliability, scalability, flexibility, and security. Java is platform independent. It is scalable and portable. J2EE can be built with many existing infrastructures. Java is pure object-oriented language. J2EE uses a highly modular component approach, for high cohesion of functions within a module, and the loose coupling of function between modules. The cost of building a J2EE system is fairly low. Most of the software can be download without charge.

4.3 High Level Design

4.3.1 System Architecture Design

The figure 4.1 shows the architecture of the Website Analyzer. It includes three layers: User Interface, Event Handler and Web Site or Local folder layers. The User Interface provides two options for the inputs: Link Check request and Report request. The Event Handler contains two modules: Link Checker and Report.

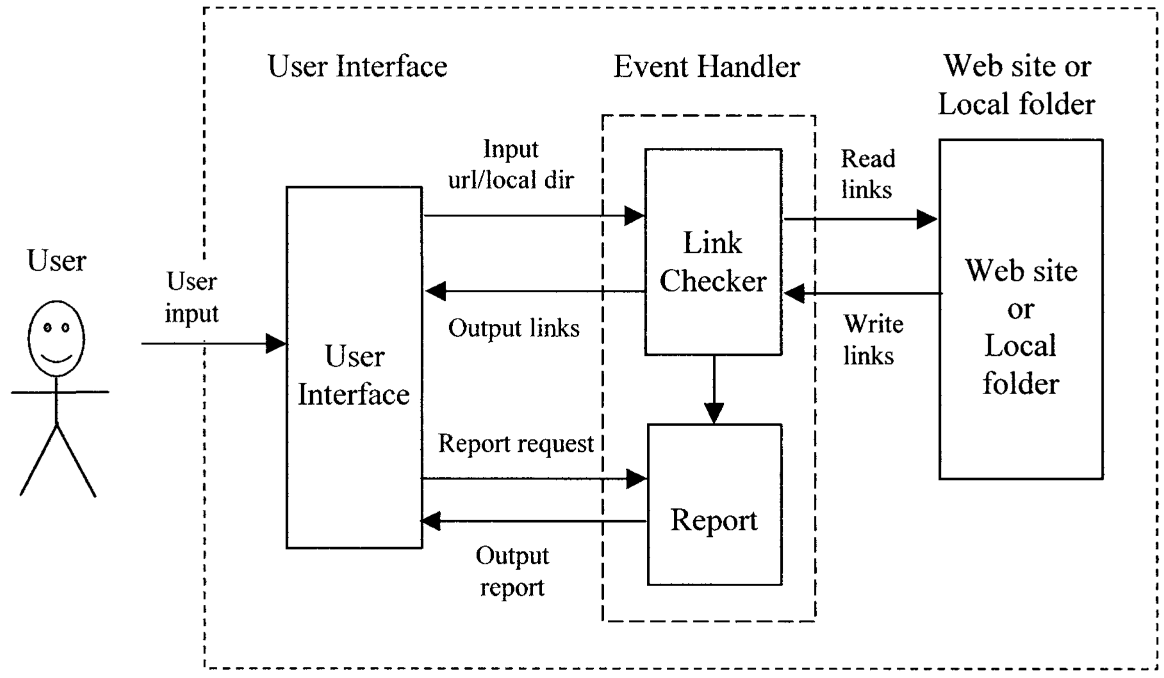


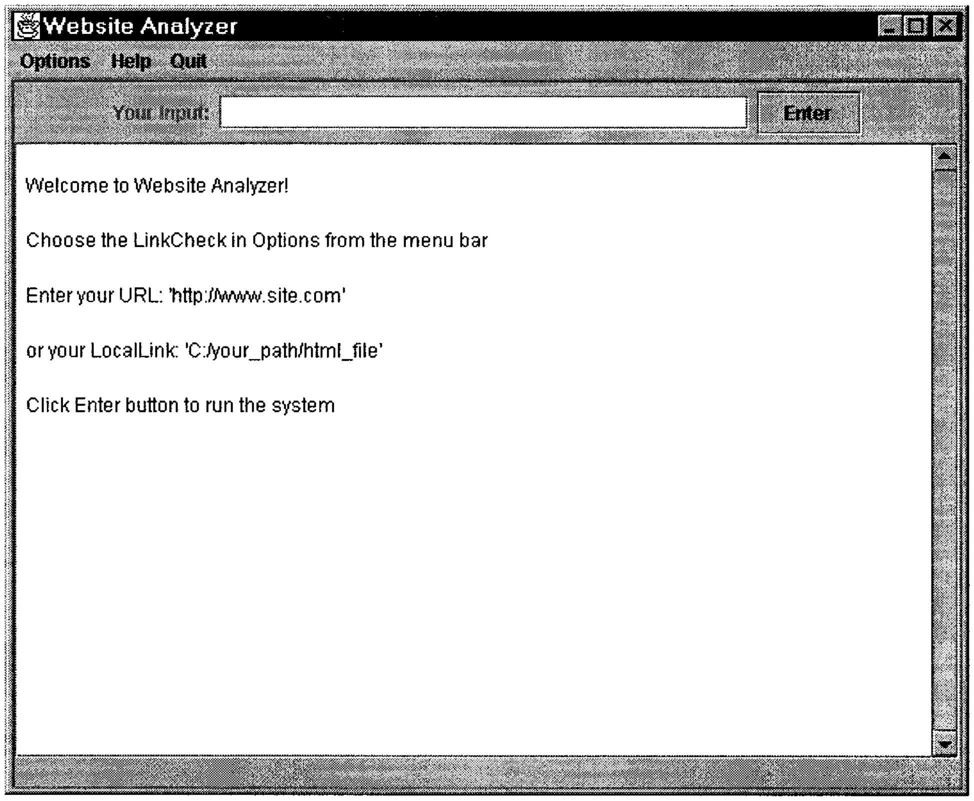
Figure 4-1 The Architecture Design diagram

The user inputs the website address or local directory from the User Interface. The input request will send to the Link Checker for processing. The Link Checker will access the home page of the input address and parse all the links inside of that site. The checked links then will be returned back to the User Interface and displayed in the output window one by one. After the link check has finished, the Link Checker will send the checked results to the Report module.

When the user requires the Report for the checked web site or local folder, the Report module will generate the report and display it in the User Interface with an HTML file.

4.3.2 User interface Design (Module Description)

4.3.2.1 Main Page Overview



Welcome to Website Analyzer!

Choose the LinkCheck in Options from the menu bar

Enter your URL: 'http://www.site.com'

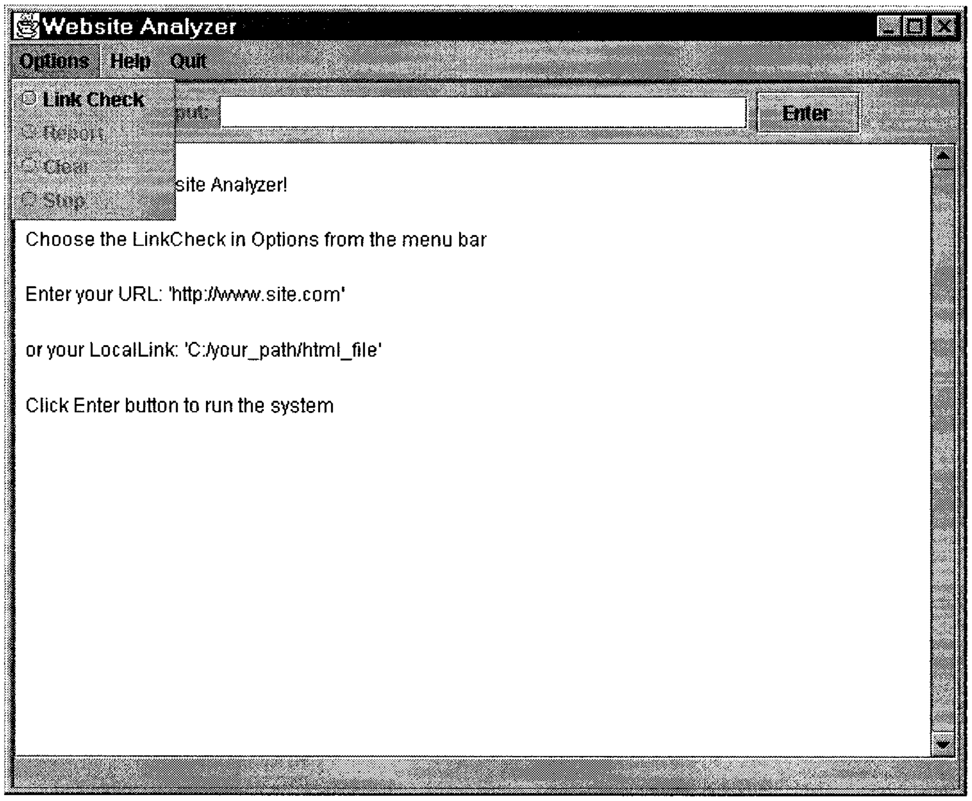
or your LocalLink: 'C:/your_path/html_file'

Click Enter button to run the system

Figure 4-2 Welcome Page

Figure 1 shows the welcome page of the Website Analyzer. It contains a menu bar, an input text box, an “Enter” button and an output window with a brief introduction explaining how to run the Website Analyzer.

4.3.2.2 Menu Bar



site Analyzer!

Choose the LinkCheck in Options from the menu bar

Enter your URL: 'http://www.site.com'

or your LocalLink: 'C:/your_path/html_file'

Click Enter button to run the system

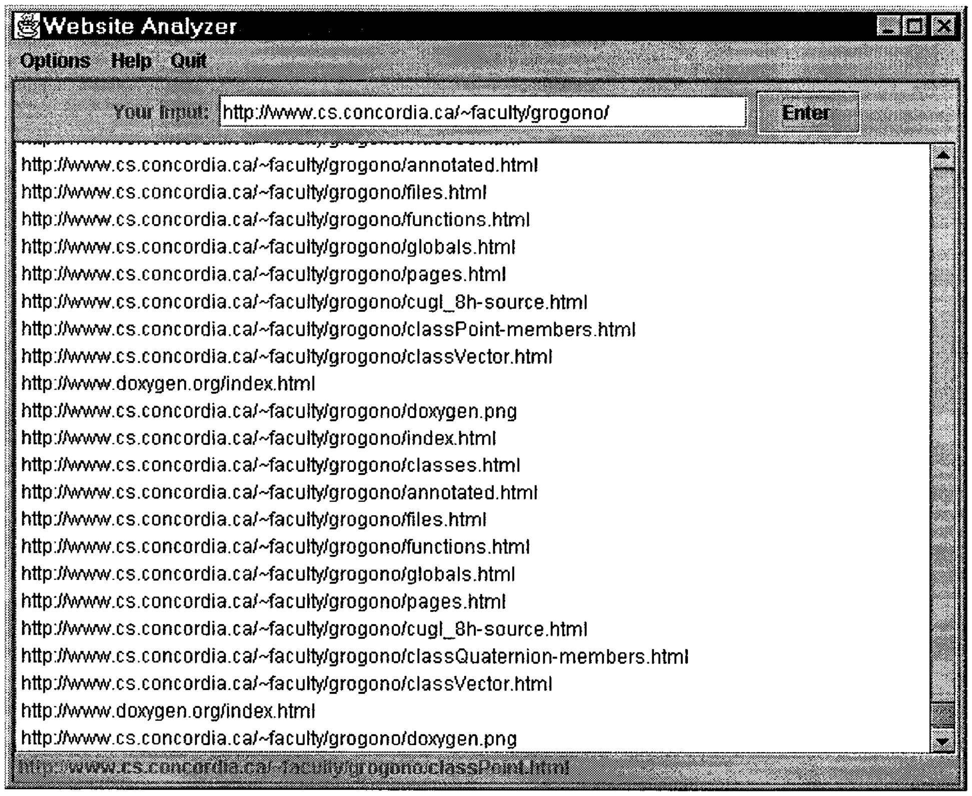
Figure 4-3 The Menu Bar in the Welcome Page

There are four choices in Options of the menu bar: Link Check, Report, Clear and Stop. When the tool is run for the first time, only the Link Check option is enabled and the others are disabled. The user needs to choose the radio button of Link Check in the menu bar and input the website link or local folder path in to the input text box before pressing the "Enter" button to run the application.

After running the Link Check, the other three options become enabled. The user can choose the "Stop" option at any time during the run to stop the current website link or to perform local folder checking and do an another link checking. The "Clear" option is to clear the Output Window, the Input Text Box, and the Status Line so as to be ready for more link checking. The "Report" option is to generate the link check report after finishing the current link check.

There are the other two items in the menu bar: Help with an About button to provide development information about the tool, and Quit with an Exit button to allow the user exit the system at any time during the run.

4.3.2.3 Output Window



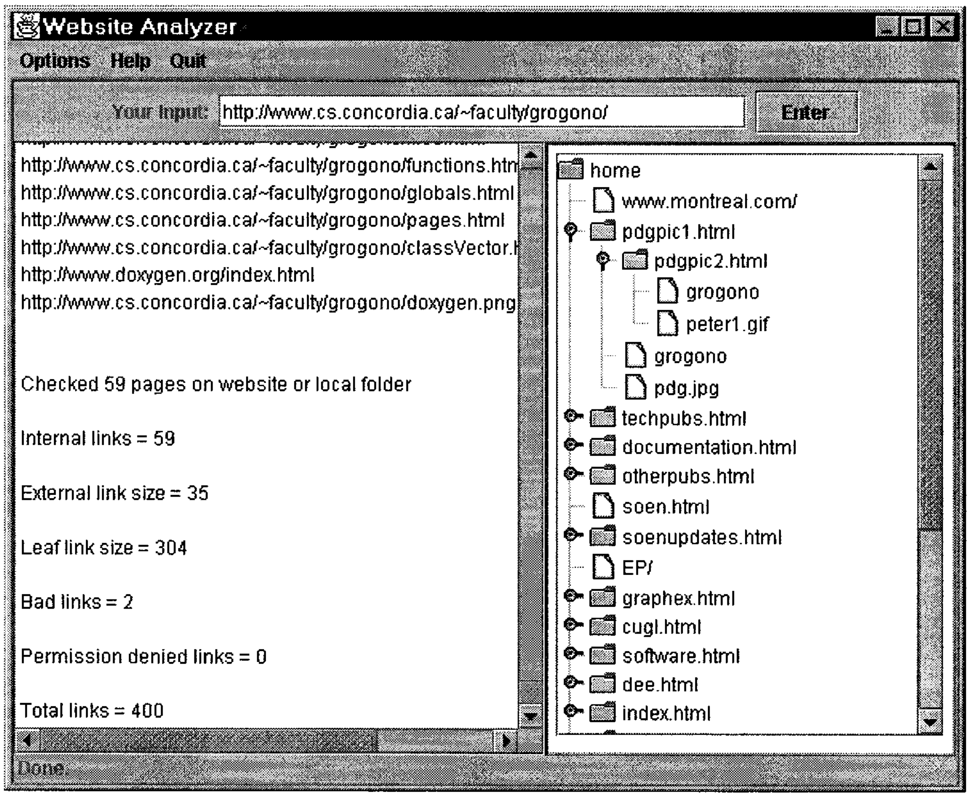
Your Input: http://www.cs.concordia.ca/~faculty/grogono/

```
http://www.cs.concordia.ca/~faculty/grogono/annotated.html
http://www.cs.concordia.ca/~faculty/grogono/files.html
http://www.cs.concordia.ca/~faculty/grogono/functions.html
http://www.cs.concordia.ca/~faculty/grogono/globals.html
http://www.cs.concordia.ca/~faculty/grogono/pages.html
http://www.cs.concordia.ca/~faculty/grogono/cugl_8h-source.html
http://www.cs.concordia.ca/~faculty/grogono/classPoint-members.html
http://www.cs.concordia.ca/~faculty/grogono/classVector.html
http://www.doxygen.org/index.html
http://www.cs.concordia.ca/~faculty/grogono/doxygen.png
http://www.cs.concordia.ca/~faculty/grogono/index.html
http://www.cs.concordia.ca/~faculty/grogono/classes.html
http://www.cs.concordia.ca/~faculty/grogono/annotated.html
http://www.cs.concordia.ca/~faculty/grogono/files.html
http://www.cs.concordia.ca/~faculty/grogono/functions.html
http://www.cs.concordia.ca/~faculty/grogono/globals.html
http://www.cs.concordia.ca/~faculty/grogono/pages.html
http://www.cs.concordia.ca/~faculty/grogono/cugl_8h-source.html
http://www.cs.concordia.ca/~faculty/grogono/classQuaternion-members.html
http://www.cs.concordia.ca/~faculty/grogono/classVector.html
http://www.doxygen.org/index.html
http://www.cs.concordia.ca/~faculty/grogono/doxygen.png
```

```
http://www.cs.concordia.ca/~faculty/grogono/classPoint.html
```

Figure 4-4 The Output Window in run time

During execution of the Website Analyzer tool, all the checked links are displayed one by one in the output window. The output links are automatically scrolled down so that the user can monitor the latest group of checked links. The movement of the links in the output window also indicates to the user that the tool is still running. The last line in the frame of this figure is the Status Line. It shows the current checked link and some indicative messages, such as "Done" after the current link check has finished, "Generate report ..." when the "Report" option is chosen, and "Stopping the current link checking!" after the user presses the "Stop" option.

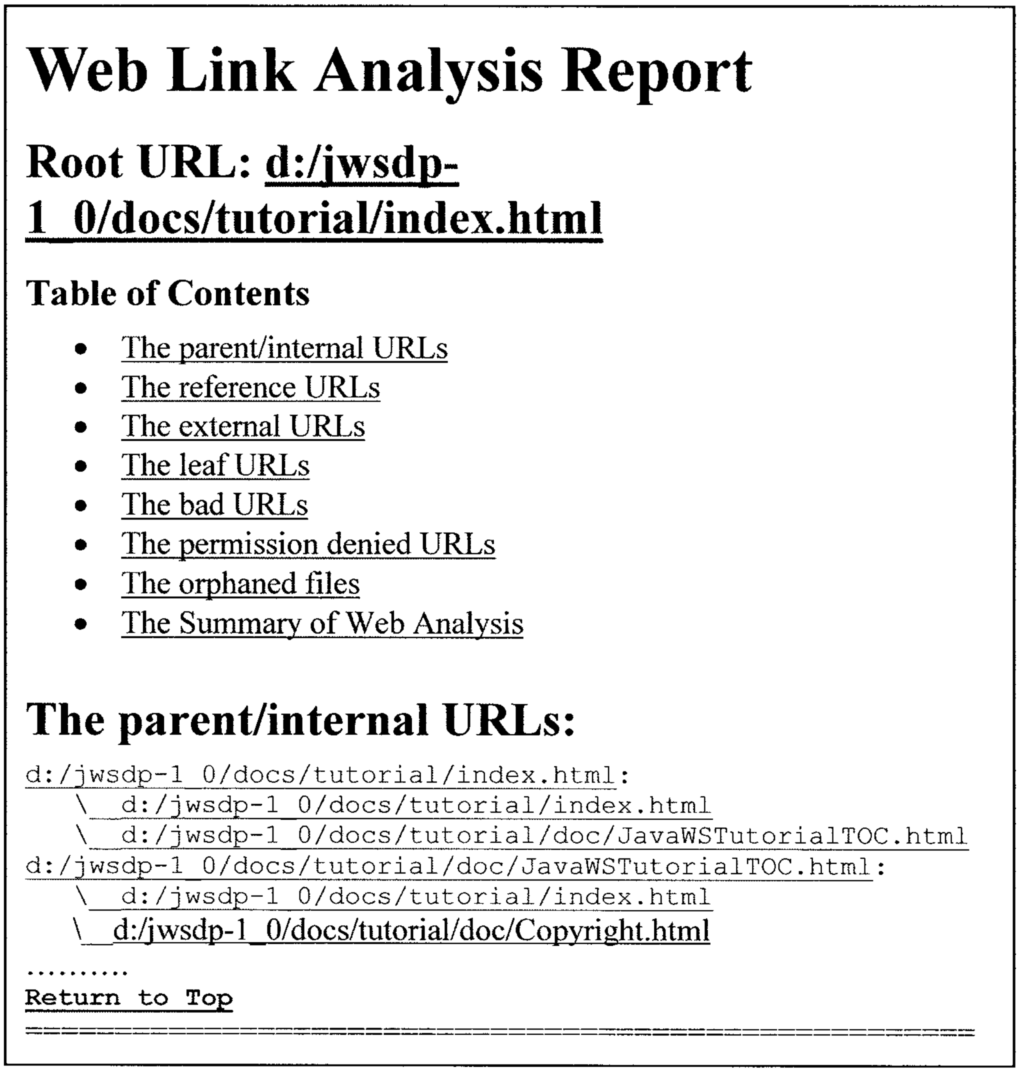


```
http://www.cs.concordia.ca/~faculty/grogono/functions.htm
http://www.cs.concordia.ca/~faculty/grogono/globals.html
http://www.cs.concordia.ca/~faculty/grogono/pages.html
http://www.cs.concordia.ca/~faculty/grogono/classVector.
http://www.doxygen.org/index.html
http://www.cs.concordia.ca/~faculty/grogono/doxygen.png


Checked 59 pages on website or local folder

Internal links = 59

External link size = 35

Leaf link size = 304

Bad links = 2

Permission denied links = 0

Total links = 400
```

Figure 4-5 The Output Window after finished the link check

When the Website Analyzer tool finished the link check, the status line shows "Done". A summary of the checked links is shown in the bottom of the Output Window. A tree structured according to the level of the link in the website or local folder will be generated and displayed in the right side of the Output Window. The user can extend the tree until the leaf page. The Figure 4.5 is an example of the Output Window after finished the link check.

4.3.2.4 Format of Web Analysis Report



Web Link Analysis Report

Root URL: d:/jwsdp-1_0/docs/tutorial/index.html

Table of Contents

- The parent/internal URLs
- The reference URLs
- The external URLs
- The leaf URLs
- The bad URLs
- The permission denied URLs
- The orphaned files
- The Summary of Web Analysis

The parent/internal URLs:

```
d:/jwsdp-1_0/docs/tutorial/index.html:
    \__d:/jwsdp-1_0/docs/tutorial/index.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorialTOC.html
d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorialTOC.html:
    \__d:/jwsdp-1_0/docs/tutorial/index.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/Copyright.html
..........
Return to Top
=============================================
```

Figure 4-6 Format of Web Analysis Report

The Figure 4.6 is the format of Web Analysis Report. There is also an example for the detail of it in the Appendix B.

4.4 Detailed Design Description

4.4.1 Class Diagram for Website Analyzer

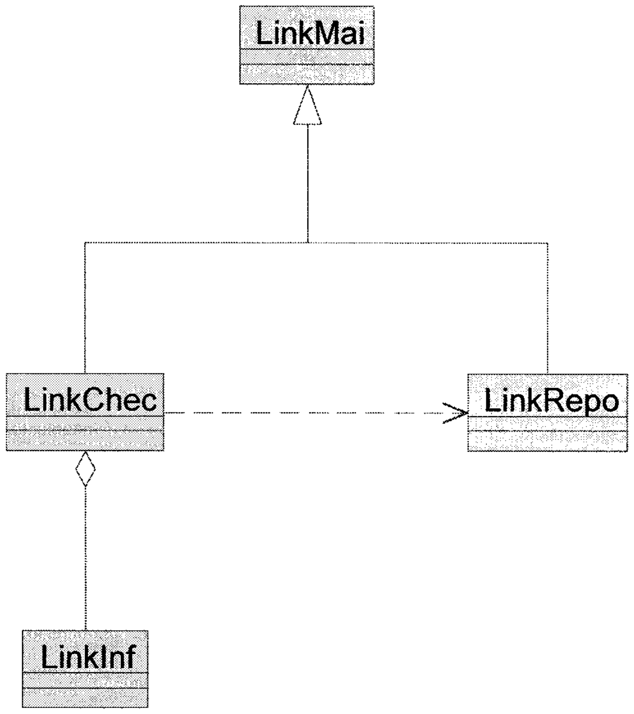


Figure 4-7 Class Diagram for the Website Analyzer

The class diagram of this system includes four classes: LinkMain, LinkCheck, linkInfo and LingReport. The LinkMain class is the main class of the system. It inherits two classes: LinkCheck and LinkReport. There is a relationship between LinkCheck and

LinkReport: the LinkReport class will use some data from the LinkCheck class, and the LinkInfo class is aggregated with the LinkCheck class.

The detailed list of the attributes and methods for each class is given in the following subsections.

4.4.1.1 LinkMain Class

This class is the main class for the system. It will create the user interface and the major executive functions which connect the interface with the event handler. Figure 4.7 is the UML presentation of the class.

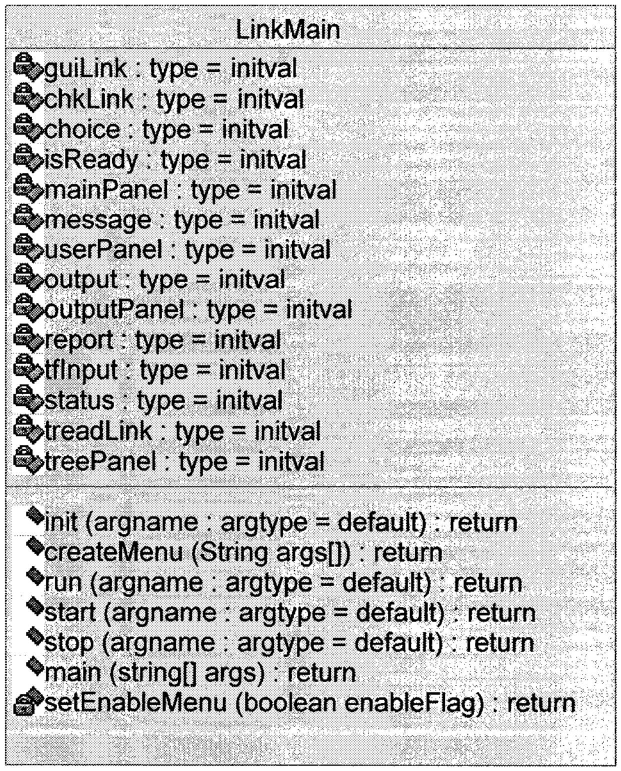
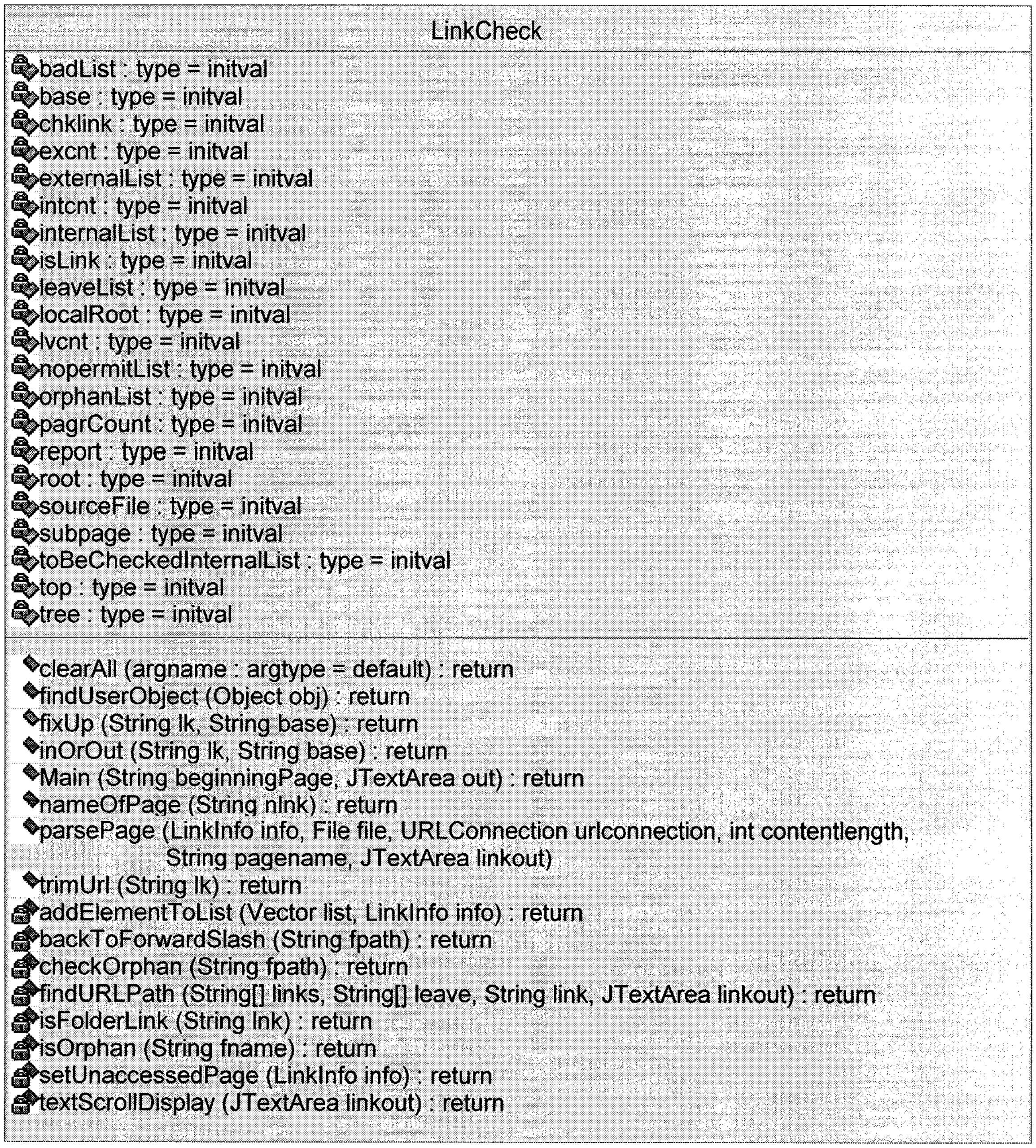

LinkMain
guiLink : type = initval
chkLink : type = initval
choice : type = initval
isReady : type = initval
mainPanel : type = initval
message : type = initval
userPanel : type = initval
output : type = initval
outputPanel : type = initval
report : type = initval
tfInput : type = initval
status : type = initval
treadLink : type = initval
treePanel : type = initval
init (argname : argtype = default) : return
createMenu (String args[]) : return
run (argname : argtype = default) : return
start (argname : argtype = default) : return
stop (argname : argtype = default) : return
main (string[] args) : return
setEnableMenu (boolean enableFlag) : return

Figure 4-8 LinkMain Class

4.4.1.2 LinkCheck Class

The LinkCheck class is called by the LinkMain class when user executes the program. It performs the link checking for all the links in the website or local folder which the user has provided from the user interface. Figure 4.8 is the UML presentation of the class with a listing of the attributes and methods with listing the attributes and methods.



```
LinkCheck
badList : type = initval
base : type = initval
chklink : type = initval
excnt : type = initval
externalList : type = initval
intcnt : type = initval
internalList : type = initval
isLink : type = initval
leaveList : type = initval
localRoot : type = initval
lvcnt : type = initval
nopermitList : type = initval
orphanList : type = initval
pagrCount : type = initval
report : type = initval
root : type = initval
sourceFile : type = initval
subpage : type = initval
toBeCheckedInternalList : type = initval
top : type = initval
tree : type = initval

clearAll (argname : argtype = default) : return
findUserObject (Object obj) : return
fixUp (String lk, String base) : return
inOrOut (String lk, String base) : return
Main (String beginningPage, JTextArea out) : return
nameOfPage (String nlnk) : return
parsePage (LinkInfo info, File file, URLConnection urlconnection, int contentlength,
           String pagename, JTextArea linkout)
trimUrl (String lk) : return
addElementToList (Vector list, LinkInfo info) : return
backToForwardSlash (String fpath) : return
checkOrphan (String fpath) : return
findURLPath (String[] links, String[] leave, String link, JTextArea linkout) : return
isFolderLink (String lnk) : return
isOrphan (String fname) : return
setUnaccessedPage (LinkInfo info) : return
textScrollDisplay (JTextArea linkout) : return
```

Figure 4-9 LinkCheck Class

4.4.1.3 LinkInfo Class

The LinkInfo Class is the part of the LinkCheck class. It stores the information of the ckecked links, such as the link strings, the source files, and the status of the links. Figure 4.9 is the UML presentation of the class with listing the attributes and methods.

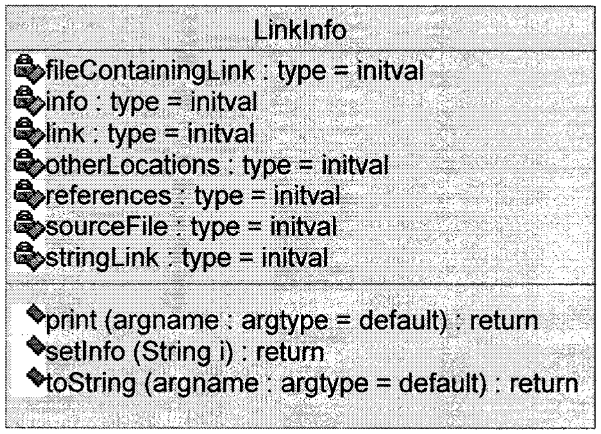
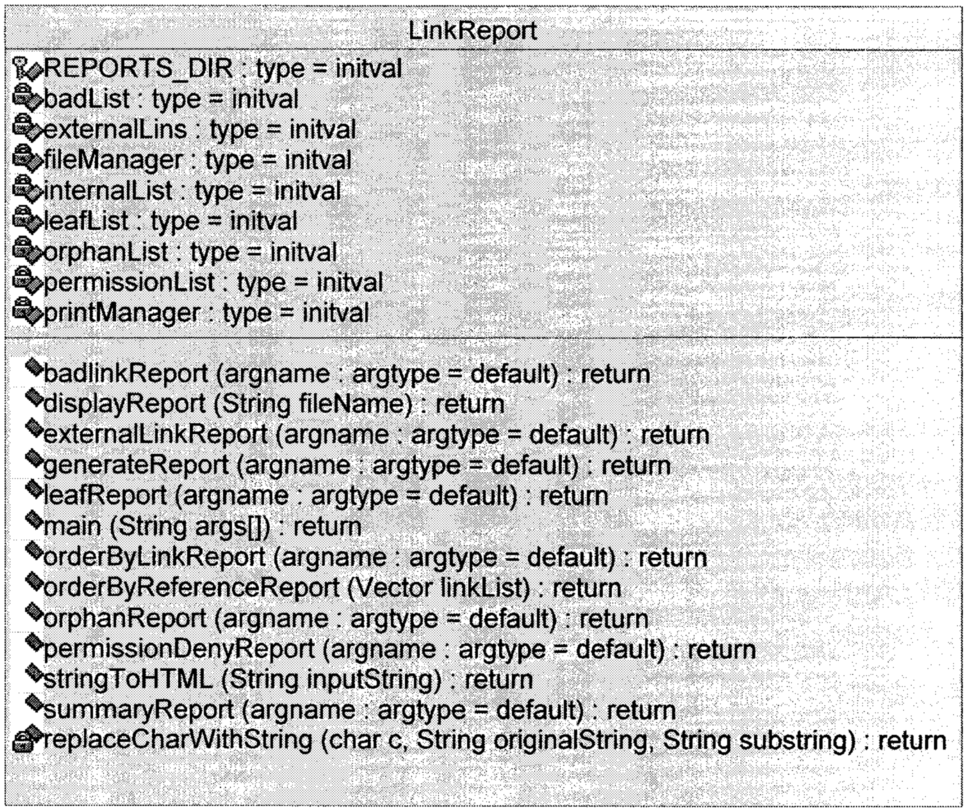

LinkInfo
fileContainingLink : type = initval
info : type = initval
link : type = initval
otherLocations : type = initval
references : type = initval
sourceFile : type = initval
stringLink : type = initval
print (argname : argtype = default) : return
setInfo (String i) : return
toString (argname : argtype = default) : return

Figure 4-10 LinkInfo Class

4.4.1.4 LinkReport Class

The main functional of the LinkReport class is to generate the report when the user requests it. It will get all the information of the checked link from the LinkCheck class and create an HTML file which will list the internal links, reference links, external links, leaf links, bad links, permission denied links, orphaned files (only for the local folder check). Figure 4.10 is the UML presentation of the class with listing the attributes and methods.



```
LinkReport
REPORTS_DIR : type = initval
badList : type = initval
externalLins : type = initval
fileManager : type = initval
internalList : type = initval
leafList : type = initval
orphanList : type = initval
permissionList : type = initval
printManager : type = initval

badlinkReport (argname : argtype = default) : return
displayReport (String fileName) : return
externalLinkReport (argname : argtype = default) : return
generateReport (argname : argtype = default) : return
leafReport (argname : argtype = default) : return
main (String args[]) : return
orderByLinkReport (argname : argtype = default) : return
orderByReferenceReport (Vector linkList) : return
orphanReport (argname : argtype = default) : return
permissionDenyReport (argname : argtype = default) : return
stringToHTML (String inputString) : return
summaryReport (argname : argtype = default) : return
replaceCharWithString (char c, String originalString, String substring) : return
```

Figure 4-11 LinkReport Class

4.4.2 Flow Chart for system processing

The Flow Chart for link check processing, shown in Figure 4.11, describes the main algorithm for checking the links.

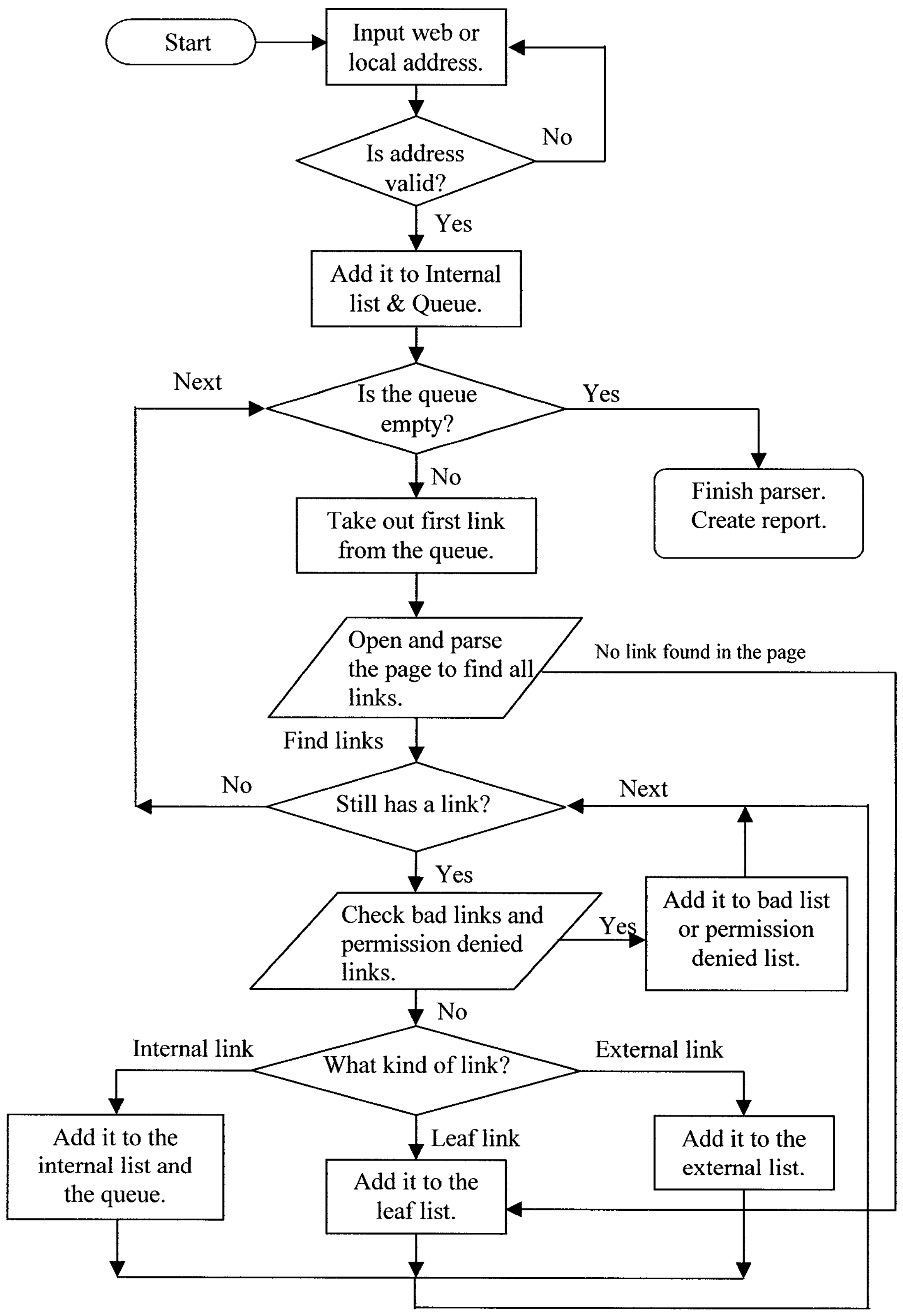

Start
Input web or local address.
Is address valid?
No
Yes
Add it to Internal list & Queue.
Next
Is the queue empty?
Yes
No
Finish parser. Create report.
Take out first link from the queue.
Open and parse the page to find all links.
No link found in the page
Find links
No
Still has a link?
Next
Yes
Check bad links and permission denied links.
Yes
Add it to bad list or permission denied list.
No
Internal link
What kind of link?
External link
Add it to the internal list and the queue.
Leaf link
Add it to the leaf list.
Add it to the external list.

Figure 4-12 Flow Chart of Web Analyzer Processing

Once the user has provided an address from the user interface and started the application, the system will first to check to see if the address is valid or not. If not, an error message will be displayed. If it is a valid address, the event handler will add it into the internal list and a storage queue. It will then take the first link from the storage queue, open and parse this page to find out all the links in it. At the same time it will check each link to determine what kind of link it is. If it is the external or leaf link, it is put into the external or leaf list accordingly. Otherwise, it should be an internal link, in which case it is put into the internal list as well as the storage queue. The system will do the loop checking until it finishes all the link checks.

4.4.3 Link Checker Module

The Link Checker Module is the core part of the event handler. It performs the major functionality of the tool to check all the links in a web site or a local folder. It analyzes those links and separates them into several categories: external links, internal links, leaf links, bad links, and permission denied links.

This Module also performs some specific important checking such as duplicate checking, dead loop checking, and orphaned file checking (for local folder).

There is some other functionality in this Module. It will build a tree structure according to the page level in the website or local folder after finishing the link search. It includes a counter as well to count the numbers of the different kind of links and give out the results in the user interface. It also creates a reference info list for each checked link with all the pages which called this link.

4.4.3.1 Page Parser

The page parser is a function to parse an opened link file in order to find all the links in the file. Since the link has a specific format, when the parser finds something that begins with "<a" or "<A", it assumes that there is a link. If it finds something beginning with "<f' or "<F", it assumes that there is a frame. If it finds something beginning with "<I" or "<I", it assumes that there is an image. All these are considered to be links and will lead to further link checking recursively.

4.4.3.2 External Link Checking

An *external link* is defined as a link specified using an Absolute URL to any project other than the current project [4]. It will access the page outside of the checked web site or local folder. If the link is not the home address or the root directory and begins with "http:", "mailto:", or "ftp:", it must be an external link. So it is put into the external link list.

4.4.3.3 Internal Link Checking

An *internal link* is defined as a link that will access the page inside of the web site or local folder which the user entered. The home page address is an internal link and it is put into the internal link list as the first element. At the same time put it into a storage queue for later, when it will be parsed to find all the links in this page. Only internal link files will be parsed. If the link ends with ".html", ".htm", ".shtml", ".xml", ".asp", ".jsp", or ".php", but does not start with "http:", "mailto:", or "ftp:", it is assumed to be an internal link and it is put into the internal link list and the storage queue.

4.4.3.4 Leaf Link Checking

A link is a *leaf link* if the file it links to does not contain any further links. It may be a link to an image, a text file, an executable file, and so on. There are two possibilities for a link to be a leaf link. If a link belongs to neither an external link nor an internal link, it will be a leaf link and it can be put in to the leaf link list. The other possibility is that, after parsing a link page (which is in the internal link list), the counter returns zero (links), this link is assumed to be a leaf link. So it is removed from the internal link list and put it into the leaf link list.

4.4.3.5 Bad Link Checking

A *bad link* is a link to a page that cannot be found. Any kind of link may be a bad link. So the first check for every link determines whether the link can be connected. If not, it is a bad link and it is put into the bad link list.

4.4.3.6 Permission Denied Link Checking

A *permission denied link* is a link that public users cannot access. In most cases, the user must enter a user name and password. As with bad link checking, the permission denied link checking is done for all the links before sorting them. This kind of link will be put into the permission denied link list.

4.4.3.7 Duplicate Link Checking

Duplicate checking is a function which will be called whenever a link will be added into any list. The function will compare the added link name with all the link names which have already been added to the list. If the name already exists in the list, cancel the link addition, otherwise add the link into the list.

4.4.3.8 Dead Loop Link Checking

Dead loop checking is mandatory for any link checking to avoid the system running into an infinite loop. For each internal link, the event handler will check if its child link is as same as its parent link or higher level parent links until the root link. If this occurs, the child link will be destroyed and is not added to the internal link list or the to-be-checked queue.

4.4.3.9 Orphaned File Checking

Orphaned files are defined as files that are present in the Home Directory (or any subdirectory thereof) but cannot be reached via one or more internal links from the Home Page [4].

Orphaned file checking finds the files which exist but are never referenced and cannot be accessed from any page in the web site or local folder. Orphaned file checking is possible only in the local folder or if you own the web site (same as the local folder). To achieve orphaned file checking it is necessary to do an exhaustive search in the site or folder to find out all the files ending with ".html", ".htm", ".shtml", ".xml", ".asp", ".jsp", ".php". Then to check if the files already exist in the internal link list or leaf link list after finishing the link check application. If a file is not found in any of the lists, the file will be an orphaned page and add it into the orphan list.

4.4.3.10 Tree Structure

The tree structure is designed to show the user a clear relationship between the links according to the level of the links. After finishing the link checking, the tree structure will be displayed in the user interface with the home page and the second level pages of the tree. The user then can open the further level pages if the tree node is a folder. The folder signs in the tree are the internal links and the page signs are the leaf links.

4.4.3.11 Counter and Results

The counter is designed as a summary of the different kind of links. During the link search, the counter will count the links in each kind of link list whenever a link is added into a list. After finished the all link checks, the counter will give out a link check result with the check details of the numbers of each kind of link list in the user interface.

4.4.3.12 Reference Info List

The reference info list is used to trace back how many pages refer to this link. The reference info list is stored in every checked link node. It will be printed out in the generated report.

4.4.4 Report Module

The Web Link Analysis Report is the report to show the user the link check results. A user friendly format is important to ensure that users have a clear view of the results, can easily link the pages, and can conveniently jump up and down.

The functionality of the report module is to display the link checking results to the user in an HTML file. The report will include all the checking information which the user would like to know. The report is generated automatically once the user presses the "Report" option after finishing the Link Check application. The detailed design for the Report is described in the following subsections.

4.4.4.1 Format of the Report

The report is an HTML file and will be automatically generated by the Report Module. Consequently, the report layout will be coded according to the format of HTML file.

The report starts with a title and table of contents, then follows the main part of the checked results which lists all the checked links in different categories. Finally, there is a summary.

4.4.4.2 Title and Subtitles

The Report Title will be "Web Link Analysis Report" followed by the "Root URL:" plus the URL address which the user inputted from the user interface. The subtitle will be the description of each section in the report.

4.4.4.3 Table of Contents

The table of contents includes: The parent/internal URLs, The reference URLs, The external URLs, The leaf URLs, The bad URLs, The permission denied URLs, The orphaned Files, and The Summary of Web Analysis. Each item in the table of contents is a hyperlink to link to the real section part. It is easy for the user to choose which section he/she would like to check.

4.4.4.4 The parent/internal URLs

The parent link is the link which contain the child links. The parent links are the internal links including the home page link. In the section of the parent/internal URLs lists all the internal links and the child links in the page of each internal link. The internal link starts at the beginning of the line. Its child links follow under the internal link and start after some spaces followed with the sign "_".

4.4.4.5 The reference URLs

The reference links for each link are the links that refer to this link, or we could say that the reference links are the parent links of this link. In the section of the reference URLs lists all the links (include internal external and leaf links) and their parent links which called this link. The reference link is placed from the beginning of the line. Its parent links are followed under the reference link, starting after some spaces followed by the sign "_".

4.4.4.6 The external URLs

The section of the external URLs just lists the external links from the external link list.

4.4.4.7 The leaf URLs

The section of the leaf URLs simply lists the leaf links from the leaf link list.

4.4.4.8 The bad URLs

The section of the bad URLs lists the bad links with an explanation of the "badness" (such as "Error: file not found.") and the parent links which called this link. The bad link is placed at the beginning of the line. Its parent links are followed under the bad link start after some spaces followed by the sign "__ In Page:".

4.4.4.9 The permission denied URLs

The section of the permission denied URLs lists the permission denied links with the reason for denial and the parent links which called this link. The permission denied link is placed at the beginning of the line. Its parent links are followed under the permission denied link start after some spaces followed by the "__ In Page:".

4.4.4.10 The Orphaned Files

An orphaned file is a file that exists but is never referenced within the site [4]. It lost the control from the site and no one can access it from any pages of the site. The section of the orphaned files lists all the orphaned files from the orphaned file list.

4.4.4.11 The Summary of Web Analysis

In the summary section, the report gives the user a summary data of the web site analysis. There are sets of data for different kind of checked links to summarize the checked results, which include internal, external, leaf, bad, permission denied links and orphaned files.

4.4.4.12 Link to the Pages

In the report, all the links can be linked to the real pages except the bad links which will show a page containing "The page cannot be found". It is very convenient for the user who is interested to immediately look at some link pages during view the report.

4.4.4.13 Return to Top

Also for convenience, there is a hyperlink of "Return to Top" designed in each end of the section. It allow the user to go back to the beginning of the report after finishing a section search and to select other section combining to use the hyperlink of the Table of Contents. That is convenient for the user to jump up and down.

5 Testing Plan and Results

The testing plan for this project is based on the requirement specification. The purposes of testing is to validate and verify this web site analyzer tool to ensure that the system runs with reliability and efficiency, and the result is reported with accuracy and correctness.

5.1 Environmental Testing

The environment testing is used to determine the hardware and software requirements for this system. The results are listed below:

Hardware:

- CPU: Pentium II Processor.
- Memory: 128 MB.
- Hard Disk Storage: 20GB.

Software:

- Operating System: Microsoft Windows 98.
- Java Environment: Java 2 Standard Edition from Sun.
- Internet Connection: Dial-Up from Bell Sympatico.
- Browser: Internet Explorer 5.0 and Netscape 4.7.
- LinkMain.bat file for execution of the Web Site Analyzer tool.

5.2 Functional Testing

5.1.1 Enter Remote and local web sites

Enter the Home Page address of remote or local web site separately, such as http://www.site.com for the web site and "D:/your_path/html_file" for the local folder, the tool will run normally for both sites. There is a constraint for the local folder path entry. It should be as same as the above example, using slash (/) but not back slash (\). If an invalid address is entered, an error message will show up in the out put window.

5.1.2 View the output links

During execution, all checked links will display in the output window and status line. The output window will scroll down automatically.

5.1.3 Check the system run time

The tool should finish the link checking in a reasonable time depending on the size of the site to ensure there is no dead loop link checking.

5.1.4 View the tree structure

After finish the link checking, a tree structure will display in the right side of the output window. The tree can be extended from the folder icons.

5.1.5 Check the menu bar

During execution, press the "Stop", "Clear" or "Exit" button to interrupt the system running.

5.1.6 View the result report

The format and results of the report are satisfied the requirement and design.

1. The report is the format of HTML file.

2. The table of contents hyperlink to the real section part.

3. A hyperlink of "Return to Top" is in each end of the section, and it can link to the beginning of the report.

4. All the checked links in the report can be linked to the real pages.

5. The checked links include the internal links, external links, leaf links, bad links, permission denied links for both remote and local web sites.

6. The checked Internal links should include the links ended with ".html", ".htm", ".shtml", ".xml", ".asp", ".jsp", or ".php".

7. The checked External links should include the links begun with "http:", "mailto:", or "ftp:".

8. The checked Leaf links should include the image files, text files, executable files and so on.

9. When click the Bad link, the browser will show a error message " The page cannot be found".

10. When click the Permission Denied link, the browser may display a window to ask user to enter the user name and password.

11. There are no duplicate links in the report.

12. This Web Site Analyzer tool can only perform the orphaned file searching in the local folder. In the remote web site link checking report, it is always empty in the orphaned file section.

13. The summary for all kind of links or files is correct.

5.3 User Manual and Installation Testing

There are two ways to start to install and run the LinkMain software.

1. Using batch file

a) Put all the java code, project file and the batch file into a new folder, e.g. "D:\LinkMain". Be sure all java files, project file and the batch file in the same folder.

b) Modify the batch file "LinkMain.bat" if the path is not correct.

c) Double click the batch file to compile and run the program.

2. Using JBuilder

a) Same as above step a).

b) Open JBuilder4 or up and open LinkMain project file "LinkMain.jpx".

c) Be sure current displayed file is LinkMain.java.

d) Click Run on the upper menu bar.

e) Select Run "LinkMain.java" using defaults.

After run the program, the window of main page will pop up. The running procedure is as below:

a) In the opened window, select Option - Link Check from the upper menu bar.

b) Type URL in the box, e.g. "http://www.concordia.ca", then click Enter button to start checking.

c) The program now is checking links of the URL. If you want to stop it, click Option - Stop. (Note: if you stop it, you could not resume the check.)

d) Wait until link check finished, the tree of links will display in the output window.

e) Click Option - Report to display results of this link checking.

f) If you want a new check, select Option - Clear to remove all information of the last check. Repeat the steps from the beginning.

6 Conclusion and Future Work

6.1 Conclusion

This report describes the development of an efficient Web Site Analyzer tool using Java Application technology. This tool can perform the link checking on both remote and local web sites. The checked links will be sorted into different categories such as the internal links, external links, leaf links, bad links, as well as permission denied links. Searching orphaned files is available during the link checking in the local folder.

This tool provides very good assistance for webmasters to repair and maintain web sites. It is easy to use, efficient in operation, and provides an accurate report. There is also a tree structure built to help the user track the relationship between the links.

I solved some difficulties and added more features in this project:

- Support both of remote and local web site checking, but with constraint for the local folder to enter the path using slash (/) but not back slash (\).
- Build a tree structure for all the checked links to show the relationships among them.
- Specify the permission denied links from the bad links, which cannot access the pages is not because the link broken but is the security reason.
- Perform duplicate checking and dead loop checking to make the tool running more efficiently.
- Achieve the orphaned file searching in the local folder.

- Develop an efficient algorithm for searching and checking the links or files, sorting them into different lists, building the tree structure, counting the summary and generating the report.

6.2 Future work

Since the time was limited, I just developed this tool according to the project requirements. Although I have already added some extra features, there are still some potential features could be added into this tool to make it more professional and with commercial value. These are listed below.

6.2.1 Refine the user interface

- Add user guide in Help
- Add Resume button to continue running after press the Stop button
- Allow the users enter the URL by copy paste
- Change messages size and color to make it more visible

All these refinements could provide more convenience for users to use the tool.

6.2.2 Check more link information

- The date of the page last modification
- The size of the page
- Number of link in
- Number of link out
- Level of depth

The above link information is very useful for the webmaster to balance and modify the web site. Further more, if the tool could give the summary of the links which exceed a specified page weight, specified page depth, specified age, or specified size, that would be wonderful for the webmaster to easily determine which web pages need to be maintained [4].

6.2.3 Modify the format of the report

If there is more information for a link, the format of the report could be changed into the table layout. One link in one row and the columns are the list of link information. This kind of format could make the data more concentrated and easy to read, at the same time save more pages of the analyzed report [3].

6.2.4 Try remote Orphaned File checking

In this project, the Orphaned File checking is only performed in the local folder. However, it is still possible to enable Orphaned File checking for remote server. In summary, we will need to execute a small, self-contained Perl program on the remote computer. It will assemble a structure of the file system and save it as a simple ASCII file. That file may be transferred to the link checking computer using FTP (or any other more secure technique) and used to perform the orphan analysis in lieu of direct access to the remote server via the network [4].

6.3 Comparison with Web Mining jMap Technology

The jMap technology [14] is a notation and a method for representing systems architecture, structures, processes and reusable templates which was first introduced by Prof.W.M. Jaworski. By using jMap Macros, Perl and VBA utilities, Mr. Puping Peng develop an automatic tool (Web site jMap builder) to explore the given websites and harvest the information from the websites and export it to Excel spreadsheet for processing.

There are some parts representing the same web site analyzing items by using Mr. Peng's tool as using the tool of Website Analyzer. Such as link search, link category, link reference and link count. But there are some differences between them also.

Mr. Peng's Major report is more concentrated in the research of Web Mining and jMap language. His automatic tool focus on the functionalities of analyzing the website design. But my project is kind of web application using a better algorithm providing by Prof. Peter Grogono. The WSA tool is more helpful in the web site maintenance.

Mr. Peng's tool can mine a lot of general information from the web site. In my tool I just focus on to find the bad links and orphaned files in the web site, then create a tree for tracking them easily by the webmaster to repair and maintain the web site.

Mr. Peng's project is using Perl and VBA to develop the tool, and Java is used in the WSA.

Therefore, the two tools focus on different application domains separately even there is some overlap. It is a good idea to analyze a same web site by using the two different tools to compare the analyzing result since there are the same functional items.

References

[1] NetMechanic, Power Tools for Your Web Site
http://www.netmechanic.com/link_check.htm

[2] SubmitShop.Com, Link Checker – A Free Web Promotion Tool
http://www.submitshop.com/tools/linkchecker.html

[3] KyoSoft.com, Link Checker Pro
http://www.kyosoft.com/linkchecker/index.htm

[4] Elsop.com, LinkScan User Guide
http://www.elsop.com/linkscan/docs/linkscan.html

[5] University of Leicester, Broken Link Checker
http://www.le.ac.uk/cc/www/tools/linkchecker/

[6] Xenu's Link Sleuth, Find broken links on web sites
http://home.snafu.de/tilman/xenulink.html

[7] IndiaBook.com, Free Link Checker
http://www.indiabook.com/webmaster/link.html

[8] REL.software, Web Link Validator: The Broken Links Doctor
http://www.relsoftware.com/wlv/

[9] BiggByte.com, InfoLink Link Checker Features
http://www.biggbyte.com/infolink/features/features.html

[10] Link Alarm.com, Protect Your Site From Link Failure
http://www.linkalarm.com/

[11] NorthernWebs.com, Link Checker Version 2.0
http://www.northernwebs.com/set/spider_view.html

[12] Cay S. Horstmann, Gary Cornell, Core Java 2 Volume I – Fundamentals, Sun Microsystems, Inc. 2000

[13] Cay S. Horstmann, Gary Cornell, Core Java 2 Volume II – Advanced Features, Sun Microsystems, Inc. 2000

[14] Puping Peng, Concordia University, Web Mining with jMap Technology, February 2002

Appendix A: Sample implementation Java codes

```
/***********************************************************************
* LinkCheck.java
* Author: Lan Jin
* Copyright: Copyright (c) 2003
* Version 1.0
* Description: Program to verify links on current website.
* Traverses a site using the first page input by a user and checks all
* links on the first page, and adds them to an internal and external list.
* If the page is internal to the web site, then it gets checked and links
* pulled from it and sorted. Finally a tree is created.
***********************************************************************/
import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.applet.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.tree.*;
import java.awt.event.*;

public class LinkCheck
{

  static Vector externalList = new Vector();
  static Vector internalList = new Vector();
  static Vector leaveList = new Vector();
  static Vector badList = new Vector();
  static Vector nopermitList = new Vector();
  static Vector orphanList = new Vector();
  static Vector toBeCheckedInternalList = new Vector();
  String top = "";
  int pageCount = 1;
  String sourceFile = "";
  DefaultMutableTreeNode root = new DefaultMutableTreeNode("home");
  DefaultMutableTreeNode subpage = null;
  LinkMain chklink;
  LinkReport report = null;
  JTree tree = null;
  String base, localRoot;
  boolean isLink = true;
  int intcnt, lvcnt, excnt;
  public LinkCheck(LinkMain glcv){
    chklink = glcv;
```

```
}
// main loop
public void Main (String beginningPage, JTextArea out )throws Exception
{
 URLConnection urlconnection=null, urlconn=null;
 File mainfile=null, file=null;
 int contentlength=0, contentlgth=0;
 //check if it is a local path
 if (beginningPage.indexOf("http:")<0){
  isLink = false;
 }

 // first -- home page
 LinkInfo first = new LinkInfo("Start page ", beginningPage,
                               beginningPage, isLink);
 top = trimUrl(beginningPage);  //get page name without path

 // If it is or not site/page
 boolean testflag = false;
 String webpage = "home";//root of the tree

 // prime main loop with user input

 if (isLink){
  urlconnection = first.link.openConnection();
  urlconnection.setAllowUserInteraction(true);
  contentlength = urlconnection.getContentLength();
  base = "http://" + urlconnection.getURL().getHost();
  String path = urlconnection.getURL().getPath();
  if (path=="") {
   base = base+"/";
  }else{
   base = base + path;
  }
 }else{
  mainfile = new File(beginningPage);
  contentlength = mainfile.getName().length();
  base = top;
  localRoot = top;
 }
 chklink.output.setText(" "+beginningPage);
 internalList.addElement(first);
 parsePage(first, mainfile, urlconnection, contentlength, webpage, chklink.output);
 if (!chklink.isReady){
  chklink.stop();
  return;
```

```
}

while ( !(toBeCheckedInternalList.isEmpty()) ) {

 // get top link off internal list
 LinkInfo tempLI = (LinkInfo)toBeCheckedInternalList.firstElement();

 // new top link
 top = trimUrl(tempLI.stringLink);
 webpage = nameOfPage(tempLI.stringLink);

 pageCount ++;

 if (isLink){
  try{
   urlconn = tempLI.link.openConnection();
   urlconn.setAllowUserInteraction(true);
   InputStream testConnection = tempLI.link.openStream();
   testflag = true;
  }catch(UnknownHostException une){
   toBeCheckedInternalList.removeElementAt(0);
   tempLI.setInfo("Error: Unknown host.");
   addElementToList(badList, tempLI);
   setUnaccessedPage(tempLI);
   continue;
  }catch(MalformedURLException mfe){
   toBeCheckedInternalList.removeElementAt(0);
   tempLI.setInfo("Error: Malformed URL.");
   addElementToList(badList, tempLI);
   setUnaccessedPage(tempLI);
   continue;
  }catch(FileNotFoundException fnex){
   toBeCheckedInternalList.removeElementAt(0);
   tempLI.setInfo("Error: File not found.");
   addElementToList(badList, tempLI);
   setUnaccessedPage(tempLI);
   continue;
  }catch(IOException ioex){
   toBeCheckedInternalList.removeElementAt(0);
   tempLI.setInfo("Error: Permission denied.");
   addElementToList(nopermitList, tempLI);
   setUnaccessedPage(tempLI);
   continue;
  }
 }else{
  try{
      file = new File(tempLI.stringLink);
      InputStream in = new FileInputStream(file);
      testflag = true;
    }catch(Exception fnex){
      toBeCheckedInternalList.removeElementAt(0);
      tempLI.setInfo("Error: File not found.");
      addElementToList(badList, tempLI);
      setUnaccessedPage(tempLI);
      continue;
    }
  }

  if(testflag){// It is a site or page

    try{

      if (isLink){
        urlconn = tempLI.link.openConnection();
        urlconn.setAllowUserInteraction(true);
        contentlgth = urlconn.getContentLength();
      }else{
        file = new File(tempLI.stringLink);
        contentlgth = file.getName().length();
      }

      // find links out of this page

      if (!chklink.isReady){
        chklink.stop();
        return;
      }
      parsePage(tempLI, file, urlconn, contentlgth, webpage, chklink.output);

      // remove the parsered link
      toBeCheckedInternalList.removeElementAt(0);

    }catch(IOException e){

      //add to bad internal list
      badList.addElement(tempLI);
      toBeCheckedInternalList.removeElementAt(0);
    }
  }
}

chklink.status.setText(" Link checking ...");
chklink.status.update(chklink.getGraphics());

//now check links outside our site
//System.out.println("\n\n\nChecking external links: ");
out.append("\n\n\n Checked " + pageCount + " pages on website or local folder\n");

Enumeration e2 = internalList.elements();
pageCount = 0;
while(e2.hasMoreElements() ){
  pageCount++;
  LinkInfo tempLinkInfo2 = (LinkInfo)e2.nextElement();
}
out.append("\n Internal links = " + internalList.size() + "\n");

textScrollDisplay(chklink.output);
//while still links in external list...
pageCount = 0;
Enumeration e5 = externalList.elements();
while(e5.hasMoreElements() ){
  pageCount ++;
  LinkInfo tempLinkInfo5 = (LinkInfo)e5.nextElement();
  //check if host site is available?
  if (tempLinkInfo5.stringLink.startsWith("mailto:") ||
      tempLinkInfo5.stringLink.startsWith("ftp:"))
    continue;
  if (isLink || tempLinkInfo5.stringLink.startsWith("http:")){
    try{
      InputStream testConnection = tempLinkInfo5.link.openStream();
      chklink.status.setText(" Link checking ..." +
                      tempLinkInfo5.stringLink+" -- OK");
    }catch(UnknownHostException e){
      chklink.status.setText(" Link checking ..." +
                      tempLinkInfo5.stringLink+" -- Unknown host");
      tempLinkInfo5.setInfo("Error: Unknown host.");
      badList.addElement(tempLinkInfo5);
      externalList.removeElementAt(--pageCount);
      continue;
    }catch(MalformedURLException e){
      chklink.status.setText(" Link checking ..." +
                      tempLinkInfo5.stringLink+" -- Malformed URL");
      tempLinkInfo5.setInfo("Error: Malformed URL.");
      badList.addElement(tempLinkInfo5);
      externalList.removeElementAt(--pageCount);
      continue;
    }catch(FileNotFoundException fnex){
      chklink.status.setText(" Link checking ..." +
                    tempLinkInfo5.stringLink+" -- File not found");
        tempLinkInfo5.setInfo("Error: File not found.");
        badList.addElement(tempLinkInfo5);
        externalList.removeElementAt(--pageCount);
        continue;
      }catch(IOException ioex){
        chklink.status.setText(" Link checking ..." +
                    tempLinkInfo5.stringLink+" -- Permission denied");
        tempLinkInfo5.setInfo("Error: Permission denied.");
        nopermitList.addElement(tempLinkInfo5);
        externalList.removeElementAt(--pageCount);
        continue;
      }
    }
  }//end enumeration loop

  out.append("\n External link size = " + externalList.size() + "\n");
  textScrollDisplay(chklink.output);

  Enumeration e4 = leaveList.elements();
  pageCount = 0;
  while(e4.hasMoreElements() ){
    pageCount++;
    LinkInfo tempLinkInfo4 = (LinkInfo)e4.nextElement();
    if (tempLinkInfo4.stringLink.substring(0,7).equalsIgnoreCase("mailto:"))
      continue;
    if (isLink || tempLinkInfo4.stringLink.indexOf("http:")>=0){
      try{
        InputStream testConnection = tempLinkInfo4.link.openStream();
        chklink.status.setText(" Link checking ... " +
                    tempLinkInfo4.stringLink+" -- OK");
      }catch(UnknownHostException e){
        chklink.status.setText(" Link checking ... " +
                    tempLinkInfo4.stringLink+" -- Unknown host");
        tempLinkInfo4.setInfo("Error: Unknown host.");
        badList.addElement(tempLinkInfo4);
        leaveList.removeElementAt(--pageCount);
        continue;
      }catch(MalformedURLException e){
        chklink.status.setText(" Link checking ... " +
                    tempLinkInfo4.stringLink+" -- Malformed URL");
        tempLinkInfo4.setInfo("Error: Malformed URL.");
        badList.addElement(tempLinkInfo4);
        leaveList.removeElementAt(--pageCount);
        continue;
      }catch(FileNotFoundException fnex){
```

```
      chklink.status.setText(" Link checking ... " +
                   tempLinkInfo4.stringLink+" -- File not found");
      tempLinkInfo4.setInfo("Error: file not found.");
      badList.addElement(tempLinkInfo4);
      leaveList.removeElementAt(--pageCount);
      continue;
    }catch(IOException ioex){
      chklink.status.setText(" Link checking ... " +
                   tempLinkInfo4.stringLink+" -- Permission denied");
      tempLinkInfo4.setInfo("Error: Permission denied.");
      nopermitList.addElement(tempLinkInfo4);
      leaveList.removeElementAt(--pageCount);
      continue;
    }
  }else if(!isLink){
    try{
      file = new File(tempLinkInfo4.stringLink);
      if (file.isDirectory() || file.isFile()){
        chklink.status.setText(" Link checking ... " +
                   tempLinkInfo4.stringLink+" -- OK");
      }else{
        chklink.status.setText(" Link checking ... "
                    + tempLinkInfo4.stringLink+" -- "
                    + "File not found");
      }
    }catch(Exception e){
      chklink.status.setText(" Link checking ... "
                   + tempLinkInfo4.stringLink+" -- "
                   + e.getMessage());
      tempLinkInfo4.setInfo("Error: " + e.getMessage());
      badList.addElement(tempLinkInfo4);
      leaveList.removeElementAt(--pageCount);
      continue;
    }
  }
}
out.append("\n Leaf link size = " + leaveList.size() + "\n");
textScrollDisplay(chklink.output);

Enumeration e3 = badList.elements();
pageCount = 0;
while(e3.hasMoreElements() ){
  pageCount++;
  LinkInfo tempLinkInfo3 = (LinkInfo)e3.nextElement();

}
```

```
    out.append("\n Bad links = " + pageCount + "\n");
    textScrollDisplay(chklink.output);

    //print list "no permission links"
    Enumeration npkey = nopermitList.elements();
    pageCount = 0;
    while(npkey.hasMoreElements() ){
      pageCount++;
      LinkInfo noplink = (LinkInfo)npkey.nextElement();

    }
    out.append("\n Permission denied links = " + pageCount + "\n");
    if (!isLink) {
      orphanList = checkOrphan(localRoot);
      out.append("\n Orphaned pages = " + orphanList.size() + "\n");
    }

    pageCount = internalList.size() + externalList.size() +
            leaveList.size() + badList.size() + nopermitList.size();
    out.append("\n Total links = " + pageCount + "\n");
    textScrollDisplay(chklink.output);

    tree = new JTree(root);
    tree.putClientProperty("JTree.lineStyle", "Angled");
    //chklink.treePanel.add(tree);
    chklink.treePanel.add(new JScrollPane(tree));
    chklink.treePanel.setBorder(BorderFactory
                    .createBevelBorder(0,Color.darkGray,Color.gray));
    chklink.mainPanel.add(chklink.treePanel,BorderLayout.EAST);
    chklink.status.setText(" Done.");
    chklink.repaint();
    report = new LinkReport(internalList,externalList,leaveList,
                    badList,nopermitList,orphanList);
  }// end main


//*********************************************************************
  // collect links found in pages on website and sort for checking

//*********************************************************************
  public void parsePage(LinkInfo info, File file,
                URLConnection urlconnection,
                int contentlength,
                String pagename,
                JTextArea linkout) throws Exception
```

```
{
 String[] links = new String[1000];
 String[] leave = new String[1000];
 String[] exlnk = new String[1000];
 String link = "", path = "";
 boolean isduplink=false, isdupleaf=false;
 //String base = chklink.getCodeBase().toString();
 intcnt = 0; lvcnt = 0; excnt = 0;
 int character;
 String source=null;
 if (isLink) {
   source = urlconnection.getURL().toString();
 }else{
   source = info.stringLink;
 }
 boolean frameFlag = false;
 boolean imageFlag = false;
 boolean anchorFlag = false;
 FileInputStream finput=null;

 //parse page and get links
 InputStream in;
 if (isLink){
   in = urlconnection.getInputStream();
 }else{
   in = new FileInputStream(file);
 }

 //if link external to site add to external list and page it is on
 //if internal add to back of internal link list if not already there
 while ( (character = in.read() ) != -1 ){

   //*find all links excepting in frames
   //if it begins '<a'  it's a link....
   if( (char)character == '<'){

     character = in.read();

     if( ( (char)character == 'a') || ( (char)character == 'A') ){
       character = in.read();

       //*dump href="
       String tmpstr = "";
       if (character == '>') continue;
       while( (char)character  != '"' && (char)character!='>'){//first '"'
         character = in.read();
```

```
  tmpstr += String.valueOf((char)character).toString();
 }
 if ((char)character=='>') {
  continue;
 }
 if (tmpstr.indexOf("name=")>=0){
  anchorFlag = true;//not real link
 }else if (tmpstr.indexOf("align=")>=0){
  anchorFlag = true;
 }else if (tmpstr.indexOf("class=")>=0){
  character = in.read();
  while( (char)character !='"')
   character = in.read();
  character = in.read();
  tmpstr = "";
  while( (char)character !='"'){
   character = in.read();
   tmpstr += String.valueOf((char)character).toString();
  }
  if (tmpstr.indexOf("href=")<0) continue;
 }

 character = in.read(); // skip over first quotation mark
 while( (char)character != '"'){
  if( (char)character == '#'){  //anchor not a real link, dump it
   anchorFlag = true;
  }
  link += (char)character;
  character = in.read();
 }

 if(!anchorFlag){
  if (!link.trim().equals("")){
   findURLPath(links, leave, link.trim(), linkout);
  }
 }
 link = "";
 anchorFlag = false;

 //add in frame checking
}
else if ( ( (char)character == 'f') || ( (char)character == 'F') ) {
 String tmpstr = String.valueOf((char)character).toString();

 //dump everything before src=" and grab the stuff between the quotes
 while( (char)character != '>' ){
    character = in.read();
    tmpstr += String.valueOf((char)character).toString();
  }
  int p1 = tmpstr.indexOf("src");
  int p2 = tmpstr.indexOf("SRC");
  if (p1<0 && p2<0) continue;

  int p=0;
  if (p1>0) { p=p1; }
  else if(p2>0){ p=p2; }
  tmpstr = tmpstr.substring(p+5);
  p = tmpstr.indexOf("\"");
  if (p>=0) {
    link = tmpstr.substring(0,p);
    frameFlag = true;
  }else{
    continue;
  }
  if( (frameFlag)&& (!anchorFlag)){
    if (!link.trim().equals("")){
      findURLPath(links, leave, link.trim(), linkout);
    }
  }
  link = "";
  frameFlag = false;

}//end char='f' || 'F'
else if ( ( (char)character == 'i') || ( (char)character == 'I') ) {
  String tmpstr = String.valueOf((char)character).toString();

  //dump everything before src=" and grab the stuff between the quotes
  while( (char)character != '>' ){
    character = in.read();
    tmpstr += String.valueOf((char)character).toString();
  }
  if (tmpstr.indexOf("img")<0 && tmpstr.indexOf("IMG")<0) continue;
  int p1 = tmpstr.indexOf("src");
  int p2 = tmpstr.indexOf("SRC");
  if (p1<0 && p2<0) continue;

  int p=0;
  if (p1>0) { p=p1; }
  else if(p2>0){ p=p2; }
  tmpstr = tmpstr.substring(p+5);
  p = tmpstr.indexOf("\"");
  if (p>=0) {
        link = tmpstr.substring(0,p);
        imageFlag = true;
      }else{
        continue;
      }
      if( (imageFlag)&& (!anchorFlag)){
        if (!link.trim().equals("")){
          findURLPath(links, leave, link.trim(), linkout);
        }
      }
     link = "";
     imageFlag = false;
    }//end char='i' || 'I'

  }//>
}// end while loop find next link
in.close();

DefaultMutableTreeNode subnode = findUserObject(pagename);

// sort into internal and external and fix up link formatting.
for(int i=0; i<intcnt; i++){

  String inLink;

  // ditch ftp links and mailto links
  if( inOrOut(links[i], top) ){
    // add to internal links if not already there and add to toBeCheckedInternal
    inLink = fixUp(links[i], top); //find whole path of the link
    links[i] = inLink;


    // ditch javascript links...
    boolean javascript = false;
    if( (links[i].indexOf("javaS")>0) || (links[i].indexOf("Javas")>0)
       || (links[i].indexOf("JAVAS")>0) || (links[i].indexOf("JavaS")>0)
       || (links[i].indexOf("javas")>0)){
      javascript = true;
    }

    // check for flash links and ditch them...
    boolean flash = false;
    if( links[i].indexOf("clsid") > 0){
      flash = true;
    }
  // duplicate checking: replace duplicate internal links
  LinkInfo tempLinkInfo = new LinkInfo(source, top, links[i], isLink);
  Enumeration e = internalList.elements();
  boolean flag = false;
  while(e.hasMoreElements()){
    LinkInfo tempLinkInfo2 = (LinkInfo)e.nextElement();
    if( (tempLinkInfo2.stringLink).equals(tempLinkInfo.stringLink) ){
      info.otherLocations.addElement(links[i]);
      tempLinkInfo2.references.addElement(source);
      flag = true;
      break;
    }
  }

  // if not in internal list, add it in.
  if( (!javascript)&&(!flash) ){
    if (!flag){
      System.out.println( "adding to internal list " + links[i]);
      info.otherLocations.addElement(links[i]);
      LinkInfo intLink = new LinkInfo(source, top, links[i], isLink);
      intLink.references.addElement(source);
      internalList.addElement(intLink);
      toBeCheckedInternalList.addElement(new
          LinkInfo(source, top,links[i], isLink));
    }
    String substr = nameOfPage(links[i]);
    DefaultMutableTreeNode subpage = new DefaultMutableTreeNode(substr);
    subnode.add(subpage);
    flag = false;
  }

}
// add to external links if not a duplicate
else{

  //check for duplicates and keep running list of
  //source file info so user knows where to find
  //links that have to be fixed.

  LinkInfo tempLinkInfo1 = new LinkInfo(source, top, links[i], isLink);
  Enumeration e1 = externalList.elements();
  boolean flag1 = false;

  while(e1.hasMoreElements() ){
    LinkInfo tempLinkInfo2 = (LinkInfo)e1.nextElement();
```

```
      if( (tempLinkInfo2.stringLink).equals(tempLinkInfo1.stringLink) ){
        info.otherLocations.addElement(links[i]);
        tempLinkInfo2.otherLocations.addElement(tempLinkInfo1.sourceFile);
        tempLinkInfo2.references.addElement(source);
        flag1 = true;
        break;
      }
    }//end while loop

    if (!flag1){
      LinkInfo intLink = new LinkInfo(source, top, links[i], isLink);
      intLink.references.addElement(source);
      externalList.addElement(intLink);
      info.otherLocations.addElement(links[i]);
      System.out.println( "adding " + links[i] + " to external list");
    }
    String substr = nameOfPage(links[i]);
    DefaultMutableTreeNode subpage = new DefaultMutableTreeNode(substr);
    subnode.add(subpage);
  }
}//for i loop

// sort into leaf list and fix up link formatting.
for(int i=0; i<lvcnt; i++){
  String inLink;
  if( inOrOut(leave[i], top) ){
    // add to leaf links if not already there and add to toBeCheckedInternal
    inLink = fixUp(leave[i], top);
    leave[i] = inLink;
    LinkInfo tempLinkInfo1 = new LinkInfo(source, top, leave[i], isLink);
    Enumeration e1 = leaveList.elements();
    boolean flag1 = false;

    while(e1.hasMoreElements() ){
      LinkInfo tempLinkInfo2 = (LinkInfo)e1.nextElement();
      if (isLink){
        if ((tempLinkInfo2.link).equals(tempLinkInfo1.link)){
          info.otherLocations.addElement(leave[i]);
          tempLinkInfo2.references.addElement(source);
          flag1 = true;
          break;
        }
      }else{
        if( (tempLinkInfo2.stringLink).equals(tempLinkInfo1.stringLink) ){
          info.otherLocations.addElement(leave[i]);
          tempLinkInfo2.references.addElement(source);
          flag1 = true;
          break;
         }
        }

       }//end while loop

       if (!flag1){
         LinkInfo intLink = new LinkInfo(source, top, leave[i], isLink);
         intLink.references.addElement(source);
         leaveList.addElement( intLink);
         info.otherLocations.addElement(leave[i]);
         System.out.println( "adding " + leave[i] + " to leaf list");
       }
       String substr = nameOfPage(leave[i]);
       if (substr!=""){
         DefaultMutableTreeNode subpage = new DefaultMutableTreeNode(substr);
         subnode.add(subpage);
       }
      }
     }
    if (intcnt==0 && lvcnt==0){
     // no link found, remove from internal list
     Enumeration e = internalList.elements();
     int eid = 0;
     while(e.hasMoreElements()){
       LinkInfo tempLinkInfo = (LinkInfo)e.nextElement();
       if( (tempLinkInfo.stringLink).equals(info.stringLink) ){
         internalList.removeElementAt(eid);
         break;
       }
       eid++;
     }
     //add to leaf list
     Enumeration elm = leaveList.elements();
     boolean inflag = false;
     while(elm.hasMoreElements()){
       LinkInfo tempLinkInfo = (LinkInfo)elm.nextElement();
       if (tempLinkInfo.stringLink.equals(info.stringLink)){
         inflag = true;
         break;
       }
     }
     if (!inflag) leaveList.addElement(info);
    }else{
     // renew the LinkInfo if it exists in the internal list, otherwise
```

```
      // add the LinkInfo to the internal list
      Enumeration eint = internalList.elements();
      LinkInfo tempinfo = (LinkInfo)eint.nextElement();
      int id = 0;
      while(!tempinfo.stringLink.equals(info.stringLink) &&
        eint.hasMoreElements() ){
       id++;
       tempinfo = (LinkInfo)eint.nextElement();
      }
      internalList.set(id,info);
    }
    //scrolling text output to the end
    int nline = linkout.getLineCount();
    textScrollDisplay(linkout);
  }


//*************************************************************
//end parse method
//*************************************************************

  //check root folder and possible sub-folder to
  //find orphan pages or orphan files
  private Vector checkOrphan(String fpath){
    Vector orphan = new Vector();
    try {
      File pathName = new File(fpath);
      String[] fileNames = pathName.list();

      // enumerate all files in the directory
      for (int i = 0; i < fileNames.length; i++){
        File f = new File(pathName.getPath(), fileNames[i]);
        String fname = backToForwardSlash(f.getPath());
        // if the file is again a directory, call
        // the function recursively
        if (f.isDirectory()) {
          System.out.println(f.getCanonicalPath());
          orphan = checkOrphan(fname);
        }else{
          if (isOrphan(fname)){
            orphan.addElement(fname);
          }
        }
      }
    }catch(IOException e){
      e.printStackTrace();
  }
  return orphan;
 }

 private String backToForwardSlash(String fpath){
  String retString="", str=fpath;
  int p;
  while((p=str.indexOf("\\"))>=0){
   retString += str.substring(0,p) + "/";
   str = str.substring(p+1);
  }
  if (str.length()>0){
   retString += str;
  }
  return retString;
 }

 //check if it is in the internal list or leave list
 private boolean isOrphan(String fname){
  if(fname.endsWith(".html") ||
    fname.endsWith(".htm") ||
    fname.endsWith(".shtml") ||
    fname.endsWith(".xml") ||
    fname.endsWith(".asp") ||
    fname.endsWith(".jsp") ||
    fname.endsWith(".php")){
   Enumeration eintl = internalList.elements();
   while(eintl.hasMoreElements()){
    LinkInfo tempLinkInfo = (LinkInfo)eintl.nextElement();
    if( (tempLinkInfo.stringLink).equals(fname) ){
     return false;
    }
   }
  }
  Enumeration eleaf = leaveList.elements();
  while(eleaf.hasMoreElements()){
   LinkInfo tempLinkInfo = (LinkInfo)eleaf.nextElement();
   if( (tempLinkInfo.stringLink).equals(fname) ){
    return false;
   }
  }
  return true;
 }

 private void addElementToList(Vector list, LinkInfo info){
  // duplicate checking: replace duplicate internal links
 Enumeration ebad = list.elements();
 while(ebad.hasMoreElements()){
   LinkInfo tempLinkInfo = (LinkInfo)ebad.nextElement();
   if( (tempLinkInfo.stringLink).equals(info.stringLink) ){
     return;
   }
 }
 list.addElement(info);
}

private void setUnaccessedPage(LinkInfo info){
 // duplicate checking: replace duplicate internal links
 Enumeration e = internalList.elements();
 int eid = 0;
 while(e.hasMoreElements()){
   LinkInfo tempLinkInfo = (LinkInfo)e.nextElement();
   if( (tempLinkInfo.stringLink).equals(info.stringLink) ){
     internalList.removeElementAt(eid);
     break;
   }
   eid++;
 }
}

// display text which scrolled to the end of contents
private void textScrollDisplay(JTextArea linkout){
 Document doc = linkout.getDocument();
 Caret currentCaret = chklink.output.getCaret();
 currentCaret.setDot(doc.getLength());
}

private boolean isFolderLink(String lnk){
 int p = lnk.lastIndexOf("/");
 if (p==lnk.length()-1){
   return true;
 }else{
   if (p>=0){
     lnk = lnk.substring(p+1);
   }
   if (lnk.indexOf(".")<0){
     return true;
   }else{
     return false;
   }
 }
}
```

```
// add the found link to it belonging to: link or leaf
private void findURLPath(String[] links, String[] leave,
                         String link, JTextArea linkout){
  boolean isduplink=false, isdupleaf=false;
  String path;
  //check internal link
  if(link.endsWith(".html") ||
    link.endsWith(".htm") ||
    link.endsWith(".shtml") ||
    link.endsWith(".xml") ||
    link.endsWith(".asp") ||
    link.endsWith(".jsp") ||
    link.endsWith(".php") ||
    link.startsWith("ftp:") ||
    link.startsWith("mailto:") ||
    isFolderLink(link)){
     for(int k=0; k<intcnt; k++){
       if (links[k].compareTo(link)==0) {//remove duplicated link
         isduplink = true;
         return;
       }
     }
     if (!isduplink){
       links[intcnt] = link;
       intcnt++;
     }
  }else{
    //check leaf link
     for(int k=0; k<lvcnt; k++){
       if (leave[k].compareTo(link)==0) {//remove duplicated leaf
         isdupleaf = true;
         return;
       }
     }
     if (!isdupleaf){
       leave[lvcnt] = link;
       lvcnt++;
     }
  }
  if (link.indexOf("http:")<0 && link.indexOf("ftp:")<0 &&
      link.indexOf("mailto:")<0 && link.indexOf("?")<0 &&
      link.indexOf("gopher:")<0 && link.indexOf("file:")<0 &&
      link.indexOf("cgi_bin")<0){
    path = base + link;//full path of the internal link
  }else{
```

```
  path = link;//external link
 }
 //print out the link
 if (!isduplink && !isdupleaf){
  linkout.append("\n "+path);
 }
 chklink.status.setText(" "+path);
 chklink.repaint();
}

// determine if link internal or external
public boolean inOrOut (String lk, String base)
{

 if( lk.startsWith(base)){
  return true;
 }
 else{ // if not off top directory and begins with http must be external

  if( lk.startsWith("http:") ||
    lk.startsWith("mailto:") ||
    lk.startsWith("ftp:")){
   return false;

  }else { // anything left must be internal
   return true;
  }
 }
}

// determine some special links and formats
public String fixUp (String lk, String base)
{
 String temp = "";

 // patch together internal links if needed before adding to vector
 // does it begin with http? if so ok do nothing and return string
 if( lk.startsWith(base) ){
  return lk;
 // else add base
 }else if (lk.length()>7 &&
      (lk.substring(0,7).equalsIgnoreCase("mailto:") ||
      lk.substring(0,4).equalsIgnoreCase("ftp:") ||
      lk.substring(0,7).equalsIgnoreCase("gopher:") ||
      lk.substring(0,8).equalsIgnoreCase("cgi_bin:") ||
      lk.substring(0,5).equalsIgnoreCase("file:"))){
```

```
      return lk;
    }else if (lk.startsWith("../")){//go to parent
      temp = base;
      while(lk.startsWith("../")){
        lk = lk.substring(3);
        if (temp.substring(temp.length()-1).equals("/")){
          temp = temp.substring(0,temp.length()-1);
        }
        temp =temp.substring(0,temp.lastIndexOf("/"));
      }
      return (temp+"/"+lk);
    }else if(lk.startsWith("./")){// itself
      return (base + lk.substring(1));
    }else if(lk.startsWith("/")){// somewhere
      if (base.indexOf("/~")>=0){
        int p = base.indexOf("/~");
        return (base.substring(0, p) + lk);
      }else if (base.endsWith("/")){
        return (base.substring(0, base.length()-1) + lk);
      }else{
        return (base + lk);
      }
    }else if (lk.startsWith("/~")) {
       temp = base;
       while(temp.lastIndexOf("/")!=6){
         temp=temp.substring(0,temp.lastIndexOf("/"));
       }
       return (temp + lk);
    }else{
      return (base + lk);
    }
  }

  // get page file name without path
  public String trimUrl (String lk){
    // see if we have a file name at the end of our url or a directory
    // if it is a file name trim file name off of url, so when we
    // attach it to internal urls we don't get confused.

    int length = lk.length();

    // first see if we have a file on the end
    // if not just return the original url
    if( lk.endsWith(".HTML") || lk.endsWith(".html") || lk.endsWith(".Html")
        || lk.endsWith(".SHTML") || lk.endsWith(".shtml") || lk.endsWith(".Shtml")
        || lk.endsWith(".HTM") || lk.endsWith(".htm") || lk.endsWith(".Htm") ){
```

```
    // we need to trim string

    char temp[] = lk.toCharArray();
    char backwards[] = new char[length];
    String bw ="";

    // first reverse the string and trim back to first '/'
    int j=0;
    boolean flag = false;

    for( j=(length-1); j>=0; j-- ){

      if ((int)temp[j] == 47){
        flag = true;
      }

      if( flag )
        bw += temp[j];
    }

    // flip trimmed string back around
    char temp2[] = bw.toCharArray();
    String tempString = "";

    for( int k=(bw.length()-1); k>0; k--){
      tempString += temp2[k];
    }
    return tempString + "/";
  }else{ // or all is well just return original string
    return lk + "/";
  }
}

//find the specific tree node
public DefaultMutableTreeNode findUserObject(Object obj){
  Enumeration e = root.breadthFirstEnumeration();
  while(e.hasMoreElements()){
    DefaultMutableTreeNode node =
        (DefaultMutableTreeNode)e.nextElement();
    if(node.getUserObject().equals(obj)){
      return node;
    }
  }
  return null;
}
```

```
  //pasing link path
  public String nameOfPage(String nlnk){
    String str = nlnk, tmpstr = nlnk;
    int tmpid = tmpstr.indexOf("/");
    while(tmpid>=0){
      tmpstr = str;
      str = tmpstr.substring(tmpid+1);
      tmpid = str.indexOf("/");
    }
    if (str.equals("")){
      return tmpstr;
    }else{
      return str;
    }
  }

  public void clearAll(){
    toBeCheckedInternalList.removeAllElements();
    internalList.removeAllElements();
    externalList.removeAllElements();
    leaveList.removeAllElements();
    badList.removeAllElements();
    nopermitList.removeAllElements();
    orphanList.removeAllElements();
    tree.removeAll();
    report = null;
  }
}
```

Appendix B: Format of the web checking report

Web Link Analysis Report

Root URL: d:/jwsdp-1_0/docs/tutorial/index.html

Table of Contents

The parent/internal URLs:

```
d:/jwsdp-1_0/docs/tutorial/index.html:
    \__d:/jwsdp-1_0/docs/tutorial/index.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorialTOC.html
    \__d:/jwsdp-1_0/docs/api/index.html
    \__d:/jwsdp-1_0/docs/index.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/Preface.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/GettingStarted.html

..........


d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorialTOC.html:
    \__d:/jwsdp-1_0/docs/tutorial/index.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/Copyright.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/images/blueline.gif
    \__d:/jwsdp-1_0/docs/tutorial/doc/images/tm.gif
d:/jwsdp-1_0/docs/index.html:
    \__d:/jwsdp-1_0/docs/api/index.html
    \__d:/jwsdp-1_0/docs/ReleaseNotes.html
    \__d:/jwsdp-1_0/docs/jaxm/index.html
    \__d:/jwsdp-1_0/docs/jaxp/index.html
    \__d:/jwsdp-1_0/docs/jaxr/index.html
    \__d:/jwsdp-1_0/docs/jaxrpc/index.html
    \__d:/jwsdp-1_0/docs/jstl/index.html
    \__d:/jwsdp-1_0/docs/tomcat/index.html
    \__d:/jwsdp-1_0/docs/ant/index.html
    \__d:/jwsdp-1_0/docs/registry-server/index.html
    \__d:/jwsdp-1_0/docs/tutorial/doc/Copyright.html
    \__d:/jwsdp-1_0/docs/jwsdponj2ee.html
    \__mailto:jwsdp-feedback@sun.com
    \__d:/jwsdp-1_0/docs/Copyright.html
d:/jwsdp-1_0/docs/tutorial/doc/Preface.html:
      \   d:/jwsdp-1_0/docs/tutorial/index.html
      \   d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorialTOC.html
      \   d:/jwsdp-1_0/docs/tutorial/doc/Preface.html
      \   d:/jwsdp-1_0/docs/tutorial/doc/IntroWS.html
      \   http://java.sun.com/docs/books/tutorial/jdbc
      \   http://java.sun.com/docs/books/tutorial/essential/threads
      \   http://java.sun.com/docs/books/tutorial/javabeans

   \   http://java.sun.com/webservices/downloads/webservicestutorial.html
      \   http://java.sun.com/webservices/downloads/webservicespack.html
      \   http://java.sun.com/j2se/1.3/
      \   http://java.sun.com/j2se/1.4/
      \   d:/jwsdp-1_0/docs/tutorial/doc/Copyright.html
      \   d:/jwsdp-1_0/docs/tutorial/doc/images/UpArrow.gif
      \   d:/jwsdp-1_0/docs/tutorial/doc/images/NextArrow.gif
      \   d:/jwsdp-1_0/docs/tutorial/doc/images/blueline.gif
      \   d:/jwsdp-1_0/docs/tutorial/doc/images/tm.gif
      \   d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorial.pdf
      \   d:/jwsdp-1_0/docs/tutorial/doc/images/arrwrite.gif

.........
```

Return to Top

```
======================================================================
```

The reference URLs:

```
d:/jwsdp-1_0/docs/tutorial/index.html
   \   d:/jwsdp-1_0/docs/tutorial/index.html
   \   d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorialTOC.html
   \   d:/jwsdp-1_0/docs/tutorial/doc/Preface.html
   \   d:/jwsdp-1_0/docs/tutorial/doc/GettingStarted.html
   \   d:/jwsdp-1_0/docs/tutorial/doc/IntroWS.html
   \   d:/jwsdp-1_0/docs/tutorial/doc/WebApp.html

.........

d:/jwsdp-1_0/docs/tutorial/examples/jaxp/sax/samples/Echo08.java
   \   d:/jwsdp-1_0/docs/tutorial/doc/JAXPSAX9.html
d:/jwsdp-1_0/docs/tutorial/examples/jaxp/sax/samples/Echo08-05.txt
   \   d:/jwsdp-1_0/docs/tutorial/doc/JAXPSAX9.html
d:/jwsdp-1_0/docs/tutorial/examples/jaxp/sax/samples/Echo09.java
   \   d:/jwsdp-1_0/docs/tutorial/doc/JAXPSAX9.html
d:/jwsdp-1_0/docs/tutorial/examples/jaxp/sax/samples/Echo09-06.txt
   \   d:/jwsdp-1_0/docs/tutorial/doc/JAXPSAX10.html
d:/jwsdp-1_0/docs/tutorial/examples/jaxp/sax/samples/Echo09-07.txt
   \   d:/jwsdp-1_0/docs/tutorial/doc/JAXPSAX10.html
```

Return to Top

```
======================================================================
```

The external URLs:

mailto:jwsdp-feedback@sun.com

```
http://java.sun.com/docs/books/tutorial/jdbc
http://java.sun.com/docs/books/tutorial/essential/threads
http://java.sun.com/docs/books/tutorial/javabeans
http://java.sun.com/webservices/downloads/webservicestutorial.html
http://java.sun.com/webservices/downloads/webservicespack.html
http://java.sun.com/j2se/1.3/
http://java.sun.com/j2se/1.4/
http://java.sun.com/products/jsp/taglibraries.html
http://java.sun.com/products/javabeans
http://java.sun.com/docs/books/tutorial/security1.2/index.html
ftp://ftp.isi.edu/in-notes/rfc1945.txt
ftp://ftp.isi.edu/in-notes/rfc2616.txt
http://www.rfc-editor.org/rfc.html
http://www.sun.com/patents
http://www.oasis-open.org/cover/elementsAndAttrs.html
```

....

Return to Top

```
===================================================================
```

The Leaf URL:

```
d:/jwsdp-1_0/docs/tutorial/doc/images/blueline.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/tm.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/UpArrow.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/NextArrow.gif
d:/jwsdp-1_0/docs/tutorial/doc/JavaWSTutorial.pdf
d:/jwsdp-1_0/docs/tutorial/doc/images/arrwrite.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/PreviousArrow.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/Fig20.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/Fig21.gif
d:/jwsdp-1_0/docs/tutorial/examples/web/hello1/src/GreetingServlet.java
d:/jwsdp-1_0/docs/tutorial/examples/web/hello1/src/ResponseServlet.java
d:/jwsdp-1_0/docs/tutorial/examples/web/hello2/web/greeting.txt
d:/jwsdp-1_0/docs/tutorial/examples/web/hello2/web/response.txt
```

....

Return to Top

```
===================================================================
```

The bad URLs:

```
d:/jwsdp-1_0/docs/api/index.html
   Error: File not found.
   \___In Page: d:/jwsdp-1_0/docs/tutorial/index.html
d:/jwsdp-1_0/docs/ReleaseNotes.html
   Error: File not found.
   \___In Page: d:/jwsdp-1_0/docs/index.html
d:/jwsdp-1_0/docs/jaxm/index.html
   Error: File not found.
   \___In Page: d:/jwsdp-1_0/docs/index.html
```

```
d:/jwsdp-1_0/docs/jaxp/index.html
   Error: File not found.
   \___In Page: d:/jwsdp-1_0/docs/index.html
d:/jwsdp-1_0/docs/jaxr/index.html
   Error: File not found.
   \___In Page: d:/jwsdp-1_0/docs/index.html

..........
```

Return to Top

```
========================================================================
```

The permission denied URLs:

```
http://localhost:8080/jstl-examples/index.html
   Error: Permission denied.
   \___In Page: d:/jwsdp-1_0/docs/tutorial/doc/JSTL9.html
```

Return to Top

```
========================================================================
```

The orphaned files:

```
d:/jwsdp-1_0/docs/tutorial/doc/images/Fig7.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/backgrnd.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/dom-api.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/p210e.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/Prefacea.gif
d:/jwsdp-1_0/docs/tutorial/doc/images/JAXRArchitecture.gif
d:/jwsdp-1_0/docs/tutorial/doc/IntroXML5.html
d:/jwsdp-1_0/docs/tutorial/doc/IntroWS10.html
d:/jwsdp-1_0/docs/tutorial/doc/GettingStarted11.html
d:/jwsdp-1_0/docs/tutorial/doc/ant_in_anger.html
```

Return to Top

```
========================================================================
```

The Summary of Web Analysis:

Internal links	**198**
External links	**95**
Leaf nodes	**267**
Bad links	**56**
Permission denied links	**1**
orphaned files	**10**

Return to Top