# CourseCrawler Product Technical Documentation

## Contents

**1 UML Diagrams** **3**

1.1 Main Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2 Html_Engine Package . . . . . . . . . . . . . . . . . . . . . . . . . 4

**2 Program Structure** **5**

**3 Class and Package Descriptions and Specifications** **5**

3.1 Database and Backend . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1.1 Database and Initialization . . . . . . . . . . . . . . . . . . . 5

3.1.2 Database Table Configuration . . . . . . . . . . . . . . . . 6

3.1.3 cc_db Class . . . . . . . . . . . . . . . . . . . . . . . . . . 9

3.2 HTML Engine Package . . . . . . . . . . . . . . . . . . . . . . . . 14

3.2.1 Control Class . . . . . . . . . . . . . . . . . . . . . . . . . 15

3.2.2 DataGrid Class . . . . . . . . . . . . . . . . . . . . . . . . 16

3.2.3 Form Class . . . . . . . . . . . . . . . . . . . . . . . . . . 17

3.2.4 Image Class . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2.5 Input Class . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.2.6 Label Class . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.2.7 Link Class . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3.2.8 List Class . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

3.2.9 Page Class . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

3.2.10 Table Class . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.2.11 TableCell Class . . . . . . . . . . . . . . . . . . . . . . . 25
3.2.12 TableRow Class . . . . . . . . . . . . . . . . . . . . . . . 26
3.3 Frontend Classes . . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.3.1 CourseWebserver . . . . . . . . . . . . . . . . . . . . . . . 27
3.3.2 DynamicPages . . . . . . . . . . . . . . . . . . . . . . . . 29
3.3.3 Html . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
3.3.4 Http . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.3.5 LogWriter . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.3.6 PostData . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
3.3.7 QueryString . . . . . . . . . . . . . . . . . . . . . . . . . 36
3.4 Crawler Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
3.4.1 Crawler Class . . . . . . . . . . . . . . . . . . . . . . . . . 38
3.4.2 Parser Class . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.4.3 PageContent Class . . . . . . . . . . . . . . . . . . . . . . 42
3.4.4 PageContentList Class . . . . . . . . . . . . . . . . . . . . 42
3.4.5 Semaphore . . . . . . . . . . . . . . . . . . . . . . . . . . 43

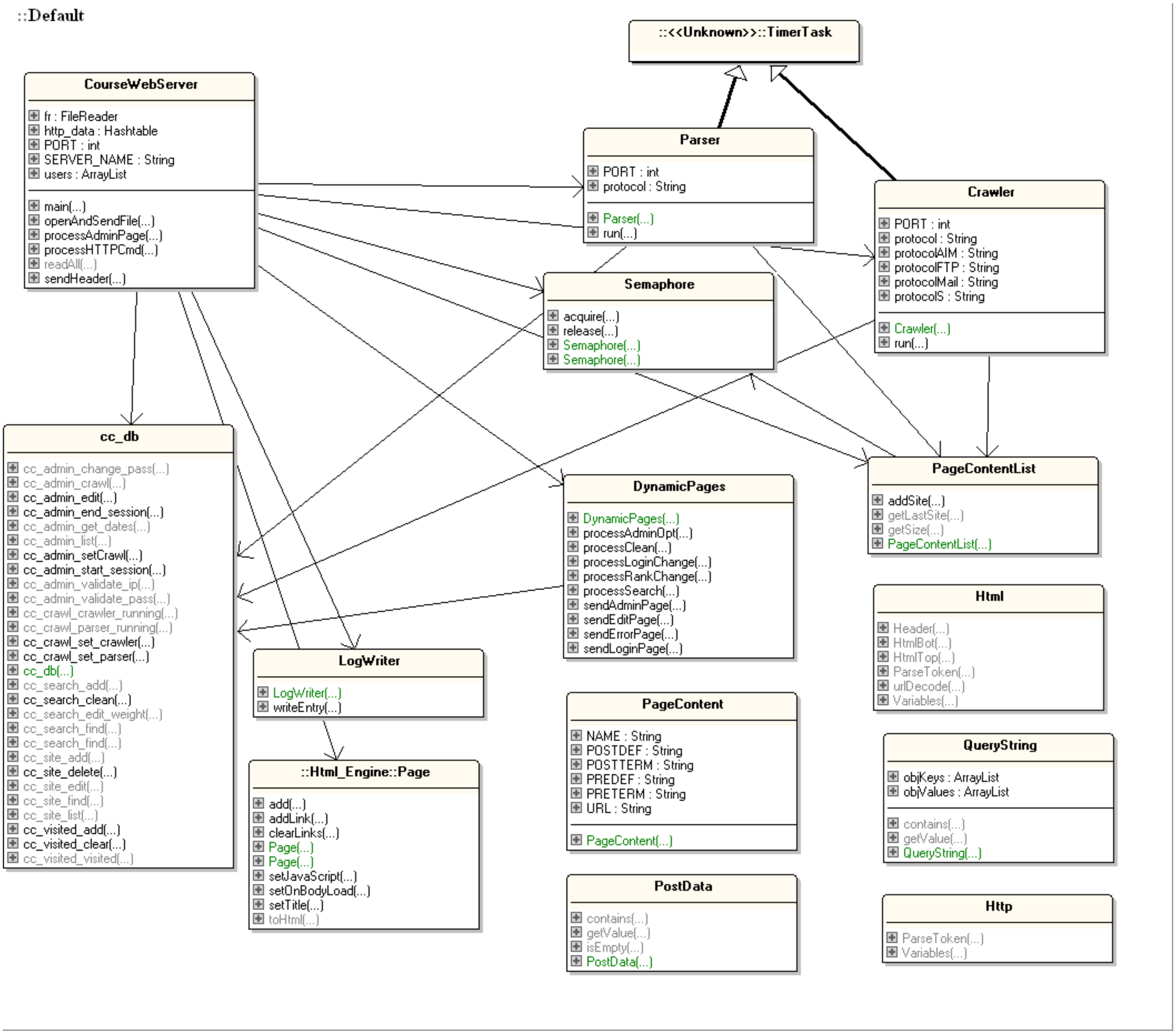
**4 Security** **44**

**5 License** **44**

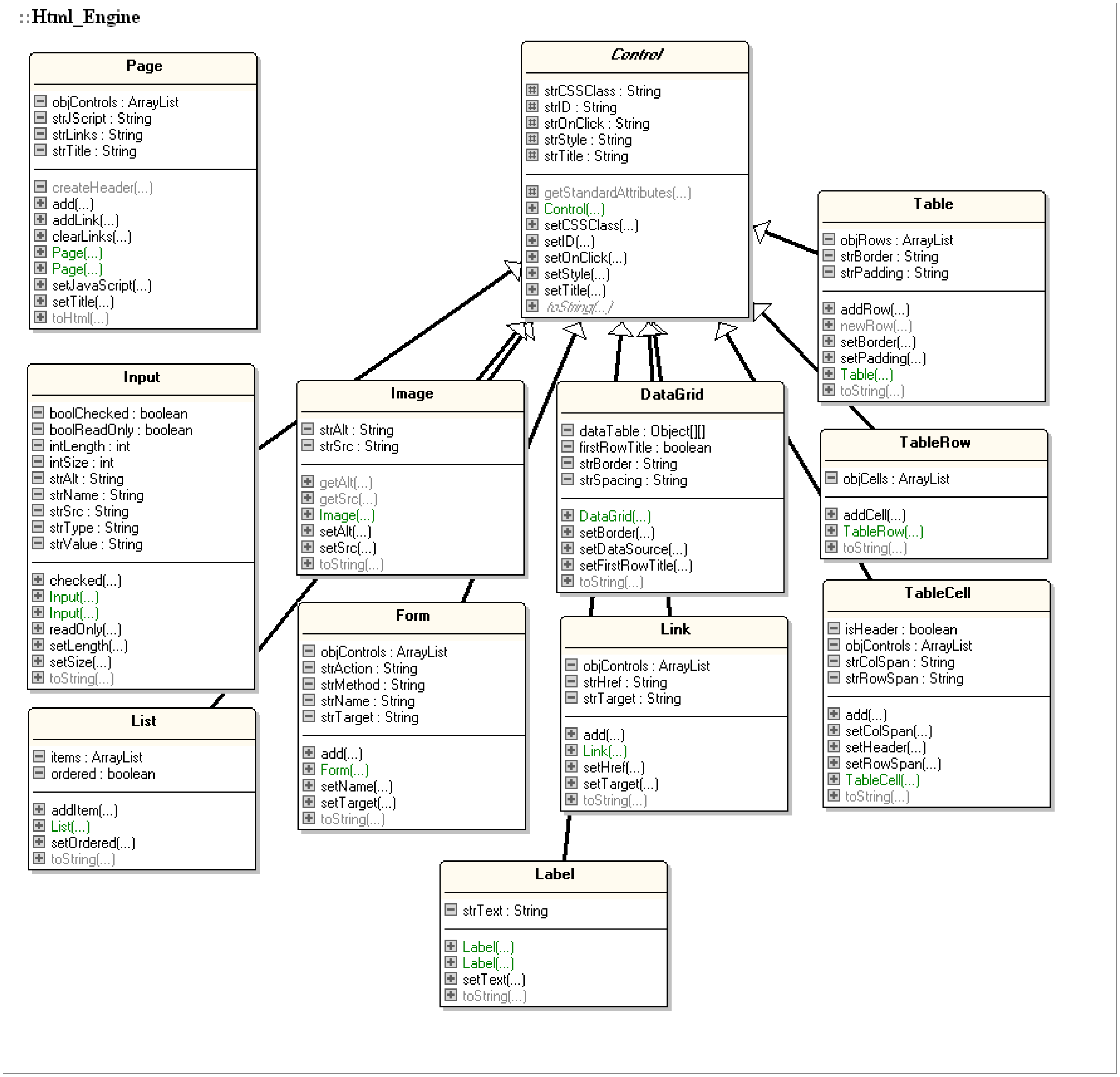
**6 Credits** **44**

# 1 UML Diagrams

## 1.1 Main Program



## 1.2 Html_Engine Package



## 2 Program Structure

The program is divided into three main parts: The Frontend, the Crawler and the Database Interface. The Crawler works without user interaction reading webpages and parsing them for term-definition pairs. The Frontend generates and serves the web content, sending the appropriate queries/updates to the Database Interface and instantiating administrative sessions as needed.

## 3 Class and Package Descriptions and Specifications

### 3.1 Database and Backend

This product utilizes four tables, outlined below, and one Java class (cc_db) to interface between the MySQL database and the Java program that uses it.

#### 3.1.1 Database and Initialization

This product was written using MySQL Version 4.0.17, although it should run on other versions. It is strongly recommended that you allocate a separate database for this product, however it may be possible to use an existing database.

If you have any questions regarding installation, configuration, etc. of MySQL please consult *http://www.mysql.com/* for assistance.

The database configuration file is CourseCrawlerSetup.sql which contains all of the SQL commands used to set up a database for use with the product. Information on how to use this file to configure the database can be found in the accompanying user documentation.

### 3.1.2 Database Table Configuration

This database uses five tables:

- **ccrawler_admin** stores a single row containing all of the configuration information

  Table Elements:

  - admin_password VARCHAR (50) NOT NULL

    Storage of administrative password (Default is ’admin’).
  - admin_expire INT DEFAULT 5

    How long after login does an administrative session terminate?
  - admin_log INT DEFAULT 0

    Log http information? 0 = false, 1 = true
  - admin_last TIMESTAMP

    Timestamp for last administrative login.
  - admin_last_crawl DATE

    Datestamp for last crawl date.
  - admin_last_clean DATE

    Datestamp for last clean date.
  - admin_current INT DEFAULT 0

    Is there a current administrative session? 0 = false, 1 = true.
  - admin_cur_ip VARCHAR (16)

    Current IP address (as varchar or string)
  - admin_freq INT DEFAULT 30

    How often will the crawler run (in days)?

Table Properties:

  - PRIMARY KEY(admin_password)

- **ccrawler_search** contains the term-definition pairs and some other information about them.

  Table Elements:

  - search_word VARCHAR (100)

    The word from the word-definition pairs
  - search_def TEXT NOT NULL

    The definition from the word-definition pairs
  - search_name VARCHAR(100) NOT NULL

    A name from the site_list table that tells which site the pair came from.
  - search_source TEXT NOT NULL

    This field stores the URL of the page that the search-pair came from.
  - search_rank INT UNSIGNED DEFAULT 0

    Rank used for ranking functionality.
  - search_date DATE

    This stores the date that this term was found. This allows for the DB to be cleaned later.

  Table Properties:

  - INDEX (search_name)
  - PRIMARY KEY (search_word, search_name)

  - FOREIGN KEY (search_name) REFERENCES ccrawler_sites(site_name) ON DELETE CASCADE ON UPDATE CASCADE

- **ccrawler_sites** stores the sites to be crawled, their names, pre and post-word/definition text, and other information.

  Table Elements:

  - site_name VARCHAR (100) NOT NULL

    The site name to be displayed in search results.
  - site_url VARCHAR(100) NOT NULL

    The base URL of the site.
  - site_allow_higher TINYINT DEFAULT 0

    Allow the crawler to search higher levels or other subdomains?
  - site_preword TEXT NOT NULL

    Preword parse text.
  - site_postword TEXT

    Postword parse text.
  - site_predef TEXT NOT NULL

    Predefinition parse text.
  - site_postdef TEXT NOT NULL

    Predefinition parse text.

  Table Properties:

  - UNIQUE(site_name)
  - PRIMARY KEY(site_url, site_name)

- **ccrawler_visited** is only used by the crawler and stores URLs that have been visited so as to avoid circular traversals.

  Table Elements:

  - visited_url VARCHAR(200) NOT NULL PRIMARY KEY

- **ccrawler_crawl** stores information about the current state of the crawler and parser.

  Table Elements:

  - crawl_crawler int DEFAULT 0
    0 if crawler not running, 1 otherwise
  - crawl_parser int DEFAULT 0
    0 if parser not running, 1 otherwise

  Table Properties:

  - PRIMARY KEY (crawl_parser, crawl_crawler)

### 3.1.3 cc_db Class

**Signature:** public cc_db()

**Pre Conditions:** None.

**Post Conditions:** Connection to MySQL database is initialized.

**Signature:** public Object[][] cc_admin_get_dates()

**Pre Conditions:** None.

**Post Conditions:** An object containing the dates of interest.

**Signature:** public boolean cc_admin_validate_pass(String inPassword)

**Pre Conditions:** inPassword is a valid string.

**Post Conditions:** Returns true is inPassword matches the password in the DB. Else, returns false.

**Signature:** public boolean cc_admin_validate_ip(String inIP)

**Pre Conditions:** inIP is a valid String representation of an IP.

**Post Conditions:** if there is a valid session w/ the specified IP, return true. Else return false. public boolean cc_admin_validate_ip(String inIP)

**Signature:** public boolean cc_admin_crawl()

**Pre Conditions:** None.

**Post Conditions:** If the last crawl (admin_last_crawl) occurred admin_freq or more days prior to the present date, return true. Else return false.

**Signature:** public void cc_admin_setCrawl()

**Pre Conditions:** None.

**Post Conditions:** last crawl date is set to current.

**Signature:** public void cc_admin_start_session(String inIP)

**Pre Conditions:** inIP is a validString representation of an IP.

**Post Conditions:** Begins a session for the specified IP.

**Signature:** public void cc_admin_end_session()

**Pre Conditions:** None.

**Post Conditions:** sets admin_current to zero, logging admin user out.

**Signature:** public Object[][] cc_admin_list()

**Pre Conditions:** None.

**Post Conditions:** returns the contents (single row) of the ccrawler_admin table.

**Signature:** public boolean cc_admin_change_pass(String inOld, String inNew)

**Pre Conditions:** inOld and inNew are valid passwords.

**Post Conditions:** if inOld is not correct password, or form on inNew is invalid, return false. Else, return true and set password to inNew.

**Signature:** public void cc_admin_edit(int inExp, int inLog, int inFreq)

**Pre Conditions:** inExp, inLog, inFreq conform to the constraints laid out in the readme and are valid Integers.

**Post Conditions:** updates administrative settings with new values.

**Signature:** public Object[][] cc_site_list()

**Pre Conditions:** None.

**Post Conditions:** returns the name and (base) URL for the contents of the sites table.

**Signature:** public void cc_site_delete(String inName)

**Pre Conditions:** inName is a valid site_name

**Post Conditions:** the site entry and all corresponding word entries are removed from the DB.

**Signature:** public boolean cc_site_add(String inName, String inURL, boolean inHigher, String inPreWord, String inPostWord, String inPreDef, String inPostDef)

**Pre Conditions:** All input is valid and inName $\leq$ 50 chars, inURL $\leq$ 100 chars, inPre/Post $<$ 65000 chars.

**Post Conditions:** if restrictions or name uniqueness are not met, return false. Else, insert entry and return true. public boolean cc_site_add(String inName, String inURL, boolean inHigher, String inPreWord, String inPostWord, String inPreDef, String inPostDef)

**Signature:** public boolean cc_site_edit(String findName, String inName, String inURL, boolean inHigher, String inPreWord, String inPostWord, String inPreDef, String inPostDef)

**Pre Conditions:** All input is valid, and conforms to the specified length constraints.

**Post Conditions:** The entry with name = findName is updated with new data.

**Signature:** public Object[][] cc_search_find(String inWord)

**Pre Conditions:** inWord is a valid word/phrase.

**Post Conditions:** returns all results for the given search

**Signature:** public boolean cc_search_edit_weight(String inWord, String inName, int inAdd)

**Pre Conditions:** inWord and inName are $<$ 50 chars, inAdd = 1 or -1.

**Post Conditions:** if there is an entry with inWord and inName, its relative weight is increased or deceased according to inAdd.

**Signature:** public boolean cc_search_add(String inWord, String inDef, String inName, String inURL)
**Pre Conditions:** all parameters are valid strings conforming to length constraints.
**Post Conditions:** if parameters are valid and name references a valid site name, continue, else return false. If there is an old (older than this crawl) entry w/ the same inWord, inName and remove it. If there is not a current entry with this inWord, inName insert it, if so, return false.

**Signature:** public boolean cc_visited_visited(String inURL)
**Pre Conditions:** inURL is a no longer than 200 characters.
**Post Conditions:** if inURL is in the visited list, returns true, else return false.

**Signature:** public void cc_visited_add(String inURL)
**Pre Conditions:** inURL is no more than 100 characters long.
**Post Conditions:** inURL is added to listed of visited URLs.

**Signature:** public void cc_visited_clear()
**Pre Conditions:** None.
**Post Conditions:** All entries in ccrawler_visited are removed.

**Signature:** private void processException(SQLException e)
**Pre Conditions:** e is a valid SQL exception
**Post Conditions:** prints the stacktrace of e.

**Signature:** public boolean cc_crawl_crawler_running()

**Pre Conditions:** None.

**Post Conditions:** Return false if database flag != 0, else true.

**Signature:** public boolean cc_crawl_parser_running()

**Pre Conditions:** None.

**Post Conditions:** Return false if database flag != 0, else true.

**Signature:** public void cc_crawl_set_parser(boolean in)

**Pre Conditions:** in is a valid boolean

**Post Conditions:** DB flag is set appropriately

**Signature:** public void cc_crawl_set_crawler(boolean in)

**Pre Conditions:** in is a valid boolean

**Post Conditions:** DB flag is set appropriately

## 3.2 HTML Engine Package

This package does all of the Java based HTML generation including page attributes (Name, stylesheet, etc.) and page elements including images, tables, links, lists, forms, etc. The classes, named according to what aspect of HTML they perform.

### 3.2.1 Control Class

**Signature:** public Control()

**Pre Conditions:** None.

**Post Conditions:** Class variables are initialized.

**Signature:** public void setID(String inID)

**Pre Conditions:** inID is a non-null string

**Post Conditions:** html element id is set to inID

**Signature:** public void setCSSClass(String inClass)

**Pre Conditions:** inClass is a non-null string

**Post Conditions:** cssclass is set to inClass

**Signature:** public void setTitle(String inTitle)

**Pre Conditions:** inTitle is a non-null string

**Post Conditions:** mouseover title is set to inTitle

**Signature:** public void setStyle(String inStyle)

**Pre Conditions:** inStyle is a non-null string

**Post Conditions:** inline style is set to inStyle

**Signature:** public void setOnClick(String inClick)

**Pre Conditions:** inClick is a non-null string

**Post Conditions:** onclick javascript is set to inClick

**Signature:** public void setName(String inName)

**Pre Conditions:** inName is a non-null string

**Post Conditions:** html element name is set to inName

**Signature:** protected String getStandardAttributes()

**Pre Conditions:** none

**Post Conditions:** protected String getStandardAttributes()

**Signature:** public abstract String toString()

**Pre Conditions:** none

**Post Conditions:** returns the html equivalent for the object

### 3.2.2 DataGrid Class

**Signature:** public DataGrid() **Pre Conditions:** none

**Post Conditions:** the DataGrid is initialized

**Signature:** public void setDataSource(Object[][] arrData)

**Pre Conditions:** arrData is a 2D array of objects with meaningful toString() methods

**Post Conditions:** the dataTable is set to arrData

**Signature:** public void setBorder(String inBorder)

**Pre Conditions:** none

**Post Conditions:** the border attribute is set to inBorder

**Signature:** public void setFirstRowTitle(boolean inValue)

**Pre Conditions:** none

**Post Conditions:** firstRowTitle is either turned on or off

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the DataGrid

### 3.2.3 Form Class

**Signature:** public Form(String inMethod, String inAction)

**Pre Conditions:** inMethod is either POST or GET, and inAction is a valid page to submit to

**Post Conditions:** the form is initialized

**Signature:** public void setTarget(String inTarget)

**Pre Conditions:** none

**Post Conditions:** the target is set to strTarget

**Signature:** public void add(Object inItem)

**Pre Conditions:** inObj is an object with a meaningful toString(), like the Control class or String itself

**Post Conditions:** inObj is added to the list of controls

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the Form

### 3.2.4 Image Class

**Signature:** public Image(String inSrc, String inAlt)

**Pre Conditions:** inSrc is a valid picture web address, inAlt is non-null alternate text

**Post Conditions:** strSrc and strAlt are set to inSrc and inAlt respectively

**Signature:** public void setSrc(String inSrc)

**Pre Conditions:** none

**Post Conditions:** sets the picture web address

**Signature:** public void setAlt(String inAlt)

**Pre Conditions:** none

**Post Conditions:** sets the picture alternate text

**Signature:** public String getSrc()

**Pre Conditions:** none

**Post Conditions:** returns the web address for the image

**Signature:** public String getAlt()

**Pre Conditions:** none

**Post Conditions:** returns the alternate text

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the Image

### 3.2.5 Input Class

**Signature:** public Input(String inType, String inName, String inValue)

**Pre Conditions:** inType is a valid type string, inName is a valid string for the name, inValue is a valid value or empty

**Post Conditions:** the item is initialized

**Signature:** public Input(Image inPic, String inName, String inValue)

**Pre Conditions:** inPic is an Html image tag, inName is a valid string for the name, inValue is a valid value or empty

**Post Conditions:** an image input is initialized

**Signature:** public void setSize(int inSize)

**Pre Conditions:** inSize is a non-zero int

**Post Conditions:** the size of the input is set to inSize

**Signature:** public void setLength(int inLength)

**Pre Conditions:** inLength is a non-zero int

**Post Conditions:** the max length of the input is set to inLength

**Signature:** public void readOnly(boolean inValue)

**Pre Conditions:** none

**Post Conditions:** boolReadOnly is set to inValue

**Signature:** public void setValue(String inValue)

**Pre Conditions:** none

**Post Conditions:** strValue is set to inValue

**Signature:** public void checked(boolean inValue)

**Pre Conditions:** none

**Post Conditions:** boolChecked is set to inValue

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the Input

### 3.2.6 Label Class

**Signature:** public Label(String inText)

**Pre Conditions:** inText is a non null string

**Post Conditions:** Label is initialized with inText

**Signature:** public Label()

**Pre Conditions:** none

**Post Conditions:** an empty label is created

**Signature:** public void setText(String inText)

**Pre Conditions:** inText is a non-null string

**Post Conditions:** strText is set to inText

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the Label

### 3.2.7 Link Class

**Signature:** public Link()

**Pre Conditions:** None.

**Post Conditions:** Class is initialized.

**Signature:** public void setHref(String inHref)

**Pre Conditions:** inHref is a non-null string

**Post Conditions:** strHref is set to inHref

**Signature:** public void setTarget(String inTarget)

**Pre Conditions:** inTarget is a non-null string

**Post Conditions:** strTarget is set to inTarget

**Signature:** public void add(Object inObj)

**Pre Conditions:** inObj is a non-null object

**Post Conditions:** inObj is added to the list

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** the link is made into an html tag and returned

### 3.2.8 List Class

**Signature:** public List()

**Pre Conditions:** none

**Post Conditions:** the list data members are initialized

**Signature:** public void setOrdered(boolean inOrdered)

**Pre Conditions:** none

**Post Conditions:** ordered is set to inOrdered

**Signature:** public void addItem(Object inItem)

**Pre Conditions:** inItem is an object with a meaningful toString(), like the Control class or String itself

**Post Conditions:** inItem is added to the list of controls

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the List

### 3.2.9 Page Class

**Signature:** public Page()

**Pre Conditions:** None.

**Post Conditions:** Page title is set to “untitled”.

**Signature:** public Page(String inTitle)

**Pre Conditions:** None.

**Post Conditions:** Page title is set to inTitle.

**Signature:** public void setTitle(String inTitle)

**Pre Conditions:** inTitle is a non-null string

**Post Conditions:** strTitle is set to inTitle

**Signature:** public void setJavaScript(String inJScript)

**Pre Conditions:** inJScript is a non-null string containing javascript

**Post Conditions:** strJScript is set to inJSCript

**Signature:** public void addLink(String rel, String href)

**Pre Conditions:** rel is a non-null string, and href is a non-null string

**Post Conditions:** a link is formed and added to the strLinks string

**Signature:** public void setOnBodyLoad(String inScript)

**Pre Conditions:** inScript is a non-null string

**Post Conditions:** strOnBodyLoad is set to InScript

**Signature:** public void clearLinks()

**Pre Conditions:** none

**Post Conditions:** strLinks is set to the empty string (””)

**Signature:** private String createHeader()

**Pre Conditions:** none

**Post Conditions:** creates the first parts of the page from ¡html¿...¡body¿

**Signature:** public void add(Object inControl)

**Pre Conditions:** inControl is a non-null object

**Post Conditions:** adds inControl to the array of objects to display on the page

**Signature:** public String toHtml()

**Pre Conditions:** none

**Post Conditions:** creates the html code representing the page and returns it as a string

### 3.2.10 Table Class

**Signature:** public Table()

**Pre Conditions:** none

**Post Conditions:** Table is initialized

**Signature:** public void setBorder(String inBorder)

**Pre Conditions:** inBorder is a valid string

**Post Conditions:** the border setting is set to inBorder

**Signature:** public void setPadding(String inPadding)

**Pre Conditions:** inPadding is a valid string

**Post Conditions:** the table padding is set to inPadding

**Signature:** public void addRow(TableRow inRow)

**Pre Conditions:** inRow is a valid TableRow item

**Post Conditions:** inRow is added to the list of Rows

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the Table

### 3.2.11 TableCell Class

**Signature:** public TableCell()

**Pre Conditions:** none

**Post Conditions:** Initializes the cell's data members

**Signature:** public void setColSpan(String inStr)

**Pre Conditions:** none

**Post Conditions:** the column span is set to inStr

**Signature:** public void setRowSpan(String inStr)

**Pre Conditions:** none

**Post Conditions:** the row span is set to inStr

**Signature:** public void add(Object inObj)

**Pre Conditions:** inObj is an object with a meaningful toString(), like the Control classes or String itself

**Post Conditions:** inObj is added to the list of controls

**Signature:** public void setHeader(boolean inHeader)

**Pre Conditions:** none

**Post Conditions:** isHeader is set to inHeader

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the TableCell

### 3.2.12 TableRow Class

**Signature:** public TableRow()

**Pre Conditions:** none

**Post Conditions:** TableRow is initialized

**Signature:** public void addCell(TableCell inCell)

**Pre Conditions:** inCell is a initialized TableCell

**Post Conditions:** inCell is added to the Row

**Signature:** public String toString()

**Pre Conditions:** none

**Post Conditions:** returns the string equivalent of the TableRow

## 3.3 Frontend Classes

The frontend consists of the webserver (CourseWebServer) and a number of supporting classes. CourseWebServer is the main file for the project, controlling the crawler and managing all other functionality. Apart from using the Html_Engine package, the webserver makes use of a number of classes outlined below.

### 3.3.1 CourseWebserver

**Signature:** private static void updateCrawler()

**Pre Conditions:** none

**Post Conditions:** the crawler scheduling is updated to the current frequency

**Signature:** public static void main(String[] args) throws IOException

**Pre Conditions:** args contains a string which is the port number

**Post Conditions:** the server is started on the port specified

**Signature:** private static Page create400Page()

**Pre Conditions:** none **Post Conditions:** the 400 error page is created

**Signature:** private static Page create404Page()

**Pre Conditions:** none

**Post Conditions:** the 404 error page is created

**Signature:** private static Page createHomePage()

**Pre Conditions:** none

**Post Conditions:** the home page is created

**Signature:** public static String readAll(BufferedReader fileIn)

**Pre Conditions:** fileIn in a valid non-null BufferedReader

**Post Conditions:** Reads and returns all the information from a client (browser). This typically is a request (get or post) from a client.

**Signature:** public static void processHTTPCmd(String cmd, BufferedReader sockIn, DataOutputStream sockOut, String ipAddress)
**Pre Conditions:** input parameters are valid and not null.
**Post Conditions:** Process the request (cmd) by a client. Determine what to do based on the parsing result. The results are directly written to the output stream back to the client.

**Signature:** private static String getVerboseStr(String inCoded)
**Pre Conditions:** inVerbose is a non-null string
**Post Conditions:** returns a valid browser decoded string, special characters are converted from their %## equivalent.

**Signature:** private static String getCodedStr(String inVerbose)
**Pre Conditions:** inVerbose is a non-null string
**Post Conditions:** returns a valid browser encoded string, special characters are converted to their appropriate %## equivalent.

**Signature:** public static void processAdminPage(BufferedReader sockIn, DataOutputStream sockOut, String ipAddress, String path)
**Pre Conditions:** sockIn contains post data, ipAddress is the IP of the client requesting access, sockOut is the output stream and path is the current request path
**Post Conditions:** Determines appropriate action to be taken then displays the admin page

**Signature:** private static void updateCrawler()

**Pre Conditions:** none

**Post Conditions:** the crawler scheduling is updated to the current frequency

**Signature:** static void sendHeader(DataOutputStream out, int returnStatus, long contentLength, String contentType)

**Pre Conditions:** all input is valid and non-null.

**Post Conditions:** Send the returning header to the client, based on the return status. The response must follow the HTTP protocol.

**Signature:** static void openAndSendFile(DataOutputStream out, String fileToSend)

**Pre Conditions:** fileToSend contains the name of a valid file.

**Post Conditions:** Send the file requested to the output stream (client).

### 3.3.2 DynamicPages

**Signature:** DynamicPages(String inServerName, cc_db inSQL)

**Pre Conditions:** inServerName is a string that represents the server name, inSQL is an active SQL Server connection (thru cc_db class)

**Post Conditions:** DynamicPages is initialized

**Signature:** private String getVerboseStr(String inCoded)

**Pre Conditions:** inCoded is a non-null string

**Post Conditions:** returns a valid browser decoded string, special characters are converted from their %## equivalent.

**Signature:** private String getCodedStr(String inVerbose)

**Pre Conditions:** inVerbose is a non-null string

**Post Conditions:** returns a valid browser encoded string, special characters are converted to their appropriate %## equivalent.

**Signature:** private void sendHeader(DataOutputStream out, int returnStatus, long contentLength, String contentType)

**Pre Conditions:** out is an active DataOutputStream, returnstatus is a valid html return code, contentlength is the length of the page to send, and content

**Post Conditions:** sends an appropriate html header

**Signature:** private void cleanDates()

**Pre Conditions:** none

**Post Conditions:** Any voter records older than the current time are removed from the list

**Signature:** private Object[][] pageResults(Object[][] arrResults, String pageNumber)

**Pre Conditions:** arrResults is a non-null array of results, pageNumber is a non-null string

**Post Conditions:** depending on the pageNumber the results list is truncated to a specified page (e.g. page 1 -¿ results 1-10). Also, page number begin and end index are recorded in global variables

**Signature:** public void processSearch(String path, DataOutputStream sockOut)

**Pre Conditions:** path is a web address containing a search query, sockOut is a

DataOutputStream for the web connection.

**Post Conditions:** the search is run, and a results page is sent to the browser

**Signature:** public void sendEditPage(DataOutputStream sockOut, QueryString qs) throws IOException

**Pre Conditions:** qs is a QueryString object, and sockOut is a socket connection to the browser

**Post Conditions:** the edit page entry web page is sent.

**Signature:** public void sendErrorPage(DataOutputStream sockOut, String strOther ) throws IOException

**Pre Conditions:** sockOut is a socket connection to the browser, strOther is the error message(s)

**Post Conditions:** the error page is sent to the browser

**Signature:** public void sendLoginPage(DataOutputStream sockOut) throws IOException

**Pre Conditions:** sockOut is a socket connection to the browser

**Post Conditions:** the login page is sent to the browser

**Signature:** public void processLoginChange(DataOutputStream sockOut, PostData pdData) throws IOException

**Pre Conditions:** sockOut is a socket connection to the browser, pdData is the post data

**Post Conditions:** the password change is processed

**Signature:** public void processAdminOpt(DataOutputStream sockOut, PostData pdData) throws IOException
**Pre Conditions:** sockOut is a socket connection to the browser, pdData is the post data
**Post Conditions:** processes the changing of administrative options

**Signature:** public void processClean(DataOutputStream sockOut, PostData pdData) throws IOException
**Pre Conditions:** sockOut is a socket connection to the browser, pdData is the post data
**Post Conditions:** the database is cleaned for the number of specified days, or for 15 if the input is invalid

**Signature:** private boolean isAlphaNumeric(String inStr)
**Pre Conditions:** inStr is non-null
**Post Conditions:** returns false if any characters are non-alphanumeric, true otherwise private boolean isAlphaNumeric(String inStr)

**Signature:** private boolean isNumeric(String inStr)
**Pre Conditions:** inStr is non-null
**Post Conditions:** returns true if the str contains only numbers, false otherwise private boolean isNumeric(String inStr)

**Signature:** private String makeAlphaNumeric(String inStr) **Pre Conditions:** in-

Str is non-null

**Post Conditions:** strips out all characters that are not letters or numbers private String makeAlphaNumeric(String inStr)

### 3.3.3 Html

**Signature:** public static Hashtable ParseToken(String inBuffer)

**Pre Conditions:** String to be decoded.

**Post Conditions:** Strips all special characters and converts them to a decoded hex equivalent.

**Signature:** public static String urlDecode(String in)

**Pre Conditions:** in is valid and non-null.

**Post Conditions:** in is decoded and returned

**Signature:** public static String Header()

**Pre Conditions:** None.

**Post Conditions:** Generate a standard HTTP HTML header.

**Signature:** public static String HtmlTop(String Title)

**Pre Conditions:** Title contains the title you want to put on the page.

**Post Conditions:** Return a String containing the top portion of an HTML file.

**Signature:** public static String HtmlBot()

**Pre Conditions:** None.

**Post Conditions:** Return A String containing the bottom portion of an HTML file.

**Signature:** public static String Variables(Hashtable form_data)
**Pre Conditions:** form_data is the Hashtable containing the form data which was parsed using the ReadParse method.
**Post Conditions:** Return A String containing an HTML representation of all of the form variables and the associated values.

### 3.3.4 Http

**Signature:** public static Hashtable ParseToken(String inBuffer)
**Pre Conditions:** inBuffer is a buffer that contains the header of the response.
**Post Conditions:** Returns a hash table that contains pairs of name and value.

**Signature:** public static Hashtable ParseToken(String inBuffer)
**Pre Conditions:** inBuffer is a valid string containing form data.
**Post Conditions:** Returns a String containing an HTML representation of all of the form variables and the associated values.

### 3.3.5 LogWriter

**Signature:** public LogWriter()
**Pre Conditions:** none
**Post Conditions:** none, LogWriter is allocated

**Signature:** public void writeEntry(String strIP, String strPath)
**Pre Conditions:** strIP is a non-null string, strPath is a non-null string
**Post Conditions:** an entry is made to the log file for the day, which includes the

IP, path, date and time

**Signature:** private String createFileName(Calendar inCal)
**Pre Conditions:** inCal is a calendar which contains the current day
**Post Conditions:** returns a string which is the name of a log file in the form MM-DD-YY.log

### 3.3.6 PostData

**Signature:** public PostData(BufferedReader sockIn, Hashtable http_data)
**Pre Conditions:** sockIn is a socket connection to the browser where the post data is contained
**Post Conditions:** the post data object is created

**Signature:** public boolean isEmpty()
**Pre Conditions:** none
**Post Conditions:** returns true if there are not entries, false otherwise

//Pre: inKey is a non-null string //Post: returns whether there is an entry with key equal to inKey public boolean contains(String inKey) **Signature:** public boolean contains(String inKey) **Pre Conditions:** **Post Conditions:** returns whether there is an entry with key equal to inKey **Signature:** public String getValue(String inKey) **Pre Conditions:** inKey is a non-null string **Post Conditions:** returns the value associated with inKey

### 3.3.7 QueryString

**Signature:** public QueryString(String inQuery)

**Pre Conditions:** inQuery is a path with a query string, or just the query string

**Post Conditions:** the query string is split into keys and values

**Signature:** private void parseString()

**Pre Conditions:** none

**Post Conditions:** parses the post data and fills objKeys and objValues

**Signature:** public boolean contains(String inKey)

**Pre Conditions:** inKey is a non-null string

**Post Conditions:** returns whether inKey is a key in the query string

**Signature:** public String getValue(String inKey)

**Pre Conditions:** inKey is a non-null string

**Post Conditions:** returns the value that corresponds with inKey

## 3.4 Crawler Classes

The crawler utilizes five classes, Crawler, Parser, PageContent PageContentList and Semaphore. Crawler and Parser extend TimerTask, which allows language-side scheduling from CourseWebServer

# Class Descriptions

**Crawler:** Creates an instance of cc_db which will provide access to the database where the list of sites to be crawled is located. Creates a local variable containing the list of sites that is retrieved from the database. Continues to remove and process sites in the list until it is empty. The process consists of attempting a connection to the URL. If successful it will copy the contents of the page to a local file. Then it will parse through the local file looking for links. If the link is an absolute link with a domain, it will check to see if the domain is the same. If it is, it will add the link to a local list which the crawler will parse through later. Also it will add the link, as a PageContent, to the PageContentList, which Parser will process later. If the link is relative, through a function called translateURL, it will create the proper full URL to add to the lists. After processing the first page, the crawler will begin to process every URL in the local list until it is empty. Through this algorithm, the order of pages parsed will be depth first search. This utilizes the file .crawling_tmp

**Parser:** Creates an instance of cc_db which will provide access to the database where the current list of terms are located. Checks to see if the size of PageContentList is greater than 0 and checks to see if a local boolean value crawlerDone is false. If either of these cases exist it will process the last PageContent in PageContentList, by removing it from the list. It will then attempt to connect to the URL from the current PageContent and if successful it will copy the contents of the page to a local file called ".parsing_tmp". Then it will parse through the current page using the PageContent fields PRETERM, POSTTERM, PREDEF, and POSTDEF. If a term and definition are found it adds the PageContent fields NAME and URL, plus the term and definition to the list of terms in the database. If the name of the current PageContent is "Done" it will set crawlerDone to true, and if the size of PageContentList is ever 0, the Parser will quit.

**PageContent:** Basically a structure class which contains 6 strings: NAME, URL, PRETERM, POSTTERM, PREDEF, and POSTDEF. Multiple instances of this structure is contained in an ArrayList, which is a field of PageContentList. Used to pass information about each page from Crawler to Parser.

**PageContentList:** Contains an ArrayList of PageContent objects. Parser and Crawler can add to and remove from this list, and also check the size of the list. Uses a Semaphore to retain mutual exclusion to the ArrayList of PageContent objects.

**Semaphore:** This is used by PageContentList to assure mutual exclusion of the Crawler and Parser classes in accessing or modifying it.

Please note that after every site in the site list is processed, Crawler will add a PageContent with the PageContent field NAME equal to ”Done”. This is so the Parser knows when the Crawler is done.

### 3.4.1 Crawler Class

**Signature:** public Crawler(PageContentList inList)
**Pre Conditions:** inList is a reference to a PageContentList created in RunCrawler.
**Post Conditions:** db gets instantiated. pList gets set to inList. siteList gets set to the return value of cc_db::cc_site_list. linkNameList, linkURLList, and disallowedURLList are instantiated.

**Signature:** private String getDomain(String linkURL)

**Pre Conditions:** linkURL has a value that includes a domain with a possible path

**Post Conditions:** return the domain only without a '/' at the end

**Signature:** private boolean checkRobotsTXT(String inURL)

**Pre Conditions:** inURL has a value with no protocol

**Post Conditions:** if page is disallowed, add to visited list in database

**Signature:** private boolean connectToSite(String inURL)

**Pre Conditions:** inURL has a value of form http://domain.ext/optional_path.

**Post Conditions:** Return true if connection was made.

**Signature:** private boolean checkLinkExternal(String inLink, String inCurURL)

**Pre Conditions:** inLink has a value.

**Post Conditions:** Return true if link is in outside domain and return false if link is in same domain.

**Signature:** private boolean checkValidPage(String inURL)

**Pre Conditions:** inURL has a value that contains the full path and page.

**Post Conditions:** Return true if it contains a valid page extension.

**Signature:** private void addLink(String inLinkName, String inLinkURL)

**Pre Conditions:** inLinkName and inLinkURL have values.

**Post Conditions:** Add items to corresponding lists.

**Signature:** private boolean checkLinkHigher(String inURL, String curURL)
**Pre Conditions:** inURL has a value without domain. curURL has a value of the current page being parsed.
**Post Conditions:** Returns true if link moves to higher directory or if it is an absolute link that's not the same directory as curURL.

**Signature:** private String translateURL(String inURL)
**Pre Conditions:** inURL has a value that possibly contains "./" or "../".
**Post Conditions:** Return a URL with occurrences of "./" changed to just "/" and occurrences of "../" processed into moving up one directory.

**Signature:** private void parseLinks(String inName, String inURL)
**Pre Conditions:** inName and inURL have values.
**Post Conditions:** Parse for links and add them to linkNameList and linkURLList and also add a PageContent to the PageContentList.

**Signature:** public void run()
**Pre Conditions:** 'this' has a value
**Post Conditions:** run Thread 'this'

### 3.4.2 Parser Class

**Signature:** public Parser(PageContentList inList)
**Pre Conditions:** inList is a reference to a PageContentList created in RunCrawler.
**Post Conditions:** db gets instantiated and pList gets set to inList. public Parser(PageContentList

inList)

**Signature:** private boolean connectToSite(String inURL)
**Pre Conditions:** inURL has a value of the form http://domain.ext/optional_path.
**Post Conditions:** Return true if connection was made. private boolean connectToSite(String inURL)

**Signature:** private int iterate(int inPos, int inSize)
**Pre Conditions:** inPos and inSize has a value where $0 \leq$ inPos <inSize.
**Post Conditions:**Return next position, with wraparound, of buffer. private int iterate(int inPos, int inSize)

**Signature:** private void parseTerms(PageContent inPage)
**Pre Conditions:** inPage is a reference to a PageContent in the PageContentList.
**Post Conditions:**Extract page name, url, preterm, postterm, predef, and postdef from inPage and use to find terms and defs from the page. Will return in the middle of the function if bad html or end of page is found.

**Signature:** public void run()
**Pre Conditions:** None.
**Post Conditions:** Removes a page from the PageContentList, attempts to connect to the url, then calls parseTerms. If a PageContent is found with the name ”Done”, then the Crawler is done, so the Thread sleeps.

**Signature:** private String getMainDomain(String inURL)

**Pre Conditions:** inURL has a value with a protocol (http://)

**Post Conditions:** Return main domain (i.e. domain.ext) without subdomains such as ”www”

### 3.4.3 PageContent Class

**Signature:** public PageContent(String inName, String inURL, String inPreterm, String inPostterm, String inPredef, String inPostdef)

**Pre Conditions:** inName, inURL, inPreterm, inPostterm, inPredef, and inPostdef have values.

**Post Conditions:** Set public fields to corresponding input values.

### 3.4.4 PageContentList Class

**Signature:** public PageContentList(Semaphore inS)

**Pre Conditions:** inS is a reference to a Semaphore created in RunCrawler

**Post Conditions:** Set local Semaphore s to inS, and instantiate list.

**Signature:** public void addSite(PageContent inPage)

**Pre Conditions:** inPage is a reference to a PageContent created in Crawler or Parser.

**Post Conditions:** Add page to local field, list.

**Signature:** public PageContent getLastSite()

**Pre Conditions:** None.

**Post Conditions:** Remove last item from list and return the PageContent, item.

**Signature:**public int getSize()

**Pre Conditions:** None.

**Post Conditions:** Return size of list.

### 3.4.5 Semaphore

**Signature:** public Semaphore()

**Pre Conditions:** None.

**Post Conditions:** The semaphore is initialized with the internal value set to zero.

**Signature:** public Semaphore(int v)

**Pre Conditions:** None.

**Post Conditions:** Semaphore is initialized with internal counter set to v.

**Signature:** public synchronized void acquire()

**Pre Conditions:** None.

**Post Conditions:** If semaphore counter is $\leq$ zero, thread waits until it is $>$ zero. When it is, semaphore counter is decremented.

**Signature:** public synchronized void release()

**Pre Conditions:** None.

**Post Conditions:** Semaphore counter is incremented and other threads are notified.

# 4 Security

This software uses a proprietary web server written in Java. As it is very low-powered the security risks are relatively low. Administrative session data is all passed in plaintext and so might be open to being intercepted. Administrative sessions are initialized based on password alone. Session validation is checked based on the IP address and login time of the current session. If there is no current session (ie. previous administrator logged out) the password must be supplied.

# 5 License

This product is Copyright 2004 by Matt Berntsen, Don Frehulfer and Evan Kaiser.
It is licensed under the Open Software License v. 2.1 as of November 15, 2004.
Please see the included License.txt file for details. (Source: *http://www.opensource.org/licenses/osl-2.1.php*)

# 6 Credits

**This program was written by:**
Matthew Berntsen, Bucknell University Class of 2005 (*http://mattberntsen.net*)
Evan Kaiser, Bucknell University Class of 2005 (*http://www.evankaiser.com*)
Don Frehulfer, Bucknell University Class of 2005 (*http://typical_dfrehulf.blogspot.com*)

**This program was commissioned by:**
Professor Peter Drexel, Plymouth State University (*http://turing.cs.plymouth.edu/ ae1t*)

**Supervision and Advice by:**

Professor Xiannong Meng, Bucknell University (*http://www.eg.bucknell.edu/ xmeng*)