The CEAS Programming Language

COMS W4115
Programming Languages and Translators
Fall 2005

Luis Alonso (lra2103)
Hila Becker (hb2143)
Kate McCarthy (km2302)
Isa Muqattash (imm2104)

Table of Contents

Introduction: The Proposal

Background

In the early days of the Internet, it was easy to surf through interesting websites without having to deal with any excess clutter. Website operators posted their information with simple formatting and a few representative images. Through the years, however, browsing one's favorite websites has become more and more challenging thanks to the proliferation of easy-to-use animation programs, interactive images, and more invasive advertising techniques. For those individuals who still use a dial-up connection, this means wasting more time waiting for a site to download and less time actually browsing. And even those with a high-speed internet connection spend a lot of their time avoiding irrelevant content as well. The CEAS programming language alleviates some of these problems by providing an easy-to-use set of commands that allows the user to customize their browsing experience by specifying what content should not be displayed on their screen.

Goal

The main goal of our project is to design a simple programming language that facilitates web surfing by allowing the programmer to customize their browsing experience by removing any unwanted content from a specified website.

How It Works

The CEAS programming language provides a programmable interface to an existing web proxy named Crunch. Crunch is a highly-customizable framework that can be used to extract data from HTML-formatted web pages. For instance, it can be configured to show a particular page without any images or to show just the information related to the top news stories. The CEAS programming language defines a simple and rich set of commands and operators that allow the user to leverage Crunch to enhance their browsing experience.

An end user starts by writing a script that defines at least one website they are interested in browsing. Then, they makes use of simple operations to manipulate the pages they would like to view. Finally, they run their script through our interpreter which will use Crunch to retrieve and modify the requested web pages. As a final step, the user can choose to write the results to a local HTML file for later browsing or to use the built-in web browser.

More About Crunch

The CEAS language makes use of a pre-existing web proxy called Crunch that extracts content from HTML web pages. Crunch is a pluggable framework that employs an extensible set of techniques for enabling and integrating heuristics concerned with "content extraction" from HTML web pages. Crunch parses the HTML of a given document and produces a Document Object

Model tree to analyze a web page for content extraction. The Document Object Model (http://www.w3c.org/DOM) is a standard for creating and manipulating in-memory representations of HTML (and XML) content. By using a DOM tree, Crunch can not only extract information from large logical units, but also manipulate smaller units such as individual links. Crunch allows the user to select specific tags for extraction as well as predefined extraction filters such as "news" and "shopping", which are preconfigured for specific website genres.

Language Characteristics

CEAS was designed as a simple language in order to ensure that it would be used for its intended purpose. Specific functions and structures were added to the language in order to give the user enough control to accomplish the tasks of navigating the Internet and extracting website content. Some of the important characteristics of the CEAS programming language are described below.

Data Types

CEAS is strongly typed. Variables are bound to a type at declaration. Constants are bound to a type based on their format as described above. CEAS supports the simple data types int, boolean, string, and void, as well as the complex built-in data types Page and list. Users of the language cannot create their own data types, but rather use this language by assigning values and manipulating the built-in structures.

Program Flow Control

In general, program flow proceeds from the first statement in a file to the last. Program flow can be modified with conditional and iterative statements or with function calls.

Functions

CEAS supports both user-defined and built-in functions. CEAS includes the built-in page creation and manipulation functions createPage(), extract(), append(), title(), rank(), and status(), the list function length(), and the output functions print(), println(), show(), and savePage().

Extraction Constants

The following identifiers are reserved constants that are used as arguments to the extract function: IMAGES, ADS, FLASH, SCRIPTS, TXTLINKS, IMGLINKS, XTRNSTYLE, STYLES, FORMS, LINKLISTS, EMPTYTBLS, INPUT, META, BUTTON, and IFRAME.

Key Features of CEAS

CEAS creates a better overall web surfing experience for the programmer. It is easy to learn and use and includes many useful features, such as integrated tabbed browsing and simplified offline browsing. Some of the key features of the CEAS programming language are explained below.

Customization

By using the functions that are built-in to CEAS, users can make the websites they visit look almost any way they want. Remove images, eliminate distracting advertisements, automatically append linked pages to the current page, and so on. CEAS puts the programmers in control of

the visual presentation of the sites they visit by allowing them to specify the content to be displayed.

User-friendly

The CEAS programming language is easy to learn and simple to use. The syntax of the language is straightforward so that complex extraction specifications can be written into succinct statements that are easily understood by even a novice programmer. Once a user is familiar with the basic language constructs, it is easy to customize the appearance of the websites that the user visits by assigning values and manipulating the built-in data structures.

Faster and more efficient web browsing

Users no longer need to waste time waiting for a site to download or searching through all the clutter on a webpage to locate the information that they want. CEAS enables programmers to eliminate extraneous content from various websites so that only the content that is important to the user is displayed on the screen. For instance, a collection of frequently visited sites can be loaded all at once according to the desired specifications by running a single script. As a result, browsing the web is faster, easier, and more efficient.

Appending "next" pages

A user can specify a keyword (i.e. "next") as an attribute to a Page type, which is used to decide whether a link on the Page should be fetched and appended to the bottom of the Page. More specifically, if the keyword is found in the description of a link, the contents of the URL specified by the link are appended to the Page type. This is a useful feature for users who read various news sites in which the articles are spread over multiple pages.

Portability

The CEAS programming language is platform independent. Its only assumptions are that the user has an Internet connection at the time of execution in order to retrieve the requested URLs, and a Java Virtual Machine. This means that CEAS is even deployable to handheld devices such as PDAs which have small screens that do not have room to display all of the extra features of a webpage.

Accessibility

CEAS makes web surfing easier for everyone, including those users who are visually-impaired. Using CEAS to extract extraneous information from a webpage makes it easier for automated screen readers to read aloud the text displayed on a computer screen.

Integrated tabbed browsing

CEAS invokes a built-in web browser that supports tabbed browsing so that you can open multiple pages within the same viewing window. Common browser operations, such as navigating forward or back to a page, are supported by the integrated browser as well. This adds to the convenience of the language, since the user does not need to install an external browser, such as Firefox or Opera, in order to utilize these features.

Simplified offline browsing

CEAS makes it possible for a collection of frequently visited sites to be loaded all at once according to the desired specifications by running a single script. Then, the user can either use the built-in web browser to view the websites immediately or write the results to a local HTML file for offline browsing at their own convenience.

The CEAS programming language makes it fast, easy, and convenient to surf the web by automating the task of content extraction from various web pages. The syntax of CEAS is straight-forward, which makes the language easy to read as well as to write. Simple and complex built-in data types are made available to the user, as well as functions that operate on the complex data types. In addition, CEAS provides loop structures and methods for list iteration. All of these elements, combined with an embedded web browser, offer flexibility and allow the user to customize the content and visual presentation of web documents. Although limited in their number, the built-in structures that CEAS provides give a wide range of functionality for the user, making CEAS the language of choice for all of your web surfing needs.

Language Tutorial

In this section, we present a few simple examples to demonstrate the basics of implementing a CEAS program. For a more extensive explanation of the language syntax, please consult the language reference manual.

The Basics

Before looking at some examples, let's give a brief overview of the individual elements that will be put together to form a complete CEAS program.

Program structure

A program in the CEAS programming language consists of a sequence of statements which represent commands in the language. The semi-colon is required as a statement terminator. In general, program flow proceeds from the first statement in a file to the last. Statements can be grouped within '{' and '}' so that they are treated as a single statement and blocks may be nested within other blocks.

Include statements

The include statement is used to include previously written programs in the current program. Using the include statement executes all of the commands contained in the included file. After the include, any functions declared in the file will be available to the including file. The syntax for including files is :

```
include "favoriteFilters.ceas";
```

Declaring variables

All variables must be declared before they are used for the first time. The variables may or may not be initialized with a value at declaration. Identifiers can be initialized with a value using the assignment operator '='. Examples of variable declarations include:

```
int x = 2;
boolean value = true;
String name = "This is fun";
Page webpage;
```

Variables must be declared before they are used.

List declarations are a special case of the general declaration rules. When declaring and initializing lists it is not necessary to declare the size of the list since the size is implied by the right side of the assignment. Examples of list declarations include:

```
int bigList[10];
Page pages[10];
int numList[] = [0, 1, 2, 3, 4];
string[] words;
```

Creating a Page

A new Page type is created using the createPage() function. In order to create a Page type, the URL of the web page must be specified in one of the following ways:

```
Page createPage("http://www.cnn.com/");
Page createPage(webpage);
Page createPage("http://www.nytimes.com/", "pages/world/index.html");
```

Manipulating a Page

Page manipulation functions are applied to a Page type in order to customize the appearance of a webpage in our tabbed browsing environment. Examples of Page manipulation functions include:

```
extract(webpage, IMAGES, ADS, );
append(webpage, newPage, "next");
title(webpage, "My Favorite Page");
rank(webpage, 2);
status(webpage);
```

Control statements

CEAS also supports several of the traditional iterative statements. Examples of iterative statements include:

```
while (i < length) {
        extract(plist[i], SCRIPTS);
        extract(plist[i], ADS);
        i = i + 1;
}

do {
        rank(pages[j], numPages – j);
        j = j – 1;
} while (j >= 0);

for (i = 0 : lastElement)
        extract(plist[i],FLASH);
```

If statements allow the programmer to choose, at runtime, which commands in a program will be executed. They have the following format:

```
if (length == 0)
        return false;

if (lastElement < 0)
        return false;
else {
        for (i = 0 : lastElement)
                plist[i].extract(FLASH);
}
```

Return statements

When encountered in the body of a function, the return statement will cause program control to return to the calling statement. An optional expression following the return keyword will cause the function to return the result of the expression to the calling statement. Return statements can only be found within functions. Examples of a return statement include:

```
return;
return false;
```

Functions

Functions are uniquely identified by their function signature which includes the name of the function and the list of types the function accepts as parameters. Function definitions may not appear within other functions. Once a function has been defined, it is available to any statement that follows it. Functions defined in included files are also available. Functions are defined with the following syntax:

```
function boolean remFlashAdsScripts(Page[] plist) {
        if (removeFlash(plist))
                if (removeAdsAndScripts(plist))
                        return true;
        return false;
}
```

Output functions

The built-in functions, print() and println(), print the passed parameter to standard output. print() is used to print without a new line and is useful for constructing error messages. println() appends a system-dependent new line to the end of the passed parameter.

```
print("Error detected while applying filters");
println("Error detected while applying filters");
```

The show() function takes a list of Page types as an argument and displays each page as a separate tab in the built-in browser. This function uses the page values for rank and title to determine the position and the title of each tab, respectively.

```
show(pages);
```

The savePage() function is applied to a Page type in order to save its HTML contents in a file for offline browsing. This function takes a string argument indicating the name of the file and returns a boolean indicating success or failure.

```
savePage(webpage, "myPages.html");
```

A Simple Example

Everyday Maria comes home from work and reviews a collection of websites from her favorite political bloggers. One day she realizes that there are more animated advertisements and annoying images than there is text on the page. She wishes that her browser was smart enough to get rid of the annoying images so she could focus on the important commentary. Maria goes in search of such a tool and discovers CEAS can solve her problems.

Even though Maria is not a trained programmer, she quickly discovers that CEAS will easily solve her problems. She writes her first script:

```
Page blog1 = createPage("http://www.myblog.com/latestpost");
extract(blog1, IMAGES);
show(blog1);
```

These five lines of code tell CEAS to build an internal representation of a webpage, remove any images, and then display the newly formatted page using an internal browser.

A More Complicated Example

Maria is fairly pleased with her initial success. However, she misses the ability to quickly load multiple tabs with her favorite websites as she used to do when she browsed with Firefox. She knows that CEAS has more functionality and decides to see if she can improve on her original script.

After doing some research she writes the following script:

```
List pl[3];
pl[0] = createPage("http://www.myblog.com/latestpost");
pl[1] = createPage("http://www.otherblog.com/latest");
pl[2] = createPage("http://www.newblog.com/");
for (int i=0 : 3)
{
        extract(pl[i], IMAGES);
}
show(pl);
```

This piece of code will allow Maria to define a list that holds three pages. She then moves through the list and removes images on each page. Finally, she displays her pages in tabbed form by calling show().

Sample Code and Output

Here is a simple way to extract content from a news article using CEAS. We choose to use the predefined extraction filters for the "news" setting provided by the language. We regard the

"news" setting as the harshest setting that extracts all contents on the page except for the text. This setting is useful for reading news articles, when the user is only interested in the article's textual content.

```
	Page p;
	p =
createPage("http://www.cnn.com/2005/WORLD/meast/09/26/mideast.ap/index.html");
	extract(p, "news");
	show(p);
```

Language Reference Manual

This section is divided into subsections for easier reading.

1. Lexical Conventions

1.1 Comments

CEAS supports common single and multi-line comments. The character combination "//" denotes single-line comments. Multi-line comments begin with "/*" and end with "*/". It is illegal to embed multi-line comments within either style of comment. However, single-line comments may be embedded within a multi-line comment block.

1.2 Identifiers

Identifiers may be of arbitrary length and must consist of a letter or an underscore followed by any combination of letters, digits, and underscores. CEAS is case sensitive, so the identifiers my_var and My_Var are considered to be distinct identifiers.

1.3 Keywords

The following identifiers are reserved as keywords within the CEAS language:

for	while	If	else	do
true	false	Int	boolean	string
Page	break	continue	include	return
function	void			

1.4 Literals

Literals in the language can be integers, strings, booleans, or lists.

1.4.1 Integers

An integer consists of a sequence of one or more consecutive digits.

1.4.2 Strings

Strings are represented as any sequence of ASCII characters found between double-quotes (“). Double-quotes can be embedded within strings by placing two double-quotes next to each other (“”). For example, the string “””” is a one-character string consisting of the double-quote character.

1.4.3 Boolean

The two boolean constants, *true* and *false* are keywords in the language and represent the logical values.

1.4.4 Lists

A list literal is created by enumerating all of the elements in a list. Each list element is separated from the following element by a comma and the whole expression is enclosed in ‘[‘ and ‘]’. All list elements must be literals of the same type or be expressions that evaluate to the same type.

1.5 Other tokens

The following characters and sequences of characters all have meaning:

```
{       }       (       )       [       ]
.       ,       =       <       <=      >
>=      ==      !=      !       +       -
%       /       *       ;       +=      -=
*=      /=      &       |       :
```

2. Types

The language is strongly typed. Variables are bound to a type at declaration. Literals are bound to a type based on their format as described above. The following types are supported:

- *int* : 32-bit integers
- *string* : list of characters
- *boolean* : either true or false
- *Page* : logical representation of a web address
- *void* : used for functions that do not return values
- *lists* : lists can be of type int, string, boolean, or Page

3. Expressions

Expressions are listed below in order of precedence.

3.1 Primary Expressions

Primary expressions are literals, identifiers, list access, function calls, and expressions contained within “(“ and “)”.

3.1.1 Literals

Literals are integers, strings, or booleans as defined above. They evaluate to the expressed value and their type is determined by the interpreter.

3.1.2 List Literals

List literals are lists composed of literals or expressions that evaluate to the same type.

3.1.3 Identifiers

Identifiers evaluate to the value they were bound to last. The type is determined by the type assigned to the identifier at declaration.

3.1.4 List Access

Access to the n^{th} element of list *A* is written as:

A[*<expression>*]

where *<expression>* is an integer expression that evaluates to *n-1*. Lists begin at index 0, so the first element is at A[0] and the second at A[1]. List bounds are checked by the interpreter. The type of this expression is equivalent to the type of the elements contained within the list.

3.1.5 Function Calls

Functions are invoked by naming the function followed by a comma-separated list of parameters contained within "(" and ")". The parameter list is optional but the parentheses are not.

Functions may have return types, in which case, the expression is of the same type as the function.

3.1.6 Parenthesized Expressions

Parentheses are used to ensure proper precedence. The value of a parenthesized expression is equal to the result of the expressions contained within the parenthesis.

3.2 Arithmetic Expressions

CEAS supports multiplication, integer division, modulus, addition, and subtraction. These expressions take primary expressions of type int as operands.

Unary Operators

'+' and '–' may be used as prefix unary operators on integer expressions. A '+' before an expression returns the value of that expression. A '–' returns the negative of the expression.

Binary Operators

Multiplication (*), division (/), and modulus (%) have the highest precedence and associate from left to right.

Addition (+) and subtraction (-) are at the next level of precedence and also associate from left to right.

3.3 Boolean Expressions

Boolean expressions always return boolean values and consist of relational expressions and logical expressions

3.3.1 Relational Expression

All relational expressions evaluate to the boolean values *true* or *false*. All are binary operators and grouped left to right. The operators accept arithmetic expressions and primary expressions that evaluate to numeric values as operands. The operators are:

- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to
- != : Not equal
- == : Equal

3.3.2 Logical Expressions

Logical expressions consist of an operator that accepts boolean expressions as operands and produces a boolean value. They are listed below from highest to lowest precedence:

- !*bool* : Not operator
- *bool* & *bool* : And operator
- *bool* | *bool* : Or operator

4. Statements

A statement represents a command in the language and is always terminated with a semi-colon (;). Statements are either an expression or one or more statements separated by semi-colons. In general, program flow proceeds from the first statement in a file to the last. Program flow can be modified with conditional and iterative statements or with function calls. The different types of statements are described in detail below.

4.1 Statement Blocks

Statements can be grouped within '{' and '}'. Statements contained between these characters are treated as a single statement. Blocks may be legally nested within other blocks. When the program enters a new block, a new variable scope is created. The scope is destroyed when the program exits the block. Variables declared prior to the block are available within the block. Variables declared within the block are lost as soon as the block exits.

4.2 Identifier Declaration

All variables must be declared before first use. Declaration statements are as follows:

<type> <identifier>;

where *identifier* is any legal identifier that has not yet been declared in the current scope and *<type>* is either int, boolean, string, or Page. Multiple variables of the same type may be declared on the same line by separating the variable identifiers with commas. For example:

```
int var2, var3, var4;
char c1, c2, c3;
```

are valid declarations.

4.2.1 List Declarations

List declarations are a special case of the general declaration rules. Lists are declared in the following ways:

<type> <*identifier*>[<*size*>];

Where *<type>* is one of the types, <*identifier*> is a legal identifier, and <*size*> is an optional integer expression. The interpreter will allocate memory for the full list if a size is present.

As with other declarations, multiple lists of the same type and size may be defined on the same line by separating identifiers with commas. The following are all legal declarations:

```
int bigList[10];
string smallStringList[2], anotherSmallList[3];
Page pages[10];
boolean results[];
```

List declarations may be mixed with non-list declarations of the same type:

```
int x, small[4], y; // declares two integers and an integer list
```

Alternatively, a list can be declared with this syntax:

<type>[] *<identifier>*;

Where *<type>* is either string, int, boolean, or Page. The declaration creates a place holder for a list of the declared type.

Lists are only one-dimensional.

4.3 Assignment

Identifiers can be assigned a value using the assignment operator, '='. An identifier must be declared before it can be used in an assignment statement. The identifier appears on the left-hand side and an expression appears on the right-hand side. The type of the expression must match the type assigned to the identifier at declaration.

<identifier> = <expression>;

The semi-colon is required as a statement terminator.

For int, string, and boolean variables, a copy of the expression value is associated with the identifier. Pages, on the other hand, are treated as objects. When assigning Pages, the identifier points to the same object that appeared on the right-hand side of the assignment statement. For example:

```
Page a;
a = createPage("http://www.somesite.org/");
Page b;
b = a; //b and a are now the same Page objects
```

4.3.1 Assignment at Declaration

As a shortcut, variables can be assigned a value at declaration. This takes the form of:

<type> <identifier> = <expression>;

<expression> may be any valid expression that evaluates to a value of the same type as *<identifier>*.

When declaring and initializing lists it is not necessary to declare the size of the list since the size is implied by the right side of the assignment. The following is legal:

```
int numList[] = [0, 1, 2, 3, 4];
```

4.4 If Statements

If statements allow the programmer to choose, at runtime, which commands in a program will be executed. They have the following format:

if (<*boolean_expression*>) <*statement*>

if (<*boolean_expression*>) <*statement*> else <*statement*>

In both versions of the if statement the boolean expression is evaluated to either true or false. If it evaluates to true the first substatement is evaluated. In the second version of the if statement, the second substatement will be evaluated when the boolean expression evaluates to false. An else is always connected to the most recent else-less if in the current block of statements.

The if and else branches can contain multiple statements if they are contained within '{' and '}'.

The statements contained within each branch occur in a new variable scope. The scope is destroyed at the end of the if statement.

4.5 Iterative Statements

The following iterative statements are defined:

while (<*boolean_expression*>) <*statement*>

do <*statement*> while (<*boolean_expression*>);

for (<*identifier*> = <*integer_expression*> : <*integer_expression*>) <*statement*>

The while statement will execute its substatement as long as the boolean expression evaluates to true. Note that if the boolean expression evaluates to false at the first iteration, the substatement will never be executed. The do statement is similar, except that it guarantees that the substatement will be executed at least once. It will continue to execute the statement as long as the boolean expression evaluates to true.

The for statement allows for iteration over a range of numbers. The loop identifier in the for statement is an implicit integer declaration. The identifier must not exist in the current scope and will not exist after the for loop has terminated.

When a for loop is encountered, the loop identifier is set to the initial value. The loop identifier is then checked against the upper bound of the loop. If the loop identifier is less than the upper bound, the body of the loop is processed and the loop identifier is incremented. The process continues until the loop identifier is no longer less than the upper bound.

4.5.1 Break Statement

When the keyword **break** is encountered within a loop, the program continues at the end of the encapsulating iterative statement. A break statement encountered outside of an iterative block is not a legal statement.

4.5.2 Continue Statement

When encountered within an iterative block, the **continue** statement will cause the program to immediately jump to the next iteration. A continue statement encountered outside of an iterative block is not a legal statement.

4.6 Function Definition

Functions are defined with this syntax:

function *<type>* *<func_name>* (*<variable_list>*) {
    <statement>
}

or

function *<type>*[] *<func_name>* (*<variable_list>*) {
    <statement>
}

<func_name> is a legal identifier used to refer to the function. <*variable_list*> is an optional list of variable declarations separated by commas. <*statement*> is zero or more statements that make up the body of the function. <*type*> is the return type of the function. Use void if the function does not return a value.

Function definitions may not appear within other functions. Also, a function may not invoke itself directly.

Functions are uniquely identified by their function signature which includes the name of the function and the list of types the function accepts as parameters.

Once a function has been defined, it is available to any statement that follows it.

The body of a function is its own scope. Statements within the body may not refer to variable identifiers that were declared outside of the function.

4.7 Return Statement

When encountered in the body of a function, the **return** statement will cause program control to return to the calling statement. An optional expression following the **return** keyword will return the value of the expression to the calling function.

A return statement may look like:

```
    return;
or
    return <expression>;
```

Return statements are only allowed within function bodies.

4.8 Include Statement

The **include** statement is used to include previously written functions in the current program. It uses this syntax:

```
    include <file_name>;
```

<file_name> is a string referring to a physical file on the drive. It is assumed to be relative to the current working directory unless an absolute path is provided.

When a file is included, all of its top-level statements will be executed and all of its functions will be declared. After the include statement, only the functions from the included file will be available.

5. Internal Functions

5.1 Output

The built-in functions, print() and println(), print the passed parameter to standard output. print() is used to print without a new line and is useful for constructing error messages. println() appends a system-dependent new line to the end of the passed parameter.

Both print() and println() take exactly one parameter, which can be an expression that evaluates to any suitable type. When a Page object is passed as a parameter, the URL of the page will be printed.

5.2 Page Creation Functions

A new Page type is created using the createPage() function. In order to create a Page type, the URL of the web page must be specified in one of the following ways:

- Page createPage(*<URL>*)
- Page createPage(*<host_URL>, <relative_URL>*)

<URL>, *<host_URL>*, and *<relative_URL>* are string arguments for the complete URL, the host, and the relative path, respectively. All of the above function calls return a Page type. In the special case that the specified URL is not found, the HTML document associated with the page is an empty document.

In addition to the previous two, a Page, and all of its extraction settings, can be duplicated using:

- Page createPage(*<Page>*)

Where *<Page>* is the Page object to be duplicated.

5.3 Page Manipulation Functions

Page Manipulation functions are applied to a Page type in order to customize the appearance of a web-page in our tabbed browsing environment.

Extract – *void extract(Page p, string extraction_constant1, string extraction_constant2, …, string extraction_constantn)*

The function extract() takes one or more extraction constant arguments that correspond to the different sections and tags of the HTML document that need to be removed. This function does not return any value.

The following identifiers are reserved constants that are used as arguments to the extract function.

- **IMAGES** – All images are to be extracted from the HTML document.
- **ADS** - All advertisements are to be extracted from the HTML document.
- **FLASH** - All flash animation is to be extracted from the HTML document.
- **SCRIPTS** - All scripts are to be extracted from the HTML document.
- **TXTLINKS** - All textual links are to be extracted from the HTML document.
- **IMGLINKS** - All images with links are to be extracted from the HTML document.
- **XTRNSTYLE** - All external stylesheets are to be ignored.
- **STYLES** - All style tags are to be extracted from the HTML document.
- **FORMS** - All forms are to be removed from the HTML document.

- **LINKLISTS** - All lists of consecutive links are to be removed from the HTML document.
- **EMPTYTBLS** – All empty tables are to be removed from the HTML document.
- **INPUT** - All input tags are to be removed from the HTML document.
- **META** - All meta tags are to be removed from the HTML document.
- **BUTTON** - All button tags are to be removed from the HTML document.
- **IFRAME** - All iframe tags are to be removed from the HTML document.

Append – *void append(Page p, string keyword)*

The function append() takes a string that specifies a keyword. If the keyword appears as part of a link on the web document (between the <a> and tags in the HTML), the URL that the link points to is fetched and its contents are appended to the end of the Page object.

Title – *string title(Page p, String s)*

The function title() takes a string argument and displays it as the title of the Page when opened in a tab.

Rank – *rank(Page p, int rank)*

The function rank() takes an integer that represents the priority assigned to the Page when opened in a tabbed browser. A larger integer corresponds to a higher priority. When the Page is added to a list and opened in the browser, its location will be determined by its rank relative to the other Page types in the collection. The display order is not defined when two pages have the same rank.

Status – *boolean status(Page p)*

The function status() returns a boolean value that indicates the status of the Page. A false return value indicates that the page cannot be displayed in the way that was specified due to a failure in either the URL fetching or content extraction process.

5.4 List Functions

Length – *int length(type list[])*

The length() function takes a list type argument and returns the number of elements in the given list as an integer.

5.5 Output Functions

Show – *void show(Page pageList[])*

The show() function takes a list of Page types as an argument and displays each page as a tab in a built-in browser. This function uses the page values for rank and title to determine the position and the title of each tab, respectively.

Save Page – *void savePage(Page p, string filename)*

The save() function is applied to a Page type in order to save its HTML contents in a file for off-line browsing. This function takes a string argument indicating the name of the file and returns a boolean indicating success or failure.

6. Namespaces

There are two namespaces exposed to the user. One namespace covers the names of functions. The second namespace covers variables, types, and language constants.

Within the function namespace, names of functions can be reused as long as the type and order of the parameters can uniquely identify the function.

Files that are included in a source file may affect the namespace of the programmer's source file. As an example, assume a programmer has two files, toInclude.ceas and main.ceas where toInclude.ceas is included in main.ceas. The names of any functions declared in toInclude.ceas will become part of the function namespace of main.ceas.

7. Scoping Rules

The language is statically scoped. In general, the user will encounter four different scopes. The first and most important is the file level. Any variables declared within a file are only available within that file.

The second scope covers function names. Function names are available throughout an instantiation of a program after they have been declared. Therefore, functions declared within an included file are available at any later point.

The third scoping level is the function-level. A function body is its own scope. Identifiers declared within the function body are only available within the function body. Identifiers declared in a file-level scope are not available within the function body.

Finally, the user can start a nested scope any time they create a statement block with '{' and '}'. This often occurs in iterative loops or selection statements. As with functions, identifiers declared within curly brackets are not accessible outside of the brackets. However, identifiers declared in the parent scope are available within the brackets.

Project Plan

Team Responsibilities

For the past three months, our group has held weekly meetings with TA Chris Conway to evaluate the progress of our project. In the beginning, our group also scheduled separate weekly meetings to discuss and organize the project, but time progressed we began to work more independently and relied heavily on email correspondence to keep each other informed of

the status of our assigned tasks. The project responsibilities were divided among the team members as follows:

Luis Alonso	Project manager, LRM, Analyzer, Interpreter
Hila Becker	Grammar, Crunch, Browser, Page data type, Unit testing
Kate McCarthy	Grammar, Symbol table, Data types, Documentation
Isa Muqattash	Grammar, Testing

Project Timeline

The deadlines for some of the major components of the projects are listed below:

Week of September 25	Set up CVS and Eclipse
September 27	Submit project proposal
Week of October 2	Learn ANTLR
Week of October 2	Brainstorm about language grammar
Week of October 9	Implement lexer and parser
October 20	Submit LRM
Week of October 30	Implement backend classes
Week of November 6	Begin interpreter implementation
Week of December 4	Begin semantic analyzer implementation
Week of December 11	Finalize AST walkers, interpreter, semantic analyzer
Week of December 18	Final stages of testing
Week of December 18	Finish up documentation
December 20	Project reports due

Project Log

The actual completion dates of the major components as compiled from the CVS logs are listed below:

September 12	First team meeting
September 19	First draft of proposal completed
September 26	Project proposal completed
October 3	Begin working on the grammar
October 5	First meeting with TA Chris Conway
October 5	Begin working on the LRM
October 12	Finalize grammar
October 19	Finalize LRM
October 26	Begin working on content extractor
October 27	Begin working on symbol table
November 6	Finish content extractor

November 7	Begin working on data type classes
November 11	Revise the LRM
November 11	Integrated browser
November 27	Begin working on walkers and interpreter
December 5	Revise content extractor
December 6	Finished data types
December 8	Revise grammar
December 8	Finish symbol table
December 8	Hello World for interpreter
December 09	Begin working on semantic analyzer
December 13	Finish first-run walker
December 13	Begin working on documentation
December 14	Begin working on presentation
December 16	Interpreter Feature Complete
December 17	Revise the AST walker
December 17	Complete semantic analyzer
December 18	Finalize all components
December 19	Complete project documentation
December 19	Organize presentation
December 20	Presentation

Software Development Environment

Operating Systems

Our team members used both Windows and Linux simultaneously for project development.

Java

With the exception of the lexer, parser, and walker, which were written in ANTLR, all of the other components of the project were written in Java. The version of the Jave Runtiime Environment used for this project is Java 1.5.0. We followed the code conventions for the Java programming language as outlined by Sun. The whole document can be viewed at http://java.sun.com/docs/codeconv/, but the main points are outlined below:

- Use the template header to list the file name, version, date, time, and author and a brief description of the file.
- Use tabs for indentation to illustrate the structure of nested control structures.
- The left brace "{" is written on the same line as preceding statement separated by a single space. The right brace "}" is written on a separate line following the last statement in the block and is aligned with the beginning of the statement that began the block.
- The "then" and "else" statements following an "if" must be written on a separate line and should be indented one tab more than the previous line.
- Use comments to explain the function of chunks of code and to point out design decisions that may not be evident from the code itself.
- All class names begin with "CEAS" followed by identifiers whose first letter is uppercase.
- Choose meaningful names for variables and functions that explain what they are used for.

- Variable names are written in all lowercase and method names are written in lowercase with uppercase letters beginning every word except the first.

ANTLR

ANTLR, ANother Tool for Language Recognition, was utilized to write the lexer, parser, and walker. ANTLR is a language tool for constructing compilers from grammatical descriptions. We used ANLTR-generated programs to construct the abstract syntax tree and to create a walker to traverse through it. We followed the following code conventions for the ANTLR programming language:

- Short ANTLR rules are written in one line with the colon ":" indented one tab and the terminating semicolon ";" aligned with the colon on the last line. Actions are written on a separate line.
- Longer ANTLR rules are written with the colon ":" indented one tab and the terminating semicolon ";" aligned with the colon on the last line. Multiple choices are written one per line beginning with a "|" which is aligned with the colon and semicolon. Actions are written on a separate line beneath the last choice.
- The names of lexemes are written in all uppercase letters and non-terminals are written in all lowercase letters.

IDE

Eclipse SDK 3.1.1 was used to facilitate the development process. The ANLTR plug-in for Eclipse that we used is ANTLR 2.7.5.

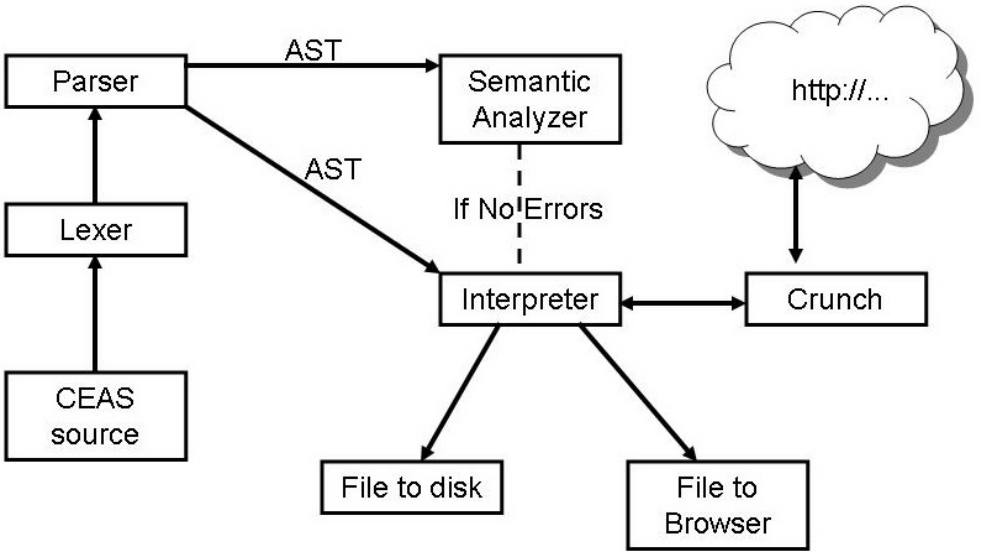
CVS

CVS is a revision control system that was used to organize our project. The CVS Repository was hosted on a CRF Linux machine and was accessed both remotely and from the CLIC lab.

Architectural Design

Major Components

The CEAS interpreter consists of several interrelated components: the lexer, the parser, the AST walker, the semantic analyzer, and the interpreter. Each of these components is explained below in more detail. The following diagram illustrates the interaction between these elements:



The Lexer

The lexer scans the source program and groups the stream of characters into a sequence of tokens. The results from this lexical analysis are then passed on to the parser. The lexer was generated using ANTLR, a language tool for constructing compilers from grammatical descriptions, and can be found in the ANTLR file *grammar.g*. The lexer was implemented by Hila Becker, Kate McCarthy, and Isa Muqattash.

The Parser

The parser uses the results obtained from the lexer to create an abstract syntax tree which represents the grammatical phrases in the source program. The user is notified during this phase if there are any syntax errors present in the code. The parser was also generated using the ANTLR tool and can be found in *grammar.g*. The parser was implemented by Hila Becker, Kate McCarthy, and Isa Muqattash.

The AST Walkers

The CEAS interpreter uses two abstract syntax tree walkers. One is used to verify the semantics of the user's program; the other is used to perform the actions described in the program. Both were generated with the aid of the ANTLR tool. The walkers were implemented by Luis Alonso.

The Semantic Analyzer

The semantic analyzer walks through the abstract syntax tree and checks that all of the expressions and statements are semantically correct. For each statement in the source program, the analyzer first determines if the statement is at a legal point in the file. For example, certain keywords including return and break can only be used at specific places in a program. In the case of variable declarations, the analyzer attempts to add the variable into the symbol table, checking to see if the variable has been previously declared. Function declarations are reviewed

to make sure that the semantics of the function body are correct. Function calls are tested to verify the function signature exists. The analyzer checks assignment statements for the existence of a left value and ensures that the type of the left value matches that of the right value. It also guarantees that all of the operators and operands used in the program are of compatible data types. In the case of function invocations, the analyzer makes sure the parameters are in the proper order and of the proper type. Semantic analysis is performed using an abstract tree walker which was generated with the aid of the ANTLR tool and can be found in ANTLR file *analyzer.g*. The walker, *CEASAnalyzerWalker.java*, invokes appropriate methods from the *CEASAnalyzer* class, which actually performs the semantic checks. CEASAnalyzer maintains a simplified state of the system as it processes the AST. The semantic analyzer can be found in the source file *CEASAnalyzer.java*. It was implemented by Luis Alonso.

The Interpreter

Once the code has been inspected and proven to be syntactically and semantically correct, the interpreter walks through the abstract syntax tree executing the operations specified by the source program as they are encountered. If the semantic analyzer is properly implemented, the user should only receive runtime errors during this phase. These errors stem from improper URLs or inability to connect to websites. The interpreter uses the abstract tree walker found in the ANTLR file *interpreter.g* as well as a collection of classes and objects that support the walker which can be found in the source file *CEASInterpreter.java*. The associated classes include the interpreter class, a symbol table class, a parent class for the data types, and child classes for each of the data types in the language.

The interpreter functions quite similarly to the semantic analyzer except that the interpreter retains more detailed state as it walks through the abstract syntax tree and allows for side effects, which are the results of the user's program. It was implemented by Luis Alonso.

Error Handling

In general, errors fall into two categories, simple and extreme. There are three places where errors may be generated. The lexer/parser phase is the first source of errors. Simple errors are reported to user on the command line. Extreme errors are caught by the main program. The interpreter stops if either type of error is discovered.

The semantic analyzer is the second source of errors. While checking types, variable declarations, function declarations, etc, the semantic analyzer always tries to continue to process the file. If it discovers a recoverable error, such as a type mismatch in an expression, an error message is continued and the analyzer continues. If an extreme error occurs, such as a problem reading an included file or an unknown variable (and therefore and unknown type), the analyzer throws a new exception. The exception is caught by the overall program which handles it as needed. If there were any simple errors during the analysis, the interpreter will not launch.

Finally, the last source of errors is in the interpreter. Since the analyzer has captured all type mismatches, declaration mistakes, and other semantic errors, the interpreter only encounters runtime errors. Runtime errors can come from bad URLs, lack of network connection, or errors reading and writing to files. These errors cause exceptions, which cause the program to clean up resources and terminate.

Main()

The main() method is located in *ceas.CEAS*. It receives as input a **.ceas* source file. It also accepts an optional command-line argument, '-c'. When the '-c' argument is present, CEAS only applies the semantic analyzer to the input file. It reports the number of errors, if any.

Data Types

The class *CEASDataType.java* is extended by a class for each specific data type: *CEASBool.java*, *CEASChar.java*, *CEASFunction.java*, *CEASInt.java*, *CEASList.java*, *CEASPage.java*, and *CEASString.java*. *CEASConstant.java* is a special type of *CEASString* that is used for language-level constants. It is the responsibility of the child classes to implement the methods for that type. The data type classes were implemented by Kate McCarthy and Hila Becker.

In addition to the language-specific data types, the semantic analyzer makes use of *CEASCheck*, a class that represents data types as integers. This simplifies the process of verifying that proper types are provided to functions and operators.

Symbol Table

A hierarchy of symbol tables is implemented by the class *CEASSymbolTable.java*. The symbol tables keep track of the name, type, and scope of all the identifiers used in the source program as well as the name, number and types of the arguments, and return type for functions. The symbol table is implemented as a HashMap data structure for easy lookup. The symbol table was implemented by Kate McCarthy.

Test Plan

Goal

Since the main goal of our project is to design a simple programming language that creates a more pleasant web browsing experience for the user, it is imperative that our compiler is thoroughly tested so that there are no complications experienced by the user at runtime.

Overview

An automated expandable test suite was developed in order to rigorously test the quality of the CEAS compiler. Although it is difficult to achieve, our ultimate goal was to exhaust all possible program cases the compiler may encounter in order to verify that execution of the program proceeds correctly. As the compiler grows in size and complexity, it is also crucial that the test suite can expand to accommodate this as well.

In an attempt to cover as many scenarios as possible, we tested each grammar specification outlined in the CEAS Language Reference Manual. The test cases were broken down into two categories. The first category contains test cases for the lexer and parser and the second category contains tests for the remaining components of the CEAS compiler. Our test plan was designed in this manner since the lexer and parser need to handle instances of the grammar that may not necessarily make sense to the remaining components of the compiler. Testing was primarily done by Isa Muqattash, but all of the group members were involved in unit testing the components of the project that they were assigned to work on.

Lexer and Parser

All test suite material is maintained within the test suite directory. Lexer and parser test cases are stored in a dedicated directory named *lexerAndParser*. Test cases are organized in the following manner: If a test contains code that should successfully parse, then the extension type is **.good*. On the other hand, if the test case should not parse correctly, then the extension type is **.bad*. All errors are piped to standard output (i.e. the screen) with the test name labeling the encountered errors. Running a "good" test case through the lexer and parser should not return any errors whereas running a "bad" test case should output at least one error. It is preferable for the "bad" test cases to contain a single grammatical error to ensure that all such desired alerts are captured by the lexer and/or parser. The test results outlined above are tested by checking the size of the output of the test instance.

Overall Components

In addition to testing the lexer and the parser, we needed to test the other components of the CEAS compiler including the AST walker, interpreter, and the backend portion used for content extraction that is adapted from Crunch. In this case, each test case is composed of two different files as well. The first file is the actual code to be tested, which had an extension of **.ceas*, and the second file contains the expected outcome and has an extension **.exp*. Both the test code and the expected outcome of a particular test case share the same name portion of the filename (i.e. *foo.ceas* and *foo.exp*). The test harness operates by looping through all of the test cases with the extension **.ceas*, running the code through the compiler up to the desired component to be tested, and then comparing the outcome against the **.exp* file. If differences do exist, then the user will be notified via standard output. While all the **.ceas* files reside in a common directory named *category2tests*, the corresponding **.exp* files reside in separate directories labeled with the name of the compiler component to be tested. This distinction is important since different components are expected to produce different outcomes for the code of the same test case.

Test Harness

```
##########################################
#         TEST HARNESSES                 #
##########################################

###FILENAME: lexarAndParser2.sh ###

#harness to test the "bad" files and makes sure that they are not accepted
#by the lexar and/or parser
#!/bin/bash

clear;

echo "------------------------"
echo "TESTING LEXAR AND PARSER"
echo "------------------------"
```

```
for testfile in lexerAndParser/*.bad #run test on each file within each directory
do
        java ceas.testsuit.TestLexarAndParser $testfile 2>>$testfile.out

        if ! test -s $testfile.out; then
                echo "TESTING: $testfile"             #do the testing here
                echo "GRAMMER SHOULDN'T HAVE PARSED!!!"    #error if no parsing errors
detected
                echo "------------------"
        fi

       rm -f *.out  *.ini 0./html

done

echo "TESTING COMPLETED!"


##########################################

###FILENAME: lexarAndParser.sh ###

#harness to test the "good" files to make sure they are accepted
#by the lexar and parser

#!/bin/bash

clear

echo "-----------------------"
echo "TESTING LEXAR AND PARSER"
echo "-----------------------"

for testfile in lexerAndParser/*.good         #run test on each file within each directory
do
        echo "TESTING: $testfile"             #do the testing here
        echo ""
        java ceas.testsuit.TestLexarAndParser $testfile
        echo "------------------"
done

       rm -f *.out  *.ini 0./html
##############################################

##########################################

###FILENAME: overallCompiler2.sh ###

#harness to test the "bad" files and makes sure that they are not accepted
#by the compiler overall. reject may happend at any element within the compiler
#!/bin/bash

clear;
```

```
echo "------------------------"
echo "TESTING OVERALL COMPILER"
echo "------------------------"

for testfile in overallCompiler/*.bad #run test on each file within each directory
do
        java ceas.CEAS $testfile 2>$testfile.out

        if ! test -s $testfile.out; then
                echo "TESTING: $testfile"             #do the testing here
                echo "CODE SHOULD NOT HAVE COMPILED SUCCESSFULLY!!!"  #error if no
parsing errors detected
                echo "------------------"
        fi

        rm $testfile.out
done

echo "TESTING COMPLETED!"


##########################################

###FILENAME: overallCompiler.sh ###

#harness to test the "good" files to make sure they compile overall

#!/bin/bash

clear

echo "------------------------"
echo "TESTING OVERALL COMPILER"
echo "------------------------"

rm -f *.out *.ini *.html
rm -f overallCompiler/*.out  overallCompiler/*.ini overallCompiler/*.html

for testfile in overallCompiler/*.ceas          #run test on each test file
do
        outfile=${testfile//.ceas/.out}
        expfile=${testfile//.ceas/.exp}
        htmlfile=${testfile//.ceas/.html}

        echo "TESTING: $testfile"
        echo ""
        java ceas.CEAS $testfile >>$outfile          #this also covers tests that produce html files

        if test -f $htmlfile; then
                rm -f $outfile
                mv $htmlfile $outfile
        fi
```

```
        diff $expfile $outfile
        echo "------------------"
done

        rm -f *.out  *.ini *.html
        rm -f overallCompiler/*.out  overallCompiler/*.ini overallCompiler/*.html

##############################################
```

Example Tests

```
##############################################
#         LEXAR AND PARSER TESTS             #
##############################################


###FILENAME: assignmentAndFunctions.good ###

//the following should work
test=test-1;
test=-test;
test=test+test;
test=+test;
test+=0;
test+=test;
test-=tset;
test-=0;
//test%=test;
//test%=4;
//test%=0;
test/=5;
test/=test2;
test/=test;
test*=(((test)));
test*=(test+anothetest*3-2/1%0);
test=((((((10*100-0+-5*test/(0-0))))% ((+test)+test*2-2*(-test)))));
test=10*100-0+-5*test/(0-0)%(+test)+test*2-2*(-test);
test[+1]=((((((10*100-0+-5*test/(0-0))))% ((+test)+test*2-2*(-test)))));
test[3-(+2)]=10*100-0+-5*test/(0-0)%(+test)+test*2-2*(-test);
test=((((((10*100-0+-5*test[1]/(0-0))))% ((+test[2])+test*2-2*(-test[3]))));
test=10*100-0+-5*test[1]/(0-0)%(+test[2])+test[2]*2-2*(-test);
//list1[]=list2[];
list[test+1]=list[test];
func();
func(0);
func(0,1);
func(firstpar, secondpar);
func((firstpar),(1+5-test));
func(1, 2, 3,4, 5,6,+7,-8);
func(func2(), func3(par));

###FILENAME: do006.bad ###
```

```
do eatwhite(string,"s",'s'); while(i==func(a,b,"test", 'char'))

##############################################
###FILENAME: do007.bad ###
do test() while(test());

##############################################
###FILENAME: do009.bad ###
do {test(); test2(); i=5}; while(i==5);

##############################################
###FILENAME: do010.bad ###
do test=i+1;; while(a<b & x>test | bool() & bool)

##############################################
###FILENAME: do011.bad ###
do i*=3 while(i!=4)

##############################################
###FILENAME: do012.bad ###
do eatwhite(s,"s",'s') while(i==func(a,b,"test", 'char');)

##############################################
###FILENAME: do013.bad ###
do ; while(bool)
```

```
##############################################

###FILENAME: do014.bad ###

do i+={1}

###FILENAME: dowhile001.ceas ###

//this tests for loops

int i=3;

do
{
        print("testline"); println(i);
        i=i+1;
} while(i<=5);
                                        //prints out:
                                        //testline3
                                        //testline4
                                        //testline5
                                        //testline6

print("value of i = "); println(i);     //shattering only applies to for loop dummies
                                        //value of i = 6;

print("value of i = "); println(i);     //prints out:
                                        //value of i = 6;

i=0;
do
{
        if(i%3==0) i=i+1;
                print("testline"); println(i);
        i=i+1;
}while(i<12);                           //prints out:
                                        //testline1
                                        //testline2
                                        //testline4
                                        //testline5
                                        //testline7
                                        //testline8
                                        //testline10
                                        //testline11


print("value of i = "); println(i);     //prints out:
                                        //value of i = 12;

i=i+2;

print("value of i = "); println(i);     //prints out:
                                        //value of i = 14;
```

##

Lessons Learned

Luis Alonso

- Make sure you really know your grammar and AST inside and out before attempting to write any other code.
- Try to think through as many different scenarios as possible in advance before designing.
- Well-designed projects are easier for teams to work on.

Hila Becker

- Check with your teammates to make sure they are sticking to their schedule.
- Things go smoother when one person takes charge.

Kate McCarthy

- Good communication is essential to a successful project. Do not be afraid to ask the TA or the other members of your group members if you have a question or concern about some aspect of the project.
- Create deadlines for yourself and stick to them. Do not wait until the last minute to complete an assignment because you are bound to make mistakes.
- Working as a team takes a lot of planning and coordination. It is essential that the group has a strong leader who can guide the project along the way and have the final say on any unsettled issues.

Isa Muqattash

- It is very important to reach a stabilized LRM before beginning of implementation. Changes to LRM down the road, although not necessarily avoidable, may result in breaking several components on the project that have already been completed.
- Testing and assurance in a large project environment may become complex.
- Ironically, the expected outcome of a test case may change throughout the lifetime of the project. Old test cases may be frequently modified.
- Always set reasonable goals. Give yourself considerable time between consecutive deadlines.

Appendix A: CEAS Code Listing

ANTLR Files

File: grammar.g

```
//worked on by Hila, Kate, Isa
//Final version by Hila
File: grammar.g
header {
        package ceas;
```

```
}

class CEASLexer extends Lexer;

options {
	k=2;
	charVocabulary= '\3'..'\377';
	testLiterals=false;
	exportVocab=CEASVocab;
}

{
	int errorCount=0;
	public void reportError(String s) {
		super.reportError(s);
		errorCount++;
	}

	public void reportError(RecognitionException e){
		errorCount++;
		super.reportError(e);
	}

}

WS	:	(' '|'\t')+
		{$setType(Token.SKIP);}
	;

NL	:	('\n'
		|('\r' '\n') => '\r' '\n'
		|'\r')
		{$setType(Token.SKIP); newline();}
		;

COMMENT	:	("/*" (
			options {greedy=false;}:
			(NL)
			| ~('\n' | '\r')
		)* "*/"
		|"//"(~('\n' | '\r'))* (NL)
		)	{$setType(Token.SKIP);}
		;

protected DIGIT: '0'..'9';

protected LETTER	:	'a'..'z' | 'A'..'Z' | '_';

NUMBER	: (DIGIT)+;

STRING	:	""! (~('"' | '\n') | ('"'! '"') )* "'!;

ID	options {  testLiterals=true;	}
	: LETTER (LETTER|DIGIT)*
	;
```

```
LBR             : '[';
RBR             : ']';
LPAR    : '(';
RPAR    : ')';
LBRACE          : '{';
RBRACE          : '}';

MINUS : '-';
PLUS    : '+';
MULT    : '*';
DIV             : '/';
MOD             : '%';

COMMA           : ',';
PERIOD          : '.';
SEMI    : ';';
COLON: ':';

ASSIGN          : '=';

EQUAL : "==";
GRT             : '>';
LST             : '<';
GRTEQ: ">=";
LSTEQ : "<=";
NOTEQ: "!=";
PLUSEQ          : "+=";
MINUSEQ         : "-=";
MULTEQ          : "*=";
DIVEQ  : "/=";

AND             : '&';
OR              : '|';
NOT             : '!';


class CEASParser extends Parser;

options
{
        k=2;
        buildAST=true;
        ASTLabelType = "ceas.CommonASTWithLines";
        exportVocab=CEASVocab;
}

tokens {
        PROG;
        STATEMENT;
        FUN_CALL;
        FUN_BODY;
        LIST_ENUM;
        LIST_TYPE;
        DEC_LIST;
        DEC_STMT;
        ASSIGN_DEC_STMT;
        LIST;
```

```
	UPLUS;
	UMINUS;
	VARS;
}

{
	int errorCount=0;
	public void reportError(String s) {
		errorCount++;
		super.reportError(s);

	}

	public void reportError(RecognitionException e){
		errorCount++;
		super.reportError(e);
	}
}


file	: (stmt|fun_def)* EOF!
		 {#file = #([PROG, "PROG"],file);};


expr	: logic_exp (OR^ logic_exp)*;


logic_exp	: logic_term (AND^ logic_term)*;


logic_term	: (NOT^)? relation_exp;


relation_exp	: arithmatic_exp ((EQUAL^|GRT^|LST^|GRTEQ^|LSTEQ^|NOTEQ^) arithmatic_exp)?;


arithmatic_exp	: arithmatic_term ((PLUS^|MINUS^) arithmatic_term)*;


arithmatic_term : arithmatic_atom ((MULT^|DIV^|MOD^) arithmatic_atom)*;


arithmatic_atom: PLUS! rvalue {#arithmatic_atom = #([UPLUS, "UPLUS"], arithmatic_atom); }
				| MINUS! rvalue {#arithmatic_atom = #([UMINUS, "UMINUS"],
arithmatic_atom); }
				| rvalue
				;


rvalue	:	NUMBER
		|	STRING
		|	list_id
		|	"true"
		|	"false"
		|	ID
		|	LPAR! expr RPAR!
		|	fun_call
```

```
                |       applied_fun
                |       list_enum
                ;


//function call is matching this, because it is a right value
list_id         : ID LBR! expr RBR!
                        {#list_id = #([LIST, "LIST"],list_id);}
                        ;


list_enum       : LBR! expr (COMMA! expr)* RBR!
                        {#list_enum = #([LIST_ENUM, "LIST_ENUM"], list_enum);}
                        ;


fun_call : ID LPAR! args RPAR!
                        {#fun_call = #([FUN_CALL, "FUN_CALL"], fun_call);}
                        ;

fun_call_stmt   : fun_call SEMI!
                                ;


args            : (expr (COMMA! expr)*)?
                        ;


applied_fun                     : ID PERIOD fun_call
                                        ; /** function applied to an identifier **/

vars            : LPAR! ( type (ID|dec_list) (COMMA! type (ID|dec_list))*)? RPAR!
                                {#vars = #([VARS, "VARS"],vars);}
                        ;

ltype   : stype LBR RBR
                        {#ltype = #([LIST_TYPE, "LIST_TYPE"],ltype);}
                ;

stype   : "int"
                | "string"
                | "boolean"
                | "Page"
                | "void"
                ;

type    : ( stype | ltype )
                ;


fun_def : "function"^ type ID vars fun_body;


fun_body: LBRACE! (stmt)* RBRACE!
                        {#fun_body = #([FUN_BODY, "FUN_BODY"], fun_body); }
                ;
```

```
stmt    : if_stmt
                | break_stmt
                | continue_stmt
                | include_stmt
                | while_stmt
                | dowhile_stmt
                | for_stmt
                | dec_stmt
                | assign_stmt
                | fun_call_stmt
                | return_stmt
        | LBRACE! (stmt)* RBRACE!
          {#stmt = #([STATEMENT,"STATEMENT"], stmt); }
        ;

if_stmt  : "if"^ LPAR! expr RPAR! stmt
                        (options {greedy=true;}: "else"! stmt)?
                        ;

return_stmt             : "return"^ (expr)? SEMI!
                                ;

break_stmt              : "break"^ SEMI!
                                ;

continue_stmt   : "continue"^ SEMI!
                                ;

include_stmt    : "include"^ STRING SEMI!
                                ;

while_stmt              : "while"^ (LPAR!) expr (RPAR!) stmt
                                ; /** NOTE: supports both single stmt as well as multiple stmts in a block **/

dowhile_stmt    : "do"^ stmt "while" LPAR! expr RPAR! SEMI!
                                ;

for_stmt                : "for"^ LPAR! ID ASSIGN! expr COLON! expr RPAR! stmt
                                ; /** NOTE: supports both single stmt as well as multiple stmts in a block **/

assign_stmt             : (ID | list_id )(ASSIGN^|PLUSEQ^|MINUSEQ^|MULTEQ^|DIVEQ^) expr SEMI!
                                ;

//for declaring a list. The size is optional
dec_list        : (ID LBR! (expr)? RBR!)
                                        { #dec_list = #([DEC_LIST, "DEC_LIST"], dec_list); }
                                ;

dec_stmt                : type ( ID | dec_list) (COMMA! ( ID | dec_list))* (ASSIGN expr)? SEMI!
                                        { #dec_stmt = #([DEC_STMT,"DEC_STMT"], dec_stmt); }
```

File: analyzer.g

```
/*
 * analyzer.g: Actually processes the AST
 *
 * @author Luis Alonso
 *
 * @version $Id: analyzer.g,v 1.1 2005/12/19 16:59:24 lra2103 Exp $
 *
 */

header {
package ceas;
}

{
import java.util.Vector;
import java.util.Enumeration;
import ceas.types.*;
}

class CEASAnalyzerWalker extends TreeParser;
options{
	importVocab=CEASVocab;
}

{
	CEASAnalyzer state;
	//Allows us to decide which interpreter to use at creation time.
	public CEASAnalyzerWalker(CEASAnalyzer state) {
		this();
		this.state = state;
	}
}



expr returns [CEASCheck ret]
{
//variable declarations go here
Vector<CEASCheck> ex_list;
FuncParameter[] args;
String typename = null;

CEASCheck left, right;
left = null;
right = null;
ret = null;
ex_list = null;
}
		//Start with constants
		: num:NUMBER{
				state.setLineNum(num.getLine());
				ret = state.createNumber( num.getText() );
			}
		| str:STRING	{
```

```
                    state.setLineNum(str.getLine());
                    ret = state.createString( str.getText() );
                }
| "true"            { ret = state.getBoolean(); }
| "false"           { ret = state.getBoolean(); }
| name:ID                  {
                    state.setLineNum(name.getLine());
                    ret = state.getVarByName( name.getText() );
                }

| #(PROG (stmt:.
        {
                            ret = expr(#stmt);
        })* )

| #(STATEMENT {state.enterBlock();} (s2:.
        {
                            ret = expr(#s2);
        })* {state.exitBlock();} )

//Boolean statements
| #(OR left=expr right=expr)
        { ret = state.checkBoolean(left, right, "|"); }

| #(AND left=expr right=expr)
        { ret = state.checkBoolean(left, right, "&"); }

| #(NOT left=expr)
        { ret = state.checkBoolean(left, "!"); }

| #(GRT left=expr right=expr)
        { ret = state.checkIntBool(left, right, ">"); }

| #(LST left=expr right=expr)
        { ret = state.checkIntBool(left, right, "<"); }

| #(GRTEQ left=expr right=expr)
        { ret = state.checkIntBool(left, right, ">="); }

| #(LSTEQ left=expr right=expr)
        { ret = state.checkIntBool(left, right, "<="); }

| #(NOTEQ left=expr right=expr)
        { ret = state.checkIntBool(left, right, "!="); }

| #(EQUAL left=expr right=expr)
        { ret = state.checkEqual(left, right, "=="); }

//Mathematics - LRA
| #(PLUS left=expr right=expr)
        { ret = state.checkInt(left, right, "+"); }

| #(MINUS left=expr right=expr)
        { ret = state.checkInt(left, right, "-"); }

| #(MULT left=expr right=expr)
        { ret = state.checkInt(left, right, "*"); }
```

```
		| #(DIV left=expr right=expr)
			{ ret = state.checkInt(left, right, "/"); }

		| #(MOD left=expr right=expr)
			{ ret = state.checkInt(left, right, "%"); }

		| #(ASSIGN left=expr right=expr)
				{ ret = state.checkAssign(left, right, "="); }

		| #(PLUSEQ left=expr right=expr)
			{ ret = state.checkAssign(left, right, "+="); }

		| #(MINUSEQ left=expr right=expr)
			{ ret = state.checkAssign(left, right, "-="); }

		| #(MULTEQ left=expr right=expr)
			{ ret = state.checkAssign(left, right, "*="); }

		| #(DIVEQ left=expr right=expr)
			{ ret = state.checkAssign(left, right, "/="); }

		| #(UMINUS nnum:.)
			{
				state.setLineNum(nnum.getLine());
				ret = state.checkUnary(expr(#nnum), "-");
			 }

		| #(UPLUS pnum:.)
			{
				state.setLineNum(pnum.getLine());
				ret = state.checkUnary(expr(#pnum), "+");
			 }

		| #(LIST id:ID
				{
					state.setLineNum(id.getLine());
					ret = state.getVarByName(id.getText());
				}
			(left=expr { ret = state.getListElement(ret,left);})?)

		| #(LIST_ENUM {Vector<CEASCheck> v = new Vector<CEASCheck>();}
				(left=expr {v.add(left);})+ )
			{ret = state.createList(v);}

		//Variable declaration
		| #(DEC_STMT typename=type_str
			((#(DEC_LIST ln:ID (left=expr)?)
					{
						state.setLineNum(ln.getLine());
						ret = state.createNewListVariable(typename, ln.getText(), left);
					}) |
				(var_name:ID
					{
						state.setLineNum(var_name.getLine());
						ret = state.createNewVariable(typename,
var_name.getText());
```

```
                        }
                )
        )+

        (ASSIGN right=expr {state.checkAssign(ret,right, "=");})?)

//Control Flow Commands
| #("if" left=expr first:. (sec:.)?)
        {
                state.checkBoolean(left, "if");
                state.enterBlock();
                expr(#first);
                state.exitBlock();

                if (sec != null) {
                        state.enterBlock();
                        expr(#sec);
                        state.exitBlock();
                }
        }

| #("include" fname:STRING)
        {
                state.setLineNum(fname.getLine());
                state.doInclude(fname.getText());
        }

| #("function" typename=type_str df_name:ID args=arg_list fb:. )
        {
                state.setLineNum(df_name.getLine());
                state.checkFunction(typename, df_name.getText(), args);
                expr(#fb);
                state.endFunctionDeclaration();
        }

| #(FUN_BODY (fs:.
        {
                        ret = expr(#fs);
        } )* )

| #(FUN_CALL fun_name:ID ex_list=expr_list)
        {
                state.setLineNum(fun_name.getLine());
                ret = state.checkFunctionCall(fun_name.getText(), ex_list);
        }

| #("return" {right = null;} (right=expr)?)
        {       state.doReturn(right); }

| #("break" {state.doBreak();})

| #("continue" {state.doContinue();} )

| #("for" index_var:ID left=expr right=expr for_body:.)
        {
                state.setLineNum(index_var.getLine());
                state.beginForLoop(index_var.getText(), left, right);
```

```
                        expr(#for_body);
                        state.exitForLoop();
                }

        | #("while" left=expr while_body:.)
                {
                        //line num is set by left=expr
                        state.checkBoolean(left, "while");
                        state.beginLoop();
                        expr(#while_body);
                        state.exitLoop();
                }

        | #("do"  do_body:. "while" left=expr)
                {
                        //line num is set by inner statements
                        state.checkBoolean(left, "do-while");
                        state.beginLoop();
                        expr(#do_body);
                        state.exitLoop();
                }


        ;


type_str returns [String t]
{
        t = null;
}
        : ty:type {t = ty.getText();}
        | #(LIST_TYPE ty1:type) {t = (ty1.getText() + "[]");}
        ;


arg_list returns [FuncParameter[] args]
{
        String t = null;
        args = null;
        Vector<FuncParameter> alist = new Vector<FuncParameter>();
}
        //line num is already set
        : #(VARS (t=type_str
                        ((#(DEC_LIST ln:ID)
                                {alist.add(new FuncParameter(t, ln.getText(), true));})
                        |
                        ( n: ID {alist.add(new FuncParameter(t, n.getText()));}) ) )*)
                { args = FuncParameter.vectorToArray(alist); }
        ;

//begin expr_list
expr_list returns [Vector<CEASCheck> rlist]
{
        CEASCheck t;
        rlist= new Vector<CEASCheck>();
}
        //line num set elsewhere
```

```
                    : (t=expr {rlist.add(t);} )*
                    ;

//end expr_list

//begin type
type    : "int"
                    | "string"
                    | "boolean"
                    | "Page"
                    | "void"
                    ;
//end type
```

File: interpreter.g

```
/*
 * interpreter.g: Actually processes the AST
 *
 * @author Luis Alonso
 *
 * @version $Id: interpreter.g,v 1.1 2005/12/19 16:59:24 lra2103 Exp $
 *
 */

header {
        package ceas;
}

{
import java.util.Vector;
import java.util.Enumeration;
import ceas.types.*;
}

class CEASWalker extends TreeParser;
options{
        importVocab=CEASVocab;
}

{
        CEASInterpreter state;
        //Allows us to decide which interpreter to use at creation time.
        public CEASWalker(CEASInterpreter state) {
                this();
                this.state = state;
        }
}



expr returns [CEASDataType ret]
{
//variable declarations go here
Vector<CEASDataType> ex_list;
```

```
FuncParameter[] args;
String typename = null;

CEASDataType left, right;
left = null;
right = null;
ret = null;
ex_list = null;
}
                    //Start with constants
                    : num:NUMBER{ ret = state.createNumber( num.getText() ); }
                    | str:STRING    { ret = state.createString( str.getText() ); }
                    | "true"        { ret = state.getTrue(); }
                    | "false"       { ret = state.getFalse(); }
                    | name:ID               { ret = state.getVarByName( name.getText() ); }

                    | #(PROG (stmt:.
                            {
                                    if (state.flowNormal())
                                            ret = expr(#stmt);
                            })* )

                    | #(STATEMENT {state.enterBlock();} (s2:.
                            {
                                    if (state.flowNormal()) {
                                            ret = expr(#s2);
                                    }
                            })* {state.exitBlock();} )

                    //Boolean statements
                    | #(OR left=expr right_or:.)
                            { ret = ((CEASBool)left).value ? CEASBool.TRUE : expr(#right_or) ; }

                    | #(AND left=expr right_and:.)
                            { ret = ((CEASBool)left).value ? expr(#right_and) : CEASBool.FALSE; }

                    | #(NOT left=expr)
                            { ret = ((CEASBool)left).not(); }

                    | #(GRT left=expr right=expr)
                            { ret = left.grt(right); }

                    | #(LST left=expr right=expr)
                            { ret = left.lst(right); }

                    | #(GRTEQ left=expr right=expr)
                            { ret = left.grteq(right); }

                    | #(LSTEQ left=expr right=expr)
                            { ret = left.lsteq(right); }

                    | #(NOTEQ left=expr right=expr)
                            { ret = left.noteq(right); }

                    | #(EQUAL left=expr right=expr)
                            { ret = left.equal(right); }
```

```
//Mathematics - LRA
| #(PLUS left=expr right=expr)
        { ret = left.plus(right); }

| #(MINUS left=expr right=expr)
        { ret = left.minus(right); }

| #(MULT left=expr right=expr)
        { ret = left.mult(right); }

| #(DIV left=expr right=expr)
        { ret = left.div(right); }

| #(MOD left=expr right=expr)
        { ret = left.mod(right); }

| #(ASSIGN left=expr right=expr)
                { ret = state.assign(left, right); }

| #(PLUSEQ left=expr right=expr)
        { ret = left.pluseq(right); }

| #(MINUSEQ left=expr right=expr)
        { ret = left.minuseq(right); }

| #(MULTEQ left=expr right=expr)
        { ret = left.multeq(right); }

| #(DIVEQ left=expr right=expr)
        { ret = left.diveq(right); }

| #(UMINUS nnum:.)
        { ret = state.negateNumber(expr(#nnum)); }

| #(UPLUS pnum:.)
        { ret = expr(#pnum); }

| #(LIST id:ID {ret = state.getVarByName(id.getText());}
        (left=expr { ret = ret.getListElement(left);})?)

| #(LIST_ENUM {Vector<CEASDataType> v = new Vector<CEASDataType>();}
                (left=expr {v.add(left);})+ )
        {ret = state.createList(v);}

| #(DEC_STMT typename=type_str
        ((#(DEC_LIST ln:ID (left=expr)?)
                        {ret = state.createNewListVariable(typename, ln.getText(), left);}) |
                (var_name:ID
                        {ret = state.createNewVariable(typename, var_name.getText());}))+
        (ASSIGN right=expr {state.assign(ret,right);})?)

//Control Flow Commands
| #("if" left=expr first:. (sec:.)?)
        {
                state.enterBlock();
                if (((CEASBool)left).value)
                        expr(#first);
```

```
                else if (sec != null)
                        expr(#sec);
                state.exitBlock();
        }

| #("include" fname:STRING)
        {state.doInclude(fname.getText());}

| #("function" typename=type_str df_name:ID args=arg_list fb:. )
        {
                state.declareFunction(typename, df_name.getText(), args, fb);
        }

| #(FUN_BODY (fs:.
        {
                if (state.flowNormal())
                        ret = expr(#fs);
        } )* )

| #(FUN_CALL fun_name:ID ex_list=expr_list)
        {       ret = state.doFunctionCall(this, fun_name.getText(), ex_list);}

| #("return" {right = null;} (right=expr)?)
        {       state.doReturn(right); }

| #("break" {state.doBreak();})

| #("continue" {state.doContinue();} )

| #("for" index_var:ID left=expr right=expr for_body:.)
        {
                state.beginForLoop(index_var.getText(), left, right);
                while (state.canForContinue()) {
                        state.enterBlock();
                        ret = expr(#for_body);
                        state.exitBlock();
                        state.incrementForState();
                }
                state.exitForLoop();
        }

| #("while" while_test:. while_body:.)
        {
                state.beginWhileLoop();

                while(state.testLoopBounds(expr(#while_test))) {
                        state.enterBlock();
                        expr(#while_body);
                        state.exitBlock();
                }
                state.exitWhileLoop();
        }

| #("do"  do_body:. "while" do_test:.)
        {
                state.beginWhileLoop();
                do {
```

```
                                        state.enterBlock();
                                        expr(#do_body);
                                        state.exitBlock();
                                }
                                while (state.testLoopBounds(expr(#do_test)));

                                state.exitWhileLoop();
                        }


                ;

type_str returns [String t]
{
        t = null;
}
                : ty:type {t = ty.getText();}
                | #(LIST_TYPE ty1:type) {t = (ty1.getText() + "[]");}
                ;

arg_list returns [FuncParameter[] args]
{
        String t = null;
        args = null;
        Vector<FuncParameter> alist = new Vector<FuncParameter>();
}
                : #(VARS (t=type_str
                                ((#(DEC_LIST ln:ID)
                                        {alist.add(new FuncParameter(t, ln.getText(), true));})
                                |
                                ( n: ID {alist.add(new FuncParameter(t, n.getText()));}) ) )*)
                        { args = FuncParameter.vectorToArray(alist); }
                ;

//begin expr_list
expr_list returns [Vector<CEASDataType> rlist]
{
        CEASDataType t;
        rlist= new Vector<CEASDataType>();
}

                : (t=expr {rlist.add(t);} )*
                ;

//end expr_list

//begin type
type    : "int"
                | "string"
                | "boolean"
                | "Page"
                | "void"
                ;
//end type
```

Package: ceas

File: ceas/CEAS.java

```
package ceas;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.FileNotFoundException;

import antlr.CommonAST;


/**
 * This is the main driver of our system.
 *
 * @author Luis Alonso
 *
 * @version $Id: CEAS.java,v 1.4 2005/12/20 21:45:30 lra2103 Exp $
 */
public class CEAS {

	private static void usage() {
		System.out.println("Usage: CEAS [-c] filename");
		System.out.println("\t-c: check only (optional)");
		System.out.println("\tfilename: The path to the file\n\n");
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		if (!(args.length == 1 || args.length == 2)) {
			usage();
			return;
		}

		String filename;
		boolean check = false;
		//check user arguments
		if (args.length == 1) {
			filename = args[0];
		}
		else {
			filename = args[1];
			if (args[0].equals("-c"))
				check = true;
			else {
				usage();
				return;
			}
		}

		boolean error = false;
		CEASLexer lexer = null;
		CEASParser parser = null;

		//try to parse
		try {
			InputStream input = (InputStream) new FileInputStream(filename);
```

```
            lexer = new CEASLexer(input);
            parser = new CEASParser(lexer);
            parser.setASTNodeClass("ceas.CommonASTWithLines");
            parser.file();
            int numErr = parser.errorCount + lexer.errorCount;
            if (numErr > 0) {
                        if (numErr == 1) {
                                    System.err.println("\n1 error while parsing " + filename + "\n");
                        }
                        else {
                                    System.err.println("\n" + numErr +
                                                " errors parsing " + filename + "\n");
                        }
                        error = true;
            }
}
catch (FileNotFoundException fe) {
            System.err.println("Error: " + filename + " could not be found.");
            error = true;
}
catch (Exception e) {
            System.err.println("Unexpected error while parsing " + filename);
            error = true;
}

if (error) return;

//if there were no parser errors, then we can do semantic analysis
try {
            CommonAST tree = (CommonAST)parser.getAST();
            CEASAnalyzerWalker walker;
            CEASAnalyzer ca = new CEASAnalyzer(filename);
            walker = new CEASAnalyzerWalker(ca);
            try {
                        walker.expr(tree);
            }
            catch (CEASException ce) {
                        System.err.println("\nUnrecoverable errors. Terminating file check.\n");
                        error = true;
            }

            //check for errors in semantic analysis
            if (ca.errors()) {
                        System.err.println();
                        if (ca.getErrorCount() == 1)
                                    System.err.print("1 error and ");
                        else
                                    System.err.print(ca.getErrorCount() +
                                                " errors and ");

                        if (ca.getWarningCount() == 1)
                                    System.err.print(" 1 warning ");
                        else
                                    System.err.print(ca.getWarningCount() +
                                                " warnings ");

                        System.err.println("while checking " + filename + "\n");
```

```
                        error = true;
                    }
                    else if (check) {
                        if (ca.warnings()) {
                            if (ca.getWarningCount() == 1)
                                System.out.println("1 warning.");
                            else
                                System.out.println(ca.getWarningCount() + " warnings.");
                        }
                        System.out.println("File was scanned successfully.");
                    }
                    else {
                        CEASInterpreter ci = new CEASInterpreter(filename);
                        CEASWalker walk2 = new CEASWalker(ci);
                        walk2.expr(tree);

                    }
                }
                catch (CEASException ce) {
                    System.err.println("Unable to process file " + filename);
                }
                catch (Exception e) {
                    System.out.println("Unkown Error");
                    e.printStackTrace();
                }
        }

}
```

File: ceas/CEASAnalyzer.java

```
package ceas;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.FileNotFoundException;
import java.util.Vector;
import java.util.Enumeration;

import ceas.types.CEASCheck;
import ceas.types.FunctionType;

import antlr.CommonAST;



/**
 * @author Luis Alonso
 * @version $Id: CEASAnalyzer.java,v 1.4 2005/12/20 21:45:30 lra2103 Exp $
 */

public class CEASAnalyzer implements CEASWalkIfc {

        /**
```

```
 * Variable to back up the symbol table when processing an include
 */
private CSymbolTable symBackup;

/**
 * Actual symbol table.
 */
private CSymbolTable sym;

/**
 * Only one function symbol table for any run of the system.
 */
private static CEASSymbolTable funcTable;

/**
 * Variables for tracking the number of errors and warnings
 */
private static int numErrors, numWarn;

/**
 * Variables for maintaing state as walker proceeds
 */
private CEASCheck retVal = null;
private CEASCheck retType = null;
private String functionName = null;
private static int inloop;
private int lineNum = 0;

/**
 * List containing the names of all included files
 */
private static Vector<String> includeList = null;

/**
 * The file being processed, for error messages
 */
private String curFileName = null;

/**
 * Default constructor for the CEASAnalyzer. Usually not used.
 *
 */
private CEASAnalyzer () {
        sym = new CSymbolTable();
        inloop = 0;
        if (funcTable == null) {
                funcTable = makeFunctionTable();
                numErrors = 0;
                numWarn = 0;
                includeList = new Vector<String>();
        }

}

/**
 * Constructor for CEASAnalyzer. Sets up all initial values.
 *
```

```
     * @param filename Name of file being checked, for error messages
     */
    public CEASAnalyzer(String filename) {
            this();
            includeList.add(filename);
            curFileName = filename;
    }

    /**
     * Creates the default function symbol table for the checker
     * @return A new, initialized, function table
     */
    private static CEASSymbolTable makeFunctionTable() {
            CEASSymbolTable cst = new CEASSymbolTable(null,null);
            cst.setValue(FUNC_APPEND_NAME, new CEASCheck(CEASCheck.VOID));
            cst.setValue(FUNC_CREATE_PAGE_1, new CEASCheck(CEASCheck.PAGE));
            cst.setValue(FUNC_CREATE_PAGE_2, new CEASCheck(CEASCheck.PAGE));
            cst.setValue(FUNC_CREATE_PAGE_3, new CEASCheck(CEASCheck.PAGE));
            cst.setValue(FUNC_RANK_NAME, new CEASCheck(CEASCheck.VOID));
            cst.setValue(FUNC_SAVE_PAGE_NAME, new CEASCheck(CEASCheck.BOOL));
            cst.setValue(FUNC_SHOW_1, new CEASCheck(CEASCheck.VOID));
            cst.setValue(FUNC_SHOW_2, new CEASCheck(CEASCheck.VOID));
            cst.setValue(FUNC_STATUS_NAME, new CEASCheck(CEASCheck.BOOL));
            cst.setValue(FUNC_TITLE_NAME, new CEASCheck(CEASCheck.VOID));
            return cst;
    }

    public int getErrorCount() {
            return numErrors;
    }

    public int getWarningCount() {
            return numWarn;
    }

    /**
     * Returns true when there are no errors
     * @return true when there are no errors
     */
    public boolean success() {
            return numErrors == 0;
    }

    /**
     * Returns true when there are warnings
     * @return true when there are warnings
     */
    public boolean warnings() {
            return numWarn != 0;
    }

    /**
     * Returns true when there are errors
     * @return true when there are errors
     */
    public boolean errors() {
            return numErrors != 0;
```

```
}

/**
 * Used by the AST walker to set the current line
 * number.
 *
 * @param line Current line number of the file
 */
public void setLineNum(int line) {
	lineNum = line;
}

/**
 * Method to report errors to System.err for the user.
 * @param message The message to give the user
 */
private void reportError(String message) {
	numErrors++;
	if (numErrors == 1) {
		System.err.println("\n\nErrors dected:");
	}
	String head;
	if (curFileName == null) {
		head = "[unknown," + lineNum + "] ";
	}
	else {
		head = "[" + curFileName + "," + lineNum + "] ";
	}
	System.err.println(head + "Error: " + message);
}

/**
 * Method for reporting warnings to the user.
 * @param message
 */
private static void reportWarning(String message) {
	numWarn++;
	System.err.println("Warning: " + message);
}

/**
 * Called by the walker. Given a string, verifies it is
 * acutally a number. If it is, a class representing an
 * integer is returned, otherwise, an error is reported.
 * On error, the walker is allowed to continue.
 *
 * @param text The string to be tested
 * @return An integer variable
 */
public CEASCheck createNumber(String text) {
	try {
		Integer.parseInt(text);
	}
	catch (NumberFormatException nfe) {
		reportError("'" + text + "' is not an integer value.");
	}
	CEASCheck c =new CEASCheck(CEASCheck.INT);
```

```
		c.initialized = true;
		return c;
}

/**
 * Given a string, makes sure it is valid, and returns
 * a string representation for the semantic analyzer.
 * If input is null, reports an error.
 *
 * @param text String to be tested
 * @return A string version of CEASCheck
 */
public CEASCheck createString(String text) {
		if (text == null) {
			reportError("Expecting string got null");
		}
		CEASCheck c = new CEASCheck(CEASCheck.STRING);
		c.initialized = true;
		return c;
}

/**
 * Given a vector, checks the contents to make sure
 * they are all of the same type. If the first element
 * is a list, then an exception is thrown. Otherwise
 * a CEASCheck version of list is returned.
 *
 * @param vals A vector of CEASCheck to be turned into a list
 * @return A CEASCheck list
 */
public CEASCheck createList(Vector<CEASCheck> vals) {
		CEASCheck t =  vals.elementAt(0);
		if (t.isList()) {
			reportError("nested lists are not allowed");
			throw new CEASException("fatal error");
		}

		CEASCheck v;
		Enumeration<CEASCheck> e = vals.elements();
		while (e.hasMoreElements()) {
			v = e.nextElement();
			if (!t.equalTypes(v)) {
				reportError("lists must be of same type. got " + v.typename() +
							" expected " + t.typename());
				break;
			}
		}
		CEASCheck c = CEASCheck.createListType(t);
		c.initialized = true;
		return c;
}

/**
 * Checks to see if the given type and name can be used to create
 * a new variable. Throws an exception if user tries to create
 * a variable of type void. Reports an error if variable is
 * already declared in this scope
```

```
 *
 * @param type String version of type name
 * @param name Name of variable
 * @return A CEASCheck of the input type
 */
public CEASCheck createNewVariable(String type, String name) {
	CEASCheck v = (CEASCheck) sym.getValue(name);
	if (v != null) {
		reportError("Variable " + name + " has already been declared.");
		return v;
	}
	else {
		if (CEASCheck.VOID_TYPE.equals(type)) {
			reportError("Type 'void' is not allowed for variables.");
			throw new CEASException("Illegal Variable type");
		}
		v = CEASCheck.createType(type, name);
		sym.setValue(name, v);
		return v;
	}
}

/**
 * Special function for generating list variables. Similar
 * to createList. In addition, this checks the optional size
 * argument to verify it is an integer.
 *
 * @param type The type of list
 * @param name The variable name
 * @param size The optional size
 * @return A CEASCheck representation of a list
 */
public CEASCheck createNewListVariable(String type, String name,
		CEASCheck size) {

	CEASCheck v = (CEASCheck) sym.getValue(name);
	if (v != null) {
		reportError("Variable " + name + " has already been declared.");
		return v;
	}
	else if (type.endsWith("[]")) {
		reportError("Can not make " + name + " a list of lists");
		throw new CEASException("Illegal type");
	}
	else {
		if (size == null) {
			v = CEASCheck.createListType(type, name);
			sym.setValue(name, v);
			return v;
		}
		else {
			if (!size.isInt()) {
				reportError("List size must be an integer");
				throw new CEASException("List length");
			}
			v = CEASCheck.createListType(type, name);
			v.initialized = true;
```

```
                        sym.setValue(name, v);
                        return v;
                }
        }
}

/**
 * Creates a new interpreter to parse the include file. If there
 * are errors in parsing, an exception is thrown. Each file can
 * only be processed once.
 *
 * @param filename The include file to be processed
 */
public void doInclude(String filename) {
        if (includeList.contains(filename))
                return;

        try {
                InputStream input = (InputStream) new FileInputStream(filename);
                CEASLexer lexer = new CEASLexer(input);
                CEASParser parser = new CEASParser(lexer);
                parser.setASTNodeClass("ceas.CommonASTWithLines");
                parser.file();

                if (parser.errorCount > 0) {
                        reportError("Errors parsing " + filename);
                        throw new CEASException("failed include");
                }
                else {
                        CommonAST tree = (CommonAST)parser.getAST();
                        includeList.add(filename);
                        CEASAnalyzer ci = new CEASAnalyzer();

                        CEASAnalyzerWalker walker = new CEASAnalyzerWalker(ci);
                        walker.expr(tree);
                }
        }
        catch (FileNotFoundException fe) {
                reportError("Unable to find include file " + filename);
                throw new CEASException("Missing file");
        }
        catch (Exception ce) {
                reportError("Unable to include file " + filename);
                throw new CEASException("In " + filename + ":\n" +
                                ce.getMessage());
        }

}

/**
 * Checks a function declaration's signature.
 * @param type Return type of function
 * @param fname Name of function
 * @param args List of arguments to the function
 */
public void checkFunction(String type, String fname,
                FuncParameter[] args) {
```

```
		if (functionName != null) {
			reportError("Can not nest function definitions");
			throw new CEASException("Nested Functions");
		}


		String sysname = FunctionType.generateSysName(fname, args);
		if (funcTable.getValue(sysname) != null) {
			reportError("Function " + sysname + " was already defined");
		}
		symBackup = sym;

		sym = new CSymbolTable();
		CEASCheck c;
		FuncParameter fp;
		for (int i=0; i < args.length; i++) {
			fp = args[i];
			if (fp.isList()) {
				if (fp.getType().endsWith("[]")) {
					reportError("In function " + fname + " can not make " +
							fp.getName() + " a list of lists");
					throw new CEASException ("Illegal List");
				}
				c = CEASCheck.createListType(fp.getType(), fp.getName());
			}
			else {
				c = CEASCheck.createType(fp.getType(), fp.getName());
			}
			c.initialized = true;
			sym.setValue(fp.getName(), c);
		}
		retType = CEASCheck.createType(type, null);
		functionName = sysname;
		retVal = null;

}

/**
 * Called to clean up the function declaration process.
 * Verifies that returns were found if expected and that
 * they were the correct type. Then inserts the function
 * into the symbol table.
 */
public void endFunctionDeclaration() {
		if (retVal == null) {
			if (!retType.isVoid()) {
				reportError("Reached end of " + functionName + "without reaching return");
			}
		}
		else if (!retType.equalTypes(retVal)) {
			reportError("Function " + functionName + " returned type " +
					retVal.typename() + " expected " + retType.typename());
		}
		funcTable.setValue(functionName, retType);
		retType = null;
		functionName = null;
```

```
		retVal = null;
		sym = symBackup;
}

/**
 * Helper function. Converts function arguments to a string
 * to simplify the process of searching for a function match.
 * @param name
 * @param args
 * @return
 */
private String funcArgsToString(String name, Vector<CEASCheck> args) {
		StringBuffer buf = new StringBuffer(name);
		buf.append("(");
		Enumeration<CEASCheck> e = args.elements();
		CEASCheck t;
		while (e.hasMoreElements()) {
			t = e.nextElement();
			if (!t.initialized) {
				reportError(initError(t.name));
			}
			buf.append(t.typename());
			if (e.hasMoreElements())
				buf.append(",");
		}
		buf.append(")");

		return buf.toString();

}

/**
 * Checks the input parameters to the internal extract method
 * @param args Function parameters
 * @return Returns a void type
 */
private CEASCheck checkExtract(Vector<CEASCheck> args) {
		if (args.size() < 2) {
			reportError("Extract expects two arguments");
		}
		else {
			CEASCheck a1, a2;
			Enumeration<CEASCheck> e = args.elements();
			a1 = e.nextElement();
			if (!a1.initialized) {
				reportError(initError(a1.name));
			}
			if (!a1.isPage()) {
				reportError(typeMismatch("extract", "Page", a1));
			}
			while (e.hasMoreElements()) {
				a2 = e.nextElement();
				if (!(a2.isString() || a2.isConstant())) {
					reportError(typeMismatch("extract", "String Constants", a2));
					break;
				}
			}
```

```
		}
		return new CEASCheck(CEASCheck.VOID);
}

/**
 * Checks input values to print functions
 *
 * @param args Input parameters
 * @return type void
 */
private CEASCheck checkPrint(Vector<CEASCheck> args) {
		if (args.size() != 1) {
			reportError("function print expects one argument got " + args.size());
		}
		else {
			CEASCheck a1 = args.elementAt(0);
			if (!a1.initialized) {
				reportError(initError(a1.name));
			}
		}
		return new CEASCheck(CEASCheck.VOID);
}

/**
 * Checks input to length function
 * @param args the arguments to length
 * @return type int
 */
private CEASCheck checkLength(Vector<CEASCheck> args) {
		if (args.size() != 1) {
			reportError("Function length expects one argument got " + args.size());
		}
		else {
			CEASCheck a1 = args.elementAt(0);
			if (!a1.initialized) {
				reportError(initError(a1.name));
			}
			if (!a1.isList()) {
				reportError(typeMismatch("length", "list", a1));
			}
		}
		return new CEASCheck(CEASCheck.INT);
}

/**
 * Verify the arguments to function calls.
 * @param name Name of function being called
 * @param args Arguments provided
 * @return return type of function or void type
 */
public CEASCheck checkFunctionCall(String name, Vector<CEASCheck> args) {

		if (functionName != null) {
			//inside of a function.
			if (functionName.equals(name)) {
				reportError("Recursive call to " + name);
			}
```

```
		}
		CEASCheck res;
		if (FUNC_EXTRACT.equals(name)) {
			res = checkExtract(args);
		}
		else if (FUNC_PRINT.equals(name)) {
			res =  checkPrint(args);
		}
		else if (FUNC_PRINTLN.equals(name)) {
			res = checkPrint(args);
		}
		else if (FUNC_LENGTH.equals(name)) {
			res = checkLength(args);
		}
		else {
			String sysname = funcArgsToString(name, args);
			CEASCheck func = (CEASCheck) funcTable.getValue(sysname);
			if (func == null) {
				reportError("Function " + sysname + " is not defined.");
				throw new CEASException("Error");
			}
			res =  func; //function will be of the correct type
		}
		res.initialized = true;
		return res;
	}

	/**
	 * Returns a boolean type
	 * @return boolean type
	 */
	public CEASCheck getBoolean() {
		CEASCheck c = new CEASCheck(CEASCheck.BOOL);
		c.initialized = true;
		return c;
	}

	/**
	 * Verifies the arguments to boolean operations
	 * @param left left side of operation
	 * @param right right side of operation
	 * @param op string representation of operation
	 * @return boolean type
	 */
	public CEASCheck checkBoolean(CEASCheck left, CEASCheck right, String op) {
		if (!left.isBool()) {
			reportError("Left hand side of " + op + " must be boolean");
		}
		if (!right.isBool()) {
			reportError("Right hand side of " + op + " must be boolean");
		}
		if (!left.initialized) {
			reportError(initError(left.name));
		}
		if (!right.initialized) {
			reportError(initError(right.name));
		}
```

```
		return left;
}

/**
 * Checks to verify that left and right side of equals operations
 * are integers and have been initialized
 * @param left
 * @param right
 * @param op
 * @return boolean type
 */
public CEASCheck checkEqual(CEASCheck left, CEASCheck right, String op) {
		if (left == null || right == null) {
			reportError("Part of expression '" + op + "' is null ");
			throw new CEASException("Unable to continue");
		}
		if (!left.equalTypes(right)) {
			reportError("Equality can only be tested between similar types.");
		}
		if (!left.initialized) {
			reportError(initError(left.name));
		}
		if (!right.initialized) {
			reportError(initError(right.name));
		}
		CEASCheck c = new CEASCheck(CEASCheck.BOOL);
		c.initialized = true;
		return c;

}

/**
 * Checks left and right of operations that take integers
 * and return booleans
 * @param left
 * @param right
 * @param op
 * @return
 */
public CEASCheck checkIntBool(CEASCheck left, CEASCheck right, String op) {
		if (!left.isInt()) {
			reportError("Left hand side of " + op + " must be int");
		}
		if (!right.isInt()) {
			reportError("Right hand side of " + op + " must be int");
		}
		if (!left.initialized) {
			reportError(initError(left.name));
		}
		if (!right.initialized) {
			reportError(initError(right.name));
		}
		CEASCheck c = new CEASCheck(CEASCheck.BOOL);
		c.initialized = true;
		return c;
}
```

```
/**
 * Checks operations that involve one boolean value
 * @param left
 * @param op
 * @return
 */
public CEASCheck checkBoolean(CEASCheck left, String op) {
	if (!left.isBool()) {
		reportError("Operand for " + op + " must be boolean");
	}
	if (!left.initialized) {
		reportError(initError(left.name));
	}
	return left;
}

/**
 * Check operations that have integers on either side
 * @param left
 * @param right
 * @param op
 * @return
 */
public CEASCheck checkInt(CEASCheck left, CEASCheck right, String op) {
	if (!left.isInt()) {
		reportError("Left hand side of " + op + " must be int");
	}
	if (!right.isInt()) {
		reportError("Right hand side of " + op + " must be int");
	}
	if (!left.initialized) {
		reportError(initError(left.name));
	}
	if (!right.initialized) {
		reportError(initError(right.name));
	}
	return left;

}

/**
 * Check unary integer operations
 * @param right
 * @param op
 * @return
 */
public CEASCheck checkUnary(CEASCheck right, String op) {
	if (right == null) {
		reportError("Unary operator " + op + " needs an int value");
	}
	else if (!right.isInt()) {
		reportError(typeMismatch(op, "int", right));
	}
	return right;
}

/**
```

```
 * Check assign. Verify that both sides are equivalent types and
 * that the right side has been initialized
 * @param left
 * @param right
 * @param op
 * @return
 */
public CEASCheck checkAssign(CEASCheck left, CEASCheck right, String op) {

	if (left == null) {
		reportError("Left side of assign does not exist");
		throw new CEASException("Missing value");
	}
	if (right == null) {
		reportError("Right side of assign does not exist");
		throw new CEASException("Missing value");
	}
	if (left.isConstant()) {
		reportError("Can not reassign values to constant " + left.name);
		return right; //keep trying to analyze
	}
	else {
		if (left.checkType() != right.checkType()) {
			reportError(typeMismatch(op, left, right));
		}
		if (!right.initialized) {
			reportError(initError(right.name));
		}
		else {
			left.initialized = true;
		}
		return left;
	}

}

/**
 * Tries to retrieve the stored value for a variable, reports
 * error if it does not exist and throws an exception.
 * @param name
 * @return
 */
public CEASCheck getVarByName(String name) {
	CEASCheck c = (CEASCheck) sym.getValue(name);
	if (c == null) {
		reportError("Variable " + name + " has not been declared.");
		throw new CEASException("unable to continue");
	}
	return c;
}

/**
 * Checks the ability to retrieve a list element. Note, this
 * does not verify list bounds.
 * @param list
 * @param expr
 * @return
```

```
 */
public CEASCheck getListElement(CEASCheck list, CEASCheck expr) {
	if (expr == null) {
		reportError("Invalid expression passed as list index");
	}
	else if (!expr.isInt()) {
		reportError(typeMismatch("list index", "int", expr));
	}
	if (list == null) {
		reportError("List variable does not exist");
		throw new CEASException("Unable to continue");
	}
	else if (!list.isList()) {
		reportError(typeMismatch("list access", "list", list));
		throw new CEASException("Bad list variable");
	}
	CEASCheck c = new CEASCheck(list.checkListType());
	c.initialized = true;
	return c;
}

/**
 * Checks legality of return statement.
 * @param ret
 * @return
 */
public CEASCheck doReturn(CEASCheck ret) {
	if (retType == null) {
		reportError("return encountered outside of function body");
	}
	else {
		if (ret == null) {
			reportError("Return value does not exist");
			throw new CEASException("Invalid tree");
		}
		if (!retType.equalTypes(ret)) {
			reportError("function " + functionName + " returns " +
					retType.typename() + " got " + ret.typename());
		}
	}
	retVal = ret;
	if (!ret.initialized) {
		reportError(initError(ret.name));
	}
	return ret;
}

/**
 * Sets up for loop variables, including new scope.
 * @param varname
 * @param init
 * @param fin
 */
public void beginForLoop(String varname, CEASCheck init, CEASCheck fin) {
	CEASCheck c = (CEASCheck) sym.getValue(varname);
	if (c != null) {
		reportError("Index variable " + varname + " has been previously declared");
```

```
		}
		if (!init.isInt()) {
			reportError(typeMismatch("for init", "int", init));
		}
		if (!fin.isInt()) {
			reportError(typeMismatch("for init", "int", fin));
		}
		c = new CEASCheck(CEASCheck.INT);
		c.name = varname;
		c.initialized = true;
		beginLoop();
		sym.setValue(varname, c);
		sym.inloop = true;
}

/**
 * Clean up any initialziation done for for loop.
 *
 */
public void exitForLoop() {
		exitLoop();
}

/**
 * Set up environment for do and while-do loops.
 *
 */
public void beginLoop() {
		enterBlock();
		inloop++;
}

/**
 * Clean up after loops end
 *
 */
public void exitLoop() {
		inloop--;
		exitBlock();
}

/**
 * Verify break is in the right place
 *
 */
public void doBreak() {
		if (inloop == 0) {
			reportError("break statement encountered outside of loop");
		}
}

/**
 * Verify continue is in right place
 *
 */
public void doContinue() {
		if (inloop == 0) {
```

```
                reportError("continue statement encountered outside of loop");
        }
}

/**
 * Enter a block of code. Creates new nested symbol table.
 *
 */
public void enterBlock() {
        sym = new CSymbolTable(sym.mainTable, sym);
}

/**
 * Exit a block of code by destroying nested symbol table
 *
 */
public void exitBlock() {
        sym = (CSymbolTable) sym.parent();
}

/**
 * Creates an error string to use when a variable has not been
 * initialized
 * @param varName
 * @return
 */
private static String initError(String varName) {
        return ("Variable '" + varName + "' has not been initialized.");
}

/**
 * Creates an error string for type mismatches
 * @param where
 * @param expect
 * @param got
 * @return
 */
private static String typeMismatch(String where, String expect, CEASCheck got) {
        StringBuffer buf = new StringBuffer("");
        buf.append(where);
        buf.append("' expects ");
        buf.append(expect);
        buf.append(" got ");
        if (got.name != null) {
                buf.append(got.name + "(" + got.typename() + ")");
        }
        else {
                buf.append(got.typename());
        }

        return buf.toString();
}

/**
 * Creates error message for type mismatches
 * @param op
 * @param left
```

```
	 * @param right
	 * @return
	 */
	private static String typeMismatch(String op, CEASCheck left, CEASCheck right) {

		StringBuffer buf = new StringBuffer("");
		buf.append(op);
		buf.append("' expects args of the same type. Got\n\t");
		if (left.name != null) {
			buf.append(left.name);
			buf.append("(" + left.typename() + ")\n\t");
		}
		else {
			buf.append(left.typename() + "\n\t");
		}
		if (right.name != null) {
			buf.append(right.name);
			buf.append("(" + right.typename() + ")\n\t");
		}
		else {
			buf.append(right.typename() + "\n");
		}

		return buf.toString();
	}


	/**
	 * Internal version of Symbol Table. Exists to make constants easier to check.
	 * @author Luis Alonso
	 *
	 */
	class CSymbolTable extends CEASSymbolTable {

		public static final long serialVersionUID = 12324L;

		public CSymbolTable() {
			super(null,null);
			//load all the language constants into top-level symbol table
			for (int i=0; i < extract.length; i++) {
				this.setValue(extract[i], new CEASCheck(CEASCheck.CONSTANT));
			}
		}

		public CSymbolTable(CEASSymbolTable p, CEASSymbolTable q) {
			super(p,q);
		}
	}

}
```

File: ceas/CEASException.java

```
package ceas;
/**
```

```
 * CEASException.java
 *
 * Author:  Kate McCarthy
 * Created: November 7, 2005
 *
 * Exception class for the CEAS language which handles language exceptions
 * with a specified message.
 *
 * @author Kare McCarthy
 * @version $Id: CEASException.java,v 1.1 2005/12/19 16:59:24 lra2103 Exp $
 */

public class CEASException extends RuntimeException {

        public static final long serialVersionUID = 10101010L;

/**
 * Constructor with an error message associated with the exception
 *
 */
  public CEASException( String msg ) {
        super("Error: " + msg);
  }
}
```

File: ceas/CEASInterpreter.java

```
package ceas;
import java.util.Vector;
import java.io.FileInputStream;
import java.io.InputStream;

import java.net.MalformedURLException;

import ceas.types.CEASBool;
import ceas.types.CEASDataType;
import ceas.types.CEASInt;
import ceas.types.CEASList;
import ceas.types.CEASString;
import ceas.types.FunctionType;
import ceas.types.PageDataType;

import antlr.collections.AST;
import antlr.CommonAST;
import antlr.RecognitionException;

/**
 * The main interpreter file. Methods are called from a tree walker,
 * CEASWalker. State is maintained within this class. This should
 * only be used after the analyzer has completed its work.
 *
 * @author Luis Alonso
 * @version $Id: CEASInterpreter.java,v 1.4 2005/12/20 14:23:11 lra2103 Exp $
 */
public class CEASInterpreter implements CEASWalkIfc {

        CEASSymbolTable sym;
```

```
static CEASSymbolTable funcTable = null;

private boolean inFunction = false;

private CEASDataType returnVal = null;

//A variable that tracks the state of the system.
//Used for flow control
private int flow_state;

private static Vector<String> includeList = null;

/**
 * Basic constructor. Creates and initializes the basic state
 * tracking mechanisms.
 *
 */
public CEASInterpreter() {
	sym = new CEASSymbolTable();
	flow_state = FLOW_NORMAL;

	if (funcTable == null)
		funcTable = new CEASSymbolTable();
	if (includeList == null)
		includeList = new Vector<String>();
}

/*
 * More commonly used constructor. Given a file name, tracks which
 * file is currently being used.
 */
public CEASInterpreter(String fileName) {
	this();
	includeList.add(fileName);
}

/**
 * Creates a string object with the value passed to it.
 *
 * @param val The value to be stored in the string
 * @return An object representing the string
 */
public CEASDataType createString(String val) {
	return new CEASString(null, val);
}

/**
 * Creates an object representing a whole number.
 *
 * @param val The integer value to be stored in the variable
 * @return An object representing the number
 */
public CEASDataType createNumber(String val) {
	return createNumber(val, false);
}

/**
```

```
     * Creates a number representing a whole number. Can also be used
     * to negate whatever is passed in via the string value.
     *
     * @param val The string representationof the number
     * @param negative If the value should be negated
     * @return A CEASDataType with the new value
     */
    public CEASDataType createNumber(String val, boolean negative) {
            int mult = negative ? -1 : 1;
            try {
                    return new CEASInt(mult * (Integer.parseInt(val)));
            }
            catch (NumberFormatException nfe) {
                    throw new CEASException("'" + val + "' is not a numeric value");
            }

    }

    /**
     * Returns a new integer value equal to the negation of the
     * previous integer value.
     *
     * @param i Value to be negated
     * @return A new value equal to the negative of i
     */
    public CEASDataType negateNumber(CEASDataType i) {
            return new CEASInt(-((CEASInt)i).value);
    }

    /**
     * Returns an object representing the boolean value of true.
     *
     * @return An object equivalent to 'true'
     */
    public CEASDataType getTrue() {
            return CEASBool.TRUE;
    }

    /**
     * Returns an object representing the boolean value of false.
     *
     * @return An object equivalent to 'false'
     */
    public CEASDataType getFalse() {
            return CEASBool.FALSE;
    }

    /**
     * Returns an object bound to the passed name in the current scope.
     * If there is no variable bound to this name, then the method
     * will signal an error by returning null.
     *
     * @param name The name of the variable
     * @return An object bound to the name in the current scope.
     */
    public CEASDataType getVarByName(String name) {
            CEASDataType var = (CEASDataType) sym.getValue(name);
```

```
		if (var == null) {
			error ("Variable '" + name + "' has not been declared or " +
					"does not exist in this scope.");
		}
		return var;
}

/**
 * Assigns the value in the object on the right to the object on the
 * left.
 *
 * @param left The variable to be changed
 * @param right The value to be stored in left
 * @return the new version of the left
 */
public CEASDataType assign(CEASDataType left, CEASDataType right) {
		if (left == null) {
			error("Left side of assign does not exist");
			return null;
		}
		else if (right == null){
			error("Right side of assign does not exist");
			return null;
		}

		//Since we're treating Pages as objects, have to handle
		//the assignment differently
		if (CEASDataType.PAGE_TYPE.equals(left.typename())) {
			sym.resetValue(left.name, right);
			return sym.getValue(left.name);
		}
		else {
			left.assign(right);
			return left;
		}


}

/**
 * Creates a new list object without a name. The size of the list
 * object must be equal to the number of elements in the vector.
 * This should also check to ensure that all of the vector types
 * are the same.
 *
 * @param v A vector, generated by the walker, of all the values in the list
 * @return  A new list, all of whose contents must be of the same type
 */
public CEASDataType createList(Vector<CEASDataType> v) {
		if (!(v.size() > 0))
			error("There are no elements in the list");

		Object[] elements = v.toArray();
		CEASDataType[] vals = new CEASDataType[elements.length];

		String typeName = ((CEASDataType)elements[0]).typename();
		if (typeName == null) {
```

```
                error("Received unknown type in list generation");
        }

        for (int i=0; i < elements.length; i++) {
                if (typeName.equals(((CEASDataType)elements[i]).typename())) {
                        vals[i] = ((CEASDataType)elements[i]).copy();
                }
                else {
                        error("All list elements must be of the same type");
                }
        }
        return new CEASList(typeName, vals);
}

/**
 * Creates a new list variable. If size is null, then the the size limits of
 * list have not yet been determined.
 *
 * @param type The string representation of the type
 * @param name The name of the variable
 * @param size An integer representation of the size
 * @return the newly created object
 */
public CEASDataType createNewListVariable(String type, String name,
                        CEASDataType size) {

        CEASList list = null;

        if (size == null) {
                list = new CEASList(type, name);
        }
        else if (size instanceof CEASInt) {
                int length = ((CEASInt)size).value;
                list = new CEASList(type, name, length);
        }
        else {
                size.error("list size requires an int");
        }

        sym.setValue(name, list);
        return list;
}

/**
 * Creates a new variable in the current scope.
 *
 * @param type The type of the variable.
 * @param name The name of the variable.
 * @return A new object that must be initialized.
 */
public CEASDataType createNewVariable(String type, String name) {
        return createNewVariable (type, name, null);
}

/**
 * Creates a new variable and assigns it a value. The variable is created in
 * the current scope.
```

```
 *
 * @param type The type of the variable
 * @param name The name of the variable
 * @param val The value to be assigned to the variable
 * @return A new DataType object, bound to the name with the given value.
 */
public CEASDataType createNewVariable(String type, String name,
                        CEASDataType val) {

        CEASDataType newvar = CEASDataType.createType(type, name);

        if (newvar == null) {
                System.err.println("Error: Could not create needed memory");
                System.exit(1);
        }
        if (val != null) {
                newvar.assign(val);
        }
        sym.setValue(name, newvar);
        return newvar;
}


/**
 * Processes the input file.
 *
 * @param filename The name of the file to be included
 */
public void doInclude(String filename) {
        if (includeList.contains(filename))
                return;

        try {
                InputStream input = (InputStream) new FileInputStream(filename);
                CEASLexer lexer = new CEASLexer(input);
                CEASParser parser = new CEASParser(lexer);
                parser.setASTNodeClass("ceas.CommonASTWithLines");
                parser.file();

                CommonAST tree = (CommonAST)parser.getAST();
                includeList.add(filename);
                CEASInterpreter ci = new CEASInterpreter();

                CEASWalker walker = new CEASWalker(ci);
                walker.expr(tree);

                //don't need to overwrite because we passed a reference to ours
                //this.funcTable = ci.funcTable; //overwrite our function table
        }
        catch (Exception ce) {

                throw new CEASException("In " + filename + ":\n" +
                                ce.getMessage());
        }

}
```

```
public void declareFunction(String type, String name, FuncParameter[] args, AST body) {
	FunctionType func = null;

	func = new FunctionType(type, name, args, body);

	funcTable.setValue(func.name, func);

}

private CEASDataType[] copyFunctionArgs(Vector<CEASDataType> argList) {
	CEASDataType[] ret = new CEASDataType[argList.size()];
	Object[] a = argList.toArray();
	for (int i=0; i < a.length; i++)
		ret[i] = ((CEASDataType)a[i]).copy();
	return ret;
}

/**
 * Executes a function call and returns the result.
 *
 * @param w A walker object, used to evaluate functions
 * @param funcName The name bound to the function
 * @param argList A list of CEASDataTypes that are the args of the function
 * @return An optional data type
 */
public CEASDataType doFunctionCall(CEASWalker w, String funcName,
		Vector<CEASDataType> argList)
	throws RecognitionException
{
	//first we make copies of the arguments

	CEASDataType[] cargs = copyFunctionArgs(argList);
	CEASDataType ret = null;

	//all type checking was already handled by the analyzer
	if (FUNC_EXTRACT.equals(funcName)) {
		FunctionType.extract(cargs);
	}
	else if (FUNC_CREATE_PAGE.equals(funcName)) {
		PageDataType p = null;
		try {
			p = FunctionType.createPage(cargs);
		}
		catch (MalformedURLException mue) {
			CEASString cs = (CEASString) cargs[0];
			error("'" + cs + "' is not a valid URL.");
			return null;
		}
		return p;
	}
	else if (FUNC_SHOW.equals(funcName)) {
		try {
			FunctionType.show(cargs);
		}
		catch (CEASException e) {
			error("Could not connect to URL");
```

```
                }
        }
        else if (FUNC_PRINT.equals(funcName)) {
                func_print(false, cargs);
        }
        else if (FUNC_PRINTLN.equals(funcName)) {
                func_print(true, cargs);
        }
        else if (FUNC_TITLE.equals(funcName)){
                FunctionType.setTitle(cargs);
        }
        else if (FUNC_RANK.equals(funcName)){
                FunctionType.setRank(cargs);
        }
        else if (FUNC_STATUS.equals(funcName)){
                PageDataType p = (PageDataType) cargs[0];
                ret = new CEASBool(p.getStatus());
        }
        else if (FUNC_LENGTH.equals(funcName)) {
                CEASList l = (CEASList)cargs[0];
                ret = new CEASInt(l.length());
        }
        else if (FUNC_SAVE_PAGE.equals(funcName)){
                try {
                        ret = FunctionType.savePage(cargs);
                }
                catch (CEASException e) {
                        error("Could not connect to URL");
                }
        }
        else if (FUNC_APPEND.equals(funcName)) {
                FunctionType.append(cargs);
        }
        else {
                ret = doUserFunction(w, funcName, cargs);
        }
        return ret;
}

private CEASDataType doUserFunction(CEASWalker walker,
                        String funcName, CEASDataType[] argList) throws RecognitionException {

        CEASSymbolTable topSym;
        StringBuffer funcid = new StringBuffer(funcName);
        funcid.append("(");

        for (int i =0; i < argList.length; i++) {
                funcid.append(argList[i].typename());
                if (i != argList.length - 1)
                        funcid.append(",");
        }
        funcid.append(")");

        String sysname = funcid.toString();

        FunctionType f = (FunctionType)funcTable.getValue(sysname);
```

```
        if (f == null) {
                error("Function " + sysname + " has not been declared.");
                return null;
        }

        topSym = sym;
        sym = new CEASSymbolTable();
        CEASDataType actRetVal = null;
        FuncParameter[] f_vars = f.getArglist();

        /*
         * Goes through each variable in the stored function,
         * checks to see if the value passed is of the correct type.
         * If it is, then the value is *copied* into the symbol table
         * with the expected name. Otherwise, there will be an error
         * which will end execution.
         */
        for (int i=0; i < f_vars.length; i++) {
                if (f_vars[i].typeString().equals(argList[i].typename()))
                        sym.setValue(f_vars[i].getName(), argList[i]);
                else
                        error("Argument " + i + " to function " + funcName + " was " +
                                        argList[i].typename() + ", expected " + f_vars[i].typeString());
        }
        inFunction = true;
        returnVal = null;
        walker.expr(f.getBody());
        if (flow_state == FLOW_RETURN) {
                //check for bad conditions first
                if (f.isVoidFunction()) {
                        if (returnVal != null) {
                                error("Function " + sysname +
                                        " can not return values.");
                        }
                }
                else {
                        if ((returnVal == null ||
                                        !f.getReturnType().equals(returnVal.typename()))) {
                                error("Function " + sysname +
                                                " must return type " + f.getReturnType());
                        }
                }

                if (returnVal != null)
                        actRetVal = returnVal;
                flow_state = FLOW_NORMAL;
                returnVal = null;
        }
        else if (!f.isVoidFunction()) {
                //got to the end without a return?
                error("Function " + sysname +
                                " must return type " + f.getReturnType());
        }
        inFunction = false;
        sym = topSym;
        return actRetVal;
}
```

```
public CEASDataType doReturn(CEASDataType retval) {
	if (!inFunction)
		error("Return encountered outside of function body.");
	flow_state = FLOW_RETURN;

	if (retval != null)
		this.returnVal = retval.copy(); //non-empty return
	else
		this.returnVal = null; //empty return statement

	return returnVal;
}

private void func_print(boolean newline, CEASDataType[] argList) {
	if (argList.length != 1) {
		error("Error: print and println expect one argument got " +
				argList.length);
	}
	CEASDataType val = argList[0];

	if (newline)
		System.out.println(val);
	else
		System.out.print(val);

}

/**
 * Called when a new block of code is entered. This happens whenver a funciton
 * is defined, a loop is executed, or a block is discovered in the AST. The
 * implementing function is expected to modify the current, lowest level
 * symbol table.
 *
 */
public void enterBlock() {
	/* creates a new table with the current main table as the main and the
	   the curren table as its direct parent.
	 */
	sym = new CEASSymbolTable(sym.mainTable, sym);
}

/**
 * Called when a block of code ends. The implementing funciton is generally going
 * to throw away its lowest-level symbol table.
 *
 */
public void exitBlock() {
	//throws away the current table
	sym = sym.parent();
}

public void doBreak() {
	flow_state = FLOW_BREAK;
}

public void doContinue() {
```

```
		flow_state = FLOW_CONTINUE;
}

/**
 * An initialization function called when a for loop is about to begin.
 *
 * @param varName The name of the variable used to iterate through the for loop
 * @param first The initial value of the variable
 * @param last The final value of the variable
 */
public void beginForLoop(String varName, CEASDataType first, CEASDataType last) {
		enterBlock(); //creates new symbol table
		sym.for_init = ((CEASInt)first).value;
		sym.for_last = ((CEASInt)last).value;
		sym.for_var = new CEASInt(varName, sym.for_init);
		sym.setValue(varName, sym.for_var); //this variable is in new block
}

/**
 * Determines if a for loop can continue to operate.
 *
 * @return true if the for loop should go through another interation
 */
public boolean canForContinue(){
		int curVal = sym.for_var.value;
		//if we've passed the bounds of the for loop, we're done

		if (flow_state == FLOW_BREAK) {
			flow_state = FLOW_NORMAL;
			return false;
		}
		else if (flow_state == FLOW_RETURN) {
			return false;
		}
		else {
			flow_state = FLOW_NORMAL; //in case we're continuing
			//returns true if the current index is less than the
			//upper boud, false otherwise
			return (curVal < sym.for_last);
		}

}

/**
 * Used to update internal state at the end of a for loop.
 *
 */
public void incrementForState() {
		sym.for_var.value++;
}

/**
 * Cleans up any state after the for loop exits.
 *
 */
public void exitForLoop() {
		sym.for_init = -1;
```

```
		sym.for_last = -1;
		sym.for_var = null;
		if (flow_state != FLOW_RETURN)
			flow_state = FLOW_NORMAL;
		exitBlock();

}

static int numLoops = 0;
/**
 * Initializes state for a while or do-while loop.
 *
 */
public void beginWhileLoop() {
		//enterBlock();
}

/**
 * Returns state to pre-loop settings
 */
public void exitWhileLoop() {
		//exitBlock();
		if (flow_state != FLOW_RETURN)
			flow_state = FLOW_NORMAL;
}

/**
 * Test to see if loop can continue. Based on current system state and
 * the input parameter.
 *
 * @param check The expression used to maintain the while or do loop
 * @return true if check is true and a break has not been encountered
 */
public boolean testLoopBounds(CEASDataType check) {
		if (!(check instanceof CEASBool)) {
			//throws an exception
			error("Type mistmatch: expected boolean got " + check.typename());
		}

		//first check to see if a break or continue was encountered
		if (flow_state == FLOW_CONTINUE) {
			flow_state = FLOW_NORMAL;
		}
		else if (flow_state == FLOW_BREAK) {
			flow_state = FLOW_NORMAL;
			return false;
		}
		else if (flow_state == FLOW_RETURN) {
			return false;
		}

		//check the acutal bounds
		return ((CEASBool) check).value;
}

/**
 * Used for determining if the interpreter can continue to move
```

```
	 * or should ignore some comments.
	 *
	 * @return true if the program can continue to flow normally
	 */
	public boolean flowNormal() {
		return flow_state == FLOW_NORMAL;
	}

	/**
	 * Create an error message that will end the program.
	 *
	 * @param message
	 */
	public void error(String message) {
		System.err.println("Error: " + message);
		throw new CEASException(message);
	}
}
```

File: ceas/CEASSymbolTable.java

```
package ceas;
/**
 * CEASSymbolTable.java
 *
 * Author:  Kate McCarthy
 * Created: October 26, 2005
 *
 * Symbol table for the CEAS language which stores variable
 * names and their associated values.  Scope is specified
 * through a hierarchy of parent tables and a main table.
 *
 * @author Kate McCarthy
 * @version $Id: CEASSymbolTable.java,v 1.1 2005/12/19 16:59:24 lra2103 Exp $
 */

import java.util.*;

import ceas.types.CEASConstant;
import ceas.types.CEASDataType;
import ceas.types.CEASInt;
import ceas.types.PageDataType;

public class CEASSymbolTable extends HashMap<String, CEASDataType> {

  /*
   * mainTable is the parent of all the other parentTable symbol tables (top-level symbol table)
   * parentTable is the symbol table located up one level of scope from the current table
   *
   * Note: Constructor for the interpreter creates the top-level symbol table
   * which does not have a mainTable or parentTable associated with itself
   *
   */


  CEASSymbolTable mainTable, parentTable;
```

```
public static final long serialVersionUID = 345678L;

public final String extract[] = {PageDataType.EXTRACT_IMAGES,
                                 PageDataType.EXTRACT_IMGLINKS,
                                 PageDataType.EXTRACT_ADS,
                                 PageDataType.EXTRACT_FLASH,
                                 PageDataType.EXTRACT_SCRIPTS,
                                 PageDataType.EXTRACT_IFRAME,
                                 PageDataType.EXTRACT_BUTTON,
                                 PageDataType.EXTRACT_INPUT,
                                 PageDataType.EXTRACT_FORMS,
                                 PageDataType.EXTRACT_EMPTYTBLS,
                                 PageDataType.EXTRACT_STYLES,
                                 PageDataType.EXTRACT_XTRNSTYLE,
                                 PageDataType.EXTRACT_META,
                                 PageDataType.EXTRACT_TXTLINKS,
                                 PageDataType.EXTRACT_LINKLISTS};

public CEASInt for_var = null;
public int for_init = -1;
public int for_last = -1;
public boolean inloop = false;

public CEASSymbolTable() {
        this(null,null);
        for (int i=0; i < extract.length; i++) {
                this.setValue(extract[i], new CEASConstant(extract[i]));
        }

}

/*
 * Creates a symbol table with the specified mainTable and parentTable
 *
 */

public CEASSymbolTable( CEASSymbolTable mainTable, CEASSymbolTable parentTable ) {
        this.mainTable = mainTable;
        this.parentTable = parentTable;
}

/*
 * Obtains the parentTable of the specified symbol table or
 * obtains the mainTable if getMain evaluates to true
 */

public CEASSymbolTable parent( boolean getMain ) {
        if( getMain )
           return mainTable;
        else
           return parentTable;
}

/*
 * Obtains the parentTable of the specified symbol table
 *
 */
```

```
public CEASSymbolTable parent() {
        return parentTable;
}

/*
 * Obtains the mainTable symbol table
 *
 */

public CEASSymbolTable getMainTable() {
        return mainTable;
}

/*
 * Checks the symbol table for the variable name
 *
 */

public boolean containsVariable( String name ) {
        return containsKey( name );
}

/*
 * Obtains the value associated with the variable name
 * by looking for the variable name in the symbol table symTable.
 * If the value for name is not there, keep looking up in the parentTable
 * of each symbol table until the value for name is found or
 * there are no more parentTables to look at, in which case look in mainTable.
 */

public CEASDataType getValue( String name ) {
        CEASSymbolTable symTable = this;

        CEASDataType isThere = symTable.get( name );

        while( isThere == null && symTable.parent() != null ) {
           symTable = symTable.parent();
           isThere = symTable.get( name );
        }

        if( isThere == null && this.parent() == null ) {
           symTable = this.mainTable;

        if( symTable != null )
           isThere = symTable.get( name );
        }

        return isThere;
}

/*
 * Obtains the value associated with the variable name
 * by looking in the scope of the symbol table symTable
 * only if searchThisScope evaluates to true.
 * Otherwise, calls the function getValue( String name ).
 */
```

```
public CEASDataType getValue( String name, boolean searchThisScope ) {
        if( searchThisScope )
           return this.get( name );
        else
           return this.getValue( name );
}

/*
 * Declares a variable name and sets the value of name in the symbol table
 *
 */

public void setValue( String name, CEASDataType value ) {
        CEASDataType t;
        t = this.get(name);
        if (t == null) {
                value.name = name;
                this.put( name, value );
        }
        else {
                throw new CEASException( "CEASSymbolTabls.setValue(): Variable '" + name +
                                "' has already been defined with a value"  );
        }
}

/*
 * Calls the function getScope( String name ) to obtain the symbol table
 * containing the variable name and sets a new value of name in the symbol table
 *
 */

public void resetValue( String name, CEASDataType value ) {
        CEASSymbolTable symTable = this.getScope( name );
        symTable.remove( name );
        value.name = name;
        symTable.put( name, value );
}

/*
 * Obtains the symbol table containing the variable name
 *
 */

public CEASSymbolTable getScope( String name ) {
        CEASSymbolTable symTable = this;
        CEASDataType isThere = symTable.get( name );

        while( isThere == null && symTable.parent() != null ) {
                symTable = symTable.parent();
                isThere = symTable.get( name );
        }

        return symTable;
}

/*
```

```
   * Prints out the values in the symbol table as well as those in its parentTable
   *
   */

  public String printTables() {
          String str = "SYMBOL TABLE\n";

          CEASSymbolTable symtable = this;
          String key = "";

          while( symtable != null ) {
                  str += "# SYMBOLS: " + symtable.keySet().size() + "\n";
                  for(Iterator it = symtable.keySet().iterator(); it.hasNext(); ) {
                          key = (String) it.next();
                          str += "key: " + key + "\tvalue: " + symtable.getValue(key) + "\n";
                  }
                  symtable = symtable.parent();
          }
          return str;
  }
}
```

File: ceas/CEASWalkIfc.java

```
package ceas;
/**
 * An interface used by the tree walker to track state of the system.
 * Should be implemented by the semantic analyzer and the interpreter.
 *
 * @author Luis Alonso
 * @version $Id: CEASWalkIfc.java,v 1.1 2005/12/19 16:59:24 lra2103 Exp $
 *
 */
import java.util.Vector;
import antlr.collections.AST;
import antlr.RecognitionException;

public interface CEASWalkIfc {

          /**
           * Constants representing built-in functions in the language.
           */
          static final String FUNC_CREATE_PAGE = "createPage";
          static final String FUNC_CREATE_PAGE_1 = "createPage(string)";
          static final String FUNC_CREATE_PAGE_2 = "createPage(Page)";
          static final String FUNC_CREATE_PAGE_3 = "createPage(string,string)";
          static final String FUNC_EXTRACT = "extract";
          static final String FUNC_SHOW = "show";
          static final String FUNC_SHOW_1 = "show(Page)";
          static final String FUNC_SHOW_2 = "show(Page[])";
          static final String FUNC_PRINT = "print";
          static final String FUNC_PRINTLN = "println";
          static final String FUNC_APPEND = "append";
          static final String FUNC_APPEND_NAME = "append(Page,string)";
          static final String FUNC_TITLE = "title";
          static final String FUNC_TITLE_NAME = "title(Page,string)";
```

```
        static final String FUNC_RANK = "rank";
        static final String FUNC_RANK_NAME = "rank(Page,int)";
        static final String FUNC_STATUS = "status";
        static final String FUNC_STATUS_NAME = "status(Page)";
        static final String FUNC_LENGTH = "length";
        static final String FUNC_SAVE_PAGE = "savePage";
        static final String FUNC_SAVE_PAGE_NAME = "savePage(Page,string)";

        static final String LIST_FLAG = "LIST";

        /**
         * For flow control.
         */
        static final int FLOW_NORMAL = 0;
        static final int FLOW_RETURN = 1;
        static final int FLOW_BREAK = 2;
        static final int FLOW_CONTINUE = 3;


}
```

File: ceas/CommonASTWithLines.java

```
package ceas;
/**
 * From Chris Conway
 *
 */
import antlr.CommonAST;
import antlr.Token;

public class CommonASTWithLines extends CommonAST {

        public static final long serialVersionUID = 12313L;
   private int line = 0;
   private int column = 0 ;

   public void initialize(Token tok) {
       super.initialize(tok);
       line=tok.getLine();
       column = tok.getColumn();
   }

   public int getLine(){
      return line;
   }

   public int getColumn() {
      return column;
   }

}
```

File: ceas/FuncParameter.java

```
package ceas;
import java.util.Enumeration;
import java.util.Vector;

/**
 * Simple class for storing information about function parameters.
 * Used to consolidate information about type and name of
 * parameters.
 */

/**
 * @author Luis Alonso
 * @version $Id: FuncParameter.java,v 1.2 2005/12/20 21:45:30 lra2103 Exp $
 */
public class FuncParameter {

	private String type;
	private String name;
	private boolean list;

	FuncParameter(String type, String name, boolean list) {
		this.type = type;
		this.list = list;
		this.name = name;
	}

	FuncParameter(String type, String name) {
		this(type, name, false);
	}

	public String getType() {
		return type;
	}

	public String getName() {
		return name;
	}

	public boolean isList() {
		return list;
	}

	public String typeString() {
		if (list)
			return (type + "[]");
		else
			return type;
	}

	public String toString() {
		return (typeString() + name);
	}

	public static FuncParameter[] vectorToArray(Vector<FuncParameter> args) {
		FuncParameter ret[] = new FuncParameter[args.size()];
```

```
                Enumeration<FuncParameter> e = args.elements();
                int i = 0;
                while (e.hasMoreElements()) {
                        ret[i] = e.nextElement();
                        i++;
                }
                return ret;
        }

}
```

Package: ceas.types

File: ceas/types/CEASDataType.java

```
package ceas.types;
/**
 * CEASDataType.java
 *
 * Author:  Kate McCarthy
 * Created: November 7, 2005
 *
 * Parent class of all data types in the CEAS language.
 * $Id: CEASDataType.java,v 1.1 2005/12/19 16:59:46 lra2103 Exp $
 */

import java.io.PrintWriter;

import ceas.CEASException;

public class CEASDataType {

  public static final String STRING_TYPE = "string";
  public static final String PAGE_TYPE = "Page";
  public static final String INT_TYPE = "int";
  public static final String NULL_TYPE = "null";
  public static final String BOOL_TYPE = "boolean";
  public static final String VOID_TYPE = "void";
  public static final String LIST_TYPE = "list";

  public String name;

  //false: variable created, not initialized
  //true: variable created and initialized
  public boolean initialized;


  /**
   * Factory method for generating empty types.
   * @param typeName Type to be created
   * @param varName name of the variable (null if has no name)
   * @return
   */
  public static CEASDataType createType(String typeName, String varName) {
```

```
		if (typeName.endsWith("[]")) {
			String t = typeName.substring(0,typeName.length()-2);
			CEASList l = new CEASList(t,varName);
			return l;
		}
		if (STRING_TYPE.equals(typeName))
			return new CEASString(varName);
		else if (PAGE_TYPE.equals(typeName))
			return new PageDataType(); //no name constructor
		else if (INT_TYPE.equals(typeName))
			return new CEASInt(varName);
		else if (BOOL_TYPE.equals(typeName))
			return new CEASBool(varName);
		else {
			System.err.println("Unknown type");
			return null;
		}

}

/**
 * Constructor - initializes the name of the data type to null
 *
 */
public CEASDataType() {
	initialized = false;
	name = null;
}

/**
 * Constructor - initializes the name of the data type
 *
 * @param name
 */
public CEASDataType( String name ) {
	initialized = false;
	this.name = name;
}

/**
 * Sets the name of this object
 *
 * @param newName
 */
public void setName(String newName) {
	this.name = newName;
}

/**
 * Obtains the value of the object
 *
 */
public String getName() {
	return name;
}

/**
```

```
 * Obtains the type name of the object
 *
 * @return CEASDataType is an unknown data type
 *
 */
public String typename() {
	return "unknown";
}

/**
 * Determines if an object has been declared and initialized. Child classes
 * should handle this properly.
 *
 * @return true when variable has been set at least once
 */
public boolean initialized() {
	 return initialized;
}

/**
 * Creates a copy of the object instance
 *
 */
public final CEASDataType copy() {
	 if (!initialized)
		return error("Variable " + this.name + " has not been initialized.");
	 CEASDataType ret = _copy();
	 if (ret != null)
		ret.initialized = true;
	return ret;
}

public CEASDataType _copy() {
	 return null;
}

/**
 * Sets the name of the object
 *
 * @param name
 */
public void setname( String name ) {
	this.name = name;
}

/**
 * Throws an exception with the specified error message to be displayed
 *
 * @param msg
 */
public CEASDataType error( String msg ) {
	throw new CEASException( "illegal operation: " + msg
					+ "( <" + typename() + "> "
					+ ( name != null ? name : "<?>" )
					+ " )" );
}
```

```
/**
 * Throws an exception with the specified error message to be displayed
 * along with the object b involved in the error
 *
 * @param b, msg
 */
public CEASDataType error( CEASDataType b, String msg ) {
        if( null == b )
                return error( msg );
        throw new CEASException( "illegal operation: " + msg
                                        + "( <" + typename() + "> "
                                        + ( name != null ? name : "<?>" )
                                        + " and "
                                        + "<" + typename() + "> "
                                        + ( name != null ? name : "<?>" )
                                        + " )" );
}

public void print( PrintWriter w ) {
        if( name != null )
                w.print( name + " = " );
        w.println( "<undefined>" );
}

public void print() {
        print( new PrintWriter( System.out, true ) );
}

public void what( PrintWriter w ) {
        w.print( "<" + typename() + "> ");
        print( w );
}

public void what() {
        what( new PrintWriter(System.out, true) );
}


public final CEASDataType minus(CEASDataType b) {
          if (!initialized)
                  return error("Variable '" + this.name + "' has not been initialized");
          else if (!b.initialized)
                  return error("Variable '" + b.name + "' has not been initialized");

          return _minus(b);
}

public CEASDataType _minus(CEASDataType b) {
          if (!initialized)
                  return error("Variable '" + this.name + "' has not been initialized");
          else if (!b.initialized)
                  return error("Variable '" + b.name + "' has not been initialized");
          return error( "-" );
}

public final CEASDataType plus( CEASDataType b ) {
          if (!initialized)
```

```
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
        return _plus(b);
}

public CEASDataType _plus( CEASDataType b) {
        return error( b, "+" );
}

public final CEASDataType mult( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
        return _mult(b);
}

public CEASDataType _mult( CEASDataType b) {
        return error( b, "*" );
}

public final CEASDataType div( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
        return _div(b);
}

public CEASDataType _div( CEASDataType b) {
        return error( b, "/" );
}

public final CEASDataType mod( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
        return _mod(b);
}

public CEASDataType _mod (CEASDataType b) {
        return error(b, "%");
}

public final CEASDataType assign( CEASDataType b ) {
        if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");

        CEASDataType ret = _assign(b);
        initialized = true;
        return ret;
}

public CEASDataType _assign( CEASDataType b) {
        return error(b, "=");
```

```
}

public final CEASDataType equal( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
      return _equal(b);
}

public CEASDataType _equal (CEASDataType b) {
        return error( b, "==");
}

public final CEASDataType grt( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
      return _grt(b);
}

public CEASDataType _grt( CEASDataType b) {
        return error( b, ">" );
}

public final CEASDataType lst( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
      return _lst(b);
}

public CEASDataType _lst (CEASDataType b) {
        return error( b, "<");
}

public final CEASDataType grteq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
      return _grteq(b);
}

public CEASDataType _grteq( CEASDataType b) {
        return error( b, ">=");
}

public final CEASDataType lsteq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
      return _lsteq(b);
}
```

```
public CEASDataType _lsteq( CEASDataType b) {
        return error( b, "<=");
}

public final CEASDataType noteq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
       return _noteq(b);
}

public CEASDataType _noteq( CEASDataType b) {
        return error(b, "!=");
}

public final CEASDataType pluseq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");

        CEASDataType ret = _pluseq(b);
        initialized = true;
       return ret;
}

public CEASDataType _pluseq( CEASDataType b) {
        return error( b, "+=");
}

public final CEASDataType minuseq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
        CEASDataType ret = _minuseq(b);
        initialized = true;
       return ret;
}

public CEASDataType _minuseq( CEASDataType b) {
        return error (b, "-=");
}

public final CEASDataType multeq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");

        CEASDataType ret = _multeq(b);
        initialized = true;
       return ret;
}
```

```
public CEASDataType _multeq( CEASDataType b) {
        return error( b, "*=" );
}

public final CEASDataType diveq( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");

        CEASDataType ret = _diveq(b);
        initialized = true;
       return ret;
}

public CEASDataType _diveq( CEASDataType b) {
        return error( b, "/=");
}

public final CEASDataType rem( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
        CEASDataType ret = _rem(b);
        initialized = true;
       return ret;
}

public CEASDataType _rem( CEASDataType b) {
        return error (b, "%=");
}

public final CEASDataType and( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
       return _and(b);
}

public CEASDataType _and( CEASDataType b) {
        return error(b, "&");
}

public final CEASDataType or( CEASDataType b ) {
        if (!initialized)
                return error("Variable '" + this.name + "' has not been initialized");
        else if (!b.initialized)
                return error("Variable '" + b.name + "' has not been initialized");
       return _or(b);
}

public CEASDataType _or( CEASDataType b) {
        return error(b, "|");
}
```

```
  public final CEASDataType not() {
            if (!initialized)
                    return error("Variable '" + this.name + "' has not been initialized");
          return _not();
  }

  public CEASDataType _not() {
            return error( "!" );
  }

  public final CEASDataType getListElement(CEASDataType index) {
            if (!initialized)
                    return error("Variable '" + this.name + "' has not been initialized");
            return _getListElement(index);
  }

  public CEASDataType _getListElement(CEASDataType index) {
            return error (this, "List Access on non-list object");
  }

  public final void setListElement(CEASDataType index, CEASDataType val) {
            if (!initialized)
                    error("Variable '" + this.name + "' has not been initialized");
            _setListElement(index, val);
            initialized = true;
  }

  public void _setListElement(CEASDataType index, CEASDataType val) {
            error (this, "List Access on non-list object");
  }

  public String toString() {
            error("Illegal operation: Cannot print " + (this.name == null ? "variable" : "'" + this.name + "'"));
            return "";
  }

}
```

File: ceas/types/CEASBool.java

```
package ceas.types;
import java.io.PrintWriter;

/**
 * Class for representing boolean values.
 *
 * @author Luis Alonso and Kate McCarthy
 * @version $Id: CEASBool.java,v 1.1 2005/12/19 16:59:46 lra2103 Exp $
 */
public class CEASBool extends CEASDataType {

            public static final CEASBool TRUE = new CEASBool(true);
            public static final CEASBool FALSE = new CEASBool(false);

            public boolean value;
```

```
/**
 * Constructor calling parent 'CEASDataType' constructor
 * @param name
 */
public CEASBool(String name) {
	super(name);
}

/**
 * Constructor calling parent 'CEASDataType' constructor and initializing the boolean value to value
 * @param value
 */
public CEASBool(boolean value) {
	this.value = value;
	this.initialized = true;
}

/**
 * Obtains the type name of the object
 *
 */
public String typename() {
	return BOOL_TYPE;
}

/**
 * Creates a copy of the object instance
 *
 */
public CEASDataType _copy() {
	return new CEASBool(value);
}

public void print( PrintWriter w ) {
	if( name != null )
		w.print( name + " = " );
	w.println( value ? "true" : "false" );
}

/**
 * Overrides the generic assign
 *
 * @param newVal - a CEASBool to be assigned to this variable
 * @return this variable
 */
public CEASDataType _assign(CEASDataType newVal) {
	if (newVal instanceof CEASBool) {
		this.value = ((CEASBool)newVal).value;
		return this;
	}
	else {
		return error(newVal, "assign");
	}
}

/**
 * Compares two CEASBool objects
```

```
 *
 *  @param b
 *  @return true if both objects are equal, false if the objects are unequal
 */
public CEASDataType _equal(CEASDataType b) {
	if (b instanceof CEASBool) {
		if (this.value == ((CEASBool) b).value) {
			return CEASBool.TRUE;
		}
		else {
			return CEASBool.FALSE;
		}
	}
	else {
		return error(b, "==");
	}
}

/**
 * Compares two CEASBool objects
 *
 *  @param b
 *  @return true if the objects are unequal, false if both objects are equal
 */
public CEASDataType _noteq(CEASDataType b) {
	if (b instanceof CEASBool) {
		if (this.value != ((CEASBool) b).value) {
			return CEASBool.TRUE;
		}
		else {
			return CEASBool.FALSE;
		}
	}
	else {
		return error(b, "!=");
	}
}

/**
 * Implements logical & of two CEASBool objects
 *
 *  @param b
 *  @return true if both objects are true, false otherwise
 */
/* These should be short circuit */
public CEASDataType _and(CEASDataType b) {
	if (b instanceof CEASBool) {
		if (value && ((CEASBool)b).value)
			return CEASBool.TRUE;
		else
			return CEASBool.FALSE;
	}
	return error(b, "&");
}

/**
 * Implements logical | of two CEASBool objects
```

```
	 *
	 *  @param b
	 *  @return true if one or more of the objects are true, false otherwise
	 */
	public CEASDataType _or(CEASDataType b) {
		if (b instanceof CEASBool) {
			if (value || ((CEASBool)b).value)
				return CEASBool.TRUE;
			else
				return CEASBool.FALSE;
		}
		return error(b, "|");
	}

	/**
	 * Implements logical ! of a CEASBool object
	 *
	 *  @return true if object is false, false if the object is true
	 */
	public CEASDataType _not() {
		if (value)
			return CEASBool.FALSE;
		else
			return CEASBool.TRUE;
	}

	/**
	 * String representation of this CEASBool object
	 * @return "true" or "false"
	 */
	public String toString() {
		return Boolean.toString(value);
	}
}
```

File: ceas/types/CEASChar.java

```
package ceas.types;
import java.io.PrintWriter;

/**
 * Data type for representing and manipulating characters.
 *
 * @author Kate McCarthy
 *
 * @version
 *
 */

public class CEASChar extends CEASDataType {

	char value;

	/**
	 * Constructor calling parent 'CEASDataType' constructor
```

```
 * @param name
 */
public CEASChar(String name) {
	super(name);
}

/**
 * Constructor calling parent 'CEASDataType' constructor and initializing the character value to val
 * @param name, val
 */
public CEASChar(String name, char val) {
	super(name);
	value = val;
}

/**
 * Default constructor creates a character data type
 * @param val
 */
public CEASChar(char val) {
	value = val;
}

/**
 * Obtains the type name of the object
 *
 */
public String typename() {
	return "char";
}

/**
 * Creates a copy of the object instance
 *
 */
public CEASDataType _copy() {
	return new CEASChar(value);
}

public void print( PrintWriter w ) {
	if( name != null )
		w.print( name + " = " );
	w.print(value);
	w.println();
}

/**
 * Obtains the value of the object
 *
 */
public char getValue() {
	return value;
}

/**
 * Overrides the generic assign
 *
```

```
	 * @param newVal - a CEASChar to be assigned to this variable
	 * @return this variable
	 */
	public CEASDataType _assign(CEASDataType newVal) {
		CEASChar newchar = (CEASChar) newVal;
		this.value = newchar.value;
		return this;
	}

	public String toString() {
		return Character.toString(value);
	}
}
```

File: ceas/types/CEASCheck.java

```
package ceas.types;

public class CEASCheck extends CEASDataType {

	public static final int STRING = 0;
	public static final int INT = 1;
	public static final int BOOL = 2;
	public static final int PAGE = 3;
	public static final int VOID = 4;
	public static final int LIST = 5;
	public static final int CONSTANT = 6;

	private int checktype;
	private int listtype;

	public CEASCheck() {
		super();
		// TODO Auto-generated constructor stub
	}

	public CEASCheck(String name, int type) {
		super(name);
		checktype = type;
		// TODO Auto-generated constructor stub
	}

	public CEASCheck(int type) {
		super();
		checktype = type;
		if (checktype == CONSTANT) {
			initialized = true;
		}
	}

	public static CEASCheck createType(String typeName, String varName) {
		if (typeName.endsWith("[]")) {
			String t = typeName.substring(0, typeName.length() - 2);
			return createListType(t, varName);
		}
		if (STRING_TYPE.equals(typeName))
			return new CEASCheck(varName, STRING);
```

```
		else if (PAGE_TYPE.equals(typeName))
			return new CEASCheck(varName, PAGE); //no name constructor
		else if (INT_TYPE.equals(typeName))
			return new CEASCheck(varName, INT);
		else if (BOOL_TYPE.equals(typeName))
			 return new CEASCheck(varName, BOOL);
		else if (VOID_TYPE.equals(typeName))
			return new CEASCheck(varName, VOID);
		else {
			System.err.println("Unknown type");
			return null;
		}
}

public static CEASCheck createListType(CEASCheck base) {
		CEASCheck c = new CEASCheck();
		c.checktype = LIST;
		c.listtype = base.checktype;
		return c;
}

public static CEASCheck createListType(String typeName, String varName) {
		CEASCheck c = new CEASCheck(varName, LIST);
		if (STRING_TYPE.equals(typeName))
			c.listtype = STRING;
		else if (PAGE_TYPE.equals(typeName))
			c.listtype = PAGE;
		else if (INT_TYPE.equals(typeName))
			c.listtype = INT;
		else if (BOOL_TYPE.equals(typeName))
			c.listtype = BOOL;
		else {
			System.err.println("Unknown type");
			c = null;
		}
		return c;
}

public boolean equalTypes(CEASCheck right) {
		if (checktype == right.checktype) {
			if (checktype == LIST) {
				return listtype == right.listtype;
			}
			else {
				return true;
			}
		}
		else {
			return false;
		}
}

public boolean isList() {
		return checktype == LIST;
}

public int checkType() {
```

```
        return checktype;
}

public int checkListType() {
        return listtype;
}

public boolean isBool() {
        return checktype == BOOL;
}

public boolean isInt() {
        return checktype == INT;
}

public boolean isPage() {
        return checktype == PAGE;
}

public boolean isString() {
        return checktype == STRING;
}

public boolean isVoid() {
        return checktype == VOID;
}

public boolean isConstant() {
        return checktype == CONSTANT;
}

public String typename() {
        return typeToString(checktype);
}

private String typeToString(int t) {
        switch (t) {
        case BOOL:
                return BOOL_TYPE;
        case INT:
                return INT_TYPE;
        case LIST:
                return (typeToString(listtype) + "[]");
        case PAGE:
                return PAGE_TYPE;
        case STRING:
                return STRING_TYPE;
        case VOID:
                return VOID_TYPE;
        case CONSTANT:
                return "String Constant";
        default:
                return "unknown";

        }
}
```

```
}
```

File: ceas/types/CEASConstant.java

```
package ceas.types;

public class CEASConstant extends CEASString {

	public CEASConstant (String name) {
		super(name, name);
	}

	public CEASDataType _assign(CEASDataType newval) {
		return error("Can not assign values to constants.");
	}
}
```

File: ceas/types/CEASInt.java

```
package ceas.types;
import java.io.PrintWriter;

/**
 * @author Kate McCarthy
 *
 * @version $Id: CEASInt.java,v 1.1 2005/12/19 16:59:46 lra2103 Exp $
 */

public class CEASInt extends CEASDataType {

	public int value;

	/**
	 * Constructor calling parent 'CEASDataType' constructor
	 * @param name
	 */
	public CEASInt(String name) {
		super(name);
		value = 0;
		initialized = false;
	}

	/**
	 * Constructor calling parent 'CEASDataType' constructor and initializing the integer value to val
	 *@param name, val
	 */
	public CEASInt(String name, int val) {
		super(name);
		value = val;
		initialized = true;
	}
```

```
	/**
	 * Default constructor creates an integer data type
	 * @param val
	 */
	public CEASInt(int val) {
		value = val;
		initialized = true;
	}

	/**
	 * Obtains the type name of the object
	 *
	 */
	public String typename() {
		return INT_TYPE;
	}

	/**
	 * Creates a copy of the object instance
	 *
	 */
	public CEASDataType _copy() {
		return new CEASInt(value);
	}

	public static int intValue(CEASDataType b) {
		if (b instanceof CEASInt)
			return ((CEASInt) b).value;
		b.error("cast to int");
		return 0;
	}

	public void print( PrintWriter w ) {
		if( name != null )
			w.print( name + " = " );
		w.println( Integer.toString(value) );
	}

	public CEASDataType _minus( CEASDataType b) {
		return new CEASInt( value - intValue(b));
	}

	public CEASDataType _plus( CEASDataType b ) {
		return new CEASInt( value + intValue(b) );
	}

	public CEASDataType _mult( CEASDataType b ) {
		return new CEASInt( value * intValue(b) );
}

	public CEASDataType _div( CEASDataType b ) {
		return new CEASInt( value / intValue(b) );
	}

	public CEASDataType _mod( CEASDataType b ) {
		return new CEASInt( value % intValue(b) );
	}
```

```
        public CEASDataType _assign( CEASDataType b ) {
                if (b instanceof CEASInt) {
                        CEASInt bval = (CEASInt) b;
                        this.value = bval.value;
                        return this;
                }
                else
                        return error(b, "=");
}

        public CEASDataType _equal( CEASDataType b ) {
        if ( b instanceof CEASInt )
        return new CEASBool( value == intValue(b) );
    return b.equal( this );
}

        public CEASDataType _grt( CEASDataType b ) {
                if ( b instanceof CEASInt )
                        return new CEASBool( value > intValue(b) );
                return b.grt( this );
        }

        public CEASDataType _lst( CEASDataType b ) {
                if ( b instanceof CEASInt )
                        return new CEASBool( value < intValue(b) );
                return b.lst( this );
        }

        public CEASDataType _grteq( CEASDataType b ) {
                if ( b instanceof CEASInt )
                        return new CEASBool( value >= intValue(b) );
                return b.lsteq( this );
        }

        public CEASDataType _lsteq( CEASDataType b ) {
                if ( b instanceof CEASInt )
                        return new CEASBool( value <= intValue(b) );
                return b.grteq( this );
        }

        public CEASDataType _noteq( CEASDataType b ) {
    if ( b instanceof CEASInt )
       return new CEASBool( value != intValue(b) );
    return b.noteq( this );
}

        public CEASDataType _pluseq( CEASDataType b ) {
                value += intValue( b );
                return this;
        }

        public CEASDataType _minuseq( CEASDataType b ) {
                value -= intValue( b );
                return this;
        }
```

```
	public CEASDataType _multeq( CEASDataType b ) {
		value *= intValue( b );
		return this;
	}

	public CEASDataType _diveq( CEASDataType b ) {
		value /= intValue( b );
		return this;
	}

	public CEASDataType _rem( CEASDataType b ) {
		value %= intValue( b );
		return this;
	}

	/**
	 * String representation of this CEASInt
	 * @return integer value
	 */
	public String toString() {
		return Integer.toString(this.value);
	}

}
```

File: ceas/types/CEASList.java

```
package ceas.types;



/**
 * Data type for representing and manipulating lists.
 *
 * @author Kate McCarthy
 *
 * @version $Id: CEASList.java,v 1.1 2005/12/19 16:59:46 lra2103 Exp $
 *
 */

	public class CEASList extends CEASDataType {

		CEASDataType list[];
		String type;

	/**
	 * Default constructor creates a generic list
	 *
	 */
	public CEASList(String type) {
	    this.list = null;
	    this.type = type;
	    name = null;
	}
```

```
/**
 * Overrides the name only constructor.
 * @param name Name of the variable this represents
 */
public CEASList(String type, String name) {
    super(name);
    this.type = type;
    this.list = null;
}

public CEASList(String type, String name, int length) {
    super(name);
    this.type = type;
    this.initialized = true;
    this.list = new CEASDataType[length];
    for (int i=0; i < length; i++) {
            this.list[i] = createType(type, null);
    }
}

public CEASList(String type, CEASDataType[] vals) {
    this.type = type;
    this.list = vals;
    this.initialized = true;
    name = null;
}

/**
 * Obtains the type name of the object
 *
 */
public String typename() {
    return (type + "[]");
   //return LIST_TYPE;
}

/**
 * Creates a copy of the object instance
 *
 */
public CEASDataType _copy() {
    CEASList ret = new CEASList(type);
    ret.list = new CEASDataType[this.list.length];

    //if the internal elements of the list have not been
    //set yet, then just create a new blank element
    for (int i=0; i < this.list.length; i++) {
            if (list[i].initialized())
                    ret.list[i] = list[i].copy();
            else {
                    ret.list[i] =
                            CEASDataType.createType(list[i].typename(),null);
            }
    }
   return ret;
}
```

```
/**
 * Obtains the elements in the list
 *
 */
public CEASDataType[] getList() {
   return list;
}

/**
 * Obtains the size of the list
 *
 */
public int getSize() {
   return list.length;
}

/**
 * Obtains the nth element in the list
 * @param index of the element to be retrieved
 */
public CEASDataType getListElement(int index) {
     if ((index >= list.length) || (index < 0))
               return error("Index out of bounds: " + index);
   return list[index];
}

public CEASDataType _getListElement(CEASDataType i) {
     int index;
     if (i instanceof CEASInt) {
               index = ((CEASInt)i).value;
               return getListElement(index);
     }
     return error(i, "List index expects int");
}

public void setListElement(int index, CEASDataType newVal) {
     if (index >= list.length || index < 0) {
               error("Index out of bounds: " + index);
     }
     if (type.equals(newVal.typename()))
               list[index] = newVal.copy();
     else
               error(newVal, "list assignment");
}

public void _setListElement(CEASDataType i, CEASDataType newVal) {
     int index = 0;

     if (i instanceof CEASInt) {
               index = ((CEASInt)i).value;
               setListElement(index, newVal);
     }
     error(newVal, "List index expects type int");
}

public void printList() {
     String out = "";
```

```
        CEASDataType element;

        if(list.length == 0)
                out += "List is empty";
        else {
                for(int i=0; i < list.length; i++) {
                        element = (CEASDataType) list[i];
                        out += "Element " + (i+1) + ": " + element + "\n";
                }
        }
                System.out.println(out);
    }

    public int length() {
        return list.length;
    }

    public CEASDataType _assign(CEASDataType nval) {

        if (nval instanceof CEASList) {
                CEASList n = (CEASList) nval;
                if (n.type.equals(this.type)) {
                        list = new CEASDataType[n.list.length];
                        for (int i=0; i < n.list.length; i++) {
                                list[i] = n.list[i].copy();
                        }
                        this.type = n.type;
                        return this;
                }
                return error("Got " + nval.typename() + " in assignment ");
        }
        return error(nval, "list assign");
    }

        /**
         * String representation of this CEASList object
         * @return list
         */
        public String toString() {
                if (list == null) {
                        return "[ ]";
                }

                StringBuffer buf = new StringBuffer("[");
                CEASDataType e;

                for (int i=0; i < list.length; i++) {
                        e = (CEASDataType) list[i];
                        buf.append(e.toString());
                        if (i != list.length - 1)
                                buf.append(",");
                }
                buf.append("]");

                return buf.toString();
        }
}
```

File: ceas/types/CEASString.java

```
package ceas.types;
import java.io.PrintWriter;

/**
 * Data type for representing and manipulating strings.
 *
 * @author Luis Alonso and Kate McCarthy
 *
 * @version $Id: CEASString.java,v 1.1 2005/12/19 16:59:46 lra2103 Exp $
 *
 */
public class CEASString extends CEASDataType {

	String value;

	/**
	 * Constructor calling parent 'CEASDataType' constructor
	 * @param name
	 */
	public CEASString(String name) {
		super(name);
		value = "";
	}

	/**
	 * Constructor calling parent 'CEASDataType' constructor and initializing the integer value to val
	 * @param name, val
	 */
	public CEASString(String name, String val) {
		super(name);
		value = val;
		this.initialized = true;
	}

	/**
	 * Obtains the type name of the object
	 *
	 */
	public String typename() {
		return STRING_TYPE;
	}

	/**
	 * Creates a copy of the object instance
	 *
	 */
	public CEASDataType _copy() {
		return new CEASString(null, value);
	}

	public void print( PrintWriter w ) {
		if( name != null )
			w.print( name + " = " );
```

```
		w.print(value);
		w.println();
	}

	/**
	 * Obtains the value of the object
	 *
	 */
	public String getValue() {
		return value;
	}

	/**
	 * Overrides the generic assign
	 *
	 * @param newVal - a CEASString to be assigned to this variable
	 * @return this variable
	 */
	public CEASDataType _assign(CEASDataType newVal) {
		//TODO Must do error checking
		CEASString newstr = (CEASString) newVal;
		this.value = newstr.value;
		return this;

	}

	/**
	 * Adds two CEASString objects
	 *
	 *  @param b
	 */
	public CEASDataType _plus(CEASDataType b) {
		if(b instanceof CEASString) {
			return new CEASString(value + ((CEASString) b).value);
		}
		return error(b, "+");
	}

	/**
	 * Adds two CEASString objects and assigns the result to the CEASString object
	 *
	 *  @param b
	 */
	public CEASDataType _pluseq(CEASDataType b) {
		if(b instanceof CEASString) {
			value = value + ((CEASString) b).value;
			return this;
		}
		return error(b, "+=");
	}

	public String toString() {
		return value;
	}
}
```

File: ceas/types/FunctionType.java

```
package ceas.types;
/**
 * An object for storing representations related to functions.
 * Inherits from CEASDataType so we can use the SymbolTable
 */

import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;

import javax.swing.JFrame;
import javax.swing.JTabbedPane;
import javax.swing.UIManager;

import ceas.CEASException;
import ceas.FuncParameter;

import Browser.Browser;
import antlr.collections.AST;

/**
 * @author Luis Alonso
 * @version $Id: FunctionType.java,v 1.3 2005/12/20 01:44:02 lra2103 Exp $
 */
public class FunctionType extends CEASDataType {

	//note: name will be the name + arg types in parens?
	private AST body = null;
	private String funcName = null;
	private FuncParameter[] args;
	private String returnType = VOID_TYPE;

	/**
	 *
	 * @param sysName The name of the function with the list of arg types for matching
	 * @param type The string version of the return type
	 * @param fname The actual function name
	 * @param argList CEASDataTypes for holding the input values
	 * @param body AST representation of the function body
	 */
	public FunctionType(String type, String fname, FuncParameter[] argList, AST body) {
		//super(sysName);
		this.body = body;
		returnType = type;
		args = argList;
		funcName = fname;
		name = generateSysName(fname, argList);
	}

	/**
	 * Given a function name and an array of FuncParameters, generates the
```

```
 * method signature for this method.
 *
 * @param fname Name of the function
 * @param argList Arguments to the function
 * @return Method signature
 */
public static String generateSysName(String fname, FuncParameter[] argList) {
	StringBuffer buf = new StringBuffer(fname);
	buf.append("(");
	for (int i=0; i < argList.length; i++) {
		buf.append(argList[i].typeString());
		if (i < argList.length - 1)
			buf.append(",");
	}
	buf.append(")");
	return buf.toString();
}

/**
 *
 */
public FunctionType() {
	super();
	// TODO Auto-generated constructor stub
}

/**
 * @param name
 */
public FunctionType(String name) {
	super(name);
	// TODO Auto-generated constructor stub
}

public AST getBody() {
	return body;
}

public String getReturnType() {
	return returnType;
}

public FuncParameter[] getArglist() {
	return args;
}

public String getActualName() {
	return funcName;
}

public boolean isVoidFunction() {
	return VOID_TYPE.equals(returnType);
}

public static void extract(CEASDataType[] args) {
	CEASString s;
	PageDataType p = (PageDataType) args[0];
```

```
		String extract[] = new String[args.length - 1];
		for (int i=0; i < extract.length; i++) {
			s = (CEASString) args[i + 1];
			extract[i] = s.getValue();
		}

		p.extract(extract);
	}

	public static PageDataType createPage(CEASDataType[] args)
		throws MalformedURLException {

		if (args.length == 1) {
			CEASDataType arg = args[0];
			if (arg instanceof PageDataType) {
				return new PageDataType((PageDataType) arg);
			}
			else {
				CEASString str = (CEASString) arg;
				return new PageDataType(str.getValue());
			}
		}
		else if (args.length == 2) {
			return new PageDataType (((CEASString)args[0]).value +
					((CEASString)args[1]).value);
		}
		else {
			//this should never happen due to semantic analyzer
			System.err.println("Unkown state");
			throw new CEASException("createPage received too many arguments.");
		}

	}

	private static void sort(PageDataType[] pages, int left, int right){

		if(left>=right) return;
		int temp = (pages[right]).getRank();
		int i = left - 1;
		int j = right;
			while(true) {
				while((pages[++i]).getRank() < (pages[right]).getRank());
				while(j > 0)
					if((pages[--j]).getRank() <= (pages[right]).getRank())
						break;
				if(i >= j) break;
				swap(pages,i,j);
			}
			swap(pages,i,right);
			sort(pages,left, i-1);
			sort(pages,i+1, right);
	}

	private static void swap(PageDataType[] pages, int a, int b) {
		PageDataType tmp = pages[a];
		pages[a] = pages[b];
		pages[b] = tmp;
```

```
 }

 private static int counter = 0;
 private static int framesOpen = 0;

 public static void show(CEASDataType[] args)  {
        PageDataType[] p = null;
        if (args[0] instanceof PageDataType) {
                p = new PageDataType[1];
                p[0] = (PageDataType) args[0];
        }
        else if (args[0] instanceof CEASList) {
                CEASList l = (CEASList) args[0];
                p = new PageDataType[l.length()];
                for (int i=0; i < l.length(); i++) {
                        p[i] = (PageDataType)l.getListElement(i);
                }
        }

        sort(p,0,p.length-1);
        URL[] fileURLs = new URL[p.length];
        for (int i=0;i<p.length;i++){
                p[i].setFile(Integer.toString(counter));
                counter++;
                if (!p[i].performExtraction()) {
                        throw new CEASException ("URL Problem");
                }
                fileURLs[i] = p[i].getFileURL();
        }
        try{

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

JFrame frame = new JFrame("CEAS Browser");

Container contentPane = frame.getContentPane();

contentPane.setLayout(new GridLayout(1, 1));
JTabbedPane tp = new JTabbedPane();



for (int i=0;i<p.length;i++){
 tp.add(new Browser(fileURLs[i]), p[i].getTitle());
}


contentPane.add(tp);


frame.addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent e) {
        framesOpen --;
        if(framesOpen == 0)
                System.exit(0);
 }
});
```

```
		frame.pack();
		frame.setVisible(true);
		framesOpen++;

			}
			catch (Exception e) {
				e.printStackTrace();
			}

		}

		public static void setTitle(CEASDataType[] args) {
			PageDataType p = (PageDataType) args[0];
			CEASString title = (CEASString) args[1];
			p.setTitle(title.value);
		}

		public static void setRank(CEASDataType[] args) {
			PageDataType p = (PageDataType) args[0];
			CEASInt rank = (CEASInt) args[1];
			p.setRank(rank.value);
		}

		public static CEASDataType savePage(CEASDataType[] args) {

			PageDataType p = (PageDataType) args[0];
			CEASString filename = (CEASString)args[1];
			p.setFile(Integer.toString(counter));
			counter++;
			if (p.performExtraction()) {
				if (p.saveToFile(filename.value)) {
					return CEASBool.TRUE;
				}
				else {
					return CEASBool.FALSE;
				}
			}
			else {
				throw new CEASException("URL Problem");
			}

		}

		public static void append(CEASDataType[] args) {
			PageDataType p = (PageDataType) args[0];
			CEASString key = (CEASString) args[1];
			p.setKeyword(key.value);
		}
}
```

File: ceas/types/PageDataType.java

```
package ceas.types;
```

```
import java.io.*;
import java.net.*;
import java.util.*;

import contentextractor.ContentExtractor;
import contentextractor.ContentExtractorConstants;
import contentextractor.ContentExtractorSettings;
import contentextractor.TypedProperties;


/**
 * This class represents the CEAS Page data type
 * @author Hila Becker
 *
 * $Id: PageDataType.java,v 1.2 2005/12/20 01:24:35 lra2103 Exp $
 */

public class PageDataType extends CEASDataType {

        /**
         * Extraction constants
         */
        public static final String EXTRACT_IMAGES = "IMAGES";
        public static final String EXTRACT_IMGLINKS = "IMGLINKS";
        public static final String EXTRACT_ADS = "ADS";
        public static final String EXTRACT_FLASH = "FLASH";
        public static final String EXTRACT_SCRIPTS = "SCRIPTS";
        public static final String EXTRACT_IFRAME = "IFRAME";
        public static final String EXTRACT_BUTTON = "BUTTON";
        public static final String EXTRACT_INPUT = "INPUT";
        public static final String EXTRACT_FORMS = "FORMS";
        public static final String EXTRACT_EMPTYTBLS = "EMPTYTBLS";
        public static final String EXTRACT_STYLES = "STYLES";
        public static final String EXTRACT_XTRNSTYLE = "XTRNSTYLE";
        public static final String EXTRACT_META = "META";
        public static final String EXTRACT_TXTLINKS = "TXTLINKS";
        public static final String EXTRACT_LINKLISTS = "LINKLISTS";



        private String url;
        private String title;
        private int rank;
        private ContentExtractorSettings mFilter;
        private String keyword = null;

        /*************************
         * Extraction Parameters   *
         *************************/
        private boolean ignoreAds = false;
        private boolean ignoreScripts = false;
        private boolean ignoreNoscript = false;
        private boolean  ignoreExternalStylesheets = false;
        private boolean ignoreStyles = false;
        private boolean ignoreStyleAttributes = false;
        private boolean ignoreStyleInDiv = false;
        private boolean ignoreImages = false;
```

```
private boolean displayAltTags = false;
private boolean ignoreImageLinks = false;
private boolean displayImageLinkAlts = false;
private boolean ignoreTextLinks = false;
private boolean ignoreForms = false;
private boolean ignoreInput = false;
private boolean ignoreButton = false;
private boolean ignoreSelect = false;
private boolean ignoreMeta = false;
private boolean ignoreIframe = false;
private boolean ignoreTableCellWidths = false;
private boolean ignoreEmbed = false;
private boolean ignoreFlash = false;

private boolean ignoreLinkLists = false;
private boolean ignoreLLTextLinks = false;
private boolean ignoreLLImageLinks = false;
private boolean ignoreOnlyTextAndLinks = false;
private double linkTextRatio = 0.5;

private boolean removeEmptyTables = false;
private boolean substanceImage = false;
private boolean substanceTextarea = false;
private boolean substanceLinks = false;
private boolean substanceButton = false;
private boolean substanceInput = false;
private boolean substanceForm = false;
private boolean substanceSelect = false;
private boolean substanceIFrame = false;
private int minimumTextLength = 3;

private boolean htmlOutput = true;
private boolean textOutput = false;
private boolean appendLinks = false;
private boolean limitLineBreaks = false;
private int         maxLineBreaks = 2;


private ContentExtractor ce;
private URL fileURL;
private boolean status = true;

File f = new File("test.html");

/**
 * Initialize a new page without a url
 */
public PageDataType(){

        url = "";
        mFilter = new ContentExtractorSettings();
        title = "untitled";
        rank = 0;
}

/**
 * Initialize a new page with a given url
```

```
 * @param url - the address of the page
 */
public PageDataType(String url) throws MalformedURLException {
	this();
	URL u = new URL(url);
	this.url = u.toString();
	initialized = true;
}

/**
 * Initialize a new page given a host and a path
 * @param host The host name, including how to reach it
 * @param path The path after the host
 */
public PageDataType(String host, String path) throws MalformedURLException {
	this();
	String url;
	if (!host.endsWith("/"))
		url = host + "/" + path;
	else
		url = host + path;
	//test make sure this is a legal URL
	URL u = new URL(url);
	this.url = u.toString();
	initialized = true;

}

/**
 * Creates a new PageDataType by copying the contents
 * of an input PageDataType
 * @param p the PagaDataType to copy
 */
public PageDataType(PageDataType p) {

	this.url = p.getURL();
	this.title = p.getTitle();
	this.rank = p.getRank();
	this.mFilter = p.getSettings();
	this.loadSettings();
	initialized = true;

}

/**
 * Set the file name that this page will be saved to for
 * display in the browser
 * @param fileName
 */
public void setFile(String fileName){
	this.f = new File(fileName + ".html");

}


/**
 * Sets the page title value
```

```
 * @param title the new title
 */
public void setTitle(String title){
	this.title = title;
}

/**
 * Sets the rank value of this page
 * @param rank the new rank value
 */
public void setRank(int rank){
	this.rank = rank;
}

/**
 * Sets the page URL value
 * @param url the new URL value
 */
public void setURL(String url){
	this.url = url;
}

/**
 * Sets the keyword for appending the next page
 * @param key
 */
public void setKeyword(String key){
	this.keyword = key;
}

/**
 * returns the keyword used to append the next page
 * @return
 */
public String getKeyword(){
	return keyword;
}

/**
 * returns the URL of this page
 * @return the URL
 */
public String getURL(){
	return url;
}

/**
 * returns the rank of this page
 * @return the rank
 */
public int getRank(){
	return rank;
}

/**
 * returns the title of this page
 * @return the title
```

```
 */
public String getTitle(){
	return title;
}

/**
 * returns the status of this page
 * @return true if status is OK
 */
public boolean getStatus(){
	return status;
}

/**
 * returns the extraction settings of this page
 * @return extraction settings
 */
public ContentExtractorSettings getSettings(){
	return mFilter;
}

/**
 * returns the InputStream generated
 * @return
 */
private InputStream getStream() throws MalformedURLException, IOException {
	URL u = new URL(url);
	URLConnection uc = u.openConnection();
	System.out.println(uc.getHeaderField(0));
	InputStream in = uc.getInputStream();
	return in;
}

/**
 * Call the content extractor, extract content of this page
 * according to the specified page settings and save the results
 * (html) to a temporary file
 */
public boolean performExtraction() {

	try{
		this.commitSettings();
		ce = new ContentExtractor(getStream());
		ce.setAddress(this.url);
		ce.setKeyword(this.keyword);
		ce.setSettings(this.mFilter);
		f.createNewFile();
		f = ce.process(f);

		String temp;
		if ((temp = ce.getNewURL()) != null){
			System.out.println("found new url");
			this.url = temp;
			System.out.println(url);
			ce = new ContentExtractor(getStream());
			ce.setAddress(this.url);
			ce.setSettings(this.mFilter);
```

```
                        File nf = new File("toAppend.html");
                        nf.createNewFile();
                        nf = ce.process(nf);
                        f = ce.appendDocument(nf,f);
                }
                fileURL = f.toURL();
        }
        catch(Exception e){

                status = false;
        }
        return status;
}

/**
 * The URL (file path) of the file where the extraction results are saved
 * @return the file path
 */
public URL getFileURL(){

        return fileURL;
}

/**
 * Saves the HTML of this page the a file
 * @param fileName the file name to save to
 */
public boolean saveToFile(String fileName){

        File newFile = new File(fileName);
        try{
                copy(f,newFile);
        }
        catch(IOException ioe){
                System.err.println("Could not save to file: " + fileName);
                return false;
        }
        return true;
}

/**
 * copies contents of source file to destination file
 * @param src source file
 * @param dst destination file
 * @throws IOException
 */
private void copy(File src, File dst) throws IOException {
        InputStream in = new FileInputStream(src);
        OutputStream out = new FileOutputStream(dst);
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
                out.write(buf, 0, len);
        }
        in.close();
        out.close();
        }
```

```
/**
 * originally used to append 2 pages.
 * @param first
 * @param second
 * @return
 * @throws IOException
 */
private File append(File first, File second) throws IOException{


	InputStream in2 = new FileInputStream(second);
	File dst = new File("temp.html");
	copy(first,dst);
	OutputStream out = new FileOutputStream(first);
	InputStream in1 = new FileInputStream(dst);
	byte[] buf = new byte[1024];
	int len;
	while ((len = in1.read(buf)) > 0) {
		out.write(buf, 0, len);
	}
	while ((len = in2.read(buf)) > 0) {
		out.write(buf, 0, len);
	}
	in1.close();
	in2.close();
	out.close();

	return dst;
}

/**
 * Turn on extraction parameters specified by the user
 * @param features the list of extraction parameters
 */
public void extract(String[] features){

	String feature;
	for (int i=0;i<features.length;i++){
		feature = features[i];

		if (feature.equalsIgnoreCase(EXTRACT_IMAGES)){
			ignoreImages = true;
		}
		else if (feature.equalsIgnoreCase(EXTRACT_ADS)){
			ignoreAds = true;
		}
		else if (feature.equalsIgnoreCase(EXTRACT_FLASH)){
			ignoreFlash = true;
		}
		else if (feature.equalsIgnoreCase(EXTRACT_SCRIPTS)){
			ignoreScripts = true;
		}
		else if (feature.equalsIgnoreCase(EXTRACT_TXTLINKS)){
			ignoreTextLinks = true;
		}
		else if (feature.equalsIgnoreCase(EXTRACT_IMGLINKS)){
```

```
                        ignoreImageLinks = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_XTRNSTYLE)){
                        ignoreExternalStylesheets = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_STYLES)){
                        ignoreStyles = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_FORMS)){
                        ignoreForms = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_LINKLISTS)){
                        ignoreLinkLists = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_EMPTYTBLS)){
                        removeEmptyTables = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_INPUT)){
                        ignoreInput = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_META)){
                        ignoreMeta = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_BUTTON)){
                        ignoreButton = true;
                    }
                    else if (feature.equalsIgnoreCase(EXTRACT_IFRAME)){
                        ignoreIframe = true;
                    }
                    else{
                        //shouldn't happen, throw an error when debugging.
                        System.out.println("error, invalid feature");
                    }



                }

        }

        /**
         * Updates the ContentExtractorSettings to reflect the Page's specifications.
         */
        public void commitSettings() {

                mFilter.changeSetting(ContentExtractorConstants.ONLY_TEXT, Boolean.toString(textOutput));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_ADS,
Boolean.toString(ignoreAds));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_BUTTON_TAGS,
Boolean.toString(ignoreButton));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_FORMS,
Boolean.toString(ignoreForms));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_IFRAME_TAGS,
Boolean.toString(ignoreIframe));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_IMAGE_LINKS,
Boolean.toString(ignoreImageLinks));
                if (ignoreImageLinks)
```

```
                        mFilter.changeSetting(ContentExtractorConstants.DISPLAY_IMAGE_LINK_ALTS,
Boolean.toString(displayImageLinkAlts));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_TEXT_LINKS,
Boolean.toString(ignoreTextLinks));

                if (ignoreImages) {
                        mFilter.changeSetting(ContentExtractorConstants.DISPLAY_IMAGE_ALTS,
Boolean.toString(displayAltTags));
                        mFilter.changeSetting(ContentExtractorConstants.IGNORE_IMAGES,
Boolean.toString(ignoreImages));
                } else
                        mFilter.changeSetting(ContentExtractorConstants.IGNORE_IMAGES,
Boolean.toString(ignoreImages));

                mFilter.changeSetting(ContentExtractorConstants.IGNORE_INPUT_TAGS,
Boolean.toString(ignoreInput));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_LINK_CELLS,
Boolean.toString(ignoreLinkLists));

                if (ignoreLinkLists) {
                        mFilter.changeSetting(ContentExtractorConstants.LC_IGNORE_IMAGE_LINKS,
Boolean.toString(ignoreLLImageLinks));
                        mFilter.changeSetting(ContentExtractorConstants.LC_IGNORE_TEXT_LINKS,
Boolean.toString(ignoreLLTextLinks));
                        mFilter.changeSetting(ContentExtractorConstants.LINK_TEXT_REMOVAL_RATIO,
Double.toString(linkTextRatio));
                        mFilter.changeSetting(ContentExtractorConstants.LC_ONLY_LINKS_AND_TEXT,
Boolean.toString(ignoreOnlyTextAndLinks));
                }

                mFilter.changeSetting(ContentExtractorConstants.IGNORE_META,
Boolean.toString(ignoreMeta));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_SCRIPTS,
Boolean.toString(ignoreScripts));

                if (ignoreNoscript)
                        mFilter.changeSetting(ContentExtractorConstants.IGNORE_NOSCRIPT_TAGS,
Boolean.toString(ignoreNoscript));

                mFilter.changeSetting(ContentExtractorConstants.IGNORE_SELECT_TAGS,
Boolean.toString(ignoreSelect));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_EXTERNAL_STYLESHEETS,
Boolean.toString(ignoreExternalStylesheets));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_DIV_STYLES,
Boolean.toString(ignoreStyleInDiv));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_STYLE_ATTRIBUTES,
Boolean.toString(ignoreStyleAttributes));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_STYLES,
Boolean.toString(ignoreStyles));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_CELL_WIDTH,
Boolean.toString(ignoreTableCellWidths));

                mFilter.changeSetting(ContentExtractorConstants.REMOVE_EMPTY_TABLES,
Boolean.toString(removeEmptyTables));
                if (removeEmptyTables) {
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_BUTTON,
Boolean.toString(substanceButton));
```

```
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_FORM,
Boolean.toString(substanceForm));
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_IFRAME,
Boolean.toString(substanceIFrame));
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_IMAGE,
Boolean.toString(substanceImage));
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_INPUT,
Boolean.toString(substanceInput));
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_LINKS,
Boolean.toString(substanceLinks));
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_SELECT,
Boolean.toString(substanceSelect));
                        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_TEXTAREA,
Boolean.toString(substanceTextarea));

        mFilter.changeSetting(ContentExtractorConstants.SUBSTANCE_MIN_TEXT_LENGTH,
Integer.toString(minimumTextLength));
                } //if


                mFilter.changeSetting(ContentExtractorConstants.IGNORE_EMBED_TAGS,
Boolean.toString(ignoreEmbed));
                mFilter.changeSetting(ContentExtractorConstants.IGNORE_FLASH,
Boolean.toString(ignoreFlash));
                mFilter.changeSetting(ContentExtractorConstants.ADD_LINKS_TO_BOTTOM,
Boolean.toString(appendLinks));
                mFilter.changeSetting(ContentExtractorConstants.LIMIT_LINEBREAKS,
Boolean.toString(limitLineBreaks));
                if (limitLineBreaks)
                        mFilter.changeSetting(ContentExtractorConstants.MAX_LINEBREAKS,
Integer.toString(maxLineBreaks));



                 mFilter.save(ContentExtractor.CUSTOM_SETTINGS_FILE_DEF);


        }

        /**
         * Returns the name of the datatype
         */
        public String typename() {
                return PAGE_TYPE;
        }


        public void loadSettings(){

                ignoreAds = (mFilter.getSetting(ContentExtractorConstants.IGNORE_ADS).equals("true"));
                ignoreScripts =
(mFilter.getSetting(ContentExtractorConstants.IGNORE_SCRIPTS).equals("true"));

        ignoreNoscript=(mFilter.getSetting(ContentExtractorConstants.IGNORE_NOSCRIPT_TAGS).equals("tr
ue"));
```

```
        ignoreExternalStylesheets=(mFilter.getSetting(ContentExtractorConstants.IGNORE_EXTERNAL_STY
LESHEETS).equals("true"));

        ignoreStyles=(mFilter.getSetting(ContentExtractorConstants.IGNORE_STYLES).equals("true"));

        ignoreStyleAttributes=(mFilter.getSetting(ContentExtractorConstants.IGNORE_STYLE_ATTRIBUTES).
equals("true"));

        ignoreStyleInDiv=(mFilter.getSetting(ContentExtractorConstants.IGNORE_DIV_STYLES).equals("true"
));
            ignoreImages=(mFilter.getSetting(ContentExtractorConstants.IGNORE_IMAGES).equals("true"));
                displayAltTags =
(mFilter.getSetting(ContentExtractorConstants.DISPLAY_IMAGE_ALTS).equals("true"));

        ignoreImageLinks=(mFilter.getSetting(ContentExtractorConstants.IGNORE_IMAGE_LINKS).equals("tr
ue"));

        displayImageLinkAlts=(mFilter.getSetting(ContentExtractorConstants.DISPLAY_IMAGE_LINK_ALTS).
equals("true"));

        ignoreTextLinks=(mFilter.getSetting(ContentExtractorConstants.IGNORE_TEXT_LINKS).equals("true")
);

        ignoreForms=(mFilter.getSetting(ContentExtractorConstants.IGNORE_FORMS).equals("true"));

        ignoreInput=(mFilter.getSetting(ContentExtractorConstants.IGNORE_INPUT_TAGS).equals("true"));

        ignoreButton=(mFilter.getSetting(ContentExtractorConstants.IGNORE_BUTTON_TAGS).equals("true")
);

        ignoreSelect=(mFilter.getSetting(ContentExtractorConstants.IGNORE_SELECT_TAGS).equals("true"))
;
                ignoreMeta=(mFilter.getSetting(ContentExtractorConstants.IGNORE_META).equals("true"));

        ignoreIframe=(mFilter.getSetting(ContentExtractorConstants.IGNORE_IFRAME_TAGS).equals("true"))
;

        ignoreTableCellWidths=(mFilter.getSetting(ContentExtractorConstants.IGNORE_CELL_WIDTH).equal
s("true"));

        ignoreEmbed=(mFilter.getSetting(ContentExtractorConstants.IGNORE_EMBED_TAGS).equals("true"))
;
                ignoreFlash=(mFilter.getSetting(ContentExtractorConstants.IGNORE_FLASH).equals("true"));

        ignoreLinkLists=(mFilter.getSetting(ContentExtractorConstants.IGNORE_LINK_CELLS).equals("true"))
;


        ignoreLLTextLinks=(mFilter.getSetting(ContentExtractorConstants.LC_IGNORE_TEXT_LINKS).equals
("true"));

        ignoreLLImageLinks=(mFilter.getSetting(ContentExtractorConstants.LC_IGNORE_IMAGE_LINKS).equ
als("true"));

        ignoreOnlyTextAndLinks=(mFilter.getSetting(ContentExtractorConstants.LC_ONLY_LINKS_AND_TEX
T).equals("true"));
```

```
		linkTextRatio=Double.parseDouble(mFilter.getSetting(ContentExtractorConstants.LINK_TEXT_REMO
VAL_RATIO));

		removeEmptyTables=(mFilter.getSetting(ContentExtractorConstants.REMOVE_EMPTY_TABLES).equ
als("true"));

		substanceImage=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_IMAGE).equals("true"))
;

		substanceTextarea=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_TEXTAREA).equals(
"true"));

		substanceLinks=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_LINKS).equals("true"));

		substanceButton=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_BUTTON).equals("true
"));

		substanceInput=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_INPUT).equals("true"));

		substanceForm=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_FORM).equals("true"));

		substanceSelect=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_SELECT).equals("true"
));

		substanceIFrame=(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_IFRAME).equals("true
"));
			minimumTextLength =
Integer.parseInt(mFilter.getSetting(ContentExtractorConstants.SUBSTANCE_MIN_TEXT_LENGTH));
	htmlOutput= (!mFilter.getSetting(ContentExtractorConstants.ONLY_TEXT).equals("true"));
			textOutput=(mFilter.getSetting(ContentExtractorConstants.ONLY_TEXT).equals("true"));

		appendLinks=(mFilter.getSetting(ContentExtractorConstants.ADD_LINKS_TO_BOTTOM).equals("true"
));

		limitLineBreaks=(mFilter.getSetting(ContentExtractorConstants.LIMIT_LINEBREAKS).equals("true"));

		maxLineBreaks=Integer.parseInt(mFilter.getSetting(ContentExtractorConstants.MAX_LINEBREAKS));

		}

		/**
		 * Returns a copy of this object.
		 */
		public CEASDataType _copy() {
			//return new PageDataType(this);
			//pass by reference
			return this; //copy is used to prepare for function calls
		}

		public String toString() {
			return (url);
		}



}
```

Appendix B: Supporting Libraries

In addition to the code found in Appendix A, the CEAS project made use of a variety of available libraries. Most of these were simply included as jar files. In some cases, as with Crunch and the Java browser, the code was slightly modified to suit our needs. The following is a list of the supporting libraries:

Name	File	Usage	URL
JDesktop: Integration Components	jdic.jar	Java Browser	http://javadesktop.org/articles/jdic/index.html
NekoHTML	nekohtml.jar	HTML Parsing	http://java-source.net/open-source/html-parsers/nekohtml
	nekohtmlXni.jar		
Xerces Java Parser	xercesImpl.jar	XML Parsing	http://xerces.apache.org/xerces-j/
	xmlParserAPIs.jar		