Pre-compiled Server Pages (PSP) Language Report

O'Neil Palmer
op2007@columbia.edu

Table of Contents

Chapter 1

Introduction

The PSP (Pre-compiled Server Pages) programming language is designed to provide developers of dynamic web content with the debugging output common to compiled languages within the structure of scripting languages popular for web programming. A PSP compiler will give debugging output against PSP code until the code fits its simple, yet strict syntax. Properly written PSP code will be compiled into guaranteed syntax-perfect JSP pages.

The strengths of the PSP language will include:

- Verbose debugging output
- Separate dynamic rendering code from static html from.
- Java functionality
- Algorithmic things, like control structures!
- Simplified Java syntax

Background

JavaServer Pages (JSP) allows you to create dynamic web pages. In its basic form JSP pages are HTML pages with special embedded tags that execute application logic to generate dynamic content. JSP are an extension to the Java Servlet API. A JSP file is compiled into a Java Servlet, which executes in a Servlet engine on the application server. Therefore, JSP pages have full Java functionality and have all the benefits of a typical Java program.

One of the pitfalls new JSP developers run across is that they don't spend much time on design and they soon end up with a "fat" JSP that contains both presentation logic and business logic. They soon realize that there JSP files, because of the mixture of HTML syntax and JSP tags, have become unreadable. By separating out the application logic for rendering dynamic content into a separate file we can maintain the purity of the HTML source and avoid a point of confusion for application logic developers.

The Advantage of Pre-compilation

With any current web programming language, such as Perl, PHP, or JSP, no code is parsed until a page is viewed. If there is a syntax error in that code, nothing will be displayed - because the program didn't parse correctly, there is ZERO OUTPUT! With no debugging output, it is the programmer's responsibility to read through his or her entire program to find a subtle missing semi-colon or closing parenthesis. Instead of wasting time and energy like this, a programmer could write his code is PSP, and use the PSP compiler to generate syntax-perfect JSP pages.

The PSP compiler will first check for proper PSP syntax. If there are mistakes, debugging output will specify what the mistakes are and which line they occur on. If syntax is correct, the PSP compiler will generate a JSP code file with perfect JSP syntax, from bits as simple as semi-colons and braces to structures as complex as for-loops and if-else statements - and of course the evasive <% and %> tags.

Java Functionality

Because PSP compiles into JSP pages, PSP offers much of the functionality of the Java language - dynamic content from 'request' and 'session', complex control structures, HTML user interfaces, and platform independence. With PSP, the powerful features of JSP are wrapped in an extremely convenient programming environment.

But PSP will not offer FULL Java functionality. PSP will force good programming habits by leaving the most powerful processing jobs to be handled by back-end server code. The PSP language is focused on providing a user interface with dynamic information gleaned from more powerful back-end software.

Design

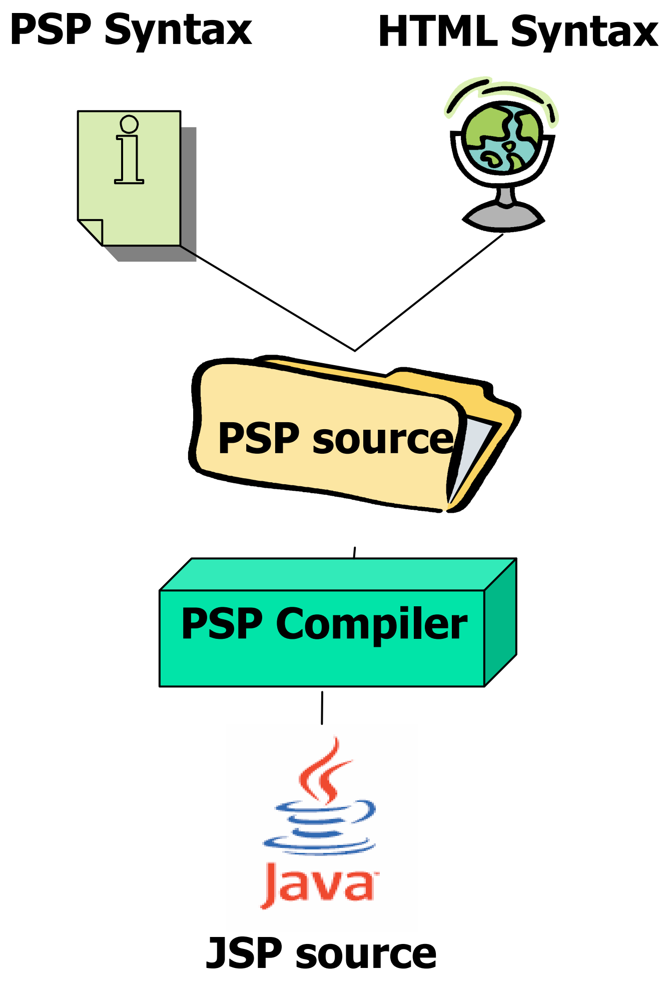

Fig1: Design overview of the PSP compiler

The PSP compiler is designed with the intention of separating the code for rendering dynamic content from the static content that is used as the baseline. It does so by reconciling a dynamic .psp and a static .html file, and combining the two into a .jsp file. As such, the compiler recognizes both PSP source files and HTML.

The compiler first parses the PSP code and compiles it then uses the PSP code to add dynamic content to the HTML file. After which, the resulting file is a JSP. If there is an error at any of these steps, the compiler exits and reports the errors in a context-specific, user-friendly manner. Error reporting will include information as to whether it was the .psp file or the linking step that generated the error, as well as line number(s) of erroneous code, and as much detail about what caused the error as possible.

PSP syntax is designed to be as convenient and familiar as possible. It is readily understandable and readily programmable by a Java developer.

Basic Syntax

Below is an overview of the basic syntax for the PSP language. The syntax is intended to be very similar to Java syntax as that is typically a required skill set for your average JSP programmer. The following syntax gives you an idea of how the language is used to create dynamic content. The PSP Syntax is highlighted in red and the HTML Syntax in purple.

```
                    HelloWorld.psp
<$
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>
<head><title>Hello World</title></head>

<body bgcolor=#FFFFFF>
<font face="Helvetica">

<h2><font color=#DB1260> Hello World </font> </h2>
$>

print("<p><b>Hello World!</b>");

<$
<hr><p></font></body></html>
$>
```

The HelloWorld.psp file will generate the popular HelloWorld.jsp source file.

```
                    HelloWorld.jsp
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>
<head><title>Hello World</title></head>

<body bgcolor=#FFFFFF>
<font face="Helvetica">

<h2><font color=#DB1260> Hello World </font> </h2>

<% out.print("<p><b>Hello World!</b>"); %>

<hr><p></font></body></html>
```

Limitations

The target audience for the PSP language is novice JSP developers. Currently, the language only provides basic functionality for generating dynamic web content. HTML and JSP are both extensive languages therefore we will only focus on a narrow area, forms, input fields and tables.

Conclusion

The goal of the PSP language is to remedy the failings of other dynamic web scripting languages. Existing languages primarily fail due to difficulties in error detection. As a result a considerable amount of time is spent debugging. We have outlined a design that should create a valuable compiler that addresses some of the shortcoming on current technologies used to render dynamic content on the web. We feel that the separation of the code for rendering dynamic and the static portions of web documents will make PSP an attractive alternative to web developers. The power of PSP lies in its ability to simplify the overall development effort by generating JSP that are guaranteed to compile. While its power is only in its infancy PSP shows a great deal of promise in terms of the solutions it may one day solve.

Chapter 2

PSP Tutorial

Installation and Setup

The makefile distributed with the PSP source code makes it simple to build the PSPCompiler and begin compiling .psp source files.

The PSPMain.java file is the main program for the PSP compiler. PSP source files, which must contain the suffix .psp, can be compiled by passing the filename as an argument to the PSPMain program. i.e.

```
$: java PSPMain ./login.psp ./logout.psp
```

Running the following make command will compile all the .psp source files in the PSP source root directory.

```
$: make run
```

Please see the Makefile for details and a list of other options available like **make distclean**, which deletes all the .class files and auto generated .java files.

Writing a PSP Program

PSP developers should review the PSP language reference manual for detail on the PSP syntax. However because the PSP syntax is very similar to standard Java syntax, most Java developer will be able to ramp up very quickly.

Because the purpose of PSP is to generate dynamic web content the **print()** method is one of the most common PSP commands. The following shows a simple example:

```
                         HelloWorld.psp
<$
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>
<head><title>Hello World</title></head>

<body bgcolor=#FFFFFF>
<font face="Helvetica">

<h2><font color=#DB1260> Hello World </font> </h2>
$>

print("<p><b>Hello World!</b>");

<$
<hr><p></font></body></html>
$>
```

The following is a more advance example. The DisplayInfo.psp file retrieves the username variable from the HTTP session and display that value on the page. It also retrieves a number of parameters from the HTTP request and displays the information in a table.

```
                           DisplayInfo.psp

String userName = session userName;

<$
<html><head> <title>displayInfo</title> </head>
<body bgcolor=#FFFFFF><font face="Helvetica"> <h2> Personal Information:$>
print(userName);
<$</h2>$>
String fname = request fname;
String lname = request lname;
String dob = request dob;
String ssn = request ssn;
String state = request state;
<$
<table> <tr> <td>Key</td> <td>Value</td> </tr>
      <tr> <td>$>print("fname");<$</td> <td>$>print(fname);<$</td></tr>$>
      <$<tr> <td>$>print("lname");<$</td> <td>$>print(lname);<$</td> </tr>$>
      <$<tr> <td>$>print("dob");<$</td> <td>$>print(dob);<$</td> </tr>$>
      <$<tr> <td>$>print("ssn");<$</td> <td>$>print(ssn);<$</td> </tr>$>
      <$<tr> <td>$>print("state");<$</td> <td>$>print(state);<$</td> </tr>
</table>
$>
<$ <a href="/psp/welcome.jsp"><h3>Home</h3></a> <hr></font></body></html> $>
```

The best practice is to first generate the html content and then convert the .html file to a .psp file.

Chapter 3

1 – Language Reference Manual

PSP, short for Precompiled Server Pages, was created as a front end programming language that designs dynamic .jsp pages intended for presentation on the World Wide Web. It is a simple language intended for the presentation of dynamic information, with a small keyword set and language definition to make it the easiest and best possible language for its intended use. PSP code will compile into .jsp pages.

1.1 'Hello World' Example

To get a taste of the language, here are examples of PSP programs that will compile to .jsp pages that output "Hello World".

```
/* example1.psp, a simple PSP print statement */
print( "Hello World");

/* example2.psp, an example of the special <$…$> print syntax used to
output html */
<$ Hello world! $>

/* example3.psp, an example of variable use in PSP */
String hi = "Hello World";
print( hi );
```

2 – Overview of Lexical Conventions

The PSP language is defined through its keywords, variable types, identifiers, operators, and basic syntax.

2.1 Keywords

The following words are reserved for use by the PSP language:
<$ $>, if, else, request, while, for, int, String, Collection, print, session, iterate

2.3 Identifiers

An identifier is a name that the PSP language uses to represent a piece of data. The user builds his or her identifiers according to the following rules:

- Identifiers can only begin with a letter or an underscore.
- Identifiers may only consist of letters, under scores, or digits.

Hence the following are valid identifiers: ***foobar, Edwards, Moosejaw_Saskatchewan***
and the following are invalid: ***95_Cowboys, boston- red- sox, WhizBang!***

2.2 Variable Types

Variables are identified by identifier names. To instantiate them an integer name is declared with a data type and assigned a value with the = assignment operator. PSP supports variables to represent three different varieties of data: int, String, and Collection.

int
An **int** represents a mathematical integer in the range between -32,768 and 32,767. The grammar to define an integer variable is:

int ***identifier*** *; /* declaration */*
identifier *= 7 ; /* integer assignment statement. */*

String
A String is a series of characters. The grammar to define a String is:

String ***identifier;***

Collection
A Collection is a set of String objects. A collection may only be assigned values using the PSP keyword 'session'. Note that a Collection object MUST be declared and initialized in the same statement. To define a collection:

Collection ***identifier; /* Invalid syntax. Will result in an error. */***
Collection ***identifier*** *= session attrname;*

Please note that declaration and assignment for all the variables can be done in one statement, i.e. *int x = 7/* sets the variable x to 7 */*

2.4 Operators

The PSP language supports the following operators, which will be elaborated upon in the next section: &&, ||, +, ++, =, = =, <, <=, >, >=, -, - -

3 – Operators

The following operator s have special significance in PSP, and are part of either the first, second, third, or fourth level of precedence, with the first level being evaluated first, the second being evaluated second, and finally the third and then fourth.

3.1 Addition / Concatenation '+'

The + operator combines the values on its left and right side and returns them to the left. In the case of integers, the operation is addition. If the + operator is used on strings, the result is concatenation.

int x = 4 + 7; /* x is 11 */
String name = "Jeremy" + "Smith"; /* name is JeremySmith */

3.2 Subtraction '-'

The – operator subtracts the value on its left from the value on its right and returns it to the left

x = 7 – 4; /* x is 3 */

3.3 Increment '++'

The ++ operator increases the value of an integer variable touching it by 1.
x = 7; /* x is 7 */
x++; /* x is now 8 */
++x; **/* Invalid statement. ++ operator must be the suffix on an identifier.*/**

3.4 Decrement '--'

The - - operator decreases the value of an integer variable touching it by 1.
x = 7; /* x is 7 */
x- - ; /* x is now 6 */
--x; **/* Invalid statement. -- operator must be the suffix on an identifier.*/**

3.5 Assignment '='

The = operator gives the variable on its left the value on its right. It may only be used on int and String variables.

```
int y = 45000000000000000; /* value of y is 45000000000000000 */
String city = "Dallas"; /* value of city is Dallas */
y = y + 1; /* y is now 45000000000000001 */
```

3.6 Less than '<'

The < operator returns a Boolean true if the value on its left side is less than the value on its right side, otherwise it returns false. It may only be used on integers.

```
if( 4 < 7 ) {
  print( "true!" ); /* prints 'true!' */
}
```

3.7 Less than or equal to '<='

The <= operator returns a Boolean true if the value on its left side is less than or equal to the value on its right side, otherwise it returns false. It may only be used on integers.

```
if( 4 <= 4 ) {
  print( "true!" ); /* prints 'true!' */
}
```

3.8 Greater than '>'

The > operator returns a Boolean true if the value on its left side is greater than the value on its right side, otherwise it returns false. It may only be used on integers.

```
if( 4 > 4 ) {
   print( "true!" ); /* doesn't print anything' */
}
```

3.9 Greater than or equal to '>='

The >= operator returns a Boolean true if the value on its right side is greater than the value on its left side, otherwise it returns false. It may only be used on integers.

```
if( 4 >= 2 ) {
  print( "true!" ); /* prints 'true!' */
}
```

3.10 Equality test ‘==’

The = = operator returns a boolean true if the value on its right is the same as the value on its left, and false otherwise. It will work for int and String variables.

```
if( 9 = = 9 ) {
  print( “Sure enough, 9 is 9.” ); /* prints “Sure enough, 9 is 9.”*/
}
```

3.11 Logical AND ‘&&’

The && operator will return true if the values on each side are Boolean true values.

```
if( 4 > 1 && 2 == 2 ) {
  print( “It’s all true.” ); /* prints “It’s all true.” */
}
```

3.12 Logical OR ‘||’

The || operator will return true if a value on either side is Boolean true value.

```
if( 4 > 888234 && 2 == 2 ) {
  print( “Fine by me.” ); /* prints “Fine by me.” */
}
```

3.13 Multiplication ‘*’

The * operator will return the product of values on either side. It only works on integers.

```
int x = 5 * 6; /* x is 30 */
```

3.14 Division ‘/’

The / operater will return the value of the number on its left divided by the value of the number on its right, truncated.

```
x = 16 / 7; /* x is 2 */
```

4 – Statements

The following statements are used for various tasks not provided by the operators: controlling the flow, retrieving Java Bean information, and printing variables and HTML. In this section they will be described in detail.

4.1 if- else

The if- else will is used to evaluate code under specific conditions. A Boolean expression is given to the ‘if’ statement, and if it evaluates to true then the code within the ‘if’ statement is executed. The else is optional and will execute its code if the if statement is false.

```
if( 5 == 5 ) {
  x = 17; /* x is now 17 */
}
if( 17 < 4 ) {
  print( “17 greater than 4? What?”);
} else {
  print(“Or else what?”); /* prints “Or else what?” */
}
```

4.2 while loop

The while loop will execute the code within its braces for as long as its test condition remains true.

```
int x = 6
while( x < 10 ) {
  print( x + “ “ ); /* prints 6 7 8 9 */
  x++;
}
```

4.3 for loop

The for loop will run a set number of times according to a Boolean expression. It contains three statement s separated by semi - colons before and code to be executed within braces. The first statement initializes a variable. The second is a Boolean condition the loop tests against every iteration. The third is some sort if incrementation, executed every iteration of the loop.

```
for( int i = 0; i < 10; i++ ) {
print( “An iteration.”); /* prints “An iteration.” 10 times */
}
```

4.4 session

A session statement accesses the Java Bean and returns a String or a Collection of information in the desired attribute for storage into a Collection.

```
session attribute ;
/* See the iterate section to see how to use a Collection.  */
Collection states = session stateList;

/* Will set the value of the attribute "username" to "O'Neil" */
session userName = "ONeil";

/* Will set the value of the attribute "username" to the value of the String variable.
    This assumes the variable has been previously defined.
*/
session userName = name;
```

4.5 request

Request is used to access and alter GET/POST data in a page.

```
request attribute ;
String name = request userName;
```

4.6 print

The print statement simply prints strings – String variables and string constant s – to the standard output stream (the web page).

```
String name = "Cody";
print("Hello, my name is " + name );  /* prints "Hello, my name is Cody." */
```

4.7 <$...$>

This special print operator will write everything within itself to the .jsp page literally as HTML code. **Note: A HTML block is simply another statement so the can be nested within PSP syntax.**

```
/* a complete HTML header for a PSP page */
<$
   <html><head>   <title>PSP Hello World Page</ title>
   </head >
$>
```

```
/* will display "PSP HelloWorld Page" or "Hello [some name]" in the html title */

String name = request userName;
if(null(name)){
   <$ <html><head> <title>PSP Hello World Page</ title> </head > $>
}else{
      <$ <html><head> <title>Hello $>  print(name);
      <$ Page </title>  </head >  $>
}
```

4.8 iterate

The iterate statement loops through a Collection. The default value for a Collection value within an iteration is 'name'.

```
iterate(states) {
  print("<TR>");
  print(next);
  print("</TR>");
}
```

4.9 null

The null statement gives psp developers a way to validate that their String variables are null. The null statement simply evaluates out to an if statement where the condition is 'id = = null', i.e. if(id = = null){ }

```
    //null as a statement              //null as an expression
    String id;                         String id;

    null(id) {                         while(null(id)) {
      print("<TR>");                     print("<TR>");
      print("null");                     print("null");
      print("</TR>");                    print("</TR>");
    }                                  }
```

4.10 notnull

The notnull statement gives psp developers a way to validate that their String variables are not null. The notnull statement simply evaluates out to an if statement where the condition is ‘id != null’, i.e. if(id != null){ }

```
//notnull as a statement        //notnull as an expression
String id;                      String id;

notnull(id) {                   while(notnull(id)) {
  print(“<TR>”);                  print(“<TR>”);
  print(id);                      print(id);
  print(“</TR>”);                 print(“</TR>”);
}                               }
```

4.11 ++

The ++ increment operator can also be used as a statement.

```
int i = 7;
i++; /* Increments the value of int i by 1.*/
```

4.12 --

The – decrement operator can also be used as a statement.

```
int i = 7;
i--; /* Decrements the value of int i by 1.*/
```

4.13 equals

The equals statement gives psp developers a way to compare two String variables. The equals statement simply evaluates out to an if statement where the condition is ‘equals(id,id2)’, i.e. if(id.equals(id2)){ }

```
//equals as a statement          //equals as an expression
String id;                       String id;
String id2;                      String id2;

equals(id,id2) {                 while(equals(id,id2)) {
  print("<TR>");                   print("<TR>");
  print(id);                       print(id);
  print("</TR>");                  print("</TR>");
}                                }
```

4.14 nequals

The nequals statement gives psp developers a way to compare two String variables. The nequals statement simply evaluates out to an if statement where the condition is ‘nequals(id,id2)’, i.e. if(! id.equals(id2)){ }

```
//equals as a statement          //equals as an expression
String id;                       String id;
String id2;                      String id2;

nequals(id,id2) {                while(nequals(id,id2)) {
  print("<TR>");                   print("<TR>");
  print(id);                       print(id);
  print("</TR>");                  print("</TR>");
}                                }
```

5 – Syntax

To compose a PSP program you must use proper syntax to delineate blocks of code, signify the end of statement s, and comment your work. The syntax should be obvious, as it is modeled after that of Java and C to accommodate programmer s making the switch.

Comments

Comments are ignored by the compiler. PSP supports both single and multi-line comments. Multi-line comments cannot be nested.

```
//Single line comment begin with double slashes
/* Everything that is between these tokens is a comment. */
```

End of Line

The end of a statement must be signified with a semicolon.

```
String lever = “2000”; /* note the semicolon */
```

Scope

PSP follows the normal java scope rules. Blocks of code are identified with curly braces: { and }. For loops, while loops, and if statements use this to lay claim to their block of code. PSP will throw a Duplicate variable name exception, similar to Java, if identifiers are declared more that once in a given scope.

```
if( TRUE ) {
  /* some code */
}
```

Chapter 4

Project Plan and TimeLine

The initial plan was for the team to meet every Monday @ 6:30pm. The following is the latest timeline that we had as a team.

Task	Due Date
Complete White Paper	9/28
Complete LRM	10/21
Complete Lexer and Parser	10/21
Complete Testing	12/21
Complete Documentation	12/12

Team Roles and Responsibilities

Member	Resonsibility
Olajide Atoyebi	Documentation and testing
Cody Hess	Documentation and testing
O'Neil Palmer	Team Leader, Development and testing
Neil Sarkar	Documentation and testing

Software Environment

The PSP language was developed on top of Suns Java 4 using ATLR 2.7.4. ANTLR was used to write the Lexer, Parser and Tree walker for the PSP Language. The other components of the application was written in Java 1.4. IBM' Websphere Studio Application Developer was used as the java development environment. WSAD was chose because it is a robust toolkit and was suitable for end to end development and testing. Test cases for PSP were written using the JUnit 3.8 testing framework. At first glace this may seem like overkill but it greatly simplified the testing process and allowed a smaller granularity of testing and debuging. Using JUnit allowed me to eliminate some 30+ (and growing) test .psp source files, combining them into a handful of JUnit test classes.

Project Log

Accomplishment	Due Date
First team meeting – Discussed PSP idea and another graphical block drawing language.	9/20
Met with professor to discuss idea. Decided on PSP as a result.	9/21
Completed White Paper	9/28
Started Lexer and Parser development and writing LRM	10/11
Completed LRM	10/21
Setup CVS for local development environment	11/07
Group split up like the Jackson 5 and I went solo like MJ.	11/23
Completed Lexer and Parser	11/27
Completed TreeWalker	12/4
Complete Testing	12/21
Complete Documentation	12/12

Programming Style Guide

The following is a list of coding standards, conventions, and guidelines used during the development on the PSP language.

Naming Conventions

- Variable Names: should clearly reflect the purpose of the variable. Using accronyms is not reccomended unless they are commonly know acronyms. Variable names should begin with a lowercase letter. Mixed case letters should be used to make identifier names with multiple words. Mixed case letters will make your identifiers more readable. i.e.
  int **nuberOfRecords**; String **userName**;

- Constant Names: should be in all upper case letters. Underscores '_' should be used to separate multiple words. i.e.
  public static final int **LIMIT** = 500;
  public static final String **PSP_START_TAG** = 500;

- Method Names & Parameters Names: should follow the same conventions as variable names. See the section on variable names for details.

Commenting

Comments are extremely important as they should provide readers with useful information about your source code.

- Single-Line Comments: should be above the line of code that the comment pertains to. Having the comments on the same line as the source code often makes the code unreadable. Sinlge line comments can be defined after a simple statement ONLY if the comment is very short. i.e.

```
int COUNT = 5; //number of names

//Loop through the list of names
for(in tx = 0; x < COUNT; x++){
    System.out.println(names[x]);
}
```

- Multi-Line Comments: The rules for multi line comments are similat to those for single line comments. See the section on Single-Line comments for details.

- JavaDoc Comments have their own guidelines. See the following for details:
http://java.sun.com/j2se/javadoc/writingdoccomments/

Formatting

Proper use of tabs and whitespace should be used to make code more readable.

Chapter 5

Architectural Design

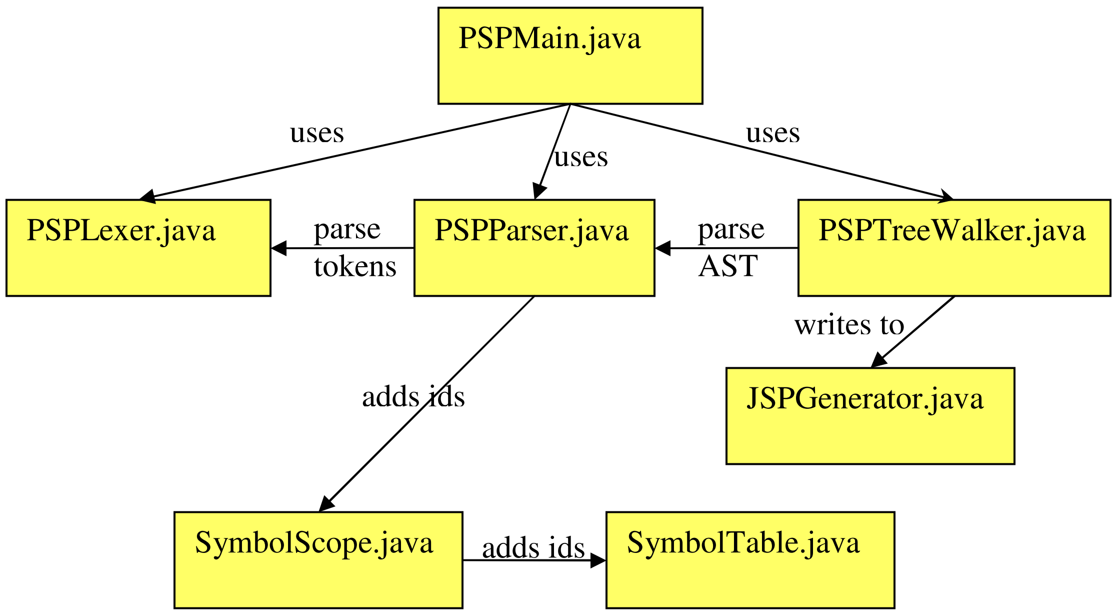

Fig 2: Architectural design of PSP compiler components.

The PSP compiler has 7 major components. The diagram above displays these components and outlines their interaction with other components in the compiler. The PSPLexer and PSPParser are automatically generated by the PSP.g grammar file. Likewise, the PSPTreeWalker is auto-generated by the PSPwalker.g ANTLR grammar file.
The PSPMain file is the main program that drives the compiler. The PSPMain is responsible for creating the PSPLexer, PSPParser and PSPTreeWalker. The PSPLexer is passed into the PSPParser which uses the semantic information to process source tokens. While parsing the source file, the PSPParser adds identifiers it sees to the SymbolTable via the SymbolScope class.
An Abstact Syntax Tree (AST) is produced as a result of running the PSPParser. At which point the AST is passes to the PSPTreeWalker. The PSPTreeWalker has two jobs. While walking the pre-defined AST the PSPTreeWalker is responsible for performing error checks, i.e. relational Vs math expression, and also for generating the .jsp source code.
The PSPTreeWalker uses the JSPGenerator class to generate the source code. While walking the AST the walker writes the source to the JSPGenerator, which buffers the content. The PSPMain has the responsibility of checking if an error occurred during the parsing of tree walking phase. Therefore, it determines if the buffered .jsp source will be written to disk.

Chapter 6

Test Plan

The following diagram outlines the development and testing process. Phases 2, 4 and 5 in the following process diagram highlights the testing phases.

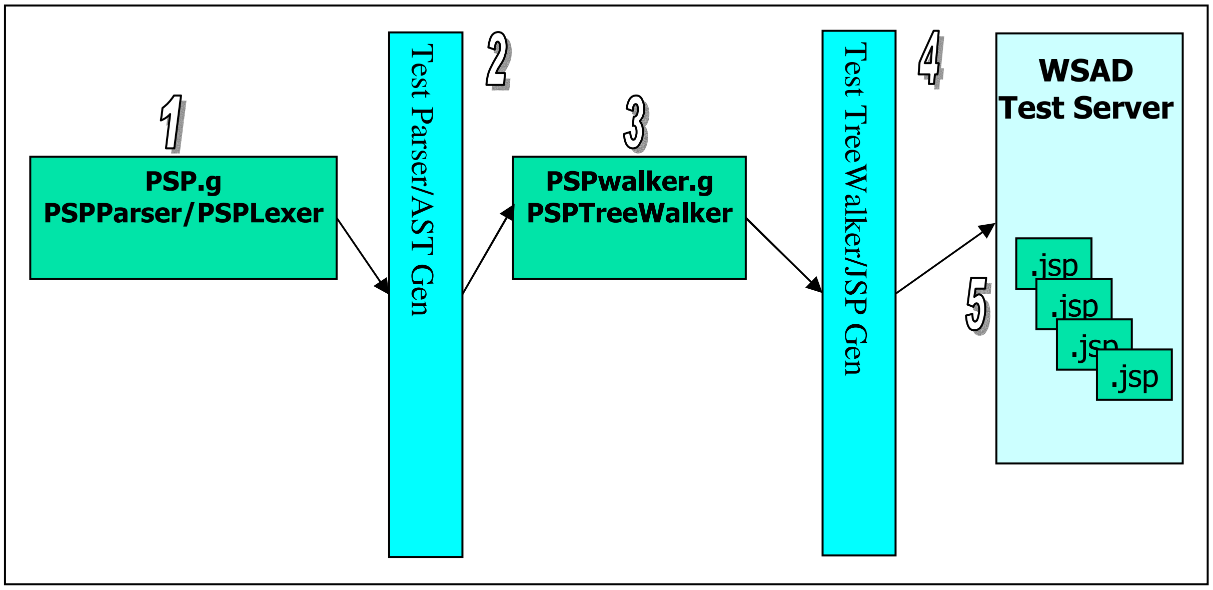

Fig 2: Development and Testing phases.

Both testing phases two and four were simplified by extending the functionality of both the PSPParser and PSPTreeWalker classes.

```
     /** Returns true if an exception was caught while parsing the psp
     file.  Otherwise, returns false. */
    public boolean isCaughtException(){
      return caughtException;
    }

      /** Sets the caughtException flag.  */
    public void setCaughtException(boolean flag){
      caughtException = flag;
    }

    /** Overides the existing Parser reportError Method.
     *  The main purpose for overiding this method is to create
     *  a point of reference that PSP test cases can test.
     */
    public void reportError(RecognitionException ex) {
       super.reportError(ex);
       setCaughtException(true);
    }
```

Overriding the reportError() method allows the PSP compiler to determine when an exception is caught by the PSPParser. The PSPTreeWalker takes advantage of this mechanism by storing a reference to the PSPParser class which allows it to call the `setCaughtException() method.`

How tests were chosen?

Test cases for the PSP language were chosen in alignment with the 3 testing phases, mainly the PSPLexer and PSPParser. When tests passed the first testing phase they could then be passed to the second testing phase, where the .jsp file is generated. Once, the jsp file was generated they were moved to theWSAD test server environment and tested there. For details on the test cases that were considered see the next section.

Testing Automation

Test cases for PSP were written using the Junit 3.8 testing framework. At first glace this may seem like overkill but it greatly simplified the testing process and allowed a smaller granularity of testing and debuging. Using JUnit, I was able to eliminate some 30+ (and growing) test .psp source files, combining them into 12 JUnit test classes. All 12 Test classes were added to the *PSPTestSuite.java* test suite. By running the test suite after any significant changes I was able to perform my system regression test.

The following is a list of all the JUnit test classes and a summary of the test cases that they account for.

JUnit Test case	**Description**

PSPTestAssignmentStmt.java	Test assignment statements: i.e.
	int d = 3;
	d = 3;
	String user = “ONeil”;

PSPTestComments.java	Tests for valid and invalid comment blocks. i.e. comments cannot be nested.
PSPTestEqualsStmt.java	Tests the equals and nequals statement and expression. i.e. verify that only String variables are accepted.
PSPTestForStmt.java	Test the for statement. i.e. for(;;){} /* ensure that this is invalid*/
PSPTestHTML.java	Tests the html statement. i.e. Ensure that PSP and HMTL statements can be nested.
PSPTestIfElseStmt.java	Tests the if else statement. i.e. 1) if(/*condition*/){} 2) if(/*condition*/){}else{}

	3) if(/*condition*/){}else if(/*condition*/){} 4) if(/*condition*/){}else if(/*condition*/)else{}
PSPTestInitStmt.java	Tests all the init statements i.e. 1) int i = 3; 2) String s = "Hello"; 3) String s = request userName; 4) Collection col = session states;
PSPTestIterStmt.java	Tests the iterate staement. i.e. Ensuring that the next keyword is available within this scope.
PSPTestNullStmt.java	Tests the null and notnull statement and expression. i.e. verify that only String variables are accepted.
PSPTestPrintStmt.java	Test the print staement checking the arguments passed to print().
PSPTestRelationExpr.java	Tests all the valid expression supported by the language.
PSPTestWhileStmt.java	Tests the while staement. i.e. Ensuring that the condition passed is a relational expression.

Sample Test PSP files

The following are examples of .psp source code that have been translated to .jsp files.

Hello.psp

```
/* will display "PSP HelloWorld Page" or "Hello [some name]" page */
 <$    <html><head>   $>

String name = request userName;

if(null(name)){
    <$ <h2>PSP Hello World Page</ h2>  $>
}else{
    <$   <h2> Hello $>  print(name); <$ Page </h2>  $>
}
<$ </head > </html>  $>
```

Hello.jsp

```
<%
/* will display "PSP HelloWorld Page" or "Hello [some name]" in the
html title */
%> <html><head>
<%
String name =  request.getParameter("userName");
if ((name== null)){
%> <h2>PSP Hello World Page</ h2>
<% }else {%>
   <h2> Hello
<% out.print(name); %>
 Page </h2>
<%} %>
 </head > </html>
```

displayInfo.psp

```
String userName = session userName;
<$ <html> <head> <title>displayInfo</title> </head> <body
bgcolor=#FFFFFF> <font face="Helvetica"><h2> Personal Information:
$> print(userName); <$</h2>$>
String fname = request fname;
String lname = request lname;
String dob = request dob;
String ssn = request ssn;
String state = request state;
<$<table>   <tr> <td>Key</td> <td>Value</td> </tr>
      <tr> <td>$>print("fname");<$</td> <td>$>print(fname);<$</td>
</tr>$>
      <$<tr> <td>$>print("lname");<$</td> <td>$>print(lname);<$</td>
</tr>$>
      <$<tr> <td>$>print("dob");<$</td> <td>$>print(dob);<$</td>
</tr>$>
      <$<tr> <td>$>print("ssn");<$</td> <td>$>print(ssn);<$</td>
</tr>$>
      <$<tr> <td>$>print("state");<$</td> <td>$>print(state);<$</td>
</tr> </table>    $>
<$
<a href="/psp/welcome.jsp"><h3>Home</h3></a>
<hr></font></body></html>
$>
```

displayInfo.jsp

```
<%
String userName = (String) session.getAttribute("userName");
%>
<html>
<head> <title>displayInfo</title> </head>
<body bgcolor=#FFFFFF>
<font face="Helvetica">
<h2> Personal Information:
<%
out.print(userName);
%>
</h2>
<%
String fname =  request.getParameter("fname");
String lname =  request.getParameter("lname");
String dob =  request.getParameter("dob");
String ssn =  request.getParameter("ssn");
String state =  request.getParameter("state");
%>
<table>
      <tr> <td>Key</td> <td>Value</td> </tr>
<tr> <td><%out.print("fname");%> </td> <td><%out.print(fname);%> </td></tr>
<tr> <td><%out.print("lname");%></td> <td><%out.print(lname);%></td> </tr>
<tr> <td><%out.print("dob");%></td> <td><%out.print(dob);%></td> </tr>
<tr> <td><%out.print("ssn");%></td> <td><%out.print(ssn);%></td> </tr>
<tr> <td><%out.print("state");%></td> <td><%out.print(state);%></td> </tr>

</table>

<a href="/psp/welcome.jsp"><h3>Home</h3></a>
<hr></font></body></html>
```

Chapter 7

Lessons Learned

This has been a very long and frustrating semester, but such is the nature of group assignments. I started of in a group of four and ended up a group of one. I transferred into the class a week late, at which point others groups had already formed and I was left to group with whoever was available.

My group and I did not see eye to eye on a number of things. We made the soft deadlines for the white paper and the LRM but the real work for each assignment would start days before the due date. We had problems making our regularly scheduled Monday meetings and did a poor job of communicating. In a nutshell, I didn't like the way was or wasn't working together and I feel something had to change. I could no longer tolerate the procrastination and decided to make the push to form my own group. I didn't feel like I had one other person who was on the same page as me as far as the level of effort needed for this assignment.

My advice to those taking PLT in the future is to get to class early so you have a choice of who you end up grouping with. This is no guarantee that you won't end up in a similar situation as I but it puts the odds in your favor. Also, regularly scheduled meeting are a great idea if your team actually makes the meeting. My final piece of advice for future PLT students is "Get Your Hands Dirty." ANTLR is pretty scary looking from a far but once you dive into it and learn how to read its error messages things become pretty straightforward.

Appendix

PSP.g

```
header {
    /*
     * PSP.g : Contains the PSP Lexer and Parser.  The Parser
     *         creates the SymbolTable while it parses the program.
     *
     * @author O'Neil Palmer
     *         op2007@columbia.edu
     *
     */
    package psp;
    import psp.parser.*;
}

/* PARSER */
class PSPParser extends Parser;

options {
k=2;
buildAST=true;
exportVocab=PSPVocab;

}

/* Define what the tokes are in our language.  This is used in the AST
   construction.  */
tokens {
   ROOT;
   EXPR;
   STMT;
   ASSIGNSTMT;
   SESSIONASSIGNSTMT;

   HTM = "html";
   COMM = "comment";
   STRCONST = "String";
   INTCONST = "int";
   IFCONST = "if";
   ELSECONST = "else";
   WHILECONST = "while";
   ITERCONST = "iterate";
   FORCONST = "for";
   COLCONST = "Collection";
   PRCONST = "print";
   PRLNCONST = "println";
   SESSCONST = "session";
   REQCONST = "request";
   NULL = "null";
   NOTNULL = "notnull";
   EQUALS = "equals";
   NEQUALS = "nequals";
```

```
}


{
   //Class instance variables.

   /** Set to true if calling a print statment. Otherwise it is set to
false. */
    protected boolean callingPrintStmt = false;

    /** The SymbolTable Stack.  All symbols in the PSP language are in
the same scope.  So
       There is only one level in the symbol table stack for this
language.  */
    protected psp.parser.SymbolScope symbolTableStack = new
psp.parser.SymbolScope();

    /** This flag is used to indicate if an error occured.  Using the
flag allows us to parse the entire psp file,
     reporting all errors, and then terminating the program at the end.
*/
    protected boolean caughtException = false;

    /** This flag is set if the java.util.Collection should be imported
in the generated JSP file.  */
    protected boolean importCollection = false;
     /** This flag is set if the java.util.Iterator should be imported
in the generated JSP file.  */
    protected boolean importIterator = false;


      /** Returns true if the java.util.Collection class should be
imported in the generated JSP. Otherwise return false.  */
    public boolean isImportCollection(){
      return importCollection;
    }

    /** Returns true if the java.util.Iterator class should be imported
in the generated JSP. Otherwise return true. */
    public boolean isImportIterator(){
      return importIterator;
    }

     /** Returns true if an exception was caught while parsing the psp
file.  Otherwise, returns false. */
    public boolean isCaughtException(){
      return caughtException;
    }

      /** Sets the caughtException flag.  */
    public void setCaughtException(boolean flag){
      caughtException = flag;
    }

    /** Overides the existing Parser reportError Method.
     *  The main purpose for overiding this method is to create
     *  a point of reference that PSP test cases can test.
     */
    public void reportError(RecognitionException ex) {
       super.reportError(ex);
       setCaughtException(true);
    }
}

/* PROGRAM */
program
    : { symbolTableStack.enterScope();  }
      ( statement| comment )+ EOF!
      {symbolTableStack.leaveScope();}

     {#program = #([ROOT, "ROOT"],program); }
    ;

comment: COMMENT^
      {#comment = #([COMM,"comment"],comment); }
      ;

html: HTML^
      {#html = #([HTM,"html"],html); }
      ;

/* STATEMENTS */

statement
    :  initStmt SEMI!
    {#statement = #([STMT, "STMT"],statement); }
    |   assignStmt SEMI!
    {#statement = #([STMT, "STMT"],statement); }
    |   sessionAssignStmt SEMI!
    {#statement = #([STMT, "STMT"],statement); }
    |   printStmt SEMI!
    {#statement = #([STMT, "STMT"],statement); }
    |   ifStmt
    {#statement = #([STMT, "STMT"],statement); }
    |   forStmt
    {#statement = #([STMT, "STMT"],statement); }
    |   whileStmt
    {#statement = #([STMT, "STMT"],statement); }
    |   iterStmt
     {#statement = #([STMT, "STMT"],statement); }
    |   nullStmt
    {#statement = #([STMT, "STMT"],statement); }
    |   notnullStmt
     {#statement = #([STMT, "STMT"],statement); }
    |  strequalsStmt
    {#statement = #([STMT, "STMT"],statement); }
    |   strnequalsStmt
     {#statement = #([STMT, "STMT"],statement); }
       | html
      {#statement = #([STMT, "STMT"],statement); }
    |  i:ID INC^ SEMI!
      {
      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.INT_SYMBOL_TYPE);
```

```
           }
        {#statement = #([STMT, "STMT"],statement); }
        |  i2:ID DEC^ SEMI!
          {
          symbolTableStack.getAndTestSymbolType(this,
i2.getText(),SymbolTable.INT_SYMBOL_TYPE);
            }
        {#statement = #([STMT, "STMT"],statement); }
            ;


nullExpr:
      NULL^ LPAREN! i:ID RPAREN!
      {
      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
    }
      ;

nullStmt
    : NULL^ LPAREN! i:ID RPAREN!  LBRACE! (statement)* RBRACE!
      {
      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
    }
    ;

notnullExpr:
      NOTNULL^ LPAREN! i:ID RPAREN!
    {
      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
    }
      ;

notnullStmt
    :   NOTNULL^ LPAREN! i:ID RPAREN! LBRACE! (statement)* RBRACE!
      {
      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
    }
    ;

strequals: EQUALS^ LPAREN! (i:ID {

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                          }
                                    | STRING)
                        COMMA (i2:ID  {

      symbolTableStack.getAndTestSymbolType(this,
i2.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                          }
                                     | STRING) RPAREN!
      ;
strequalsStmt
```

```
    :  EQUALS^ LPAREN! (i:ID     {

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                          }
                              | STRING)
                        COMMA (i2:ID {

      symbolTableStack.getAndTestSymbolType(this,
i2.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                          }
                              | STRING) RPAREN!  LBRACE! (statement)*
RBRACE!
    ;

strnequals: NEQUALS^ LPAREN! (i:ID     {

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                                      }
                                    | STRING) COMMA (i2:ID {

      symbolTableStack.getAndTestSymbolType(this,
i2.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                                      }
                                    | STRING) RPAREN!
      ;
strnequalsStmt
    :  NEQUALS^ LPAREN! (i:ID {

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                    }
                                    | STRING) COMMA (i2:ID {

      symbolTableStack.getAndTestSymbolType(this,
i2.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                           }
                                          | STRING) RPAREN!  LBRACE!
(statement)* RBRACE!
    ;

initStmt
    :    (intInitStmt |stringInitStmt|colInitStmt)
    ;

intInitStmt
    :    (INTCONST^ i:ID (ASSIGN expression)? )
    {
       symbolTableStack.addSymbol(this,
i.getText(),SymbolTable.INT_SYMBOL_TYPE);

     };

stringInitStmt
    :  STRCONST^ s:ID (ASSIGN (STRING | getString))?
             {
                 symbolTableStack.addSymbol(this,
s.getText(),SymbolTable.STRING_SYMBOL_TYPE);

              }
    ;

colInitStmt
      : COLCONST^ i:ID ASSIGN "session" ID
      {
                 symbolTableStack.addSymbol(this,
i.getText(),SymbolTable.COLLECTION_SYMBOL_TYPE);
                 importCollection = true;
      }
   ;

getString:
      ("session" | "request") ID;


assignStmt
        :   i:ID ASSIGN (expression
                                          {

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.INT_SYMBOL_TYPE);
                                          }
                                | STRING
                                          {

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                          }
                                          | getString
                                 )
      {#assignStmt = #([ASSIGNSTMT, "assignment"],assignStmt);    }
      ;

sessionAssignStmt:   (SESSCONST^) ID ASSIGN (STRING | NULL | i:ID{

      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.STRING_SYMBOL_TYPE);
                                          }
                                          )
    ;

ifStmt
    :   IFCONST^ LPAREN! expression RPAREN! LBRACE! (statement)* RBRACE!
      (elseStmt)?
    ;
elseStmt:
      ELSECONST^ ( ifStmt | LBRACE! (statement)* RBRACE! )
      ;

whileStmt
    :   WHILECONST^ LPAREN! expression RPAREN! LBRACE! (statement)*
RBRACE!
    ;
```

```
increment: i:ID INC^
       {
       symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.INT_SYMBOL_TYPE);
    }
  ;

decrement: i:ID DEC^
       {
       symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.INT_SYMBOL_TYPE);
        }
  ;


forStmt
    :   FORCONST^ LPAREN! (initStmt  |assignStmt ) SEMI! expression
SEMI! (increment |decrement|assignStmt | sessionAssignStmt|printStmt)
RPAREN! LBRACE!
        (statement)* RBRACE!
    ;

printStmt
    :
        PRCONST^ LPAREN! {callingPrintStmt = true;} (expression|STRING)
RPAREN!  {callingPrintStmt = false;}
    |   PRLNCONST^ {callingPrintStmt = true;} LPAREN!
(expression|STRING) RPAREN!  {callingPrintStmt = false;}


    ;

iterStmt
    :   ITERCONST^ { symbolTableStack.enterIterScope(this);  } LPAREN!
i:ID RPAREN! LBRACE! (statement)* RBRACE!
{ symbolTableStack.leaveIterScope();  }
    {
      symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.COLLECTION_SYMBOL_TYPE);
      importIterator = true;
    }
    ;

/* EXPRESSIONS */

expression
    : (relationExpr|nullExpr|notnullExpr|strequals|strnequals)
((AND^|OR^) (relationExpr|nullExpr|notnullExpr|strequals|strnequals))*
    ;

relationExpr
    :  mathExpr ((EQ^|NEQ^|GT^|GTE^|LT^|LTE^) mathExpr)*
    ;

mathExpr
    :  prodExpr ( ( PLUS^  | MINUS^) prodExpr )*
    ;

prodExpr
    :   atom  ((TIMES^ | DIVBY^) atom )*
    ;

atom
    :  INT
    |  i:ID (INC^ | DEC^)?
    {
      if(!callingPrintStmt){
            symbolTableStack.getAndTestSymbolType(this,
i.getText(),SymbolTable.INT_SYMBOL_TYPE);
      }else
            symbolTableStack.getAndTestSymbolType(this, i.getText());
    }
    |   LPAREN! expression RPAREN!
    ;




/* LEXER */


class PSPLexer extends Lexer;

options {
    k=2; //for newline stuff
    charVocabulary='\u0000'..'\u007F';
}


protected DIGIT: '0'..'9';
protected LETTER: ('a'..'z' | 'A'..'Z' | '_');


INT : (DIGIT)+ ;


STRING :
        '"'  //begin double quotes (thrown away)
        ( ~('"'|'\n')
        | ('"''"') )* //any number of characters that aren't " or
endline
        '"'  //end double quotes (thrown away)
       ;

/* OPERATORS */
LPAREN: '('  ;
RPAREN: ')'  ;
LBRACE: '{'  ;
RBRACE: '}'  ;
PLUS:   '+'  ;
MINUS:  '-'  ;
```

```
TIMES:  '*'  ;
DIVBY:  '/'  ;
ASSIGN: '='  ;
AND:    "&&" ;
OR:     "||" ;
EQ:     "==" ;
NEQ:    "!=" ;
LT:     '<'  ;
LTE:    "<=" ;
GT:     '>'  ;
GTE:    ">=" ;
INC:   "++";
DEC:   "--";
COMMA:  ','  ;

/* IDENTIFIERS */

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

/* COMMENTS, WHITE SPACE AND SEMICOLON */

COMMENT
    :   "/*"
        ( options {greedy=false;} :
            ~('\n'|'\r')
        | ('\n'|'\r') {newline();}
        )*
        "*/"
    |   "//" (~('\n'|'\r'))*
    ;

WS
    :   (' '
        |'\t'
        | ( "\r\n"
          | '\n'
          | '\r'
          )
          {newline(); }
        )
        { $setType(Token.SKIP); }
    ;

SEMI : ';' ;

/* HTML */

HTML
    :   "<$"!
        (options {greedy=false;} :
            ~('\n'|'\r')
        | ('\n'|'\r') {newline();}
        )*
        "$>"!
    ;
```

PSPwalker.g

```
header {
    /*
     * PSP.g : Contains the PSP Tree walker.
     *
     * @author O'Neil Palmer
     *         op2007@columbia.edu
     *
     */
    package psp;

    import psp.walker.JSPGenerator;
    import java.io.FileNotFoundException;
}

/* PARSER */
class PSPTreeWalker extends TreeParser;

options {
buildAST=true;
importVocab=PSPVocab;

}


/* Define what the tokes are in our language.  This is used in the AST
   construction.  */

{
      //Class instance variables.

     /** This flag is set to true when parsing a relational expression.
Othewise, it is set to false.*/
      protected boolean relExpressionType = false;

      /** This counter is appended to each Iterator variable to avout
duplicate variable names.  */
      private int iteratorCount = 0;

      /** The generator is used to create the JSP file.  */
      protected JSPGenerator generator = null;

      /** A reference to the PSPParser class that created the AST being
parsed by the walker.  */
      protected PSPParser pspparser = null;

      /**
      *  Constructor
      */
      public PSPTreeWalker(PSPParser parser, String fileName){
            super();
            generator = new JSPGenerator(fileName);
            pspparser = parser;
      }

      /**
      * This method checks if the relExpressionType is set to true.  If
not
      * an exception is thrown.
      */
      protected void checkRelExpressionType(){

            //The expression is a regular math expression
            if(relExpressionType == false)
                  pspparser.setCaughtException(true);

            //reset relExpressionType
      relExpressionType = false;
      }

      /**
      * This method checks if the relExpressionType is set to false.
If not
      * an exception is thrown.
      */
      protected void checkMathExpressionType(){

            //The expression is a relational expression.
            if(relExpressionType == true)
                  pspparser.setCaughtException(true);

            //reset relExpressionType
      relExpressionType = false;
      }

      /**
      * Adds the import declaration for java.util.Iterator.
      */
      public void importIterator(){
            generator.writeImport("<%@page
import=\"java.util.Iterator\"  %>");
            generator.newline();
      }

      /**
      * Adds the import declaration for java.util.Collection.
      */
      public void importCollection(){
            generator.writeImport("<%@page
import=\"java.util.Collection\"  %>");
            generator.newline();
      }

      /** This function must be called after parsing the AST.
Otherwise
       *   the jsp file will not be created.
       */
      public void close(){
            generator.close();
      }

}
```

```
/* PROGRAM */
program
    :
      #(ROOT (#(STMT statement) | html | comment )*)
    ;

comment:
       #(COMM c:COMMENT)
       {generator.write(c.getText());
        generator.newline(); }
      ;

html:
       #(HTM h:HTML)
       {generator.writeHTML(h.getText());
       generator.newline(); }
      ;

/* STATEMENTS */

statement
    :  initStmt
    |  assignStmt
      |  sessionAssignStmt
    |  ifStmt
    |    whileStmt
    |        forStmt
    |    printStmt
    |  iterStmt
    |  nullStmt
    |    notnullStmt
    |  strequalsStmt
    |   strnequalsStmt
    |   html
      | #(INC i:ID)
      {generator.write(i.getText()+"++;");
            generator.newline();
    }
      | #(DEC i2:ID)
      {generator.write(i2.getText()+"--;");
            generator.newline();
    }
    ;

strequalsStmt
    :   #(EQUALS {generator.write("if(");}(i:ID
{generator.write(i.getText());}
                                                       | s:STRING
{generator.write("\""+s.getText()+"\"");})
      COMMA  {generator.write("");} (i2:ID
{generator.write(i2.getText());}
                                                       | s2:STRING
{generator.write("\""+s2.getText()+"\"");})

      { generator.write("){ ");
            generator.newline();
      }
       (#(STMT statement))*  )
    {
      generator.newline();
      generator.write("}");
            generator.newline();
    }
      ;

strnequalsStmt
    :   #(NEQUALS {generator.write("if(");}(i:ID
{generator.write(i.getText());}
                                                       | s:STRING
{generator.write("\""+s.getText()+"\"");})
      COMMA  {generator.write("");} (i2:ID
{generator.write(i2.getText());}
                                                       | s2:STRING
{generator.write("\""+s2.getText()+"\"");})

      { generator.write("){ ");
            generator.newline();
      }
       (#(STMT statement))* )
    {
      generator.newline();
      generator.write("}");
            generator.newline();
    }
    ;


nullStmt
    :   #(NULL i:ID
           {generator.write("if("+i.getText()+" == null){ ");
            generator.newline();
           }
        (#(STMT statement))*  )
    {
      generator.newline();
      generator.write("}");
            generator.newline();
    }
    ;

notnullStmt
   :   #(NOTNULL i:ID
           {generator.write("if("+i.getText()+" != null){ ");
            generator.newline();
           }
        (#(STMT statement))* )
    {
      generator.newline();
      generator.write("}");
            generator.newline();
    }
    ;
```

```
initStmt
    :   (intInitStmt|stringInitStmt|colInitStmt)
    ;

intInitStmt:
     #(INTCONST {boolean initialized = false; generator.write("int ");}
i:ID {generator.write(i.getText());} (ASSIGN {initialized = true;
generator.write("=");} expression )? )

     {
      //initialize the int value to zero
      if(!initialized)
            generator.write(" = 0");

      checkMathExpressionType();

      generator.write(";");
            generator.newline();
     }

    ;


stringInitStmt
    :  #(STRCONST {boolean initialized = false; generator.write("String
");} i:ID {generator.write(i.getText());} (ASSIGN {initialized = true;
generator.write(" = ");}
      (s:STRING {generator.write(s.getText());} | getString))? )
         {
         //initialize the int value to null
      if(!initialized)
            generator.write(" = null");

            generator.write(";");
            generator.newline();
      }
     ;

colInitStmt
      : #(COLCONST i:ID ASSIGN "session" i2:ID )
       {
      generator.write("Collection ");

      generator.write(i.getText()+" = (Collection) ");

      generator.write("session.getAttribute(\"");
            generator.write(i2.getText());
            generator.write("\");");

            generator.newline();
      }
   ;

getString:
     (SESSCONST i:ID)
            {
                  generator.write("(String) session.getAttribute(\"");
                  generator.write(i.getText());
                  generator.write("\")");

            }
       | (REQCONST i2:ID)
            {
                  generator.write(" request.getParameter(\"");
                  generator.write(i2.getText());
                  generator.write("\")");
            }
            ;

assignStmt
    :  #(ASSIGNSTMT i:ID {generator.write(i.getText());} ASSIGN
{generator.write("=");}  ( expression | s:STRING
{generator.write(s.getText());} | | getString ))
      {
            generator.write(";");
            generator.newline();
      }
    ;

sessionAssignStmt:
         #(SESSCONST key:ID ASSIGN (val:STRING|val2:NULL | val3:ID))
            {
                  generator.write("session.setAttribute(\"");
                  generator.write(key.getText());
                  generator.write("\",");
                  if(val != null)
                        generator.write("\""+val.getText()+"\"");
                  else if(val2 != null)
                              generator.write(val2.getText());
                  else
                        generator.write(val3.getText());
                  generator.write(");");
                  generator.newline();
            }
    ;

ifStmt
    :   #(IFCONST {generator.write("if (");}  expression
                {checkRelExpressionType();
generator.write("){");generator.newline();}
                (#(STMT statement))*
                 {generator.write("}");} (elseStmt)? )
      {
                  generator.newline();
      }
    ;
elseStmt:
      #(ELSECONST {generator.write("else ");} ( ifStmt |
{generator.write("{");} (#(STMT statement))* ) {generator.write("}");} )
            {
                  generator.newline();
      }
      ;
```

```
whileStmt
    :   #(WHILECONST  {generator.write("while( ");}expression
      {checkRelExpressionType();
       generator.write("){ ");
       generator.newline();}
      (#(STMT statement))* {generator.write("}");} )
     {
      generator.newline();
    }
    ;

increment: #(INC i:ID)
            {generator.write(i.getText()+"++");
            generator.newline();
    }
  ;

decrement: #(DEC i:ID)
      {generator.write(i.getText()+"--");
            generator.newline();
    }
  ;

forStmt
    :   #(FORCONST {generator.write("for( ");} (initStmt  |assignStmt)
expression {checkRelExpressionType(); generator.write(";");} (increment
|decrement|assignStmt | sessionAssignStmt|printStmt)
{generator.write(")");}
      {generator.write("{"); generator.newline();}
        (#(STMT statement))*  {generator.write("}"); })
         {
            generator.newline();
      }
    ;

printStmt
    :   #(PRCONST  {generator.write("out.print(");}(expression
{checkMathExpressionType();}|s:STRING {generator.write(s.getText());})
{generator.write(");");})
     {
      generator.newline();
     }
    |   #(PRLNCONST
{generator.write("out.println(");}(expression{checkMathExpressionType()
;}|s2:STRING  {generator.write(s2.getText());})  {generator.write(");");})
    {
      generator.newline();
    }
    ;

iterStmt
    :   #(ITERCONST i:ID
     {
      String iterName = "iter"+iteratorCount++;

      generator.write("if("+i.getText()+" != null) {");
      generator.newline();
```

```
       generator.write("Iterator ");
       generator.write(iterName);
       generator.write(" = ");
       generator.write(i.getText());
       generator.write(".iterator();");
       generator.newline();

       generator.write("while("+iterName+".hasNext()){");
       generator.newline();
       generator.write("String next = (String) "+iterName+".next();");
       generator.newline();

     }
      (#(STMT statement))* )
       {

             generator.newline();
             generator.write("}//End while");
       generator.newline();
       generator.write("}//End if");
       generator.newline();

      }
     ;
/* EXPRESSIONS */

expression
    :
              #(AND { relExpressionType = true; generator.lparen(); }
expression { generator.write(" && "); } expression
{ generator.rparen(); } )
      | #(OR { relExpressionType = true; generator.lparen(); }
expression { generator.write(" || "); } expression
{ generator.rparen(); })

      | #(EQUALS { relExpressionType = true; }
            (i:ID {generator.write(i.getText());}| s:STRING
{generator.write(s.getText());}) COMMA
            {generator.write(".equals(");}
            (i2:ID {generator.write(i2.getText());}|s2:STRING
{generator.write(s2.getText());})) {generator.write(")"); }
      | #(NEQUALS { relExpressionType = true; }
                  (i3:ID {generator.write(i3.getText());}| s3:STRING
{generator.write(s3.getText());}) COMMA
                  {generator.write(".equals(");}
                  (i4:ID{generator.write(i4.getText());}|s4:STRING
{generator.write(s4.getText());})) {generator.write(")"); }

            | #(NULL i5:ID){ relExpressionType = true;
generator.write("("+i5.getText()+"== null)"); }
      | #(NOTNULL i6:ID){ relExpressionType = true;
generator.write("("+i6.getText()+" != null)"); }


      | #(PLUS { generator.lparen(); } expression
{ generator.write("+"); } expression { generator.rparen(); })
```

```
      | #(MINUS { generator.lparen(); } expression { generator.write("-
"); } expression { generator.rparen(); })

      | #(TIMES { generator.lparen(); } expression
{ generator.write("*"); } expression { generator.rparen(); })
      | #(DIVBY { generator.lparen(); } expression
{ generator.write("/"); } expression { generator.rparen(); })

      | #(EQ { relExpressionType = true; generator.lparen(); }
expression { generator.write("=="); } expression
{ generator.rparen(); })
            | #(NEQ { relExpressionType = true; generator.lparen(); }
expression { generator.write("!="); } expression
{ generator.rparen(); })
        | #(GT { relExpressionType = true; generator.lparen(); }
expression { generator.write(">"); } expression { generator.rparen(); })
        | #(GTE { relExpressionType = true; generator.lparen(); }
expression { generator.write(">="); } expression
{ generator.rparen(); })
        | #(LT { relExpressionType = true; generator.lparen(); }
expression { generator.write("<"); } expression { generator.rparen(); })
        | #(LTE { relExpressionType = true; generator.lparen(); }
expression { generator.write("<="); } expression
{ generator.rparen(); })
      | i7:INT  { generator.write(i7.getText()); }
      | #(INC i8:ID { generator.write(i8.getText()+"++");} )
      | #(DEC i9:ID { generator.write(i9.getText()+"--");} )
      | i10:ID { generator.write(i10.getText()); }
    ;
```

SymbolScope.java

```
/*
 * Created on Nov 20, 2004
 */
package psp.parser;

import java.util.HashMap;
import java.util.Stack;
import psp.PSPParser;
import psp.parser.exception.PSPParsingException;

/**
 * SymbolScope is a type of Stack that stores a
<code>SymbolTable</code>
 * object for each "scope" entered.   When you enter a scope an entry
is placed
 * on the stack.  When you exit a scope that entry is popped off.
 * <b>The PSP language will only have one scope.<b>
 *
 * @author O'Neil Palmer
 *         op2007@columbia.edu
 */
public class SymbolScope extends Stack {

      /** Map that store the expected keywords.  */
      public static HashMap KEYWORDS_MAP = new HashMap();

      /** Initialize the expected KEYWORDS_MAP */
      static { KEYWORDS_MAP.put("next","next");
                  KEYWORDS_MAP.put("int","int");
                  KEYWORDS_MAP.put("String","String");
                  KEYWORDS_MAP.put("Collection","Collection");
      }

      /** This flag is set to true when within the scope of a iterate
statement.  */
      protected boolean iterStmtScope = false;

      /**
       * Constructor
       */
      public SymbolScope() {
            super();
      }

      /**
       * This method checks that identifiers are not predefined
keywords.
       * @param key Object
       * @throws PSPParsingException
       */
      protected void checkForKeywords(Object key) throws
PSPParsingException{

            String value = (String) KEYWORDS_MAP.get(key);
            if(value != null){

                  throw new PSPParsingException("Attempting to use
keyword< "+key+" >as identifier.");
            }

      }
      /**
       * When you add an object, you are always adding it to the symbol
table of the
       * current scope.
       *
       * @param key
       * @param value  The type of the identifier.  i.e. int or String
       * @return boolean
       */
      public boolean addSymbol(PSPParser parser, Object key, Object
value) {
            SymbolTable st = null;
            boolean addSuccessful = true;

            try {
                  checkForKeywords(key);
                  //Add check if the value is valid.
                  st = (SymbolTable) this.pop();
                  st.addSymbol(key, value);
                  //Put the symbol table back on the stack.
                  this.push(st);
            } catch (PSPParsingException e) {
                  System.err.println(e.getMessage());
                  addSuccessful = false;
                  parser.setCaughtException(true);
                  this.push(st);
            }

            return addSuccessful;
      }

      /**
       * Check the symbol table to see if a variable with the name key
       * has already been added to the symbol table.  If so, notify the
       * parse that an exception has occured.
       *
       * @param parser PSPParser
       * @param key Object
       * @param type String
       * @return
       */
      public boolean getAndTestSymbolType(PSPParser parser, Object key)
{
            SymbolTable st = null;
            boolean testSuccesful = true;

            try {
                  //Check if using the "next" identifier in the scope
of a iter statement.
                  //but Do not add the next identifier to the symbol
table.
                  if( ((String) key).equals("next")){
                        if(!isIterStmtScope())
                              throw new PSPParsingException("The 'next'
keyword can only be used in the scope of an iter statement.");

                        return true;
                  }

                  st = (SymbolTable) this.pop();

                  st.getAndTestSymbolType(key);

                  //Put the symbol table back on the stack.
                  this.push(st);

            } catch (PSPParsingException e) {
                  System.err.println(e.getMessage());
                  testSuccesful = false;
                  parser.setCaughtException(true);
                  this.push(st);

            }

            return testSuccesful;
      }


      /**
       * Check the symbol table to see if a variable with the name key
       * has already been added to the symbol table.  If so, notify the
       * parse that an exception has occured.
       *
       * @param parser PSPParser
       * @param key Object
       * @param type String
       * @return
       */
      public boolean getAndTestSymbolType(PSPParser parser, Object key,
String type) {
            SymbolTable st = null;
            boolean testSuccesful = true;

            try {

                  //Check if using the "next" identifier in the scope
of a iter statement.
                  //but Do not add the next identifier to the symbol
table.
                  if( ((String) key).equals("next")){
                        if(!isIterStmtScope())
                              throw new PSPParsingException("The 'next'
keyword can only be used in the scope of an iter statement.");
```

```
                        return true;
                }

                st = (SymbolTable) this.pop();

                st.getAndTestSymbolType(key, type);

                //Put the symbol table back on the stack.
                this.push(st);

            } catch (PSPParsingException e) {
                System.err.println(e.getMessage());
                testSuccesful = false;
                parser.setCaughtException(true);
                this.push(st);

            }

            return testSuccesful;
    }

    /**
     * When you enter a scope you want to creat a symbol table for
the scope that
     * you are about to enter and place it on the stack.
     */
    public void enterScope() {
            SymbolTable st = new SymbolTable();
            this.push(st);
    }

    /**
     * When you leave a scope you want to remove the symbol table for
that scope.
     *
     */
    public void leaveScope() {
            SymbolTable st = (SymbolTable) this.pop();
    }
    /**
     * Returns true if within an iterate statement scope.
     * @return
     */
    protected boolean isIterStmtScope() {
            return iterStmtScope;
    }

    /**
     * Sets the iterStmtScope flag.
     * @param b
     */
    protected void setIterStmtScope(boolean b) {
            iterStmtScope = b;
    }

    /**
     * Called when you enter the scope of an iterate statment.
```

```
	 * "next" is a predefined identifier visible within this context.
	 * 
	 * @param parser PSPParser
	 */
	public void enterIterScope(PSPParser parser) {
		
		if(isIterStmtScope()){
			System.err.println("Iterate statements cannot be 
nested.");
			parser.setCaughtException(true);
		}
		
		setIterStmtScope(true);
	}
	
	/**
	 * When you leave a scope you want to remove the symbol table for 
that scope.
	 * 
	 */
	public void leaveIterScope() {
		setIterStmtScope(false);
	}
}
```

SymbolTable.java

```
/*
 * Created on Nov 20, 2004
 *
 */
package psp.parser;

import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import psp.parser.exception.PSPParsingException;

/**
 * A SymbolTable object is created when you enter a new scope.
 * The Symbol table stores all the variable names definied within 
 * the scope.  The variable name is used as the key for each entry and
 * the variable type is used as the value.
 * i.e.  int number;  key = "number" , value = "int"
 * 
 * @author O'Neil Palmer
 *         op2007@columbia.edu
 */
public class SymbolTable extends HashMap {
	
	/** The int symbol type.  */
	public static final String INT_SYMBOL_TYPE = "int";
	/** The String symbol type.  */
```

```
      public static final String STRING_SYMBOL_TYPE = "String";
      /** The Collection symbol type.  */
      public static final String COLLECTION_SYMBOL_TYPE = "Collection";

      /**
       * Constructor
       */
      public SymbolTable() {
            super();
      }

      /**
       * This method add the variable name key to the Symbol table if
it
       * doesnt exist.  Otherwise, and Exception is thrown.
       *
       * @param key Object
       * @param value Object
       * @throws psp.parser.exception.PSPException
       */
      public void addSymbol(Object key, Object value) throws
PSPParsingException {
            Object val = this.get(key);

            if (val == null) {
                  //The identifier has not yet been defined within this
scope.
                  this.put(key, value);
            } else {
                  //Identifier already defined. throw an exception
                  throw new PSPParsingException("Duplicate Variable:
"+key);
            }
      }

      /**
       * This method check if the variable name is already defined.  If
so,
       * an Exception is thrown.  Otherwise the type of the variable
name is
       * returned.
       *
       * @param key Object
       * @throws psp.parser.exception.PSPParsingException
       */
      public Object getAndTestSymbolType(Object key) throws
PSPParsingException  {

            String keyType = (String) this.get(key);

            if(keyType == null){
                  StringBuffer sb = new StringBuffer("Identifier not
yet defined: ");
                  sb.append(key);
                  throw new PSPParsingException(sb.toString());
            }
            return keyType;

      }

      /**
       * This method check if the variable name is already defined.  If 
so,
       * an Exception is thrown.  Otherwise the type of the variable 
name is
       * returned.
       *
       * @param key Object
       * @param type String
       * @throws psp.parser.exception.PSPParsingException
       */
      public Object getAndTestSymbolType(Object key, String type)
throws PSPParsingException  {

            String keyType = (String) this.get(key);

            if(keyType == null){
                  StringBuffer sb = new StringBuffer("Identifier not 
yet defined: ");
                  sb.append(key);
                  throw new PSPParsingException(sb.toString());
            }

            if(!keyType.equals(type)){
                  StringBuffer sb = new StringBuffer("Invalid type for 
identifier: ");
                  sb.append(key).append(" Actual value = 
").append(keyType).append(", Expects: ").append(type);
                  throw new PSPParsingException(sb.toString());
            }

            return keyType;

      }
}
```

JSPGenerator.java

```
/*
 * Created on Nov 22, 2004
 *
 */
package psp.walker;

import java.io.BufferedOutputStream;
import java.io.BufferedWriter;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileDescriptor;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import psp.parser.exception.PSPWalkerException;

/**
 * The JSP Generator is used by the PSPTreeWalker for generating
 * the JSP file.  This class has utility methods for writing to the
 * JSP file.
 *
 * @author O'Neil Palmer
 *         op2007@columbia.edu
 *
 */
public class JSPGenerator {

      /** buffers the generated JSP.  */
      protected ByteArrayOutputStream buffer = null;
      /** The generated JSP filename.  */
      protected String filename = null;
      /** The PSP tag that begins the start of HTML.  */
      public static final String PSP_HTML_START = "<$";
      /** The PSP tag that begins the end of HTML.  */
      public static final String PSP_HTML_END = "$>";
      /** This flag is set to true when the generator is writing PSP
code
       * is set to false when writing HTML.  This flag is used to know
when to
       * generate start and end JSP tags.
       */
      public boolean writingPSP = false;

      /**
       * Constructor.
       * @param name
       */
      public JSPGenerator(String name) {
            super();
            filename = name;
            buffer = new ByteArrayOutputStream(512);
      }

      /**
       * Called when writing JSP Import tags.
       * @param content
       */
      public void writeImport(String content) {
            try {
                  if (content != null)
                        buffer.write(content.getBytes());
            } catch (IOException e) {
                  throw new PSPWalkerException(e);
            }
      }

      /**
       * Writes the given string to the generated JSP buffer.
       * @param content
       */
```

```
public void write(String content) {
	try {
		if (content != null) {
			if (!writingPSP) {
				buffer.write("<%".getBytes());
				newline();
			}
			buffer.write(content.getBytes());
		}

		writingPSP = true;
	} catch (IOException e) {
		throw new PSPWalkerException(e);
	}
}

/**
 * Called when writing HTML content to the generated JSP buffer.
 * @param content
 */
public void writeHTML(String content) {
	try {
		if (content != null) {
			if (writingPSP) {
				buffer.write("%>".getBytes());
				newline();
			}
			buffer.write(content.getBytes());
		}
		writingPSP = false;
	} catch (IOException e) {
		throw new PSPWalkerException(e);
	}
}

/**
 * Writes a left brace to the generated JSP.
 *
 */
public void lbrace() {
	try {
		buffer.write("{".getBytes());
	} catch (IOException e) {
		throw new PSPWalkerException(e);
	}
}

/**
 * Writes a right brace to the generated JSP.
 *
 */
public void rbrace() {
	try {
		buffer.write("}".getBytes());
	} catch (IOException e) {
		throw new PSPWalkerException(e);
	}
```

```
        }
        /**
         * Writes a left parenthesis to the generated JSP.
         *
         */
        public void lparen() {
                try {
                        buffer.write("(".getBytes());
                } catch (IOException e) {
                        throw new PSPWalkerException(e);
                }
        }
        /**
         * Writes a right parenthesis to the generated JSP.
         *
         */
        public void rparen() {
                try {
                        buffer.write(")".getBytes());
                } catch (IOException e) {
                        throw new PSPWalkerException(e);
                }
        }

        /**
         * Writes a newline character to the generated JSP.
         *
         */
        public void newline() {
                String newline = System.getProperty("line.separator");

                try {
                        buffer.write(newline.getBytes());
                } catch (IOException e) {
                        throw new PSPWalkerException(e);
                }
        }

        /**
         * Close the buffer and ouput the generated JSP file.
         *
         */
        public void close() {
                try {
                        if (writingPSP)
                                buffer.write("%>".getBytes());

                        FileOutputStream fout = new
FileOutputStream(filename);
                        buffer.writeTo(fout);

                        buffer.close();
                } catch (IOException e) {
                        throw new PSPWalkerException(e);
                }
        }
}
```

PSPException.java

```
package psp.parser.exception;

/**
 * PSPException is the root PSP exception class.
 * It is only accessible at the package level and therefore
 * cannot be subclassed.
 *
 * @author O'Neil Palmer
 *         op2007@columbia.edu
 */
class PSPException extends Exception {

      /**
       * Constructor
       */
      protected PSPException() {
            super();
      }

      /**
       * Constructor
       * @param message
       */
      protected PSPException(String message) {
            super(message);
      }

      /**
       * Constructor
       * @param cause
       */
      protected PSPException(Throwable cause) {
            super(cause);
      }

      /**
       * Constructor
       * @param message
       * @param cause
       */
      protected PSPException(String message, Throwable cause) {
            super(message, cause);
      }

}
```

PSPParsingException.java

```
/*
 * Created on Nov 23, 2004
 *
 */
package psp.parser.exception;

/**
 * PSPParsingException is a type of PSPException that is thrown
 * when an exception is encountered while parsing the psp file.
 *
 * @author O'Neil Palmer
 *         op2007@columbia.edu
 */
public class PSPParsingException extends PSPException {

      /**
       * Constructor
       */
      public PSPParsingException() {
            super();
      }

      /**
       * Constructor
       * @param message
       */
      public PSPParsingException(String message) {
            super(message);
      }

      /**
       * Constructor
       * @param cause
       */
      public PSPParsingException(Throwable cause) {
            super(cause);
      }

      /**
       * Constructor
       * @param message
       * @param cause
       */
      public PSPParsingException(String message, Throwable cause) {
            super(message, cause);
      }

}
```

PSPWalkerException.java

```
/*
 * Created on Nov 23, 2004
 *
 */
package psp.parser.exception;

/**
 * PSPWalkerException is and exception that is thrown
 * when an exception is encountered while walking the AST tree
 * generated by the PSPParser.
 *
 * @author O'Neil Palmer
 *         op2007@columbia.edu
 */
public class PSPWalkerException extends RuntimeException {

      /**
       * Constructor
       */
      public PSPWalkerException() {
            super();
      }

      /**
       * @param message
       */
      public PSPWalkerException(String message) {
            super(message);
      }

      /**
       * @param cause
       */
      public PSPWalkerException(Throwable cause) {
            super(cause);
      }

      /**
       * @param message
       * @param cause
       */
      public PSPWalkerException(String message, Throwable cause) {
            super(message, cause);
      }

}
```

Test Cases

PSPTestSuite.java

```
/*
 * Created on Dec 7, 2004
 */
package psp.tests;

import junit.framework.Test;
import junit.framework.TestSuite;

/**
 *  This Test suite contains all the PSP test cases.
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestSuite {

      /**
       * Assembles and returns a test suite
       * containing all known tests.
       *
       * New tests should be added here!
       *
       * @return A non-null test suite.
       */
      public static Test suite() {

            TestSuite suite = new TestSuite();

            suite.addTest(new TestSuite(PSPTestAssignmentStmt.class));
            suite.addTest(new TestSuite(PSPTestComments.class));
            suite.addTest(new TestSuite(PSPTestEqualsStmt.class));
            suite.addTest(new TestSuite(PSPTestForStmt.class));
            suite.addTest(new TestSuite(PSPTestHTML.class));
            suite.addTest(new TestSuite(PSPTestIfElseStmt.class));
            suite.addTest(new TestSuite(PSPTestInitStmt.class));
            suite.addTest(new TestSuite(PSPTestIterStmt.class));
            suite.addTest(new TestSuite(PSPTestNullStmt.class));
            suite.addTest(new TestSuite(PSPTestPrintStmt.class));
            suite.addTest(new TestSuite(PSPTestRelationExpr.class));
            suite.addTest(new TestSuite(PSPTestWhileStmt.class));

            return suite;
      }
      /**
       * Runs the test suite.
       */
      public static void main(String args[]) {
            junit.textui.TestRunner.run(suite());
      }
}
```

PSPTestAssignmentStmt.java

```java
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the assignment statemnts supported and not
 * supported by the language.
 * i.e.
 * int i = 3;
 * String s = "Hello";
 * String s = session svalue;
 * String s = request svalue;
 * Collection col = session list;
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestAssignmentStmt extends TestCase {

      /**
       * Constructor for PSPTestRelationExpr.
       * @param arg0
       */
      public PSPTestAssignmentStmt(String arg0) {
            super(arg0);

      }

      public static void main(String[] args) {
            junit.textui.TestRunner.run(PSPTestAssignmentStmt.class);
      }

      /*
       * This method is called before each test method.
       * @see TestCase#setUp()
       */
      protected void setUp() throws Exception {
            super.setUp();
      }

      /*
       * This method is called after each test method.
       * @see TestCase#tearDown()
```

```
         */
        protected void tearDown() throws Exception {
                super.tearDown();
        }

        /**
         * Creates a PSP Parser for parses the given data (psp source)
string.
         * @param data
         * @return
         * @throws IOException
         * @throws RecognitionException
         * @throws TokenStreamException
         */
        public PSPParser getPSPParser(String data)
                throws IOException, RecognitionException,
TokenStreamException {

                ByteArrayInputStream bytesIn =
                        new ByteArrayInputStream(data.getBytes());
                DataInputStream input = new DataInputStream(bytesIn);

                //Create the lexer and parser and feed them the input
                PSPLexer lexer = new PSPLexer(input);
                PSPParser parser = new PSPParser(lexer);
                // "program" is the main rule in the parser
                parser.program();
                return parser;
        }

        /**
         * Test Simple Math Expressions.
         *
         */
        public void testAssignStmt() {
                String methodName = "testAssignStmt";
                System.out.println("Entering " + methodName);
                try {

                     PSPParser parser = getPSPParser("val = \"BartMan\";");
                      assertTrue(parser.isCaughtException());

                      parser = getPSPParser("String val; val = \"Hello\";
null(val){}");
                      assertFalse(parser.isCaughtException());

                      parser = getPSPParser("int val; null(val){}");
                      assertTrue(parser.isCaughtException());

                      parser = getPSPParser("int val; val = (3+4)*2;");
                      assertFalse(parser.isCaughtException());

                      parser = getPSPParser("int val; val = val--
+(3+4)*2;");
                      assertFalse(parser.isCaughtException());

                      parser = getPSPParser("Collection col;");
```

```
			assertTrue(parser.isCaughtException());

			parser = getPSPParser("Collection col = session
list;");
			assertFalse(parser.isCaughtException());

		} catch (Exception e) {
			System.out.println(e.getMessage());
			assertTrue(false);
		}
		System.out.println("Exiting " + methodName);
	}
}
```

PSPTestComments.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 *  Tests All the comment statemnts supported and not supported by the
language.
 * i.e.
 *  1) Single comments
 *  2) Multiline comments.
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestComments extends TestCase {

	/**
	 * Constructor for PSPTestRelationExpr.
	 * @param arg0
	 */
	public PSPTestComments(String arg0) {
		super(arg0);
	}

	public static void main(String[] args) {
		junit.textui.TestRunner.run(PSPTestComments.class);
	}
```

```
        /*
         * This method is called before each test method.
         * @see TestCase#setUp()
         */
        protected void setUp() throws Exception {
              super.setUp();
        }

        /*
         * This method is called after each test method.
         * @see TestCase#tearDown()
         */
        protected void tearDown() throws Exception {
              super.tearDown();
        }

        /**
         * Creates a PSP Parser for parses the given data (psp source)
  string.
         * @param data
         * @return
         * @throws IOException
         * @throws RecognitionException
         * @throws TokenStreamException
         */
        public PSPParser getPSPParser(String data)
              throws IOException, RecognitionException,
  TokenStreamException {

              ByteArrayInputStream bytesIn =
                    new ByteArrayInputStream(data.getBytes());
              DataInputStream input = new DataInputStream(bytesIn);

              //Create the lexer and parser and feed them the input
              PSPLexer lexer = new PSPLexer(input);
              PSPParser parser = new PSPParser(lexer);
              // "program" is the main rule in the parser
              parser.program();
              return parser;
        }

        /**
         * Test Comments.
         *
         */
        public void testComments() {
              String methodName = "testComments";
              System.out.println("Entering " + methodName);
              try {

              PSPParser parser =
                          getPSPParser("//Hello this is a single line
                          comment");
                    assertFalse(parser.isCaughtException());

                    parser = getPSPParser("//* Hello this is a single
                    line comment");
```

```
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("/Hello this is a single line
			comment");
			assertTrue(parser.isCaughtException());

			parser = getPSPParser("/*Hello this is a multi-line
			comment */");
			assertFalse(parser.isCaughtException());

			boolean exception = false;
			try{
				parser = getPSPParser("/*Hello this is a multi-
			line comment /");

			}catch (Exception e) {
				exception = true;
			}
			assertTrue(exception);


			parser =
				getPSPParser("/*Hello this is a multi-line
			comment/* Nested Comments */ */");
			assertTrue(parser.isCaughtException());

			parser = getPSPParser("/Hello this is a multi-line
			comment */");
			assertTrue(parser.isCaughtException());

		} catch (Exception e) {
			System.out.println(e.getMessage());
			assertTrue(false);
		}
		System.out.println("Exiting " + methodName);
	}

}
```

PSPTestEqualsStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;
import junit.framework.TestCase;

/**
 *
 * Tests all the <b>equal and nequal</b> statements supported and not
 * supported by the language.
 * i.e.
 *    nequals(String1, String2){}
 *    equals(String1, String2){}
 *    nequals(int, String2){}
 *
 *      if(equals(s1,s2)){}
 * The equals and nequals statments can also be used as relational
 * expressions.
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestEqualsStmt extends TestCase {

      /**
       * Constructor for PSPTestRelationExpr.
       * @param arg0
       */
      public PSPTestEqualsStmt(String arg0) {
            super(arg0);
      }

      public static void main(String[] args) {
            junit.textui.TestRunner.run(PSPTestEqualsStmt.class);
      }

      /*
       * This method is called before each test method.
       * @see TestCase#setUp()
       */
      protected void setUp() throws Exception {
            super.setUp();
      }

      /*
       * This method is called after each test method.
       * @see TestCase#tearDown()
       */
      protected void tearDown() throws Exception {
            super.tearDown();
      }

      /**
   * Creates a PSP Parser for parses the given data (psp source) string.
       * @param data
       * @return
       * @throws IOException
       * @throws RecognitionException
       * @throws TokenStreamException
       */
      public PSPParser getPSPParser(String data)
```

```
            throws IOException, RecognitionException,
TokenStreamException {

            ByteArrayInputStream bytesIn =
                  new ByteArrayInputStream(data.getBytes());
            DataInputStream input = new DataInputStream(bytesIn);

            //Create the lexer and parser and feed them the input
            PSPLexer lexer = new PSPLexer(input);
            PSPParser parser = new PSPParser(lexer);
            // "program" is the main rule in the parser
            parser.program();
            return parser;
      }

      /**
       * Test.
       *
       */
      public void testEquals() {
            String methodName = "testEquals";
            System.out.println("Entering " + methodName);
            try {

                  PSPParser parser = getPSPParser("equals(){}");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("String s; equals(s1){}");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("String s; equals(s,s){}");
                  assertFalse(parser.isCaughtException());

                 parser = getPSPParser("String s; equals(s,s){print(3);
                 print(4);}");
                  assertFalse(parser.isCaughtException());

                  parser =
                        getPSPParser("String s;
                        if(equals(s,s)){print(3); print(4);}");
                  assertFalse(parser.isCaughtException());

            } catch (Exception e) {

                  System.out.println(e.getMessage());
                  assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
      }

      /**
             * Test.
             *
             */
      public void testNotEquals() {
            String methodName = "testNotEquals";
            System.out.println("Entering " + methodName);
            try {

                  PSPParser parser = getPSPParser("nequals(){}");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("String s; nequals(s1){}");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("String s; nequals(s,s){}");
                  assertFalse(parser.isCaughtException());

                  parser =
                        getPSPParser("String s; nequals(s,s){print(3);
                  print(4);}");
                  assertFalse(parser.isCaughtException());

                  parser =
                        getPSPParser("String s;
                        if(nequals(s,s)){print(3); print(4);}");
                  assertFalse(parser.isCaughtException());

            } catch (Exception e) {

                  System.out.println(e.getMessage());
                  assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
      }

}
```

PSPTestForStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the <b>for</b> statements supported and not
 * supported by the language.
 * i.e.
 *    for(){}   //Invalid statement.
 *    for(;;){}  //Invalid statement.
 *     for(int x = 0; x < 4; print(x)){}
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestForStmt extends TestCase {

      /**
       * Constructor for PSPTestRelationExpr.
       * @param arg0
       */
      public PSPTestForStmt(String arg0) {
            super(arg0);
      }

      public static void main(String[] args) {
            junit.textui.TestRunner.run(PSPTestForStmt.class);
      }

      /*
       * This method is called before each test method.
       * @see TestCase#setUp()
       */
      protected void setUp() throws Exception {
            super.setUp();
      }

      /*
       * This method is called after each test method.
       * @see TestCase#tearDown()
       */
      protected void tearDown() throws Exception {
            super.tearDown();
      }

      /**
       * Creates a PSP Parser for parses the given data (psp source)
string.
       * @param data
       * @return
       * @throws IOException
       * @throws RecognitionException
       * @throws TokenStreamException
       */
      public PSPParser getPSPParser(String data)
            throws IOException, RecognitionException,
TokenStreamException {

            ByteArrayInputStream bytesIn =
                  new ByteArrayInputStream(data.getBytes());
            DataInputStream input = new DataInputStream(bytesIn);

            //Create the lexer and parser and feed them the input
            PSPLexer lexer = new PSPLexer(input);
            PSPParser parser = new PSPParser(lexer);
            // "program" is the main rule in the parser
            parser.program();
            return parser;
        }
        /**
         * Test Simple Math Expressions.
         *
         */
        public void testForStmt() {
            String methodName = "testForStmt";
            System.out.println("Entering " + methodName);
            try {
                PSPParser parser = getPSPParser("for(){}");
                assertTrue(parser.isCaughtException());

                parser = getPSPParser("for(;;){}");
                assertTrue(parser.isCaughtException());

                parser = getPSPParser("for(int i=0; i < 4;i++){}");
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("int i = 2; for(i=0; i <
                4;i++){}");
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("int i = 2; for(i=0; i <
                4;print(i)){}");
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("int i = 2; for(i=0; i < 4;i--
                ){}");
                assertFalse(parser.isCaughtException());

                parser =
                    getPSPParser("int i = 2; for(i=0; i < 4; String
                s = \"Hey\"){}");
                assertTrue(parser.isCaughtException());

                parser =
                    getPSPParser("String s; int i = 2; for(i=0; i <
                4; s = \"Hey\"){}");
                assertFalse(parser.isCaughtException());

                parser =
                    getPSPParser("int i = 2; for(i=0; i < 4;session
                user = \"op2007\"){}");
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("for(int i=0; i < 4;){}");
                assertTrue(parser.isCaughtException());

            } catch (Exception e) {
                System.out.println(e.getMessage());
                assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
        }

    }
```

PSPTestHTML.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the <b>html</b> statements supported and not
 * supported by the language.
 * i.e. nested psp and html blocks.
 *
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestHTML extends TestCase {

	/**
	 * Constructor for PSPTesHTML.
	 * @param arg0
	 */
	public PSPTestHTML(String arg0) {
		super(arg0);
	}

	public static void main(String[] args) {
		junit.textui.TestRunner.run(PSPTestHTML.class);
	}

	/*
	 * This method is called before each test method.
	 * @see TestCase#setUp()
	 */
	protected void setUp() throws Exception {
		super.setUp();
	}

	/*
	 * This method is called after each test method.
	 * @see TestCase#tearDown()
	 */
	protected void tearDown() throws Exception {
		super.tearDown();
	}
```

```
        /**
         * Creates a PSP Parser for parses the given data (psp source)
string.
         * @param data
         * @return
         * @throws IOException
         * @throws RecognitionException
         * @throws TokenStreamException
         */
        public PSPParser getPSPParser(String data)
                throws IOException, RecognitionException,
TokenStreamException {

                ByteArrayInputStream bytesIn =
                        new ByteArrayInputStream(data.getBytes());
                DataInputStream input = new DataInputStream(bytesIn);

                //Create the lexer and parser and feed them the input
                PSPLexer lexer = new PSPLexer(input);
                PSPParser parser = new PSPParser(lexer);
                // "program" is the main rule in the parser
                parser.program();
                return parser;
        }

        /**
         * Test Simple Math Expressions.
         *
         */
        public void testHTMLStmt() {
                String methodName = "testHTMLStmt";
                System.out.println("Entering " + methodName);
                try {
                        boolean exception = false;
                        PSPParser parser = null;
                        try{
                        parser = getPSPParser("< <HTML><BODY></BODY></HTML>
                        $>");
                        }catch (Exception e) {
                                exception = true;
                        }
                        assertTrue(exception); exception = false;

                        parser = getPSPParser("<HTML><BODY></BODY></HTM>");
                        assertTrue(parser.isCaughtException());

                        parser = getPSPParser("<$ <HTML><BODY></BODY></HTML>
                        $>");
                        assertFalse(parser.isCaughtException());

                        parser = getPSPParser("<$ Any Text $>");
                        assertFalse(parser.isCaughtException());

                        parser = getPSPParser("<$$$$$$$$$>");
                        assertFalse(parser.isCaughtException());
```

```
                  parser =
                        getPSPParser("if(2==3){<$
<HTML><BODY></BODY></HTML> $>} else{<$ <HTML><BODY></BODY></HTML> $>}");
                  assertFalse(parser.isCaughtException());

                  parser =
                        getPSPParser("while(2==3){<$
                  <HTML><BODY></BODY></HTML> $>} String s; null(s){<$
                  <HTML><BODY></BODY></HTML> $>}");
                  assertFalse(parser.isCaughtException());

            } catch (Exception e) {

                  System.out.println(e.getMessage());
                  assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
      }

}
```

PSPTestIfElseStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;

import psp.PSPLexer;
import psp.PSPParser;
import psp.PSPTreeWalker;

import junit.framework.TestCase;

/**
 * Tests all the <b>if else</b> statements supported and not
 * supported by the language.
 * i.e.
 *    if(condition){}
 *    if(condition){}else{}
 *    if(condition){}else if(condition){}
 *    if(condition){}else if(condition)else{}
 *
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestIfElseStmt extends TestCase {
	/**
	 * Constructor for PSPTestRelationExpr.
	 * @param arg0
	 */
	public PSPTestIfElseStmt(String arg0) {
		super(arg0);
	}

	public static void main(String[] args) {
		junit.textui.TestRunner.run(PSPTestIfElseStmt.class);
	}

	/*
	 * This method is called before each test method.
	 * @see TestCase#setUp()
	 */
	protected void setUp() throws Exception {
		super.setUp();
	}

	/*
	 * This method is called after each test method.
	 * @see TestCase#tearDown()
	 */
	protected void tearDown() throws Exception {
		super.tearDown();
	}

	/**
	 * Creates a PSP Parser for parses the given data (psp source)
string.
	 * @param data
	 * @return
	 * @throws IOException
	 * @throws RecognitionException
	 * @throws TokenStreamException
	 */
	public PSPParser getPSPParser(String data)
		throws IOException, RecognitionException,
TokenStreamException {

		ByteArrayInputStream bytesIn =
			new ByteArrayInputStream(data.getBytes());
		DataInputStream input = new DataInputStream(bytesIn);

		//Create the lexer and parser and feed them the input
		PSPLexer lexer = new PSPLexer(input);
		PSPParser parser = new PSPParser(lexer);
		// "program" is the main rule in the parser
		parser.program();
		return parser;
	}

	/**
	 * Test If-Else statements.
	 *
 */
public void testIfElseStmt() {
      String methodName = "testIfElseStmt";
      System.out.println("Entering " + methodName);
      try {

            //Test If statement
            PSPParser parser = getPSPParser("if(){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if(2==3){}");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("if(\"Hello\"){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if(2){}");
            PSPTreeWalker walker = new PSPTreeWalker(parser,
            "tmp.jsp");
            AST parseTree = parser.getAST();
            walker.program(parseTree);
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if(2==3){ print(2);
            print(3); }");
            assertFalse(parser.isCaughtException());

            //Test If Else Statement.
            parser = getPSPParser("else {}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("else if(2==3){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if(2==3){}else {}");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("if(2==3){}else {print(2);
            print(3);}");
            assertFalse(parser.isCaughtException());

            parser =
                  getPSPParser("if(2==3){}else if(2==2){print(2);
            print(3);}");
            assertFalse(parser.isCaughtException());

            parser =
                  getPSPParser("if(2==3){}else if(2==2){print(2);
            print(3);} else{}");
            assertFalse(parser.isCaughtException());

            parser =
                  getPSPParser("if(2==3){}else if(2==2){print(2);
            print(3);} else if(2==3){} else{}");
            assertFalse(parser.isCaughtException());

      } catch (Exception e) {
```

```
                        System.out.println(e.getMessage());
                        assertTrue(false);
                }
                System.out.println("Exiting " + methodName);
        }
}
```

PSPTestInitStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the <b>declaration</b> statements supported and not
 * supported by the language.
 * i.e.
 *    1)    int i = 3;
      2)    String s = "Hello";
      3)    String s = request userName;
      4)    Collection col = session states;
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestInitStmt extends TestCase {

        /**
         * Constructor for PSPTestRelationExpr.
         * @param arg0
         */
        public PSPTestInitStmt(String arg0) {
                super(arg0);
        }

        public static void main(String[] args) {
                junit.textui.TestRunner.run(PSPTestInitStmt.class);
        }

        /*
         * This method is called before each test method.
         * @see TestCase#setUp()
         */
        protected void setUp() throws Exception {
              super.setUp();
        }

        /*
         * This method is called after each test method.
         * @see TestCase#tearDown()
         */
        protected void tearDown() throws Exception {
              super.tearDown();
        }

        /**
         * Creates a PSP Parser for parses the given data (psp source)
  string.
         * @param data
         * @return
         * @throws IOException
         * @throws RecognitionException
         * @throws TokenStreamException
         */
        public PSPParser getPSPParser(String data)
              throws IOException, RecognitionException,
  TokenStreamException {

              ByteArrayInputStream bytesIn =
                    new ByteArrayInputStream(data.getBytes());
              DataInputStream input = new DataInputStream(bytesIn);

              //Create the lexer and parser and feed them the input
              PSPLexer lexer = new PSPLexer(input);
              PSPParser parser = new PSPParser(lexer);
              // "program" is the main rule in the parser
              parser.program();
              return parser;
        }

        /**
         * Test Simple Math Expressions.
         *
         */
        public void testInitIntStmt() {
              String methodName = "testInitIntStmt";
              System.out.println("Entering " + methodName);
              try {

                    PSPParser parser =
                          getPSPParser("int val = \"BartMan\"+5;
                    null(val){}");
                    assertTrue(parser.isCaughtException());

                    parser = getPSPParser("String val = \"Hello\";
                    null(val1){}");
                    assertTrue(parser.isCaughtException());

                    parser = getPSPParser("String val; null(val){}");
```

```
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("String val = \"Hello\";
                null(val){}");
                assertFalse(parser.isCaughtException());

        } catch (Exception e) {

                System.out.println(e.getMessage());
                assertTrue(false);
        }
        System.out.println("Exiting " + methodName);
}

/**
 * Test Simple Math Expressions.
 *
 */
public void testInitStringStmt() {
        String methodName = "testInitStringStmt";
        System.out.println("Entering " + methodName);

        try {

                PSPParser parser =
                        getPSPParser("int val = \"BartMan\"+5;
                notnull(val){}");
                assertTrue(parser.isCaughtException());

                parser = getPSPParser("String val = \"Hello\";
                notnull(val1){}");
                assertTrue(parser.isCaughtException());

                parser = getPSPParser("String val; notnull(val){}");
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("String val = \"Hello\";
                notnull(val){}");
                assertFalse(parser.isCaughtException());

        } catch (Exception e) {
                System.out.println(e.getMessage());
                assertTrue(false);
        }
        System.out.println("Exiting " + methodName);
}

/**
 * Test Simple Math Expressions.
 *
 */
public void testInitCollectionStmt() {
        String methodName = "testInitCollectionStmt";
        System.out.println("Entering " + methodName);

        try {
```

```
				PSPParser parser = getPSPParser("Collection val = 
				\"BartMan\";");
				assertTrue(parser.isCaughtException());
				
				parser = getPSPParser("Collection val;");
				assertTrue(parser.isCaughtException());
				
				parser = getPSPParser("Collection val = session 
				list;");
				assertFalse(parser.isCaughtException());
				
				parser = getPSPParser("Collection val = request 
				list;");
				assertTrue(parser.isCaughtException());
				
			} catch (Exception e) {
				System.out.println(e.getMessage());
				assertTrue(false);
			}
			System.out.println("Exiting " + methodName);
	}
}
```

PSPTestIterStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the <b>iterate</b> statements supported and not
 * supported by the language.
 * i.e.
 *    1)    iterate(Collection){}
      2)    iterate(collection){print(next)};
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestIterStmt extends TestCase {

	/**
	 * Constructor for PSPTestRelationExpr.
```

```
         * @param arg0
         */
        public PSPTestIterStmt(String arg0) {
              super(arg0);
        }

        public static void main(String[] args) {
              junit.textui.TestRunner.run(PSPTestIterStmt.class);
        }

        /*
         * This method is called before each test method.
         * @see TestCase#setUp()
         */
        protected void setUp() throws Exception {
              super.setUp();
        }

        /*
         * This method is called after each test method.
         * @see TestCase#tearDown()
         */
        protected void tearDown() throws Exception {
              super.tearDown();
        }

        /**
         * Creates a PSP Parser for parses the given data (psp source)
  string.
         * @param data
         * @return
         * @throws IOException
         * @throws RecognitionException
         * @throws TokenStreamException
         */
        public PSPParser getPSPParser(String data)
              throws IOException, RecognitionException,
  TokenStreamException {

              ByteArrayInputStream bytesIn =
                    new ByteArrayInputStream(data.getBytes());
              DataInputStream input = new DataInputStream(bytesIn);

              //Create the lexer and parser and feed them the input
              PSPLexer lexer = new PSPLexer(input);
              PSPParser parser = new PSPParser(lexer);
              // "program" is the main rule in the parser
              parser.program();
              return parser;
        }

        /**
         * Tests
         *
         */
        public void testIterStmt() {
              String methodName = "testIterStmt";
```

```
            System.out.println("Entering " + methodName);
            try {

                  PSPParser parser = getPSPParser("iterate (c1)
                  { print(3); print(3);    } ");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("String s1; iterate (s1)
                  { print(3); print(3);} ");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("Collection c1= session afds;
                  iterate (4) { print(3); print(3);} ");
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("Collection c1= session afds;
                  iterate (c1) { print(3); print(3);} ");
                  assertFalse(parser.isCaughtException());

                  parser = getPSPParser(" Collection c1= session afds;
                  iterate (c1){ print(next);  print(3);} ");
                  assertFalse(parser.isCaughtException());


                  } catch (Exception e) {
                  System.out.println(e.getMessage());
                  assertTrue(false);
                  }
            System.out.println("Exiting " + methodName);
      }

}
```

PSPTestNullStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the <b>null and notnull</b> statements supported and not
 * supported by the language.
 * i.e.
 *    nequals(String1, String2){}
 *    equals(String1, String2){}
 *    nequals(int, String2){}
 *
 * if(null(s1) || notnull(s2)){}
 * The null and notnull statments can also be used as relational
 * expressions.
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestNullStmt extends TestCase {

      /**
       * Constructor for PSPTestRelationExpr.
       * @param arg0
       */
      public PSPTestNullStmt(String arg0) {
            super(arg0);
      }

      public static void main(String[] args) {
            junit.textui.TestRunner.run(PSPTestNullStmt.class);
      }

      /*
       * This method is called before each test method.
       * @see TestCase#setUp()
       */
      protected void setUp() throws Exception {
            super.setUp();
      }

      /*
       * This method is called after each test method.
       * @see TestCase#tearDown()
       */
      protected void tearDown() throws Exception {
            super.tearDown();
      }

      /**
       * Creates a PSP Parser for parses the given data (psp source)
string.
       * @param data
       * @return
       * @throws IOException
       * @throws RecognitionException
       * @throws TokenStreamException
       */
      public PSPParser getPSPParser(String data)
            throws IOException, RecognitionException,
TokenStreamException {

            ByteArrayInputStream bytesIn =
                  new ByteArrayInputStream(data.getBytes());
            DataInputStream input = new DataInputStream(bytesIn);
```

```
        //Create the lexer and parser and feed them the input
        PSPLexer lexer = new PSPLexer(input);
        PSPParser parser = new PSPParser(lexer);
        // "program" is the main rule in the parser
        parser.program();
        return parser;
    }

    /**
     * Test.
     *
     */
    public void testNull() {
        String methodName = "testNull";
        System.out.println("Entering " + methodName);
        try {

            PSPParser parser = getPSPParser("null(){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("String s; null(s1){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("String s; null(s){}");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("String s; null(s){print(3);
            print(4);}");
            assertFalse(parser.isCaughtException());

        } catch (Exception e) {

            System.out.println(e.getMessage());
            assertTrue(false);
        }
        System.out.println("Exiting " + methodName);
    }

    /**
     * Test.
     *
     */
    public void testNotNull() {
        String methodName = "testNotNull";
        System.out.println("Entering " + methodName);

        try {

            PSPParser parser = getPSPParser("notnull(){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("String s; notnull(s1){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("String s; notnull(s){}");
            assertFalse(parser.isCaughtException());
				parser = getPSPParser("String s; notnull(s){print(3);
				print(4);}");
				assertFalse(parser.isCaughtException());

			} catch (Exception e) {
				System.out.println(e.getMessage());
				assertTrue(false);
			}
			System.out.println("Exiting " + methodName);
	}

}
```

PSPTestPrintStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;

import psp.PSPLexer;
import psp.PSPParser;
import psp.PSPTreeWalker;

import junit.framework.TestCase;

/**
 * Tests all the <b>print</b> statements supported and not
 * supported by the language.
 * i.e.
 *    print(String1);
 *    print(3);
 *    print(3 == 3); //Invalid
 *
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestPrintStmt extends TestCase {

	/**
	 * Constructor for PSPTestRelationExpr.
	 * @param arg0
	 */
	public PSPTestPrintStmt(String arg0) {
		super(arg0);
	}
	public static void main(String[] args) {
		junit.textui.TestRunner.run(PSPTestPrintStmt.class);
	}

	/*
	 * This method is called before each test method.
	 * @see TestCase#setUp()
	 */
	protected void setUp() throws Exception {
		super.setUp();
	}

	/*
	 * This method is called after each test method.
	 * @see TestCase#tearDown()
	 */
	protected void tearDown() throws Exception {
		super.tearDown();
	}

	/**
	 * Creates a PSP Parser for parses the given data (psp source)
string.
	 * @param data
	 * @return
	 * @throws IOException
	 * @throws RecognitionException
	 * @throws TokenStreamException
	 */
	public PSPParser getPSPParser(String data)
		throws IOException, RecognitionException,
TokenStreamException {

		ByteArrayInputStream bytesIn =
			new ByteArrayInputStream(data.getBytes());
		DataInputStream input = new DataInputStream(bytesIn);

		//Create the lexer and parser and feed them the input
		PSPLexer lexer = new PSPLexer(input);
		PSPParser parser = new PSPParser(lexer);
		// "program" is the main rule in the parser
		parser.program();
		return parser;
	}

	/**
	 * Tests
	 *
	 */
	public void testPrintStmt() {
		String methodName = "testPrintStmt";
		System.out.println("Entering " + methodName);
		try {

			PSPParser parser = getPSPParser("print(3==3);");
			PSPTreeWalker walker = new PSPTreeWalker(parser,
			"tmp.jsp");
```

```
            AST parseTree = parser.getAST();
            walker.program(parseTree);
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("Collection c = session list;
            print(c);");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("print(3);");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("print(\"hello\");");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("String s = \"what\";
            print(s);");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("int i = 3; print(i);");
            assertFalse(parser.isCaughtException());

        } catch (Exception e) {

            System.out.println(e.getMessage());
            assertTrue(false);
        }
        System.out.println("Exiting " + methodName);
    }

    /**
     * Tests
     *
     */
    public void testPrintlnStmt() {
        String methodName = "testPrintlnStmt";
        System.out.println("Entering " + methodName);
        try {

            PSPParser parser = getPSPParser("println(3==3);");
            PSPTreeWalker walker = new PSPTreeWalker(parser,
            "tmp.jsp");
            AST parseTree = parser.getAST();
            walker.program(parseTree);
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("Collection c = session list;
            println(c);");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("println(3);");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("println(\"hello\");");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("String s = \"what\";
            println(s);");
```

```
                assertFalse(parser.isCaughtException());

                parser = getPSPParser("int i = 3; println(i);");
                assertFalse(parser.isCaughtException());

            } catch (Exception e) {

                System.out.println(e.getMessage());
                assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
        }
}
```

PSPTestRelationExpr.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;

import psp.PSPLexer;
import psp.PSPParser;

import junit.framework.TestCase;

/**
 * Tests all the <b>relational and math</b> expressions supported and
not
 * supported by the language.
 * i.e.
 *    (3+4/3)
 *    (3 == 3)
 *
 *
 * Relational expressions by themselves will not be accepted by the
parser so they
 * are defined in and int assignment statment.  Of the form:
 * ID ASSIGN relationExpr SEMI
 * i.e.  val = 3+3;
 *
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestRelationExpr extends TestCase {

    /**
```

```
	 * Constructor for PSPTestRelationExpr.
	 * @param arg0
	 */
	public PSPTestRelationExpr(String arg0) {
		super(arg0);
	}

	public static void main(String[] args) {
		junit.textui.TestRunner.run(PSPTestRelationExpr.class);
	}

	/*
	 * This method is called before each test method.
	 * @see TestCase#setUp()
	 */
	protected void setUp() throws Exception {
		super.setUp();
	}

	/*
	 * This method is called after each test method.
	 * @see TestCase#tearDown()
	 */
	protected void tearDown() throws Exception {
		super.tearDown();
	}

	/**
	 * Creates a PSP Parser for parses the given data (psp source)
string.
	 * @param data
	 * @return
	 * @throws IOException
	 * @throws RecognitionException
	 * @throws TokenStreamException
	 */
	public PSPParser getPSPParser(String data)
		throws IOException, RecognitionException,
TokenStreamException {

		ByteArrayInputStream bytesIn =
			new ByteArrayInputStream(data.getBytes());
		DataInputStream input = new DataInputStream(bytesIn);

		//Create the lexer and parser and feed them the input
		PSPLexer lexer = new PSPLexer(input);
		PSPParser parser = new PSPParser(lexer);
		// "program" is the main rule in the parser
		parser.program();
		return parser;
	}

	/**
	 * Test Simple Math Expressions.
	 *
	 */
	public void testMathExpr1() {
        String methodName = "testMathExpr1";
        System.out.println("Entering " + methodName);
        try {
              StringBuffer sb = new StringBuffer("int val = 4+5;");
              sb.append("val = 91*321;");
              sb.append("val = 812/22;");
              sb.append("val = 812-22;");

              PSPParser parser = getPSPParser(sb.toString());
              assertFalse(parser.isCaughtException());

        } catch (Exception e) {
              System.out.println(e.getMessage());
              assertTrue(false);
        }
        System.out.println("Exiting " + methodName);
  }

  /**
   * Test Simple Math Expressions.
   *
   */
  public void testMathExpr2() {
        String methodName = "testMathExpr2";
        System.out.println("Entering " + methodName);

        try {
              PSPParser parser = getPSPParser("int val =
              \"BartMan\"+5;");
              assertTrue(parser.isCaughtException());

              parser = getPSPParser("int val = val2+5;");
              assertTrue(parser.isCaughtException());

        } catch (Exception e) {
              System.out.println(e.getMessage());
              assertTrue(false);
        }
        System.out.println("Exiting " + methodName);
  }

  /**
   * Test Simple Math Expressions.
   *
   */
  public void testMathExpr3() {
        String methodName = "testMathExpr3";
        System.out.println("Entering " + methodName);
        try {

              PSPParser parser = getPSPParser("int val =
              ((4+5)*3)+4;");
              assertFalse(parser.isCaughtException());

              parser = getPSPParser("int val = (91*321)/(5-3);");
              assertFalse(parser.isCaughtException());
			parser = getPSPParser("int val = ((8+2)-(22*3))/8;");
			assertFalse(parser.isCaughtException());

			//Division by zero is a PSP runtime exception. It
should pass
			//through the parser succesfully.

			parser = getPSPParser("int val = (1+6)/0;");
			assertFalse(parser.isCaughtException());

		} catch (Exception e) {
			System.out.println(e.getMessage());
			assertTrue(false);
		}
		System.out.println("Exiting " + methodName);
	}

	/**
	 * Test Relational Expressions.
	 *
	 */
	public void testRelationExpr() {
		String methodName = "testRelationExpr";
		System.out.println("Entering " + methodName);
		try {

			PSPParser parser = getPSPParser("if(4 == 3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(4 != 3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(4 > 3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(4 >= 4){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(4 < 3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(4 <= 3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(4+3 == 8*3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if((4/3)*2 <= 3){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if((3+4) <= (1-2)){}");
			assertFalse(parser.isCaughtException());

			parser = getPSPParser("if(((3+4) <= (1-2)) && 4
==3){}");
			assertFalse(parser.isCaughtException());
```

```
            parser = getPSPParser("if(((3+4) <= (1-2)) || (4+1
            ==3)){}");
            assertFalse(parser.isCaughtException());

            parser =
                  getPSPParser("if(((3+4) <= (1-2)) || (4+1 ==3)
            && (1 ==1)){}");
            assertFalse(parser.isCaughtException());

           parser = getPSPParser("String s; if(4 != null(s)){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("String s; if(4 != 3 &&
            null(s)){}");
            assertFalse(parser.isCaughtException());

      } catch (Exception e) {
            System.out.println(e.getMessage());
            assertTrue(false);
      }
      System.out.println("Exiting " + methodName);
}

/**
 * Test Simple Math Expressions.
 *
 */
public void testRelationExpr2() {
      String methodName = "testRelationExpr2";
      System.out.println("Entering " + methodName);
      try {
            PSPParser parser = getPSPParser("if(4 = 3){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if 4 != 3){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if(4  3){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if(4 >= ){}");
            assertTrue(parser.isCaughtException());

            parser = getPSPParser("if( >= 3){}");
            assertTrue(parser.isCaughtException());

            //This should be rejected by the PSPWalker but it is
            accepted by the PSPParser.
            parser = getPSPParser("if((4/3)*2){}");
            assertFalse(parser.isCaughtException());

            parser = getPSPParser("if(((3+4) <= (1-2)) && i ==
            i2){}");
            assertTrue(parser.isCaughtException());

            parser =
                        getPSPParser("int i = 1; int i2 = 2; if(((3+4)
                  <= (1-2)) && i == i2){}");
                  assertFalse(parser.isCaughtException());

            } catch (Exception e) {
                  System.out.println(e.getMessage());
                  assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
      }

}
```

PSPTestWhileStmt.java

```
/*
 * Created on Nov 23, 2004
 */
package psp.tests;

import java.io.*;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;

import psp.PSPLexer;
import psp.PSPParser;
import psp.PSPTreeWalker;

import junit.framework.TestCase;

/**
 * Tests all the <b>while</b> statements supported and not
 * supported by the language.
 * i.e.
 *    while(condition){}
 *
 * @author ONeil Palmer
 *         op2007@columbia.edu
 */
public class PSPTestWhileStmt extends TestCase {

      /**
       * Constructor for PSPTestRelationExpr.
       * @param arg0
       */
      public PSPTestWhileStmt(String arg0) {
            super(arg0);
      }

      public static void main(String[] args) {
            junit.textui.TestRunner.run(PSPTestWhileStmt.class);
      }
```

```
        /*
         * This method is called before each test method.
         * @see TestCase#setUp()
         */
        protected void setUp() throws Exception {
                super.setUp();
        }

        /*
         * This method is called after each test method.
         * @see TestCase#tearDown()
         */
        protected void tearDown() throws Exception {
                super.tearDown();
        }

        /**
         * Creates a PSP Parser for parses the given data (psp source)
string.
         * @param data
         * @return
         * @throws IOException
         * @throws RecognitionException
         * @throws TokenStreamException
         */
        public PSPParser getPSPParser(String data)
                throws IOException, RecognitionException,
TokenStreamException {

                ByteArrayInputStream bytesIn =
                        new ByteArrayInputStream(data.getBytes());
                DataInputStream input = new DataInputStream(bytesIn);

                //Create the lexer and parser and feed them the input
                PSPLexer lexer = new PSPLexer(input);
                PSPParser parser = new PSPParser(lexer);
                // "program" is the main rule in the parser
                parser.program();
                return parser;
        }

        /**
         * Tests
         *
         */
        public void testWhileStmt() {
                String methodName = "testWhileStmt";
                System.out.println("Entering " + methodName);
                try {

                        PSPParser parser = getPSPParser("while(){}");
                        assertTrue(parser.isCaughtException());

                        parser = getPSPParser("while(3){}");
                        PSPTreeWalker walker = new PSPTreeWalker(parser,
                        "tmp.jsp");
```

```
                  AST parseTree = parser.getAST();
                  walker.program(parseTree);
                  assertTrue(parser.isCaughtException());

                  parser = getPSPParser("while(3==4){}");
                  assertFalse(parser.isCaughtException());

                  parser = getPSPParser("String s; while(null(s)){}");
                  assertFalse(parser.isCaughtException());

                  parser = getPSPParser("String s;
                  while(notnull(s)){}");
                  assertFalse(parser.isCaughtException());

                  parser =
                        getPSPParser("String s; while(null(s) &&
                  notnull(s) || equals(s,s) || nequals(s,s)  ){}");
                  assertFalse(parser.isCaughtException());

            } catch (Exception e) {

                  System.out.println(e.getMessage());
                  assertTrue(false);
            }
            System.out.println("Exiting " + methodName);
      }
}
```

Makefile

```
# Makefile used in tutorial
.PHONY :	all clean build run test createjar zip

DIR_PSP = psp/
DIR_PARSER = psp/parser/
DIR_EXC = psp/parser/exception/
DIR_WALKER = psp/walker/
DIR_TESTS = psp/tests/
DIR_EXAMPLES = psp/examples/

GRAMMAR = $(DIR_PSP)PSPVocabTokenTypes.java
$(DIR_PSP)PSPVocabTokenTypes.txt $(DIR_PSP)PSPLexer.java
$(DIR_PSP)PSPParser.java

WALKER = $(DIR_PSP)PSPTreeWalker.java
$(DIR_PSP)PSPTreeWalkerTokenTypes.java
$(DIR_PSP)PSPTreeWalkerTokenTypes.txt


CLASSES = $(DIR_PSP)PSPLexer.class $(DIR_PSP)PSPMain.class
$(DIR_PSP)PSPParser.class \
	$(DIR_PSP)PSPTreeWalker.class $(DIR_PSP)PSPTreeWalkerTokenTypes.class
\
	$(DIR_PSP)PSPVocabTokenTypes.class \
	$(DIR_PARSER)SymbolTable.class $(DIR_PARSER)SymbolScope.class \
	$(DIR_EXC)PSPException.class $(DIR_EXC)PSPParsingException.class \
	$(DIR_EXC)PSPWalkerException.class \
	$(DIR_WALKER)JSPGenerator.class $(TESTCLASSES)

TESTCLASSES = $(DIR_TESTS)PSPTestAssignmentStmt.class
$(DIR_TESTS)PSPTestComments.class $(DIR_TESTS)PSPTestEqualsStmt.class \
	$(DIR_TESTS)PSPTestForStmt.class $(DIR_TESTS)PSPTestHTML.class \
	$(DIR_TESTS)PSPTestIfElseStmt.class \
	$(DIR_TESTS)PSPTestInitStmt.class $(DIR_TESTS)PSPTestIterStmt.class \
	$(DIR_TESTS)PSPTestNullStmt.class $(DIR_TESTS)PSPTestPrintStmt.class \
	$(DIR_TESTS)PSPTestRelationExpr.class  $(DIR_TESTS)PSPTestSuite.class\
	$(DIR_TESTS)PSPTestWhileStmt.class

all :	$(DIR_PSP)PSPLexer.class $(DIR_PSP)PSPMain.class
$(DIR_PSP)PSPParser.class \
	$(DIR_PSP)PSPTreeWalker.class $(DIR_PSP)PSPTreeWalkerTokenTypes.class
\
	$(DIR_PSP)PSPVocabTokenTypes.class \
        $(DIR_PARSER)SymbolTable.class $(DIR_PARSER)SymbolScope.class \
        $(DIR_EXC)PSPException.class $(DIR_EXC)PSPParsingException.class \
        $(DIR_EXC)PSPWalkerException.class \
        $(DIR_WALKER)JSPGenerator.class  \
        $(DIR_TESTS)PSPTestAssignmentStmt.class
$(DIR_TESTS)PSPTestComments.class $(DIR_TESTS)PSPTestEqualsStmt.class \
        $(DIR_TESTS)PSPTestForStmt.class $(DIR_TESTS)PSPTestHTML.class \
        $(DIR_TESTS)PSPTestIfElseStmt.class \
        $(DIR_TESTS)PSPTestInitStmt.class $(DIR_TESTS)PSPTestIterStmt.class \
        $(DIR_TESTS)PSPTestNullStmt.class $(DIR_TESTS)PSPTestPrintStmt.class \
        $(DIR_TESTS)PSPTestRelationExpr.class  $(DIR_TESTS)PSPTestSuite.class\
        $(DIR_TESTS)PSPTestWhileStmt.class

run :   build
        java -classpath .:./antlr.jar $(DIR_PSP)PSPMain $(wildcard *.psp)

examples :      build
        java -classpath .:./antlr.jar $(DIR_PSP)PSPMain $(wildcard
$(DIR_EXAMPLES)*.psp)

test :  build
        java -classpath .:./antlr.jar:./junit.jar $(DIR_TESTS)PSPTestSuite

build : grammar
        make walker
        make all

clean :
        rm -f $(DIR_PSP)*.class
        rm -f $(DIR_PARSER)*.class
        rm -f $(DIR_EXC)*.class
        rm -f $(DIR_WALKER)*.class
        rm -f $(DIR_TESTS)*.class

distclean :     clean
        rm -f $(GRAMMAR)
        rm -f $(WALKER)

$(DIR_PSP)%.class :  $(DIR_PSP)%.java
        javac -classpath .:./antlr.jar:./junit.jar $<

grammar :       $(DIR_PSP)PSP.g
        rm -f $(GRAMMAR)
        java -classpath ./antlr.jar antlr.Tool -o $(DIR_PSP) $<

walker :        $(DIR_PSP)PSPwalker.g
```

```
	rm -f $(WALKER)
	java -classpath ./antlr.jar antlr.Tool -o $(DIR_PSP) $<

jar :	build
	jar cvf psp.jar $(CLASSES) $(TESTCLASSES)

createjar :
	jar cvf psp.jar $(CLASSES) $(TESTCLASSES)

zip :	examples
	make createjar
	zip -r pspsrc`date +%m%d%y`.zip $(wildcard $(DIR_PSP)*.java
$(DIR_PARSER)*.java $(DIR_EXC)*.java $(DIR_WALKER)*.java
$(DIR_TESTS)*.java $(DIR_EXAMPLES)*.psp $(DIR_EXAMPLES)*.jsp
$(DIR_PSP)*.g) *.jar Makefile
```

Sample .psp files

Helloworld.psp

```
<$
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>

<head> <title>Hello World</title> </head>
<body bgcolor=#FFFFFF>
<font face="Helvetica">
<h2>
<font color=#DB1260>
Hello World
</font>
</h2>
$>
print("<p><b>Hello World!</b>");
<$ <hr> <p></font></body></html> $>
```

Helloworld2.psp

```
/* will display "PSP HelloWorld Page" or "Hello [some name]" page */
 <$    <html><head>   $>
String name = request userName;
if(null(name)){
    <$ <h2>PSP Hello World Page</ h2>  $>
}else{
    <$   <h2> Hello $>  print(name); <$ Page </h2>  $>
}
<$ </head > </html>  $>
```

Example1.psp

```
<$
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>
<head><title>example1</title></head>
<body bgcolor=#FFFFFF><font face="Helvetica">
<h2><font color=#DB1260>List: </font></h2>
<table> <tr>
$>
       int i = 3;
       String name = "O'Neil";

<$    <td> Id </td> <td> Name</td> </tr>  $>
      for(int x=0;x < i; x++){
         print("<tr>");
            print("<td>"); print(x); print("</td>");
            print("<td>"); print("Enter Name"); print("</td>");
            print("</tr>");
      }
<$ <hr></font></body></html> $>
```

Example2.psp

```
<$
<%@page import="java.util.ArrayList"  %>
<%
      //This is a hack.  But it allows me to test
      //the iterate PSP statment.
      ArrayList list = new ArrayList();
      list.add("ONeil");
      list.add("Bart");
      list.add("List");
      session.setAttribute("names",list);
%>
<html>
<head><title>example2</title></head>
<body bgcolor=#FFFFFF><font face="Helvetica">
<h2><font color=#DB1260>List Names</font></h2>
<table>
      <tr>  <td> Id </td><td> Name</td> </tr>
$>
      Collection listOfNames = session names;
      int x;
      iterate(listOfNames){
         print("<tr>");
            print("<td>"); print(x); print("</td>");
            print("<td>"); print(next); print("</td>");
            print("</tr>");
            x++;
      }
<$ <hr></font></body></html>  $>
```

Welcome.psp

```
//Get the username from the request if it exists
String userName = request userName;

if(notnull(userName)){
      //adds the username to the http session
      session userName = userName;
}else {
      //Checks if the username is not on the session then
      //initialize it to the empty string.
      userName = session userName;
      if(null(userName)){
            userName = "";
      }
}

<$ <!-- tpl:insert page="/theme/A_blue_.htpl" --><!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<meta http-equiv="Content-Style-Type" content="text/css">
<link rel="stylesheet" href="/psp/theme/blue.css" type="text/css">
<!-- tpl:put name="headarea" -->
<title>welcome</title>
<!-- /tpl:put -->
</head>
<body>
<table width="760" cellspacing="0" cellpadding="0" border="0">
   <tbody>
      <tr>  <td valign="top">
         <table class="header" cellspacing="0" cellpadding="0"
border="0" width="100%">
            <tbody>
               <tr> <td width="150"><img border="0" width="150"
height="55" alt="Company's LOGO" src="/psp/theme/logo_blue.gif"></td>
                  <td></td> </tr>
            </tbody>
         </table>
         </td>
      </tr>
      <tr><td valign="top" class="nav_head" height="20"></td> </tr>
      <tr class="content-area">
            <td valign="top" height="350"><!-- tpl:put name="bodyarea"
--><h2>Welcome
         $>
         print(userName);
         if(equals(userName,"op2007") ||equals(userName,"bart") ||
equals(userName,"lisa") ){
         <$ </h2>
   <FORM action="/psp/displayInfo.jsp" method="post"><TABLE border="1">
      <TBODY> <TR> <TD>
     <a href="/psp/personalInfo.jsp"> Enter Personal Info</a></TD></TR>
    <TR> <TD><a href ="/psp/logout.jsp">logout</a></TD>
</TR></TBODY></TABLE></FORM> $> }else {
<$ <BR>     INVALID Username/Passsword <br>
        <a href="/psp/login.html">retry</a> </h2>
$>
  } <$  <!-- /tpl:put --></td>    </tr>   <tr>
         <td valign="top" height="20" class="footer"></td>
      </tr>
   </tbody>
</table>
</body>
</html><!-- /tpl:insert -->$>
```

PersonalInfo.psp

```
String userName = session userName;
<$<!-- tpl:insert page="/theme/A_blue_.htpl" --><!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<meta http-equiv="Content-Style-Type" content="text/css">
<link rel="stylesheet" href="/psp/theme/blue.css" type="text/css">
<!-- tpl:put name="headarea" -->
<title>personalInfo</title>
<!-- /tpl:put -->
</head>
<body>
<table width="760" cellspacing="0" cellpadding="0" border="0">
   <tbody><tr><td valign="top">
   <table class="header" cellspacing="0" cellpadding="0" border="0"
width="100%">
    <tbody><tr> <td width="150"><img border="0" width="150" height="55"
alt="Company's LOGO" src="/psp/theme/logo_blue.gif"></td> <td></td>
       </tr></tbody></table></td></tr><tr>
         <td valign="top" class="nav_head" height="20"></td>
      </tr>
      <tr class="content-area">
         <td valign="top" height="350"><!-- tpl:put name="bodyarea" --
><h2>Personal Information: $>
         print(userName);  <$</h2>
 <FORM action="/psp/displayInfo.jsp" method="post"><TABLE border="1">
      <TBODY><TR><TD>First Name:</TD><TD width="227">
      <INPUT type="text" name="fname" size="20" maxlength="20"></TD>
      </TR><TR><TD>Last Name</TD> <TD width="227">
      <INPUT type="text" name="lname" size="20" maxlength="20"></TD>
      </TR><TR><TD>DOB</TD><TD width="227">
      <INPUT type="text" name="dob" size="20" maxlength="20"></TD>
      </TR><TR><TD>SSN</TD><TD width="227">
      <INPUT type="text" name="ssn" size="20" maxlength="20"></TD>
      </TR><TR><TD>State</TD><TD width="227">
      <INPUT type="text" name="state" size="20"maxlength="20"></TD>
      </TR><TR>   <TD colspan="2" align="center">
```

```
<TABLE border="1">
<TBODY><TR><TD width="10">
  <INPUT type="submit" name="submit" value="Submit"></TD>
  <TD width="10"><INPUT type="reset" value="Reset"></TD>
 </TR></TBODY></TABLE></TD></TR></TBODY>
      </TABLE>
</FORM>
      <BR>
<h3><a href="/psp/welcome.jsp">home</a></h3><!-- /tpl:put --></td>
      </tr>
      <tr>
         <td valign="top" height="20" class="footer"></td>
      </tr>
   </tbody>
</table>
</body>
</html><!-- /tpl:insert --> $>
```

DisplayInfo.psp

```
String userName = session userName;
<$
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>
<head> <title>displayInfo</title> </head>
<body bgcolor=#FFFFFF>
<font face="Helvetica">
<h2> Personal Information:$> print(userName); <$</h2>
$>

String fname = request fname;
String lname = request lname;
String dob = request dob;
String ssn = request ssn;
String state = request state;
<$
<table>
      <tr> <td>Key</td> <td>Value</td> </tr>
      <tr> <td>$>print("fname");<$</td> <td>$>print(fname);<$</td>
</tr>$>
      <$<tr> <td>$>print("lname");<$</td> <td>$>print(lname);<$</td>
</tr>$>
      <$<tr> <td>$>print("dob");<$</td> <td>$>print(dob);<$</td>
</tr>$>
      <$<tr> <td>$>print("ssn");<$</td> <td>$>print(ssn);<$</td>
</tr>$>
      <$<tr> <td>$>print("state");<$</td> <td>$>print(state);<$</td>
</tr>
</table>
$>

<$
<a href="/psp/welcome.jsp"><h3>Home</h3></a>
<hr></font></body></html>
$>
```

Logout.psp

```
//remove the user name from the session
session userName = null;

<$ <!-- tpl:insert page="/theme/A_blue_.htpl" --><!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<meta http-equiv="Content-Style-Type" content="text/css">
<link rel="stylesheet" href="/psp/theme/blue.css" type="text/css">
<!-- tpl:put name="headarea" -->
<title>logout</title>
<!-- /tpl:put -->
</head>
<body>
<table width="760" cellspacing="0" cellpadding="0" border="0">
   <tbody>
      <tr>
         <td valign="top">
         <table class="header" cellspacing="0" cellpadding="0"
border="0" width="100%">
            <tbody>
               <tr>
                  <td width="150"><img border="0" width="150"
height="55" alt="Company's LOGO" src="/psp/theme/logo_blue.gif"></td>
                  <td></td>
               </tr>
            </tbody>
         </table>
         </td>
      </tr>
      <tr>
         <td valign="top" class="nav_head" height="20"></td>
      </tr>
      <tr class="content-area">
         <td valign="top" height="350"><!-- tpl:put name="bodyarea" -->
                            <BR>
                      <h2>Thanks for visiting PSP Land!</h2>
         <BR> <BR>
                      <h3><a href="/psp/login.html">login</a></h2>
                      <!-- /tpl:put --></td>
      </tr>
      <tr>
         <td valign="top" height="20" class="footer"></td>
      </tr>

   </tbody>
</table>
</body>
</html><!-- /tpl:insert --> $>
```