PSP

Language Reference Manual

October 21, 2004

Jide Atoyebi (oea5@columbia.edu)
Cody Hess (cjh2011@columbia.edu)
O'Neil Palmer (op2007@columbia.edu)
Neil Sarkar (neilSarkar66@gmail.com)

Table of Contents:

1 – Introduction

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' Examples

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

2.2 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 ;* */**

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'. To define a collection:

Collection ***identifier*** *;*
identifier *= session attrname;*

Also, not that declaration and assignment for all the variables can be done in one statement, i.e.

int x = 7 */* x is 7 */*

2.3 Indentifiers

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, underscores, or digits.

Hence the following are valid identifiers:

foobar, Edwards, Moosejaw_Saskatchewan

and the following are invalid:

95_Cowboys, boston- red- sox, WhizBang!

2.4 Operators

The PSP language supports the following operators, which will be elaborated upon in the next section:

&&	||	+	++
=	==	<	<=
>	>=	-	--
*	/		

3 – Operators

The following operators 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 */
```

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 */
```

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 */
```

3.15 Negation '!'

The negation operator will reverse the Boolean value of any expression it is applied to.

```
if( !(17 == 17) ) {
   print( "Negation in yo FACE." );/* prints "Negation in yoFACE." */
}
```

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 statements 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** ;*

```
Collection states = session stateList;
```

4.5 request

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

r*equest **attribute** ;*

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

4.6 print

The print statement simply prints strings – String variables and string constants – 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.

```
/* a complete HTML header for a PSP page */
<$
      <html>
      <head>
      <title>PSP Hello World 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("<T>R");
   print(name);
   print("</TR>");
}
```

5 – Syntax

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

Comments

Comments are ignored by the compiler. Everything that is between the tokens /* and */ is a comment.

```
/* This is a comment */
```

End of Line

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

```
String lever = "2000";   /* note the semicolon */
```

Scope

Blocks of code are identified with curly braces: { and }. For loops, while loops, and if statements use this to lay claim to a their block of code.

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

6 – Example Program

```
<$
      <html>
      <head>
      <title>Password Verification Error</title>
      </head>
      <body>
$>

String name = request userName;
String pass = request passWord;

print (“Thank you for trying to log in, <strong>” + name);
print( “</strong>, but your password, <strong>”);
print(pass + “</strong>, was incorrect”);

<$
      </body>
      </html>
$>
```