LRM: SHIL

Simulated Human Input Language

COMS W4115 Programming Languages and Translators

Fall 2008

Moses Vaughan (mjv2123@columbia.edu)

Binh Vo (bdv2112@columbia.edu)

Ian Vo (idv2101@columbia.edu)

Chun Yai Wang (cw2244@columbia.edu)

Table of Contents:

Introduction

- An Overview of Simulated Human Interaction Language(SHIL)

SHIL is a language used primarily for developing HTML based automated bots. It provides the developer with an abstraction for automating interaction with web sites and users. From the server's perspective SHIL can be used to simulate user interactions, which is useful for many applications ranging from creating spiders to website test scripts. From the user's perspective SHIL can be used to implement custom user interfaces. In conjunction with automated server interaction this potentially can be used to alter existing interfaces for websites or provide interfaces to additional functionality built on top of existing website functionality.

One of the main motivations for the production of our language are that many automated browsing tasks are written now in various languages, primarily PERL and Python. For example many services such as web search engines need to crawl across existing pages on the internet, or independent users often wish to automate data collection over various sites. SHIL intends to provide a language designed specifically for this task which will reduce the complexity of writing applications of this nature.

Lexical Conventions

- Overview

This section covers the lexical conventions within the SHIL language that constitute various tokens including elements such as data types, data structures, reserved words and symbols. A token is a series of contiguous characters that the compiler will treat as one individual element. The scanner will parse tokens to be the longest string of characters that can create a token type.

- White Space

White space is classified only by blank spaces, newlines, tabs, and within the scanner comments are considered whitespace. The only purpose of whitespace is to separate tokens, and can essentially be rendered useless except for human readability issues.

- Comments

/* is the opening of a comment and */ is the closing the respective block. There are no single line only comment lexemes. Once a comment opening is seen, everything up until the end of the */ lexeme is considered invisible to the compiler.

- Identifier

An identifier is a sequence of alphanumeric and non-alphanumeric characters. Note that the first character can be anything other than a digit. Casing is distinctive in all positions of an identifier string, meaning that one identifier is not equivalent to another unless they both follow identical character order and their characters must have identical casing in their respective positions.

- Keywords

The following list is the keywords within the SHIL language. They cannot be used for any programmatic purpose other than their distinct function.

integer	real	boolean
struct	map	array
if	while	foreach
break	end	fun
use	return	true
false	maybe	

- Operators

An operator is used to specify an operation to be performed. The chart below gives operators as well as their necessary functions.

Operator	*Functions*
<=	assignment
+ - / *	math
"	string**
;	statement termination
.	struct reference
[ ]	array reference
( )	Logical grouping
& | ! =	Boolean
< > >=	Operators

Note: ** signifies that this operator must occur in a pairing

Data Types and Structures

- Data Types

String - *string* is any finite sequence of characters which include letters, numerals, symbols and punctuation marks. A “ is used to signify the beginning of a string and an additional one used to signify the end of the string.

Integer - An integer is a whole number that can be positive, negative, or zero.

Real - Real numbers include rational and irrational numbers, but must be signified in decimal format within SHIL. So therefore pi is not an acceptable value.

Boolean - Boolean represents logical variables and can be of the values true or false.

- Data Structures

Struct - A type which can hold a grouping of variables within one variable. To reference the individual variables within a struct you must have the formatting:

<Struct Name>.<Variable Name>

Map – An associated array which holds key value pairings. The operation of finding a value with a given key is called a lookup.

Array - A linear data structure where each element holds the same data type. The structure itself occupies a contiguous block of storage.

Functions

SHIL functions can take multiple arguments, return either a single basic data type value or nothing, and modify multiple existing values

```
function (type1 arg1, type2 arg2 …) => <return_type> {
        /* arbitrary code */
};
```

Args are passed by value (with the exception of data structures). The return value is specified with the 'return' operator.

For example, the following function can return a sum:

```
sum <= function (integer x, integer y) => integer {
        return x + y;
};
```

User-defined libraries of functions can be stored in a separate file and included

with the 'use' directive:

use “filename”;

Expressions

Expressions are token groups which result in a value, they fall into several categories.

Constants: result <= “string”; result <= 1; result <= 1.0; result = true;

String constants are always surrounded by double quotes. Digits containing a '.' are real valued, otherwise integer-valued, and booleans are one of either 'true' or 'false'.

Function Calls: result <= func_name(arg1, arg2 => ref1, ref2);

Function calls return at most one value, not necessarilly related to the *arguments* passed by reference.

Math/boolean operators: result <= val1 <operator> val2;

Boolean operators ('&', '|', '!') always return a boolean value. Mathematical operators ('+', '-', '*', '/', '%') return a real value if either of the input values is real, and an integer value otherwise.

Comparison operators: result <= “asdf” = “asdf”;

Comparison operators ('=', '!=', '<' , '>', '<=', '>=') return a boolean value from two values of same type.

Assignment: result <= foo <= bar;

Assignment operators also return the value assigned.

Statements

Statements are complete SHIL instructions, are terminated by the ';' character, and fall into four categories:

Expression

<expression>;

A lone expression may be evaluated as a statement.

Block

{<statement>; <statement>; <statement>;};

Curly braces can be used to group several statements into one statement.

Conditional

if <expression> then <statement>; else <statement>;

'if' can be used with a boolean-valued expression to execute one of two statements.

Iteration

while <expression> <statement>;

'while' can be used with a boolean-valued expression to repeatedly execute a statement.

foreach <key_name> <value_name> in <array_or_map_name> statement;

'foreach' can be used to execute a statement once for each element in a map or array. key_name and value_name will become variables within the context of this statement. For an array, key_name is an integer index from 0 to the length of the array, and for a map, key_name is the key of the map.

Return

return <expression>;

Within a function body, this can be used to terminate execution of the function and return a value.

Namespace

Variable names and function names will occupy the same case sensitive namespace, and can be assigned with the <= operator. Function bodies will use a private namespace, and statement blocks will inherit the parent namespace, however any variables declared within the block will expire with the termination of the block.

SHIL specific functions

The SHIL language has a number of built-in functions that are always available. They are categorized here according to purpose:

Internet Interaction

*string <= (**SendRequest** map)*

Returns the HTML result as a string, given an HTTP request of type map.

*map <= (**ParseHTML** string)*

Returns a nested map representation of an HTML page in string format.

*string <= (**GenerateHTML** map)*

Returns a string representation of an HTML page in map format.

*map <= (**ShowHTML** string)*

Displays a given HTML code of string in the user's default web browser and returns the next HTTP request as a map.

String Manipulation

*string <= (**SubString** string int int)*

Returns a string that is the substring of the given string, bounded by the starting and ending integer indexes.

*int <= (**StringLength** string)*

Returns the length of the given string.

*int <= (**StringFind** string string)*

Returns the index within the given string of the first occurrence of the specified substring.

*Int[ ] <= (**StringFindAll** string string)*

Returns an array of indexes within the given string of all occurrences of the specified substring.

*String[ ] <= (**SplitString** string string)*

Returns an array of strings resulting from splitting the given string according to a regular expression provided in string format.

*string <= (**StringReplace** string string string)*

Replaces all instances of the search string with the replacement string in the given string.

*string <= (**StringToUpper** string)*

Returns the given string with all its characters converted to upper case using the rules of the default locale.

*string <= (**StringToLower** string)*

Returns the given string with all its characters converted to lower case using the rules of the default locale.

Data Manipulation

*array <= (**Sort** array)*

Returns a sorted version of the provided array.

*array <= (**Randomize** array)*

Returns a randomized permutation of the provided array.

*array <= (**GetKeys** map)*

Returns an array of the keys for the provided map.

*array <= (**GetValues** map)*

Returns an array of the values for the provided map.

*int <= (**Length** array)*

Returns the number of elements for the provided array.

*arraytype <= (**Aggregate** array, fun(arraytype, sometype -> sometype), sometype)*

Aggregate all the values of the provided array with the given function.

*Array <= (**Modify** array, fun(arraytype -> arraytype))*

Apply a function to every element in the provided array.