(GROUP LEADER) - Rui Kuang (rkuang@cs.columbia.edu)
Andrey Butov (ab2301@columbia.edu, andreybutov@yahoo.com)
Arvid Bessen (ajb2103@columbia.edu, bessen@cs.columbia.edu)
Svetlana Starshinina (svs29@columbia.edu)

WHOM
Language Reference Manual

1. Introduction

The WHOM language is designed to allow developers to simulate the creation and management of a household. Using WHOM, it is possible to define a household as well as a set of rules to govern the automatic operations of that household. WHOM is designed to be a simple but powerful, object-oriented, event-driven, flexible, and robust language.

2. Lexical Conventions

There are six kinds of tokens: identifiers, keywords, constants, expressions, operators, and other separators. In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers and constants
If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters, which could possibly constitute a token.

2.1 Comments

The characters /* start a comment terminated by the first */, while the characters // start a single-line comment that runs until the end of the line.

2.2 Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. The underscore ‘‘_’’ counts as alphabetic. Upper and lower case letters are considered different.

2.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
number      string
while       group
return      realtime
once        class
void        extends
if          import
else        put
once        in
for
any
```

2.4 Constants
There are two types of constants in WHOM: numbers and strings.

2.4.1 Number
A number consists of an integer part, a decimal point and a fraction part. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the decimal point and the fraction part (not both) may be missing.

2.4.2 String
A string is a sequence of characters surrounded by double quotes ‘‘ " ’’. A double quote inside a string is represented by two consecutive double quotes

2.5 Other tokens
```
{       }       (       )       ,       ;
+       -       *       /       =
>=      <=      ==      !=      >       <
|       !       &       .
```

3. Expressions
The WHOM expressions follow the standard precedence and associativity rules.

3.1 Primary expressions
Primary expressions include identifiers, constants and parenthesized expressions.

3.1.1 Identifiers
An identifier itself could be a left-value expression or a right-value expression. It if it is a right value expression it is evaluated.

3.1.2. Constants
A constant, whether it is a number or a string, is a right-value expression. It can be evaluated either directly to a constant itself.

3.1.3 Parenthesized expressions
Parenthesized expression is a primary expression. Parentheses take precedence over all other operators.

3.2 Arithmetic expressions
Arithmetic expressions take primary expressions as operands and evaluate them using operators

3.2.1. Unary operators
Unary operators + and – can be prefixed to an expression. The + operator returns the expression itself whereas the – operator returns the negative of the primary expression. These operators are applicable to expressions which evaluate to type number.

3.2.2 Binary operators
Binary operators +, -, *, / indicate addition, subtraction, multiplication, and division, respectively. Multiplication and division take precedence over addition and subtraction. These operators are applicable to expressions which evaluate to type number.

3.3 Relational expressions
Binary relational operators >=, <=, ==, !=, > and < indicate whether the first operand is greater than or equal to, less than or equal to, equal to, not equal to, greater than, or less than the second operand, respectively. The arithmetic operators take precedence over the relational operators. These expressions evaluate to one (1) if the condition holds, and zero (0) otherwise.

3.4 Logical expressions
Logical operators take relational expressions and numbers as operands, so logical expressions have the lowest precedence. Logical operators !, |, & indicate “negate”, “or” and “and”, respectively. Among these operators, operator ! has the highest precedence, then operator &, and operator | has the lowest precedence.

4. Statements
Except as indicated, statements are executed in sequence.

4.1 Expression statement
Most statements are expression statements, which have the form
expression ;
Usually expression statements are assignments or method calls.

4.2 Compound statements
A group of zero or more statements can be surrounded by { and }, in which case they are treated as a single statement. This group of statements constitutes a block for variable name visibility.

4.3 Assignments
An assignment assigns a constant to a specified identifier. Assignments take the form:
left_value_expression = right_value_expression;

4.4 Conditional statement
The two forms of the conditional statement are
`if (`*expression*`)` *statement*
`if (`*expression*`)` *statement* `else` *statement*
In both cases the expression is evaluated and if it is true, the first substatement is executed. In the second case the second substatement is executed if the expression is false. The dangling ‘‘else’’ ambiguity is resolved by connecting an `else` with the last encountered elseless `if`.

4.5 Iterative statements
There are two types of iterative statements, while statements and for statements.

4.5.1 while statements
The `while` statement has the form
`while (` *expression* `)` *statement*
The substatement is executed repeatedly so long as the value of the expression remains nonzero. The test takes place before each execution of the statement.

4.5.2 for statements
The `for` statement has the form
`for (` $expression1_{opt}$ `;` $expression2_{opt}$ `;` $expression3_{opt}$ `)` *statement*
Thus the first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the expression becomes false; the third expression typically specifies an incrementation which is performed after each iteration.

4.6 Method invocation and return
4.6.1. Method invocation
Operator . is used to invoke a method of a specified class. Method invocation takes the form:
class_instance_name.method_name([arguments])
where class_instance_name is the name of the instance of the class of which the method is invoked, method_name is the name of the method within that class and arguments is an optional list of values, separated by commas, matching the parameters of the defined method.

A method call on all objects of a given class, takes the form:
class_name.method_name([arguments])

A method call on all defined objects within a given room, takes the form:
room_name.method_name([arguments])

4.6.2 return statement
A method returns to its caller by means of the `return` statement, which has one of the forms
`return;`
`return` *expression* `;`
In the first case no value is returned. In the second case, the value of the expression is returned to the caller of the method.

5. Classes and Realtime Variables
5.1 Classes
All objects in the house are represented as instances of classes.

5.1.1 Class definition

The user can define classes that extend the pre-defined objects or create entirely new classes. Classes are defined prior to event implementation. Each class inherits the attributes from the standard WHOM superclass and follows the standard WHOM interface. It can define its own attributes, methods, and events. All the attributes, events, and methods are accessible globally. Class definition takes the following form:

`class` *class_name* `extends` *superclass_name*
{
    attributes
    methods
    events
}

5.1.2 Standard library (whom.wl)

The standard library contains standard classes that have already been implemented. The standard library and the user-defined libraries need to be imported by the user, which is described in detail in section 6: House Definition. Since all the attributes and methods are public, the user is free to change any of the attributes and to use any of the methods. All libraries have the .wl extension.

5.1.3 Attributes

Each class can have a set of attributes that could be the type of any previously defined class, number, or string.

5.1.4 Methods

The methods of the class are defined in the class definition section. The method implementation takes the form:
return_type method_name ([parameters]) { statement }

5.1.5 Events

Each class can have a set of events that applies to all instances of that class. The control system will be notified every time this event occurs. The event declaration takes the following form:
event_name;

5.1.6 List of built-in classes

Built-in classes include Room, Window, Blinds, Door, Bed, Table, Chair, Light, Bathtub, Faucet, Oven, Microwave, AirConditioner, Computer, LaundryMachine, AlarmClock, Refrigerator, Fan, Television, VCR, Telephone, FireAlarm, Lawn, Pool, Gate.

5.2 Realtime Variables

Realtime variables specify a set of environmental conditions to which the WHOM objects react. Some realtime variables are standard, but the users can specify their own.

5.2.1 Declaration

The declaration of a realtime variable (unless it is a part of a standard list of realtime variables) has the following form:
`realtime` *variable_type variable_name;*

where variable_type is either number or string;

5.2.2 Built-in realtime variables
Built-in realtime variables include temperature, humidity, weather, hour, minute, second, air pressure, precipitation, and noise.

6. House Definition
After the classes are specified, the house objects are defined.

6.1 Including standard libraries
In order to use the standard libraries, the user needs to include the .wl file in the program. The user can use the standard library whom.wl or create a new library and later include it in the program. The inclusion of a library takes the following form:
`import` *"library_name.wl" ;*

6.2 Object creation
Object creation (or declaration) takes the following form:
class_name id_name ;
The class_name is a name of a class that is contained either in the standard or user-defined libraries and id_name is a name for a particular instance of a class that is specified by the user.

6.3 Putting the objects together
Putting the objects into a room takes the following form:
`put` *(object_list)* `in` *room_name;*
The object list is the list of objects that will be associated with the room specified by the *room_name.*

7. Scoping

WHOM places a variable in one of three scopes at the time of definition.

The highest resolution scope is the program global scope. Variables in the program global scope may be referenced and modified in any part of the source code.

The second highest resolution scope is class instance scope. Variables defined within a class, such as class data members have scope spanning the entire class. This means that any method within that class can reference/modify all variables of that class (as well as all variables defined in the program global scope). All methods, attributes and events defined within the class, may be accessed from outside the class by using the . (dot) operator on the instance of that class.

The last resolution of scoping is at the statement block level. Variables defined within a given statement block of a class have the scope of that block. Any expression or statement of that block may reference/modify that variable (as well as variables defined in the class instance scope, as well as in the program global scope).

8. Events

This portion of the program specifies how the objects will react to the realtime variables and events that are associated with specific objects and that are not specified. Responding to an event takes the following form:

`once` *condition {statement}*

Condition can be either specified by the user directly using the realtime variables or can be a built-in event associated with a specific object or type of object. Statement specifies what needs to be done in response to the event. Conditions are evaluated and actions are carried out whenever realtime variables in the condition or the standard events change.

8.1 Responding to realtime variables

Responding to realtime variables involves testing the current conditions of variables against some prespecified conditions. The condition takes a form of a logical expression.

8.2 Responding to events associated with objects

Responding to an event that is associated with objects involves checking whether this event is true for a specific object. The condition has the following form:

class_instance_name.event_name

The `any` keyword can be used to check if an event has been triggered in any object that is an instance of class.

`once any (`*class_name.event_name*`) {` *statement* `}`