MAPMe
Mapping **A**pplication **P**rogramming language **M**ade for **E**veryone

Language Reference Manual

COMS 4115 Programming Languages and Translators
Professor Stephen A. Edwards

Team Members:

Jonecia Keels	jtk2128
Denelle Baptiste	dgb2122
Changyu Liu	cl2997
Eric Ellis	ede2106

1. Introduction

This manual is a quick reference guide for using MAPMe language. It includes the specific introduction of lexical conventions, types, operators, statements, and samples.

2. Lexical conventions

2.1 Comments

The '~' character introduces a comment and another '~' character ends a comment, except within a character constant or string literal.

The number of the comment character '~' must be even. So the compiler will automatically scan all '~' which are not in a character constant or string literal. If the number of '~' is odd, the compiler will complain. If it is odd, compiler will divide into pairs and treat the content between a pair of '~' as comments.

2.2 Key Words

MAPMe defines several keywords, each with special meaning to the compiler.

- **Map**
- **Point** (Longitude/Latitude Points)
- **Path** (Paths, Trails, Roads, Sidewalks, etc.)
- **Object** (Buildings, Vehicles, People, etc.)
- **build** (To instantiate a new object)
- **Array <Type>**
- **HashTable**
- **double, float, string, char, bool**
- **if, else, while, return**

2.3 Identifier

In MAPMe, an identifier is a sequence of characters that represents a name for the following:

- **Variable**
- **Function**
- **Map**
- **Object**
- **Point**
- **Array**
- **Some other data structure like int, char, etc.**

For example, Point point1 = new Point(10, 10). Here point1 is an identifier. Notice that keywords ***cannot*** be identifiers.

2.4 Constant

2.4.1 Integer constants

Integer constants are used to represent whole numbers. In MAPMe, an integer constant can only be specified in decimal without suffix. To specify an integer constant, use a sequence of decimal digits in which the first digit is not 0.

2.4.2 String constants

A string constant is a sequence of characters enclosed in double quotation.

For example: **string** a = “hello”.

3 Types

3.1 Primitive Data Types

3.1.1 String
3.1.2 Double
3.1.3 Boolean
3.1.4 Array

3.2 Object Types

3.2.1 Path
3.2.2 Map
3.2.3 Object
3.2.4 Point

4. Operators

4.1. Logical Operators

Logical Operators	
<	Less than
>	Greater than
<=	Less than or Equal to
>=	Greater than or Equal to
==	Equal to
OR	Or Operator
AND	And Operator
^	Not Operator

4.2. I/O operation

I/O Operation Keywords	
display	Prints to file or stdout
stream	Reads from files and stdin

5 Statement

5.1 Declaration/Assignment:

Declaration Format: *Data/Object type identifier = build Data/Object type*
Primitive Data type identifier = expression
Ex: *Array <Point> [ ] Points= build Array<Point> [2];*
double distanceToChipotle;

Assignment Format: lvalue = expression
Ex. distanceToChipotle = myApartment.getDistanceTo(chipotle);

Syntax:

5.3 Looping Construct

The iteration statement included is started by the while keyword. The while construct is structured as follows:

```
while (conditional expression) {
    statement
}
```

As long as the conditional expression is satisfied, the statements within the while statement brackets is executed continuously.

5.4 Built In Methods

Built In Method	
Object1.getDistance(Object2)	Returns distance in miles from Object1 to Object 2. Each Object must have a Point that define it's location (longitude/latitude)
Getter methods for Object types	Example: Object.point, Object.type, Object.speed (if applicable)

MAPMe has built in methods for mapping calculations and manipulation.

6. Sample Program

Sample XML file named MapData.xml, that stores all of the input data such as Points of Interests, roads, and longitude/latitude positions:

```
<Path>
        <Name>
                Broadway
        </Name>
        <Type>
                Road
        </Type>
        <NumOfPoints>
                2
        </NumOfPoints>
        <Points>
                40.473245,-73.9251; 40.807991,-73.963829
        </Points>
</Path>
<Object>
        <Name>
                Chipotle
        </Name>
        <Type>
                Building
        </Type>
        </Point>
                40.798737, -73.970947
```

```
</Point>
</Object>
<Object>
        <Name>
                myApartment
        </Name>
        <Type>
                Building
        </Type>
        </Point>
                40.808417, -73.963737
        </Point>
</Object>
<Object>
        <Name>
                myCar
        </Name>
        <Type>
                Car
        </Type>
        <Speed>
                40mph
        </Speed>
</Object>
```

MAPMe Code:

~This method reads in data which creates object types that the user can reference~

```
stream(MapData.xml);

double distanceToChipotle = myApartment.getDistanceTo(chipotle);
```

~flexibility to calculate estimated time of arrival given users input data~

```
double timeToGetToDest = myCar.speed / distanceToChipotle;
```