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

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or Equal to</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Greater than or Equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>OR</td>
<td>Or Operator</td>
</tr>
<tr>
<td>AND</td>
<td>And Operator</td>
</tr>
<tr>
<td>^</td>
<td>Not Operator</td>
</tr>
</tbody>
</table>
4.2. I/O operation

<table>
<thead>
<tr>
<th>I/O Operation Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>Prints to file or stdout</td>
</tr>
<tr>
<td>stream</td>
<td>Reads from files and stdin</td>
</tr>
</tbody>
</table>

5 Statement

5.1 Declaration/Assignment:
Declaration Format: \textit{Data/Object type identifier} = \textit{build Data/Object type}
\textit{Primitive Data type identifier} = \textit{expression}
Ex: \textit{Array <Point> [ ] Points= build Array<Point> [2]; double distanceToChipotle;}

Assignment Format: \textit{lvalue = expression}
\textit{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:
\begin{verbatim}
while (\textit{conditional expression}) {
  \textit{statement}
}
\end{verbatim}
As long as the conditional expression is satisfied, the statements within the while statement brackets is executed continuously.

5.4 Built In Methods
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:

```xml
<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>
    <Point>
        <Name>myApartment</Name>
        <Type>Building</Type>
        <Point>40.808417, -73.963737</Point>
    </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;