Project Kilo: A Mobile Map Service
XML Document Specification

1 Introduction

This document defines the XML datatype used for the representation of map data, as stored locally on the server. The map data must conform to the document type declaration (DTD), and the type-checking requirements of the server - both of which are specified below.

2 Document Type Declaration

The XML map documents will be required to conform to the following document type declaration:

```
<!ELEMENT map (node*, location*, metalocation*, road*, category*)>

<!ELEMENT node (edge+)>
  <!ATTLIST node id ID #REQUIRED
                 position CDATA #REQUIRED>

<!ELEMENT edge (subnode*)>
  <!ATTLIST edge to IDREF #REQUIRED
                 road IDREF #REQUIRED
                 length CDATA #REQUIRED
                 weight CDATA #IMPLIED
                 type (C|B|P|CB|CP|BP|CBP) "CBP"
                 side (left|right) #IMPLIED>

<!ELEMENT subnode EMPTY>
  <!ATTLIST subnode position CDATA #REQUIRED>

<!ELEMENT location (#PCDATA)>
```

```
  <!ATTLIST location id ID #REQUIRED
                     node IDREF #REQUIRED
                     category IDREFS #REQUIRED
                     notability CDATA "1">

<!ELEMENT metalocation (#PCDATA)>
  <!ATTLIST metalocation id ID #REQUIRED
                         sublocations IDREFS #REQUIRED>

<!ELEMENT road (#PCDATA)>
  <!ATTLIST road id ID #REQUIRED
                 notability CDATA "2">

<!ELEMENT category (#PCDATA)>
  <!ATTLIST category id ID #REQUIRED>
```

In order to specify this requirement, all map documents must begin with the following header:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE map SYSTEM "Map.dtd">
```

This specifies that they will conform to the above DTD, and contain the document entity `<map> </map>`. Within the map entity there are five main sections:

1. A list of `node` elements, each with a unique identifier, `id`, and coordinates specified by `position`. Each node lists one or more `edge` elements, specifying a route to another node in the `to` attribute (the value of `to` should be the `id` of a node). The attributes of an edge must also specify a reference to a `road` element and the `length` of the edge as a distance. There are a further three optional attributes, allowing you to specify the `type` of the edge, its `weight` (indicating the scaling factor of its length during routing), and the `side` of the edge that the node it is linking to is located.

   There is the option of listing a number of `subnode` elements within an edge, which each specify only a `position` coordinate. These will not be used for routing on the graph; only for drawing the edge, when they will be taken as points that the edge must pass through.

2. A list of `location` elements, each with a unique identifier, `id`, and a reference to the `node` at which the location is situated. It is also

necessary to provide one or more references to the `category` of location (where the references to category `id` are separated by whitespace). An optional `notability` attribute may be specified, to indicate how prominent the location is as a landmark.

The name of the location must be stated in the content of the element.

3 A list of `metalocation` elements, each with a unique identifier, `id`, which provide a spacial grouping of locations or other metalocations. The locations/metalocations that are to be grouped together are specified in the `sublocations` attribute by a list of `id`s, separated by whitespace.

The name of the metalocation must be stated in the content of the element.

4 A list of `road` elements, each with a unique identifier, `id`, which provide the name of a 'geographical' road in the element content. The optional notability field provides control over when the road name is displayed when rendering the map. Each `edge` element then references a road.

5 A list of `category` elements, each with a unique identifier, `id`, which provide the name of a category of location in the element content. Each `location` element then references a category.

3 Server Requirements

In addition to the structural requirements of the DTD, the server requires that the values of the attributes be in an appropriate format. This involves imposing restrictions that will be checked at the time that the server reads the XML document, and rejecting those documents that do not comply.

The following list indicates the specific requirements of each attribute in the XML document (where the symbol x may be substituted for any string matching (0|1|2|3|4|5|6|7|8|9)+):

- `node`
  - `id` - a unique node identifier, of the form nx
  - `position` - a coordinate taken from the origin of the map, in metres, of the form x,x
- `edge`
  - `to` - a reference to a node, of the form nx

  - `road` - a reference to a road, of the form rx
  - `length` - a distance in metres, of the form x
  - `weight` - a floating point value (multiplying factor) of the form $x.x$ (optional)
  - `type` - one of the following values (default is `CBP`):
    * `C` - cars only
    * `B` - bikes only
    * `P` - pedestrians only
    * `CB` - cars and bikes only
    * `CP` - cars and pedestrians only
    * `BP` - bikes and pedestrians only
    * `CBP` - cars, bikes and pedestrians
  - `side` - either `left` or `right` (optional)

- `subnode`
  - `position` - a coordinate taken from the origin of the map, in metres, of the form x,x

- `location`
  - `id` - a unique node identifier, of the form lx
  - `node` - a reference to a node, of the form nx
  - `category` - a series of references to categories, each of the form cx, separated by whitespace
  - `notability` - a numerical value of the form x (default is 1)

- `metalocation`
  - `id` - a unique node identifier, of the form mx
  - `sublocations` - a series of references to locations or metalocations, each of the form lx or mx, separated by whitespace

- `road`
  - `id` - a unique node identifier, of the form rx
  - `notability` - a numerical value of the form x (default is 2)

- `category`
  - `id` - a unique node identifier, of the form cx

4 Example XML File

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE map SYSTEM "Map.dtd">

<map>
  <node id="n1" position="56,87">
    <edge to="n2" road="r1" length="23"/>
    <edge to="n3" road="r2" length="12"/>
  </node>

  <node id="n2" position="65,43">
    <edge to="n1" road="r1" length="23"/>
  </node>

  <node id="n3" position="22,98">
    <edge to="n1" road="r2" length="12"/>
  </node>


  <location id="l1" node="n1" category="c1">Fred's Fish And Chips</location>
  <location id="l2" node="n2" category="c2">Bob's Barbers</location>


  <metalocation id="m1" sublocations="l1 l2">Downtown</metalocation>


  <road id="r1">Silly Street</road>
  <road id="r2">Random Road</road>


  <category id="c1">Places To Eat</category>
  <category id="c2">Places To Sleep</category>

</map>
```

5 Final Remarks

There are both advantages and disadvantages to using DTDs for XML verification, as opposed to a more powerful document constrain mechanism such

as XML Schema. The primary advantage of DTDs is the plethora of compliant parsers, however there are also some serious flaws. For example, there is no way of specifying types (other than the native string), which does not allow for data constraint at the time of entry. Similarly, IDREFs cannot be contrained to referencing a specific element type.

In this context, the primary use of the DTD is to specify the structure of the document. The Java server will then parse the XML file, and apply type checking before storing it internally. Thus, to the writer of the XML map document, there is no apparent difference - documents that do not conform to either the DTD or the server requirements will be rejected.