
EmulationScript Schema Description

Protean Research Group, Naval Research Laboratory Code 5522

Abstract

This document describes an XML schema being developed to support generation and manipulation of "planning" and "scripting" documents and files used in support mobile network modeling. The schema provides for documents that can be used to orchestrate mobile network modeling using emulation environments such as the Naval Research Laboratory (NRL) "extensible Mobile Ad-hoc Network Emulator" (eMANE) framework or using network simulation tools such as *ns-2*. The initial focus of this development is on planning, scripting, and generating mobile node location and motion properties for wireless network emulations and simulations. Some initial tools are available from <http://cs.itd.nrl.navy.mil/products> to create and manipulate files in the XML formats described here. Additional schema and related tools are planned for future development.

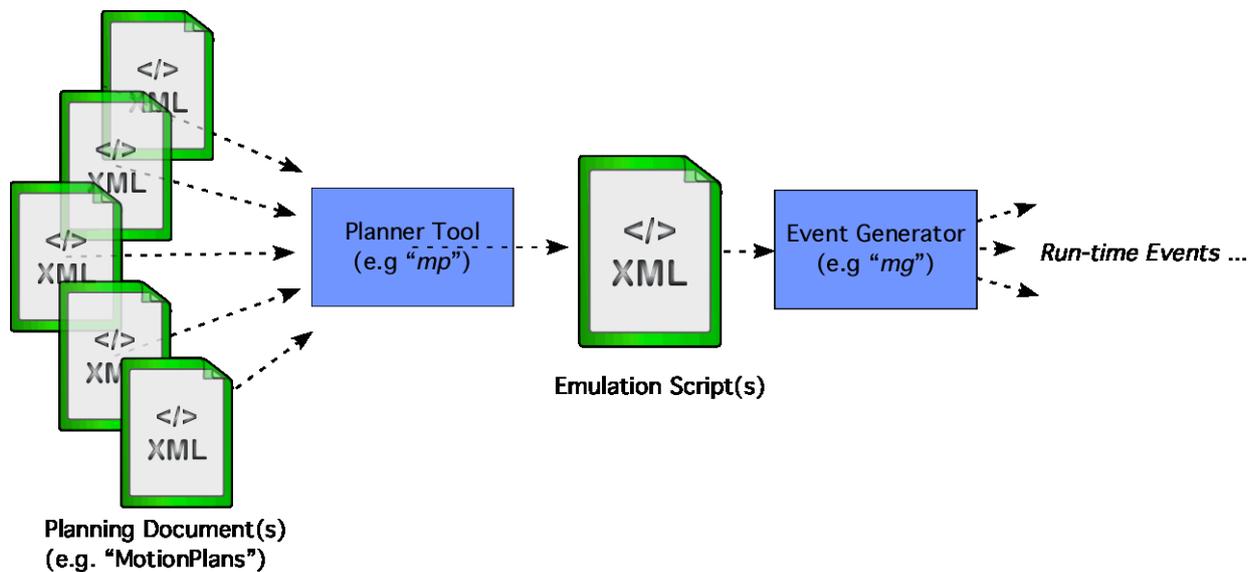
1. Introduction	1
2. EmulationScript Documents	3
2.1. Top-Level Elements	4
2.1.1. EmulationScript Document	4
2.1.2. Event Element	4
2.2. Data Types	4
2.2.1. "time" Types	4
2.2.2. "location" Types	5
2.2.3. "motion" Types	5
2.3. Event Targets	7
2.3.1. "Node" Module	7
2.3.2. TBD - Additional Scriptable Module Definitions	7
3. MotionPlan Documents	7
3.1. Motion Primitives	8
3.1.1. "location" Primitive	9
3.1.2. "waypoint" Primitive	9
3.1.3. "vector" Primitive	10
3.1.4. "circle" Primitive	10
3.1.5. "loiter" Primitive	11
3.1.6. "randpoint" Primitive	11
3.1.7. "pause" Primitive	12
3.1.8. "pattern" Primitive	12
3.1.9. "mark" and "wait" Primitives	12
3.2. Motion Pattern Definition	14
3.3. Random Waypoint Generator Definition	14
3.4. Node Motion Plan Specification	14
4. Example Utilities	15
5. Alternative File Formats	15
6. Usage Notes	15

1. Introduction

This document describes an accompanying eXtensible Markup Language (XML) Schema ("EmulationScriptSchema.xsd" document) specifying an XML document format that can be used for scripting Mobile Network Emulation run-time events. More specifically, this design is provided to support operation of the extensible Mobile Network Emulation (eMANE) system developed by the Naval Research Laboratory (NRL) and CENGEN, Inc. The script events include defining certain mobile "node" characteristics (e.g. location, velocity, orientation, etc) and parameters of other functional modules and changes to those characteristics or parameters over the course of emulation execution. Note the eMANE system is sufficiently modular that other scripting or control techniques

might be applied to its operation, depending upon the design of the specific Network Emulation Modules (NEMs) and other components loaded into an instantiation of the eMANE system. The script format described here is intended to provide a default standard approach to be used by the core set of eMANE modules developed with the initial version of eMANE. It is expected that these scripts will generally be the product of scenario generation programs or other tools, but of course, may be manually created as well. In fact, the schema also defines a second XML document type, the “MotionPlan” document. While the “EmulationScript” schema specifies a document structure to contain time-ordered “Events” that tightly script (dynamically update) emulation module properties (e.g. Node location/motion and other), the “MotionPlan” document type provides a means to specify, in a little looser fashion, a mobility profile on a per-node basis. An example tool is provided to automatically convert “MotionPlan” documents into “EmulationScript” documents containing the corresponding “Node:motion” events. Figure 1, “Emulation Planning and Scripting Work Flow” illustrates the proposed work flow of consolidating multiple “planning” documents (including MotionPlans) into a time-ordered event script that can be used to generate run-time events for a mobile network emulation experiment.

Figure 1. Emulation Planning and Scripting Work Flow



This XML schema is defined to enable the use of existing and emerging XML content manipulation tools to be able to perform operations on scripts. This includes filtering of events within scripts, potential merging of scripts, and manipulations that might parametrically alter a script. However, note the XML format described here is not intended to be a complete replacement for other simpler file formats that might be used to describe mobile node location and/or motion. In fact, this schema is purposefully designed to allow for conversion to and from such simpler file formats as needed and such alternative formats are described in this document. For example, in some cases such as node location and motion, much more compact (as compared to XML) representations can be realized and may be more practical for some purposes due to file sizing and other factors. Similarly, this schema is expected to be capable of representing the content of several existing mobility script formats.

So, additionally, this document will specify alternate compact textual representations of some of the XML elements the described scheme introduces. While the “Script Schema” described here will be able to contain (and provide context for via XML tags) multiple script event types, and in fact will be able to evolve over time as new event types are determined, the simpler text formats will generally contain events for some particular subtext (e.g. node mobility). This ability to provide translation compatibility with other formats can make this schema useful for specifying and/or scripting scenarios outside of strictly eMANE or emulation purposes, including use with tools for discrete event network simulation (e.g. ns-2, OPNET, etc) or for analytical programs.

The first iteration of this document (and accompanying schema) primarily focuses on describing elements that can represent mobile node location and optionally velocity (or even richer motion patterns). However, the structure of the schema attempts to be sufficiently general to contain XML elements allow future inclusion of events that can initialize or update characteristics of any modules within the emulation system. This generalized capability will

allow users to construct scripts that can describe alterations to arbitrary aspects of the emulation over time (provided that the eMANE or other run-time framework supports interpretation of these events and control of those aspects).

2. *EmulationScript* Documents

To support this goal of generalized event scripting, the top-level hierarchy of the Emulation Event Schema described here consists of the “Emulation Script” document that principally contains a list of abstract “Event” elements. Each “Event” contains a “time” value element corresponding to the time at which the event should be executed and one or more sub-elements that identify the emulation “Module” instance(s) (each instance is identified by an “id” attribute string) and any properties that are being set or altered (i.e. the Event action(s) to be taken). These sub-elements are specified in a hierarchical fashion so that possibly multiple or complex characteristics and/or sub-attributes can be described as needed. The following pseudo-script outlines this hierarchy:

```
<EmulationScript>
  <Event>
    <time> time1 </time>
    <Module1 id="moduleA">
      <property1> value </>
      <property2> value </>
      ...
    </Module1>
    <Module1 id="moduleB">
      <property1> value </>
      <property2> value </>
      ...
    </Module1>
    <Module2 id="moduleZ">
      <propertyX> value </>
      <propertyY> value </>
      ...
    </Module2>
  </Event>
  <Event>
    <time> time2</time>
    <Module1 id="moduleA">
      <property1> value </>
      <property2> value </>
      ...
    </Module1>
  </Event>
  ...
</EmulationScript>
```

A specific example of this hierarchy is scripting of mobile “Node” mobility properties within an EmulationScript document. In this case, the “Node” corresponds to the “Module” being addressed with specific node instances being identified by their identifier (name as indicated by the required “id” attribute). The current version of the .xsd schema file accompanying this document specifies the “Event:Node” type and some properties related to scripting node location and/or motion. The simple example given here scripts the location of two nodes (“node01” and “node02”) with an initial location at time “0.0” (seconds) and updated locations at time “1.0”:

```
<EmulationScript>
  <Event>
    <time> 0.0 </time>
    <Node id="node01">
      <location>38.123,-78.524,1000</>
    </Node>
    <Node id="node02">
      <location>38.232,-78.254,800 </>
    </Node>
  </Event>
  <Event>
    <time> 1.0 </time>
    <Node id="node01">
      <location>38.124,-78.525,1000</>
    </Node>
    <Node id="node02">
```

```

    <location>38.233,-78.255,800 </>
  </Node>
</Event>
...
</EmulationScript>

```

This section describes the top-level EmulationScript” document and Event elements, describes data types that are used throughout the document (i.e. by multiple sub-element types), and then provides sections describing the various sub-element types (Modules and their properties). This document and its associated schema will continue to be updated as this script format evolves.

2.1. Top-Level Elements

The top-level elements that comprise an emulation script are the Emulation Script Document itself and the Event element type. The “Emulation Script” essentially consists of a time-ordered list of “Events”.

2.1.1. EmulationScript Document

An emulation script document is encapsulated with the <EmulationScript> tag. The content of the document is a list of <Event> elements as described below. The list of script Events **MUST** be sorted by order of their <time> value.

The <EmulationScript> also can contain an optional <startTime> element that specifies an absolute time (possibly including date) that corresponds to the script’s “time zero” starting time. This element, if included, **MUST** be the first element within the <EmulationScript> document.

2.1.2. Event Element

The <Event> element is used to contain a list of emulation modules (with their associated properties) that are to be updated at a given time. Each <Event> corresponds to a specific instance of time, and the <time> element of the <Event> indicates this. Note that the <time> element is a complex type that may express either an absolute time (using XML date/time conventions) or a relative time in units of seconds. The format of the <time> element is further described in the “Global Data Types” section below.

2.2. Data Types

This section describes some data types that are defined globally in the “EmulationScript” namespace. These data types include formats to describe some fundamental values that are used throughout the emulation system (e.g. “time”, “location”, etc).

2.2.1. "time" Types

The `Event:time` element is a complex type that may represent either a relative or absolute instant of time (Note: future versions of this specification may possibly limit the `Event:time` to be a relative time only). The “format” attribute of the `Event:time` element indicates the how the element’s text content should be interpreted. Possible `Event:time` formats include:

1. “secs” indicating an floating point value of relative time in seconds.
2. “chron” indicating a relative chronograph in the “xs:duration” built-in format of “P[nY][nM][nD][T[nH][nM][n[.nnn]S]]” where: P denotes a period (required), nY indicates 'n' years, nM indicates 'n' months, nD indicates 'n' days, T indicates the start of a time section (required if you are going to specify hours, minutes, or seconds) nH indicates 'n' hours, nM indicates 'n' minutes, and n[.nnn]S indicates 'n.nnn' seconds.
3. “clock” indicating an absolute clock time in the “xs:time” built-in format of “HH:MM:SS[.SSS]”. The respective 'hours' (HH), 'minutes' (MM), and 'seconds' (SS.[.SSS] fields **MUST** be specified as 2-digit integers (leading zeroes as required) although the 'seconds' field can be extended with a decimal point and additional digits to describe a floating point value.

4. “dateTime” indicating an absolute date and time in the “xs:dateTime” built-in format of “YYYY-MM-DDThh:mm:ss[.sss]” where: ‘YYYY’ indicates the year, ‘MM’ indicates the month, and ‘DD’ indicates the day. ‘T’ indicates the start of the required time section where ‘hh’ indicates the hour, ‘mm’ indicates the minute, and ‘ss[.sss]’ indicates the second (a decimal point and digits may extend the ‘ss’ field to a float value). Note: All components are required! To specify a time zone, you can either enter a dateTime in UTC time by adding a “Z” at the end, or you can specify an offset from the UTC time by adding a positive or negative ({+|-}) “hh:mm” quantity at the end (e.g., “2008-09-21T21:43:00+05:00” for “21 Sept, 2008 9:43 PM EST”).

Note the `EmulationScript:startTime` element is similar to the `Event:time` but its text content is restricted to the absolute time formats of either “clock” or “dateTime”. It also has a “type” attribute to indicate the text content format.

2.2.2. "location" Types

The emulation script schema validates coordinates within the script to be a comma-delimited set of at least two and optionally three floating point values (i.e. “a,b[,c]”). The schema defines a “locationType” complex type that has a “type” attribute that indicates the geometry of the given coordinates. The current two possible geometry types are:

1. “gps” indicating the coordinates represent a geographic location of “latitude, longitude[,altitude]” where ‘altitude’ is in units of meters, and
2. “cartesian” indicating the coordinates represent an “x,y[,z]” location.

Note that the ‘altitude’ and ‘z-axis’ are the optional third value. When this third value is omitted, the altitude or z-axis value is implicitly zero. It is also RECOMMENDED that a consistent geometry type be used within a given EmulationScript instance. However, future versions of this specification may provide a element that allows one type of geometry to be mapped to another.

This “locationType” is used in both MotionPlan and EmulationScript documents including use as the Node:location property element (described later) and to indicate locations as required embedded elements describing motion (e.g. `EmulationScript:Event:Node:motion` and motion primitive elements).

The format of “locationType” elements is

```
<location type="gps"><lat>,<lon>[,<alt>]</location>
```

or

```
<location type="cartesian"><x>,<y>[,<z>]</location>
```

Note the `<alt>` and `<z>` ordinates are optional in the “gps” and “cartesian” coordinate types, respectively.

(TBD – allow option to indicate that `<alt>` or `<z>` represents an “above ground level” (relative to terrain) elevation instead of strictly representing an absolute elevation. (E.g. add an attribute “alt” with possible values of “rel” (relative) or “abs” (absolute)).

2.2.3. "motion" Types

The schema currently specifies an element form that is used to describe the property of “motion” (i.e. movement in space) within the emulation script. For example, the “Node” module (described later) may have a “motion” property associated with it. The “motion” element form consists of two portions:

1. A “location” element indicating the current location of the applicable object, and
2. One of a set of different motion types.

Note the mandatory inclusion of the current location as part of the motion specification is intentional. The reasons for this include the fact that the “location” given is implicitly the starting point for the specified motion and that this simplifies the process for run-time emulation controllers to “jog-shuttle” (i.e. rewind/fast-forward) to specific

points in time as needed. The inclusion of this “location” element in the motion specification requires that EmulationScript writers or generation tools need to provide a current “location” that is consistent with prior “location” and “motion” events to evoke smooth patterns of motion. However, it is possible that “instant teleportation” be scripted if desired as well.

A few different basic motion types are currently described in the schema. In the future, additional types will be added and this document will be updated to describe these added types. The sections below describe the motion types currently supported.

2.2.3.1. 'waypoint' Motion Type

The “waypoint” motion type is specified with a “destination” location and the “velocity” at which the “Node” moves there. The intended interpretation of this motion type is that the node will move directly towards the “destination” at the given “velocity” until it either reaches the destination or its motion is superseded with a different motion (or fixed location) property in the script (i.e. motion is halted when the waypoint destination is reached). Note that the requirement that “motion” elements contain a current “location” location means that it is possible that Nodes may instantly “teleport” to a new location if the location value given in a new “location” or “motion” update is inconsistent with any prior specified motion or location.

2.2.3.2. 'vector' Motion Type

The “vector” motion type is specified with “azimuth” and optionally “elevation” angle(s) indicating the direction (or “bearing”) of movement and the “velocity” at which the “Node” moves there. Like the “waypoint” type, motion in a straight line (using “great circle” when GPS coordinates are used) is thus defined, but with no specified final destination or halting point. The motion continues indefinitely until it is superseded by another motion (or “location”) update. The “circle” motion type parameter elements include:

1. “velocity” of motion (default units of meters per second) ,
2. “azimuth” (i.e. bearing) angle in units of degrees ($\pm 360^\circ$). This is relative to due North in GPS coordinates or to the y-axis in Cartesian coordinates, and
3. “elevation” angle (optional) in units of degrees ($\pm 90^\circ$). This indicates the rate of change in altitude (or along the z-axis) as motion occurs. Note that at plus (or minus) 90° , the motion velocity is 100% applied towards altitude (height) change and not motion in the given bearing direction occurs! If the optional "elevation" element is not include, a value of 0° (no altitude change) is assumed.

2.2.3.3. 'circle' Motion Type

The “circle” motion type describes a circular path about a center location with a fixed radius and altitude (if applicable) at a fixed velocity. The “circle” motion type parameter elements include:

1. “center” location coordinates with optional altitude/height which, if omitted, implies the motion follows any applicable terrain. (Note that the “center” element is optional for MotionPlan documents and that the circle center will be set to the node’s current location in the motion plan and the motion will spiral outwards to the circle perimeter at a constant angular velocity based on the objective circle “radius” and “velocity”),
2. “radius” of the circle in units of meters, and
3. “velocity” of motion (default units of meters per second) where a positive “velocity” implies a clockwise direction (as viewed from a higher altitude looking downward) and a negative “velocity” is used to specify a counter-clockwise direction.

The intended interpretation of this motion type is that the node will follow the circular path continuously at the given “velocity” until its motion is superseded with a different motion (or fixed location) property in the script. The required “motion:location” element accompanying the “circle” motion specification MUST be a point that is consistent (valid) with respect to the specified circle and serves as the starting point of the circular motion.

2.2.3.4. "loiter" Motion Type

The “loiter” motion type describes a circular path of fixed radius and relative height (if applicable) about a location that is moving according to another “reference” motion type. The “loiter” motion type parameter elements include:

1. An embedded reference "motion" specification that consists of one of the motion types described here. For “MotionPlan” documents, the “reference motion” may actually be a “pattern” which was previously defined using the “PatternDefinition” element.
2. The “radius” of the loiter circle in units of meters.
3. The “velocity” of the loiter motion (default units of meters per second) where a positive “velocity” implies a clockwise direction (as viewed from a higher altitude looking downward) and a negative “velocity” is used to specify a counter-clockwise direction.
4. An optional “height” (in meters by default) that is an altitude (z-axis) offset from the reference motion current location.

The intended interpretation of this motion type is that the node will follow the circular path continuously at the given loiter “velocity” until its motion is superseded with a different motion (or fixed location) property in the script. The required reference motion element accompanying the “loiter” motion specification provides the moving “center” location for the loiter circle that is conducted. Note that the “reference motion” may also be a fixed “location” element, and is thus equivalent to a circle motion definition in this case.

2.3. Event Targets

The targets of the contents of the <Event> element are the various “modules” that comprise the emulation system. These include “modules” that represent the virtualized (emulated) components of the systems such as “Nodes” and their associated configuration items (e.g., software processes such as network routing or application daemons) or environmental characteristics (e.g., geographic location and motion properties). These also may include “modules” that are control components of the emulation system itself.

2.3.1. "Node" Module

The “Node” module identifies and sets properties for a physically distinct entity that is being virtually represented within the mobile network emulation system. Examples of “Nodes” may include mobile or fixed platforms such as vehicles, building, people, devices (e.g. robots), etc. Typically a “Node” also corresponds to a single intermediate (e.g. router) or end (e.g. host) system within the possibly mobile network, but it is possible that some complex emulation scenarios may include Nodes that contain multiple network systems as well as network interfaces. These emulated systems and interfaces are anchored to the Node location as they are a physical component of the Node itself.

2.3.2. TBD - Additional Scriptable Module Definitions

(TBD – enumerate/describe additional “modules” of the emulation system that may be dynamically scripted/controlled during operation)

3. MotionPlan Documents

The “EmulationScriptSchema.xsd” also defines a “MotionPlan” XML element that can be used at the top level in a document that describes motion sequencing on a nodal basis. The goal of this document is to provide a relatively free-form method for users to sequence node motions. A software tool is provided then that generates “EmulationScript” documents from these “MotionPlan” documents. To summarize, the “EmulationScript” specifies motion events on a time-ordered basis with the tight requirement that motion updates specifically list the current location of the node when the update occurred. This makes the emulation script useful for run-time motion generation that might be dynamically controlled (e.g. shuttled over in time), but complicated to generate. The MotionPlan is, in contrast, based on defining the desired sequences of motion on a nodal basis in such a way that it provides a relatively

simple user (or algorithmic) format for specifying node motion. Then, the MotionPlan format can be used by software tools to generate EmulationScript Node:motion update events.

The “MotionPlan” schema lets complex motion be described as a concatenation of motion primitives (directly related to the motion types the EmulationScript schema supports). Furthermore, sequences of motion primitives can be encapsulated as named motion “patterns” and those patterns can be re-used by reference to specify, possibly repetitive, node motion or as part of more complex pattern definitions. The following pseudo-XML provides a simplified overview of the MotionPlan schema:

```
<MotionPlan>
  <PatternDefinition name="loop">
    <waypoint>
      <destination> lat1, lon1, alt1</>
      <velocity>12.0</>
    </waypoint>
    <waypoint>
      <destination> lat2, lon2, alt2</>
      <velocity>12.0</>
    </waypoint>
    <waypoint>
      <destination> lat3, lon3, alt3</>
      <velocity>12.0</>
    </waypoint>
    <waypoint>
      <destination> lat1, lon1, alt1</>
      <velocity>12.0</>
    </waypoint>
  </PatternDefinition>
  <Node id="node01">
    <location> lat0,lon0,alt0</location>
    <pause>120.0</pause>
    <pattern repeat="-1" duration="600.0">loop</pattern>
    <waypoint>
      <destination>lat4,lon4,alt4</>
      <velocity>20.0</>
    </waypoint>
    <pause>120.0</>
    <circle>
      <center>lat5,lon5,alt5</>
      <radius>30.0</>
      <velocity>15.0</>
    </circle>
  </Node>
</MotionPlan>
```

In this approximate MotionPlan example, a “loop” pattern is defined that consists of a triangle of three waypoints. Then, the “node01” motion plan is specified with an initial location and pause of 120.0 seconds followed by the triangle “loop” pattern with an undefined number of “repeats”, but limited to a “duration” of 600.0 seconds. After the 600.0 seconds of time spent in the “loop”, the node traverses to a specified “waypoint” and begins another 120.0 second “pause”. After this pause, our “node01” finally enters into a 30 meter radius “circle” motion pattern for an indefinite amount of time.

The file “mpExample1.xml” included with the example tools and schema documents distribution contains a version of the pseudo-example of Figure 3, but with real GPS coordinates. The “mp” utility can be used to parse this example MotionPlan file and generate a resultant EmulationScript document with the corresponding location/motion events for the “node01” entity.

3.1. Motion Primitives

The “MotionPlan” motion primitives include basic elements that correspond to the “EmulationScript” motion types and some added elements to pace and sequence these types. In the “MotionPlan”, an emulation node’s motion is expressed as a concatenated sequence of primitive motion types. The current motion types include “location”, “waypoint”, “vector”, “circle”, and “loiter” plus a “pause” primitive to temporarily halt motion at the current point in the motion plan. The motion types also have a “duration” attribute that can be set to a specify time limits for

each primitive to pace the execution of the concatenated set of motion primitives. An intended use of the "MotionPlan" is to generate a set of Node mobility events for an "EmulationScript".

The "MotionPlan" is a slightly loose specification of motion elements and so specific behaviors must be described for transitioning from one motion primitive to the next. For example, the transition from a prior location to a waypoint is relatively obvious. The "waypoint" motion primitive implicitly indicates movement from a prior location to a given "waypoint:destination" at the "waypoint:velocity". However, transitioning to a "circle" motion from a previous location may require that an intermediate "waypoint" motion event be generated to smoothly transition to a location on the circle's perimeter. As an example for this case, the RECOMMENDED approach for this is to establish a direct route (great circle route when GPS geometry is used) from the last motion location towards the "circle:center" to a point that intersects the circle perimeter path. This intersecting location will serve as the initial location for the circle motion. In the case that the previous location is *_within_* the circle's radius, then an intermediate waypoint specifying direct (shortest) route to the circle perimeter SHOULD be generated. The "velocity" for these intermediate waypoint transitions SHOULD be that of the target circle motion (i.e. "circle:velocity"). And, as noted below, the time required to complete the transition SHOULD be considered an inclusive part of any maximum motion pattern "duration" time set for the circle motion specification.

3.1.1. "location" Primitive

The "location" primitive specifies immediate re-location (i.e., teleportation) to specified coordinates. The "location" element has a "type" attribute with an implicit default value of "gps" (geodetic coordinates). The attribute may alternatively be set to a value of "cartesian" to indicate the coordinates represent Cartesian coordinates.

The "duration" attribute can be used to specify how long the node should remain at the specified location. Also a "pause" primitive following a "location" element equivalently indicates the dwell time the node should remain at the specified location before beginning transition to any succeeding motion primitive (or pattern). Note that the "duration" attribute value and a succeeding "pause" time will be additive.

The default format of the "location" element is:

```
<location>lat,lon,alt</location>
```

The transition to the location is an immediate change in location from the last location of a preceding motion primitive.

3.1.2. "waypoint" Primitive

The "waypoint" primitive specifies a "destination" towards which the node should move at specified "velocity". The motion proceeds until the destination is reached or the "duration" attribute time expires, if specified. When the waypoint motion has completed (i.e. when the "duration" time has expired if applicable, or the waypoint destination is reached if no "duration" time limit is specified), the subsequent motion primitive is considered. If a specified "duration" time is longer than it takes to reach the "destination", then the node will pause at the "destination" location until the "duration" time expires.

Note the destination element type is the same as that of the "location" primitive previously described where its coordinate format is "gps" by default with the "cart" value option available via the "geometry" attribute. The default units of "velocity" elements in this document is "meters per second", but alternative "units" are available if the corresponding attribute is set. However, the example tools (described later) currently implemented against this specification assume units of "meters" and "meters per seconds" for distance and velocity values, respectively and do not yet support other unit types.

The format of the "waypoint" element is:

```
<waypoint>
  <destination>lat,lon[,alt]</destination>
  <velocity>metersPerSec</velocity>
</waypoint>
```

The transition to the waypoint motion from a prior location is simply the waypoint specification itself.

3.1.3. "vector" Primitive

The “vector” primitive specifies a direction, given in “azimuth” and optional “elevation” angles at which the node should move at specified “velocity”. The motion proceeds indefinitely or until its “duration” attribute time expires, if applicable. Thus a “duration” attribute **MUST** be specified unless the desired behavior is continuous, unbounded motion along the given vector direction.

The default units type for “velocity” elements in this document is “meters per second”, but alternative “units” are available if the corresponding attribute is set. However, the tools currently implemented against this specification assume units of “meters” and “meters per seconds” for distance and velocity values, respectively. The “azimuth” angle is with respect to due North (GPS coordinates) or the y-axis (Cartesian coordinates) and **MUST** be in the range of $\pm 360^\circ$. The optional “elevation” angle **MUST** be in the range of $\pm 90^\circ$ and sets the portion of the motion velocity applied towards altitude (or z-axis) position change. Note that at “elevation” values of $+90^\circ$ or -90° , there is no horizontal motion in the bearing direction given by the “azimuth” angle (i.e. the motion is applied 100% as a change in altitude or z-axis position). The default “elevation” value is 0.0 degrees (i.e. no vertical motion).

The format of the “vector” element is:

```
<vector>
  <velocity>metersPerSec</velocity>
  <azimuth>degrees</azimuth >
  <elevation>degrees</elevation >
</vector>
```

The transition to the vector motion is simply to begin the indicated motion direction/velocity from the current location. Note that since the “vector” motion primitive contains no absolute location parameters, it is very applicable for “PatternDefinition” uses where relative motion behaviors can be specified from an arbitrary initial location.

3.1.4. "circle" Primitive

The “circle” primitive is used specify motion along the perimeter of a circle of a specified “radius” (in meters by default) about a “center” location at a given “velocity” (again, “meters per second” by default). The “altitude” (or “z-axis”) coordinate is fixed and indicated as part of the “center” location specification. Note that the “center” may be omitted and will be assumed to be the current node location according to the motion plan (i.e. based upon the prior node location and/or motion specified). The “circle” also has an optional “revs” element that indicates the maximum number (or fractional number) of circle revolutions to complete before motion is completed (and halted if no subsequent motion is specified). Note the “revs” value is a floating-point number and thus can specify partial completion of the specified circle as desired. If the “revs” are completed before any specified “duration” time expires, then the motion remains halted at the last location until the “duration” expires before proceeding to the next motion primitive.

The default format of the “circle” element is:

```
<circle>
  <center>lat,lon[,alt]</center>
  <radius>meters</radius>
  <velocity>metersPerSec</velocity>
  <revs>value</revs>
</circle>
```

The transition to the “circle” motion from a prior location is to set an intermediate waypoint motion pattern (using the circle motion velocity) directly towards the “center” location that terminates on the perimeter of the circle. In the case the prior location is within the radius of the circle, an outward-spiraling motion is used to reach the circle perimeter with the spiral progressing at a fixed angular velocity based on the circle’s objective “radius” and “velocity”. Note that the time consumed by this transition motion is considered part of motion “duration” time limit, if applicable.

3.1.5. "loiter" Primitive

(TBD – describe the “loiter” motion primitive - can loiter (i.e. “hover” in a circle) around another motion primitive or pattern at an offset radius and height - see example for meantime)

3.1.6. "randpoint" Primitive

The "randpoint" primitive can be used to generate randomly selected node waypoints or locations. Each time a "randpoint" primitive is process it may generate a random waypoint or location within the bounding location box, time, and speed bounds of the given RandpointGenerator. Note that the named RandpointGenerator MUST be defined within a previous or within the same file for the "randpoint" to be successfully processed.

The format of the “randpoint” element is:

```
<randpoint seed="1" mode="variable">generatorName</randpoint>
```

The optional "seed" attribute value can be used to initialize the random number generator of the referenced RandpointGenerator instance. The default seed value is "1". An example use of this seed attribute would be to create a repeatable "pattern" compromised of random waypoints. The "seed" attribute could be used on the first "randpoint" in the pattern to ensure the random number generation state for the associated RandpointGenerator is set to a specific value at the start of pattern generation.

The optional "mode" attribute is one of three possible behaviors that are observed when the "randpoint" primitive is processed:

1. "variable" - the default behavior (when no "mode" is specified) results in a fresh random waypoint destination each time the given "randpoint" is processed (i.e. if it is embedded within a "pattern").
2. "fixed" - In this "mode", a fresh waypoint destination is generated ONLY on the first time the "randpoint" is processed (i.e. if it is embedded within a pattern). On subsequent encounters of an embedded fixed "randpoint", the same destination location is assigned to the waypoint, but with potentially different random speed within any specified speed or time bounds for the associated RandpointGenerator. A "pattern" with a mix of "variable" and "fixed" randpoints can be used so that multiple nodes can periodically converge (in time and/or space given the bounds) in a repeated fashion at the "fixed" (but randomly generated) location(s).
3. "instant" - In this "mode" a "waypoint" is NOT generated, but instead a random "location" is generated to which the node instantly "teleports". Nodes MAY be assigned an instant "randpoint" instead of a explicit "location" as an initial location within MotionPlan documents. The location bounding box of the associated RandpointGenerator is used, but any time or speed bounds are ignored.

The following example illustrates the use of the "randpoint" primitive to set random initial locations for nodes and a subsequent random waypoint. A "pattern" is defined since the 3 Nodes shown follow the same pattern (random initial location and random waypoint).

```
<MotionPlan>
  <RandpointGenerator name="rgen">
    <minLocation type="gps">38.778750, -77.082917</minLocation>
    <maxLocation type="gps">38.895139, -76.969306</maxLocation>
    <minVelocity>5.0</minVelocity>
    <maxVelocity>25.0</maxVelocity>
    <minTime>10.0</minTime>
    <maxTime>180.0</maxTime>
    <seed>20212</seed>
  </RandpointGenerator>

  <PatternDefinition name="rand">
    <randpoint mode="instant">rgen</randpoint>
    <randpoint>rgen</randpoint>
  </PatternDefinition>

  <Node id="node01">
    <pattern>rand</pattern>
```

```

</Node>

<Node id="node02">
  <pattern>rand</pattern>
</Node>

<Node id="node03">
  <pattern>rand</pattern>
</Node>

</MotionPlan>

```

Note that if the first "randpoint" in the PatternDefinition had its optional "seed" attribute, then all three nodes would end up with the same initial random location and subsequent random waypoint (including velocity) being generated.

3.1.7. "pause" Primitive

The "pause" primitive element is provided to specify intervals of immobility from the completion of a prior motion primitive or pattern before beginning transition to a subsequent motion primitive or pattern.

The format of the "pause" element is:

```
<pause>seconds</pause>
```

where "seconds" is a floating point value greater than or equal to zero.

3.1.8. "pattern" Primitive

The "pattern" primitive identifies (by name) a motion sequence that was previously-defined using the "PatternDefinition" element described below. The "pattern" element also has an optional "repeat" attribute that can be used to indicate how many times the motion sequence should repeat before completing and proceeding to any subsequent motion. The default "repeat" value is "0" indicating the motion pattern is executed once while a negative "repeat" value indicates the pattern should be repeated indefinitely.

The format of the "pattern" element is:

```
<pattern repeat="0">patternName</pattern>
```

Note that, as with other motion primitives, the "duration" attribute can be applied to limit the time period during which the motion pattern is executed before transitioning to the next motion, if any. If not "duration" is specified, the "pattern" motion is considered complete when the number of "repeats" has occurred.

3.1.9. "mark" and "wait" Primitives

The "mark" and "wait" primitive elements provide a special planning mechanism that can be used in the MotionPlan documents (and potentially other, to-be-defined EmulationScript planning document types) to annotate (i.e., "mark") a specific emulation script event/time with a "name" and subsequently specify other planning action(s) with a dependency (i.e., "wait") on that annotated cue (i.e. "marker name"). For example, if a "mark" primitive follows a "waypoint" primitive in a MotionPlan, the given "mark" (identified by "name") is to be cached in a resultant EmulationScript at the Event time coresponding to `_when_` that waypoint destination is reached (or its "duration" expires). Other plans may then specify a dependency to have an action take place upon the named "mark" event.

The "mark" and "wait" `<markerName>` values are considered *global* across the set of planning documents being processed. In the future, when additional planning documents (i.e. in addition to MotionPlan) documents are supported, the "mark" and "wait" primitives may be used to cue Node motion events off of other planning events or vice versa (e.g. Trigger a planned "comms" transmission off of a Node's arrival to a waypoint). Since the `<markerName>` instances are global, the occurrences of "mark `<markerName>`" MUST be unique with respect to execution of the planning documents. Thus, the "mark" primitive SHOULD NOT typically be embedded within a "pattern" primitive. In the future, a form of context-specific "mark" primitive may be defined that allows a `<markerName>` to be instantiated within the namespace of some specific planning element (e.g. annotations implicitly marked

with a concatenation of <nodeName:markerName>) that would allow embedding of "mark" annotations within "patterns.

The format of the "mark" and "wait" primitive elements is, respectively:

```
<mark>markerName</mark>
```

and

```
<wait>markerName</wait>
```

The intended use of the "mark" and "wait" primitives is to enable coordination and dependencies of multiple "plans" and their actions with one another. For example, if one MotionPlan document (or element), identifies a "mark" after a given waypoint, another MotionPlan may use the "wait" primitive to key other MotionPlan actions to occur when the given waypoint is reached (by the given "Node"). More specifically, this allows one to specify motion of one or more Nodes that is dependent on the actions of another Node. The "mark" and "wait" primitives allow script generation tools to parse planning documents (e.g. MotionPlan documents) and iteratively process them to create a final EmulationScript that implements the series of dependent actions. The following MotionPlan excerpt provides an example of "mark" and "wait" usage:

```
<MotionPlan>
  <Node id="node1">
    <location type="cartesian">100,100,0</location>
    <waypoint>
      <destination type="cartesian">100,200</destination>
      <velocity>10.0</velocity>
    </waypoint>
    <mark>node1Waypoint1</mark>
    <wait>node2Waypoint1</wait>
    <waypoint>
      <destination type="cartesian">100,300</destination>
      <velocity>10.0</velocity>
    </waypoint>
    <mark>node1Waypoint2</mark>
    <waypoint>
  </Node>

  <Node id="node2">
    <location type="cartesian">200,100,0</location>
    <wait>node1Waypoint1</wait>
    <waypoint>
      <destination type="cartesian">200,200</destination>
      <velocity>10.0</velocity>
    </waypoint>
    <mark>node2Waypoint1</mark>
    <wait>node1Waypoint2</wait>
    <waypoint>
      <destination type="cartesian">200,300</destination>
      <velocity>10.0</velocity>
    </waypoint>
    <mark>node2Waypoint2</mark>
    <waypoint>
  </Node>
</MotionPlan>
```

In this example, "node1" begins at an initial location of (100,100) and immediately begins moving towards the waypoint destination of (100,200). However, "node2" waits until "node1" reaches its first waypoint (marked as "node1Waypoint1") before beginning its first motion. Similarly, "node1" waits at its first waypoint until "node2" reaches its first waypoint (marked as "node2Waypoint1") before moving to its second waypoint. In this example, the use of "mark" and "wait" primitives allow a staggered motion among the two nodes to be planned.

3.2. Motion Pattern Definition

A “PatternDefinition” element is provided to encapsulate a concatenated set of motion primitives and assign a “name” to the defined pattern. Then, the “pattern” element can be used to refer to that definition by name and used as a motion primitive to specify node motion or even as part of other, more complex “PatternDefinition” instances. The content of the “PatternDefinition” is a set of the motion primitive elements previously described and the “name” attribute is used to assign a name string to the pattern defined. The format of the “PatternDefinition” element is:

```
<PatternDefinition name="patternName">
  <motion primitive 1/>
  <motion primitive 2/>
  ...
</PatternDefinition>
```

The purpose of the “PatternDefinition” element is to allow a motion “circuit” or “path” to be defined that might be re-used by multiple mobile nodes at different “Node:offsetTime” values (see below). “PatternDefinition” elements are top-level elements of “MotionPlan” documents.

3.3. Random Waypoint Generator Definition

A “RandpointGenerator” element is provided to define a random waypoint generator instance that can be referenced by name by <randpoint> motion primitives to generate a random waypoints as part of a motion plan. Each “RandpointGenerator” instance has a set of parameters that control the nature of the waypoints generated (e.g. x,y,z location bounding box, min/max velocity bounds, etc).

```
<RandpointGenerator name="generatorName">
  <seed> value </seed>
  <minLocation> x,y[,z] min bounds </minLocation>
  <maxLocation> x,y[,z] max bounds </maxLocation>
  <minVelocity> value< /minVelocity>
  <maxVelocity> value </maxVelocity>
  <minTime> value< /minTime>
  <maxTime> value </maxTime>
</RandpointGenerator>
```

(TBD - describe what the parameters roles are in random waypoint generation)

3.4. Node Motion Plan Specification

In addition to “PatternDefinition” elements, “MotionPlan” documents can contain one or more “Node” elements that encapsulate motion primitives and/or patterns that are assigned to the node instance indicated by the “Node:id” attribute. The Node element may start with an *optional* “offsetTime” element. It then has a mandatory “location” element that specifies the initial location of the node. After that, any number of motion primitives and/or patterns may be applied to specify the node’s motion. The format of the “Node” element is:

```
<Node id="nodeId">
  <offsetTime>seconds</offsetTime>
  <offsetLocation>
    <distance>meters</distance>
    <azimuth>degrees</azimuth>
    <elevation>degrees</elevation>
  </offsetLocation>
  <location>lat,lon,alt</location>
  <motion primitive or pattern 1>
  <motion primitive or pattern 2>
  <motion primitive or pattern 3>
  ...
</Node>
```

As mentioned, multiple “Node” elements may be included within a “MotionPlan” document to describe motion for multiple nodes. While it may be possible to specify disjoint motions for a single node through use of multiple “Node” elements with different offset times, this practice is not generally recommended due to complexity.

4. Example Utilities

The EmulationScript schema distribution also includes source code for a number of useful command-line utilities that implement the motion planning and generation behaviors that are defined. The two principal utilities are:

- mp “motion planner” tool that parses MotionPlan XML documents and generates “EmulationScript” XML.
- mg “motion generator” tool that parses EmulationScript XML documents and can generate location updates in various formats (currently SDT) at a specified interval.

There are some additional tools for converting to/from some different file formats used by some mobile network modeling tools. The accompanying "ScriptToolsUserGuide" document describes the usage of the "mp", "mg" and these other utilities. The suite of tools and their capabilities will be expanded over time.

5. Alternative File Formats

TBD – describe “simple text” format for select portion of emulation script content (e.g., node location/mobility only). For example, something like the following, but perhaps with some type of key/header line that identifies the numeric content:

```
#key: time, nodeId, lat, lon, alt, [waypoint: lat, lon, alt, vel]  
<time>, <nodeId>, <lat>, <lon>, <alt>[, <lat>, <lon>, <alt>, <vel>]
```

6. Usage Notes

(TBD - Describe some example usages of the EmulationScript document types.)