# GEXF 1.3 Primer

GEXF Working Group

March 3, 2022

**Abstract**

GEXF Primer is a non-normative document intended to provide an easily readable description of the GEXF facilities, and is oriented towards quickly understanding how to create GEXF documents. This primer describes the language features through examples which are complemented by references to normative texts. Specification is in RelaxNG Compact grammar.

## Contents

# 1 Introduction

This document, GEXF Primer, provides an description of GEXF, and should be used alongside the formal descriptions of the language contained in the GEXF specification. The intended audience of this document includes application developers whose programs read and write GEXF files, and users who want to communicate with programs using GEXF import/export. The text assumes that you have a basic understanding of XML 1.0 and XML-Namespaces. Basic knowledge of XML Schema is also assumed for some parts of this document. Each major section of the primer introduces new features of the language, and describes those features in the context of concrete examples.

Section 2 covers the basic mechanisms of GEXF. It describes how to declare a simple graph by defining its nodes and edges and how to add simple user data to the graph.

Section 5 describes dynamic graph model (i.e. graphs over time).

Section 6 describes mechanisms for extending GEXF to add specific data with the Visualization module in example.

The primer is a non-normative document, which means that it does not provide a definitive specification of the GEXF language. The examples and other explanatory material in this document are provided to help you understand GEXF, but they may not always provide definitive answers. In such cases, you will need to refer to the GEXF specification, and to help you do this, we provide many links pointing to the relevant parts of the specification.

In this document, we may use the terms network and graph interchangeably.

# 2 Basic Concepts

The purpose of a GEXF document is to define a graph representing a network. Let us start by considering the minimal graph shown in the figure below. It contains 2 nodes and 1 edge.

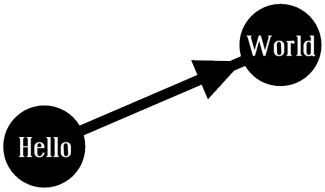


Figure 1: Hello-world graph

## 2.1 A Simple Graph

This is a dummy graph:

Listing 1: Hello world!

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.3"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.gexf.net/1.3
                          http://www.gexf.net/1.3/gexf.xsd"
      version="1.3">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gephi.org</creator>
    <description>A hello world! file</description>
  </meta>
  <graph defaultedgetype="directed">
    <nodes>
      <node id="0" label="Hello"/>
      <node id="1" label="Word"/>
    </nodes>
    <edges>
      <edge source="0" target="1"/>
    </edges>
  </graph>
</gexf>
```

The GEXF document consists of a gexf element and a variety of subelements. In the remainder of this section we will discuss these elements in detail and show how they define a graph.

## 2.2 Header

In this section we discuss the parts of the document which are common to all GEXF documents, basically the gexf element and the meta declaration.

Listing 2: Header

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.3"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.gexf.net/1.3
                          http://www.gexf.net/1.3/gexf.xsd"
    version="1.3">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gephi.org</creator>
    <description>A hello world! file</description>
    <keywords>basic, web</keywords>
  </meta>
  ...
</gexf>
```

The first line of the document is an XML process instruction which defines that the document adheres to the XML 1.0 standard and that the encoding of the document is UTF-8, the standard encoding for XML documents. Of course other encodings can be chosen for GEXF documents.

The second line contains the root-element element of a GEXF document: the gexf element. The gexf element, like all other GEXF elements, belongs to the namespace http://www.gexf.net/1.3. For this reason we define this namespace as the default namespace in the document by adding the XML Attribute xmlns="http://www.gexf.net/1.3" to it. The two other XML Attributes are needed to specify the XML Schema for this document. In our example we use the standard schema for GEXF documents located on the gexf.net server. The first attribute, xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance", defines xsi as the XML Schema namespace. The second attribute, xsi:schemaLocation ="http://www.gexf.net/1.3 http://www.gexf.net/1.3/gexf.xsd", defines the XML Schema location for all elements in the GEXF namespace.

The XML Schema reference is not required, but it provides means to validate the document and is therefore strongly recommended.

The meta element contains additional information about the network. Element leafs are assumed to be text, and lastmodifieddate is an international standard date (yyyy-mm-dd). The graph element must be declared after the meta element.

## 2.3 Network Topology

The network topology structure containing nodes and edges is called the graph. A graph is, not surprisingly, denoted by a graph element. Nested inside a graph element are the declarations of nodes and edges. A node is declared with the node element inside a nodes element, and an egde with the edge element inside an edges element.

Listing 3: The definition of the graph

```
<graph defaultedgetype="directed">
  <nodes>
    <node id="0" label="Hello" />
    <node id="1" label="Word" />
    ...
  </nodes>
  <edges>
    <edge source="0" target="1" weight="3.167" />
    ...
  </edges>
</graph>
```

### 2.3.1 Declaring a Graph

Graphs in GEXF are mixed, in other words, they can contain directed and undirected edges at the same time. If no direction is specified when an edge is declared, the default direction defaultedgetype is applied to the edge.

The default direction is declared as the optional XML-attribute defaultedgetype of the graph element. The three possible values for this XML-attribute are *directed*, *undirected* and *mutual*. Note that the default direction is optional and would be assumed as *undirected*.

The optional XML-attribute mode set the kind of network: static or dynamic. The latter provides time support (see the section 5 on Dynamics). Static mode is assumed by default.

The edges element must be declared after the nodes element.

Listing 4: An empty graph!

```
<graph>
  <nodes>
  </nodes>
  <edges>
  </edges>
</graph>
```

### 2.3.2 Declaring a Node

Nodes in the graph are declared by the node element. Each node has an identifier, which must be unique within the entire document, i.e., in a document there must be no two nodes with the same identifier. The identifier of a node is defined by the XML-attribute id, which is a string. Each node also may have a XML-attribute label that acts as a description, which is a string.

Listing 5: A node!

```
<node id="0" label="Hello world" />
```

### 2.3.3 Declaring an Edge

Edges in the graph are declared by the edge element. Each edge must define its two endpoints with the XML-Attributes source and target. The value of the source, resp. target, must be the identifier of a node in the same document.

The identifier of an edge is defined by the XML-Attribute id and is optional. There is no order notion applied to edges.

Edges with only one endpoint, also called loops, selfloops, or reflexive edges, are defined by having the same value for source and target.

Each edge can have a optional XML-attribute label, which is a string.

The optional XML-attribute type declares if the edge is *directed*, *undirected* or *mutual* (directed *from source to target and from target to source*). If the direction is not explicitely defined, the default direction is applied to this edge as defined in the enclosing graph via the 'defaultedgetype' attribute. The edge's endpoint are called source and target regardless whether the edge is directed or not.

The weight of the edge is set by the optional XML-attribute weight and is a double. By default, the weight is 1.0 and zero should be avoided although it's not explicitly forbidden.

Assuming two nodes having respectively the id value set to *0* and *1*:

Listing 6: An edge!

```
<edge source="0" target="1"/>
```

Listing 7: A more complete edge

```
<edge id="0" source="0" target="1" type="directed" weight="2.4" />
```

### 2.3.4 Parallel edges

A multigraph is a graph where multiple edges can exist between two nodes. GEXF supports this type of graph. Note that this is different from hypergraph, which GEXF doesn't support.

Parallel edges must include the additional *kind* attribute to characterise the edge. The triplet source-target-kind must still be unique. In other words, only parallel edges of different kinds can exist in GEXF. The *kind* attribute is a string. In case the graph doesn't have any parallel edges, *kind* can be omitted.

Listing 8: Parallel edges

```
<edge source="0" target="1" weight="1.0" kind="friend"/>
<edge source="0" target="1" weight="1.0" kind="neighbor"/>
```

## 2.4 Network Data

In the previous section we discussed how to describe the topology of a graph in GEXF. In this section we see how additional data can be associated with nodes and edges.

A bunch of data can be stored within attributes. The concept is the same as table data or SQL. An attribute has a title/name and a value. Attribute's name/title must be declared for the whole graph. It could be for instance "degree", "valid" or "url". Besides the name of the attribute a column also contains the type.

### 2.4.1 Data types

GEXF uses the XML Schema Data Types (XSD 1.1) for the following primitives: string, integer, long, float, double, boolean, short, byte, date, and anyURI.

In addition, GEXF supports additional Data Types: bigdecimal, biginteger, char, liststring, listboolean, listinteger, listlong, listfloat, listdouble, listbyte, listshort, listbigdecimal, listinteger and listchar.

### 2.4.2 Attributes Example

Each Node of this graph has three attributes : an url, an indegree value and a boolean for french websites which is set to *true* by default.

Listing 9: A (small) Web Graph

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.3"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.gexf.net/1.3
                          http://www.gexf.net/1.3/gexf.xsd"
    version="1.3">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gephi.org</creator>
    <description>A Web network</description>
  </meta>
  <graph defaultedgetype="directed">
    <attributes class="node">
      <attribute id="0" title="url" type="string"/>
      <attribute id="1" title="indegree" type="float"/>
      <attribute id="2" title="frog" type="boolean">
        <default>true</default>
      </attribute>
    </attributes>
    <nodes>
      <node id="0" label="Gephi">
        <attvalues>
          <attvalue for="0" value="http://gephi.org"/>
          <attvalue for="1" value="1"/>
        </attvalues>
      </node>
      <node id="1" label="Webatlas">
        <attvalues>
          <attvalue for="0" value="http://webatlas.fr"/>
          <attvalue for="1" value="2"/>
        </attvalues>
      </node>
      <node id="2" label="RTGI">
        <attvalues>
          <attvalue for="0" value="http://rtgi.fr"/>
          <attvalue for="1" value="1"/>
```

```
        </attvalues>
      </node>
      <node id="3" label="BarabasiLab">
        <attvalues>
          <attvalue for="0" value="http://barabasilab.com"/>
          <attvalue for="1" value="1"/>
          <attvalue for="2" value="false"/>
        </attvalues>
      </node>
    </nodes>
    <edges>
      <edge source="0" target="1"/>
      <edge source="0" target="2"/>
      <edge source="1" target="0"/>
      <edge source="2" target="1"/>
      <edge source="0" target="3"/>
    </edges>
  </graph>
</gexf>
```

### 2.4.3 Declaring Attributes

Attributes are declared inside an attributes element. The XML-attribute class apply nested attributes on nodes (*node* value) or edges (*edge* value). You need to specify the data type in type and may specify a default value.

Listing 10: Attributes Definition

```
<graph mode="static">
  <attributes class="node">
    <attribute id="0" title="my-text-attribute" type="string"/>
    <attribute id="1" title="my-int-attribute" type="integer"/>
    <attribute id="2" title="my-bool-attribute" type="boolean"/>
  </attributes>
  <attributes class="edge">
    <attribute id="0" title="my-float-attribute" type="float">
      <default>2.0</default>
    </attribute>
  </attributes>
  ...
</graph>
```

Note about the *options* attribute: it defines the available values, separated by a coma and surrounded by brackets. It is both used as a type constraint and for parser optimization. The combined default value must be an available option, like the following example.

Listing 11: Options

```
<graph mode="static">
  <attributes class="node">
    <attribute id="0" title="my-string-attribute" type="string">
        <default>foo</default>
        <options>[foo, bar, foobar]</options>
    </attribute>
    <attribute id="1" title="my-integer-attribute" type="integer">
        <default>5</default>
        <options>[1, 2, 5, 6]</options>
    </attribute>
  </attributes>
  ...
</graph>
```

### 2.4.4 Defining Attribute Values

You may understand attributes while looking at this node definition. Besides native fields (id, label), node values are set for three attributes. Omitting an attribute will set the default value as its value. If no default value is set, the value will be empty.

Listing 12: Node Attributes

```
<node id="0" label="Hello world">
  <attvalues>
    <attvalue for="0" value="samplevalue"/>
    <attvalue for="1" value="1831"/>
    <attvalue for="2" value="true"/>
  </attvalues>
</node>
```

Listing 13: Edge Attributes

```
<edge source="0" target="1">
  <attvalues>
    <attvalue for="0" value="1.5"/>
  </attvalues>
</edge>
```

### 2.4.5 List Attributes

GEXF offers list data types liststring, listboolean, listinteger, listlong, listfloat, listdouble, listbyte, listshort, listbigdecimal, listinteger and listchar.

The list format is coma separated surrounded by brackets. For instance [1, 2, 3] or [foo, bar]. It supports single and double quotes for text, for instance ['foo', 'bar']. An empty list is simply [].

Note that list attributes are an unsafe types! Values are therefore parsed, and this parsing may fail in certain complex cases.

Listing 14: Liststring Definition

```
<graph mode="static">
  <attributes class="node">
    <attribute id="0" title="my-liststring-attribute" type="liststring">
    </attribute>
  </attributes>
  ...
</graph>
```

Listing 15: Liststring usage

```
<node id="0" label="Hello world">
  <attvalues>
    <attvalue for="0" value="[foo, bar]"/>
  </attvalues>
</node>
```

A complete example:

Listing 16: Boolean version

```
<attributes>
```

```
 <attribute id="0" title="hobby" type="liststring">
  </attribute>
</attributes>
<nodes>
 <node id="42" label="a node">
      <attvalues>
          <attvalue for="0" value="[dance, ski]">
      </attvalues>
   </node>
</nodes>
```

Also note that when the *options* attribute is used for lists, it gives all possible elements of the list:

Listing 17: Valid values

```
<attributes>
 <attribute id="0" title="foo-attr" type="liststring">
   <options>[foo1, foo2, foo3]</options>
 </attribute>
</attributes>
<nodes>
 <node id="42" label="node A">
      <attvalues>
          <attvalue for="0" value="[foo3, foo2]">
      </attvalues>
   </node>
 <node id="43" label="node B">
      <attvalues>
          <attvalue for="0" value="">
      </attvalues>
   </node>
 <node id="44" label="node C">
      <attvalues>
          <attvalue for="0" value="[foo1, foo2, foo3]">
      </attvalues>
   </node>
</nodes>
```

Listing 18: Invalid value foo4

```
...
<node id="42" label="node A">
   <attvalues>
       <attvalue for="0" value="[foo1, foo4]">
   </attvalues>
</node>
```

# 3 Advanced Concepts I: Hierarchy structure

## 3.1 Introduction

GEXF format allows creating hierarchical graph structure essentially for clustering representation. We modelize both a tree structure of ancestors and descendents, and a flat graph of nodes bound by edges.

Two ways are available:

1. Nodes can simply host other nodes and so on.
2. Each node refers to a parent node id with the XML-attribute pid.

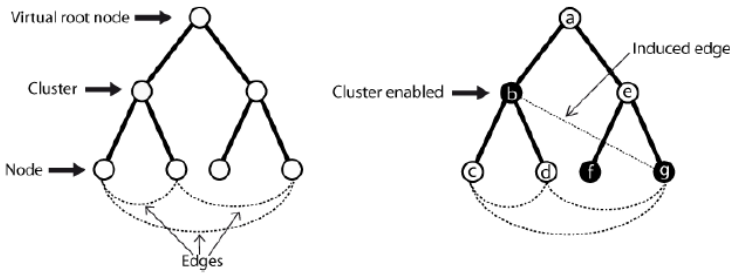


Figure 2: Graph tree with a virtual edge from a cluster to a leaf

The first style is preferred when the structure written is previously ordered. Sequential reading of this kind of GEXF is safe because no node reference is used. But in the case your program can't provide this, the second way allows writing (and then reading) nodes randomly, but linear reading is at your own risks.

## 3.2 Sequential-safe Reading

Listing 19: First way

```
<graph mode="static" defaultedgetype="directed">
    <nodes>
      <node id="a" label="Kevin Bacon">
        <nodes>
          <node id="b" label="God">
            <nodes>
              <node id="c" label="human1"/>
              <node id="d" label="human2"/>
            </nodes>
          </node>
          <node id="e" label="Me">
            <nodes>
              <node id="f" label="frog1"/>
              <node id="g" label="frog2"/>
            </nodes>
          </node>
        </nodes>
      </node>
    </nodes>
    <edges>
        <edge source="b" target="e" />
        <edge source="c" target="d" />
        <edge source="g" target="b" />
        <edge source="f" target="a" />
    </edges>
</graph>
```

Note that edges are not necessarily written at the end:

Listing 20: First way with edges inside clusters

```
<graph mode="static" defaultedgetype="directed">
    <nodes>
      <node id="a" label="Kevin Bacon">
        <nodes>
          <node id="b" label="God">
            <nodes>
```

```
            <node id="c" label="human1"/>
            <node id="d" label="human2"/>
          </nodes>
          <edges>
            <edge source="c" target="d" />
          </edges>
        </node>
        <node id="e" label="Me">
          <nodes>
            <node id="f" label="frog1"/>
            <node id="g" label="frog2"/>
          </nodes>
        </node>
      </nodes>
      <edges>
        <edge source="b" target="e" />
        <edge source="f" target="a" />
        <edge source="g" target="b" />
      </edges>
    </node>
  </nodes>
  <edges />
</graph>
```

## 3.3 Random Writing

If you can't structure your graph topology before writing a GEXF file, you may use the second style. Nodes sent to Gephi from a live data source, i.e. a web crawler, are written like this. Note that edges are always written randomly.

Listing 21: Second way

```
<nodes>
  <node id="a" label="Kevin Bacon" />
  <node id="b" label="God" pid="a" />
  <node id="c" label="human1" pid="b" />
  <node id="d" label="human2" pid="b" />
  <node id="e" label="Me" pid="a" />
  <node id="f" label="frog1" pid="e" />
  <node id="g" label="frog2" pid="e" />
</nodes>
```

With using pid, node order doesn't matter. An implementation should manage the case when a node reference (pid) is used before the node declaration. This listings could also be:

Listing 22: Second way randomized

```
<nodes>
  <node id="g" label="frog2" pid="e" />
  <node id="a" label="Kevin Bacon" />
  <node id="c" label="human1" pid="b" />
  <node id="b" label="God" pid="a" />
  <node id="e" label="Me" pid="a" />
  <node id="d" label="human2" pid="b" />
  <node id="f" label="frog1" pid="e" />
</nodes>
```

# 4 Advanced Concepts II: Phylogeny structure

Multiple parents can be addressed with the following syntax, where a and b are c's parents:

Listing 23: Multiple parents

```
<nodes>
  <node id="a" label="cheese">
  <node id="b" label="cherry">
  <node id="c" label="cake">
    <parents>
      <parent for="a" />
      <parent for="b" />
    </parents>
  </node>
</nodes>
```

# 5 Advanced Concepts III: Dynamics

As networks dynamics is a growing topic of research, GEXF format includes extensive time support. Enable it by setting the mode attribute of the graph to *dynamic*.

Listing 24: Dynamic Enabled!

```
<graph mode="dynamic">
  ...
</graph>
```

Both network topology and attributes can be dynamic. Time values in GEXF can be represented either via numbers or date or date plus time. In addition, the time representation can either use timestamps or intervals. We'll go into more details later. In the example below is uses the date time format and the default interval time representation.

## 5.1 Example

Listing 25: A (small) Dynamic Web Graph with date format

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.3"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.gexf.net/1.3
                          http://www.gexf.net/1.3/gexf.xsd"
      version="1.3">
  <graph mode="dynamic" defaultedgetype="directed" timeformat="date">
    <attributes class="node" mode="static">
      <attribute id="0" title="url" type="string"/>
      <attribute id="1" title="frog" type="boolean">
        <default>true</default>
      </attribute>
    </attributes>
    <attributes class="node" mode="dynamic">
      <attribute id="2" title="indegree" type="float"/>
    </attributes>
    <nodes>
      <node id="0" label="Gephi" start="2009-03-01">
        <attvalues>
          <attvalue for="0" value="http://gephi.org"/>
```

```
        <attvalue for="2" value="1"/>
      </attvalues>
    </node>
    <node id="1" label="Webatlas">
      <attvalues>
        <attvalue for="0" value="http://webatlas.fr"/>
        <attvalue for="2" value="1" end="2009-03-01"/>
        <attvalue for="2" value="2" start="2009-03-01" end="2009-03-10"/>
        <attvalue for="2" value="1" start="2009-03-11"/>
      </attvalues>
    </node>
    <node id="2" label="RTGI" end="2009-03-10">
      <attvalues>
        <attvalue for="0" value="http://rtgi.fr"/>
        <attvalue for="2" value="0" end="2009-03-01"/>
        <attvalue for="2" value="1" start="2009-03-01"/>
      </attvalues>
    </node>
    <node id="3" label="BarabasiLab">
      <attvalues>
        <attvalue for="0" value="http://barabasilab.com"/>
        <attvalue for="1" value="false"/>
        <attvalue for="2" value="0" end="2009-03-01"/>
        <attvalue for="2" value="1" start="2009-03-01"/>
      </attvalues>
    </node>
  </nodes>
  <edges>
    <edge source="0" target="1" start="2009-03-01"/>
    <edge source="0" target="2"
          start="2009-03-01" end="2009-03-10"/>
    <edge source="1" target="0" start="2009-03-01"/>
    <edge source="2" target="1" end="2009-03-10"/>
    <edge source="0" target="3" start="2009-03-01"/>
  </edges>
 </graph>
</gexf>
```

## 5.2 Time representation

GEXF aims to offer equivalent features for both time representations: *timestamp* and *intervals*. Timestamps are unique points in time while intervals can represent a duration and may include infinity as one of its bounds.

The two time representations can't however be mixed in GEXF and should be defined and consistent throughout the graph.

Listing 26: Timestamp Time Representation

```
<graph mode="dynamic" timerepresentation="timestamp">
...
</graph>
```

The default time representation is interval to be backward compatible with previous GEXF versions as the timestamp representation is only introduced in 1.3.

Listing 27: Interval Time Representation

```
<graph mode="dynamic" timerepresentation="interval">
...
</graph>
```

Note that intervals are always defined with a pair of attributes *start* and *end*. Omitting one of the two translates into an infinity bound. For instance, node id="0" label="Foo" start="2005" translates into a [2005, +INF] interval for the node.

## 5.3 Time format

Time in GEXF can be encoded in two ways: via numbers or via dates and times. This is controlled via the timeformat attribute set on the graph element.

As a number, it is an *integer* or a *double*. As a date, it is encoded as an international standard *date* (yyyy-mm-dd) or a *dateTime* defined by the corresponding XSD Datatype. If omitted, the default type is *double*.

Listing 28: Double format

```
<graph mode="dynamic" timeformat="double">
  <nodes>
    <node id="0" label="Gephi" start="1.0" />
    <node id="1" label="Gexf" start="2.0" end="3.0" />
    ...
  </nodes>
</graph>
```

Listing 29: Date format

```
<graph mode="dynamic" timeformat="date">
  <nodes>
    <node id="0" label="Gephi" start="2003-01-01" />
    <node id="1" label="Gexf" start="2003-01-01" end="2004-01-01" />
    ...
  </nodes>
</graph>
```

Listing 30: Datetime format (also with timezone)

```
<graph mode="dynamic" timeformat="dateTime">
  <nodes>
    <node id="0" label="Gephi" start="2012-09-12T15:04:01" />
    <!-- With timezone -->
    <node id="1" label="Gexf" start="2012-11-04T11:00:01+03:00"
      end="2012-15-04T11:00:01+03:00" />
    ...
  </nodes>
</graph>
```

The timezone can also set globally for the entire graph via the timezone graph attribute.

Listing 31: Global timezone setting

```
<graph mode="dynamic" timeformat="dateTime" timezone="America/Los_Angeles">
...
</graph>
```

## 5.4 Dynamic Topology

Graph nodes and edges can exist at various points in time. Existence in time can be represented in different ways.

### 5.4.1 Unique interval or timestamp

The simplest way is by setting a pair of start and end for the interval time representation and timestamp for the timestamp time representation.

Listing 32: Node Scope Example with Intervals

```
<graph mode="dynamic" timerepresentation="interval">
  <nodes>
    <node id="0" label="Hello" start="2019-01-01" end="2019-02-01" />
    <node id="1" label="World" start="2019-01-15" end="2019-03-20" />
    ...
  </nodes>
</graph>
```

Listing 33: Node Scope Example with Timestamps

```
<graph mode="dynamic" timerepresentation="timestamp">
  <nodes>
    <node id="0" label="Hello" timestamp="2019-01-01" />
    <node id="1" label="World" timestamp="2019-03-20" />
    ...
  </nodes>
</graph>
```

Each edge must declare time limits inside the join scope of its source and target nodes:

- edge.start $\leq$ (source.start and target.start)
- edge.end $\geq$ (source.end and target.end)

Listing 34: Edge Scope Example

```
<nodes>
  <node id="0" label="Hello" start="2009-01-01" end="2009-02-01" />
  <node id="1" label="World" start="2009-01-15" end="2009-03-20" />
  ...
</nodes>
<edges>
  <edge source="0" target="1" start="2009-01-20" end="2009-02-01"/>
</edges>
```

Important: start and end values are inclusive, i.e. the following line is allowed:

Listing 35: Smallest time scope

```
<edge source="0" target="1" start="2009-01-20" end="2009-01-20"/>
```

And of course the end value must be equal or later than the start value.

### 5.4.2 Spells

If a node or an edge exists at multiple timestamps or intervals, we use the concept of spells. Use the xml-element spells for topology like this:

Listing 36: Node with multiple spells (Intervals)

```
<graph mode="dynamic" timeformat="date" timerepresentation="interval">
  <nodes>
    <node id="0" label="Hello">
      <spells>
        <spell start="2009-01-01" end="2009-01-15" />
        <spell start="2009-01-30" end="2009-02-01" />
      </spells>
    </node>
    ...
  </nodes>
</graph>
```

If no start is provided, the spell begins with the network. If no end is provided, the spell ends with the network. If two spells are covering a same period of time, parsers should consider them as a unique spell.

Listing 37: Node with multiple spells (Timestamps)

```
<graph mode="dynamic" timeformat="date" timerepresentation="timestamp">
  <nodes>
    <node id="0" label="Hello">
      <spells>
        <spell timestamp="2009-01-15" />
        <spell timestamp="2009-02-01" />
      </spells>
    </node>
    ...
  </nodes>
</graph>
```

If spells are provided, only their content are taken into account and other time attributes on the element like timestamp will be ignored.

Important: Intervals can't overlap for an element.

Listing 38: Overlapping intervals (not allowed)

```
<graph mode="dynamic" timeformat="date" timerepresentation="interval">
  <nodes>
    <node id="0" label="Hello">
      <spells>
        <spell start="2009-01-01" end="2009-01-15" />
        <spell start="2009-01-05" end="2009-02-01" />
      </spells>
    </node>
  ...
  </nodes>
</graph>
```

### 5.4.3 Alternative to spells

For a more compact spell representation, GEXF also offers timestamps and intervals attributes as lists. It's however less safe as it relies more heavily on the parser. Make sure to check the following examples.

Listing 39: Timestamp list (Date)

```
<graph mode="dynamic" timerepresentation="timestamp" timeformat="date">>
  <nodes>
    <node id="0" label="Hello" timestamps="<[2019-03-20, 2019-03-21]>" />
    ...
```

```
  </nodes>
</graph>
```

Listing 40: Timestamp list (Double)

```
<graph mode="dynamic" timerepresentation="timestamp" timeformat="double">>
  <nodes>
    <node id="0" label="Hello" timestamps="<[1, 2, 21.0, 124.0]>" />
    ...
  </nodes>
</graph>
```

Listing 41: Interval list (Double)

```
<graph mode="dynamic" timerepresentation="timestamp" timeformat="double">>
  <nodes>
    <node id="0" label="Hello" intervals="<[4.0, 14.0]; [21.0, 124.0]>" />
    ...
  </nodes>
</graph>
```

Note that the separator between intervals is a semi-colon as the comma is reserved for the interval itself.

About the weight: dynamic weight can be used with the reserved *title* "weight" in attributes. In dynamic mode, the static XML-attribute *weight* should be ignored if the dynamic one is provided.

## 5.5 Dynamic Data

Node and edges data can take different values over time. Attributes must be declared as dynamic, allowing values to exist for a given time.

### 5.5.1 Declaring Dynamic Attributes

Listing 42: Indegree may change over time

```
<gexf ...>
  ...
  <graph mode="dynamic" defaultedgetype="directed">
    <attributes class="node">
      <attribute id="2" title="indegree" type="float"/>
      ...
    </attributes>
    ...
  </graph>
</gexf>
```

### 5.5.2 Defining Dynamic Values

Attvalues have their scopes limited by the xml-attributes start and end.

Listing 43: Data value changing over time

```
<node id="3" label="BarabasiLab">
  <attvalues>
```

```
    <attvalue for="2" value="0" start="2009-01-01" end="2009-03-01"/>
    <attvalue for="2" value="1" start="2009-03-02" end="2009-03-10"/>
  </attvalues>
</node>
```

Listing 44: Using timestamp representation

```
<node id="3" label="BarabasiLab">
  <attvalues>
    <attvalue for="2" value="15.0" timestamp="2000" />
    <attvalue for="2" value="20.0" timestamp="2005" />
    <attvalue for="2" value="25.0" timestamp="2010" />
  </attvalues>
</node>
```

### 5.5.3 Dynamic Values and Spells

If an attvalue is covering a period out of any spell, this period should be ignored by parsers. In the following example, the day 2009-01-03 is ignored:

Listing 45: Spells and attvalues

```
<gexf ...>
  ...
  <graph mode="dynamic">
    <node id="0" label="Hello">
      <attvalues>
        <attvalue for="0" value="1" start="2009-01-01" end="2009-01-05"/>
      </attvalues>
      <spells>
        <spell start="2009-01-01" end="2009-01-02" />
        <spell start="2009-01-04" end="2009-01-05" />
      </spells>
    </node>
    ...
  </graph>
</gexf>
```

# 6 Advanced Concepts IV: Extending GEXF

GEXF is designed to be easily extensible. Additional namespaces are defined by an XML Schema. The default namespace is always the gexf namespace. Gephi team actually provides a module for storing visualization data called *viz*.

## 6.1 VIZ module

Using the visualization module must be declared by adding the XML Attribute xmlns:viz="http://www.gexf.net/1.3/viz" to the document namespaces. The xsi:schemaLocation attribute includes the XML-Schema declaration of the VIZ module. The RelaxNG Compact specification is available in viz.rnc, and independent XSD declaration in viz.xsd.

Color, position, size and shape are stored as attributes.

### 6.1.1 Node Example

The following gexf contains a node having a color, a position, a shape and a specified size.

Listing 46: VIZ Attributes

```
<gexf xmlns="http://www.gexf.net/1.3"
      xmlns:viz="http://www.gexf.net/1.3/viz">
...
  <node ... >
    <viz:color r="239" g="173" b="66" a="0.5"/>
    <viz:position x="15.783598" y="40.109245" z="0.0"/>
    <viz:size value="2.0375757"/>
    <viz:shape value="disc"/>
  </node>
...
</gexf>
```

### 6.1.2 Edge Example

The following gexf contains an edge having a color, a thickness and a shape.

Listing 47: VIZ Attributes

```
<gexf xmlns="http://www.gexf.net/1.3"
      xmlns:viz="http://www.gexf.net/1.3/viz">
...
  <edge ... >
    <viz:color r="157" g="213" b="78"/>
    <viz:thickness value="5.124"/>
    <viz:shape value="solid"/>
  </edge>
...
</gexf>
```

### 6.1.3 Color

Colors are defined by the RGBA color model. Each XML-attribute value `r`, `g` or `b` is hence an integer from 0 to 255, and the alpha value `a` is a float from 0.0 to 1.0.

Listing 48: VIZ Color Declaration

```
<viz:color r="239" g="173" b="66" a="0.5"/>
```

Colors can alternatively be defined with a unique `hex` attribute. It can still be combined with the alpha parameter

Listing 49: Color Hex Attribute

```
<viz:color hex="#FF7700" alpha="0.5"/>
```

### 6.1.4 Position

Space positions are set in three dimensions called `x`, `y` and `z`. Note that Gephi associates `z` (optional) as the height, and most of layout algorithms only use `x` and `y`. They are floats.

Listing 50: VIZ Position Declaration

```
<viz:position x="15.783598" y="40.109245" z="0.0"/>
```

### 6.1.5 Size

Node size is a scale. It is set to *1.0* by default and is a non-negative float. Network viz softwares assume that an object representing a node of size *2.0* is twice bigger as one of *1.0*.

Listing 51: VIZ Size Declaration

```
<viz:size value="2.0375757"/>
```

### 6.1.6 Thickness

Edge thickness is a scale. It is set to *1.0* by default and is a non-negative float. Network viz softwares assume that an object representing an edge of thickness *2.0* is twice bigger as one of *1.0*.

Listing 52: VIZ Thickness Declaration

```
<viz:thickness value="2.0375757"/>
```

### 6.1.7 Node Shape

Default node is shaped as a disc. Four shapes are proposed: *disc*, *square*, *triangle* and *diamond*. Images require an additional xml-attribute to set their location: uri.

Listing 53: Image Declaration

```
<viz:shape value="image" uri="http://my.image.us/blah.jpg"/>
```

### 6.1.8 Edge Shape

Default edge is shaped as solid. Four shapes are proposed: *solid*, *dotted*, *dashed* and *double*.

# 7 Advices: Parser optimization

This section provides some good tips to write parser-friendly files.

- Always place the edges after the nodes, as it is mandatory since version 1.2. Some parsers, depending on their implementation, may reject an edge if its linked nodes haven't been declared before, due to conceptual or data integrity reason.

- Use the *count* XML-attribute in *nodes* and *edges* declaration: the parser will know how much memory it should allocate, and will speed up the file reading. Note that count only refers to direct children, not the whole sub-graph!
- Prefer *liststring* to *string* attributes if you can. A smart parser will store the strings in one place, and just set pointers to them from the related nodes/edges.
- Identifiers may be interpreted as integers if you only use numbers. We encourage this practice, as an integer takes much less size in memory than an equivalent string. Tell the parser to optimize IDs storage by filling the optional graph XML-attribute called *idtype* with *string*, *integer* or *long*.

# 8 Web Services

The content-type for serving GEXF files via HTTP/REST is *application/gexf+xml*.

# 9 Resources

Specifications are available on GitHub at https://github.com/gephi/gexf/. Contributions and feedback are welcome.

The website https://www.gexf.net contains additional examples and links to popular libraries capable of reading and writing the format.

# 10 Changelog

Note that we used to use "draft" in certain version (e.g "1.2draft") up until 1.3 when we decided to simply use full versions numbers.

The GEXF specification history:

## Version 1.3

- Add 'kind' attribute on 'edge' to support multi-graph (i.e. parallel edges)
- The edge 'weight' is now a 'double' instead of a 'float'
- The edge 'id' is now optional
- Add 'xsd:long'as possible 'idtype' on 'graph '
- Add new attribute types 'bigdecimal', 'biginteger', 'char', 'short' and 'byte'
- Add new list attributes like 'listboolean' or 'listinteger' for each atomic type
- Dynamics

  - Add a ‘timezone‘ attribute on ‘graph ‘ to use as a timezone in case it’s omitted in the element timestamps
  - Open intervals attributes ‘startopen‘ and ‘endopen‘ are removed. Use regular inclusive ‘start‘ and ‘end‘ instead
  - Remove ‘mode‘, ‘start‘ and ‘end‘ attributes on ‘attributes ‘ as it was redundant with ‘graph ‘ attributes
  - Timestamp support
    * Add the ability to represent time with single timestamps instead of intervals. We want feature parity between the two time representations but note they can’t be mixed.
    * Add a ‘timerepresentation‘ enum in ‘graph ‘ with either ‘interval‘ (default) or ‘timestamp‘ to configure the way the time is represented
    * Add ‘timestamp‘ attribute to ‘node ‘, ‘edge ‘, ‘spell ‘ and ‘attvalue ‘ to support this new time representation
  - Alternative to spell elements
    * Add a ‘timestamps‘ attribute to ‘node ‘ and ‘edge ‘ to represent a list of timestamps without having to use spells
    * Similarly, add a ‘intervals‘ attribute to ‘node ‘ and ‘edge ‘
  - New slice mode
    * The optional ‘mode‘ attribute on ‘graph ‘ now has an additional ‘slice‘ value, in addition of ‘static‘ and ‘dynamic‘. With slice, the expectation is that the ‘graph ‘ also has either a ‘timestamp‘ or ‘start‘ / ‘end‘ intervals.
    * Add a ‘timestamp‘ attribute on ‘graph ‘ to characterise the slice this graph represent
    * Change the meaning of the ‘start‘ and ‘end‘ attributes on ‘graph ‘ to either characterise the slide instead of the time bounds, which should rather be inferred

- Viz
  - Add ‘hex‘ attribute on ‘color ‘ so it can support values like ‘FF00FF‘
  - The ‘z‘ position is no longer required
  - Dynamic attributes like ‘start‘, ‘end‘ or child elements ‘spells ‘ are no longer supported for viz attributes. To represent viz attributes over time, an alternative is to create multiple graphs each representing a slice.

## Version 1.2

- The node and edge ‘label‘ attribute are now optional
- meta should be placed before graph
- Dynamics

  - Rename the ‘timetype‘ attribute to ‘timeformat‘. This attribute is set on ‘graph ‘ to specify how time information is encoded, either like a date or like a double
  - The ‘timeformat‘ is currently either ‘float‘ or ‘date‘ and default value is ‘date‘. The ‘float‘ type is replaced by ‘double‘, and is now the default value
  - Added ‘timeformat‘ types ‘integer‘ and ‘dateTime‘. DateTime is equivalent to timestamps
  - Add open intervals (non-inclusive): ‘startopen‘ and ‘endopen‘ attributes
  - ‘slices ‘ and ‘slice ‘ are renamed ‘spells ‘ and ‘spell ‘ respectively because slices are a different concept as remarked

- Viz
  - Add viz attributes support for dynamics
  - Add the alpha channel to RGB. Colors are now encoded in RGBA. It is a float from 0.0 (invisible) to 1.0 (fully visible). If omitted, the default alpha-value is 1.0 (no transparency).

## Version 1.1

Modules are stabilized and new ones appear: hierarchy and phylogeny.

## Version 1.0

First specification. Basic topology, associated data and dynamics attempt constitute the core, plus a visualization extension.