

# LinksPlatform's Platform.Communication Class Library

## 1.1 ./csharp/Platform.Communication/Protocol/Gexf/Edge.cs

```
1 using System.Globalization;
2 using System.Runtime.CompilerServices;
3 using System.Xml;
4 using System.Xml.Serialization;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Communication.Protocol.Gexf
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the edge.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public class Edge
17     {
18         /// <summary>
19         /// <para>
20         /// The element name.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         public static readonly string ElementName = "edge";
25         /// <summary>
26         /// <para>
27         /// The id attribute name.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         public const string IdAttributeName = "id";
32         /// <summary>
33         /// <para>
34         /// The source attribute name.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         public const string SourceAttributeName = "source";
39         /// <summary>
40         /// <para>
41         /// The target attribute name.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         public const string TargetAttributeName = "target";
46         /// <summary>
47         /// <para>
48         /// The label attribute name.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         public const string LabelAttributeName = "label";
53
54         /// <summary>
55         /// <para>
56         /// Gets or sets the id value.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         [XmlAttribute(AttributeName = IdAttributeName)]
61         public long Id
62         {
63             [MethodImpl(MethodImplOptions.AggressiveInlining)]
64             get;
65             [MethodImpl(MethodImplOptions.AggressiveInlining)]
66             set;
67         }
68
69         /// <summary>
70         /// <para>
71         /// Gets or sets the source value.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         [XmlAttribute(AttributeName = SourceAttributeName)]
76         public long Source
77     {
```

```

78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         get;
80         [MethodImpl(MethodImplOptions.AggressiveInlining)]
81         set;
82     }
83
84     /// <summary>
85     /// <para>
86     /// Gets or sets the target value.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     [XmlAttribute(AttributeName = TargetAttributeName)]
91     public long Target
92     {
93         [MethodImpl(MethodImplOptions.AggressiveInlining)]
94         get;
95         [MethodImpl(MethodImplOptions.AggressiveInlining)]
96         set;
97     }
98
99     /// <summary>
100    /// <para>
101    /// Gets or sets the label value.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    [XmlAttribute(AttributeName = LabelAttributeName)]
106    public string Label
107    {
108        [MethodImpl(MethodImplOptions.AggressiveInlining)]
109        get;
110        [MethodImpl(MethodImplOptions.AggressiveInlining)]
111        set;
112    }
113
114    /// <summary>
115    /// <para>
116    /// Writes the xml using the specified writer.
117    /// </para>
118    /// <para></para>
119    /// </summary>
120    /// <param name="writer">
121    /// <para>The writer.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Source, Target, Label);
126
127    /// <summary>
128    /// <para>
129    /// Writes the xml using the specified writer.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="writer">
134    /// <para>The writer.</para>
135    /// <para></para>
136    /// </param>
137    /// <param name="id">
138    /// <para>The id.</para>
139    /// <para></para>
140    /// </param>
141    /// <param name="sourceNodeId">
142    /// <para>The source node id.</para>
143    /// <para></para>
144    /// </param>
145    /// <param name="targetNodeId">
146    /// <para>The target node id.</para>
147    /// <para></para>
148    /// </param>
149    [MethodImpl(MethodImplOptions.AggressiveInlining)]
150    public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
    ↪ targetNodeId) => WriteXml(writer, id, sourceNodeId, targetNodeId, null);
151
152    /// <summary>
153    /// <para>
154    /// Writes the xml using the specified writer.
155    /// </para>

```

```

156     /// <para></para>
157     /// </summary>
158     /// <param name="writer">
159     /// <para>The writer.</para>
160     /// <para></para>
161     /// </param>
162     /// <param name="id">
163     /// <para>The id.</para>
164     /// <para></para>
165     /// </param>
166     /// <param name="sourceNodeId">
167     /// <para>The source node id.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="targetNodeId">
171     /// <para>The target node id.</para>
172     /// <para></para>
173     /// </param>
174     /// <param name="label">
175     /// <para>The label.</para>
176     /// <para></para>
177     /// </param>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
180         ↪ targetNodeId, string label)
181     {
182         // <edge id="0" source="0" target="0" label="..." />
183         writer.WriteStartElement(ElementName);
184         writer.WriteAttributeString(IdAttributeName,
185             ↪ id.ToString(CultureInfo.InvariantCulture));
186         writer.WriteAttributeString(SourceAttributeName,
187             ↪ sourceNodeId.ToString(CultureInfo.InvariantCulture));
188         writer.WriteAttributeString(TargetAttributeName,
189             ↪ targetNodeId.ToString(CultureInfo.InvariantCulture));
190         if (!string.IsNullOrEmpty(label))
191         {
192             writer.WriteAttributeString(LabelAttributeName, label);
193         }
194         writer.WriteEndElement();
195     }
196 }

```

## 1.2 ./csharp/Platform.Communication/Protocol/Gexf/Gexf.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using System.Xml;
4 using System.Xml.Serialization;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Communication.Protocol.Gexf
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the gexf.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     [XmlRoot(ElementName = ElementName, Namespace = Namespace)]
17     public class Gexf
18     {
19         /// <summary>
20         /// <para>
21         /// The element name.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public const string ElementName = "gexf";
26         /// <summary>
27         /// <para>
28         /// The namespace.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         public const string Namespace = "http://www.gexf.net/1.2draft";
33         /// <summary>
34         /// <para>

```

```

35     /// The version attribute name.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     public const string VersionAttributeName = "version";
40     /// <summary>
41     /// <para>
42     /// The graph element name.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     public const string GraphElementName = "graph";
47     /// <summary>
48     /// <para>
49     /// The current version.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     public static readonly string CurrentVersion = "1.2";
54
55     /// <summary>
56     /// <para>
57     /// Gets or sets the version value.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     [XmlAttribute(AttributeName = VersionAttributeName)]
62     public string Version
63     {
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         get;
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         set;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Gets or sets the graph value.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     [XmlElement(ElementName = GraphElementName)]
77     public Graph Graph
78     {
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         get;
81         [MethodImpl(MethodImplOptions.AggressiveInlining)]
82         set;
83     }
84
85     /// <summary>
86     /// <para>
87     /// Initializes a new <see cref="Gexf"/> instance.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     public Gexf() => (Version, Graph) = (CurrentVersion, new Graph());
93
94     /// <summary>
95     /// <para>
96     /// Writes the xml using the specified writer.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="writer">
101    /// <para>The writer.</para>
102    /// <para></para>
103    /// </param>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
106        ↪ Version);
107
108    /// <summary>
109    /// <para>
110    /// Writes the xml using the specified writer.
111    /// </para>
112    /// <para></para>
113    /// </summary>

```

```

113     /// <param name="writer">
114     /// <para>The writer.</para>
115     /// <para></para>
116     /// </param>
117     /// <param name="writeGraph">
118     /// <para>The write graph.</para>
119     /// <para></para>
120     /// </param>
121     [MethodImpl(MethodImplOptions.AggressiveInlining)]
122     public static void WriteXml(XmlWriter writer, Action writeGraph) => WriteXml(writer,
    ↪ writeGraph, CurrentVersion);
123
124     /// <summary>
125     /// <para>
126     /// Writes the xml using the specified writer.
127     /// </para>
128     /// <para></para>
129     /// </summary>
130     /// <param name="writer">
131     /// <para>The writer.</para>
132     /// <para></para>
133     /// </param>
134     /// <param name="writeGraph">
135     /// <para>The write graph.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="version">
139     /// <para>The version.</para>
140     /// <para></para>
141     /// </param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     public static void WriteXml(XmlWriter writer, Action writeGraph, string version)
144     {
145         writer.WriteStartDocument();
146         writer.WriteStartElement(ElementName, Namespace);
147         writer.WriteAttributeString(VersionAttributeName, version);
148         writeGraph();
149         writer.WriteEndElement();
150         writer.WriteEndDocument();
151     }
152
153     /// <summary>
154     /// <para>
155     /// Writes the xml using the specified writer.
156     /// </para>
157     /// <para></para>
158     /// </summary>
159     /// <param name="writer">
160     /// <para>The writer.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="writeNodes">
164     /// <para>The write nodes.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="writeEdges">
168     /// <para>The write edges.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
    ↪ WriteXml(writer, writeNodes, writeEdges, CurrentVersion, GraphMode.Static,
    ↪ GraphDefaultEdgeType.Directed);
173
174     /// <summary>
175     /// <para>
176     /// Writes the xml using the specified writer.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <param name="writer">
181     /// <para>The writer.</para>
182     /// <para></para>
183     /// </param>
184     /// <param name="writeNodes">
185     /// <para>The write nodes.</para>
186     /// <para></para>
187     /// </param>

```

```

188     /// <param name="writeEdges">
189     /// <para>The write edges.</para>
190     /// <para></para>
191     /// </param>
192     /// <param name="version">
193     /// <para>The version.</para>
194     /// <para></para>
195     /// </param>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
198     ↪     string version) => WriteXml(writer, writeNodes, writeEdges, version,
199     ↪     GraphMode.Static, GraphDefaultEdgeType.Directed);
200
201     /// <summary>
202     /// <para>
203     /// Writes the xml using the specified writer.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="writer">
208     /// <para>The writer.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="writeNodes">
212     /// <para>The write nodes.</para>
213     /// <para></para>
214     /// </param>
215     /// <param name="writeEdges">
216     /// <para>The write edges.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="version">
220     /// <para>The version.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="mode">
224     /// <para>The mode.</para>
225     /// <para></para>
226     /// </param>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
229     ↪     string version, GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, version,
230     ↪     mode, GraphDefaultEdgeType.Directed);
231
232     /// <summary>
233     /// <para>
234     /// Writes the xml using the specified writer.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="writer">
239     /// <para>The writer.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="writeNodes">
243     /// <para>The write nodes.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="writeEdges">
247     /// <para>The write edges.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="version">
251     /// <para>The version.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="mode">
255     /// <para>The mode.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="defaultEdgeType">
259     /// <para>The default edge type.</para>
260     /// <para></para>
261     /// </param>
262     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

259         public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↳ string version, GraphMode mode, GraphDefaultEdgeType defaultEdgeType) =>
        ↳ WriteXml(writer, () => Graph.WriteXml(writer, writeNodes, writeEdges, mode,
        ↳ defaultEdgeType), version);
260     }
261 }

```

### 1.3 ./csharp/Platform.Communication/Protocol/Gexf/Graph.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using System.Xml;
5  using System.Xml.Serialization;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Communication.Protocol.Gexf
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the graph.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public class Graph
18     {
19         /// <summary>
20         /// <para>
21         /// The element name.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public static readonly string ElementName = "graph";
26         /// <summary>
27         /// <para>
28         /// The mode attribute name.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         public const string ModeAttributeName = "mode";
33         /// <summary>
34         /// <para>
35         /// The default edge type attribute name.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         public const string DefaultEdgeTypeAttributeName = "defaultedgetype";
40         /// <summary>
41         /// <para>
42         /// The nodes element name.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         public const string NodesElementName = "nodes";
47         /// <summary>
48         /// <para>
49         /// The node element name.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         public const string NodeElementName = "node";
54         /// <summary>
55         /// <para>
56         /// The edges element name.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         public const string EdgesElementName = "edges";
61         /// <summary>
62         /// <para>
63         /// The edge element name.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         public const string EdgeElementName = "edge";
68
69         /// <summary>
70         /// <para>
71         /// Gets or sets the mode value.

```

```

72     /// </para>
73     /// <para></para>
74     /// </summary>
75     [XmlAttribute(AttributeName = ModeAttributeName)]
76     public GraphMode Mode
77     {
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         get;
80         [MethodImpl(MethodImplOptions.AggressiveInlining)]
81         set;
82     }
83
84     /// <summary>
85     /// <para>
86     /// Gets or sets the default edge type value.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     [XmlAttribute(AttributeName = DefaultEdgeTypeAttributeName)]
91     public GraphDefaultEdgeType DefaultEdgeType
92     {
93         [MethodImpl(MethodImplOptions.AggressiveInlining)]
94         get;
95         [MethodImpl(MethodImplOptions.AggressiveInlining)]
96         set;
97     }
98
99     /// <summary>
100    /// <para>
101    /// Gets or sets the nodes value.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    [XmlArray(ElementName = NodesElementName)]
106    [XmlArrayItem(ElementName = NodeElementName)]
107    public List<Node> Nodes
108    {
109        [MethodImpl(MethodImplOptions.AggressiveInlining)]
110        get;
111        [MethodImpl(MethodImplOptions.AggressiveInlining)]
112        set;
113    }
114
115    /// <summary>
116    /// <para>
117    /// Gets or sets the edges value.
118    /// </para>
119    /// <para></para>
120    /// </summary>
121    [XmlArray(ElementName = EdgesElementName)]
122    [XmlArrayItem(ElementName = EdgeElementName)]
123    public List<Edge> Edges
124    {
125        [MethodImpl(MethodImplOptions.AggressiveInlining)]
126        get;
127        [MethodImpl(MethodImplOptions.AggressiveInlining)]
128        set;
129    }
130
131    /// <summary>
132    /// <para>
133    /// Initializes a new <see cref="Graph"/> instance.
134    /// </para>
135    /// <para></para>
136    /// </summary>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    public Graph() => (Nodes, Edges) = (new List<Node>(), new List<Edge>());
139
140    /// <summary>
141    /// <para>
142    /// Writes the xml using the specified writer.
143    /// </para>
144    /// <para></para>
145    /// </summary>
146    /// <param name="writer">
147    /// <para>The writer.</para>
148    /// <para></para>
149    /// </param>
150    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

151 public void WriteXml(XmlWriter writer) => WriteXml(writer, () => WriteNodes(writer), ()
    ↳ => WriteEdges(writer), Mode, DefaultEdgeType);
152 [MethodImpl(MethodImplOptions.AggressiveInlining)]
153 private void WriteEdges(XmlWriter writer)
154 {
155     for (var i = 0; i < Edges.Count; i++)
156     {
157         Edges[i].WriteXml(writer);
158     }
159 }
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 private void WriteNodes(XmlWriter writer)
162 {
163     for (var i = 0; i < Nodes.Count; i++)
164     {
165         Nodes[i].WriteXml(writer);
166     }
167 }
168
169 /// <summary>
170 /// <para>
171 /// Writes the xml using the specified writer.
172 /// </para>
173 /// <para></para>
174 /// </summary>
175 /// <param name="writer">
176 /// <para>The writer.</para>
177 /// <para></para>
178 /// </param>
179 /// <param name="writeNodes">
180 /// <para>The write nodes.</para>
181 /// <para></para>
182 /// </param>
183 /// <param name="writeEdges">
184 /// <para>The write edges.</para>
185 /// <para></para>
186 /// </param>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
    ↳ WriteXml(writer, writeNodes, writeEdges, GraphMode.Static,
    ↳ GraphDefaultEdgeType.Directed);
189
190 /// <summary>
191 /// <para>
192 /// Writes the xml using the specified writer.
193 /// </para>
194 /// <para></para>
195 /// </summary>
196 /// <param name="writer">
197 /// <para>The writer.</para>
198 /// <para></para>
199 /// </param>
200 /// <param name="writeNodes">
201 /// <para>The write nodes.</para>
202 /// <para></para>
203 /// </param>
204 /// <param name="writeEdges">
205 /// <para>The write edges.</para>
206 /// <para></para>
207 /// </param>
208 /// <param name="mode">
209 /// <para>The mode.</para>
210 /// <para></para>
211 /// </param>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
    ↳ GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, mode,
    ↳ GraphDefaultEdgeType.Directed);
214
215 /// <summary>
216 /// <para>
217 /// Writes the xml using the specified writer.
218 /// </para>
219 /// <para></para>
220 /// </summary>
221 /// <param name="writer">
222 /// <para>The writer.</para>

```

```

223     /// <para></para>
224     /// </param>
225     /// <param name="writeNodes">
226     /// <para>The write nodes.</para>
227     /// <para></para>
228     /// </param>
229     /// <param name="writeEdges">
230     /// <para>The write edges.</para>
231     /// <para></para>
232     /// </param>
233     /// <param name="mode">
234     /// <para>The mode.</para>
235     /// <para></para>
236     /// </param>
237     /// <param name="defaultEdgeType">
238     /// <para>The default edge type.</para>
239     /// <para></para>
240     /// </param>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
    ↪ GraphMode mode, GraphDefaultEdgeType defaultEdgeType)
243     {
244         writer.WriteStartElement(ElementName);
245         writer.WriteAttributeString(ModeAttributeName, mode.ToString().ToLower());
246         writer.WriteAttributeString(DefaultEdgeTypeAttributeName,
    ↪ defaultEdgeType.ToString().ToLower());
247         writer.WriteStartElement(NodesElementName);
248         writeNodes();
249         writer.WriteEndElement();
250         writer.WriteStartElement(EdgesElementName);
251         writeEdges();
252         writer.WriteEndElement();
253         writer.WriteEndElement();
254     }
255 }
256 }

```

#### 1.4 ./csharp/Platform.Communication/Protocol/Gexf/GraphDefaultEdgeType.cs

```

1 using System.Xml.Serialization;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Communication.Protocol.Gexf
6 {
7     /// <summary>
8     /// <para>
9     /// The graph default edge type enum.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public enum GraphDefaultEdgeType
14    {
15        /// <summary>
16        /// <para>
17        /// The directed graph default edge type.
18        /// </para>
19        /// <para></para>
20        /// </summary>
21        [XmlEnum(Name = "directed")]
22        Directed
23    }
24 }

```

#### 1.5 ./csharp/Platform.Communication/Protocol/Gexf/GraphMode.cs

```

1 using System.Xml.Serialization;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Communication.Protocol.Gexf
6 {
7     /// <summary>
8     /// <para>
9     /// The graph mode enum.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public enum GraphMode
14    {

```

```

15     /// <summary>
16     /// <para>
17     /// The static graph mode.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     [XmlEnum(Name = "static")]
22     Static,
23
24     /// <summary>
25     /// <para>
26     /// The dynamic graph mode.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     [XmlEnum(Name = "dynamic")]
31     Dynamic
32 }
33 }

```

## 1.6 ./csharp/Platform.Communication/Protocol/Gexf/Node.cs

```

1 using System.Globalization;
2 using System.Runtime.CompilerServices;
3 using System.Xml;
4 using System.Xml.Serialization;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Communication.Protocol.Gexf
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the node.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public class Node
17     {
18         /// <summary>
19         /// <para>
20         /// The element name.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         public static readonly string ElementName = "node";
25         /// <summary>
26         /// <para>
27         /// The id attribute name.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         public const string IdAttributeName = "id";
32         /// <summary>
33         /// <para>
34         /// The label attribute name.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         public const string LabelAttributeName = "label";
39
40         /// <summary>
41         /// <para>
42         /// Gets or sets the id value.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         [XmlAttribute(AttributeName = IdAttributeName)]
47         public long Id
48         {
49             [MethodImpl(MethodImplOptions.AggressiveInlining)]
50             get;
51             [MethodImpl(MethodImplOptions.AggressiveInlining)]
52             set;
53         }
54
55         /// <summary>
56         /// <para>
57         /// Gets or sets the label value.
58         /// </para>

```

```

59     /// <para></para>
60     /// </summary>
61     [XmlAttribute(AttributeName = LabelAttributeName)]
62     public string Label
63     {
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         get;
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         set;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Writes the xml using the specified writer.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="writer">
77     /// <para>The writer.</para>
78     /// <para></para>
79     /// </param>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Label);
82
83     /// <summary>
84     /// <para>
85     /// Writes the xml using the specified writer.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="writer">
90     /// <para>The writer.</para>
91     /// <para></para>
92     /// </param>
93     /// <param name="id">
94     /// <para>The id.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="label">
98     /// <para>The label.</para>
99     /// <para></para>
100    /// </param>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    public static void WriteXml(XmlWriter writer, long id, string label)
103    {
104        // <node id="0" label="..." />
105        writer.WriteStartElement(ElementName);
106        writer.WriteAttributeString(IdAttributeName,
107            ↪ id.ToString(CultureInfo.InvariantCulture));
107        writer.WriteAttributeString(LabelAttributeName, label);
108        writer.WriteEndElement();
109    }
110 }
111 }

```

## 1.7 ./csharp/Platform.Communication/Protocol/Udp/UdpClientExtensions.cs

```

1  using System.Net;
2  using System.Net.Sockets;
3  using System.Runtime.CompilerServices;
4  using System.Text;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Communication.Protocol.Udp
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the udp client extensions.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public static class UdpClientExtensions
17     {
18         /// <summary>
19         /// <para>
20         /// The utf.
21         /// </para>
22         /// <para></para>
23         /// </summary>

```

```

24     public static readonly Encoding DefaultEncoding = Encoding.UTF8;
25
26     /// <summary>
27     /// <para>
28     /// Sends the string using the specified udp.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     /// <param name="udp">
33     /// <para>The udp.</para>
34     /// <para></para>
35     /// </param>
36     /// <param name="ipEndPoint">
37     /// <para>The ip end point.</para>
38     /// <para></para>
39     /// </param>
40     /// <param name="message">
41     /// <para>The message.</para>
42     /// <para></para>
43     /// </param>
44     /// <returns>
45     /// <para>The int</para>
46     /// <para></para>
47     /// </returns>
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     public static int SendString(this UdpClient udp, IPEndPoint ipEndPoint, string message)
50     {
51         var bytes = DefaultEncoding.GetBytes(message);
52         return udp.Send(bytes, bytes.Length, ipEndPoint);
53     }
54
55     /// <summary>
56     /// <para>
57     /// Receives the string using the specified udp.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="udp">
62     /// <para>The udp.</para>
63     /// <para></para>
64     /// </param>
65     /// <returns>
66     /// <para>The string</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     public static string ReceiveString(this UdpClient udp)
71     {
72         IPEndPoint remoteEndPoint = default;
73         return DefaultEncoding.GetString(udp.Receive(ref remoteEndPoint));
74     }
75 }
76 }

```

## 1.8 ./csharp/Platform.Communication/Protocol/Udp/UdpReceiver.cs

```

1  using System;
2  using System.Net.Sockets;
3  using System.Runtime.CompilerServices;
4  using System.Threading;
5  using Platform.Disposables;
6  using Platform.Exceptions;
7  using Platform.Threading;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Communication.Protocol.Udp
12 {
13     /// <summary>
14     /// <para>
15     /// The message handler callback.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public delegate void MessageHandlerCallback(string message);
20
21     /// <summary>
22     /// <para>Represents the receiver of messages transfered via UDP protocol.</para>
23     /// <para>Представляет получателя сообщений по протоколу UDP.</para>
24     /// </summary>

```

```

25 public class UdpReceiver : DisposableBase //-V3073
26 {
27     private const int DefaultPort = 15000;
28     private bool _receiverRunning;
29     private Thread _thread;
30     private readonly UdpClient _udp;
31     private readonly MessageHandlerCallback _messageHandler;
32
33     /// <summary>
34     /// <para>
35     /// Gets the available value.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     public bool Available
40     {
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         get => _udp.Available > 0;
43     }
44
45     /// <summary>
46     /// <para>
47     /// Initializes a new <see cref="UdpReceiver"/> instance.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     /// <param name="listenPort">
52     /// <para>A listen port.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="autoStart">
56     /// <para>A auto start.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="messageHandler">
60     /// <para>A message handler.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public UdpReceiver(int listenPort, bool autoStart, MessageHandlerCallback messageHandler)
65     {
66         _udp = new UdpClient(listenPort);
67         _messageHandler = messageHandler;
68         if (autoStart)
69         {
70             Start();
71         }
72     }
73
74     /// <summary>
75     /// <para>
76     /// Initializes a new <see cref="UdpReceiver"/> instance.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="listenPort">
81     /// <para>A listen port.</para>
82     /// <para></para>
83     /// </param>
84     /// <param name="messageHandler">
85     /// <para>A message handler.</para>
86     /// <para></para>
87     /// </param>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public UdpReceiver(int listenPort, MessageHandlerCallback messageHandler) :
90         ↪ this(listenPort, true, messageHandler) { }
91
92     /// <summary>
93     /// <para>
94     /// Initializes a new <see cref="UdpReceiver"/> instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="messageHandler">
99     /// <para>A message handler.</para>
100     /// <para></para>
101     /// </param>
102     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

102 public UdpReceiver(MessageHandlerCallback messageHandler) : this(DefaultPort, true,
103     ↳ messageHandler) { }
104
105 /// <summary>
106 /// <para>
107 /// Initializes a new <see cref="UdpReceiver"/> instance.
108 /// </para>
109 /// </summary>
110 [MethodImpl(MethodImplOptions.AggressiveInlining)]
111 public UdpReceiver() : this(DefaultPort, true, message => { }) { }
112
113 /// <summary>
114 /// <para>
115 /// Starts this instance.
116 /// </para>
117 /// </summary>
118 [MethodImpl(MethodImplOptions.AggressiveInlining)]
119 public void Start()
120 {
121     if (!_receiverRunning && _thread == null)
122     {
123         _receiverRunning = true;
124         _thread = new Thread(Receiver);
125         _thread.Start();
126     }
127 }
128
129 /// <summary>
130 /// <para>
131 /// Stops this instance.
132 /// </para>
133 /// </summary>
134 [MethodImpl(MethodImplOptions.AggressiveInlining)]
135 public void Stop()
136 {
137     if (_receiverRunning && _thread != null)
138     {
139         _receiverRunning = false;
140         _thread.Join();
141         _thread = null;
142     }
143 }
144
145 /// <summary>
146 /// <para>
147 /// Receives this instance.
148 /// </para>
149 /// </summary>
150 /// <returns>
151 /// <para>The string</para>
152 /// </returns>
153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 public string Receive() => _udp.ReceiveString();
155
156 /// <summary>
157 /// <para>
158 /// Receives the and handle.
159 /// </para>
160 /// </summary>
161 [MethodImpl(MethodImplOptions.AggressiveInlining)]
162 public void ReceiveAndHandle() => _messageHandler(Receive());
163 [MethodImpl(MethodImplOptions.AggressiveInlining)]
164 private void Receiver()
165 {
166     while (_receiverRunning)
167     {
168         try
169         {
170             if (Available)
171             {
172                 ReceiveAndHandle();
173             }
174         }
175     }
176 }
177
178

```

```

179         else
180         {
181             ThreadHelpers.Sleep();
182         }
183     }
184     catch (Exception exception)
185     {
186         exception.Ignore();
187     }
188 }
189 }
190
191 /// <summary>
192 /// <para>
193 /// Disposes the manual.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="manual">
198 /// <para>The manual.</para>
199 /// <para></para>
200 /// </param>
201 /// <param name="wasDisposed">
202 /// <para>The was disposed.</para>
203 /// <para></para>
204 /// </param>
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 protected override void Dispose(bool manual, bool wasDisposed)
207 {
208     if (!wasDisposed)
209     {
210         Stop();
211         _udp.DisposeIfPossible();
212     }
213 }
214 }
215 }

```

## 1.9 ./csharp/Platform.Communication/Protocol/Udp/UdpSender.cs

```

1 using System.Net;
2 using System.Net.Sockets;
3 using System.Runtime.CompilerServices;
4 using Platform.Disposables;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Communication.Protocol.Udp
9 {
10     /// <summary>
11     /// <para>Represents the sender of messages transferred via UDP protocol.</para>
12     /// <para>Представляет отправителя сообщений по протоколу UDP.</para>
13     /// </summary>
14     public class UdpSender : DisposableBase //-V3073
15     {
16         private readonly UdpClient _udp;
17         private readonly IPEndPoint _ipendpoint;
18
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="UdpSender"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="ipendpoint">
26         /// <para>A ipendpoint.</para>
27         /// <para></para>
28         /// </param>
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public UdpSender(IPEndPoint ipendpoint) => (_udp, _ipendpoint) = (new UdpClient(),
31             ↪ ipendpoint);
32
33         /// <summary>
34         /// <para>
35         /// Initializes a new <see cref="UdpSender"/> instance.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="address">
40         /// <para>A address.</para>

```



```

40     /// <para></para>
41     /// </param>
42     /// <param name="port">
43     /// <para>A port.</para>
44     /// <para></para>
45     /// </param>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public UdpSender(IPAddress address, int port) : this(new IPEndPoint(address, port)) { }
48
49     /// <summary>
50     /// <para>
51     /// Initializes a new <see cref="UdpSender"/> instance.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     /// <param name="hostname">
56     /// <para>A hostname.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="port">
60     /// <para>A port.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public UdpSender(string hostname, int port) : this(IPAddress.Parse(hostname), port) { }
65
66     /// <summary>
67     /// <para>
68     /// Initializes a new <see cref="UdpSender"/> instance.
69     /// </para>
70     /// <para></para>
71     /// </summary>
72     /// <param name="port">
73     /// <para>A port.</para>
74     /// <para></para>
75     /// </param>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public UdpSender(int port) : this(IPAddress.Loopback, port) { }
78
79     /// <summary>
80     /// <para>
81     /// Sends the message.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <param name="message">
86     /// <para>The message.</para>
87     /// <para></para>
88     /// </param>
89     /// <returns>
90     /// <para>The int</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public int Send(string message) => _udp.SendString(_ipendpoint, message);
95
96     /// <summary>
97     /// <para>
98     /// Disposes the manual.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="manual">
103    /// <para>The manual.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="wasDisposed">
107    /// <para>The was disposed.</para>
108    /// <para></para>
109    /// </param>
110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
111    protected override void Dispose(bool manual, bool wasDisposed)
112    {
113        if (!wasDisposed)
114        {
115            _udp.DisposeIfPossible();
116        }
117    }

```

```
118 }
119 }
```

## 1.10 ./csharp/Platform.Communication/Protocol/Xml/Serializer.cs

```
1 using System.IO;
2 using System.Runtime.CompilerServices;
3 using System.Text;
4 using System.Xml.Serialization;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Communication.Protocol.Xml
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the serializer.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public static class Serializer<T>
17     {
18         /// <summary>
19         /// <para>
20         /// The .
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         public static readonly XmlSerializer Instance = new XmlSerializer(typeof(T));
25
26         /// <summary>
27         /// <para>
28         /// Creates the file using the specified path.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="path">
33         /// <para>The path.</para>
34         /// <para></para>
35         /// </param>
36         /// <returns>
37         /// <para>The</para>
38         /// <para></para>
39         /// </returns>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public static T FromFile(string path)
42         {
43             using var stream = File.OpenRead(path);
44             return (T)Instance.Deserialize(stream);
45         }
46
47         /// <summary>
48         /// <para>
49         /// Creates the string using the specified xml.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="xml">
54         /// <para>The xml.</para>
55         /// <para></para>
56         /// </param>
57         /// <returns>
58         /// <para>The</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         public static T FromString(string xml)
63         {
64             using var reader = new StringReader(xml);
65             return (T)Instance.Deserialize(reader);
66         }
67
68         /// <summary>
69         /// <para>
70         /// Returns the file using the specified object.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="@object">
75         /// <para>The object.</para>
```

```

76     /// <para></para>
77     /// </param>
78     /// <param name="path">
79     /// <para>The path.</para>
80     /// <para></para>
81     /// </param>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public static void ToFile(T @object, string path)
84     {
85         using var stream = File.OpenWrite(path);
86         Instance.Serialize(stream, @object);
87     }
88
89     /// <summary>
90     /// <para>
91     /// Returns the string using the specified object.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="@object">
96     /// <para>The object.</para>
97     /// <para></para>
98     /// </param>
99     /// <returns>
100    /// <para>The string</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    public static string ToString(T @object)
105    {
106        var sb = new StringBuilder();
107        using (var writer = new StringWriter(sb))
108        {
109            Instance.Serialize(writer, @object);
110        }
111        return sb.ToString();
112    }
113 }
114 }

```

#### 1.11 ./csharp/Platform.Communication.Tests/SerializerTests.cs

```

1  using System;
2  using System.IO;
3  using Xunit;
4  using Platform.Singletons;
5  using Platform.Communication.Protocol.Xml;
6
7  namespace Platform.Communication.Tests
8  {
9      public static class SerializerTests
10     {
11         [Fact]
12         public static void SerializeToFileTest()
13         {
14             var tempFilename = Path.GetTempFileName();
15             Serializer<object>.ToFile(Default<object>.Instance, tempFilename);
16             Assert.Equal(File.ReadAllText(tempFilename), $"<?xml
17                 ↪ version=\"1.0\"?>{Environment.NewLine}<anyType
18                 ↪ xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
19                 ↪ xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" />");
20             File.Delete(tempFilename);
21         }
22
23         [Fact]
24         public static void SerializeAsXmlStringTest()
25         {
26             var serializedObject = Serializer<object>.ToString(Default<object>.Instance);
27             Assert.Equal(serializedObject, $"<?xml version=\"1.0\"
28                 ↪ encoding=\"utf-16\"?>{Environment.NewLine}<anyType
29                 ↪ xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
30                 ↪ xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" />");
31         }
32     }
33 }

```

#### 1.12 ./csharp/Platform.Communication.Tests/UdpReceiverTests.cs

```

1  using Xunit;
2  using Platform.Communication.Protocol.Udp;

```

```
3
4 namespace Platform.Communication.Tests
5 {
6     public static class UdpReceiverTests
7     {
8         [Fact]
9         public static void DisposalTest()
10        {
11            using var receiver = new UdpReceiver();
12        }
13    }
14 }
```

## Index

- ./csharp/Platform.Communication.Tests/SerializerTests.cs, 19
- ./csharp/Platform.Communication.Tests/UdpReceiverTests.cs, 19
- ./csharp/Platform.Communication/Protocol/Gexf/Edge.cs, 1
- ./csharp/Platform.Communication/Protocol/Gexf/Gexf.cs, 3
- ./csharp/Platform.Communication/Protocol/Gexf/Graph.cs, 7
- ./csharp/Platform.Communication/Protocol/Gexf/GraphDefaultEdgeType.cs, 10
- ./csharp/Platform.Communication/Protocol/Gexf/GraphMode.cs, 10
- ./csharp/Platform.Communication/Protocol/Gexf/Node.cs, 11
- ./csharp/Platform.Communication/Protocol/Udp/UdpClientExtensions.cs, 12
- ./csharp/Platform.Communication/Protocol/Udp/UdpReceiver.cs, 13
- ./csharp/Platform.Communication/Protocol/Udp/UdpSender.cs, 16
- ./csharp/Platform.Communication/Protocol/Xml/Serializer.cs, 18