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

```
using System.Globalization;
using System.Runtime.CompilerServices;
using System.Xml;
using System.Xml.Serialization;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Gexf
{
    /// <summary>
    /// <para>
    /// Represents the edge.
    /// </para>
    /// <para></para>
    /// </summary>
    public class Edge
    {
        /// <summary>
        /// <para>
        /// The element name.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly string ElementName = "edge";
        /// <summary>
        /// <para>
        /// The id attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string IdAttributeName = "id";
        /// <summary>
        /// <para>
        /// The source attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string SourceAttributeName = "source";
        /// <summary>
        /// <para>
        /// The target attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string TargetAttributeName = "target";
        /// <summary>
        /// <para>
        /// The label attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string LabelAttributeName = "label";

        /// <summary>
        /// <para>
        /// Gets or sets the id value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = IdAttributeName)]
        public long Id
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Gets or sets the source value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = SourceAttributeName)]
        public long Source
        {
```

```
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Gets or sets the target value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = TargetAttributeName)]
        public long Target
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Gets or sets the label value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = LabelAttributeName)]
        public string Label
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Writes the xml using the specified writer.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="writer">
        /// <para>The writer.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Source, Target, Label);

        /// <summary>
        /// <para>
        /// Writes the xml using the specified writer.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="writer">
        /// <para>The writer.</para>
        /// <para></para>
        /// </param>
        /// <param name="id">
        /// <para>The id.</para>
        /// <para></para>
        /// </param>
        /// <param name="sourceNodeId">
        /// <para>The source node id.</para>
        /// <para></para>
        /// </param>
        /// <param name="targetNodeId">
        /// <para>The target node id.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
        ↪  targetNodeId) => WriteXml(writer, id, sourceNodeId, targetNodeId, null);

        /// <summary>
        /// <para>
        /// Writes the xml using the specified writer.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="writer">
        /// <para>The writer.</para>
        /// <para></para>
        /// </param>
        /// <param name="id">
        /// <para>The id.</para>
        /// <para></para>
        /// </param>
        /// <param name="sourceNodeId">
        /// <para>The source node id.</para>
        /// <para></para>
        /// </param>
        /// <param name="targetNodeId">
        /// <para>The target node id.</para>
        /// <para></para>
        /// </param>
        /// <param name="label">
        /// <para>The label.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
        ↪  targetNodeId, string label)
        {
            // <edge id="0" source="0" target="0" label="..." />
            writer.WriteStartElement(ElementName);
            writer.WriteAttributeString(IdAttributeName,
            ↪  id.ToString(CultureInfo.InvariantCulture));
            writer.WriteAttributeString(SourceAttributeName,
            ↪  sourceNodeId.ToString(CultureInfo.InvariantCulture));
            writer.WriteAttributeString(TargetAttributeName,
            ↪  targetNodeId.ToString(CultureInfo.InvariantCulture));
            if (!string.IsNullOrWhiteSpace(label))
            {
                writer.WriteAttributeString(LabelAttributeName, label);
            }
            writer.WriteEndElement();
        }
    }
}
```

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

```
using System;
using System.Runtime.CompilerServices;
using System.Xml;
using System.Xml.Serialization;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Gexf
{
    /// <summary>
    /// <para>
    /// Represents the gexf.
    /// </para>
    /// <para></para>
    /// </summary>
    [XmlRoot(ElementName = ElementName, Namespace = Namespace)]
    public class Gexf
    {
        /// <summary>
        /// <para>
        /// The element name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string ElementName = "gexf";
        /// <summary>
        /// <para>
        /// The namespace.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string Namespace = "http://www.gexf.net/1.2draft";
        /// <summary>
        /// <para>
```

```
        /// The version attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string VersionAttributeName = "version";
        /// <summary>
        /// <para>
        /// The graph element name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string GraphElementName = "graph";
        /// <summary>
        /// <para>
        /// The current version.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly string CurrentVersion = "1.2";

        /// <summary>
        /// <para>
        /// Gets or sets the version value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = VersionAttributeName)]
        public string Version
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Gets or sets the graph value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlElement(ElementName = GraphElementName)]
        public Graph Graph
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Gexf"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Gexf() => (Version, Graph) = (CurrentVersion, new Graph());

        /// <summary>
        /// <para>
        /// Writes the xml using the specified writer.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="writer">
        /// <para>The writer.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
        ↪  Version);

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

```
        /// <summary>
        /// <para>
        /// The static graph mode.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlEnum(Name = "static")]
        Static,

        /// <summary>
        /// <para>
        /// The dynamic graph mode.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlEnum(Name = "dynamic")]
        Dynamic
    }
}
```

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

```
using System.Globalization;
using System.Runtime.CompilerServices;
using System.Xml;
using System.Xml.Serialization;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Gexf
{
    /// <summary>
    /// <para>
    /// Represents the node.
    /// </para>
    /// <para></para>
    /// </summary>
    public class Node
    {
        /// <summary>
        /// <para>
        /// The element name.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly string ElementName = "node";
        /// <summary>
        /// <para>
        /// The id attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string IdAttributeName = "id";
        /// <summary>
        /// <para>
        /// The label attribute name.
        /// </para>
        /// <para></para>
        /// </summary>
        public const string LabelAttributeName = "label";

        /// <summary>
        /// <para>
        /// Gets or sets the id value.
        /// </para>
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = IdAttributeName)]
        public long Id
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Gets or sets the label value.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        [XmlAttribute(AttributeName = LabelAttributeName)]
        public string Label
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Writes the xml using the specified writer.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="writer">
        /// <para>The writer.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Label);

        /// <summary>
        /// <para>
        /// Writes the xml using the specified writer.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="writer">
        /// <para>The writer.</para>
        /// <para></para>
        /// </param>
        /// <param name="id">
        /// <para>The id.</para>
        /// <para></para>
        /// </param>
        /// <param name="label">
        /// <para>The label.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, long id, string label)
        {
            // <node id="0" label="..." />
            writer.WriteStartElement(ElementName);
            writer.WriteAttributeString(IdAttributeName,
            ↪  id.ToString(CultureInfo.InvariantCulture));
            writer.WriteAttributeString(LabelAttributeName, label);
            writer.WriteEndElement();
        }
    }
}
```

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

```
using System.Net;
using System.Net.Sockets;
using System.Runtime.CompilerServices;
using System.Text;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Udp
{
    /// <summary>
    /// <para>
    /// Represents the udp client extensions.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class UdpClientExtensions
    {
        /// <summary>
        /// <para>
        /// The utf.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        public static readonly Encoding DefaultEncoding = Encoding.UTF8;

        /// <summary>
        /// <para>
        /// Sends the string using the specified udp.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="udp">
        /// <para>The udp.</para>
        /// <para></para>
        /// </param>
        /// <param name="ipEndPoint">
        /// <para>The ip end point.</para>
        /// <para></para>
        /// </param>
        /// <param name="message">
        /// <para>The message.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static int SendString(this UdpClient udp, IPEndPoint ipEndPoint, string message)
        {
            var bytes = DefaultEncoding.GetBytes(message);
            return udp.Send(bytes, bytes.Length, ipEndPoint);
        }

        /// <summary>
        /// <para>
        /// Receives the string using the specified udp.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="udp">
        /// <para>The udp.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string ReceiveString(this UdpClient udp)
        {
            IPEndPoint remoteEndPoint = default;
            return DefaultEncoding.GetString(udp.Receive(ref remoteEndPoint));
        }
    }
}
```

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

```
using System;
using System.Net.Sockets;
using System.Runtime.CompilerServices;
using System.Threading;
using Platform.Disposables;
using Platform.Exceptions;
using Platform.Threading;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Udp
{
    /// <summary>
    /// <para>
    /// The message handler callback.
    /// </para>
    /// <para></para>
    /// </summary>
    public delegate void MessageHandlerCallback(string message);

    /// <summary>
    /// <para>Represents the receiver of messages transfered via UDP protocol.</para>
    /// <para>Представляет получателя сообщений по протоколу UDP.</para>
    /// </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) :
            ↪  this(listenPort, true, messageHandler) { }
90
91          /// <summary>
92          /// <para>
93          /// Initializes a new <see cref="UdpReceiver"/> instance.
94          /// </para>
95          /// <para></para>
96          /// </summary>
97          /// <param name="messageHandler">
98          /// <para>A message handler.</para>
99          /// <para></para>
100         /// </param>
101         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
        public UdpReceiver(MessageHandlerCallback messageHandler) : this(DefaultPort, true,
        ↪  messageHandler) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UdpReceiver"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpReceiver() : this(DefaultPort, true, message => { }) { }

        /// <summary>
        /// <para>
        /// Starts this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Start()
        {
            if (!_receiverRunning && _thread == null)
            {
                _receiverRunning = true;
                _thread = new Thread(Receiver);
                _thread.Start();
            }
        }

        /// <summary>
        /// <para>
        /// Stops this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Stop()
        {
            if (_receiverRunning && _thread != null)
            {
                _receiverRunning = false;
                _thread.Join();
                _thread = null;
            }
        }

        /// <summary>
        /// <para>
        /// Receives this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public string Receive() => _udp.ReceiveString();

        /// <summary>
        /// <para>
        /// Receives the and handle.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void ReceiveAndHandle() => _messageHandler(Receive());
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void Receiver()
        {
            while (_receiverRunning)
            {
                try
                {
                    if (Available)
                    {
                        ReceiveAndHandle();
                    }
```

```
                    else
                    {
                        ThreadHelpers.Sleep();
                    }
                }
                catch (Exception exception)
                {
                    exception.Ignore();
                }
            }
        }

        /// <summary>
        /// <para>
        /// Disposes the manual.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="manual">
        /// <para>The manual.</para>
        /// <para></para>
        /// </param>
        /// <param name="wasDisposed">
        /// <para>The was disposed.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                Stop();
                _udp.DisposeIfPossible();
            }
        }
    }
}
```

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

```
using System.Net;
using System.Net.Sockets;
using System.Runtime.CompilerServices;
using Platform.Disposables;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Udp
{
    /// <summary>
    /// <para>Represents the sender of messages transfered via UDP protocol.</para>
    /// <para>Представляет отправителя сообщений по протоколу UDP.</para>
    /// </summary>
    public class UdpSender : DisposableBase //-V3073
    {
        private readonly UdpClient _udp;
        private readonly IPEndPoint _ipendpoint;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UdpSender"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="ipendpoint">
        /// <para>A ipendpoint.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(IPEndPoint ipendpoint) => (_udp, _ipendpoint) = (new UdpClient(),
        ↪  ipendpoint);

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UdpSender"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="address">
        /// <para>A address.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="port">
        /// <para>A port.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(IPAddress address, int port) : this(new IPEndPoint(address, port)) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UdpSender"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="hostname">
        /// <para>A hostname.</para>
        /// <para></para>
        /// </param>
        /// <param name="port">
        /// <para>A port.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(string hostname, int port) : this(IPAddress.Parse(hostname), port) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UdpSender"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="port">
        /// <para>A port.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(int port) : this(IPAddress.Loopback, port) { }

        /// <summary>
        /// <para>
        /// Sends the message.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="message">
        /// <para>The message.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public int Send(string message) => _udp.SendString(_ipendpoint, message);

        /// <summary>
        /// <para>
        /// Disposes the manual.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="manual">
        /// <para>The manual.</para>
        /// <para></para>
        /// </param>
        /// <param name="wasDisposed">
        /// <para>The was disposed.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                _udp.DisposeIfPossible();
            }
        }
```

```
    }
}
```

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

```
using System.IO;
using System.Runtime.CompilerServices;
using System.Text;
using System.Xml.Serialization;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Communication.Protocol.Xml
{
    /// <summary>
    /// <para>
    /// Represents the serializer.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class Serializer<T>
    {
        /// <summary>
        /// <para>
        /// The .
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly XmlSerializer Instance = new XmlSerializer(typeof(T));

        /// <summary>
        /// <para>
        /// Creates the file using the specified path.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="path">
        /// <para>The path.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T FromFile(string path)
        {
            using var stream = File.OpenRead(path);
            return (T)Instance.Deserialize(stream);
        }

        /// <summary>
        /// <para>
        /// Creates the string using the specified xml.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="xml">
        /// <para>The xml.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T FromString(string xml)
        {
            using var reader = new StringReader(xml);
            return (T)Instance.Deserialize(reader);
        }

        /// <summary>
        /// <para>
        /// Returns the file using the specified object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@object">
        /// <para>The object.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="path">
        /// <para>The path.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ToFile(T @object, string path)
        {
            using var stream = File.OpenWrite(path);
            Instance.Serialize(stream, @object);
        }

        /// <summary>
        /// <para>
        /// Returns the string using the specified object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@object">
        /// <para>The object.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string ToString(T @object)
        {
            var sb = new StringBuilder();
            using (var writer = new StringWriter(sb))
            {
                Instance.Serialize(writer, @object);
            }
            return sb.ToString();
        }
    }
}
```

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

```
using System;
using System.IO;
using Xunit;
using Platform.Singletons;
using Platform.Communication.Protocol.Xml;

namespace Platform.Communication.Tests
{
    public static class SerializerTests
    {
        [Fact]
        public static void SerializeToFileTest()
        {
            var tempFilename = Path.GetTempFileName();
            Serializer<object>.ToFile(Default<object>.Instance, tempFilename);
            Assert.Equal(File.ReadAllText(tempFilename), $"<?xml
            ↪  version=\"1.0\"?>{Environment.NewLine}<anyType
            ↪  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
            ↪  xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" />");
            File.Delete(tempFilename);
        }

        [Fact]
        public static void SerializeAsXmlStringTest()
        {
            var serializedObject = Serializer<object>.ToString(Default<object>.Instance);
            Assert.Equal(serializedObject, $"<?xml version=\"1.0\"
            ↪  encoding=\"utf-16\"?>{Environment.NewLine}<anyType
            ↪  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
            ↪  xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" />");
        }
    }
}
```

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

```
using Xunit;
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