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
{
    public class Edge
    {
        public static readonly string ElementName = "edge";
        public const string IdAttributeName = "id";
        public const string SourceAttributeName = "source";
        public const string TargetAttributeName = "target";
        public const string LabelAttributeName = "label";

        [XmlAttribute(AttributeName = IdAttributeName)]
        public long Id
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [XmlAttribute(AttributeName = SourceAttributeName)]
        public long Source
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [XmlAttribute(AttributeName = TargetAttributeName)]
        public long Target
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [XmlAttribute(AttributeName = LabelAttributeName)]
        public string Label
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Source, Target, Label);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
        ↪  targetNodeId) => WriteXml(writer, id, sourceNodeId, targetNodeId, null);

        [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
{
    [XmlRoot(ElementName = ElementName, Namespace = Namespace)]
    public class Gexf
    {
        public const string ElementName = "gexf";
        public const string Namespace = "http://www.gexf.net/1.2draft";
        public const string VersionAttributeName = "version";
        public const string GraphElementName = "graph";
        public static readonly string CurrentVersion = "1.2";

        [XmlAttribute(AttributeName = VersionAttributeName)]
        public string Version
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [XmlElement(ElementName = GraphElementName)]
        public Graph Graph
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Gexf() => (Version, Graph) = (CurrentVersion, new Graph());

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
        ↪  Version);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, Action writeGraph) => WriteXml(writer,
        ↪  writeGraph, CurrentVersion);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, Action writeGraph, string version)
        {
            writer.WriteStartDocument();
            writer.WriteStartElement(ElementName, Namespace);
            writer.WriteAttributeString(VersionAttributeName, version);
            writeGraph();
            writer.WriteEndElement();
            writer.WriteEndDocument();
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
        ↪  WriteXml(writer, writeNodes, writeEdges, CurrentVersion, GraphMode.Static,
        ↪  GraphDefaultEdgeType.Directed);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↪  string version) => WriteXml(writer, writeNodes, writeEdges, version,
        ↪  GraphMode.Static, GraphDefaultEdgeType.Directed);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↪  string version, GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, version,
        ↪  mode, GraphDefaultEdgeType.Directed);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
67          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);
68      }
69  }
```

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      public class Graph
12      {
13          public static readonly string ElementName = "graph";
14          public const string ModeAttributeName = "mode";
15          public const string DefaultEdgeTypeAttributeName = "defaultedgetype";
16          public const string NodesElementName = "nodes";
17          public const string NodeElementName = "node";
18          public const string EdgesElementName = "edges";
19          public const string EdgeElementName = "edge";
20
21          [XmlAttribute(AttributeName = ModeAttributeName)]
22          public GraphMode Mode
23          {
24              [MethodImpl(MethodImplOptions.AggressiveInlining)]
25              get;
26              [MethodImpl(MethodImplOptions.AggressiveInlining)]
27              set;
28          }
29
30          [XmlAttribute(AttributeName = DefaultEdgeTypeAttributeName)]
31          public GraphDefaultEdgeType DefaultEdgeType
32          {
33              [MethodImpl(MethodImplOptions.AggressiveInlining)]
34              get;
35              [MethodImpl(MethodImplOptions.AggressiveInlining)]
36              set;
37          }
38
39          [XmlArray(ElementName = NodesElementName)]
40          [XmlArrayItem(ElementName = NodeElementName)]
41          public List<Node> Nodes
42          {
43              [MethodImpl(MethodImplOptions.AggressiveInlining)]
44              get;
45              [MethodImpl(MethodImplOptions.AggressiveInlining)]
46              set;
47          }
48
49          [XmlArray(ElementName = EdgesElementName)]
50          [XmlArrayItem(ElementName = EdgeElementName)]
51          public List<Edge> Edges
52          {
53              [MethodImpl(MethodImplOptions.AggressiveInlining)]
54              get;
55              [MethodImpl(MethodImplOptions.AggressiveInlining)]
56              set;
57          }
58
59          [MethodImpl(MethodImplOptions.AggressiveInlining)]
60          public Graph() => (Nodes, Edges) = (new List<Node>(), new List<Edge>());
61
62          [MethodImpl(MethodImplOptions.AggressiveInlining)]
63          public void WriteXml(XmlWriter writer) => WriteXml(writer, () => WriteNodes(writer), ()
            ↪   => WriteEdges(writer), Mode, DefaultEdgeType);
64
65          [MethodImpl(MethodImplOptions.AggressiveInlining)]
66          private void WriteEdges(XmlWriter writer)
67          {
68              for (var i = 0; i < Edges.Count; i++)
69              {
70                  Edges[i].WriteXml(writer);
71              }
```

```
72          }
73
74          [MethodImpl(MethodImplOptions.AggressiveInlining)]
75          private void WriteNodes(XmlWriter writer)
76          {
77              for (var i = 0; i < Nodes.Count; i++)
78              {
79                  Nodes[i].WriteXml(writer);
80              }
81          }
82
83          [MethodImpl(MethodImplOptions.AggressiveInlining)]
84          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
            ↪  WriteXml(writer, writeNodes, writeEdges, GraphMode.Static,
            ↪  GraphDefaultEdgeType.Directed);
85
86          [MethodImpl(MethodImplOptions.AggressiveInlining)]
87          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
            ↪  GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, mode,
            ↪  GraphDefaultEdgeType.Directed);
88
89          [MethodImpl(MethodImplOptions.AggressiveInlining)]
90          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
            ↪  GraphMode mode, GraphDefaultEdgeType defaultEdgeType)
91          {
92              writer.WriteStartElement(ElementName);
93              writer.WriteAttributeString(ModeAttributeName, mode.ToString().ToLower());
94              writer.WriteAttributeString(DefaultEdgeTypeAttributeName,
                ↪  defaultEdgeType.ToString().ToLower());
95              writer.WriteStartElement(NodesElementName);
96              writeNodes();
97              writer.WriteEndElement();
98              writer.WriteStartElement(EdgesElementName);
99              writeEdges();
100             writer.WriteEndElement();
101             writer.WriteEndElement();
102         }
103     }
104 }
```

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      public enum GraphDefaultEdgeType
8      {
9          [XmlEnum(Name = "directed")]
10         Directed
11     }
12 }
```

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      public enum GraphMode
8      {
9          [XmlEnum(Name = "static")]
10         Static,
11
12         [XmlEnum(Name = "dynamic")]
13         Dynamic
14     }
15 }
```

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
```

```

namespace Platform.Communication.Protocol.Gexf
{
    public class Node
    {
        public static readonly string ElementName = "node";
        public const string IdAttributeName = "id";
        public const string LabelAttributeName = "label";

        [XmlAttribute(AttributeName = IdAttributeName)]
        public long Id
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [XmlAttribute(AttributeName = LabelAttributeName)]
        public string Label
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Label);

        [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;
using Platform.Singletons;

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

namespace Platform.Communication.Protocol.Udp
{
    public static class UdpClientExtensions
    {
        public static readonly Encoding DefaultEncoding = Encoding.UTF8;

        [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);
        }

        [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
{
    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>
    public class UdpReceiver : DisposableBase //-V3073
    {
        private const int DefaultPort = 15000;

        private bool _receiverRunning;
        private Thread _thread;
        private readonly UdpClient _udp;
        private readonly MessageHandlerCallback _messageHandler;

        public bool Available
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _udp.Available > 0;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpReceiver(int listenPort, bool autoStart, MessageHandlerCallback messageHandler)
        {
            _udp = new UdpClient(listenPort);
            _messageHandler = messageHandler;
            if (autoStart)
            {
                Start();
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpReceiver(int listenPort, MessageHandlerCallback messageHandler) :
        ↪  this(listenPort, true, messageHandler) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpReceiver(MessageHandlerCallback messageHandler) : this(DefaultPort, true,
        ↪  messageHandler) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpReceiver() : this(DefaultPort, true, message => { }) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Start()
        {
            if (!_receiverRunning && _thread == null)
            {
                _receiverRunning = true;
                _thread = new Thread(Receiver);
                _thread.Start();
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Stop()
        {
            if (_receiverRunning && _thread != null)
            {
                _receiverRunning = false;
                _thread.Join();
                _thread = null;
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public string Receive() => _udp.ReceiveString();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void ReceiveAndHandle() => _messageHandler(Receive());

        /// <remarks>
        /// <para>The method receives messages and runs in a separate thread.</para>
```

```
        /// <para>Метод получает сообщения и работает в отдельном потоке.</para>
        /// </remarks>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void Receiver()
        {
            while (_receiverRunning)
            {
                try
                {
                    if (Available)
                    {
                        ReceiveAndHandle();
                    }
                    else
                    {
                        ThreadHelpers.Sleep();
                    }
                }
                catch (Exception exception)
                {
                    exception.Ignore();
                }
            }
        }

        [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;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(IPEndPoint ipendpoint) => (_udp, _ipendpoint) = (new UdpClient(),
        ↪  ipendpoint);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(IPAddress address, int port) : this(new IPEndPoint(address, port)) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(string hostname, int port) : this(IPAddress.Parse(hostname), port) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UdpSender(int port) : this(IPAddress.Loopback, port) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public int Send(string message) => _udp.SendString(_ipendpoint, message);

        [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
{
    public static class Serializer<T>
    {
        public static readonly XmlSerializer Instance = new XmlSerializer(typeof(T));

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T FromFile(string path)
        {
            using var stream = File.OpenRead(path);
            return (T)Instance.Deserialize(stream);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T FromString(string xml)
        {
            using var reader = new StringReader(xml);
            return (T)Instance.Deserialize(reader);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ToFile(T @object, string path)
        {
            using var stream = File.OpenWrite(path);
            Instance.Serialize(stream, @object);
        }

        [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;

namespace Platform.Communication.Tests
{
    public static class UdpReceiverTests
    {
        [Fact]
        public static void DisposalTest()
        {
            using var receiver = new UdpReceiver();
        }
    }
}
```

Index