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

```
using System.Globalization;
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 { get; set; }

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

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

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

        public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Source, Target, Label);

        public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
        ↪  targetNodeId) => WriteXml(writer, id, sourceNodeId, targetNodeId, null);

        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();
        }
    }
}
```

./Platform.Communication/Protocol/Gexf/Gexf.cs

```
using System;
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 { get; set; }

        [XmlElement(ElementName = GraphElementName)]
        public Graph Graph { get; set; }

        public Gexf() => (Version, Graph) = (CurrentVersion, new Graph());
```

```
25
26          public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
            ↪  Version);
27
28          public static void WriteXml(XmlWriter writer, Action writeGraph) => WriteXml(writer,
            ↪  writeGraph, CurrentVersion);
29
30          public static void WriteXml(XmlWriter writer, Action writeGraph, string version)
31          {
32              writer.WriteStartDocument();
33              writer.WriteStartElement(ElementName, Namespace);
34              writer.WriteAttributeString(VersionAttributeName, version);
35              writeGraph();
36              writer.WriteEndElement();
37              writer.WriteEndDocument();
38          }
39
40          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
            ↪  WriteXml(writer, writeNodes, writeEdges, CurrentVersion, GraphMode.Static,
            ↪  GraphDefaultEdgeType.Directed);
41
42          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
            ↪  string version) => WriteXml(writer, writeNodes, writeEdges, version,
            ↪  GraphMode.Static, GraphDefaultEdgeType.Directed);
43
44          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
            ↪  string version, GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, version,
            ↪  mode, GraphDefaultEdgeType.Directed);
45
46          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);
47      }
48  }
```

./Platform.Communication/Protocol/Gexf/Graph.cs

```
1   using System;
2   using System.Collections.Generic;
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      public class Graph
11      {
12          public static readonly string ElementName = "graph";
13          public const string ModeAttributeName = "mode";
14          public const string DefaultEdgeTypeAttributeName = "defaultedgetype";
15          public const string NodesElementName = "nodes";
16          public const string NodeElementName = "node";
17          public const string EdgesElementName = "edges";
18          public const string EdgeElementName = "edge";
19
20          [XmlAttribute(AttributeName = ModeAttributeName)]
21          public GraphMode Mode { get; set; }
22
23          [XmlAttribute(AttributeName = DefaultEdgeTypeAttributeName)]
24          public GraphDefaultEdgeType DefaultEdgeType { get; set; }
25
26          [XmlArray(ElementName = NodesElementName)]
27          [XmlArrayItem(ElementName = NodeElementName)]
28          public List<Node> Nodes { get; set; }
29
30          [XmlArray(ElementName = EdgesElementName)]
31          [XmlArrayItem(ElementName = EdgeElementName)]
32          public List<Edge> Edges { get; set; }
33
34          public Graph() => (Nodes, Edges) = (new List<Node>(), new List<Edge>());
35
36          public void WriteXml(XmlWriter writer) => WriteXml(writer, () => WriteNodes(writer), ()
            ↪  => WriteEdges(writer), Mode, DefaultEdgeType);
37
38          private void WriteEdges(XmlWriter writer)
39          {
40              for (var i = 0; i < Edges.Count; i++)
41              {
```

```
                Edges[i].WriteXml(writer);
            }
        }

        private void WriteNodes(XmlWriter writer)
        {
            for (var i = 0; i < Nodes.Count; i++)
            {
                Nodes[i].WriteXml(writer);
            }
        }

        public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
        ↪ WriteXml(writer, writeNodes, writeEdges, GraphMode.Static,
        ↪ GraphDefaultEdgeType.Directed);

        public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↪ GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, mode,
        ↪ GraphDefaultEdgeType.Directed);

        public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↪ GraphMode mode, GraphDefaultEdgeType defaultEdgeType)
        {
            writer.WriteStartElement(ElementName);
            writer.WriteAttributeString(ModeAttributeName, mode.ToString().ToLower());
            writer.WriteAttributeString(DefaultEdgeTypeAttributeName,
            ↪  defaultEdgeType.ToString().ToLower());
            writer.WriteStartElement(NodesElementName);
            writeNodes();
            writer.WriteEndElement();
            writer.WriteStartElement(EdgesElementName);
            writeEdges();
            writer.WriteEndElement();
            writer.WriteEndElement();
        }
    }
}
```

./Platform.Communication/Protocol/Gexf/GraphDefaultEdgeType.cs

```
using System.Xml.Serialization;

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

namespace Platform.Communication.Protocol.Gexf
{
    public enum GraphDefaultEdgeType
    {
        [XmlEnum(Name = "directed")]
        Directed
    }
}
```

./Platform.Communication/Protocol/Gexf/GraphMode.cs

```
using System.Xml.Serialization;

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

namespace Platform.Communication.Protocol.Gexf
{
    public enum GraphMode
    {
        [XmlEnum(Name = "static")]
        Static,

        [XmlEnum(Name = "dynamic")]
        Dynamic
    }
}
```

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

```
using System.Globalization;
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 Node
```

```
    {
        public static readonly string ElementName = "node";
        public const string IdAttributeName = "id";
        public const string LabelAttributeName = "label";

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

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

        public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Label);

        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();
        }
    }
}
```

./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
    {
        private static readonly Encoding _defaultEncoding = Singleton.Get(() =>
        ↪  Encoding.GetEncoding(0));

        [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));
        }
    }
}
```

./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>
    /// Represents the receiver of messages transfered via UDP protocol.
    /// Представляет получателя сообщений по протоколу UDP.
    /// </summary>
    public class UdpReceiver : DisposableBase //-V3073
    {
        private const int DefaultPort = 15000;

```

```
23          private bool _receiverRunning;
24          private Thread _thread;
25          private readonly UdpClient _udp;
26          private readonly MessageHandlerCallback _messageHandler;
27
28          public bool Available => _udp.Available > 0;
29
30          public UdpReceiver(int listenPort, bool autoStart, MessageHandlerCallback messageHandler)
31          {
32              _udp = new UdpClient(listenPort);
33              _messageHandler = messageHandler;
34              if (autoStart)
35              {
36                  Start();
37              }
38          }
39
40          public UdpReceiver(int listenPort, MessageHandlerCallback messageHandler) :
            ↪  this(listenPort, true, messageHandler) { }
41
42          public UdpReceiver(MessageHandlerCallback messageHandler) : this(DefaultPort, true,
            ↪  messageHandler) { }
43
44          public UdpReceiver() : this(DefaultPort, true, message => { }) { }
45
46          public void Start()
47          {
48              if (!_receiverRunning && _thread == null)
49              {
50                  _receiverRunning = true;
51                  _thread = new Thread(Receiver);
52                  _thread.Start();
53              }
54          }
55
56          public void Stop()
57          {
58              if (_receiverRunning && _thread != null)
59              {
60                  _receiverRunning = false;
61                  _thread.Join();
62                  _thread = null;
63              }
64          }
65
66          [MethodImpl(MethodImplOptions.AggressiveInlining)]
67          public string Receive() => _udp.ReceiveString();
68
69          [MethodImpl(MethodImplOptions.AggressiveInlining)]
70          public void ReceiveAndHandle() => _messageHandler(Receive());
71
72          // Функция извлекающая пришедшие сообщения
73          // и работающая в отдельном потоке.
74          private void Receiver()
75          {
76              while (_receiverRunning)
77              {
78                  try
79                  {
80                      if (Available)
81                      {
82                          ReceiveAndHandle();
83                      }
84                      else
85                      {
86                          ThreadHelpers.Sleep();
87                      }
88                  }
89                  catch (Exception exception)
90                  {
91                      exception.Ignore();
92                  }
93              }
94          }
95
96          protected override void Dispose(bool manual, bool wasDisposed)
97          {
98              if (!wasDisposed)
99              {
```

```
                Stop();
                _udp.DisposeIfPossible();
            }
        }
    }
}
```

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

        public UdpSender(IPEndPoint ipendpoint) => (_udp, _ipendpoint) = (new UdpClient(),
        ↪  ipendpoint);

        public UdpSender(IPAddress address, int port) : this(new IPEndPoint(address, port)) { }

        public UdpSender(string hostname, int port) : this(IPAddress.Parse(hostname), port) { }

        public UdpSender(int port) : this(IPAddress.Loopback, port) { }

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

        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                _udp.DisposeIfPossible();
            }
        }
    }
}
```

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

```
using System.IO;
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));

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

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

        public static void ToFile(T @object, string path)
        {
```

```
            using (var stream = File.OpenWrite(path))
            {
                Instance.Serialize(stream, @object);
            }
        }

        public static string ToString(T @object)
        {
            var sb = new StringBuilder();
            using (var writer = new StringWriter(sb))
            {
                Instance.Serialize(writer, @object);
            }
            return sb.ToString();
        }
    }
}
```

./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\" />");
        }
    }
}
```

./Platform.Communication.Tests/UdpReceiverTests.cs

```
using Xunit;
using Platform.Communication.Protocol.Udp;

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

Index