LinksPlatform's Platform.Communication Class Library

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

```csharp
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

```csharp
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()
```

```
25          {
26              Version = CurrentVersion;
27              Graph = new Graph();
28          }
29
30          public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
   ↪    Version);
31
32          public static void WriteXml(XmlWriter writer, Action writeGraph) => WriteXml(writer,
   ↪    writeGraph, CurrentVersion);
33
34          public static void WriteXml(XmlWriter writer, Action writeGraph, string version)
35          {
36              writer.WriteStartDocument();
37              writer.WriteStartElement(ElementName, Namespace);
38              writer.WriteAttributeString(VersionAttributeName, version);
39              writeGraph();
40              writer.WriteEndElement();
41              writer.WriteEndDocument();
42          }
43
44          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
   ↪    WriteXml(writer, writeNodes, writeEdges, CurrentVersion, GraphMode.Static,
   ↪    GraphDefaultEdgeType.Directed);
45
46          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
   ↪    string version) => WriteXml(writer, writeNodes, writeEdges, version,
   ↪    GraphMode.Static, GraphDefaultEdgeType.Directed);
47
48          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
   ↪    string version, GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, version,
   ↪    mode, GraphDefaultEdgeType.Directed);
49
50          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);
51      }
52  }
```

./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()
35         {
36             Nodes = new List<Node>();
37             Edges = new List<Edge>();
38         }
```

```
39
40          public void WriteXml(XmlWriter writer) => WriteXml(writer, () => WriteNodes(writer), ()
        ↪ => WriteEdges(writer), Mode, DefaultEdgeType);
41
42          private void WriteEdges(XmlWriter writer)
43          {
44              for (var i = 0; i < Edges.Count; i++)
45              {
46                  Edges[i].WriteXml(writer);
47              }
48          }
49
50          private void WriteNodes(XmlWriter writer)
51          {
52              for (var i = 0; i < Nodes.Count; i++)
53              {
54                  Nodes[i].WriteXml(writer);
55              }
56          }
57
58          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
        ↪ WriteXml(writer, writeNodes, writeEdges, GraphMode.Static,
        ↪ GraphDefaultEdgeType.Directed);
59
60          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↪ GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, mode,
        ↪ GraphDefaultEdgeType.Directed);
61
62          public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
        ↪ GraphMode mode, GraphDefaultEdgeType defaultEdgeType)
63          {
64              writer.WriteStartElement(ElementName);
65              writer.WriteAttributeString(ModeAttributeName, mode.ToString().ToLower());
66              writer.WriteAttributeString(DefaultEdgeTypeAttributeName,
        ↪ defaultEdgeType.ToString().ToLower());
67              writer.WriteStartElement(NodesElementName);
68              writeNodes();
69              writer.WriteEndElement();
70              writer.WriteStartElement(EdgesElementName);
71              writeEdges();
72              writer.WriteEndElement();
73              writer.WriteEndElement();
74          }
75      }
76  }
```

./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  }
```

./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  }
```

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

```csharp
1  using System.Globalization;
2  using System.Xml;
3  using System.Xml.Serialization;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Communication.Protocol.Gexf
8  {
9      public class Node
10     {
11         public static readonly string ElementName = "node";
12         public const string IdAttributeName = "id";
13         public const string LabelAttributeName = "label";
14
15         [XmlAttribute(AttributeName = IdAttributeName)]
16         public long Id { get; set; }
17
18         [XmlAttribute(AttributeName = LabelAttributeName)]
19         public string Label { get; set; }
20
21         public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Label);
22
23         public static void WriteXml(XmlWriter writer, long id, string label)
24         {
25             // <node id="0" label="..." />
26             writer.WriteStartElement(ElementName);
27             writer.WriteAttributeString(IdAttributeName,
                    id.ToString(CultureInfo.InvariantCulture));
28             writer.WriteAttributeString(LabelAttributeName, label);
29             writer.WriteEndElement();
30         }
31     }
32 }
```

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

```csharp
1  using System.Net;
2  using System.Net.Sockets;
3  using System.Runtime.CompilerServices;
4  using System.Text;
5  using Platform.Singletons;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Communication.Protocol.Udp
10 {
11     public static class UdpClientExtensions
12     {
13         private static readonly Encoding _defaultEncoding = Singleton.Get(() =>
                Encoding.GetEncoding(0));
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static int SendString(this UdpClient udp, IPEndPoint ipEndPoint, string message)
17         {
18             var bytes = _defaultEncoding.GetBytes(message);
19             return udp.Send(bytes, bytes.Length, ipEndPoint);
20         }
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public static string ReceiveString(this UdpClient udp)
24         {
25             IPEndPoint remoteEndPoint = default;
26             return _defaultEncoding.GetString(udp.Receive(ref remoteEndPoint));
27         }
28     }
29 }
```

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

```csharp
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 {
```

```csharp
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;

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

    public bool Available => _udp.Available > 0;

    public UdpReceiver(int listenPort, bool autoStart, MessageHandlerCallback messageHandler)
    {
        _udp = new UdpClient(listenPort);
        _messageHandler = messageHandler;
        if (autoStart)
        {
            Start();
        }
    }

    public UdpReceiver(int listenPort, MessageHandlerCallback messageHandler)
        : this(listenPort, true, messageHandler)
    {
    }

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

    public UdpReceiver()
        : this(DefaultPort, true, message => { })
    {
    }

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

    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());

    // Функция извлекающая пришедшие сообщения
    // и работающая в отдельном потоке.
    private void Receiver()
    {
        while (_receiverRunning)
        {
            try
            {
                if (Available)
                {
                    ReceiveAndHandle();
                }
```

```
 93                else
 94                {
 95                    ThreadHelpers.Sleep();
 96                }
 97            }
 98            catch (Exception exception)
 99            {
100                exception.Ignore();
101            }
102        }
103    }
104
105    protected override void Dispose(bool manual, bool wasDisposed)
106    {
107        if (!wasDisposed)
108        {
109            Stop();
110            _udp.DisposeIfPossible();
111        }
112    }
113    }
114 }
```

./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      /// Represents the sender of messages transfered via UDP protocol.
12      /// Представляет отправителя сообщений по протоколу UDP.
13      /// </summary>
14      public class UdpSender : DisposableBase //-V3073
15      {
16          private readonly UdpClient _udp;
17          private readonly IPEndPoint _ipendpoint;
18
19          public UdpSender(IPEndPoint ipendpoint)
20          {
21              _udp = new UdpClient();
22              _ipendpoint = ipendpoint;
23          }
24
25          public UdpSender(IPAddress address, int port)
26              : this(new IPEndPoint(address, port))
27          {
28          }
29
30          public UdpSender(string hostname, int port)
31              : this(IPAddress.Parse(hostname), port)
32          {
33          }
34
35          public UdpSender(int port)
36              : this(IPAddress.Loopback, port)
37          {
38          }
39
40          [MethodImpl(MethodImplOptions.AggressiveInlining)]
41          public int Send(string message) => _udp.SendString(_ipendpoint, message);
42
43          protected override void Dispose(bool manual, bool wasDisposed)
44          {
45              if (!wasDisposed)
46              {
47                  _udp.DisposeIfPossible();
48              }
49          }
50      }
51  }
```

./Platform.Communication/Protocol/Xml/Serializer.cs
```
 1  using System.IO;
 2  using System.Text;
```

```csharp
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

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

```
26        }
27    }
```

./Platform.Communication.Tests/UdpReceiverTests.cs

```csharp
1   using Xunit;
2   using Platform.Communication.Protocol.Udp;
3
4   namespace Platform.Communication.Tests
5   {
6       public class UdpReceiverTests
7       {
8           [Fact]
9           public static void DisposalTest()
10          {
11              using (var receiver = new UdpReceiver())
12              {
13              }
14          }
15      }
16  }
```

Index