LinksPlatform's Platform.Communication Class Library

./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/Enums.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
    }

    public enum GraphDefaultEdgeType
    {
        [XmlEnum(Name = "directed")]
        Directed
    }
}
```

./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()
        {
            Version = CurrentVersion;
            Graph = new Graph();
        }

        public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
         ↪  Version);

        public static void WriteXml(XmlWriter writer, Action writeGraph) => WriteXml(writer,
         ↪  writeGraph, CurrentVersion);

        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();
        }

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

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

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

        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);
    }
}
```

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

```csharp
using System;
using System.Collections.Generic;
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 Graph
    {
        public static readonly string ElementName = "graph";
        public const string ModeAttributeName = "mode";
        public const string DefaultEdgeTypeAttributeName = "defaultedgetype";
        public const string NodesElementName = "nodes";
        public const string NodeElementName = "node";
        public const string EdgesElementName = "edges";
        public const string EdgeElementName = "edge";

        [XmlAttribute(AttributeName = ModeAttributeName)]
        public GraphMode Mode { get; set; }

        [XmlAttribute(AttributeName = DefaultEdgeTypeAttributeName)]
        public GraphDefaultEdgeType DefaultEdgeType { get; set; }

        [XmlArray(ElementName = NodesElementName)]
        [XmlArrayItem(ElementName = NodeElementName)]
        public List<Node> Nodes { get; set; }

        [XmlArray(ElementName = EdgesElementName)]
        [XmlArrayItem(ElementName = EdgeElementName)]
        public List<Edge> Edges { get; set; }

        public Graph()
        {
            Nodes = new List<Node>();
            Edges = new List<Edge>();
        }

        public void WriteXml(XmlWriter writer) => WriteXml(writer, () => WriteNodes(writer), ()
         ↪  => WriteEdges(writer), Mode, DefaultEdgeType);

        private void WriteEdges(XmlWriter writer)
        {
            for (var i = 0; i < Edges.Count; i++)
            {
                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/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;
```

```csharp

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

        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                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 = new UdpClient();
            _ipendpoint = 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