

# LinksPlatform's Platform.Communication Class Library

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

```
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 Edge
10     {
11         public static readonly string ElementName = "edge";
12         public const string IdAttributeName = "id";
13         public const string SourceAttributeName = "source";
14         public const string TargetAttributeName = "target";
15         public const string LabelAttributeName = "label";
16
17         [XmlAttribute(AttributeName = IdAttributeName)]
18         public long Id { get; set; }
19
20         [XmlAttribute(AttributeName = SourceAttributeName)]
21         public long Source { get; set; }
22
23         [XmlAttribute(AttributeName = TargetAttributeName)]
24         public long Target { get; set; }
25
26         [XmlAttribute(AttributeName = LabelAttributeName)]
27         public string Label { get; set; }
28
29         public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Source, Target, Label);
30
31         public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
32             ↪ targetNodeId) => WriteXml(writer, id, sourceNodeId, targetNodeId, null);
33
34         public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
35             ↪ targetNodeId, string label)
36         {
37             // <edge id="0" source="0" target="0" label="..." />
38             writer.WriteStartElement(ElementName);
39             writer.WriteAttributeString(IdAttributeName,
40                 ↪ id.ToString(CultureInfo.InvariantCulture));
41             writer.WriteAttributeString(SourceAttributeName,
42                 ↪ sourceNodeId.ToString(CultureInfo.InvariantCulture));
43             writer.WriteAttributeString(TargetAttributeName,
44                 ↪ targetNodeId.ToString(CultureInfo.InvariantCulture));
45             if (!string.IsNullOrEmpty(label))
46             {
47                 writer.WriteAttributeString(LabelAttributeName, label);
48             }
49             writer.WriteEndElement();
50         }
51     }
52 }
```

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

```
1 using System;
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     [XmlRoot(ElementName = ElementName, Namespace = Namespace)]
10     public class Gexf
11     {
12         public const string ElementName = "gexf";
13         public const string Namespace = "http://www.gexf.net/1.2draft";
14         public const string VersionAttributeName = "version";
15         public const string GraphElementName = "graph";
16         public static readonly string CurrentVersion = "1.2";
17
18         [XmlAttribute(AttributeName = VersionAttributeName)]
19         public string Version { get; set; }
20
21         [XmlElement(ElementName = GraphElementName)]
22         public Graph Graph { get; set; }
23
24         public Gexf() => (Version, Graph) = (CurrentVersion, new Graph());
25     }
26 }
```

```

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 }

```

### 1.3 ./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             {

```

```

42         Edges[i].WriteXml(writer);
43     }
44 }
45
46 private void WriteNodes(XmlWriter writer)
47 {
48     for (var i = 0; i < Nodes.Count; i++)
49     {
50         Nodes[i].WriteXml(writer);
51     }
52 }
53
54 public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
55     ↪ WriteXml(writer, writeNodes, writeEdges, GraphMode.Static,
56     ↪ GraphDefaultEdgeType.Directed);
57
58 public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
59     ↪ GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, mode,
60     ↪ GraphDefaultEdgeType.Directed);
61
62 public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
63     ↪ GraphMode mode, GraphDefaultEdgeType defaultEdgeType)
64 {
65     writer.WriteStartElement(ElementName);
66     writer.WriteAttributeString(ModeAttributeName, mode.ToString().ToLower());
67     writer.WriteAttributeString(DefaultEdgeTypeAttributeName,
68     ↪ defaultEdgeType.ToString().ToLower());
69     writer.WriteStartElement(NodesElementName);
70     writeNodes();
71     writer.WriteEndElement();
72     writer.WriteStartElement(EdgesElementName);
73     writeEdges();
74     writer.WriteEndElement();
75     writer.WriteEndElement();
76 }
77 }

```

#### 1.4 ./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         [XmlAttribute(Name = "directed")]
10         Directed
11     }
12 }

```

#### 1.5 ./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         [XmlAttribute(Name = "static")]
10         Static,
11
12         [XmlAttribute(Name = "dynamic")]
13         Dynamic
14     }
15 }

```

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

```

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,
28             ↪ id.ToString(CultureInfo.InvariantCulture));
29         writer.WriteAttributeString(LabelAttributeName, label);
30         writer.WriteEndElement();
31     }
32 }

```

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

```

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(() =>
14             ↪ Encoding.GetEncoding(0));
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static int SendString(this UdpClient udp, IPEndPoint ipEndPoint, string message)
18         {
19             var bytes = _defaultEncoding.GetBytes(message);
20             return udp.Send(bytes, bytes.Length, ipEndPoint);
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static string ReceiveString(this UdpClient udp)
25         {
26             IPEndPoint remoteEndPoint = default;
27             return _defaultEncoding.GetString(udp.Receive(ref remoteEndPoint));
28         }
29     }
30 }

```

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

```

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 {
13     public delegate void MessageHandlerCallback(string message);
14
15     /// <summary>
16     /// Represents the receiver of messages transfered via UDP protocol.
17     /// Представляет получателя сообщений по протоколу UDP.
18     /// </summary>
19     public class UdpReceiver : DisposableBase //-V3073
20     {
21         private const int DefaultPort = 15000;
22     }
23 }

```

```

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) :
41     → this(listenPort, true, messageHandler) { }
42
43 public UdpReceiver(MessageHandlerCallback messageHandler) : this(DefaultPort, true,
44     → messageHandler) { }
45
46 public UdpReceiver() : this(DefaultPort, true, message => { }) { }
47
48 public void Start()
49 {
50     if (!_receiverRunning && _thread == null)
51     {
52         _receiverRunning = true;
53         _thread = new Thread(Receiver);
54         _thread.Start();
55     }
56 }
57
58 public void Stop()
59 {
60     if (_receiverRunning && _thread != null)
61     {
62         _receiverRunning = false;
63         _thread.Join();
64         _thread = null;
65     }
66 }
67
68 [MethodImpl(MethodImplOptions.AggressiveInlining)]
69 public string Receive() => _udp.ReceiveString();
70
71 [MethodImpl(MethodImplOptions.AggressiveInlining)]
72 public void ReceiveAndHandle() => _messageHandler(Receive());
73
74 // Функция извлекающая пришедшие сообщения
75 // и работающая в отдельном потоке.
76 private void Receiver()
77 {
78     while (_receiverRunning)
79     {
80         try
81         {
82             if (Available)
83             {
84                 ReceiveAndHandle();
85             }
86             else
87             {
88                 ThreadHelpers.Sleep();
89             }
90         }
91         catch (Exception exception)
92         {
93             exception.Ignore();
94         }
95     }
96 }
97
98 protected override void Dispose(bool manual, bool wasDisposed)
99 {
100     if (!wasDisposed)
101     {

```

```

100         Stop();
101         _udp.DisposeIfPossible();
102     }
103 }
104 }
105 }

```

### 1.9 ./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 transferred 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) => (_udp, _ipendpoint) = (new UdpClient(),
20             ↪ ipendpoint);
21
22         public UdpSender(IPAddress address, int port) : this(new IPEndPoint(address, port)) { }
23
24         public UdpSender(string hostname, int port) : this(IPAddress.Parse(hostname), port) { }
25
26         public UdpSender(int port) : this(IPAddress.Loopback, port) { }
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public int Send(string message) => _udp.SendString(_ipendpoint, message);
30
31         protected override void Dispose(bool manual, bool wasDisposed)
32         {
33             if (!wasDisposed)
34             {
35                 _udp.DisposeIfPossible();
36             }
37         }
38     }
39 }

```

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

```

1  using System.IO;
2  using System.Text;
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.Xml
8  {
9     public static class Serializer<T>
10     {
11         public static readonly XmlSerializer Instance = new XmlSerializer(typeof(T));
12
13         public static T FromFile(string path)
14         {
15             using (var stream = File.OpenRead(path))
16             {
17                 return (T)Instance.Deserialize(stream);
18             }
19         }
20
21         public static T FromString(string xml)
22         {
23             using (var reader = new StringReader(xml))
24             {
25                 return (T)Instance.Deserialize(reader);
26             }
27         }
28
29         public static void ToFile(T @object, string path)
30         {

```

```

31         using (var stream = File.OpenWrite(path))
32         {
33             Instance.Serialize(stream, @object);
34         }
35     }
36
37     public static string ToString(T @object)
38     {
39         var sb = new StringBuilder();
40         using (var writer = new StringWriter(sb))
41         {
42             Instance.Serialize(writer, @object);
43         }
44         return sb.ToString();
45     }
46 }
47 }

```

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

```

1  using System;
2  using System.IO;
3  using Xunit;
4  using Platform.Singletons;
5  using Platform.Communication.Protocol.Xml;
6
7  namespace Platform.Communication.Tests
8  {
9      public static class SerializerTests
10     {
11         [Fact]
12         public static void SerializeToFileTest()
13         {
14             var tempFilename = Path.GetTempFileName();
15             Serializer<object>.ToFile(Default<object>.Instance, tempFilename);
16             Assert.Equal(File.ReadAllText(tempFilename), $"<?xml
17                 ↪ version=\"1.0\"?>{Environment.NewLine}<anyType
18                 ↪ xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
19                 ↪ xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" />");
20             File.Delete(tempFilename);
21         }
22
23         [Fact]
24         public static void SerializeAsXmlStringTest()
25         {
26             var serializedObject = Serializer<object>.ToString(Default<object>.Instance);
27             Assert.Equal(serializedObject, $"<?xml version=\"1.0\"
28                 ↪ encoding=\"utf-16\"?>{Environment.NewLine}<anyType
29                 ↪ xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
30                 ↪ xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" />");
31         }
32     }
33 }

```

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

```

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

- ./Platform.Communication.Tests/SerializerTests.cs, 7
- ./Platform.Communication.Tests/UdpReceiverTests.cs, 7
- ./Platform.Communication/Protocol/Gexf/Edge.cs, 1
- ./Platform.Communication/Protocol/Gexf/Gexf.cs, 1
- ./Platform.Communication/Protocol/Gexf/Graph.cs, 2
- ./Platform.Communication/Protocol/Gexf/GraphDefaultEdgeType.cs, 3
- ./Platform.Communication/Protocol/Gexf/GraphMode.cs, 3
- ./Platform.Communication/Protocol/Gexf/Node.cs, 3
- ./Platform.Communication/Protocol/Udp/UdpClientExtensions.cs, 4
- ./Platform.Communication/Protocol/Udp/UdpReceiver.cs, 4
- ./Platform.Communication/Protocol/Udp/UdpSender.cs, 6
- ./Platform.Communication/Protocol/Xml/Serializer.cs, 6