

LinksPlatform's Platform.Communication Class Library

1.1 ./csharp/Platform.Communication/Protocol/Gexf/Edge.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
7
8  namespace Platform.Communication.Protocol.Gexf
9  {
10     public class Edge
11     {
12         public static readonly string ElementName = "edge";
13         public const string IdAttributeName = "id";
14         public const string SourceAttributeName = "source";
15         public const string TargetAttributeName = "target";
16         public const string LabelAttributeName = "label";
17
18         [XmlAttribute(AttributeName = IdAttributeName)]
19         public long Id
20         {
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             get;
23             [MethodImpl(MethodImplOptions.AggressiveInlining)]
24             set;
25         }
26
27         [XmlAttribute(AttributeName = SourceAttributeName)]
28         public long Source
29         {
30             [MethodImpl(MethodImplOptions.AggressiveInlining)]
31             get;
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             set;
34         }
35
36         [XmlAttribute(AttributeName = TargetAttributeName)]
37         public long Target
38         {
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             get;
41             [MethodImpl(MethodImplOptions.AggressiveInlining)]
42             set;
43         }
44
45         [XmlAttribute(AttributeName = LabelAttributeName)]
46         public string Label
47         {
48             [MethodImpl(MethodImplOptions.AggressiveInlining)]
49             get;
50             [MethodImpl(MethodImplOptions.AggressiveInlining)]
51             set;
52         }
53
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Source, Target, Label);
56
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
59             ↪ targetNodeId) => WriteXml(writer, id, sourceNodeId, targetNodeId, null);
60
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         public static void WriteXml(XmlWriter writer, long id, long sourceNodeId, long
63             ↪ targetNodeId, string label)
64         {
65             // <edge id="0" source="0" target="0" label="..." />
66             writer.WriteStartElement(ElementName);
67             writer.WriteAttributeString(IdAttributeName,
68                 ↪ id.ToString(CultureInfo.InvariantCulture));
69             writer.WriteAttributeString(SourceAttributeName,
70                 ↪ sourceNodeId.ToString(CultureInfo.InvariantCulture));
71             writer.WriteAttributeString(TargetAttributeName,
72                 ↪ targetNodeId.ToString(CultureInfo.InvariantCulture));
73             if (!string.IsNullOrEmpty(label))
74             {
75                 writer.WriteAttributeString(LabelAttributeName, label);
76             }
77             writer.WriteEndElement();
78         }
79     }
80 }
```

```
74     }
75 }
```

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

```
1 using System;
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
7
8 namespace Platform.Communication.Protocol.Gexf
9 {
10     [XmlRoot(ElementName = ElementName, Namespace = Namespace)]
11     public class Gexf
12     {
13         public const string ElementName = "gexf";
14         public const string Namespace = "http://www.gexf.net/1.2draft";
15         public const string VersionAttributeName = "version";
16         public const string GraphElementName = "graph";
17         public static readonly string CurrentVersion = "1.2";
18
19         [XmlAttribute(AttributeName = VersionAttributeName)]
20         public string Version
21         {
22             [MethodImpl(MethodImplOptions.AggressiveInlining)]
23             get;
24             [MethodImpl(MethodImplOptions.AggressiveInlining)]
25             set;
26         }
27
28         [XmlElement(ElementName = GraphElementName)]
29         public Graph Graph
30         {
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             get;
33             [MethodImpl(MethodImplOptions.AggressiveInlining)]
34             set;
35         }
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public Gexf() => (Version, Graph) = (CurrentVersion, new Graph());
39
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public void WriteXml(XmlWriter writer) => WriteXml(writer, () => Graph.WriteXml(writer),
42             ↪ Version);
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public static void WriteXml(XmlWriter writer, Action writeGraph) => WriteXml(writer,
46             ↪ writeGraph, CurrentVersion);
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public static void WriteXml(XmlWriter writer, Action writeGraph, string version)
50         {
51             writer.WriteStartDocument();
52             writer.WriteStartElement(ElementName, Namespace);
53             writer.WriteAttributeString(VersionAttributeName, version);
54             writeGraph();
55             writer.WriteEndElement();
56             writer.WriteEndDocument();
57         }
58
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges) =>
61             ↪ WriteXml(writer, writeNodes, writeEdges, CurrentVersion, GraphMode.Static,
62             ↪ GraphDefaultEdgeType.Directed);
63
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
66             ↪ string version) => WriteXml(writer, writeNodes, writeEdges, version,
67             ↪ GraphMode.Static, GraphDefaultEdgeType.Directed);
68
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
71             ↪ string version, GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, version,
72             ↪ mode, GraphDefaultEdgeType.Directed);
73
74         [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     }
76 }
```

```

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 }

```

```

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) =>
85         ↪ WriteXml(writer, writeNodes, writeEdges, GraphMode.Static,
86         ↪ GraphDefaultEdgeType.Directed);
87
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
90         ↪ GraphMode mode) => WriteXml(writer, writeNodes, writeEdges, mode,
91         ↪ GraphDefaultEdgeType.Directed);
92
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public static void WriteXml(XmlWriter writer, Action writeNodes, Action writeEdges,
95         ↪ GraphMode mode, GraphDefaultEdgeType defaultEdgeType)
96     {
97         writer.WriteStartElement(ElementName);
98         writer.WriteAttributeString(ModeAttributeName, mode.ToString().ToLower());
99         writer.WriteAttributeString(DefaultEdgeTypeAttributeName,
100             ↪ defaultEdgeType.ToString().ToLower());
101         writer.WriteStartElement(NodesElementName);
102         writeNodes();
103         writer.WriteEndElement();
104         writer.WriteStartElement(EdgesElementName);
105         writeEdges();
106         writer.WriteEndElement();
107         writer.WriteEndElement();
108     }
109 }

```

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

```

```

7
8 namespace Platform.Communication.Protocol.Gexf
9 {
10     public class Node
11     {
12         public static readonly string ElementName = "node";
13         public const string IdAttributeName = "id";
14         public const string LabelAttributeName = "label";
15
16         [XmlAttribute(AttributeName = IdAttributeName)]
17         public long Id
18         {
19             [MethodImpl(MethodImplOptions.AggressiveInlining)]
20             get;
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             set;
23         }
24
25         [XmlAttribute(AttributeName = LabelAttributeName)]
26         public string Label
27         {
28             [MethodImpl(MethodImplOptions.AggressiveInlining)]
29             get;
30             [MethodImpl(MethodImplOptions.AggressiveInlining)]
31             set;
32         }
33
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public void WriteXml(XmlWriter writer) => WriteXml(writer, Id, Label);
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public static void WriteXml(XmlWriter writer, long id, string label)
39         {
40             // <node id="0" label="..." />
41             writer.WriteStartElement(ElementName);
42             writer.WriteAttributeString(IdAttributeName,
43                 ↪ id.ToString(CultureInfo.InvariantCulture));
44             writer.WriteAttributeString(LabelAttributeName, label);
45             writer.WriteEndElement();
46         }
47     }

```

1.7 ./csharp/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         public static readonly Encoding DefaultEncoding = Encoding.UTF8;
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 }

```

1.8 ./csharp/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     /// <para>Represents the receiver of messages transfered via UDP protocol.</para>
17     /// <para>Представляет получателя сообщений по протоколу UDP.</para>
18     /// </summary>
19     public class UdpReceiver : DisposableBase //-V3073
20     {
21         private const int DefaultPort = 15000;
22
23         private bool _receiverRunning;
24         private Thread _thread;
25         private readonly UdpClient _udp;
26         private readonly MessageHandlerCallback _messageHandler;
27
28         public bool Available
29         {
30             [MethodImpl(MethodImplOptions.AggressiveInlining)]
31             get => _udp.Available > 0;
32         }
33
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public UdpReceiver(int listenPort, bool autoStart, MessageHandlerCallback messageHandler)
36         {
37             _udp = new UdpClient(listenPort);
38             _messageHandler = messageHandler;
39             if (autoStart)
40             {
41                 Start();
42             }
43         }
44
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public UdpReceiver(int listenPort, MessageHandlerCallback messageHandler) :
47             → this(listenPort, true, messageHandler) { }
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public UdpReceiver(MessageHandlerCallback messageHandler) : this(DefaultPort, true,
51             → messageHandler) { }
52
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         public UdpReceiver() : this(DefaultPort, true, message => { }) { }
55
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public void Start()
58         {
59             if (!_receiverRunning && _thread == null)
60             {
61                 _receiverRunning = true;
62                 _thread = new Thread(Receiver);
63                 _thread.Start();
64             }
65         }
66
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public void Stop()
69         {
70             if (_receiverRunning && _thread != null)
71             {
72                 _receiverRunning = false;
73                 _thread.Join();
74                 _thread = null;
75             }
76         }
77
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         public string Receive() => _udp.ReceiveString();
80
81         [MethodImpl(MethodImplOptions.AggressiveInlining)]
82         public void ReceiveAndHandle() => _messageHandler(Receive());
83
84         /// <remarks>
85         /// <para>The method receives messages and runs in a separate thread.</para>

```

```

84     /// <para>Метод получает сообщения и работает в отдельном потоке.</para>
85     /// </remarks>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     private void Receiver()
88     {
89         while (_receiverRunning)
90         {
91             try
92             {
93                 if (Available)
94                 {
95                     ReceiveAndHandle();
96                 }
97                 else
98                 {
99                     ThreadHelpers.Sleep();
100                 }
101             }
102             catch (Exception exception)
103             {
104                 exception.Ignore();
105             }
106         }
107     }
108
109     [MethodImpl(MethodImplOptions.AggressiveInlining)]
110     protected override void Dispose(bool manual, bool wasDisposed)
111     {
112         if (!wasDisposed)
113         {
114             Stop();
115             _udp.DisposeIfPossible();
116         }
117     }
118 }
119 }

```

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

```

```

41     }
42 }
43 }

```

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

```

1  using System.IO;
2  using System.Runtime.CompilerServices;
3  using System.Text;
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.Xml
9  {
10     public static class Serializer<T>
11     {
12         public static readonly XmlSerializer Instance = new XmlSerializer(typeof(T));
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public static T FromFile(string path)
16         {
17             using var stream = File.OpenRead(path);
18             return (T)Instance.Deserialize(stream);
19         }
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public static T FromString(string xml)
23         {
24             using var reader = new StringReader(xml);
25             return (T)Instance.Deserialize(reader);
26         }
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public static void ToFile(T @object, string path)
30         {
31             using var stream = File.OpenWrite(path);
32             Instance.Serialize(stream, @object);
33         }
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public static string ToString(T @object)
37         {
38             var sb = new StringBuilder();
39             using (var writer = new StringWriter(sb))
40             {
41                 Instance.Serialize(writer, @object);
42             }
43             return sb.ToString();
44         }
45     }
46 }

```

1.11 ./csharp/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         }
28     }
29 }

```



```

24         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\" />");
25     }
26 }
27 }

```

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

```

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

```

Index

- ./csharp/Platform.Communication.Tests/SerializerTests.cs, 8
- ./csharp/Platform.Communication.Tests/UdpReceiverTests.cs, 9
- ./csharp/Platform.Communication/Protocol/Gexf/Edge.cs, 1
- ./csharp/Platform.Communication/Protocol/Gexf/Gexf.cs, 2
- ./csharp/Platform.Communication/Protocol/Gexf/Graph.cs, 3
- ./csharp/Platform.Communication/Protocol/Gexf/GraphDefaultEdgeType.cs, 4
- ./csharp/Platform.Communication/Protocol/Gexf/GraphMode.cs, 4
- ./csharp/Platform.Communication/Protocol/Gexf/Node.cs, 4
- ./csharp/Platform.Communication/Protocol/Udp/UdpClientExtensions.cs, 5
- ./csharp/Platform.Communication/Protocol/Udp/UdpReceiver.cs, 5
- ./csharp/Platform.Communication/Protocol/Udp/UdpSender.cs, 7
- ./csharp/Platform.Communication/Protocol/Xml/Serializer.cs, 8