# CAPÍTULO IV

**PRUEBAS**

## INTRODUCCIÓN

En este capítulo primeramente se realizarán pruebas unitarias para la comprobación de distintos componentes del programa tales como clases y métodos, las cuáles estarán documentadas mediante una estructura de tablas donde se incluye la llamada, la descripción, la entrada, la referencia, el código y la salida.

A continuación se describirá el ambiente de pruebas, el cual tendrá como base a 4 computadores comunicándose entre si por medio del protocolo RTPS y en contraste se realizará la misma prueba con el protocolo RT-CORBA o Ada-DSA o DRTSJ. Además se adjunta capturas de pantalla tanto de la aplicación utilizando los protocolos y de capturas del flujo de datos con la herramienta wireshark, y se presenta un manual de usuario de las aplicaciones y del protocolo RTPS.

Finalmente, se realizará una comparación midiendo tiempos de respuesta y eficiencia del protocolo dentro de nuestro ambiente de pruebas.

## Pruebas unitarias del api rtps

### Codificadores

#### Prueba de los Elementos de los Mensajes.

|  |  |
| --- | --- |
| **Llamada:**  public static DataEncapsulation Serialize<T>(T obj, Encapsulation scheme = Encapsulation.CDR\_BE)  public static DataEncapsulation Deserialize(IoBuffer buffer, int length) | |
| ***Descripción*** | En esta prueba se verifica que el fichero de configuración no sea nulo |
| ***Entrada*** | Inicialmente se tiene el fichero de configuración |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestLocatorIpV4CDR\_BE()  {  Encapsulation Scheme = Encapsulation.CDR\_BE;  int bufferSize = 16 + 4 + 4 + CDRHeaderSize;  Locator v1 = new Locator(IPAddress.Parse("10.20.30.40"), 2700);  SerializedPayload payload = new SerializedPayload();  payload.DataEncapsulation = EncapsulationManager.Serialize<Locator>(v1, Scheme);  IoBuffer buffer = IoBuffer.Wrap(payload.DataEncapsulation.SerializedPayload);  Assert.AreEqual(bufferSize, buffer.Remaining);  Assert.AreEqual("00 00 00 00 00 00 00 01 00 00 0A 8C 00 00 00 00 00 00 00 00 00 00 00 00 0A 14 1E 28", buffer.GetHexDump());  Locator v2 = EncapsulationManager.Deserialize<Locator>(buffer);  Assert.AreEqual(v1, v2);  } |
| ***Salida*** | Nombre de la prueba: TestExistConfiguration  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,0286387 |

#### Prueba de Mensajes

### Transporte

#### Prueba de Detección de paquetes RTPS.

|  |  |
| --- | --- |
| **Llamada:**  public UDPReceiver(Uri uri, int bufferSize) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento de los receptores UDP, utilizando mensajes RTPS, a los cuales se verifica que sus datos sean correctos con pequeñas pruebas assert |
| ***Entrada*** | Inicialmente no se tiene inicializado al *Receiver UDP* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestPublishData()  {  object key = new object();  UDPReceiver rec = new UDPReceiver(new Uri("udp://" + Host + ":" + Port), 1024);  rec.MessageReceived += (s, m) =>  {  Message msg = m.Message;  Debug.WriteLine("New Message has arrived from {0}", m.Session.RemoteEndPoint);  Debug.WriteLine("Message Header: {0}", msg.Header);  Assert.AreEqual(ProtocolId.PROTOCOL\_RTPS, msg.Header.Protocol);  Assert.AreEqual(VendorId.OCI, msg.Header.VendorId);  Assert.AreEqual(ProtocolVersion.PROTOCOLVERSION\_2\_1, msg.Header.Version);  Assert.AreEqual(2, msg.SubMessages.Count);  foreach (var submsg in msg.SubMessages)  {  Debug.WriteLine("SubMessage: {0}", submsg);  if (submsg is Data)  {  Data d = submsg as Data;  foreach (var par in d.InlineQos.Value)  Debug.WriteLine("InlineQos: {0}", par);  }  }  lock (key) Monitor.Pulse(key);  };  rec.Start();  simulator.SendUDPPacket("SamplePackets/packet1.dat", Host, Port);  lock (key)  {  Assert.IsTrue(Monitor.Wait(key, 1000), "Time-out. Message has not arrived or there is an error on it.");  }  rec.Close();  }  } |
| ***Salida*** | Nombre de la prueba: *TestPublishData*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,640815  Salida estándar de Result:  Trace du débogage :  no configuration section <common/logging> found - suppressing logging output  Sent 613/613 bytes to 224.0.1.111:9999  New Message has arrived from 172.30.82.26:56951  Message Header: [RTPS, 2.1, 01-03, 01-03-00-00-01-23-45-67-89-AB-CD-EF]  SubMessage: InfoTimestamp:header[9, 1, 8], 12/12/1944 16:04:37 [1418400277:3853715240]  SubMessage: Data:header[21, 7, 577], Payload[Rtps.Messages.Submessages.Elements.SerializedPayload]  InlineQos: ParameterId=PID\_KEY\_HASH, Content=09-23-09-23-00-00-00-00-00-00-00-00-00-00-00-00  InlineQos: ParameterId=PID\_SENTINEL, Content= |

#### Pruebas de paquetes RTPS.

|  |  |
| --- | --- |
| **Llamada:**  public UDPReceiver(Uri uri, int bufferSize) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento de los receptores UDP, utilizando mensajes RTPS de otros vendors con los cuales se verifica que sus datos sean correctos con pequeñas pruebas assert |
| ***Entrada*** | Inicialmente no se tiene inicializado al *Receiver UDP* |
| ***Código*** | [TestMethod]  public void TestPublishPacket2()  {  object key = new object();  UDPReceiver rec = new UDPReceiver(new Uri("udp://" + Host + ":" + Port), 1024);  rec.MessageReceived += (s, m) =>  {  Message msg = m.Message;  Debug.WriteLine("New Message has arrived from {0}", m.Session.RemoteEndPoint);  Debug.WriteLine("Message Header: {0}", msg.Header);  Assert.AreEqual(ProtocolId.PROTOCOL\_RTPS, msg.Header.Protocol);  Assert.AreEqual(VendorId.OCI, msg.Header.VendorId);  Assert.AreEqual(ProtocolVersion.PROTOCOLVERSION\_2\_1, msg.Header.Version);  Assert.AreEqual(2, msg.SubMessages.Count);  foreach (var submsg in msg.SubMessages)  {  Debug.WriteLine("SubMessage: {0}", submsg);  switch (submsg.Kind)  {  case SubMessageKind.DATA:  Data d = submsg as Data;  foreach (var par in d.InlineQos.Value)  Debug.WriteLine("InlineQos: {0}", par);  break;  case SubMessageKind.INFO\_TS:  InfoTimestamp its = submsg as InfoTimestamp;  Debug.WriteLine("The TimeStampFlag value state is: {0}", its.HasInvalidateFlag);  Debug.WriteLine("The EndiannessFlag value state is: {0}", its.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", its.Header.SubMessageLength);  if (its.HasInvalidateFlag == false)  {  Debug.WriteLine("The Timestamp value is: {0}", its.TimeStamp);  }  break;  default:  Assert.Fail("Only Timestamp and Data submesages are expected");  break;  }  }  lock (key) Monitor.Pulse(key);  };  rec.Start();  simulator.SendUDPPacket("SamplePackets/packet3.dat", Host, Port);  lock (key)  {  Assert.IsTrue(Monitor.Wait(key, 10000), "Time-out. Message has not arrived or there is an error on it.");  }  rec.Close();  } |
| ***Salida*** | Nombre de la prueba: *TestPublishPacket2*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,2431361  Salida estándar de Result:  Trace du débogage :  no configuration section <common/logging> found - suppressing logging output  Sent 196/196 bytes to 224.0.1.111:7400  New Message has arrived from 172.30.82.26:63179  Message Header: [RTPS, 2.1, 01-03, 01-03-00-00-01-23-45-67-89-AB-CD-EF]  SubMessage: InfoTimestamp:header[9, 1, 8], 01/01/1900 00:00:00 [0:0]  The TimeStampFlag value state is: False  The EndiannessFlag value state is: True  The octetsToNextHeader value is: 8  The Timestamp value is: 01/01/1900 00:00:00 [0:0]  SubMessage: Data:header[21, 11, 0], Payload[Rtps.Messages.Submessages.Elements.SerializedPayload]  InlineQos: ParameterId=PID\_STATUS\_INFO, Content=00-00-00-01  InlineQos: ParameterId=PID\_TOPIC\_NAME, Content=0A-00-00-00-4D-79-20-54-6F-70-69-63-20-00-00-00  InlineQos: ParameterId=PID\_PRESENTATION, Content=E7-03-00-00-00-00-00-00  InlineQos: ParameterId=PID\_PARTITION, Content=01-00-00-00-06-00-00-00-48-65-6C-6C-6F-00-00-00  InlineQos: ParameterId=PID\_OWNERSHIP\_STRENGTH, Content=0C-00-00-00  InlineQos: ParameterId=PID\_LIVELINESS, Content=02-00-00-00-FF-FF-FF-7F-FF-FF-FF-7F  InlineQos: ParameterId=PID\_RELIABILITY, Content=00-00-00-00-00-00-00-00-00-E1-F5-05  InlineQos: ParameterId=PID\_TRANSPORT\_PRIORITY, Content=0D-00-00-00  InlineQos: ParameterId=PID\_LIFESPAN, Content=0E-00-00-00-FF-FF-FF-7F  InlineQos: ParameterId=PID\_DESTINATION\_ORDER, Content=01-00-00-00  InlineQos: ParameterId=PID\_SENTINEL, Content= |

|  |  |
| --- | --- |
| **Llamada:**  public UDPReceiver(Uri uri, int bufferSize) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento de los receptores UDP, utilizando mensajes RTPS de otros vendors con los cuales se verifica que sus datos sean correctos con pequeñas pruebas assert. De esta prueba se pueden derivar otras. |
| ***Entrada*** | Inicialmente no se tiene inicializado al *Receiver UDP* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void GeneralRTPSMessageTesterMethod()  {  object key = new object();  UDPReceiver rec = new UDPReceiver(new Uri("udp://" + Host + ":" + Port), 1024);  rec.MessageReceived += (s, m) =>  {  Message msg = m.Message;  Debug.WriteLine("New Message has arrived from {0}", m.Session.RemoteEndPoint);  Debug.WriteLine("Message Header: {0}", msg.Header);  Assert.AreEqual(ProtocolId.PROTOCOL\_RTPS, msg.Header.Protocol);  Debug.WriteLine("The Header Protocol is: {0}", msg.Header.Protocol);  Assert.AreEqual(VendorId.OCI, msg.Header.VendorId);  Debug.WriteLine("The VendorId value state is: {0}", msg.Header.VendorId);  Assert.AreEqual(ProtocolVersion.PROTOCOLVERSION\_2\_1, msg.Header.Version);  Debug.WriteLine("The Protocol Version value state is: {0}", msg.Header.Version);  Debug.WriteLine("The number of SubMessages in the message is: {0}", msg.SubMessages.Count);  //Assert.AreEqual(2, msg.SubMessages.Count);  foreach (var submsg in msg.SubMessages)  {  Debug.WriteLine("SubMessage: {0}", submsg.Kind);  switch (submsg.Kind)  {  case SubMessageKind.DATA:  {  Data d = submsg as Data;    Debug.WriteLine("The KeyFlag value state is: {0}", d.HasKeyFlag);  Debug.WriteLine("The DataFlag value state is: {0}", d.HasDataFlag);  Debug.WriteLine("The InlineQoSFlag value state is: {0}", d.HasInlineQosFlag);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The extraFlags value is: {0}", d.ExtraFlags.Value);  Debug.WriteLine("The octetsToInlineQos value is: Aun no logro");  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The writerSN is: {0}", d.WriterSN);  if (d.HasInlineQosFlag)  {  foreach (var par in d.InlineQos.Value)  {  Debug.WriteLine("InlineQos: {0}", par);  }  }  if (d.HasDataFlag || d.Header.Flags.IsLittleEndian)  {  for (int i = 0; i <= d.SerializedPayload.DataEncapsulation.SerializedPayload.Length - 1; i++)  {  Debug.WriteLine("SerializedPayload: {0}", d.SerializedPayload.DataEncapsulation.SerializedPayload.GetValue(i));  }  }  break;  }  case SubMessageKind.ACKNACK:  {  AckNack d = submsg as AckNack;  Debug.WriteLine("The FinalFlag value state is: {0}", d.HasFinalFlag);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The readerSNState is: {0}", d.ReaderSNState);  Debug.WriteLine("The Count is: {0}", d.Count);  break;  }  case SubMessageKind.NACK\_FRAG:  {  NackFrag d = submsg as NackFrag;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The writerSN is: {0}", d.WriterSequenceNumber);  Debug.WriteLine("The fragmentNumberState value is: {0}", d.FragmentNumberState);  break;  }  case SubMessageKind.DATA\_FRAG:  {  DataFrag d = submsg as DataFrag;  Debug.WriteLine("The KeyFlag value state is: {0}", d.HasKeyFlag);  Debug.WriteLine("The InlineQoSFlag value state is: {0}", d.HasInlineQosFlag);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The extraFlags value is: {0}", d.ExtraFlags);  Debug.WriteLine("The octetsToInlineQos value is: Aun no logro");  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The writerSN is: {0}", d.WriterSequenceNumber);  Debug.WriteLine("The FragmentNumber is: {0}", d.FragmentStartingNumber);  Debug.WriteLine("The fragmentsInSubmessage is: {0}", d.FragmentsInSubmessage);  Debug.WriteLine("The samplesize is: {0}", d.SampleSize);  if (d.HasInlineQosFlag)  {  foreach (var par in d.ParameterList.Value)  {  Debug.WriteLine("InlineQos: {0}", par);  }  }  for (int i = 0; i <= d.SerializedPayload.Length - 1; i++)  {  Debug.WriteLine("SerializedPayload: {0}", d.SerializedPayload.GetValue(i));  }  break;  }  case SubMessageKind.GAP:  {  Gap d = submsg as Gap;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The GapStart number is: {0}", d.GapStart);  Debug.WriteLine("The GapList value is: {0}", d.GapList);  break;  }  case SubMessageKind.HEARTBEAT:  {  Heartbeat d = submsg as Heartbeat;  Debug.WriteLine("The LivelinessFlag value state is: {0}", d.HasLivelinessFlag);  Debug.WriteLine("The FinalFlag value state is: {0}", d.HasFinalFlag);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The firstSN is: {0}", d.FirstSequenceNumber);  Debug.WriteLine("The lastSN is: {0}", d.LastSequenceNumber);  Debug.WriteLine("The Count is: {0}", d.Count);  break;  }  case SubMessageKind.HEARTBEAT\_FRAG:  {  HeartbeatFrag d = submsg as HeartbeatFrag;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The readerID is: {0}", d.ReaderId);  Debug.WriteLine("The writerID is: {0}", d.WriterId);  Debug.WriteLine("The writerSN is: {0}", d.WriterSequenceNumber);  Debug.WriteLine("The FragmentNumber is: {0}", d.LastFragmentNumber);  Debug.WriteLine("The Count is: {0}", d.Count);  break;  }  case SubMessageKind.INFO\_DST:  {  InfoDestination d = submsg as InfoDestination;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The guidPrefix value is: {0}", d.GuidPrefix);  break;  }  case SubMessageKind.INFO\_TS:  {  InfoTimestamp d = submsg as InfoTimestamp;  Debug.WriteLine("The TimeStampFlag value state is: {0}", d.HasInvalidateFlag);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  if (d.HasInvalidateFlag == false)  {  Debug.WriteLine("The Timestamp value is: {0}", d.TimeStamp);  }  break;  }  case SubMessageKind.INFO\_SRC:  {  InfoSource d = submsg as InfoSource;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The ProtocolVersion value is: {0}", d.ProtocolVersion);  Debug.WriteLine("The vendorId value is: {0}", d.VendorId);  Debug.WriteLine("The guidPrefix value is: {0}", d.GuidPrefix);  break;  }  case SubMessageKind.INFO\_REPLY:  {  InfoReply d = submsg as InfoReply;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The MulticastFlag value state is: {0}", d.HasMulticastFlag);  Debug.WriteLine("The unicastLocatorList value state is: {0}", d.UnicastLocatorList);  if (d.HasMulticastFlag)  {  Debug.WriteLine("The multicastLocatorList value state is: {0}", d.MulticastLocatorList);  }  break;  }  case SubMessageKind.INFO\_REPLY\_IP4:  {  InfoReplyIp4 d = submsg as InfoReplyIp4;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Debug.WriteLine("The MulticastFlag value state is: {0}", d.HasMulticastFlag);  Debug.WriteLine("The unicastLocatorList value state is: {0}", d.UnicastLocator);  if (d.HasMulticastFlag)  {  Debug.WriteLine("The multicastLocatorList value state is: {0}", d.MulticastLocator);  }  break;  }  case SubMessageKind.PAD:  {  Pad d = submsg as Pad;  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  break;  }  }  }  lock (key) Monitor.Pulse(key);  };  rec.Start();  simulator.SendUDPPacket("SamplePackets/TestOpenDDS\_rtps\_reliability\_runtest\_local/Packet04.dat", Host, Port);  lock (key)  {  Assert.IsTrue(Monitor.Wait(key, 10000), "Time-out. Message has not arrived or there is an error on it.");  }  rec.Close();  } |
| ***Salida*** | Nombre de la prueba: *GeneralRTPSMessageTesterMethod*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,2446508  Nombre de la prueba: GeneralRTPSMessageTesterMethod  Resultado de la prueba: Superada  Salida estándar de Result:  Trace du débogage :  no configuration section <common/logging> found - suppressing logging output  Sent 52/52 bytes to 224.0.1.111:7400  New Message has arrived from 172.30.82.26:54070  Message Header: [RTPS, 2.1, 01-03, 01-03-08-00-27-B9-29-47-0A-AF-00-00]  The Header Protocol is: RTPS  The VendorId value state is: 01-03  The Protocol Version value state is: 2.1  The number of SubMessages in the message is: 1  SubMessage: DATA  The KeyFlag value state is: False  The DataFlag value state is: True  The InlineQoSFlag value state is: False  The EndiannessFlag value state is: True  The octetsToNextHeader value is: 0  The extraFlags value is: 0  The octetsToInlineQos value is: Aun no logro  The readerID is: 0-USER\_DEFINED\_UNKNOWN  The writerID is: 3-USER\_DEFINED\_WRITER\_W\_KEY  The writerSN is: 3  SerializedPayload: 205  SerializedPayload: 171  SerializedPayload: 205  SerializedPayload: 171 |

|  |  |
| --- | --- |
| **Llamada:**  public UDPReceiver(Uri uri, int bufferSize) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento de los receptores UDP, utilizando mensajes RTPS de otros vendors con los cuales se verifica que sus datos sean correctos con pequeñas pruebas assert. |
| ***Entrada*** | Inicialmente no se tiene inicializado al *Receiver UDP* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TesOpenDDS\_rtps\_reliability\_runtest\_localPacket01()  {  object key = new object();  UDPReceiver rec = new UDPReceiver(new Uri("udp://" + Host + ":" + Port), 1024);  rec.MessageReceived += (s, m) =>  {  Message msg = m.Message;  Debug.WriteLine("New Message has arrived from {0}", m.Session.RemoteEndPoint);    Debug.WriteLine("Message Header: {0}", msg.Header);  Assert.AreEqual(ProtocolId.PROTOCOL\_RTPS.ToString(), msg.Header.Protocol.ToString());  Debug.WriteLine("The Header Protocol is: {0}", msg.Header.Protocol);  Assert.AreEqual(ProtocolVersion.PROTOCOLVERSION\_2\_1.ToString(), msg.Header.Version.ToString());  Debug.WriteLine("The Protocol Version value state is: {0}", msg.Header.Version);  Assert.AreEqual(VendorId.OCI.ToString(), msg.Header.VendorId.ToString());  Debug.WriteLine("The VendorId value state is: {0}", msg.Header.VendorId);  Assert.AreEqual("01-03-08-00-27-B9-29-47-0A-AF-00-00", msg.Header.GuidPrefix.ToString());  Debug.WriteLine("The guidPrefix value state is: {0}", msg.Header.GuidPrefix);  Assert.AreEqual(1, msg.SubMessages.Count);  Debug.WriteLine("The number of SubMessages in the message is: {0}", msg.SubMessages.Count);    foreach (var submsg in msg.SubMessages)  {  Assert.AreEqual(SubMessageKind.DATA, submsg.Kind );  Debug.WriteLine("SubMessage: {0}", submsg.Kind);    switch (submsg.Kind)  {  case SubMessageKind.DATA:  {  Data d = submsg as Data;    Assert.AreEqual(false, d.HasKeyFlag);  Debug.WriteLine("The KeyFlag value state is: {0}", d.HasKeyFlag);  Assert.AreEqual(true, d.HasDataFlag);  Debug.WriteLine("The DataFlag value state is: {0}", d.HasDataFlag);  Assert.AreEqual(false, d.HasInlineQosFlag);  Debug.WriteLine("The InlineQoSFlag value state is: {0}", d.HasInlineQosFlag);  Assert.AreEqual(true, d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Assert.AreEqual(0, d.Header.SubMessageLength);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Assert.AreEqual(0, d.ExtraFlags .Value);  Debug.WriteLine("The extraFlags value is: {0}", d.ExtraFlags.Value);  Debug.WriteLine("The octetsToInlineQos value is: ");  Assert.AreEqual(0, d.ReaderId.EntityKey0);  Assert.AreEqual(0, d.ReaderId.EntityKey1);  Assert.AreEqual(0, d.ReaderId.EntityKey2);  Debug.WriteLine("The readerIDEntityKey is: {0}-{1}-{2}", d.ReaderId.EntityKey0,d.ReaderId.EntityKey1,d.ReaderId.EntityKey2);  Assert.AreEqual(0,(int) d.ReaderId.TypeID);  Debug.WriteLine("The readerIDEntityKind value is: {0} ",(int)d.ReaderId.TypeID);  Assert.AreEqual(0, d.WriterId.EntityKey0);  Assert.AreEqual(1, d.WriterId.EntityKey1);  Assert.AreEqual(2, d.WriterId.EntityKey2);  Debug.WriteLine("The writerID is: {0}-{1}-{2}", d.WriterId.EntityKey0, d.WriterId.EntityKey1, d.WriterId.EntityKey2);  Assert.AreEqual(2, (int)d.WriterId.TypeID);  Debug.WriteLine("The writerIDEntityKind value is:{0} ",(int) d.WriterId.TypeID);  Assert.AreEqual("1", d.WriterSN.ToString());    Debug.WriteLine("The writerSN is: {0}", d.WriterSN);  if (d.HasInlineQosFlag)  {  /\*foreach (var par in d.InlineQos.Value)  {  Debug.WriteLine("InlineQos: {0}", par);  }\*/  }      if (d.HasDataFlag || d.Header.Flags.IsLittleEndian)  {  for (int i = 0; i <= d.SerializedPayload.DataEncapsulation.SerializedPayload.Length - 1; i++)  {    Debug.WriteLine("SerializedPayload: {0}", d.SerializedPayload.DataEncapsulation.SerializedPayload.GetValue(i));  }  }  break;  }  }  }  lock (key) Monitor.Pulse(key);  };  rec.Start();  simulator.SendUDPPacket("SamplePackets/TestOpenDDS\_rtps\_reliability\_runtest\_local/Packet01.dat", Host, Port);  lock (key)  {  Assert.IsTrue(Monitor.Wait(key, 10000), "Time-out. Message has not arrived or there is an error on it.");  }  rec.Close();  } |
| ***Salida*** | Nombre de la prueba: *TestOpenDDS\_rtps\_reliability\_runtest\_localPacket01*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,2312824  Salida estándar de Result:  Trace du débogage :  no configuration section <common/logging> found - suppressing logging output  Sent 52/52 bytes to 224.0.1.111:7400  New Message has arrived from 172.30.82.26:63701  Message Header: [RTPS, 2.1, 01-03, 01-03-08-00-27-B9-29-47-0A-AF-00-00]  The Header Protocol is: RTPS  The Protocol Version value state is: 2.1  The VendorId value state is: 01-03  The guidPrefix value state is: 01-03-08-00-27-B9-29-47-0A-AF-00-00  The number of SubMessages in the message is: 1  SubMessage: DATA  The KeyFlag value state is: False  The DataFlag value state is: True  The InlineQoSFlag value state is: False  The EndiannessFlag value state is: True  The octetsToNextHeader value is: 0  The extraFlags value is: 0  The octetsToInlineQos value is:  The readerIDEntityKey is: 0-0-0  The readerIDEntityKind value is: 0  The writerID is: 0-1-2  The writerIDEntityKind value is:2  The writerSN is: 1  SerializedPayload: 205  SerializedPayload: 171  SerializedPayload: 205  SerializedPayload: 171 |

|  |  |
| --- | --- |
| **Llamada:**  public UDPReceiver(Uri uri, int bufferSize) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento de los receptores UDP, utilizando mensajes RTPS de otros vendors con los cuales se verifica que sus datos sean correctos con pequeñas pruebas assert. |
| ***Entrada*** | Inicialmente no se tiene inicializado al *Receiver UDP* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TesOpenDDS\_rtps\_reliability\_runtest\_localPacket02()  {  object key = new object();  UDPReceiver rec = new UDPReceiver(new Uri("udp://" + Host + ":" + Port), 1024);  rec.MessageReceived += (s, m) =>  {  Message msg = m.Message;  Debug.WriteLine("New Message has arrived from {0}", m.Session.RemoteEndPoint);  Debug.WriteLine("Message Header: {0}", msg.Header);  Assert.AreEqual(ProtocolId.PROTOCOL\_RTPS.ToString(), msg.Header.Protocol.ToString());  Debug.WriteLine("The Header Protocol is: {0}", msg.Header.Protocol);  Assert.AreEqual(ProtocolVersion.PROTOCOLVERSION\_2\_1.ToString(), msg.Header.Version.ToString());  Debug.WriteLine("The Protocol Version value state is: {0}", msg.Header.Version);  Assert.AreEqual(VendorId.OCI.ToString(), msg.Header.VendorId.ToString());  Debug.WriteLine("The VendorId value state is: {0}", msg.Header.VendorId);  Assert.AreEqual("01-03-08-00-27-B9-29-47-0A-AF-00-00", msg.Header.GuidPrefix.ToString());  Debug.WriteLine("The guidPrefix value state is: {0}", msg.Header.GuidPrefix);  Assert.AreEqual(1, msg.SubMessages.Count);  Debug.WriteLine("The number of SubMessages in the message is: {0}", msg.SubMessages.Count);  foreach (var submsg in msg.SubMessages)  {  Assert.AreEqual(SubMessageKind.HEARTBEAT, submsg.Kind);  Debug.WriteLine("SubMessage: {0}", submsg.Kind);  switch (submsg.Kind)  {  case SubMessageKind.HEARTBEAT:  {  Heartbeat d = submsg as Heartbeat;  Assert.AreEqual(false, d.HasLivelinessFlag);  Debug.WriteLine("The LivelinessFlag value state is: {0}", d.HasLivelinessFlag);  Assert.AreEqual(false, d.HasFinalFlag);  Debug.WriteLine("The FinalFlag value state is: {0}", d.HasFinalFlag);  Assert.AreEqual(true, d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Assert.AreEqual(0, d.Header.SubMessageLength);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Assert.AreEqual(0, d.ReaderId.EntityKey0);  Assert.AreEqual(0, d.ReaderId.EntityKey1);  Assert.AreEqual(0, d.ReaderId.EntityKey2);  Debug.WriteLine("The readerIDEntityKey is: {0}-{1}-{2}", d.ReaderId.EntityKey0, d.ReaderId.EntityKey1, d.ReaderId.EntityKey2);  Assert.AreEqual(0, (int)d.ReaderId.TypeID);  Debug.WriteLine("The readerIDEntityKind value is: {0} ", (int)d.ReaderId.TypeID);  Assert.AreEqual(0, d.WriterId.EntityKey0);  Assert.AreEqual(1, d.WriterId.EntityKey1);  Assert.AreEqual(2, d.WriterId.EntityKey2);  Debug.WriteLine("The writerID is: {0}-{1}-{2}", d.WriterId.EntityKey0, d.WriterId.EntityKey1, d.WriterId.EntityKey2);  Assert.AreEqual(2, (int)d.WriterId.TypeID);    Debug.WriteLine("The writerIDEntityKind value is:{0} ", (int)d.WriterId.TypeID);  Assert.AreEqual(1,d.FirstSequenceNumber);  Debug.WriteLine("The firstSN is: {0}", d.FirstSequenceNumber);  Assert.AreEqual(1,d.LastSequenceNumber);  Debug.WriteLine("The lastSN is: {0}", d.LastSequenceNumber);  Assert.AreEqual(1,d.Count);  Debug.WriteLine("The Count is: {0}", d.Count);  break;  }    }  }  lock (key) Monitor.Pulse(key);  };  rec.Start();  simulator.SendUDPPacket("SamplePackets/TestOpenDDS\_rtps\_reliability\_runtest\_local/Packet02.dat", Host, Port);  lock (key)  {  Assert.IsTrue(Monitor.Wait(key, 10000), "Time-out. Message has not arrived or there is an error on it.");  }  rec.Close();  } |
| ***Salida*** | Nombre de la prueba: *TestOpenDDS\_rtps\_reliability\_runtest\_localPacket02*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,2411492  Nombre de la prueba: TesOpenDDS\_rtps\_reliability\_runtest\_localPacket02  Resultado de la prueba: Superada  Salida estándar de Result:  Trace du débogage :  no configuration section <common/logging> found - suppressing logging output  Sent 52/52 bytes to 224.0.1.111:7400  New Message has arrived from 172.30.82.26:59411  Message Header: [RTPS, 2.1, 01-03, 01-03-08-00-27-B9-29-47-0A-AF-00-00]  The Header Protocol is: RTPS  The Protocol Version value state is: 2.1  The VendorId value state is: 01-03  The guidPrefix value state is: 01-03-08-00-27-B9-29-47-0A-AF-00-00  The number of SubMessages in the message is: 1  SubMessage: HEARTBEAT  The LivelinessFlag value state is: False  The FinalFlag value state is: False  The EndiannessFlag value state is: True  The octetsToNextHeader value is: 0  The readerIDEntityKey is: 0-0-0  The readerIDEntityKind value is: 0  The writerID is: 0-1-2  The writerIDEntityKind value is:2  The firstSN is: 1  The lastSN is: 1  The Count is: 1 |

|  |  |
| --- | --- |
| **Llamada:**  public UDPReceiver(Uri uri, int bufferSize) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento de los receptores UDP, utilizando mensajes RTPS de otros vendors con los cuales se verifica que sus datos sean correctos con pequeñas pruebas assert. |
| ***Entrada*** | Inicialmente no se tiene inicializado al *Receiver UDP* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TesOpenDDS\_rtps\_reliability\_runtest\_localPacket03()  {  object key = new object();  UDPReceiver rec = new UDPReceiver(new Uri("udp://" + Host + ":" + Port), 1024);  rec.MessageReceived += (s, m) =>  {  Message msg = m.Message;  Debug.WriteLine("New Message has arrived from {0}", m.Session.RemoteEndPoint);  Debug.WriteLine("Message Header: {0}", msg.Header);  Assert.AreEqual(ProtocolId.PROTOCOL\_RTPS.ToString(), msg.Header.Protocol.ToString());  Debug.WriteLine("The Header Protocol is: {0}", msg.Header.Protocol);  Assert.AreEqual(ProtocolVersion.PROTOCOLVERSION\_2\_1.ToString(), msg.Header.Version.ToString());  Debug.WriteLine("The Protocol Version value state is: {0}", msg.Header.Version);  Assert.AreEqual(VendorId.OCI.ToString(), msg.Header.VendorId.ToString());  Debug.WriteLine("The VendorId value state is: {0}", msg.Header.VendorId);  Assert.AreEqual("01-03-08-00-27-B9-29-47-0A-AF-00-01", msg.Header.GuidPrefix.ToString());  Debug.WriteLine("The guidPrefix value state is: {0}", msg.Header.GuidPrefix);  Assert.AreEqual(2, msg.SubMessages.Count);  Debug.WriteLine("The number of SubMessages in the message is: {0}", msg.SubMessages.Count);  foreach (var submsg in msg.SubMessages)  {    Debug.WriteLine("SubMessage: {0}", submsg.Kind);  switch (submsg.Kind)  {  case SubMessageKind.INFO\_DST:  {  InfoDestination d = submsg as InfoDestination;  Assert.AreEqual(true, d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Assert.AreEqual(12, d.Header.SubMessageLength);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Assert.AreEqual("01-03-08-00-27-B9-29-47-0A-AF-00-00", d.GuidPrefix.ToString());  Debug.WriteLine("The guidPrefix value is: {0}", d.GuidPrefix);  break;  }  case SubMessageKind.ACKNACK:  {  AckNack d = submsg as AckNack;  Assert.AreEqual(true, d.HasFinalFlag);  Debug.WriteLine("The FinalFlag value state is: {0}", d.HasFinalFlag);  Assert.AreEqual(true, d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);  Debug.WriteLine("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);  Assert.AreEqual(0, d.ReaderId.EntityKey0);  Assert.AreEqual(1, d.ReaderId.EntityKey1);  Assert.AreEqual(5, d.ReaderId.EntityKey2);  Debug.WriteLine("The readerIDEntityKey is: {0}-{1}-{2}", d.ReaderId.EntityKey0, d.ReaderId.EntityKey1, d.ReaderId.EntityKey2);  Assert.AreEqual(7, (int)d.ReaderId.TypeID);  Debug.WriteLine("The readerIDEntityKind value is: {0} ", (int)d.ReaderId.TypeID);  Assert.AreEqual(0, d.WriterId.EntityKey0);  Assert.AreEqual(1, d.WriterId.EntityKey1);  Assert.AreEqual(2, d.WriterId.EntityKey2);  Debug.WriteLine("The writerID is: {0}-{1}-{2}", d.WriterId.EntityKey0, d.WriterId.EntityKey1, d.WriterId.EntityKey2);  Assert.AreEqual(2, (int)d.WriterId.TypeID);  Debug.WriteLine("The writerIDEntityKind value is:{0} ", (int)d.WriterId.TypeID);  Assert.AreEqual("2", d.ReaderSNState.BitmapBase.ToString());  Assert.AreEqual(1, d.ReaderSNState.NumBits);  Assert.AreEqual(0, d.ReaderSNState.Bitmaps[0]);    Debug.WriteLine("The readerSNState is: {0}", d.ReaderSNState);  Debug.WriteLine("The Count is: {0}", d.Count);  break;  }  }  }  lock (key) Monitor.Pulse(key);  };  rec.Start();  simulator.SendUDPPacket("SamplePackets/TestOpenDDS\_rtps\_reliability\_runtest\_local/Packet03.dat", Host, Port);  lock (key)  {  Assert.IsTrue(Monitor.Wait(key, 1000), "Time-out. Message has not arrived or there is an error on it.");  }  rec.Close();  }    } |
| ***Salida*** | Nombre de la prueba: *TestOpenDDS\_rtps\_reliability\_runtest\_localPacket03*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,2464427  Salida estándar de Result:  Trace du débogage :  no configuration section <common/logging> found - suppressing logging output  Sent 68/68 bytes to 224.0.1.111:7400  New Message has arrived from 172.30.82.26:63366  Message Header: [RTPS, 2.1, 01-03, 01-03-08-00-27-B9-29-47-0A-AF-00-01]  The Header Protocol is: RTPS  The Protocol Version value state is: 2.1  The VendorId value state is: 01-03  The guidPrefix value state is: 01-03-08-00-27-B9-29-47-0A-AF-00-01  The number of SubMessages in the message is: 2  SubMessage: INFO\_DST  The EndiannessFlag value state is: True  The octetsToNextHeader value is: 12  The guidPrefix value is: 01-03-08-00-27-B9-29-47-0A-AF-00-00  SubMessage: ACKNACK  The FinalFlag value state is: True  The EndiannessFlag value state is: True  The octetsToNextHeader value is: 28  The readerIDEntityKey is: 0-1-5  The readerIDEntityKind value is: 7  The writerID is: 0-1-2  The writerIDEntityKind value is:2  The readerSNState is: 2/1:[0x0000]  The Count is: 1 |

### Utils

#### Pruebas del generador de identidad.

|  |  |
| --- | --- |
| **Llamada:**  static GuidGenerator() | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento del Guid Generator. |
| ***Entrada*** | Inicialmente no se tiene inicializado al *GuidGenerator* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestGeneration1()  {  GuidGenerator generator = new GuidGenerator();  GUID guid = generator.GenerateGuid();  Assert.AreEqual(GuidGenerator.VENDORID\_DOOPEC[0], guid.Prefix.Prefix[0]);  Assert.AreEqual(GuidGenerator.VENDORID\_DOOPEC[1], guid.Prefix.Prefix[1]);  } |
| ***Salida*** | Nombre de la prueba: *TestGeneration1*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,172692 |

|  |  |
| --- | --- |
| **Llamada:**  static GuidGenerator() | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento del Guid Generator. |
| ***Entrada*** | Inicialmente no se tiene inicializado al *GuidGenerator* |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestGeneration2()  {  GuidGenerator generator = new GuidGenerator();  GUID guid1 = generator.GenerateGuid();  GUID guid2 = generator.GenerateGuid();  Assert.AreNotEqual(guid2.Prefix.ToString(), guid1.Prefix.ToString());  } |
| ***Salida*** | Nombre de la prueba: *TestGeneration2*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,1552052 |

#### Pruebas del PeriodicWorker.

|  |  |
| --- | --- |
| **Llamada:**  private void KeepWorkerRunning() | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento del Worker en el cual se realiza tareas de actualización y descubrimiento |
| ***Entrada*** | Inicialmente no se tiene inicializado al Worker |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestWorkerVerySlow()  {  int period = 2 \* 1000;  int sleepTime = 20 \* 1000+90;  PeriodicWorker worker = new PeriodicWorker();  worker.Start(period);  Thread.Sleep(sleepTime);  worker.End();  Assert.AreEqual(sleepTime / period, worker.Count);  } |
| ***Salida*** | Nombre de la prueba: *TestWorkerVerySlow*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:20,2217009 |

|  |  |
| --- | --- |
| **Llamada:**  private void KeepWorkerRunning() | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento del Worker en el cual se realiza tareas de actualización y descubrimiento |
| ***Entrada*** | Inicialmente no se tiene inicializado al Worker |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestWorkerSlow()  {  int period = 2 \* 100;  int sleepTime = 20 \* 100 + 50;  PeriodicWorker worker = new PeriodicWorker();  worker.Start(period);  Thread.Sleep(sleepTime);  worker.End();  Assert.AreEqual(sleepTime / period, worker.Count);  } |
| ***Salida*** | Nombre de la prueba: *TestWorkerSlow*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:02,1763206 |

|  |  |
| --- | --- |
| **Llamada:**  private void KeepWorkerRunning() | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento del Worker en el cual se realiza tareas de actualización y descubrimiento |
| ***Entrada*** | Inicialmente no se tiene inicializado al Worker |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestWorkerQuick()  {  int period = 2 \* 10;  int sleepTime = 20 \* 10 + 50;  PeriodicWorker worker = new PeriodicWorker();  worker.Start(period);  Thread.Sleep(sleepTime);  worker.End();  Assert.AreEqual(sleepTime / period, worker.Count);  } |
| ***Salida*** | Nombre de la prueba: *TestWorkerQuick*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,3822312 |

#### Prueba de tiempo

|  |  |
| --- | --- |
| **Llamada:**  public Time(long systemCurrentMillis) | |
| ***Descripción*** | En esta prueba se verifica el correcto funcionamiento del temporizador |
| ***Entrada*** | Inicialmente no se tiene inicializado al Time |
| ***Referencia*** |  |
| ***Código*** | [TestMethod]  public void TestTimeSeconds()  {  long timeMillis = 1000; // 1 sec  Time t = new Time(timeMillis);  long timeConverted = t.TimeMillis;  Assert.AreEqual(timeMillis, timeConverted);  } |
| ***Salida*** | Nombre de la prueba: *TestTimeSeconds*  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:02,8221745 |

### Serializador

#### Pruebas del BuiltinTopic

#### Pruebas de Encapsulación CDR.

#### Pruebas de exploración de tipo.

#### Pruebas de paquetes.

#### Pruebas de primitivas.

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** | En esta prueba se verifica que el fichero de configuración no sea nulo |
| ***Entrada*** | Inicialmente se tiene el fichero de configuración |
| ***Referencia*** |  |
| ***Código*** | public void TestExistConfiguration()  {  Assert.IsNotNull(ddsConfig);  } |
| ***Salida*** | Nombre de la prueba: TestExistConfiguration  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,0286387 |