**DOCUMENTO DE PRUEBAS UNITARIAS DEL API RTPS.**

**Introducción**

Paquetes del API a probar:

* **CORE:** Contiene todas las clases referentes a las diferentes excepciones que se manejan dentro de DDS, así también las políticas de calidad de servicio de cada uno de los componentes, y maneja los estados de los mismos.
* **DOMAIN:** Dentro de esta carpeta se encuentran las clases y métodos necesarios para crear un dominio participante dentro del entorno DDS
* **PUB:** En esta carpeta tenemos las clases para crear una entidad publicadora, capaz de escribir los datos y enviarlos a la red DDS
* **SUB:** En esta carpeta se definen las clases necesarias para crear una entidad suscriptora, la que participará en la recepción y lectura de los datos publicados dentro de un grupo de dominios participantes.
* **TOPIC:** Aquí se define el tema que se va a manejar dentro de un grupo de dominios.
* **TYPE:** Define el tipo de datos que se van a manjar dentro de un tema determinado

El orden de prioridad en las pruebas unitarias del software se presenta a continuación.

**Fábrica de dominios y dominios**

Creación de fábricas de dominios, de dominios y la habilitación estos

Calidad de servicio en la fábrica de dominios

**Publicador**

Creación de Publicador y DataWriter

Llamadas al publicador y al DataWriter

Calidad de servicio en Publicador y DataWriter

**Suscriptor**

Creación de DataReader y Suscriptor

Llamadas al DataReader y al Suscriptor

Calidad de servicio en el DataReader y el Suscriptor

**Tema**

Creación de un tema

Probar los tipos de tema

Llamadas al tema

Calidad de servicio del tema

Descriptor del tema

**Tipo de Datos**

Pruebas generales de los tipos de datos que maneja DDS

**Políticas**

Creación de entidades

Calidad de servicio en las entidades

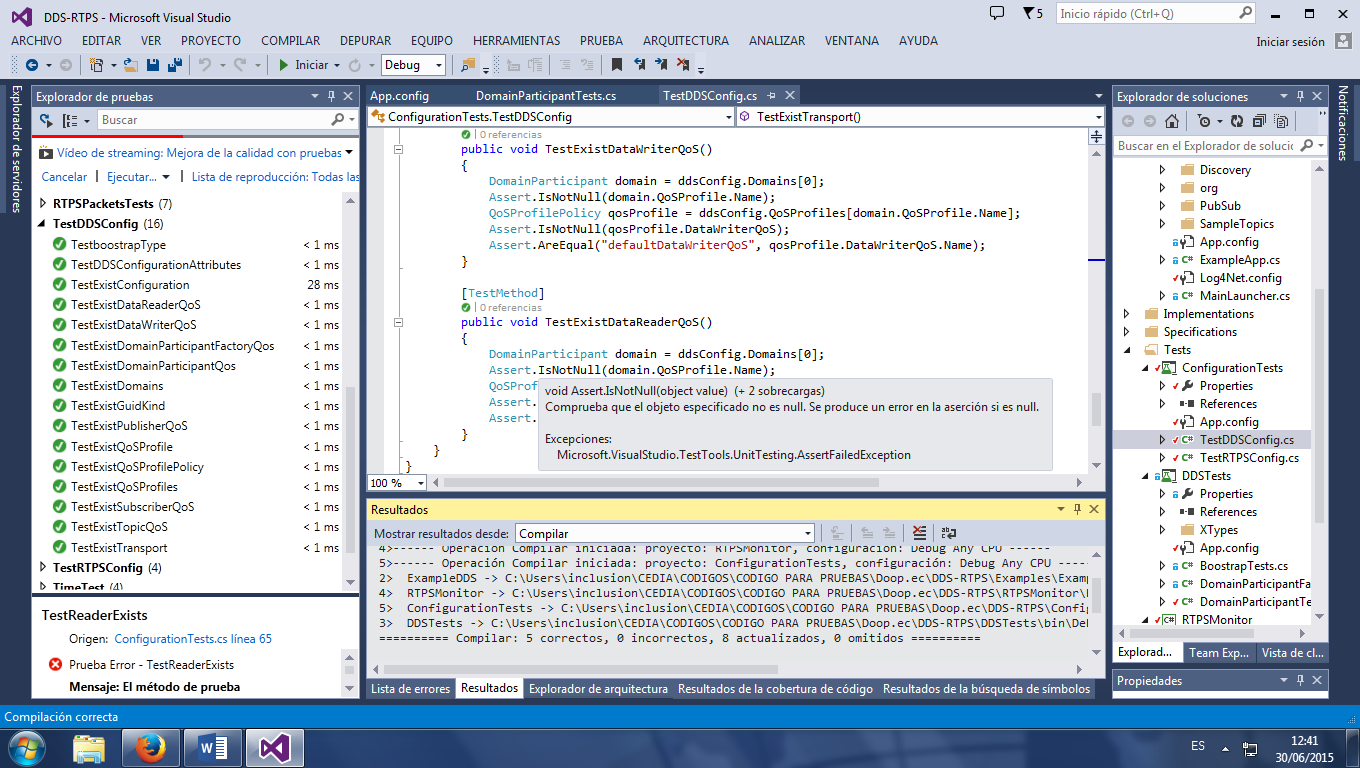
Manejo de excepciones

Manejo de los objetos del DDS

**Descripción de las pruebas**

En esta sección se detallan las pruebas que se van realizando en cada uno de los componentes del API DDS.

***Pruebas de configuración de DDS:***



|  |  |
| --- | --- |
| **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 |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** | En esta prueba se verifica que el fichero de configuración no sea nulo, así también que las características de vendor y versión tampoco sean nulas.  También se verifica que vendor tenga el valor “Doopec” y en la versión se tenga el valor “2.1” |
| ***Entrada*** | <DDS xmlns="urn:Configuration" vendor="Doopec" version="2.1"> |
| ***Referencia*** |  |
| ***Código*** | public void TestDDSConfigurationAttributes()  {  Assert.IsNotNull(ddsConfig);  Assert.IsNotNull(ddsConfig.Vendor);  Assert.IsNotNull(ddsConfig.Version);  Assert.AreEqual("Doopec", ddsConfig.Vendor);  Assert.AreEqual("2.1", ddsConfig.Version);  Assert.AreEqual(new Version(2, 1), Version.Parse(ddsConfig.Version));  } |
| ***Salida*** | Nombre de la prueba: TestDDSConfigurationAttributes  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,0003093 |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** | En esta prueba se verifica que el fichero de configuración y que la etiqueta de configuración BoostrapType no contengan un valor nulo.  Se verifica también que en la propiedad Name de la etiqueta de configuración BoostrapType se encuentre el valor “default” y que en la propiedad Type se encuentre el valor “Doopec.Dds.Core.BootstrapImpl, Doopec” |
| ***Entrada*** | <boostrapType name="default" type="Doopec.Dds.Core.BootstrapImpl, Doopec"/> |
| ***Referencia*** |  |
| ***Código*** | public void TestboostrapType()  {  Assert.IsNotNull(ddsConfig);  Assert.IsNotNull(ddsConfig.BoostrapType);  Assert.AreEqual("default", ddsConfig.BoostrapType.Name);  Assert.AreEqual("Doopec.Dds.Core.BootstrapImpl, Doopec", ddsConfig.BoostrapType.Type);  } |
| ***Salida*** | Nombre de la prueba: TestboostrapType  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,0002879 |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** | Se verifica que ellos atributos Name y Id correspondientes a la etiqueta Domains sean “Main” y 0 respectivamente.  Además se confirma que el contador del arreglo de dominios tenga el valor de 1. |
| ***Entrada*** | <domain name="Main" id="0"> |
| ***Referencia*** |  |
| ***Código*** | public void TestExistDomains()  {  Assert.AreEqual(1, ddsConfig.Domains.Count);  Assert.AreEqual("Main", ddsConfig.Domains[0].Name);  Assert.AreEqual(0, ddsConfig.Domains[0].Id);  } |
| ***Salida*** | Nombre de la prueba: TestExistDomains  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,000304 |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistQoSProfiles()  {  Assert.AreEqual(1, ddsConfig.QoSProfiles.Count);  Assert.AreEqual("defaultQoS", ddsConfig.QoSProfiles[0].Name);  Assert.IsNotNull(ddsConfig.QoSProfiles["defaultQoS"]);  Assert.AreEqual("defaultQoS", ddsConfig.QoSProfiles["defaultQoS"].Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistTransport()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.TransportProfile);  Assert.AreEqual("defaultRtps", domain.TransportProfile.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistQoSProfile()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile);  Assert.AreEqual("defaultQoS", domain.QoSProfile.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistGuidKind()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.Guid);  Assert.AreEqual(Doopec.Configuration.GuiKind.Fixed, domain.Guid.Kind);  Assert.AreEqual("7F294ABE-33F2-40B9-BFF5-7D9376EA061C", domain.Guid.Val);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistQoSProfilePolicy()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile);  Assert.AreEqual("defaultQoS", qosProfile.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistDomainParticipantFactoryQos()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.DomainParticipantFactoryQos);  Assert.AreEqual("defaultDomainParticipantFactoryQoS", qosProfile.DomainParticipantFactoryQos.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistDomainParticipantQos()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.DomainParticipantQos);  Assert.AreEqual("defaultDomainParticipantQoS", qosProfile.DomainParticipantQos.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistTopicQoS()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.TopicQoS);  Assert.AreEqual("defaultTopicQoS", qosProfile.TopicQoS.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistPublisherQoS()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.PublisherQoS);  Assert.AreEqual("defaultPublisherQoS", qosProfile.PublisherQoS.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistSubscriberQoS()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.SubscriberQoS);  Assert.AreEqual("defaultSubscriberQoS", qosProfile.SubscriberQoS.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistDataWriterQoS()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.DataWriterQoS);  Assert.AreEqual("defaultDataWriterQoS", qosProfile.DataWriterQoS.Name);  } |
| ***Salida*** |  |

|  |  |
| --- | --- |
| **Llamada:**  App.config | |
| ***Descripción*** |  |
| ***Entrada*** |  |
| ***Referencia*** |  |
| ***Código*** | public void TestExistDataReaderQoS()  {  DomainParticipant domain = ddsConfig.Domains[0];  Assert.IsNotNull(domain.QoSProfile.Name);  QoSProfilePolicy qosProfile = ddsConfig.QoSProfiles[domain.QoSProfile.Name];  Assert.IsNotNull(qosProfile.DataReaderQoS);  Assert.AreEqual("defaultDataReaderQoS", qosProfile.DataReaderQoS.Name);  } |
| ***Salida*** |  |

**DOMAIN**

Se definen las siguientes pruebas:

**Clase:** DomainPaticipantFactory

|  |  |
| --- | --- |
| **Llamada:**  public static DomainParticipantFactory GetInstance(Bootstrap bootstrap) | |
| ***Descripción*** | En esta prueba se crea una instancia de tipo fábrica de dominios, y se comprueba que la misma no contenga un valor nulo, y adicionalmente que el tipo de la instancia de fábrica sea igual a “typeof(Doopec.Dds.Domain.DomainParticipantFactoryImpl)” |
| ***Entrada*** | Inicialmente no existe ninguna instancia de fábrica de dominios |
| ***Referencia*** | **2.2.2.2.2.3** get\_instance  Esta operación devuelve el producto único de DomainParticipantFactory. La operación se puede llamar varias veces sin efectos secundarios y volverá la misma instancia DomainParticipantFactory. |
| ***Código*** | public void TestGetInstance()  {  DomainParticipantFactory factory = DomainParticipantFactory.GetInstance(Bootstrap.CreateInstance());  Assert.IsNotNull(factory);  Assert.AreEqual(typeof(Doopec.Dds.Domain.DomainParticipantFactoryImpl), factory.GetType());  } |
| ***Salida*** | Nombre de la prueba: TestGetInstance  Resultado de la prueba: https://i-msdn.sec.s-msft.com/dynimg/IC689871.png  Duración de la prueba: 0:00:00,0343843 |

**Clase:** DomainPaticipant

|  |  |
| --- | --- |
| **Llamada:**  Publisher CreatePublisher(PublisherQos qos,  PublisherListener listener,  ICollection<Type> statuses); | |
| ***Descripción*** | En esta prueba se genera el método necesario para crear una entidad publicadora, colocando los datos necesarios para estructurar la entidad |
| ***Entrada*** | Inicialmente no existe ningún objeto de tipo entidad publicadora |
| ***Referencia*** |  |
| ***Código*** | public void TestGetInstance()  {  DomainParticipantFactory factory = DomainParticipantFactory.GetInstance(Bootstrap.CreateInstance());  Assert.IsNotNull(factory);  Assert.AreEqual(typeof(Doopec.Dds.Domain.DomainParticipantFactoryImpl), factory.GetType());  } |
| ***Salida*** | https://i-msdn.sec.s-msft.com/dynimg/IC689871.png |

|  |  |
| --- | --- |
| **Llamada:**  Publisher CreatePublisher(string qosLibraryName,  string qosProfileName,  PublisherListener listener, ICollection<Type> statuses); | |
| ***Descripción*** | En esta prueba se genera el método necesario para crear una entidad publicadora, colocando los datos necesarios para estructurar la entidad |
| ***Entrada*** | Inicialmente no existe ningún objeto de tipo entidad publicadora |
| ***Referencia*** |  |
| ***Código*** | public void TestGetInstance()  {  DomainParticipantFactory factory = DomainParticipantFactory.GetInstance(Bootstrap.CreateInstance());  Assert.IsNotNull(factory);  Assert.AreEqual(typeof(Doopec.Dds.Domain.DomainParticipantFactoryImpl), factory.GetType());  } |
| ***Salida*** | https://i-msdn.sec.s-msft.com/dynimg/IC689871.png |

**CORE**

Dentro de esta carpeta tenemos las siguientes:

Modifiable

Policy

Status