Laboratorio: Creación de un Message Driven Bean

Objetivos

El objetivo de este laboratorio es enseñar a los alumnos a crear una aplicación J2EE que utilice Message Driven Beans con Rational Application Developer v.6.01 (RAD) o posterior, utilizando para ello el Embedded Messaging que provee WebSphere 6.0 para soportar comunicaciones asíncronas utilizando el API de JMS.

En este laboratorio crearemos un JSP que invocará un servlet pasándole una serie de parámetros. Esos parámetros serán leídos por el servlet y utilizados para crear un mensaje JMS que será depositado en una cola. Los mensajes serán procesados por un MDB que simplemente escribirá al log el contenido del mismo.

Prerrequisitos

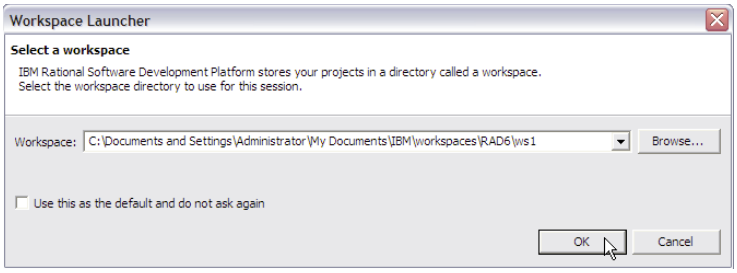
RAD 6.0 1 o posterior debe estar instalado en su equipo.

Instrucciones

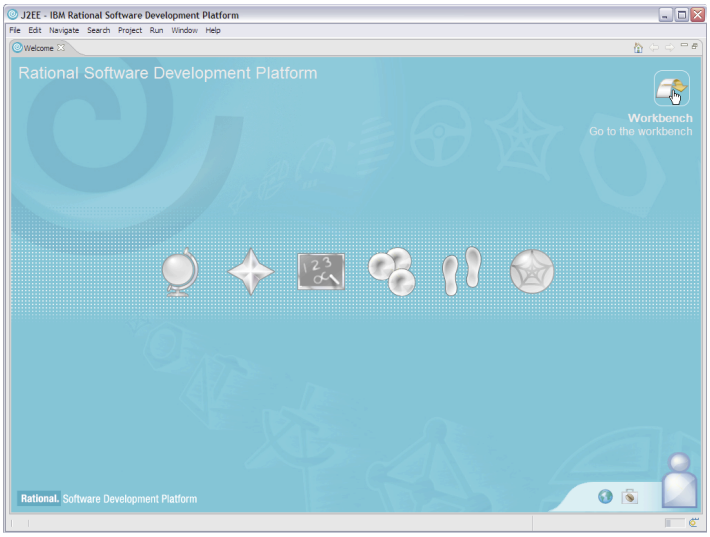
Primera parte - Configuración del servidor de aplicaciones y de la cola de mensajes

Inicie el entorno de desarrollo pulsando el siguiente icono en la barra de tareas  .

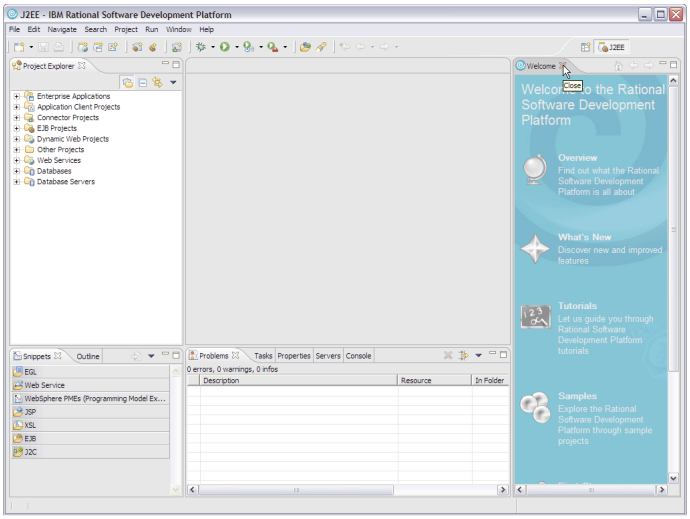
A continuación aparece una pantalla en la que se solicita la ubicación de un directorio en el que se guardarán todos los archivos del proyecto (workspace). Elija un directorio que no exista en su máquina para empezar desde cero.



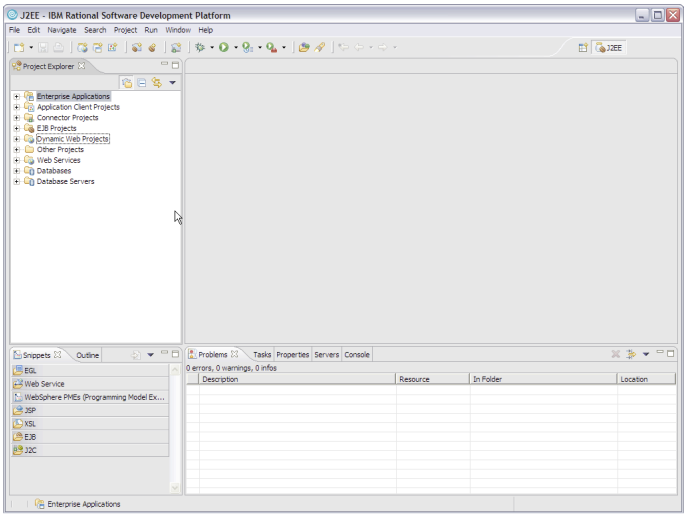
Una vez completado este paso, el entorno de desarrollo arranca y aparece la siguiente pantalla



Pulse el ícono en forma de flecha situado en el extremo superior derecho de la pantalla para salir de la pantalla de bienvenida.

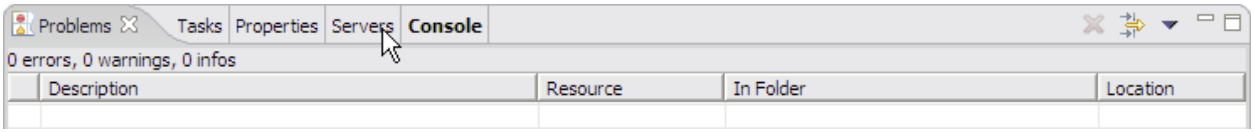


Cierre la pestaña “Welcome” como se muestra a continuación para poder empezar a trabajar sin que nos estorbe.

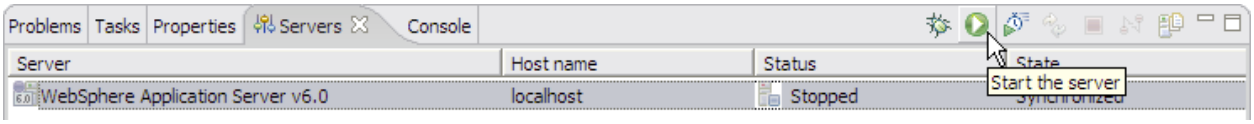


Ahora que ya tenemos nuestro entorno de desarrollo listo, vamos a configurar la cola de mensajes asi como todos los parámetros que necesitará el MDB para comunicarse con la misma. Todo esto se hace a través de la consola de administración del servidor de aplicaciones. Por lo tanto, lo primero que debemos hacer es arrancar el servidor que crear por defecto RAD para realizar las pruebas.

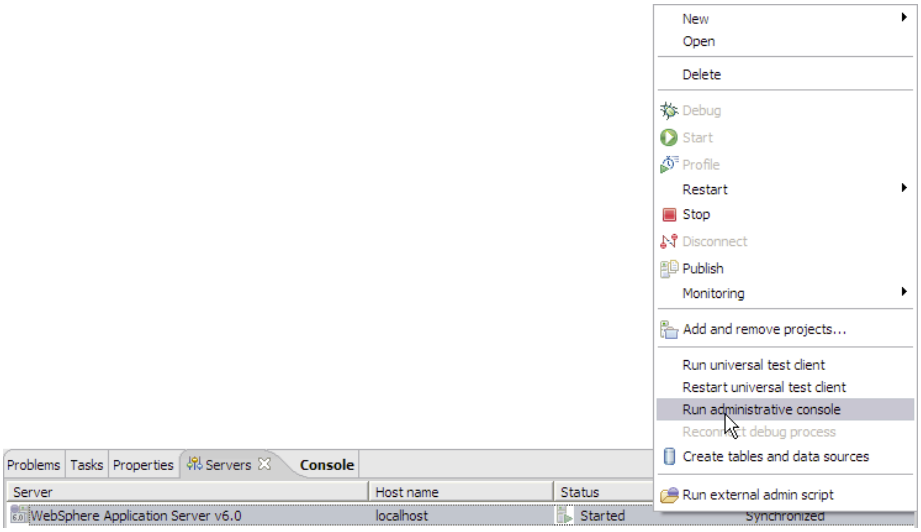
Haga click sobre la pestaña “Servers” como se muestra a continuación.



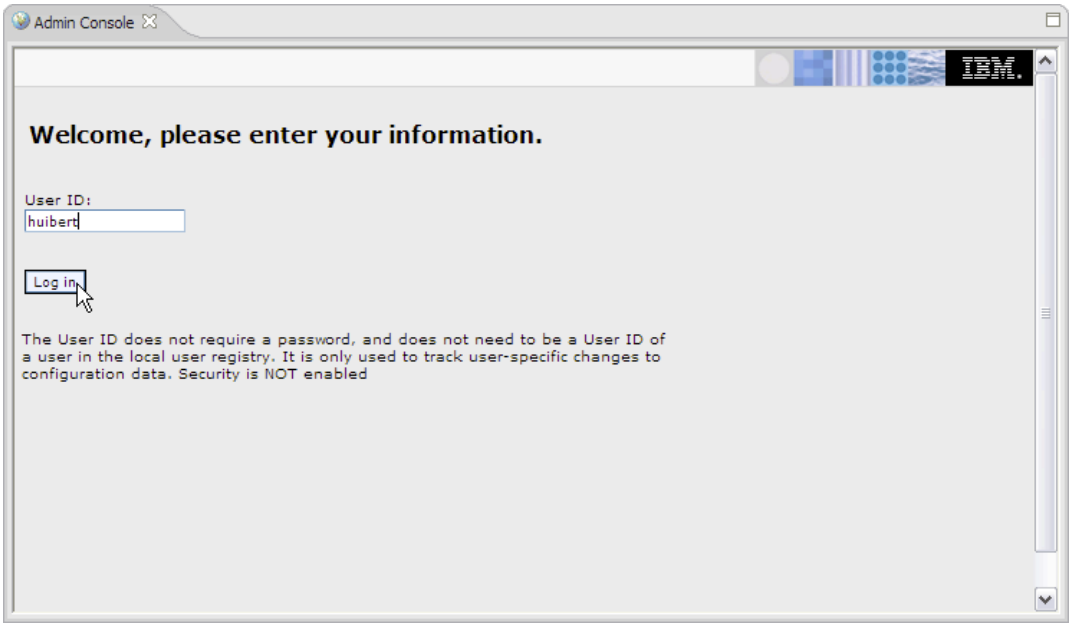
Como parte del proceso de instalación, RAD ya ha creado un servidor por defecto llamado “WebSphere Application Server 6.0”. Selecciónelo haciendo click sobre el nombre. Para arrancarlo existen varias maneras, en este caso simplemente haremos click sobre el ícono verde con un triángulo blanco.



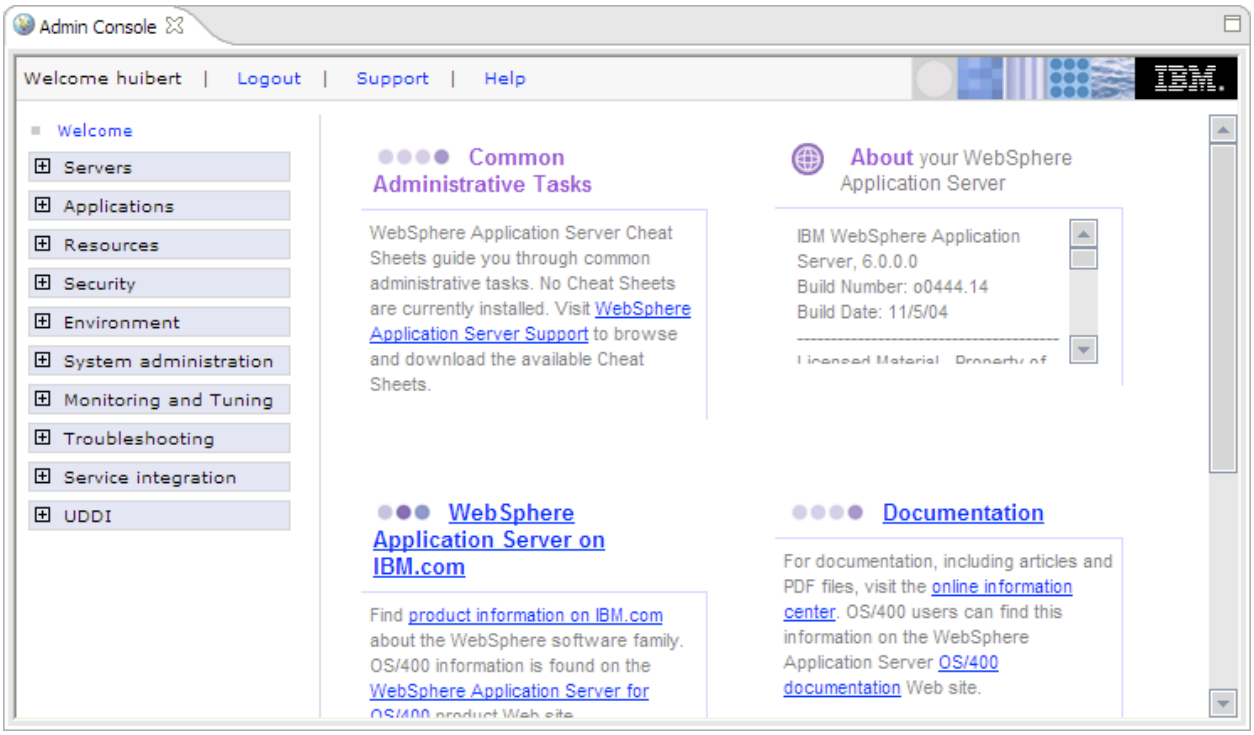
El servidor de aplicaciones tarda un poco en arrancar. Durante ese proceso pueden observar los mensajes que manda a la consola haciendo click sobre la pestaña “Console”. Regrese a la pestaña “Servers” y espere a que el estatus cambie a “Started”. Entonces haga right-click sobre el servidor y seleccione la opción “Run administrative console” del menú contextual como se muestra a continuación.



Esta opción abre una nueva pestaña que en realidad solo es un browser que se conecta al puerto de administración del servidor seleccionado, como pueden observar en la siguiente ilustración.

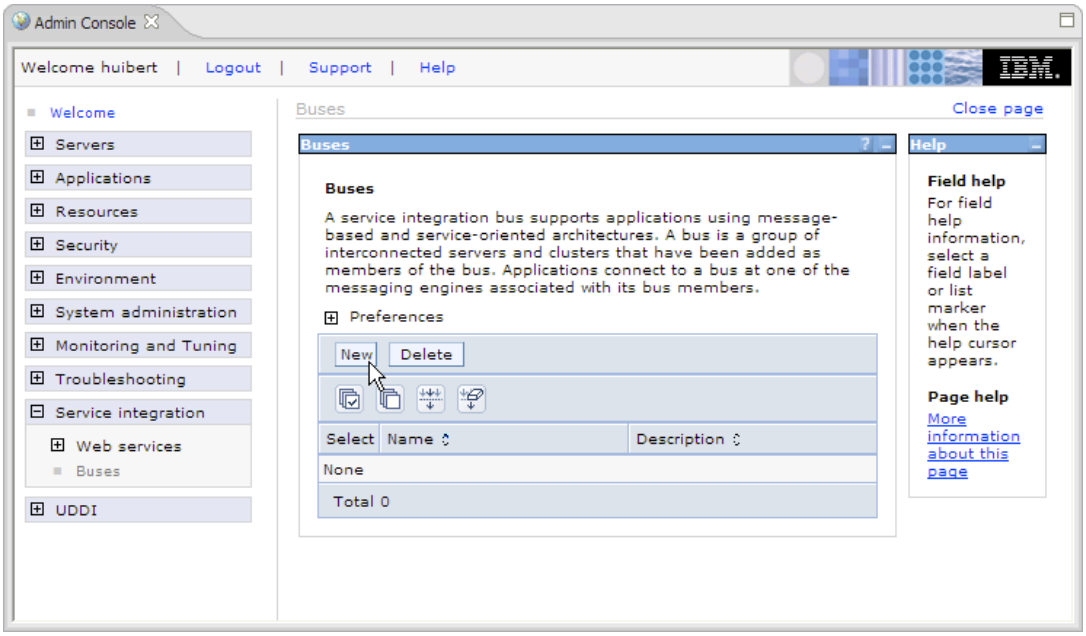


Como no está activada la seguridad de WebSphere pueden utilizar cualquier nombre como “User ID”. Pulsen el botón “Log In” para continuar.

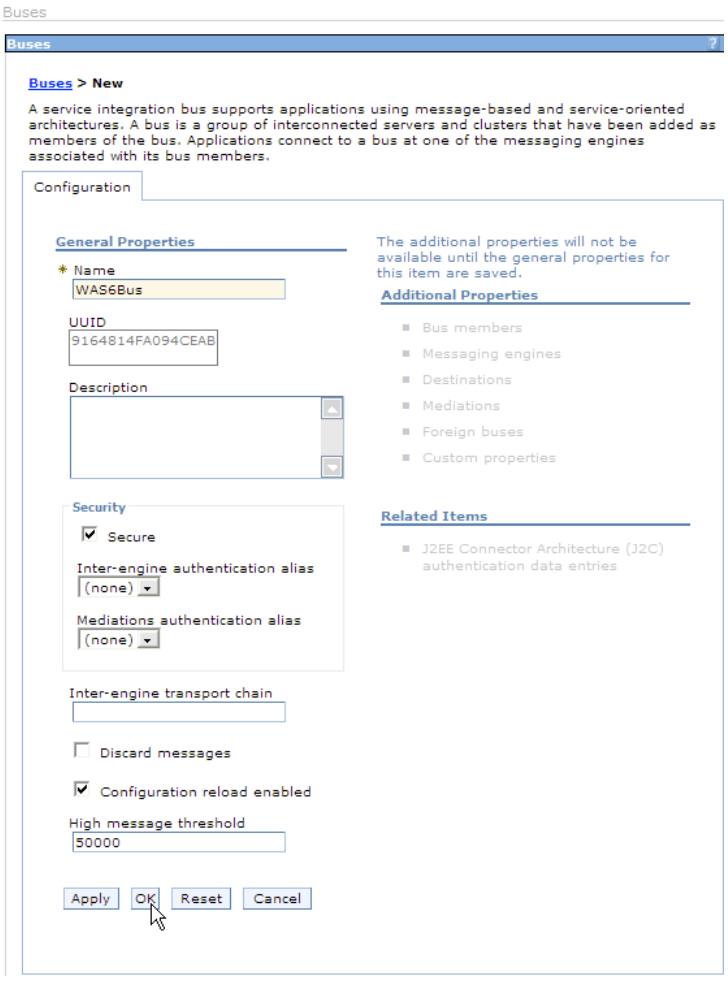


Lo primero que debemos hacer es crear un nuevo “Service Integration Bus”. Se trata de una nueva característica de WebSphere 6.0 que permite soportar aplicaciones que usan arquitecturas tanto basadas en mensajes como basadas en servicios.

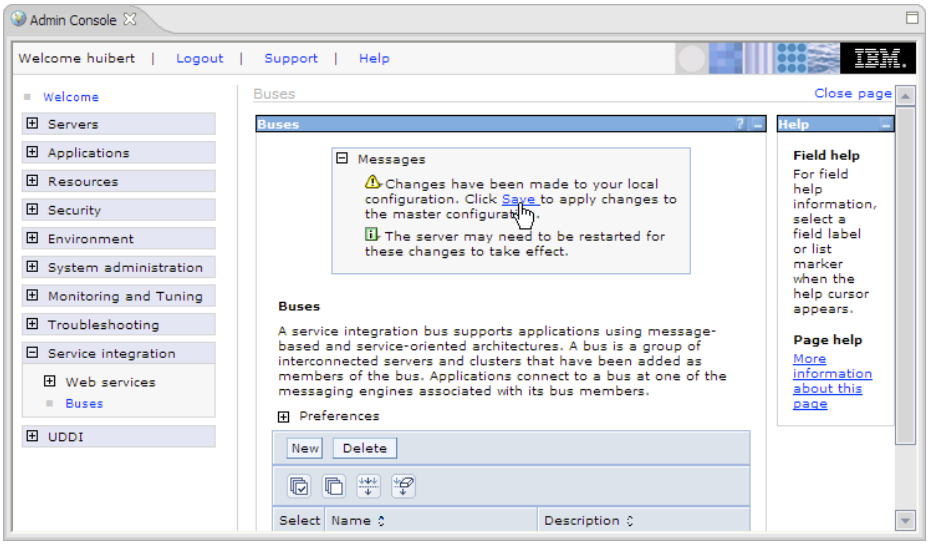
En la consola expanda el nodo “Service Integration”, seleccione “Buses” y haga click sobre el botón de “New” para crear el nuevo bus.



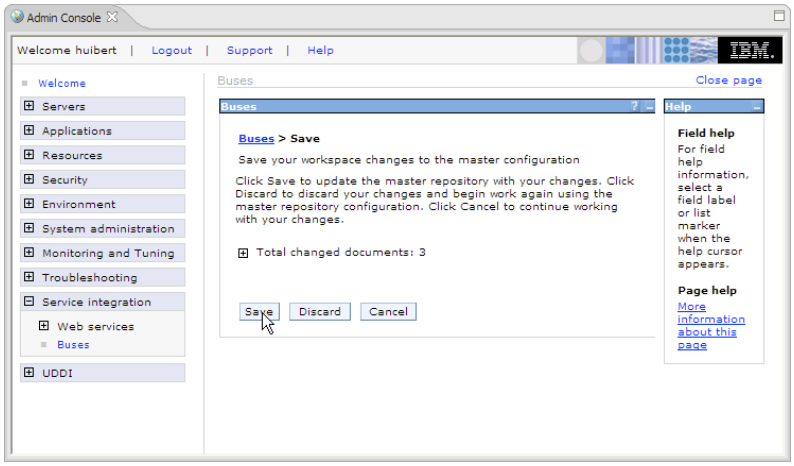
Seleccione un nombre para el bus que acaba de crear. En nuestro caso vamos a usar “WAS6BUS”.



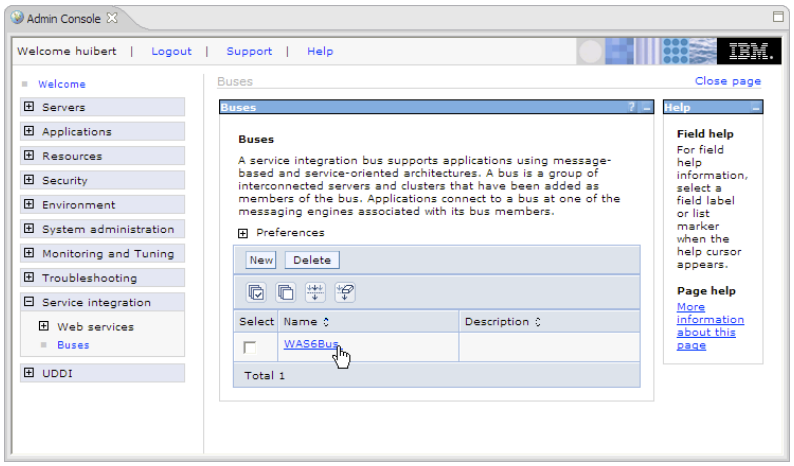
Pulse “Ok”. Ahora salve los cambios que se han producido en la configuración haciendo click “sobre la liga que se muestra a continuación.



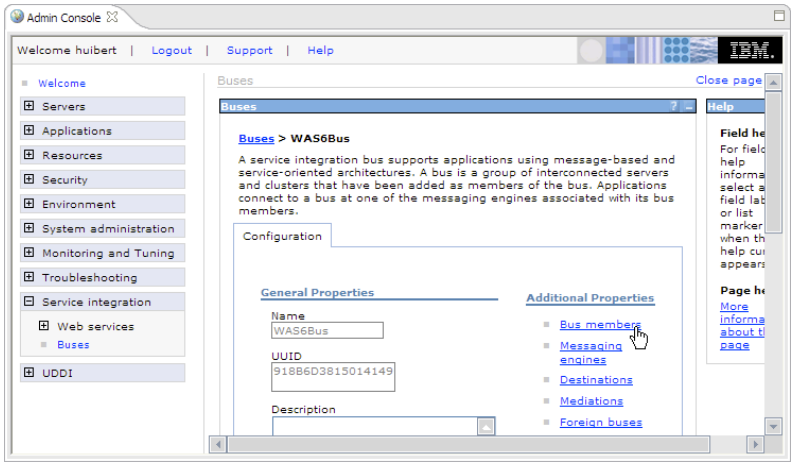
Pulse ahora el botón “Save” para guardar los cambios.



Ahora crearemos un nuevo Messaging Engine, lo cual se logra al asociar un servidor al bus que acabamos de crear. Un Messaging Engine es análogo a un Queue Manager de WebSphere MQ.

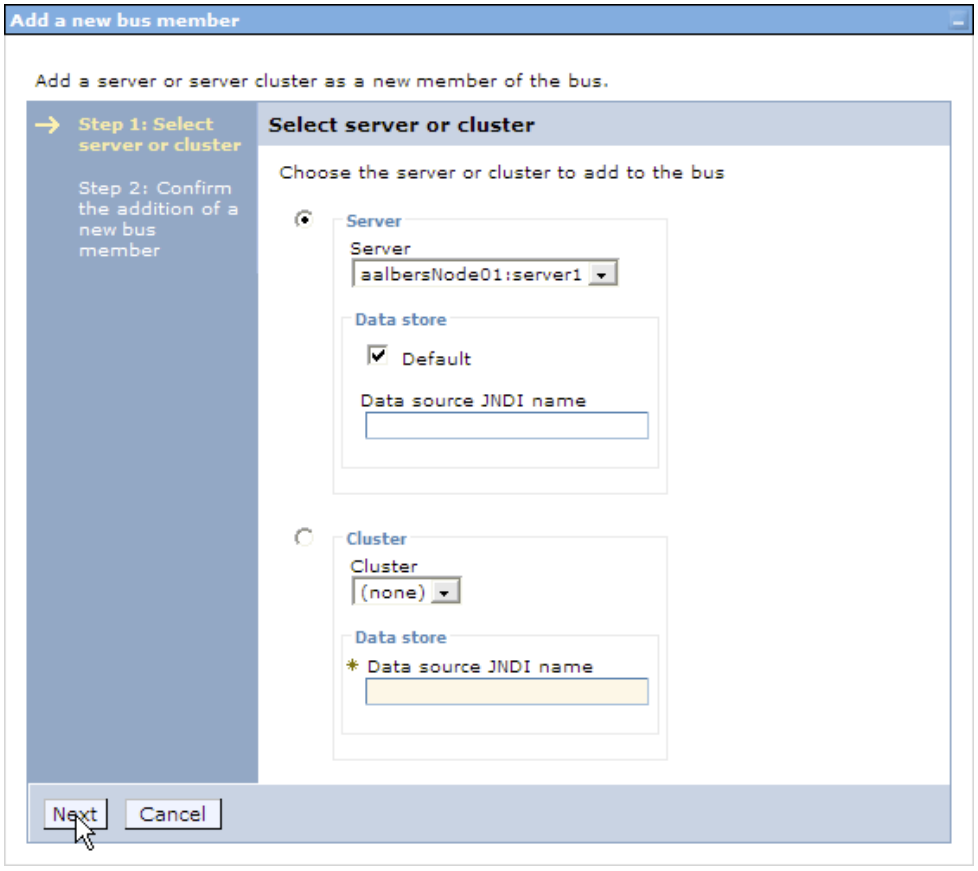


Ahora seleccionamos la opción “Bus members” bajo “Additional properties

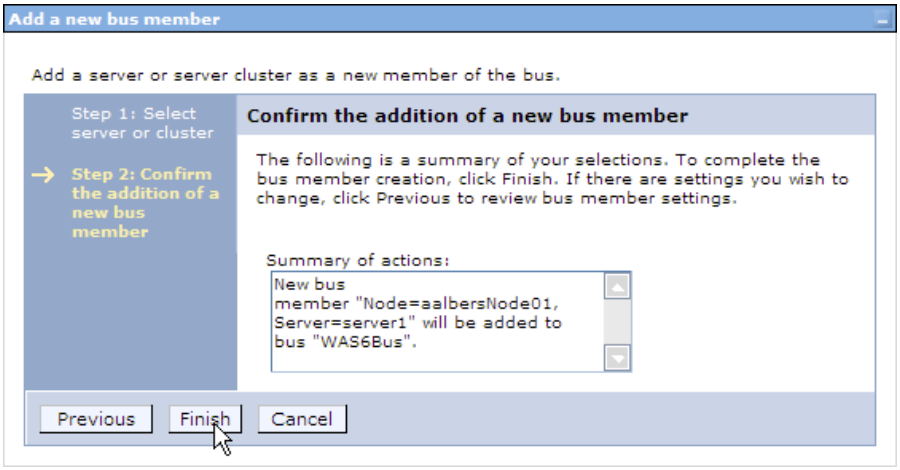


Pulse el botón “Add”.

Agregue el servidor. En principio no necesita hacer cambios porque en nuestro caso solo tenemos un servidor y no hemos definido ningún cluster.

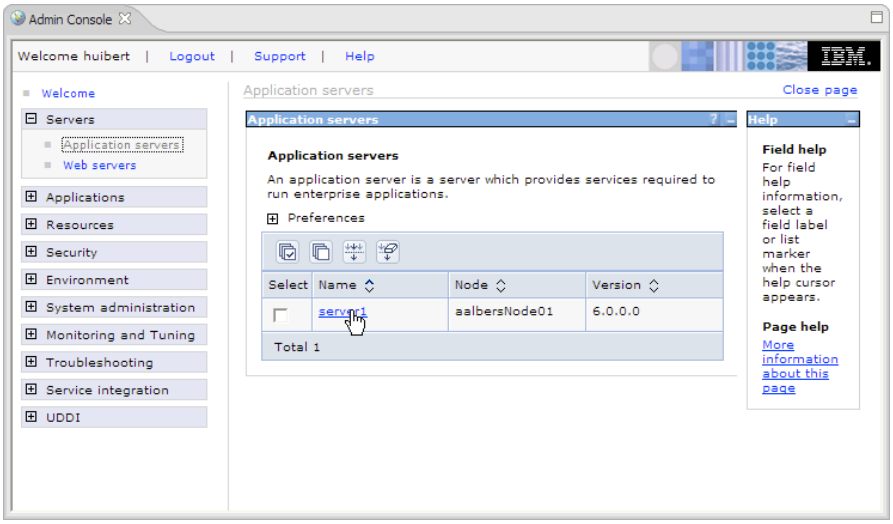


Pulse el botón “Next” para pasar al siguiente paso.



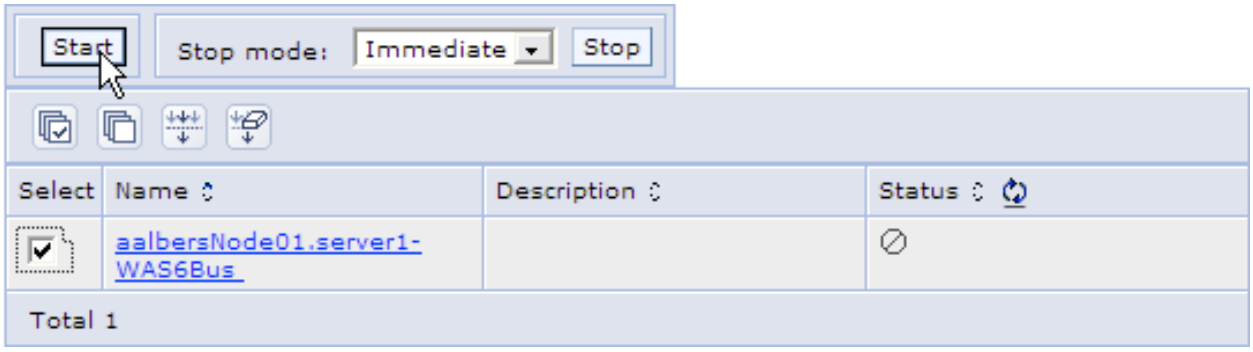
Pulse “Finish” para terminar. Guarde los cambios realizados a la configuración, tal y como lo hizo anteriormente.

Al crear el bus, automáticamente se ha creado un motor de mensajería ubicado en el servidor. Vamos a ahondar un poco más en este punto. En la consola de administración, expanda el nodo “Servers” y haga click sobre “Application servers”. Seleccione “server1”.

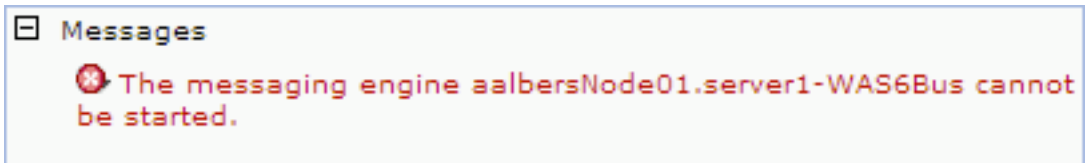


En la siguiente pantalla seleccione la opción de “Messaging engines” bajo “Server messaging”.

Seleccione el servidor y pulse el botón de “Start” para arrancar el bus.



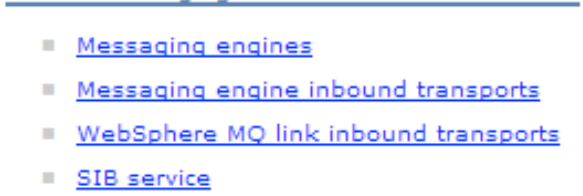
En ese momento le debe aparecer el siguiente error.

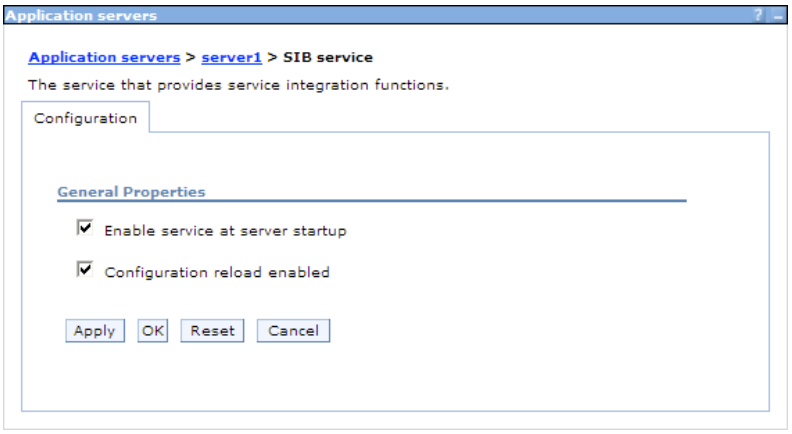


Esto es normal ya que tras crear el motor de mensajería es necesario reiniciar el servidor, lo mismo ocurre a nivel de cluster si lo que se agrega es un cluster. Más adelante en este laboratorio reiniciaremos el servidor.

Regrese a la pantalla del servidor haciendo click sobre el nombre del servidor (”server1”).

Seleccione la opción “SIB service” bajo “Server messaging”.

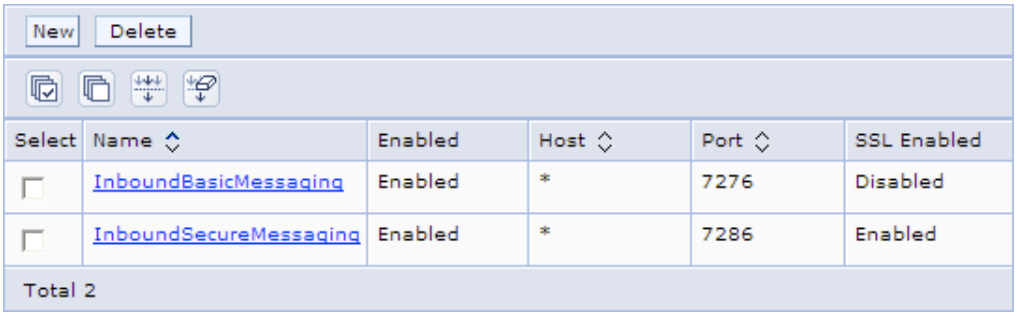




Por defecto, el bus arranca junto con el servidor. Esto es algo que normalmente es deseable. Sin embargo, es posible cambiarlo en esta pantalla. En este caso no vamos a realizar cambios. Regrese a la pantalla de configuración del servidor.

Haga click sobre “Messaging engine inbound transports”.

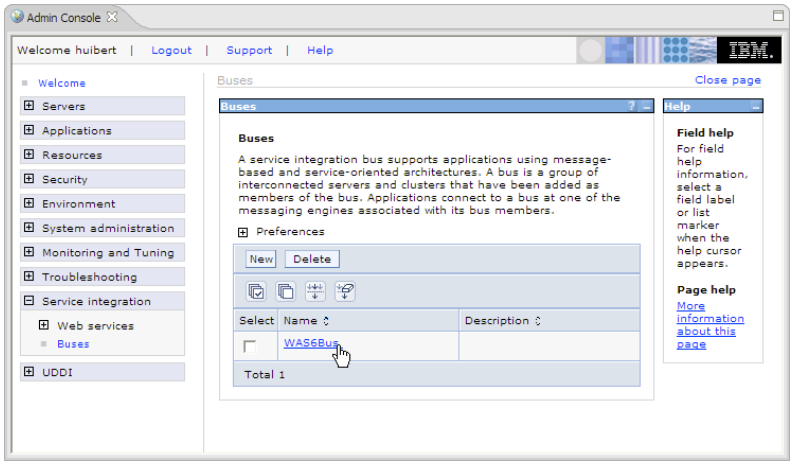
En esta pantalla se pueden apreciar los puertos utilizados por el servicio de mensajería, tanto para mensajes normales como encriptados con SSL.



Recordemos que estamos usando el puerto 7276.

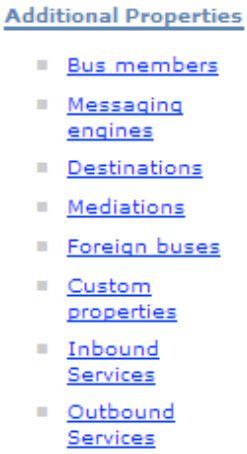
Ahora vamos a crear lo que se conoce como un destino (o “destination” en inglés). Un destino es un lugar virtual al que se le puede ligar “producers” (productores de mensajes), “consumers” (consumidores de mensajes) o ambos. Existen varios tipos de destinos, pero en nuestro caso como vamos a trabajar con un MDB que va a consumir los mensajes de una cola, usaremos una ”queue” (cola en inglés).

En la consola de administración, expanda el nodo “Service Integration” y haga click sobre “Buses”

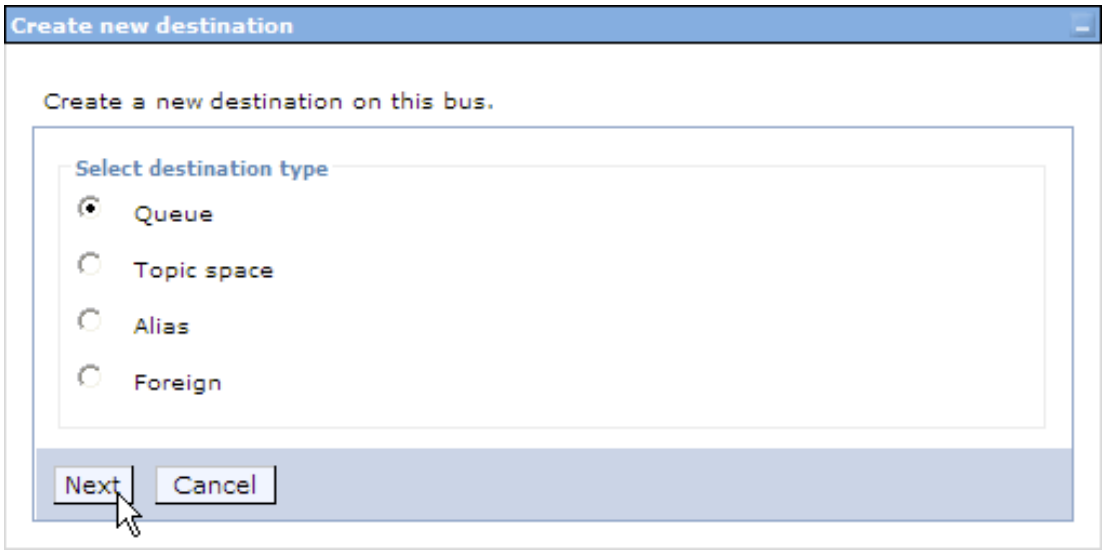


Haga click sobre el bus que creamos anteriormente (”WAS6Bus”).

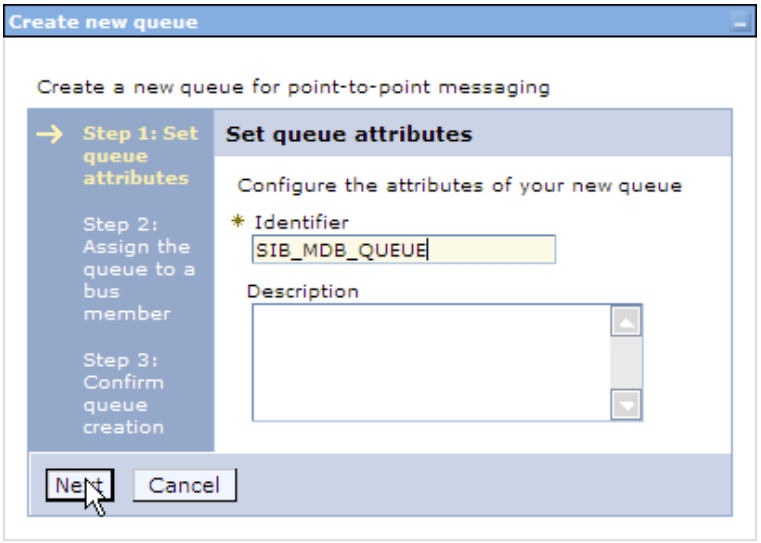
Seleccione la opción “Destinations” bajo “Additional Properties”.



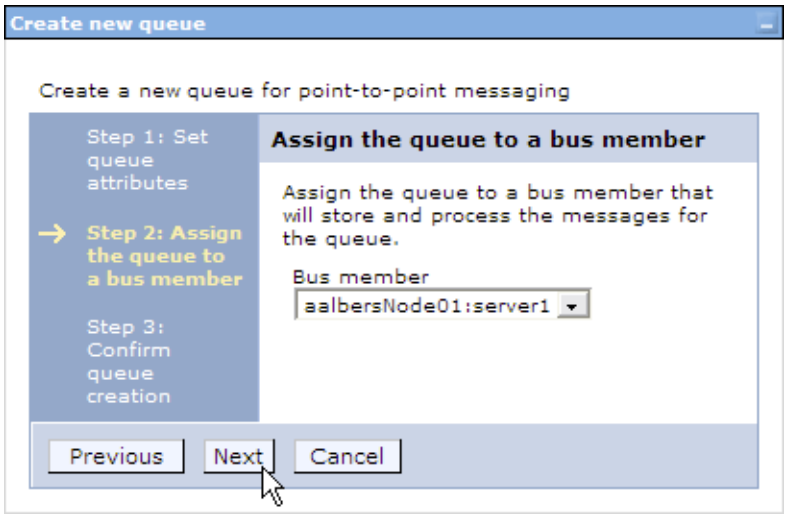
Haga click sobre el botón “New”. Seleccione la opción “Queue” para crear una nueva cola de mensajes. Pulse “Next” para avanzar.



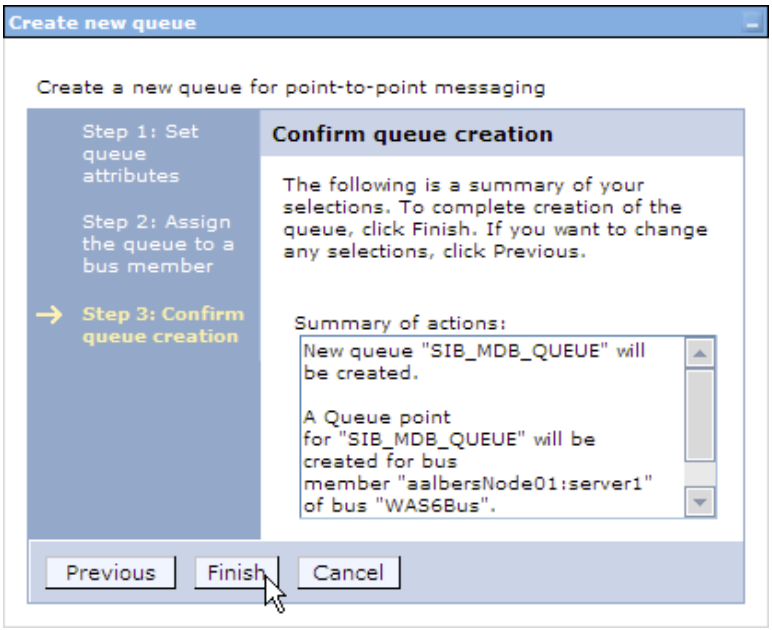
Escriba el nombre de la nueva cola “SIB_MDB_QUEUE” y pulse el botón “Next”.



En principio no necesita cambiar nada en esta pantalla ya que solo hay un servidor conectado al bus que creamos inicialmente.

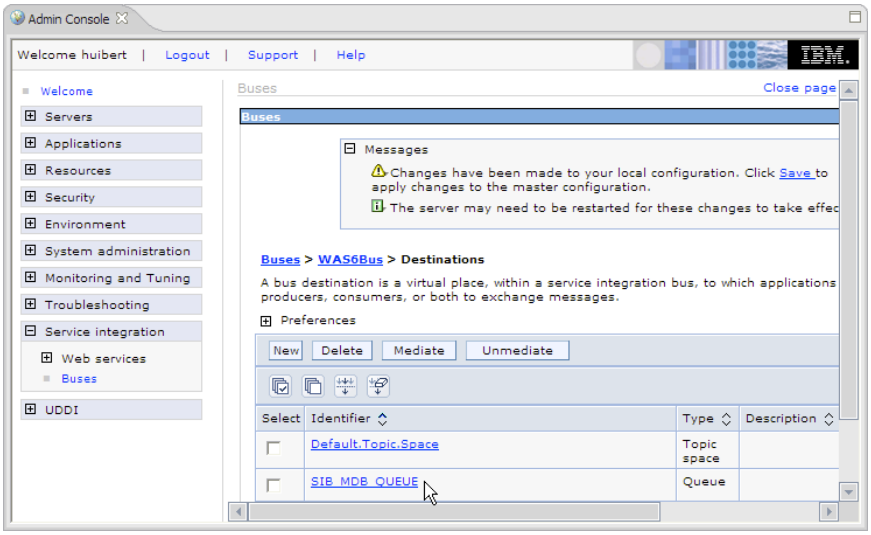


Ahora simplemente pulse “Finish” para confirmar la creación de la cola.



Finalmente, como en ocasiones anteriores tenemos que salvar los cambios realizados a la configuración del servidor, utilizando el mismo procedimiento.

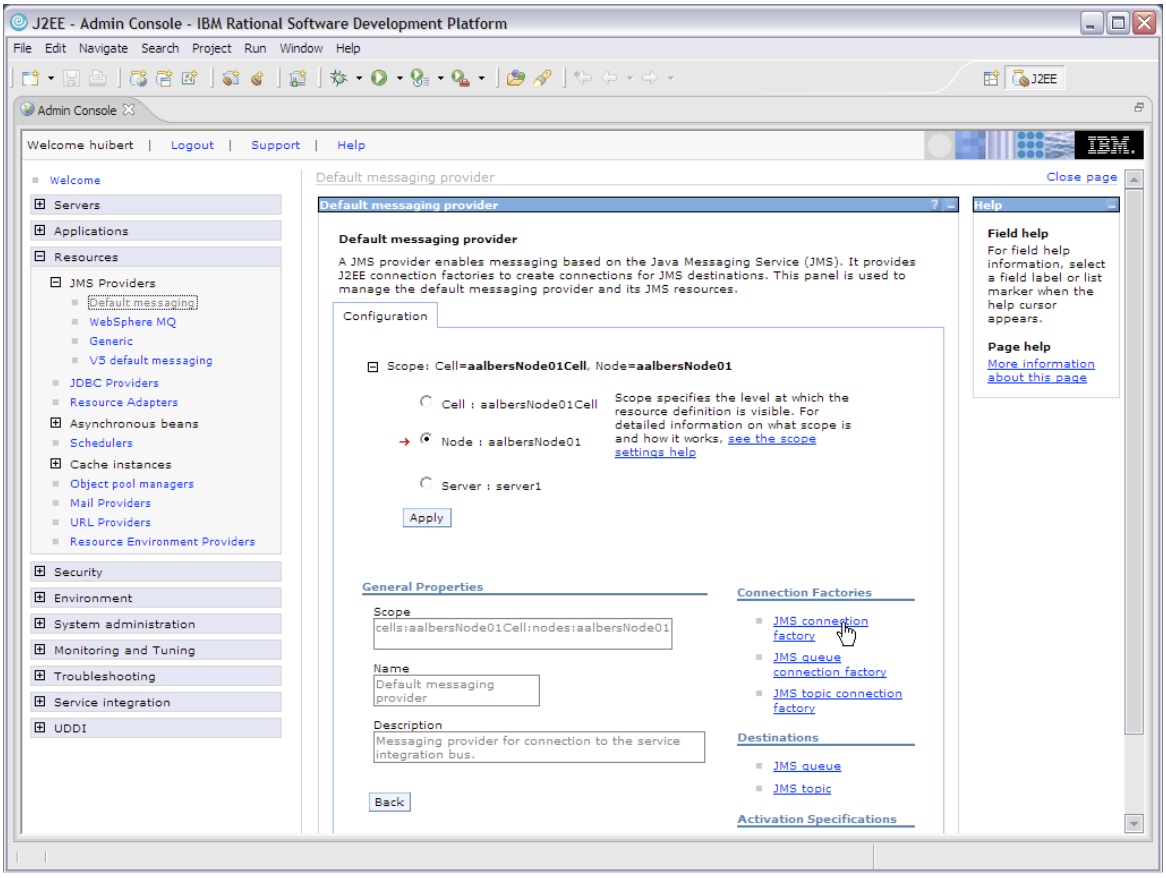
Si no se ha producido nada extraño, la cola debe haberse creado normalmente, tal y como se muestra en la siguiente ilustración.



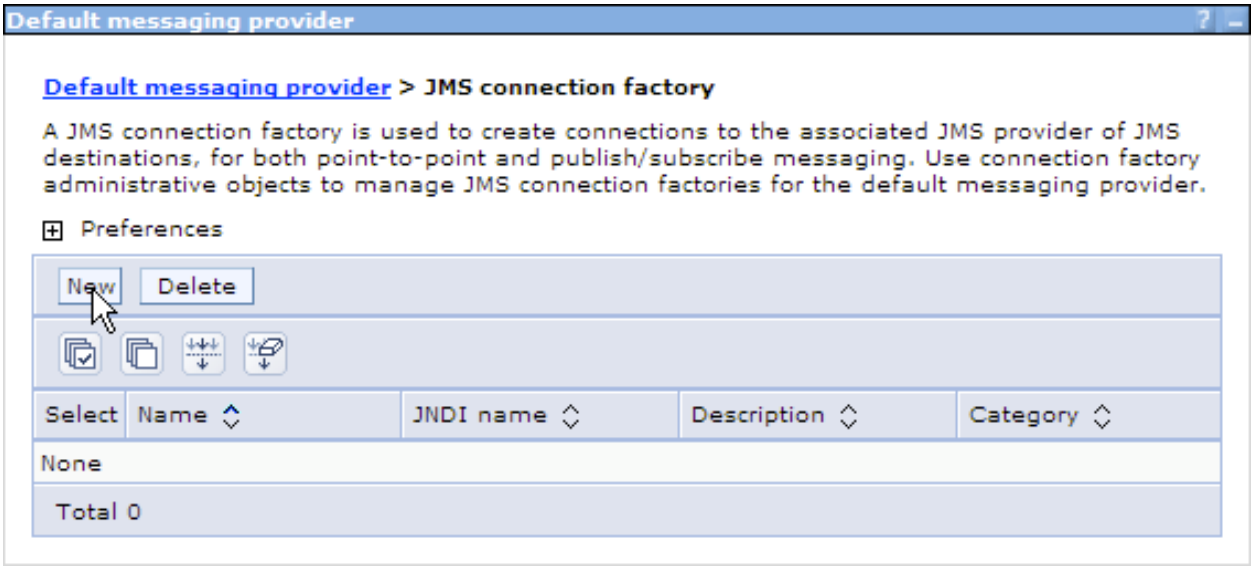
Ya hemos creado la cola. Sin embargo, para que nuestra aplicación pueda interactuar con ella a través de JMS todavía tenemos que configurar otros parámetros adicionales.

En nuestro caso primero tenemos que crear una “Connection factory” la cual podrá ser accesada a través de JNDI a través del nombre “/jms/mdb/ConnectionFactory”.

En la consola de administración, expanda el nodo “Resources>>JMS Providers” y haga click sobre “Default messaging”. Luego bajo “Connection Factories” haga click sobre “JMS connection factory”.



Pulse el botón “New” para crear una nueva “JMS connection factory”.



En a siguiente página deberá insertar los siguientes valores:

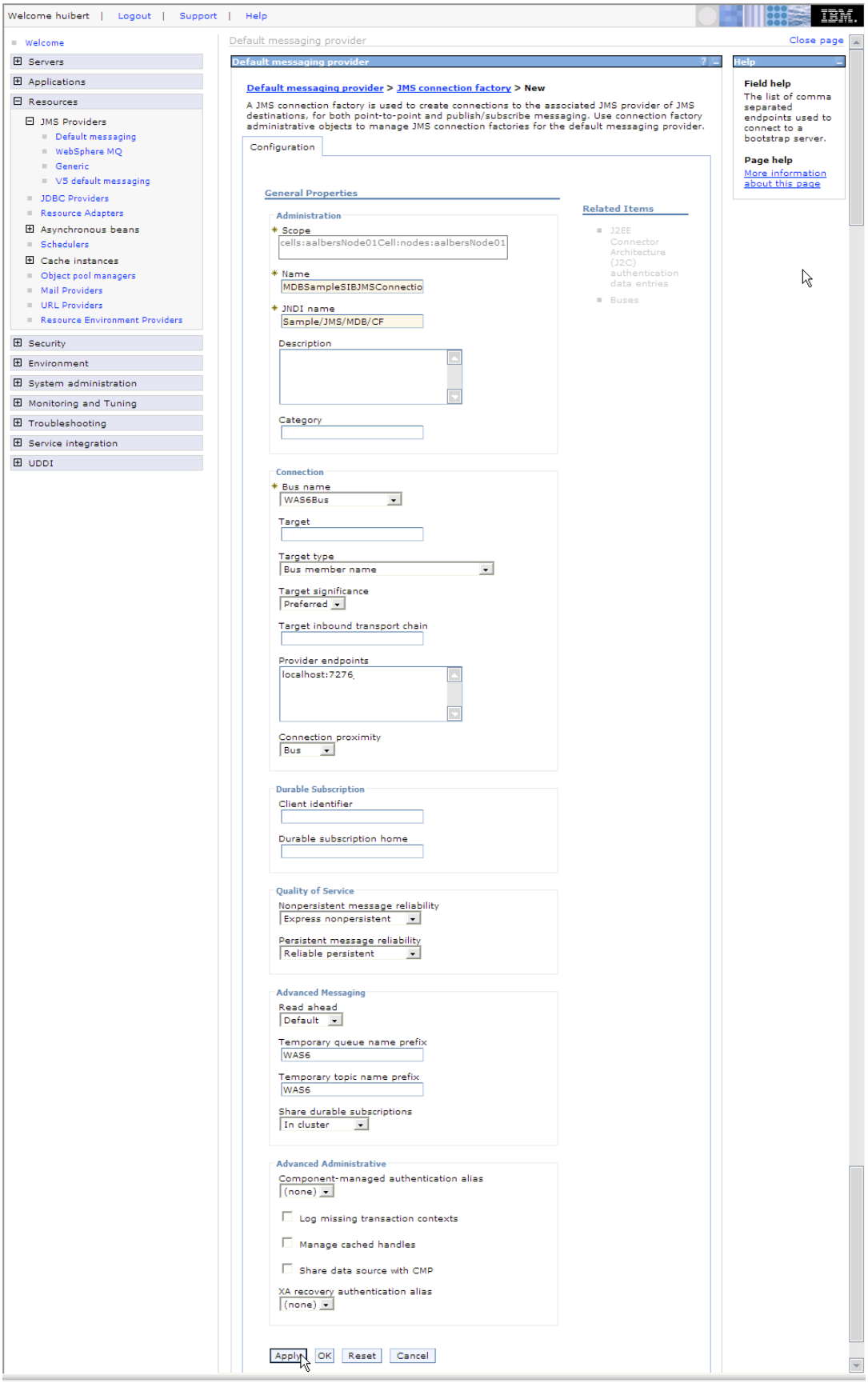
```
Name:                          MDBSampleSIBJMSConnectionFactory
JNDI Name:                     Sample/JMS/MDB/CF
Bus Name:                      WAS6Bus
Temporary Queue Name Prefix:   WAS6
Temporary Topic Name Prefix:   WAS6
Provider endpoints:            localhost:7276
```

En cuanto al último parámetro (”Provider endpoints”) es posible omitir el puerto si se usa el puerto por defecto (7276), aunque en este caso lo hemos incluído para evitar cualquier tipo de confusión.

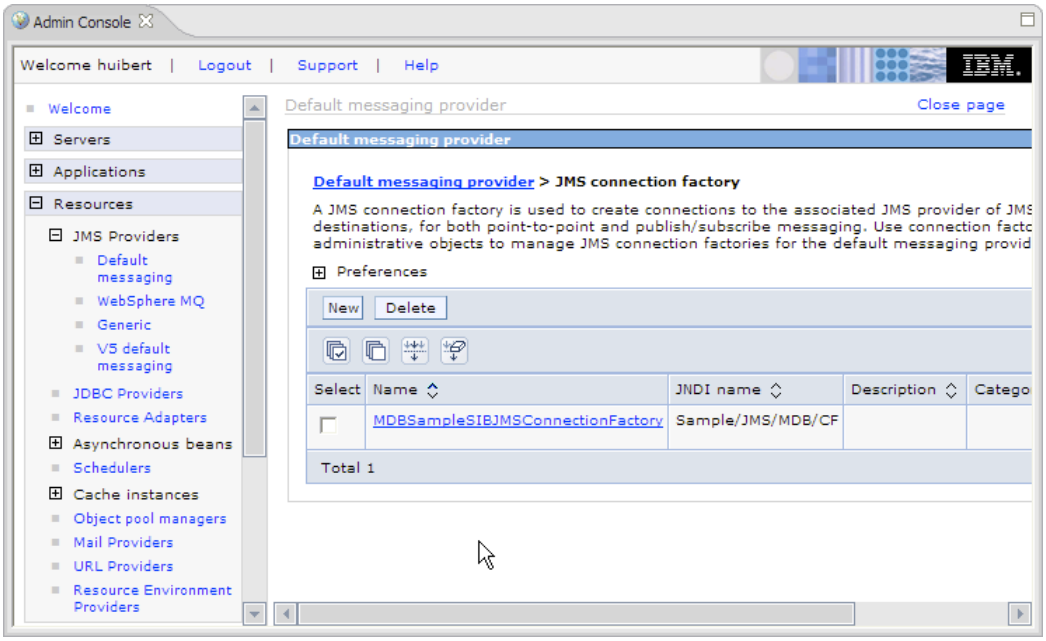
También es importante señalar que a partir de JMS 1.1, se utiliza el mismo “Connection factory” tanto para “queues” como para “topics”.

En la siguiente página se representan los valores tal y como deben quedar y pueden usar la imagen como guía para rellenar la forma.

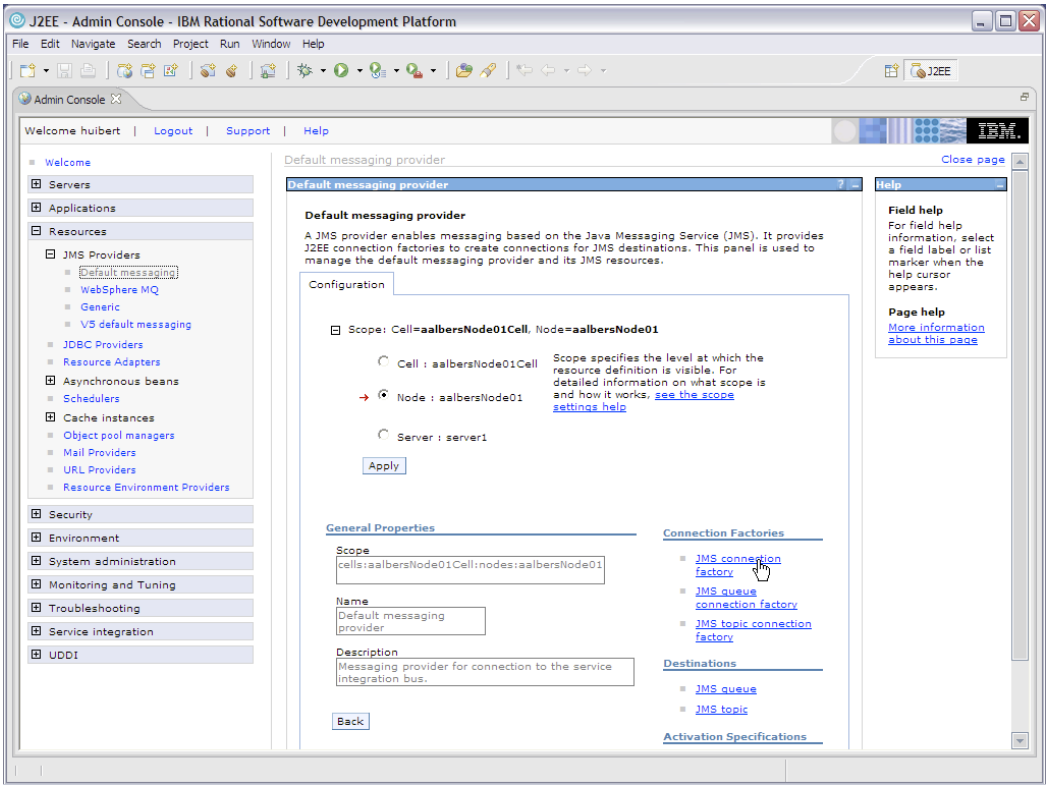
Cuando terminen pulse el botón “Ok”.


Welcome huibert | Logout | Support | Help
IBM.
Welcome
Servers
Applications
Resources
JMS Providers
Default messaging
WebSphere MQ
Generic
V5 default messaging
JDBC Providers
Resource Adapters
Asynchronous beans
Schedulers
Cache instances
Object pool managers
Mail Providers
URL Providers
Resource Environment Providers
Security
Environment
System administration
Monitoring and Tuning
Troubleshooting
Service integration
UDDI
Default messaging provider
Close page
Default messaging provider
Default messaging provider > JMS connection factory > New
A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.
Configuration
General Properties
Administration
* Scope
cells:aalbersNode01Cell:nodes:aalbersNode01
* Name
MDBSampleSIBJMSConnectio
* JNDI name
Sample/JMS/MDB/CF
Description
Category
Connection
* Bus name
WAS6Bus
Target
Target type
Bus member name
Target significance
Preferred
Target inbound transport chain
Provider endpoints
localhost:7276
Connection proximity
Bus
Durable Subscription
Client identifier
Durable subscription home
Quality of Service
Nonpersistent message reliability
Express nonpersistent
Persistent message reliability
Reliable persistent
Advanced Messaging
Read ahead
Default
Temporary queue name prefix
WAS6
Temporary topic name prefix
WAS6
Share durable subscriptions
In cluster
Advanced Administrative
Component-managed authentication alias
(none)
Log missing transaction contexts
Manage cached handles
Share data source with CMP
XA recovery authentication alias
(none)
Apply
OK
Reset
Cancel
Related Items
J2EE Connector Architecture (J2C) authentication data entries
Buses
Help
Field help
The list of comma separated endpoints used to connect to a bootstrap server.
Page help
More information about this page

Con esto hemos terminado de crear nuestra “JMS connection factory”. Solamente le falta guardar los cambios realizados a la configuración del servidor. Cuando lo haga debe tener ante usted una pantalla como la que se muestra a continuación.



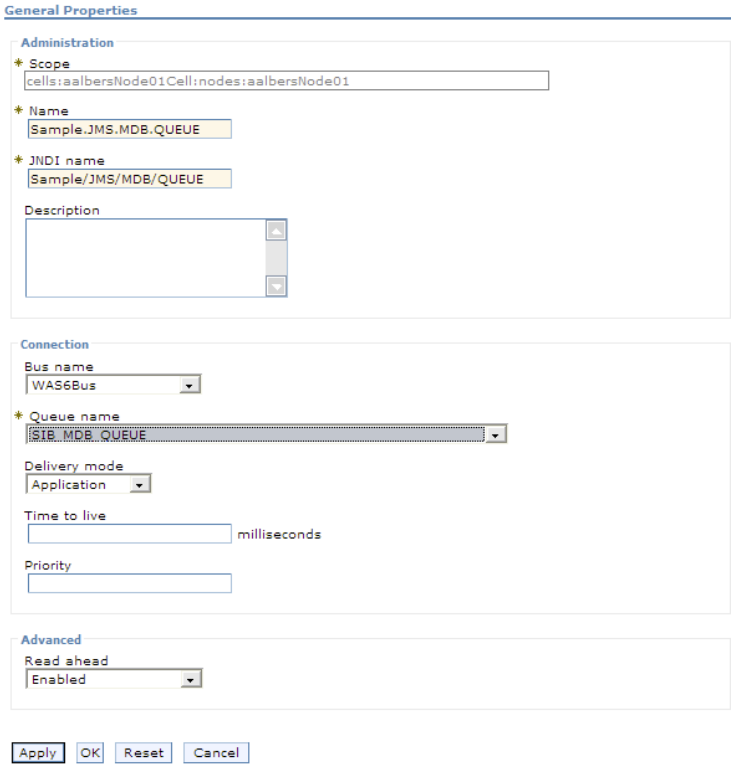
Ahora debemos crear una cola JMS (”JMS Queue”). Esta cola JMS es el interfaz que permite a JMS conectarse con la cola que creamos anteriormente. Para ello regrese a la pantalla de “Default messaging provider” y ahora pulse la opción “JMS Queues” bajo “Destinations”.



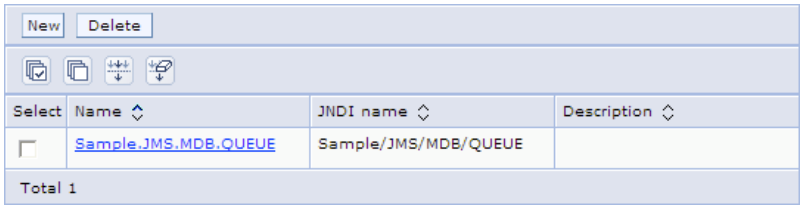
Pulse el botón “New” para crear la nueva “JMS Queue”.

Ahora rellene la forma con los siguientes datos. La forma debe quedar tal y como se muestra a continuación.

```
Name:        Sample.JMS.MDB.QUEUE
JNDI Name:   Sample/JMS/MDB/QUEUE
Bus Name:    WAS6Bus
Queue Name:  SIB_MDB_QUEUE
```

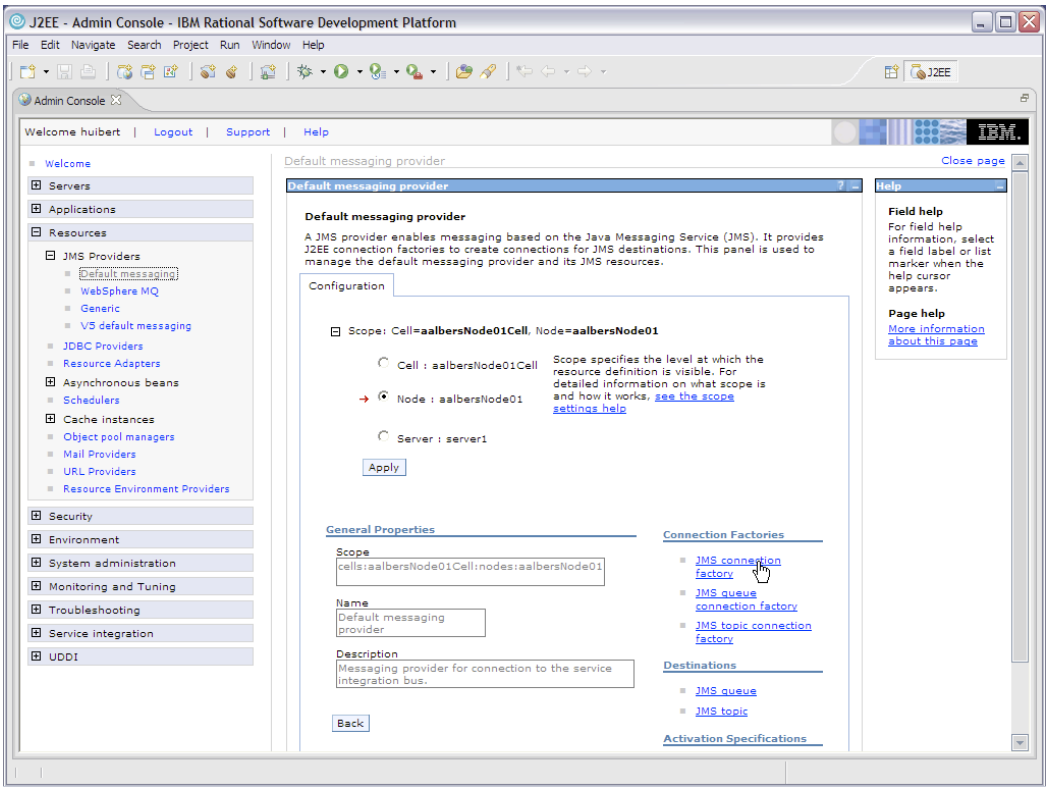


Pulse “Ok” para continuar. Guarde los cambios realizados a la configuración del servidor. Ante Vd debe tener una pantalla igual a la que se representa a continuación.



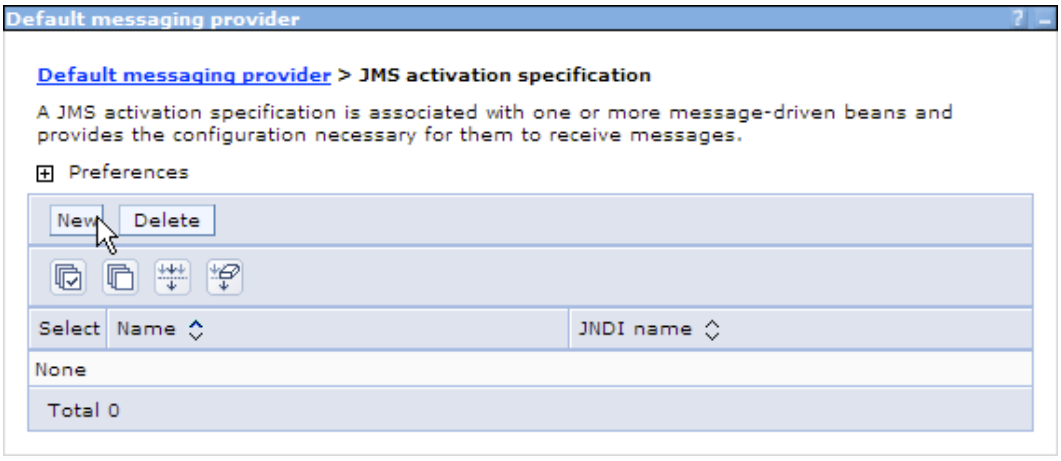
Ahora ya solo nos queda por crear lo que se conoce como un “JMS Activation specification” para la cola JMS que acabamos de crear.

Para ello debemos regresar una vez más a la pantalla de “Default messaging provider”.



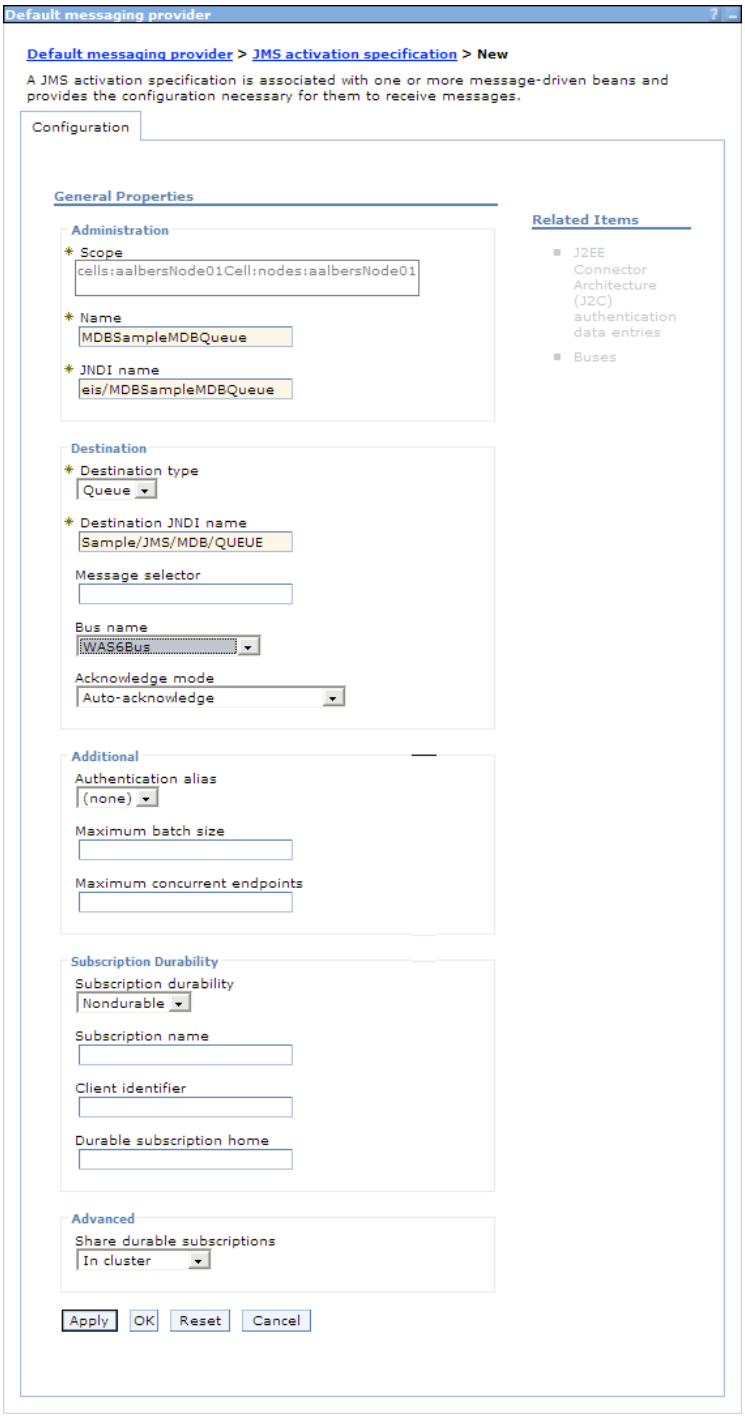
Haga click sobre la opción “JMS activation specification” bajo “Activation specification”.

Ahora haga click sobre el botón de “New” para crear una nueva “JMS activation specification”.

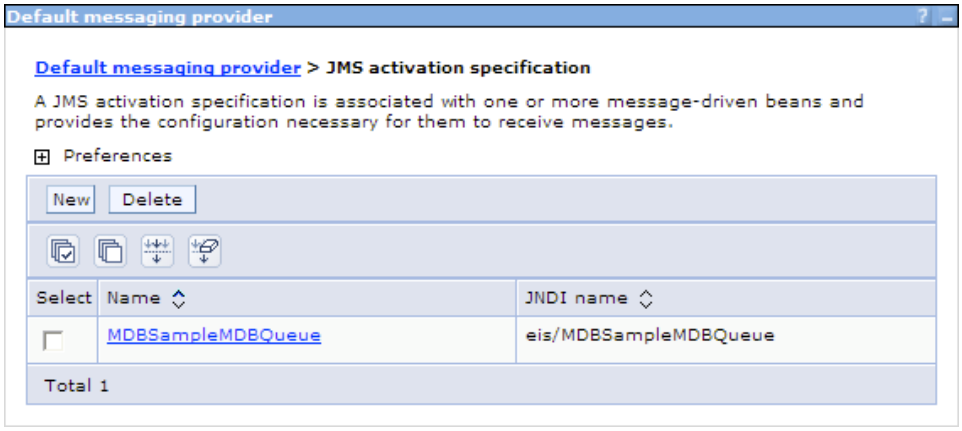


Estos son los valores que debe introducir en la forma:

```
Name:                    MDBSampleMDBQueue
JNDI Name:               eis/MDBSampleMDBQueue
Destination Type:        Queue
Destination JNDI Name: Sample/JMS/MDB/QUEUE
```



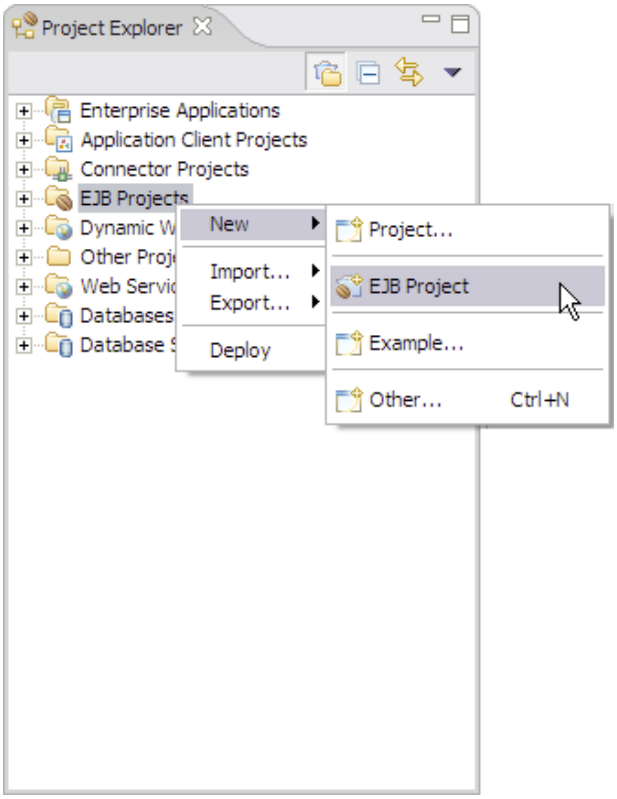
Guarde los cambios realizados a la configuración del servidor. Ahora ante Vd. debe aparecer la siguiente pantalla.



Esto es todo, hemos terminado de configurar el servidor.

Segunda parte - Desarrollo del Message Driven Bean

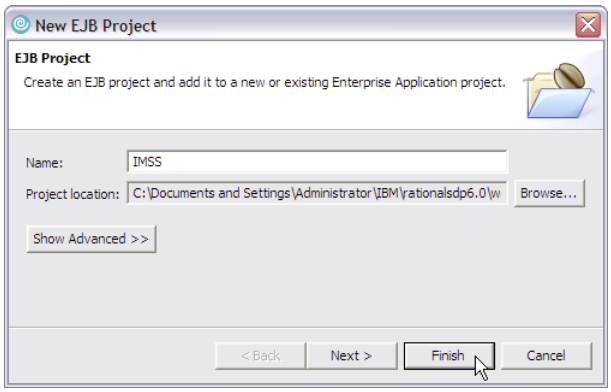
El objetivo de este laboratorio no es crear un MDB complejo. Lo que queremos es asegurarnos de que nuestra configuración es correcta y que los mensajes enviados a la cola por la aplicación que produce los mensajes sean consumidos de manera automática por un MDB. Por esta razón nuestro MDB será realmente muy sencillo y solo se encargará de escribir a la consola del servidor de aplicaciones un mensaje fijo que nos muestre que el mensaje ha sido recibido.



Lo primero que debemos hacer es crear un proyecto J2EE (.ear). En lugar de hacerlo de manera explícita, vamos a crear un proyecto EJB, lo cual tendrá como consecuencia la creación automática de la aplicación empresarial.

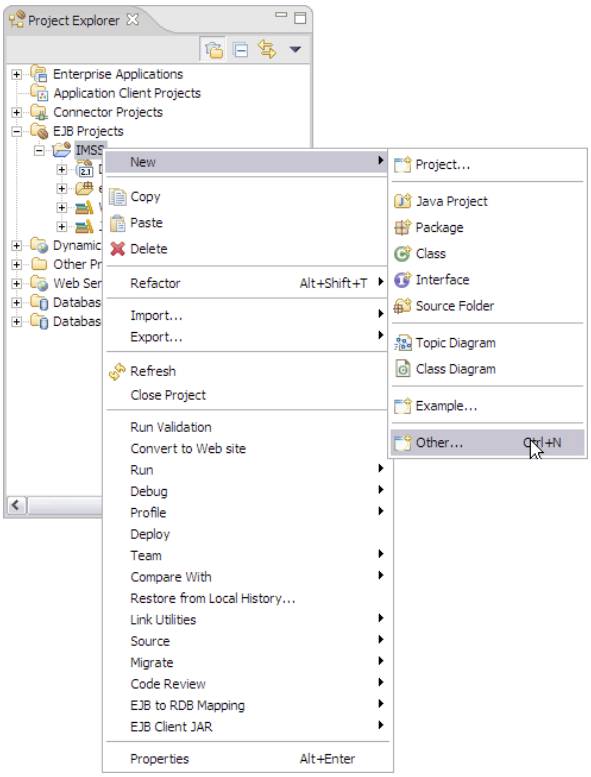
Haga right-click sobre “EJB Projects” en la ventana de “Project Explorer”. En el menú contextual que aparece, seleccione la opción “New >> EJB Project”.

En la siguiente pantalla seleccione un nombre para su proyecto. Del nombre que elija para el proyecto de EJBs se derivará el nombre de la aplicación empresarial.



Pulse el botón de “Finish” para crear el proyecto.

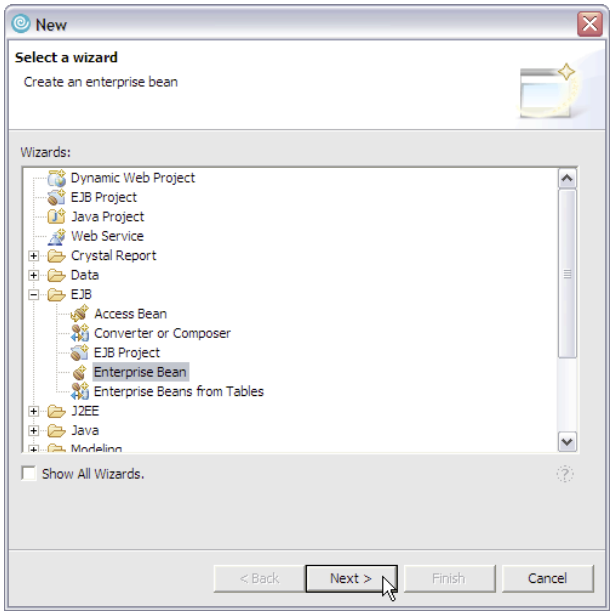
Ahora podemos crear el Message Driven Bean. Este tipo de EJB es muy distinto a los otros dos tipos de EJBs (Session beans y Entity beans). La diferencia estriba en que no tenemos que escribir un cliente que los ubique e invoque. Los MDBs se ejecutan de manera automática cuando llega un mensaje a la cola a la que están ligados. De hecho, los MDB solo tienen un método interesante, onMessage, que es el que el servidor de aplicaciones invoca cuando un mensaje llega a la cola. Por lo tanto, los programadores solo tienen que implementar ese método.



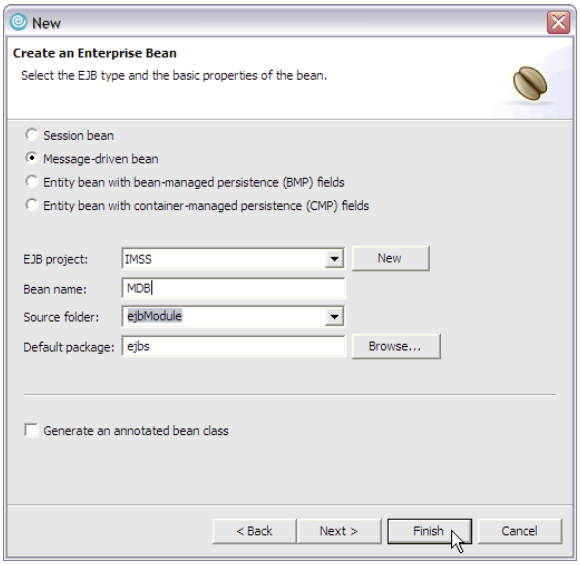
No hay un wizard específico para crear un Message Driven Bean, por lo que usaremos el que permite crear cualquier tipo de Enterprise Java Beans.

Haga right-click sobre el nombre del proyecto de EJBs y del menú contextual seleccione “New >> Other”.

En la siguiente ventana elija Enterprise Bean en la carpeta EJB.



Pulse “Next >” para continuar.



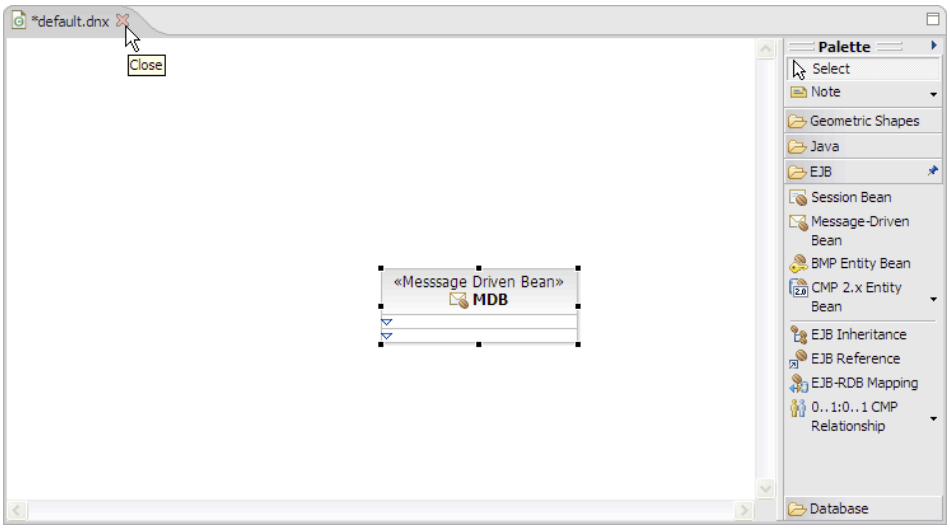
Seleccione “Message driven bean” de la lista de tipos de EJBs.

Elija un nombre para el bean. En este ejemplo usaremos el nombre “MDB”.

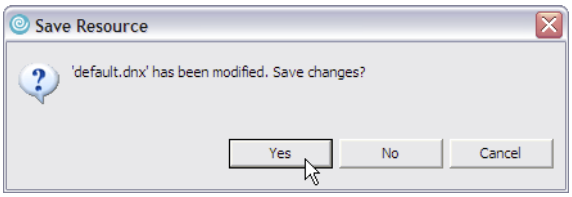
Asegúrese de que el EJB se vaya a crear en el proyecto correcto.

Pulse el botón “Finish” para crear el bean.

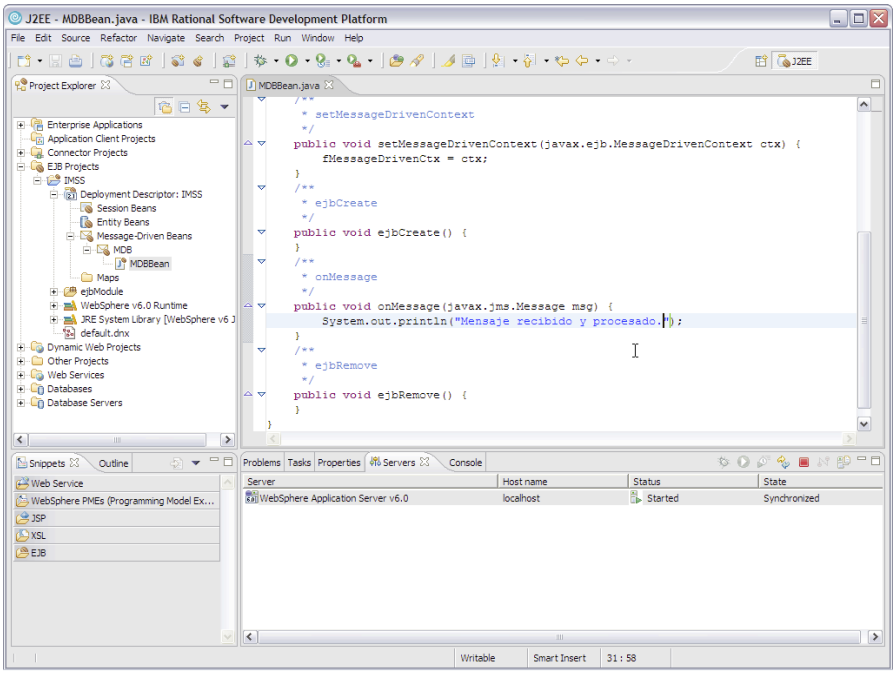
Como parte del proceso de creación del EJB, RAD 6.0 crea de manera automática un modelo visual UML. Esto es algo que no vamos a necesitar en este proyecto. Por lo tanto, podemos cerrar la ventana haciendo click sobre la pestaña del modelo.



Como el modelo no ha sido guardado, aparece la siguiente ventana.



Pulse el botón “Yes” para terminar. El wizard creó todos los archivos necesarios, incluyendo el más importante, MDBBean.java que contiene el código del MDB y que tenemos que editar para agregarle la funcionalidad necesaria.



Agregue la siguiente línea dentro del método onMessage(javax.jms.Message msg):

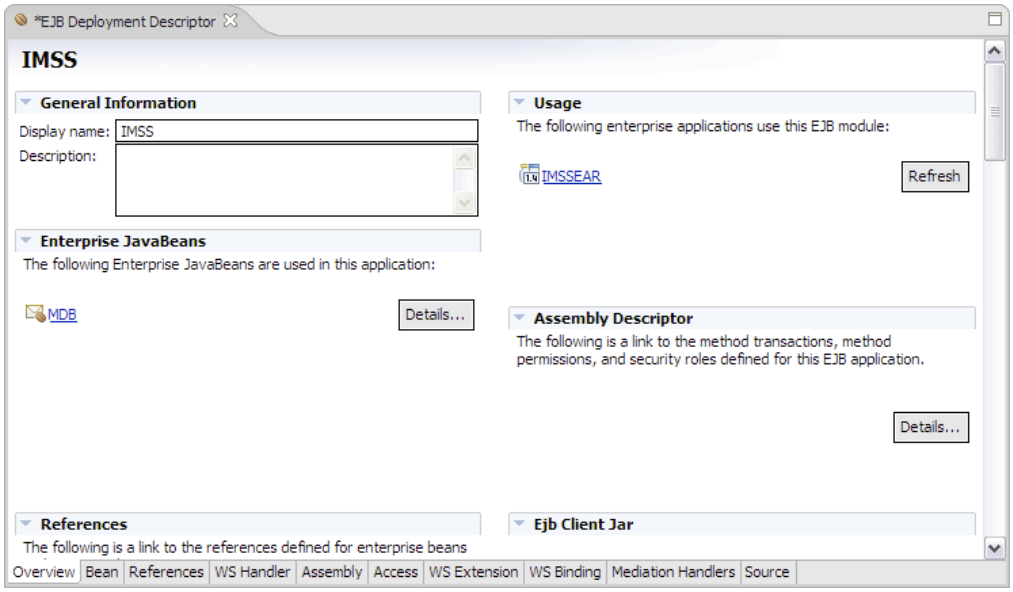
```
System.out.println("Mensaje recibido y procesado");
```

Guarde los cambios realizados al código pulsando Ctrl-S. Si lo desea puede cerrar la pestaña.

De esta manera, cuando un mensaje sea depositado en la cola, se creará automáticamente una instancia del MDB que consumirá el mensaje y escribirá el texto " Mensaje recibido y procesado" a la consola.

Sin embargo, para que esto pase tenemos que ligar el MDB al "Activation specification" que creamos en la primera parte y a la cola de la que se van a recuperar los mensajes.

Esta configuración se hace en el "EJB Deployment Descriptor". Haga doble click sobre el archivo "Deployment Descriptor: IMSS" para editarlo. Cuando se abre muestra la pestaña de "Overview".

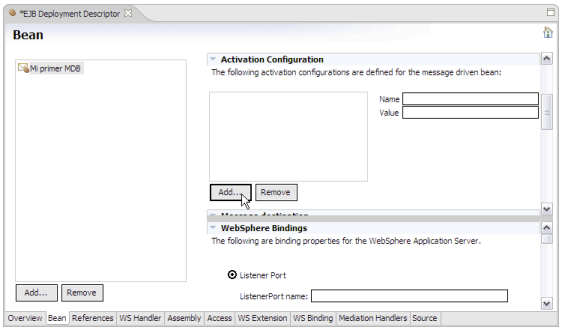


Haga click sobre el nombre del EJB que acabamos de crear ("MDB"). Esto nos lleva a la pestaña "Bean" en la que haremos los cambios que requerimos.

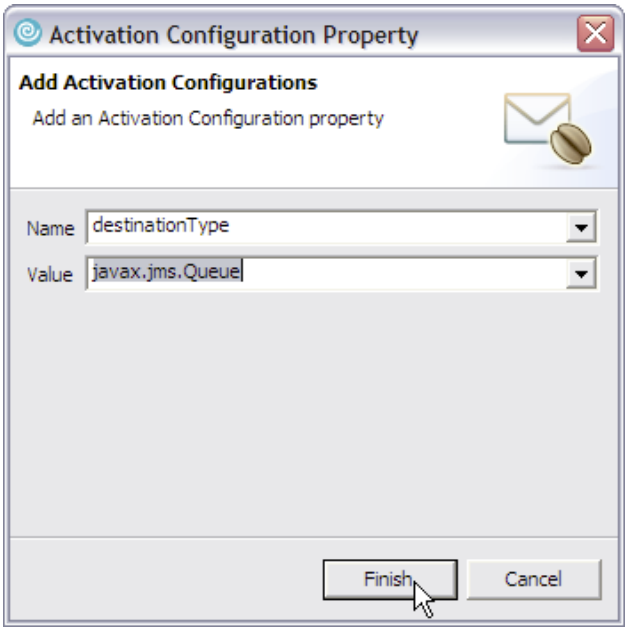
Si lo desea puede poner un nombre más fácil de recordar al bean, asi como una descripción. Nosotros usaremos "Mi primer MDB" como "Display name". Sin embargo eso no tiene ningún efecto práctico ya que lo que realmente utilizamos para referenciar un EJB es su nombre JNDI.

En el apartado de Message-Driven Destination seleccione "javax.jms.Queue" como "Destination type".

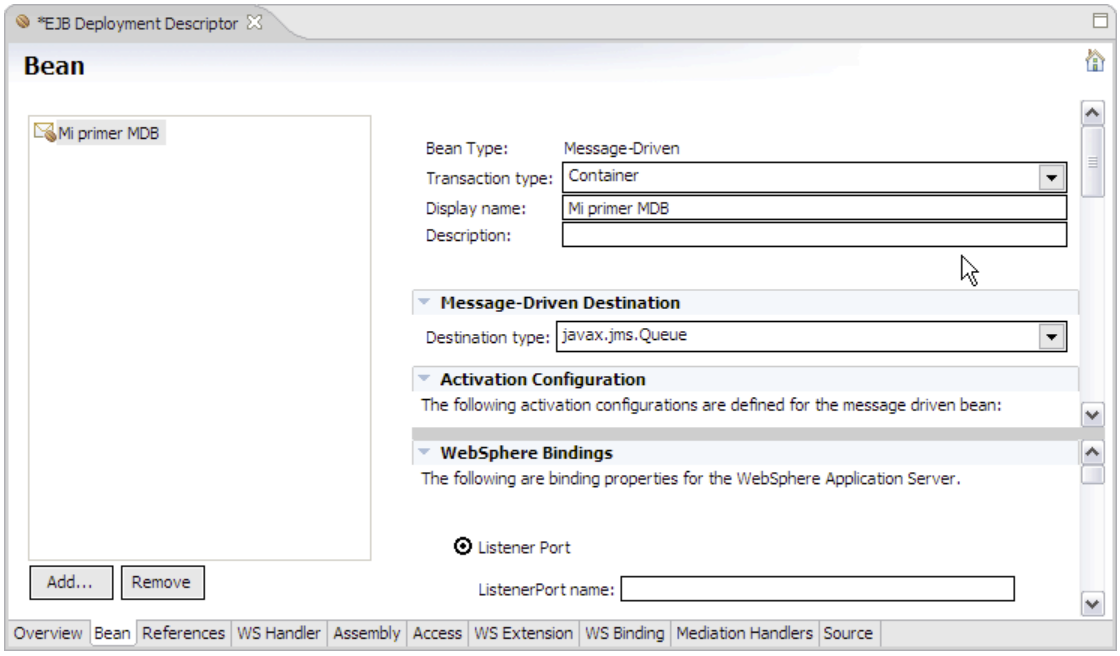
Más abajo, en "Activation Configuration" pulse el botón de "Add" para agregar una configuración.



Seleccione los valores que se muestran en la siguiente ilustración y pulse “Finish”.



La pantalla, con los cambios que ha realizado hasta ahora, se debe ver muy similar a la que se muestra a continuación.



Ahora, bajo WebSphere Bindings, seleccione “JCA Adapter” y utilice los siguientes valores:

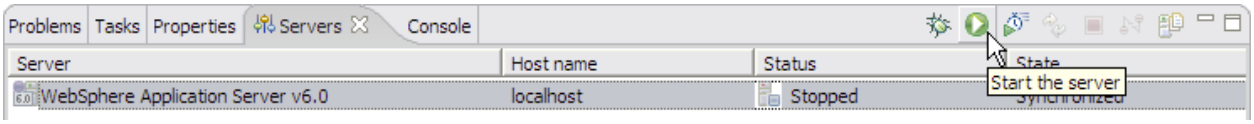
```
ActivationSpec JNDI name: eis/MDBSampleMDBQueue

Destination JNDI name: Sample/JMS/MDB/QUEUE
```

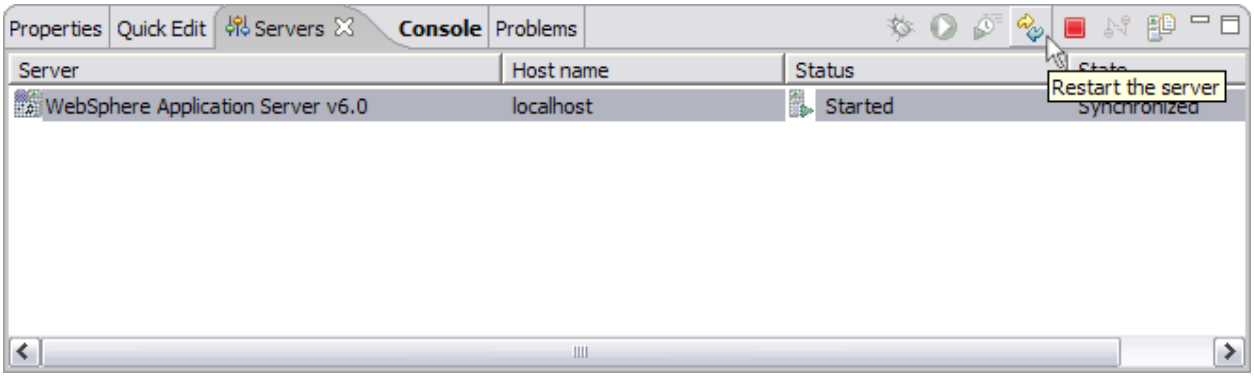
Pulse Ctrl-S para guardar los cambios. El MDB está listo para empezar a procesar mensajes, una vez que haya reiniciado el servidor, tal y como lo mencionábamos anteriormente.

Para reiniciar el servidor, vaya a la pestaña de “Servers” y seleccione el servidor de aplicaciones con el que estamos trabajando.

Si el servidor está detenido, arránquelo como se muestra a continuación.

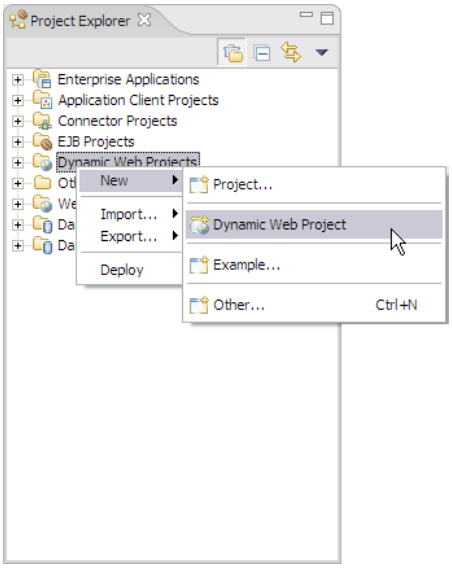


Si el servidor está levantado, reinícielo, haciendo click sobre el ícono de “Restart” tal y como se muestra en la siguiente ilustración.

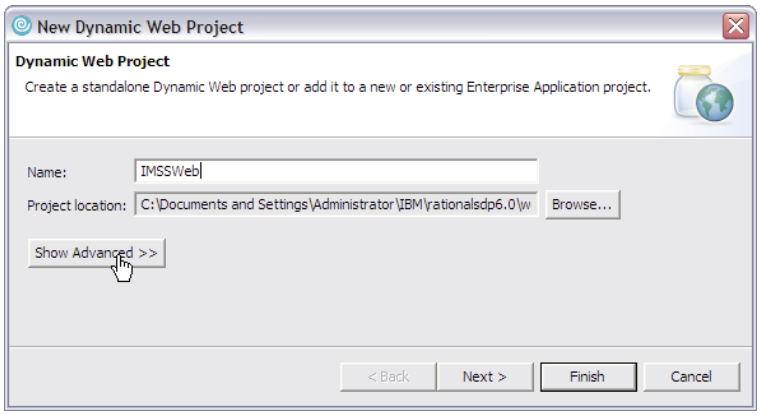


Tercera parte - Desarrollo del servlet y del JSP
En esta parte del laboratorio vamos a crear un simple JSP con una forma para capturar los datos que se pretenden publicar a la cola.

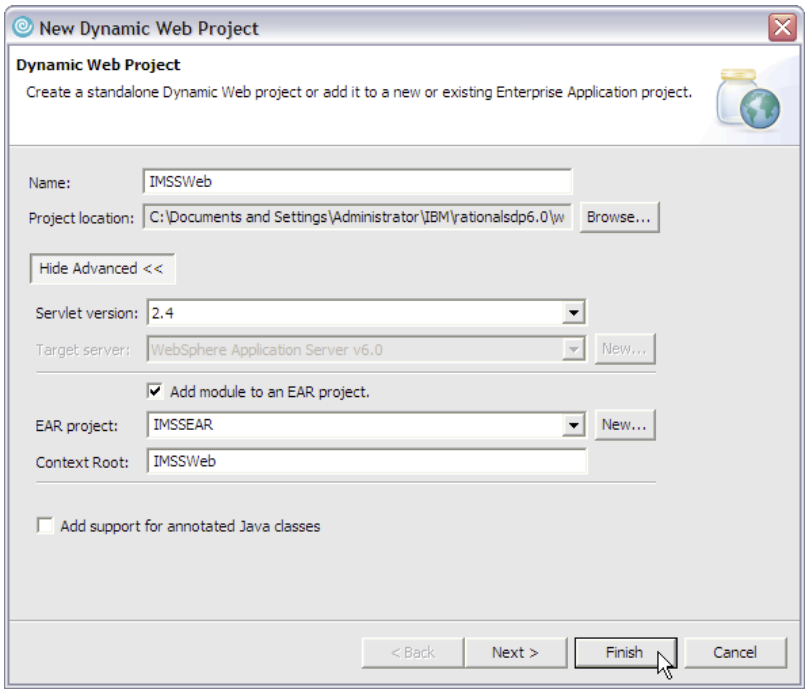
Para poder manejar JSPs dentro de nuestro proyecto J2EE necesitamos crear un Proyecto de Web Dinámico (”Dynamic Web Project”). Esto es muy sencillo, simplemente haga right-click sobre “Dinamic Web Projects en la ventana de “Project Explorer” y seleccione la opción “New >> Dynamic Web Project” del menú contectual, tal y como se muestra a continuación.



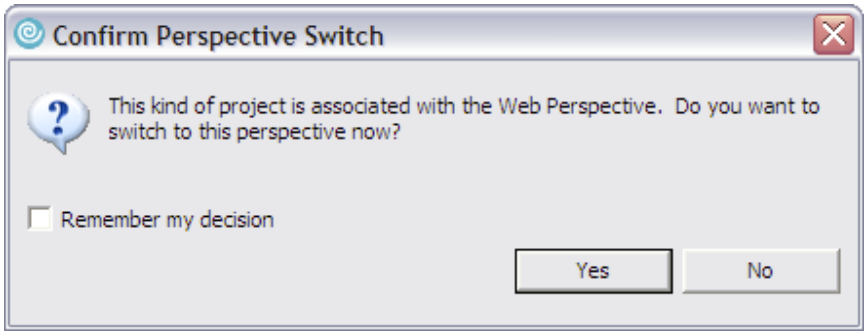
Al igual que cada vez que creamos un nuevo proyecto, el primer paso consiste en ponerle un nombre. Para evitar conflictos con los otros proyectos que ya creamos, lo nombraremos “IMSS-Web”.



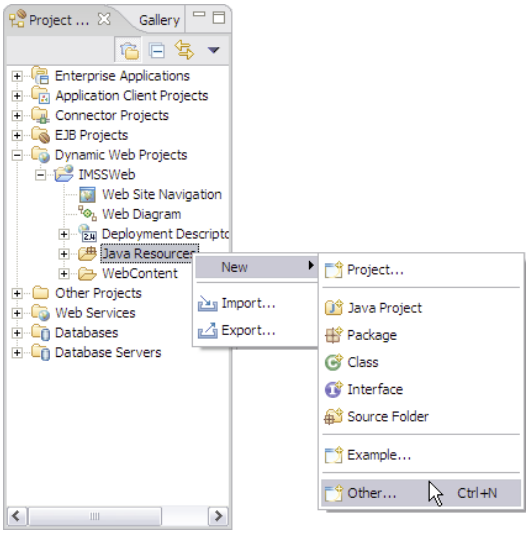
Ahora pulse el botón “Show Advanced”. Esto es muy importante ya que debemos asegurarnos que el nuevo proyecto será parte del existente “IMSSEar” en lugar de crear otro proyecto J2EE.



Al crear un nuevo proyecto Web, RAD pregunta si desea cambiar a la perspectiva de Web.



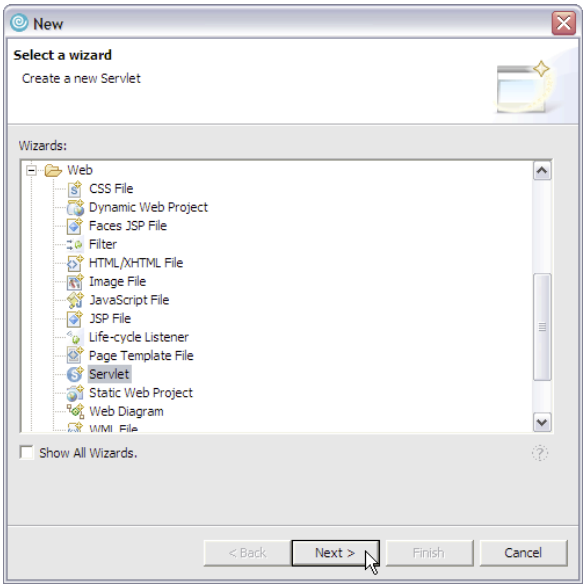
Como vamos a editar una página web usando el editor de JSPs, pulse “Yes” para continuar.



Ahora vamos a crear el servlet que publicará el mensaje a la cola y que será invocado desde una JSP que desarrollaremos un poco más adelante.

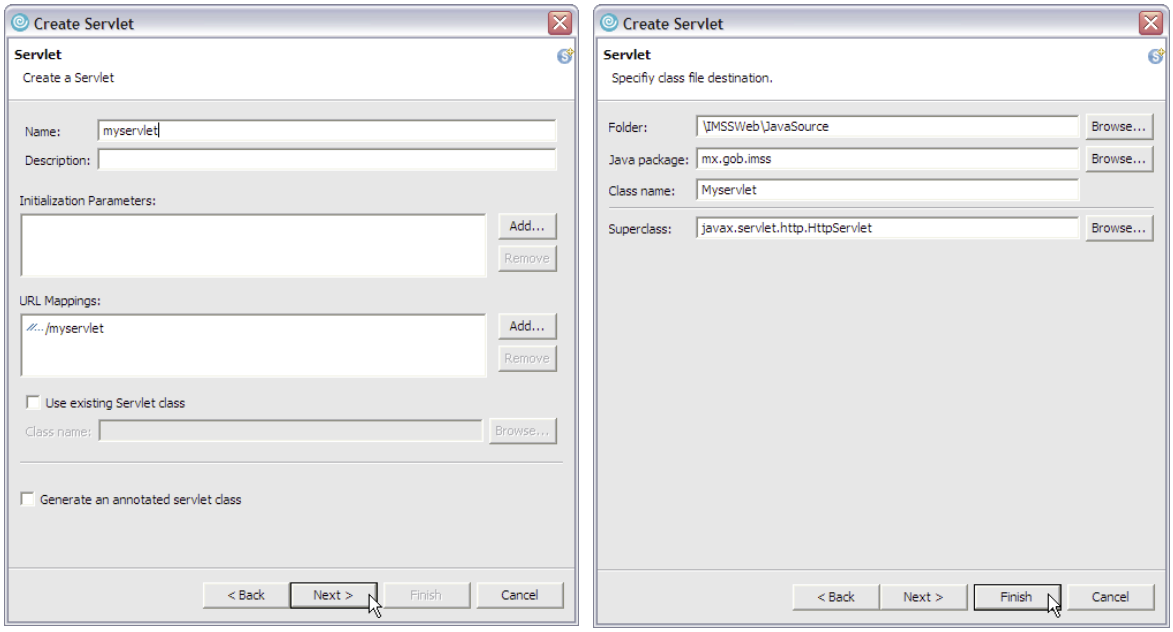
Vamos a utilizar un Wizard para crear el servlet, lo que nos evitará tener que modificar manualmente el descriptor web.xml.

Haga right-click sobre Java Resources y seleccione “New >> Other...” del menú contextual.



Seleccione “Servlet” de la carpeta “Web” y pulse “Next >”.

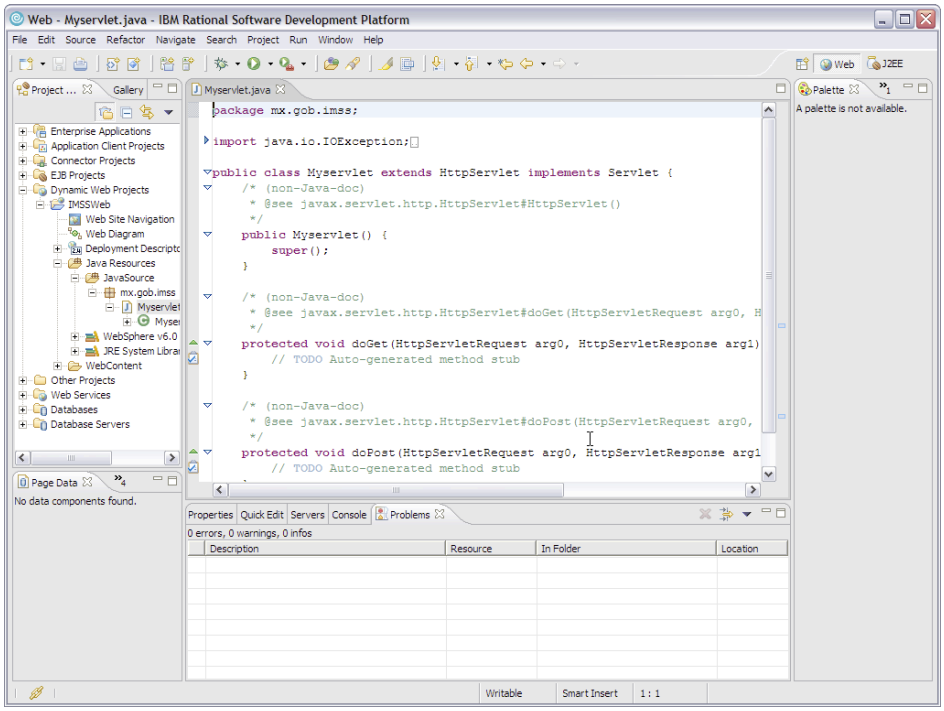
Vamos a crear un nuevo servlet que llamaremos “myservlet”. Pulse el botón “Next” para continuar.



Como podrá observar, el nombre del servlet solo se usa tal cual para definir el URL mapping. RAD modifica el nombre para que sea un nombre de clase aceptable (por ejemplo se asegura que empieza por una mayúscula). No olvide indicar un nombre de paquete válido para su clase porque si no lo hace la clase quedará en el “default package” y eso no es una buena idea. En este ejemplo, la clase residirá en el paquete mx.gob.imss.

Pulse el botón “Finish” para terminar de generar el servlet.

RAD ha creado un servlet que aún no hace nada. Para agregarle funcionalidad debemos editar el archivo “Myservlet.java”, tal y como se muestra a continuación.



Los servlets tienen dos métodos que generalmente hacen lo mismo, doGet y doPost. Para evitar duplicar el código vamos a crear un método service que será invocado tanto por doGet como por doPost.

Agregue el siguiente código a su servlet:

```
import javax.jms.*;
import javax.naming.*;
public void service(HttpServletRequest arg0, HttpServletResponse arg1) {
  Context ctx = null;
  ConnectionFactory cf = null;
  Destination requestDest = null;
  Connection connection = null;
  Session session = null;
  MessageProducer producer = null;
  TextMessage requestMessage = null;

  try {
    ctx = new InitialContext();
    cf = (ConnectionFactory)ctx.lookup("java:comp/env/jms/mdb/ConnectionFactory");
    requestDest = (Destination)ctx.lookup("java:comp/env/jms/mdb/Queue");
    connection = cf.createConnection();
    connection.start();
    session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
    producer = session.createProducer(requestDest);
    requestMessage = session.createTextMessage(arg0.getParameter("Name"));
    producer.send(requestMessage);
    System.out.println("El mensaje fue mandado exitosamente...");
  } catch (NamingException e) {
    e.printStackTrace();
  } catch (JMSException e) {
    e.printStackTrace();
  }
  finally{
    if ( connection != null) {
      try {
        connection.close();
        connection = null;
      } catch (JMSException e1) {
        e1.printStackTrace();
      }
    }
  }
}
```

Ahora agregue la siguiente línea de código dentro de los métodos doGet y doPost:

service(arg0, arg1);

Ahora, cuando el servlet sea invocado, va a tomar el valor del parámetro “Name” y lo va a utilizar para generar un mensaje que se va a publicar a la cola que definimos en la primera parte, utilizando los APIs de JMS.

Guarde los cambios realizados a la clase del servlet haciendo Ctrl-S. Si lo desea puede cerrar la pestaña con el código de “Myservlet.java”.

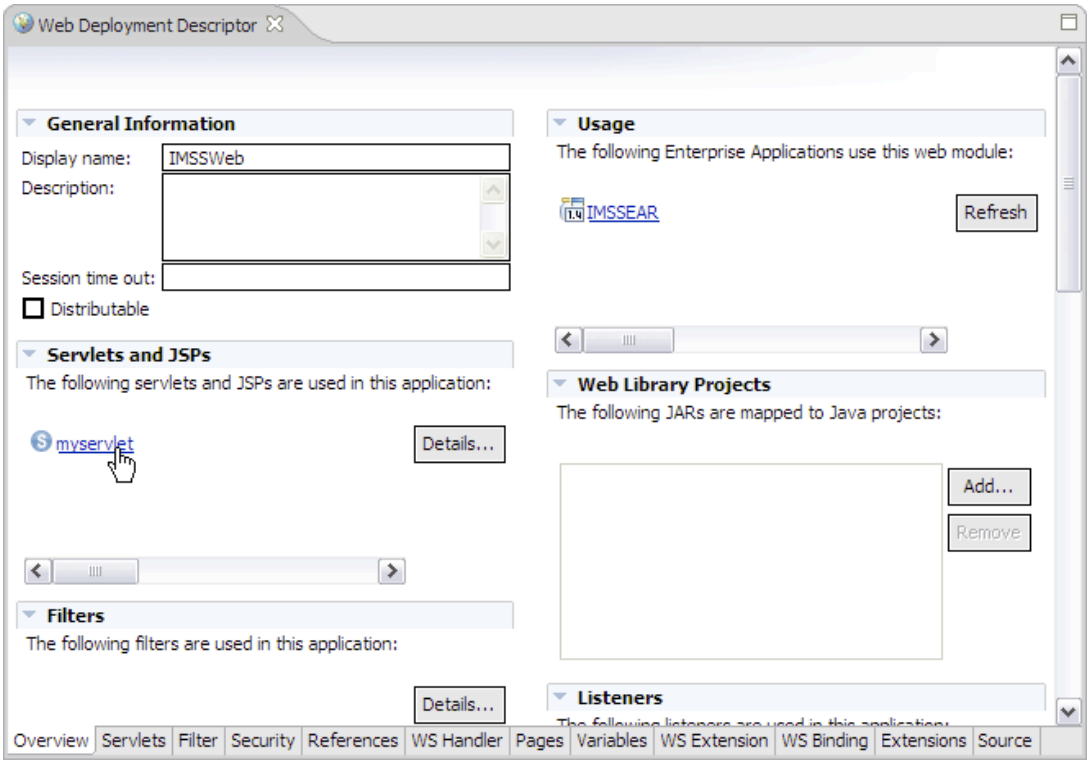
Es posible que se haya dado cuenta de algo raro en el código. Fíjese en las siguientes líneas de código:

```
cf = (ConnectionFactory)ctx.lookup("java:comp/env/jms/mdb/ConnectionFactory");
requestDest = (Destination)ctx.lookup("java:comp/env/jms/mdb/Queue");
```

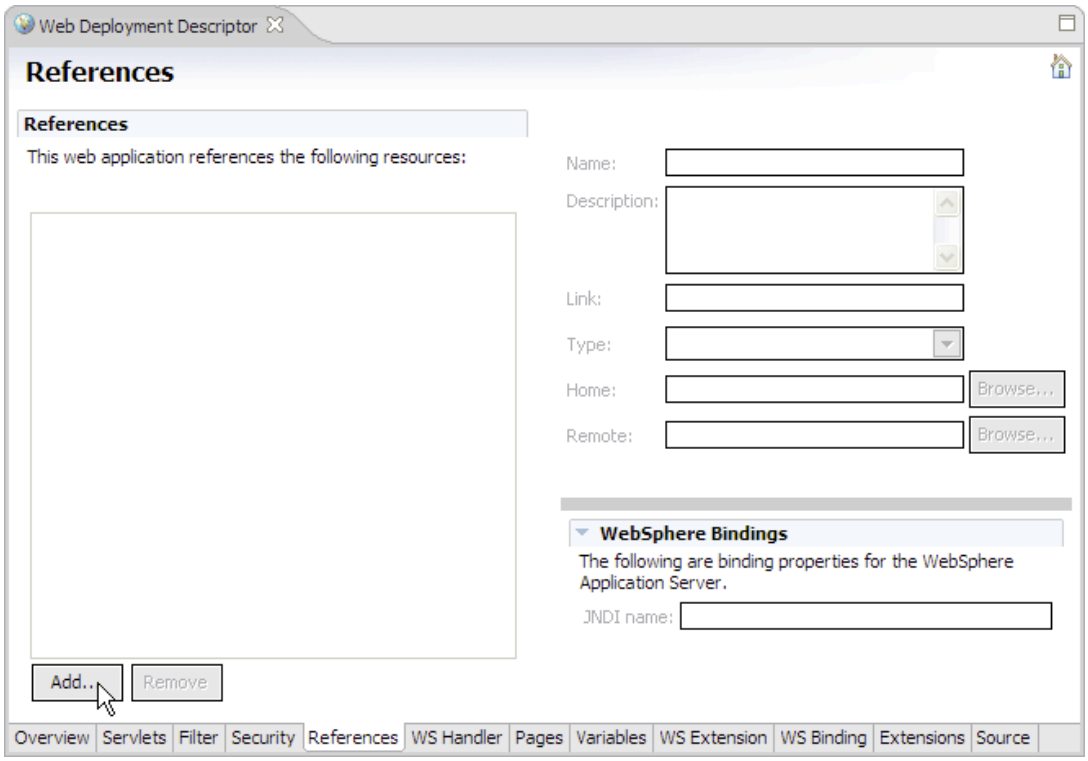
En ningún momento hemos definido jms/mdb/ConnectionFactory ni env/jms/mdb/Queue. En realidad, los nombres JNDI que definimos para la “ConnectionFactory” y la “Queue” fueron Sample/JMS/MDB/CF y Sample/JMS/MDB/QUEUE respectivamente. Para resolver esa discrepancia debemos ligar los nombres usados en nuestro programa con los que definimos en la primera parte. Para lograrlo debemos crear referencias.

Para realizar esta operación con componentes del Web Container (tales como servlets o JSPs) debemos modificar el descriptor web.xml. Este archivo se encuentra situado dentro del directorio WEB-INF situado dentro de WebContent en el proyecto de web dinámico.

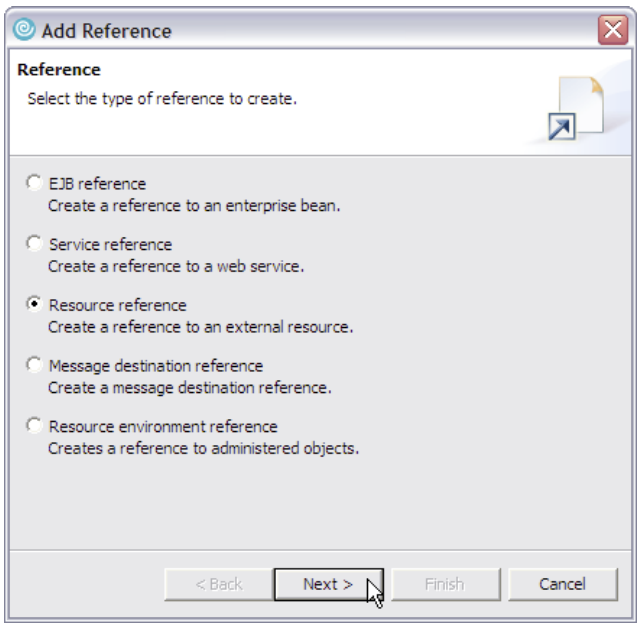
Abra el archivo web.xml haciendo doble-click sobre el mismo.



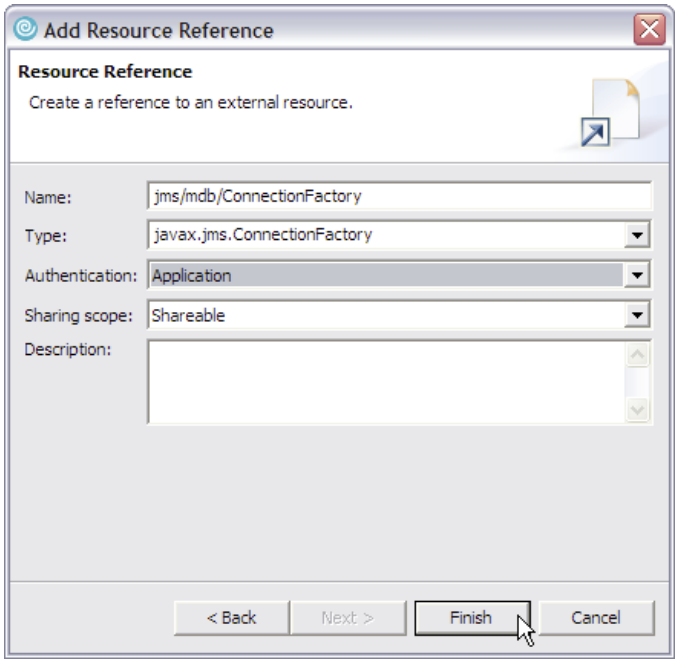
Para crear las referencias, debemos hacer click sobre la pestaña llamada “References”.



Haga click sobre el botón “Add...” para crear la primera referencia. El primer paso consiste en definir el tipo de objeto al que vamos a hacer referencia. El primer objeto con el que vamos a trabajar es un “ConnectionFactory”, de tipo javax.jms.ConnectionFactory.

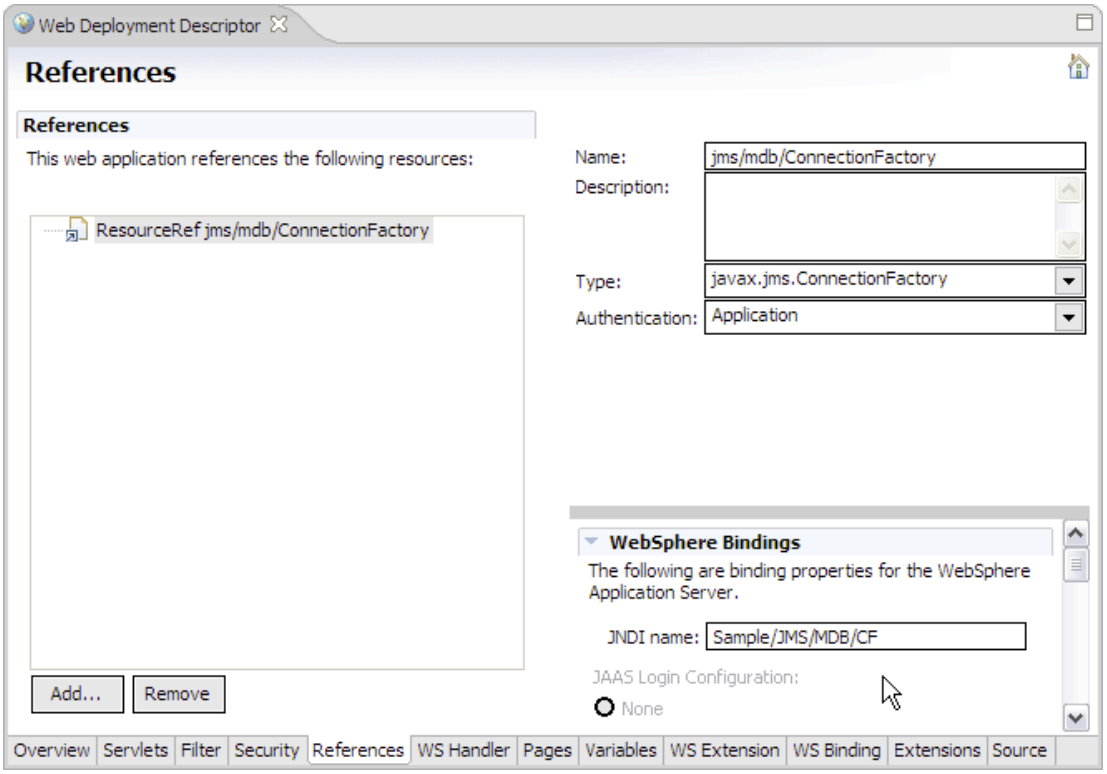


Seleccione “Resource reference” y pulse “Next >”.



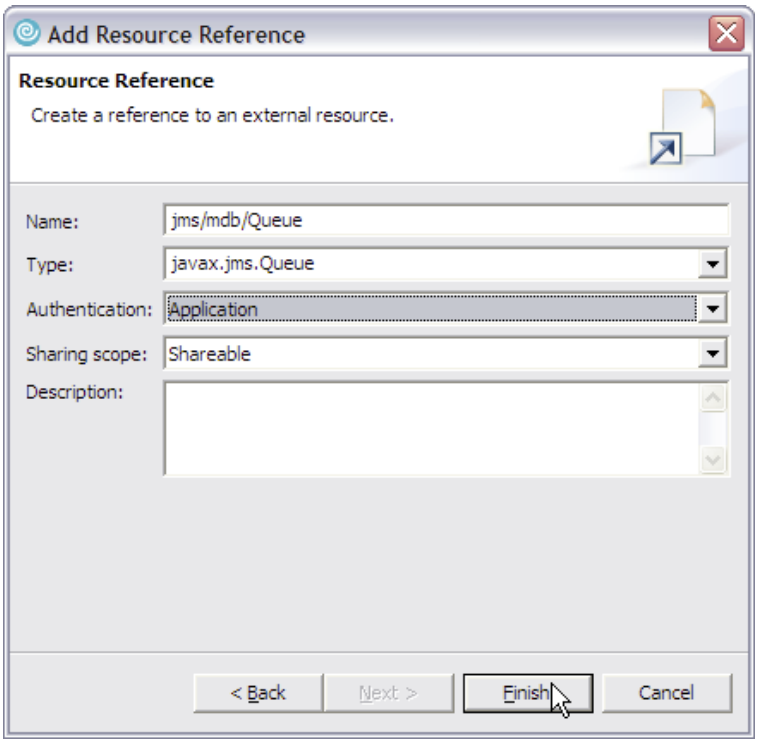
Copie los valores que aparecen en la ilustración precedente y pulse el botón “Finish”.

Ahora ponga el nombre JNDI al que hace referencia jms/mdb/ConnectionFactory, en este caso Sample/JMS/MDB/CF.

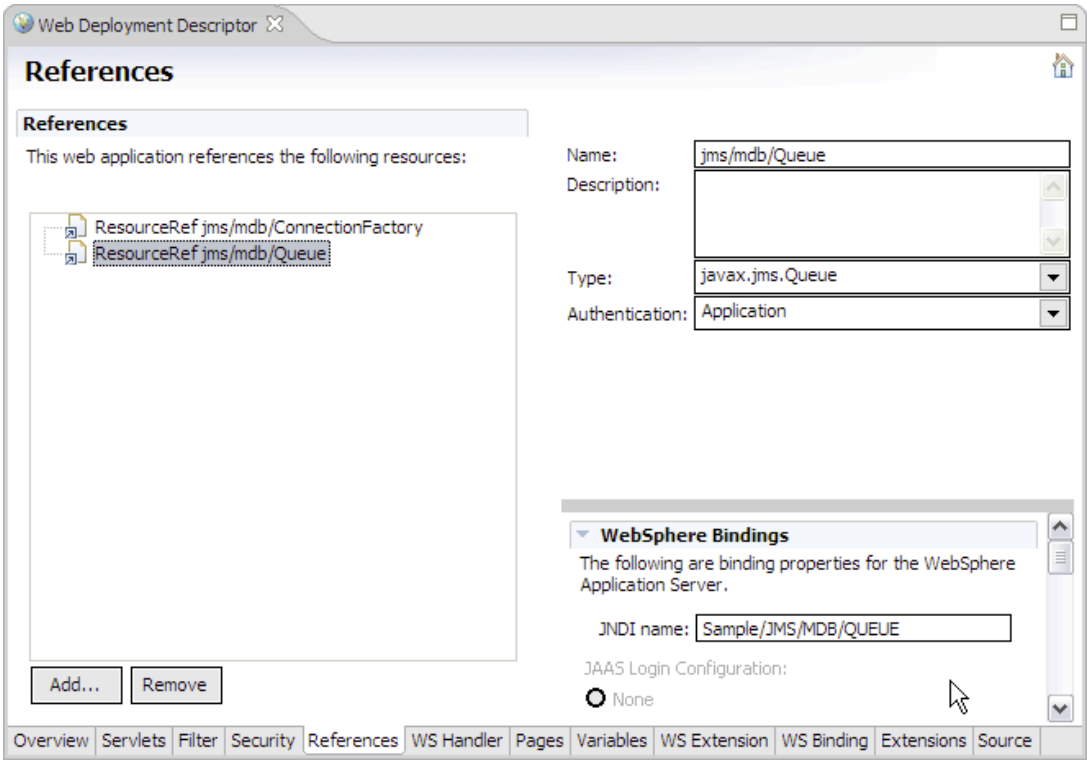


Haga Ctrl-S para guardar los cambios realizados al archivo web.xml.

Vamos a repetir la misma operación para la otra referencia que tenemos que declarar. Pulse el botón “Add...”. Rellene los campos tal y como se muestra a continuación.

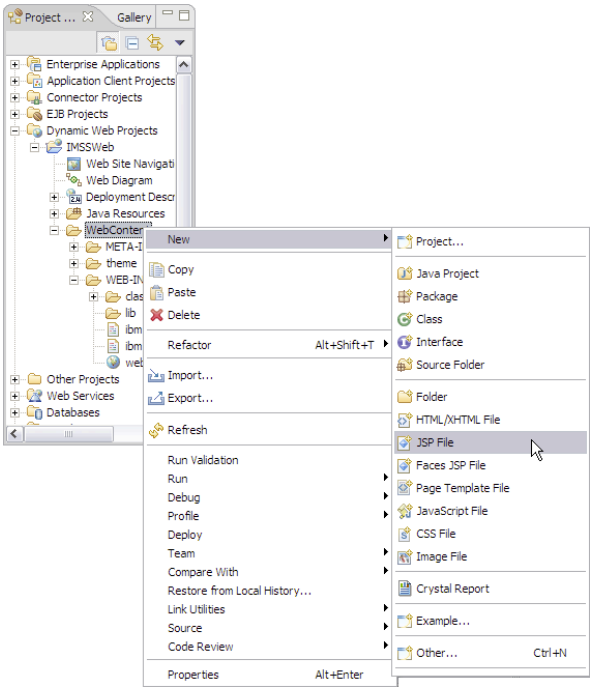


Pulse “Finish” para terminar.

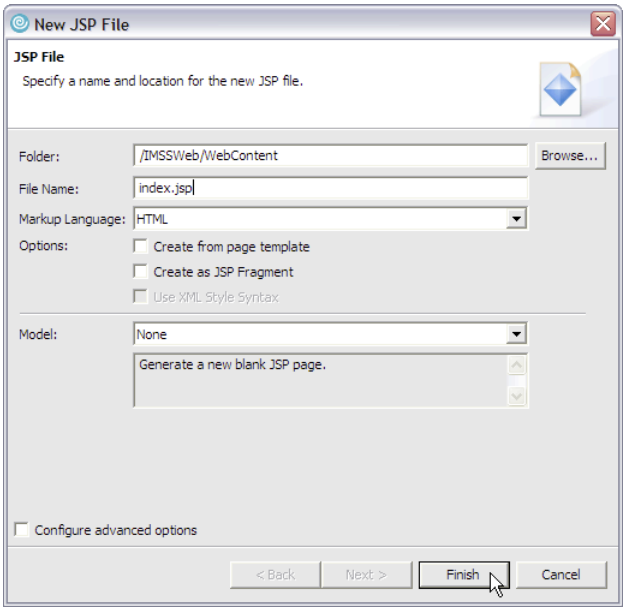


Agregue el nombre JNDI de la cola, Sample/JMS/MDB/QUEUE y pulse Ctrl-S para guardar los cambios realizados al archivo web.xml. Ya puede cerrar la pestaña si lo desea ya que la configuración de este archivo ha concluido.

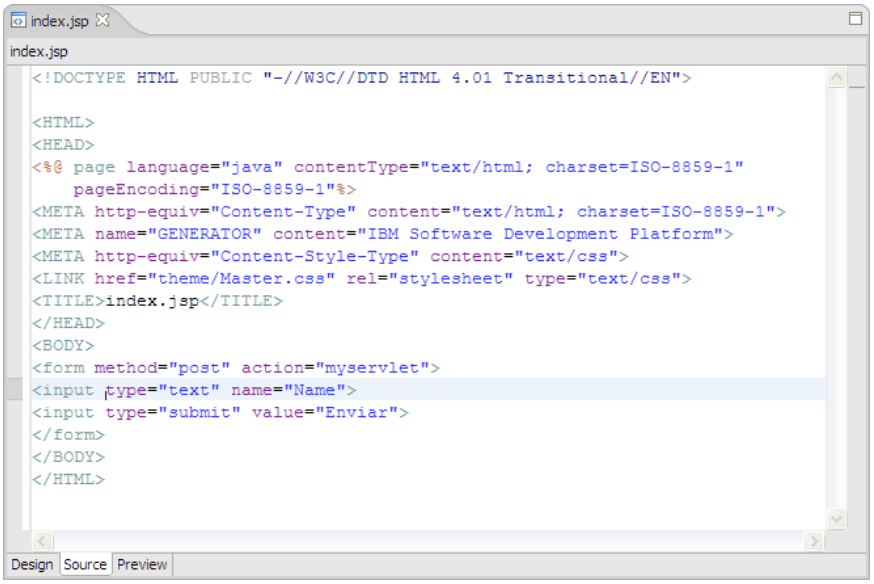
Ahora estamos listos para crear la JSP que invocará el servlet. Para ello vamos a utilizar un wizard. Haga right-click sobre WebContent y seleccione del menú contextual la opción “New >> JSP File”



El nombre de nuestra nueva página será index.jsp, tal y como se muestra a continuación.



Pulse el botón “Finish” para crear la nueva página.



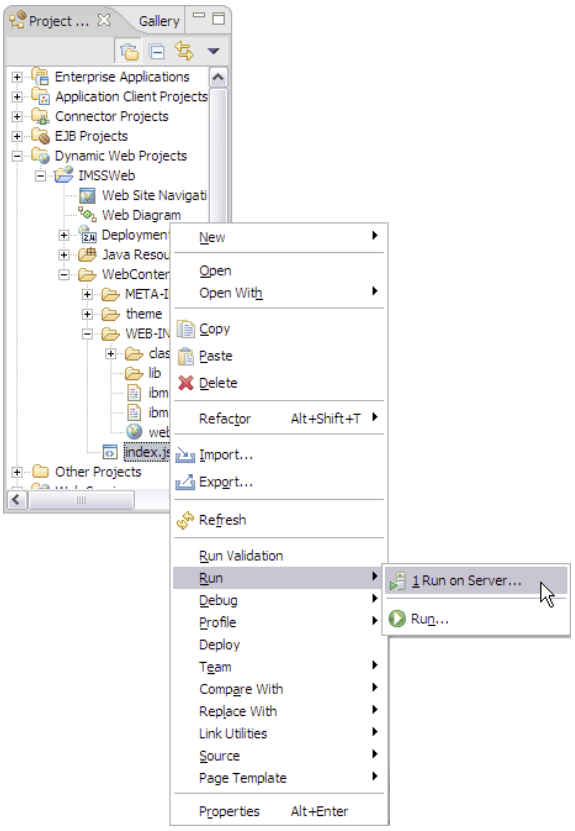
Modifique la página para que dentro del <BODY></BODY> se incluya la siguiente forma, tal y como aparece en la anterior pantalla.

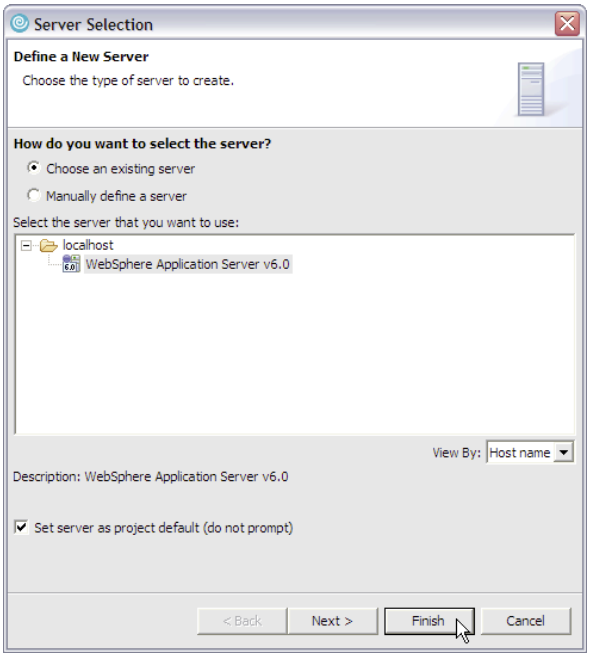
```
<form method=”post” action=”myservlet”>
<input type=”text” name=”Name”>
<input type=”submit” value=”Enviar”>
</form>
```

Pulse Ctrl-S para guardar los cambios realizados al JSP. Si lo desea puede cerrar la pestaña, ya no haremos más cambios a esta página.

Cuarta parte - Prueba del sistema

Todo está listo para realizar una prueba completa del sistema. Para lograrlo vamos a lanzar el servidor de aplicaciones y ejecutar la página “index.jsp”.

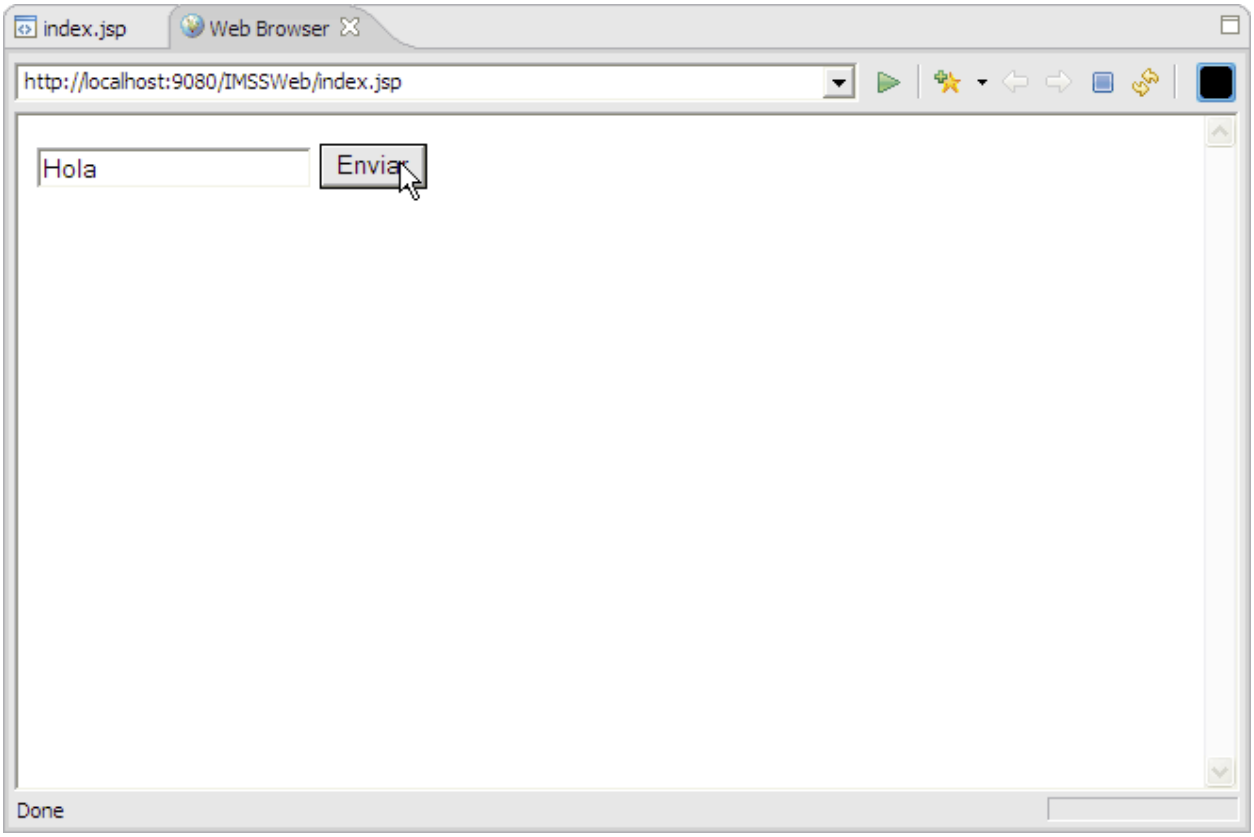




Marque la opción “Set server as project default (do not prompt)” para evitar que esta ventana aparezca cada vez que se quiera probar la aplicación.

Pulse el botón “Finish”. Si el servidor no está funcionando, RAD lo arranca automáticamente y publica la aplicación. RAD abre un browser en el que se pruede probar la aplicación tal y como se puede apreciar en la siguiente ilustración.

Escriba cualquier texto y pulse el botón “Enviar”.



Seleccione la pestaña “Console” para ver si el mensaje ha sido procesado correctamente o si se ha producido algún tipo de error.