Teil 3
WebSphere and Message Driven Beans

1 Überblick

1.1 Test Konfiguration

Dieses Tutorial zeigt, wie eine einfache EJB Message Driven Bean Anwendung mit Rational Application Developver 7.5 (kurz RAD7.5) erstellt und auf einem WebSphere Application Server 6.1 (kurz WAS6.1) deployed werden kann. Wir verwenden die gleiche RAD7.5 und WAS6.1 Konfiguration wie im letzten EJB Tutorial.

Die in diesem Tutorial zu entwickelnde Test-Konfiguration ist in Abbildung 1.1 wiedergegeben. Sie besteht aus einem Browser, einem WebSphere Application Server6.1, der unter Windows läuft, sowie einem Konsole Fenster.

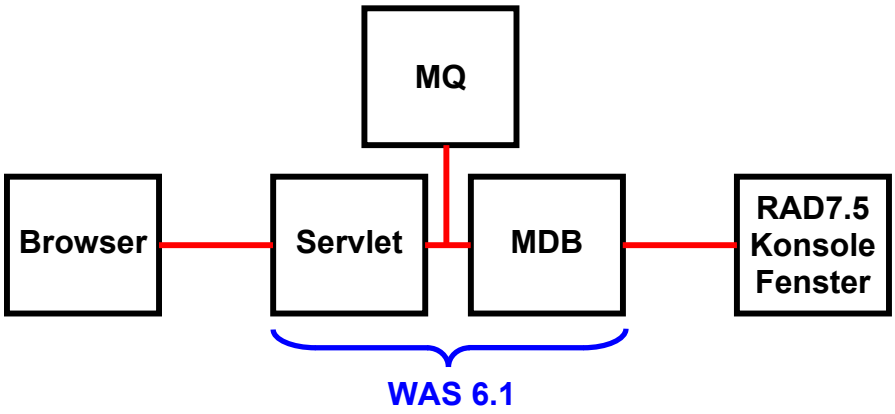


Abbildung 1.1

Der Browser übergibt einen Message Text an das Servlet. Dieses agiert als ein Klient für eine Message Driven Bean und deren Message Queue. Die MDB gibt den Text der Message in einem RAD7.5-Konsolenfenster wieder. Die integrierte Version von MQSeries verwaltet die Queue (siehe Band 1 Abschnitt 10.1.6, sowie Band 2, Abschnitt 15.4.5) .

1.2 Inhalt Übersicht

Selbst-Test

- **Könnten sie zuerst die Anwendung unter RAD 7.5 entwickeln und dann erst des WAS6.1 konfigurieren?**
- **Benutzen Sie für die Konfiguration von WAS6.1 die RAD7.5 Entwicklungsumgebung?**

2 JMS (Java Message Service)

Messaging ist ein Verfahren zur Kommunikation zwischen Softwarekomponenten oder zwischen Anwendungen. Ein Messaging-System ist eine Peer-to-Peer-Einrichtung: Ein Messaging-Client sendet Nachrichten an und/oder empfängt Nachrichten von anderen Klienten.

Messaging ermöglicht eine verteilte (distributed) Kommunikation, die lose gekoppelt ist. Eine Komponente sendet eine Nachricht an ein Ziel, und der Empfänger kann die Nachricht vom Ziel abrufen. Es ist nicht erforderlich, dass Sender und Empfänger gleichzeitig kommunizieren. Der Absender braucht nichts über den Zustand des Empfängers zu wissen, und der Empfänger muss nichts über den Absender kennen. Der Sender und der Empfänger brauchen nur zu wissen welches Nachrichtenformat benutzt, und welche Ziel Adresse verwendet wird. In dieser Hinsicht unterscheidet sich Messaging von eng gekoppelten Technologien wie Remote Method Invocation (RMI). Im Gegensatz zu E-Mail wird Messaging für die Kommunikation zwischen Software-Anwendungen oder Software-Komponenten verwendet, und implementiert eine garantierte Kommunikation unter ACID Bedingungen.

Der Java Message Service (JMS) definiert den Standard für eine zuverlässige Enterprise Messaging.
Die JMS API ist ein integraler Bestandteil der Java Enterprise Edition (JEE)-Plattform, und Anwendungsentwickler können Nachrichten mit Komponenten mit JEE APIs ("JEE Komponenten") zu verwenden.

Message Driven Beans (MDB) sind diejenigen Komponenten, die EJB-Systeme für asynchrone Kommunikation zugänglich machen. Hierzu wird der Java Message Service (JMS) verwendet. Diese Sorte von Beans wird z. B. häufig für die Kommunikation mit Legacy-Systemen genutzt. Auch für die Ausführung von klassischerweise asynchron auszuführenden Operationen, von deren Erfolg und Dauer die Performanz einer übergeordneten Anwendung nicht abhängen soll, bieten sich Message Driven Beans an.

Sinn und Zweck von Message Driven Beans ist es, Nachrichten asynchron abzuarbeiten. Dazu schickt ein Client, wie z.B. ein Session Bean oder ein Servlet, eine Nachricht in eine Queue. Die Queue übergibt die Nachricht an eine MDB und diese verarbeitet sie dann.

Der Java Message Service ist ein Satz von Java-APIs, der es ermöglicht:

- **Anwendungen zu erstellen**
- **Nachrichten zu senden**
- **zu empfangen**
- **zu lesen**

Für die Anwendung braucht man einen Provider, der die API umsetzt und somit den Dienst bereitstellt. WAS6.1 kann als ein derartiger Provider dienen und MDBs sind geeignet, die Dienstleistung zu erbringen. Die WAS6.1 Implementierung benutzt die integrierte Version von WebSphere MQ.

Das JMS API Programmiermodell besteht aus:

- **Connections (Verbindungen)**
- **Sessions**
- **Messages**
- **Message Producers**
- **Message Consumers**

Die JMS API definiert eine Reihe von Schnittstellen und die damit verbundene Semantik, die es Programmierern ermöglicht, Messaging-Komponenten in Java zu entwickeln, die eine Kommunikation mit anderen Messaging-Implementierungen ermöglichen.

Verwaltete (administered) Objekte sind vorkonfigurierte JMS-Objekte, die von einem Administrator erstellt wurden, und aus zwei Komponenten bestehen: *Connection Factories* und *Destinations.*

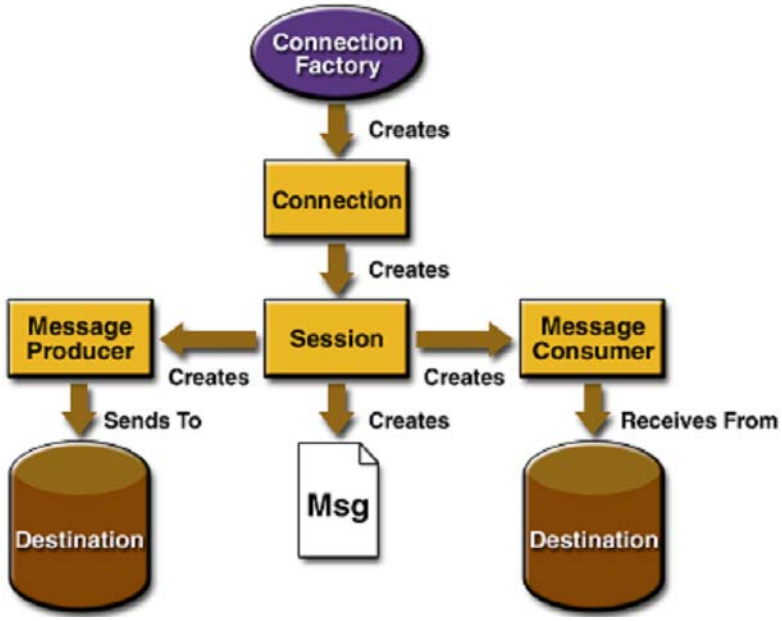
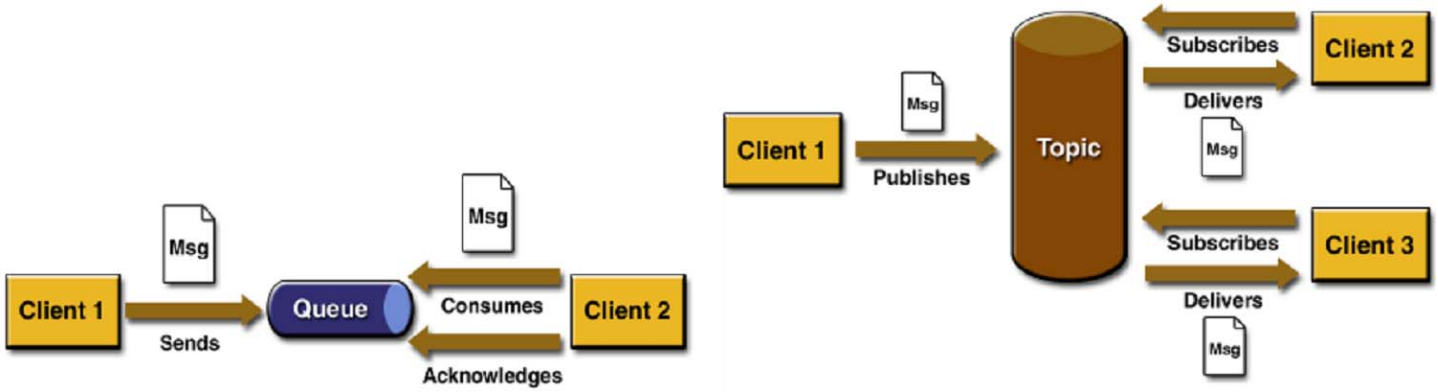


Abbildung 2.1 JMS Programming Architecture

Eine *Connection Factory* ist ein Client-Objekt, welches eine Verbindung zu einem Provider erzeugt. Sie kapselt einen Satz der Verbindungseinstellungen Konfigurationsparameter, der von einem Administrator definiert wurde. Jede Connection Factory ist eine Instanz der

- **QueueConnectionFactory oder**
- **TopicConnectionFactory Schnittstelle.**



Queue Modell **Topic Modell**

Abbildung 2.2 Queue / Topic Destination

Eine Destination ist ein Objekt, welches das Ziel (Target) spezifiziert, wohin ein Message Producer die Message hin liefert, und von wo der Message Consumer (Verbraucher) die Message ausliest. Es existieren zwei Arten von Zielen: Queue und Topic. Im Queue Modell muss jede Nachricht von genau einem Consumer abgeholt werden. Im Topic-Modell kann jede Nachricht von vielen Verbrauchern verarbeitet werden. Die Nachricht wird so lange im Speicher abgelegt, bis alle Verbraucher sie bekommen haben, siehe Abbildung 2.2.

Eine Connection erstellt eine virtuelle TCP/IP-Socket-Verbindung zwischen einem Client und einem Provider Service Daemon.

Eine Sitzung (Session) ist ein Single-Threaded-Context für die Erstellung und Verarbeitung von Nachrichten. Die Sitzungen werden durch eine Verbindung (Connection) erstellt.

Ein Message Producer implementiert die MessageProducer-Schnittstelle, die von einer Sitzung erstellt und verwendet wird, um Nachrichten an ein Ziel zu senden.

Ein Message Consumer ist ein Objekt, das von einer Sitzung erstellt wird, und die MessageConsumer-Schnittstelle implementiert, um Nachrichten zu empfangen, die entweder Queue oder Topic sein können.

Der Zweck einer JMS-Anwendung ist es, Nachrichten zu erstellen, die von anderen Komponenten verwendet werden können. Ein JMS-Nachricht besteht aus drei Teilen: einem Nachrichten-Kopf (Header), Eigenschaften (Properties) und einem Body. Nur der Header ist unbedingt erforderlich; die anderen zwei Teile können in einer Nachricht fehlen.

Header Field	Set By
JMSDestination	send or publish method
JMSDeliveryMode	send or publish method
JMSExpiration	send or publish method
JMSPriority	send or publish method
JMSMessageID	send or publish method
JMSTimestamp	send or publish method
JMSCorrelationID	Client
JMSReplyTo	Client
JMSType	Client
JMSRedelivered	JMS provider

Tabelle 2.1 Properties of Message Header

Ein JMS Message-Header hat eine Reihe von vordefinierten Feldern, die von Clients und Producern verwendet werden, um Nachrichten zu identifizieren und weiterzuleiten. Jedes Header-Feld verfügt über eigene Setter- und Getter-Methoden. Tabelle 2.1 zeigt alle Felder und ihre Anordnung.

Wenn wir zusätzliche Informationen in eine Nachricht für andere Komponenten setzen wollen, könnten wir Message-Properties setzen, zB. wenn eine Eigenschaft für einen Message-Selector benötigt wird.

Message Type	Body Contains
TextMessage	A java.lang.String object
MapMessage	A set of name-value pairs, with names as String objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
BytesMessage	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format
StreamMessage	A stream of primitive values in the Java programming language, filled and read sequentially.
ObjectMessage	A Serializable object in the Java programming language.
Message	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

Tabelle 2.2 JMS Message Types

Der Body enthält den Inhalt einer Nachricht. Jeder Messsage-Inhalt muss einem vordefinierten Nachrichtenformat folgen, als Message-Typ bezeichnet. Dies ermöglicht es den Software-Komponenten Daten in verschiedenen Formaten zu senden und zu empfangen. Tabelle 2.2 zeigt die möglichen Nachrichtentypen.

Wie Session Beans und Entity Beans benutzen Message Driven Beans (MDB) auch einen Deployment Descriptor (XML-Datei). Ab Version 3 können die meisten Angaben, für die zuvor der Deployment Descriptor notwendig war, mit Annotationen direkt im Java-Code implementiert werden. Dadurch kann der Deployment Descriptor entweder ganz entfallen, oder durch die Angaben in den Annotations überschrieben werden.

Selbst-Test

- **Was ist der Unterschied zwischen einem Message Producer und einem Message Provider?**
- **Benutzen JMS Messages einen Standard wie IDL?**

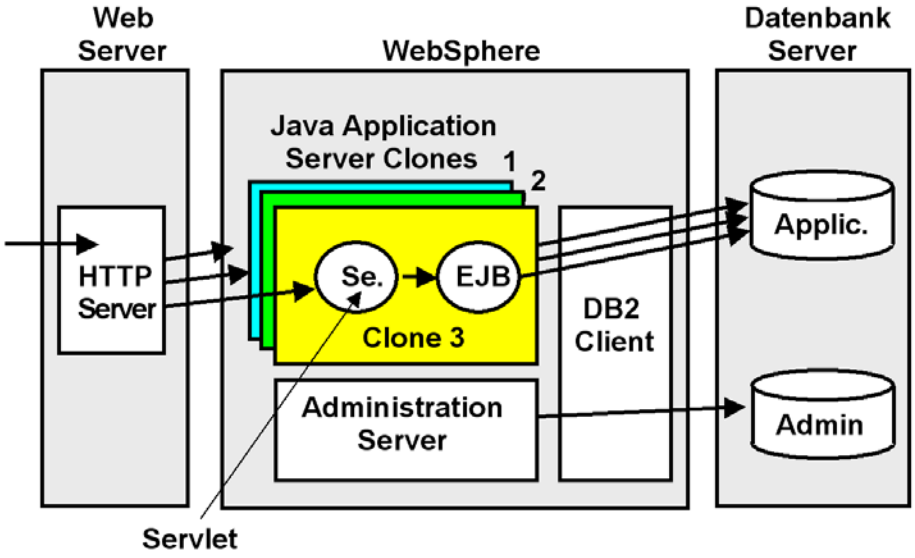
3 Entwickeln einer Message Driven Bean-Anwendung

Eine Message Driven Bean ist ein asynchroner Nachrichtenverbraucher.

In JEE wirkt eine MDB normalerweise als Verbraucher und implementiert die Listener-Schnittstelle, die nur eine Methode „onMessage()“ nach dem JMS-Standard hat. Ein MDB bietet keine Client-Schnittstelle und kann nur passiv ausgeführt werden. Die MDB ist mit einer Queue verbunden (oder einem Topic-Modell, wenn eine Nachricht von mehren MDBs empfangen werden soll), und tut nichts, bis eine Nachricht eintrifft.

WAS6.1 benutzt für die JMS Implementierung eine integrierte Version von WebSphere MQ. An dieser Stelle ist es sinnvoll, wenn Sie Thema 10 „WebSphere MQ“ des Moodle-Skriptes „Einführung in z/OS“ aus dem Wintersemester rekapitulieren.

3.1 Konfiguration in WAS6.1



Die hier gezeigte Abbildung ist eine Kopie aus dem Moodle-Skript „z/OS Internet Integration“, Thema WebSphere, Teil 3.

Aus Performance-Gründen muss der Application Server multiprogrammiert arbeiten. Hierzu sind mehrfache Clones des Application Servers vorhanden, die Java-Anwendungen parallel verarbeiten können. Die Clones des Java Application Servers verfügen jeder über eine eigene Java Virtuelle Maschine. Auf mehreren Clones kann die gleiche Anwendung laufen. Es ist auch möglich, dass auf den Clones unterschiedliche Anwendungen laufen, oder dass eine bestimmte Anwendung erst bei Bedarf geladen wird.

WAS6.1 beinhaltet die Möglichkeit, dass die Clones eines WAS6.1 Servers miteinander kommunizieren können, sowie auch mit logischen anderen Application Servern, die sich auf anderen physischen Rechnern befinden. Hierzu unterhält WAS6.1 einen internen Communication Bus, der als Service Integration Bus (SIB) bezeichnet wird.

Ein Service Integration Bus ist ein verwalteter Kommunikationsmechanismus eines WebSphere Application Servers, der die Serviceintegration durch synchrones oder asynchrones Messaging unterstützt. Er unterstützt Anwendungen, die nachrichtenbasierte und serviceorientierte Architekturen verwenden. Ein Service Integration Bus unterstützt eine Gruppe miteinander verbundener Server (oder Server-Cluster), die dem Bus als Member hinzugefügt wurden. Anwendungen stellen über eine Messaging-Steuerkomponente, die den Bus-Membern zugeordnet ist, eine Verbindung zum Bus her.

In unserem Tutorial ist der SIB wirklich overkill, da wir nur einen einzigen Application Server (hier als Server1 bezeichnet) benötigen. Die WebSphere Application Server Architektur ist aber so ausgelegt, dass auch sehr komplexe Anwendungen implementiert werden können. Auf Grund des großen Funktionsumfanges ist die Konfiguration eines WebSpere Web Applikation Vorgangs eine komplexere Aufgabe, als dies bei einfachen Web Application Servern wie z.B. Geronimo oder JBoss der Fall ist. Für die Konfiguration enthält WAS6.1 einen integrierten Administration Server mit entsprechender Benutzer Oberfläche in einem als Konsole bezeichnetem Fenster.

Das Ziel (Destination) eines Busses ist eine logische Adresse, die den Anwendungen als Produzent und/oder als Konsument zugeordnet werden kann. Im Falle einer Java Message Service (JMS) Application ist das Ziel eines Busses eine Warteschlange, welches für das Punkt-zu-Punkt-Messaging verwendet wird.

Ein Service Integration Bus bietet die folgenden Funktionen:

- **Eine Anwendung kann Nachrichten mit jeder anderen Anwendung austauschen. Hierfür wird eine *Destination* verwendet, an die eine Anwendung Nachrichten sendet und von der die andere Anwendung Nachrichten empfängt.**
- **Eine nachrichtenerzeugende Anwendung, d.h. ein *Erzeuger*, kann unabhängig von der Messaging-Steuerkomponente, die der Erzeuger für den Zugriff auf den Bus verwendet, Nachrichten für eine Destination erzeugen.**
- **Eine nachrichtenkonsumierende Anwendung, d.h. ein *Konsument*, kann unabhängig von der Messaging-Steuerkomponente, die der Konsument für den Zugriff auf den Bus verwendet, Nachrichten von einer Destination konsumieren (wenn diese Destination verfügbar ist).**

Zur Implementierung auf unserem WAS 6.1 öffnen Sie einen Browser für den Zugriff auf die WAS Administration Console. Geben Sie "https://localhost:9043/ibm/console/" in die Adresszeile ein. 9043 ist der Default-Port für die WAS6.1 Konsole. Wenn Sie nicht wissen, welchen Port Ihr WAS benutzt, schauen Sie in der Datei <*was-home*>*AppServerprofiles**AppSrv01**properties**portdef.props* nach, wobei <was-home> der WAS-Installationspfad ist. Für die von WAS benutzten Default Port-Nummern siehe auch http://publib.boulder.ibm.com/infocenter/wsdoc400/v6r0/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/adrportbase.htm.**

Selbst-Test

- **Würde man einen SIB benötigen, wenn der Web Application Server nur eine einzige JVM mit je einem Servlet Container und einem EJB Container unterstützt?**
- **Können die SIBs mehrer physischen Server miteinander verbunden werden?**

3.1.1 Erstellen eines Service Integration Bus

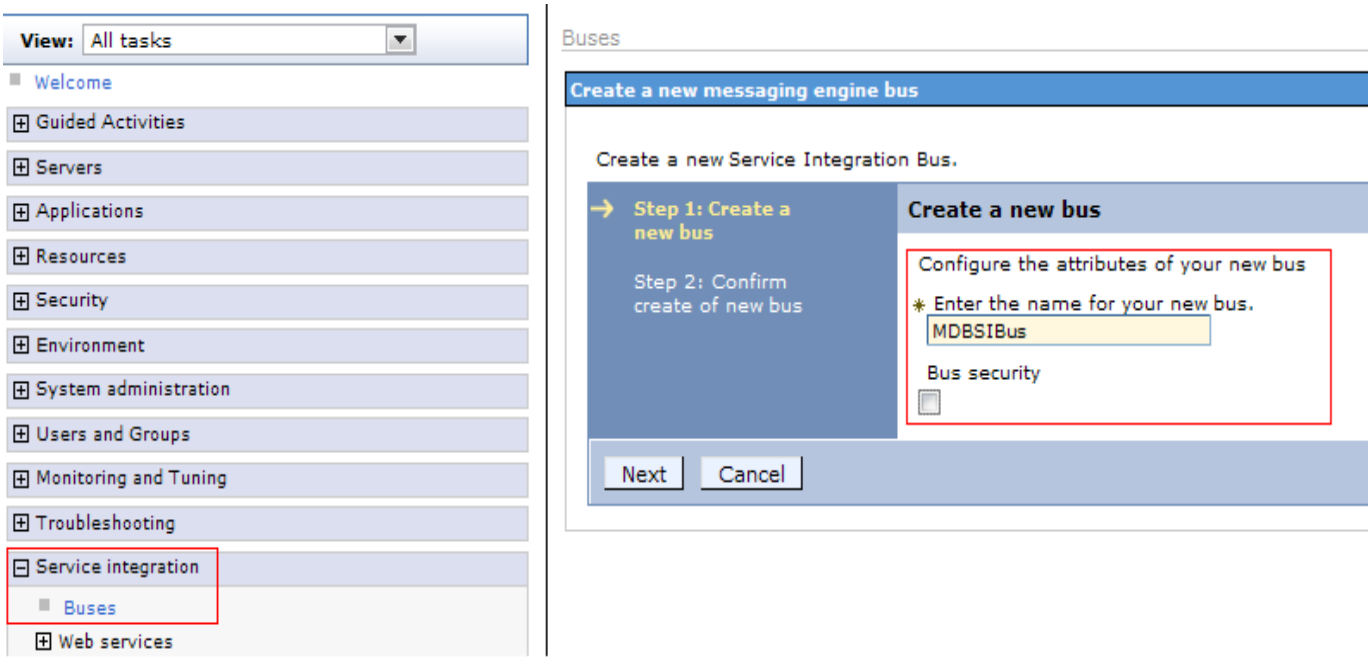


Abbildung 3.1.1-2 Erstellen eines Service Integration Bus

Wählen Sie Service-Integration → Buses und klicken Sie auf New.

Geben Sie MDBSIBus als Name ein, und *deaktivieren* Sie Bus security. (Abbildung 3.1.1-2)

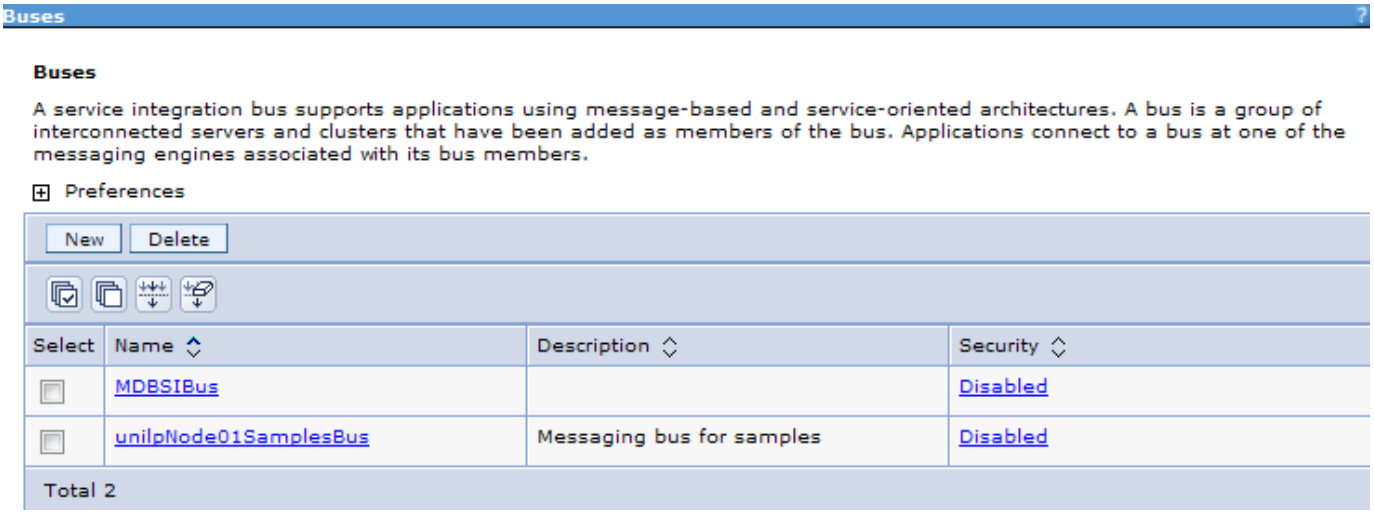


Abbildung 3.1.1-3 Erstellen eines Service Integration Bus (2)

Klicken Sie auf Next und klicken Sie dann auf Finish, um die MDBSIBus-Konfiguration zu bestätigen.

***Hinweis:* Nach jeder Änderung müssen Sie auf Save drücken, um alle Änderungen im WAS zu speichern. Bitte beachten sie das bei allen weiteren Änderungen, wir werden dies im Text nicht wiederholen.**

3.1.2 Erstellen der Messaging-Engine

Ein SIB wird benötigt, um einen (oder mehrere) Applikations-Server (oder Application Server-Cluster) zu betreiben. Für die meisten Entwicklungsaktivitäten, werden Sie einen einzigen Bus-Member für den Bus erstellen und zuweisen. Wenn ein Bus-Member hinzugefügt wird, wird eine Messaging-Engine für diese WebSphere-Instanz erstellt. Diese unterhält ihren eigenen Datenspeicher für Nachrichten.

Siehe auch:
http://www.ibm.com/developerworks/websphere/techjournal/0501_reinitz/0501_reinitz.html

http://www.ibm.com/developerworks/websphere/techjournal/0502_reinitz/0502_reinitz.html#ibm-pcon

Eine Messaging-Engine ist eine Server-Komponente, welche die Kern-Messaging-Funktion eines Service Integration Busses zur Verfügung stellt. Eine Messaging-Engine verwaltet Bus-Ressourcen und ermöglicht es Anwendungen, mit dem Bus zu kommunizieren.

Jede Messaging-Engine ist einem Server zugeordnet, der als ein Mitglied eines Busses assoziiert wurde. Wenn Sie einen Anwendungsserver als Bus-Member hinzufügen, wird eine Messaging-Engine automatisch für dieses neue Mitglied erstellt. Wenn Sie denselben Server als Mitglied mehrerer Busse hinzufügen, wird der Server mit mehreren Messaging-Engines (eine Messaging-Engine für jeden Bus) verbunden. In ihrer einfachsten Form kann ein Bus von einer einzigen Messaging-Engine realisiert werden.

Messaging-Engines haben einen Namen, der sich aus den Namen des Bus Members ergibt. Jede Messaging-Engine hat auch eine universelle eindeutige Kennung (UUID), die eine eindeutige Identität für die Messaging-Engine bietet. Wenn Sie eine Messaging-Engine löschen und neu erstellen, wird sie eine andere UUID haben. Sie wird nicht durch den Bus als die gleiche Engine erkannt werden, auch wenn sie vielleicht den gleichen Namen hat.

Auf geht's mit dem Erstellen der Messagine-Engine!

Selbst-Test

- **Wie viele Messaging-Engines benötigt ein Server?**

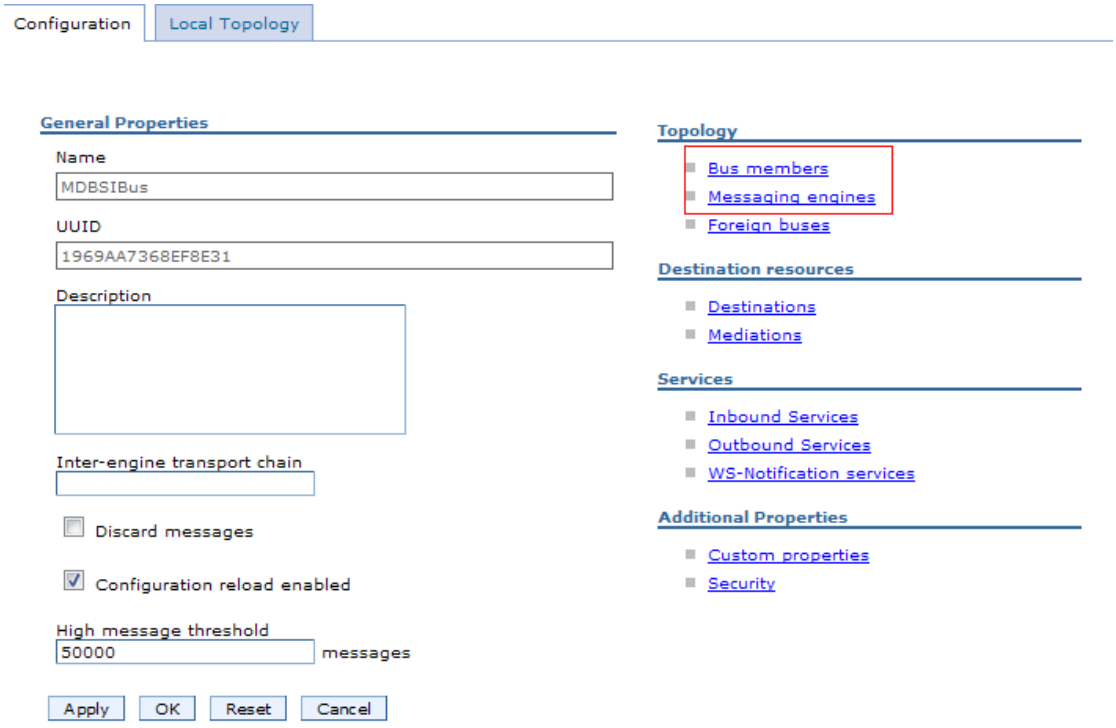


Abbildung 3.1.2-1 Erstellen einer Messaging-Engine

1k auf MDBSIBus um in die Properties Configuration Page zu gelangen (Abbildung 3.1.2-1).

1k auf Bus members in der Topology-Auswahl.

Klick Add und selektiere server1 in dem Server-DropDown-Menu. 1k auf Next.
Selektiere File store, und 1k auf Next.

Ändere die Default Log-Größe auf 20MB, die Minimum permanente und temporäre Store-Größe auf 20MB, und das Maximum auf 100MB.

1k auf Next und Finish.

Abbildung 3.1.2-2 Erstellen einer Messaging-Engine (2)

Zurück zum MDBSIBus Panel, Klick Messaging-Engines um alle verfügbaren Messaging-Engines für den MDBSIBus (Abbildung 3.1.2-2) aufzulisten.

3.1.3 Erstellen einer Destination

Mess

aging ermöglicht eine verteilte (distributed) Kommunikation, die lose gekoppelt ist. Eine Komponente sendet eine Nachricht an ein Ziel, und der Empfänger kann die Nachricht vom Ziel abrufen.

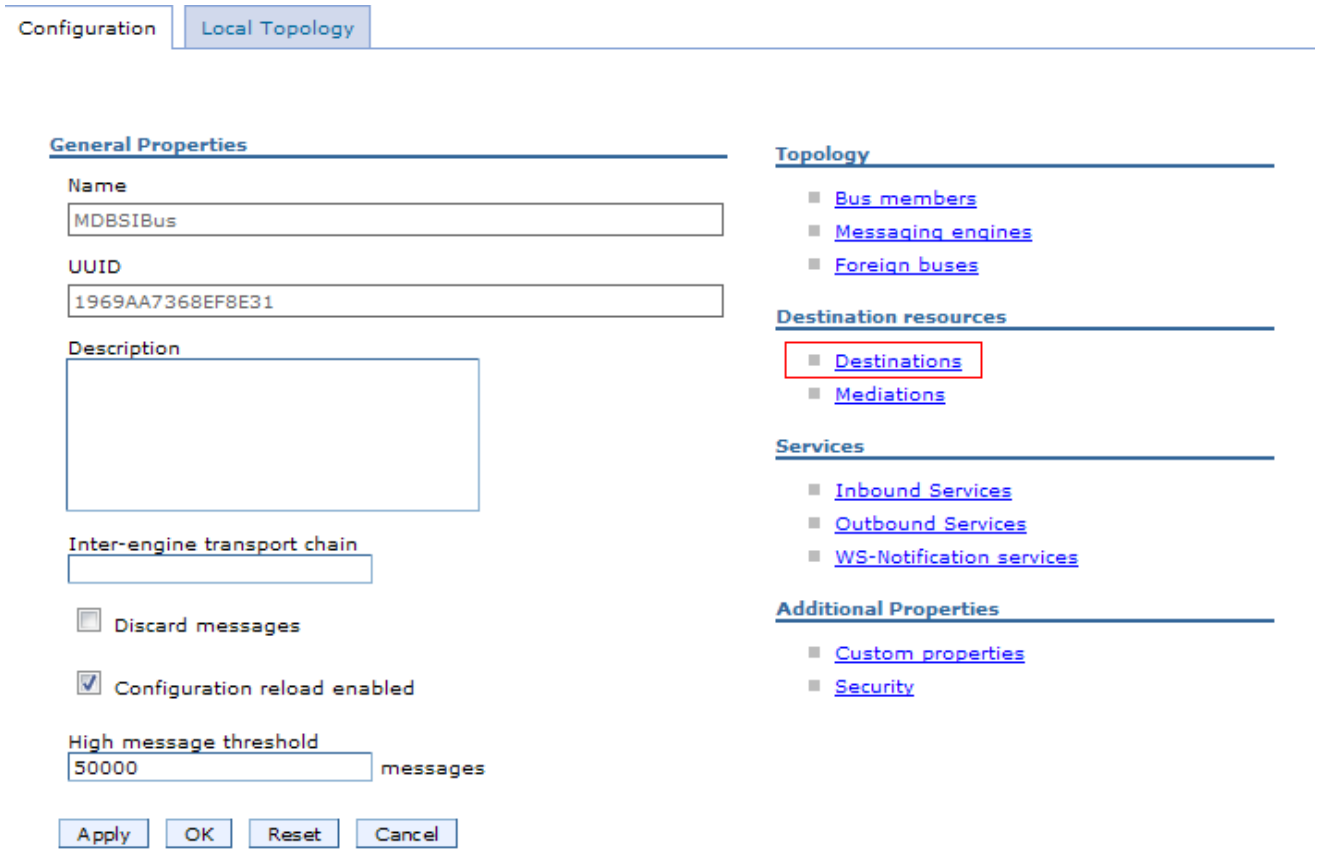


Abbildung 3.1.3-1 Ein Ziel erstellen

Zurück zum MDBSIBus-Panel, Destination auswählen (Abb. 3.1.3-1). 1k auf New.

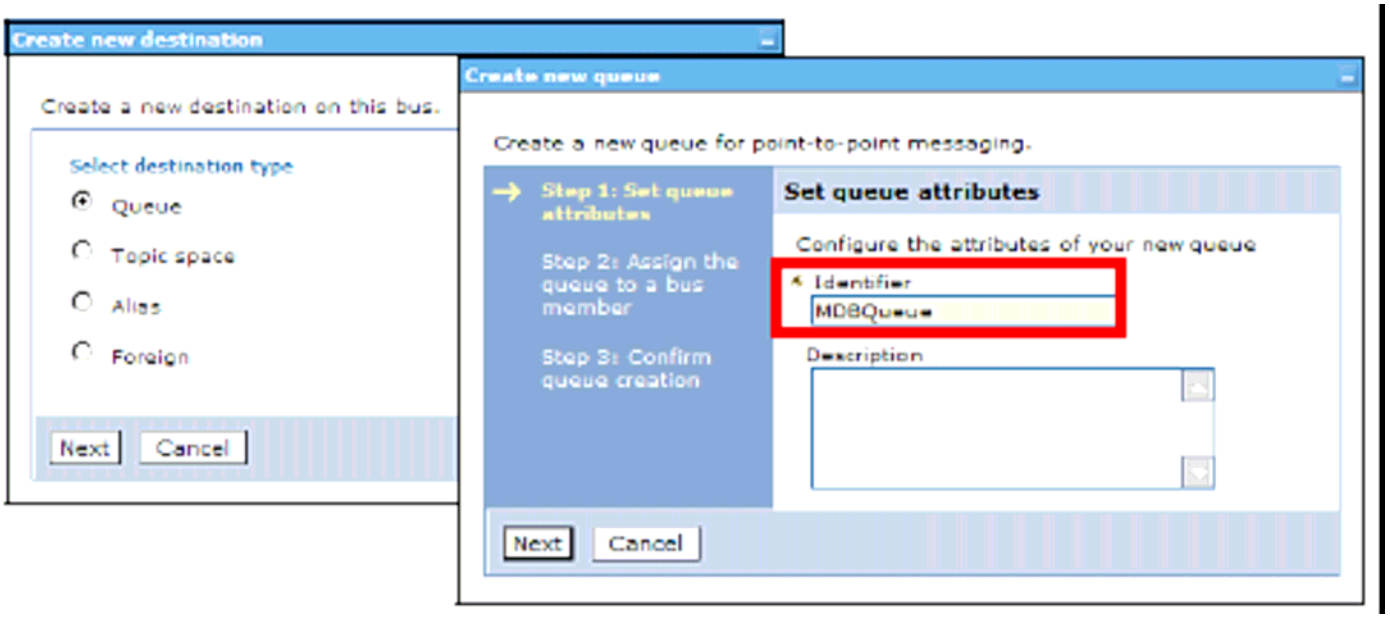


Abbildung 3.1.3-2 Ein Ziel erstellen

Selektiere Queue und klick Next (Abbildung 3.1.3-2).

MDBQueue als Identifier eingeben, und 1k auf Next.

Stellen Sie sicher, dass die Queue dem Bus-Member von MDBSIBus zugeordnet ist und 1k auf Next und Finish.

3.1.4 Konfigurieren des JMS-Provider

Nun beginnen wir mit der Konfiguration des JMS-Providers. In unserem Fall ist dies eine einzige MDB in ihrem EJB Container. Die MDB empfängt eine Nachricht über eine Queue, welche von der in WAS6.1 integrierten MQ-Komponente zur Verfügung gestellt wird. Hierzu definieren wir eine Queue Connection Factory und eine Queue mit dem gleichen Namen wie der JNDI-Name, der in dem Servlet benutzt wird. Die Queue ist Bestandteil der in WAS6.1 integrierten MQSeries-Komponente.

Zusätzlich definieren wir eine JMS-Aktivierungsspezifikationen passend zu der MDB.

Diese werden verwendet, um Message-Queues den MDB-Anwendungen zuzuordnen, welche die MDB-onMessage-Methode implementieren. Die MDB-Aktivierungsspezifikationen ersetzt die in früheren Versionen verwendeten Listener-Ports

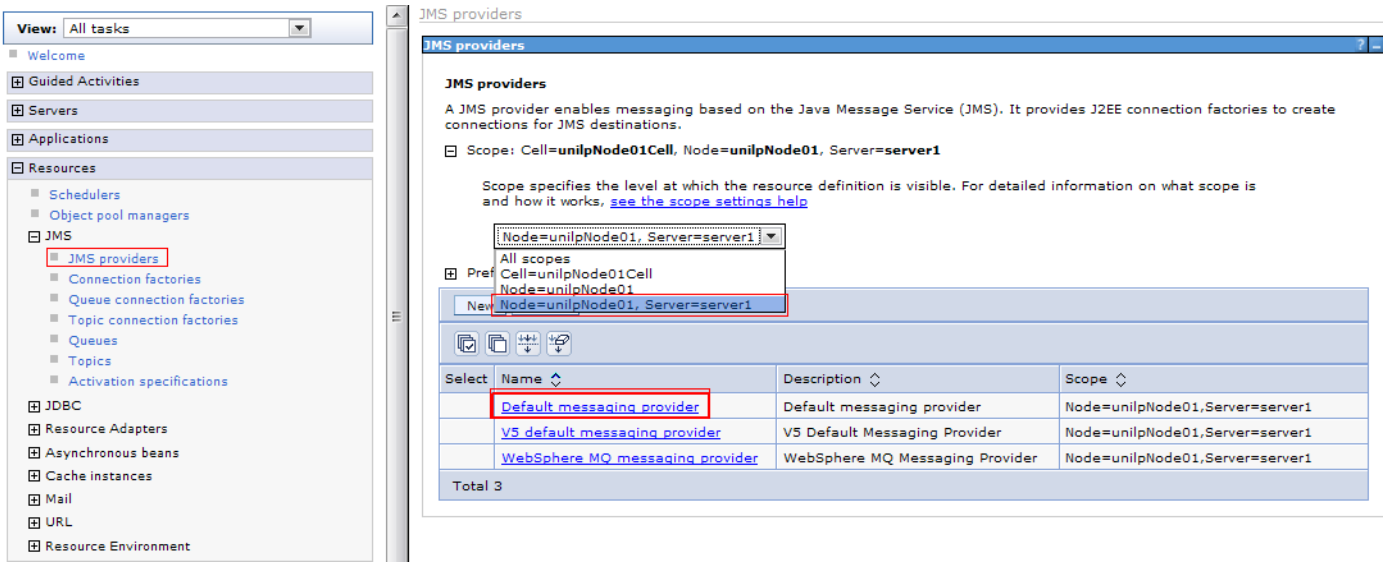


Abbildung 3.1.4-1 Selektiere *Default message provider*

Wählen Sie Ressourcen → JMS → JMS providers.

klicken Sie auf die Standard-Messaging-Provider auf der Server-Ebene. (Abbildung 4.1.4-1)

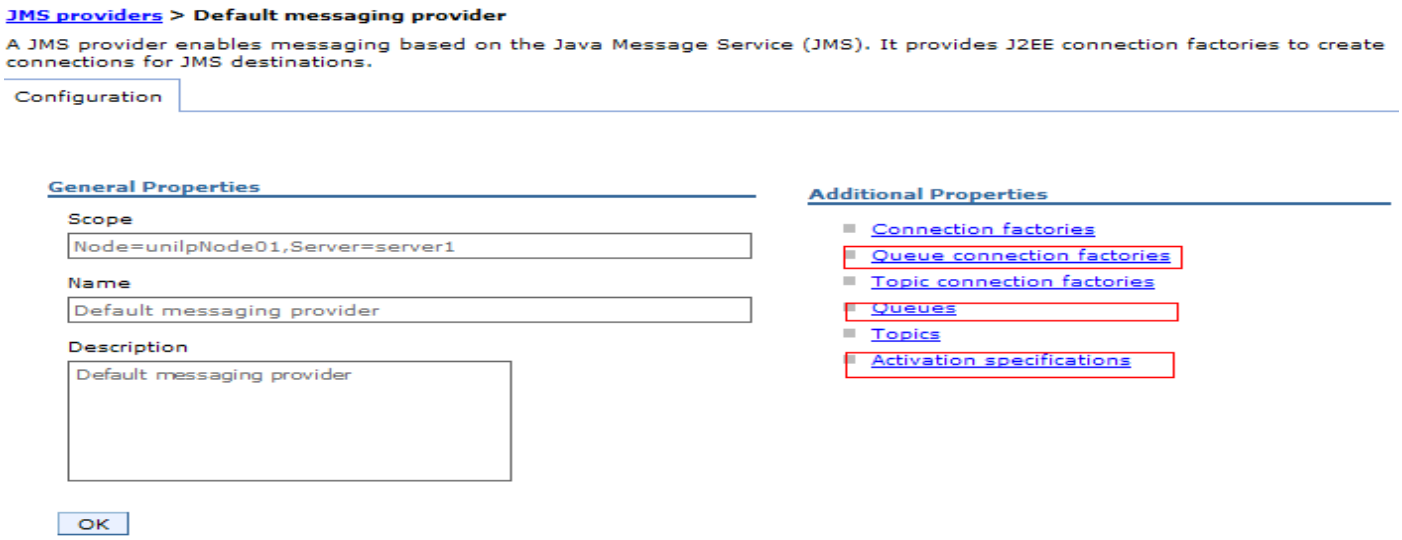


Abbildung 3.1.4-2 Properties Configuration

Im Folgenden werden Queue Connecton Factories, Queues und Activation Specifications konfiguriert (Abbildung 3.1.4.-2)

1k auf Queue Connection Factories.

Selbst-Test

- **Ist ein JMS Provider ein JMS Producer oder ein Message Consumer ?**
- **Gilt dies für dieses Tutorial oder generell ?**

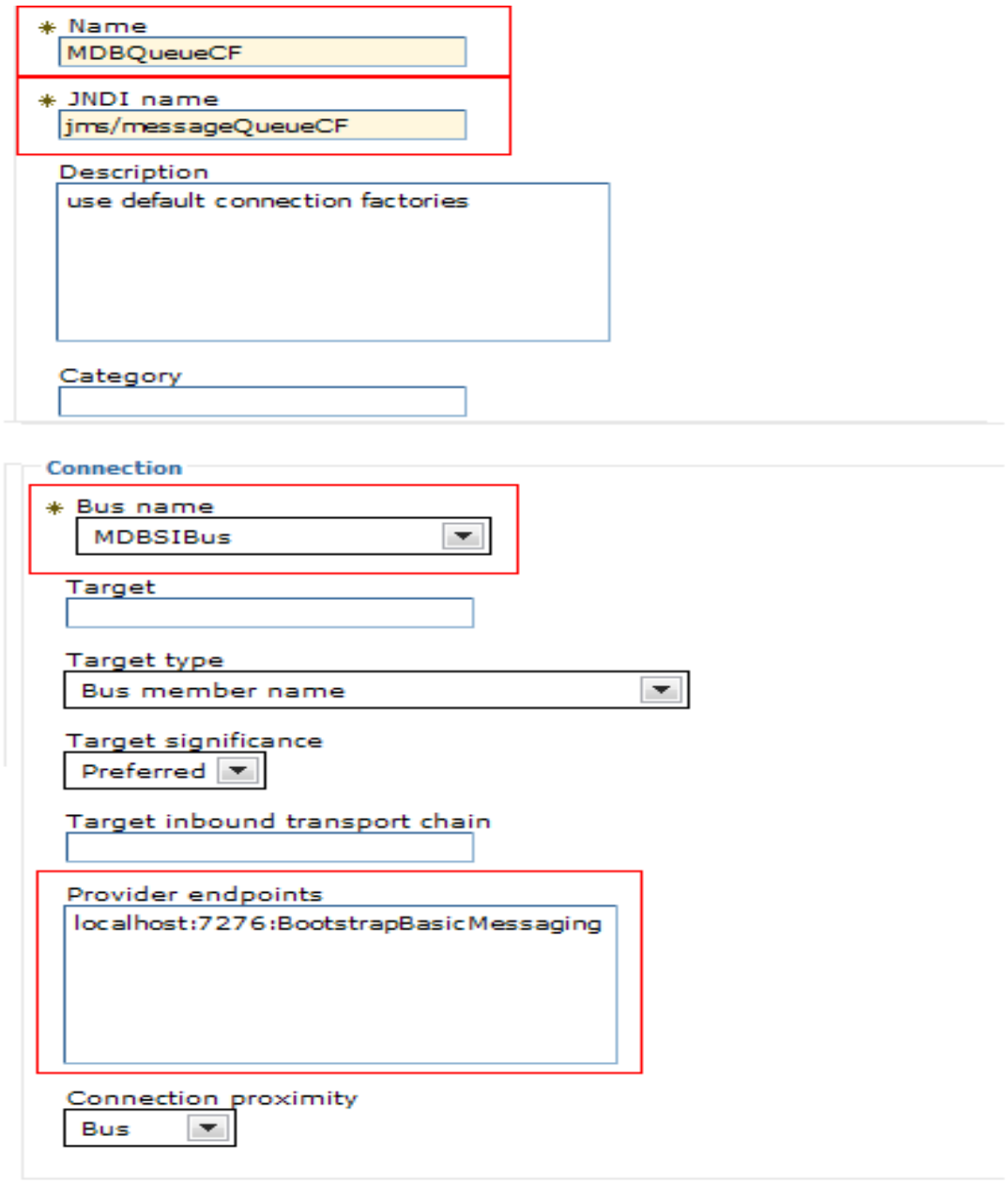


Abbildung 3.1.4-3 Queue Connection Factories Configuration

Klick New. Eingeben MDBQueueCF als Name und jms/messageQueueCF als JNDI Name. Selektiere MDBSIBus als Bus Name.

Als „Provider endpoints“: localhost:7276:BootstrapBasicMessaging eingeben, wobei 7276 der Standard-SIB_ENDPOINT_ADDRESS-Port für den Server ist. Wenn Sie nicht wissen, welchen Port Ihr WAS benutzt, siehe die Datei *<was-home>\AppServer\profiles\AppSrv01\properties\portdef.props*, wobei <was-home> der WAS Installationspfad ist.

1k auf OK (Abbildung 3.1.4-3)

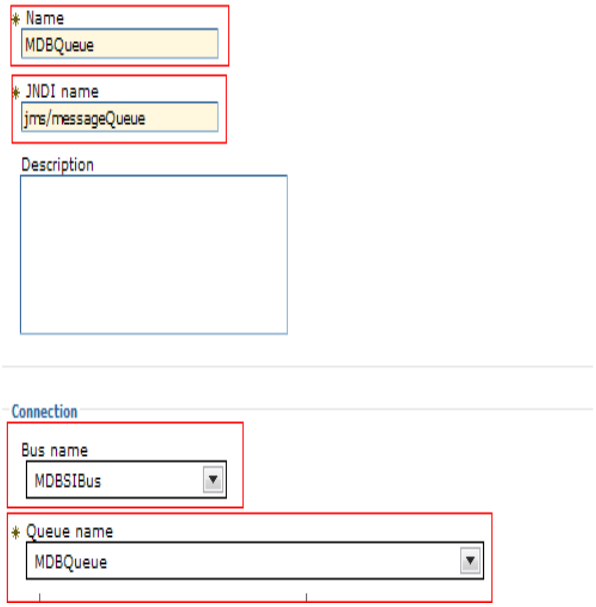


Abbildung 3.1.4-4 Queue Configuration

Zurück zu der *Default Messaging Provider*-Seite, 1k auf Queues. (Abbildung 3.1.4-4)

1k auf New. MDBQueue als Name und jms/messageQueue als JNDI name eingeben. MDBSIBus als Bus Name, MDBQueue als Queue Name auswählen und OK.

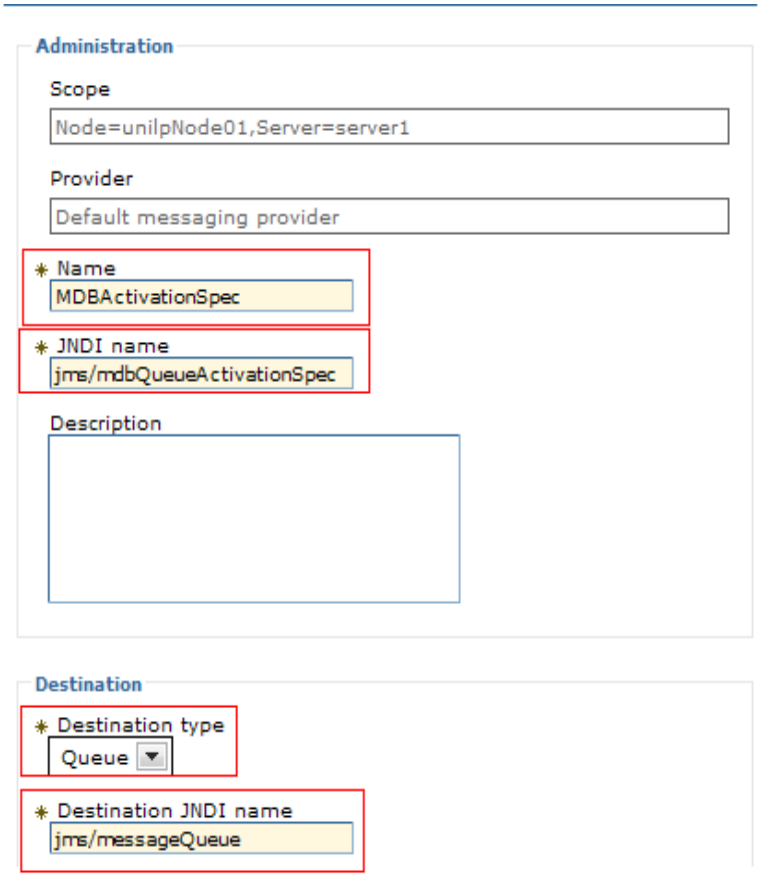


Abbildung 3.1.4-5 Activation specification Configuration

Zurück zu der *Default Messaging Provider*-Seite, 1k auf Activation specifications (Abbildung 3.1.4-5)

1k auf New. MDBActivationSpec als Name und jms/mdbQueueActivationSpec als JNDI Name eingeben. Wählen Sie Queue als Destination Type und geben Sie jms/messageQueue als Destination JNDI Name ein. MDBSIBus als Bus Name auswählen und 1k auf OK.

Den Server neu starten um die Messaging Engine zu aktivieren.

Damit ist die Konfiguration unseres WAS6.1 Application Servers abgeschlossen.

3.2 Erstellen einer Enterprise-Applikation

Wir gehen in diesem Abschnitt ähnlich wie in dem letzten Tutorial vor. Wir erstellen zunächst ein Enterprise Application-Projekt und dessen EAR Deployment-Deskriptor. Dann erstellen wir ein EJB Projekt mit dem Code der MDB und dem dazugehörigen Deployment-Deskriptor. Darauf erstellen wir ein Dynamic Web Project, welches ein Servlet enthält.

3.2.1 Erstellen eines Enterprise Application-Projektes

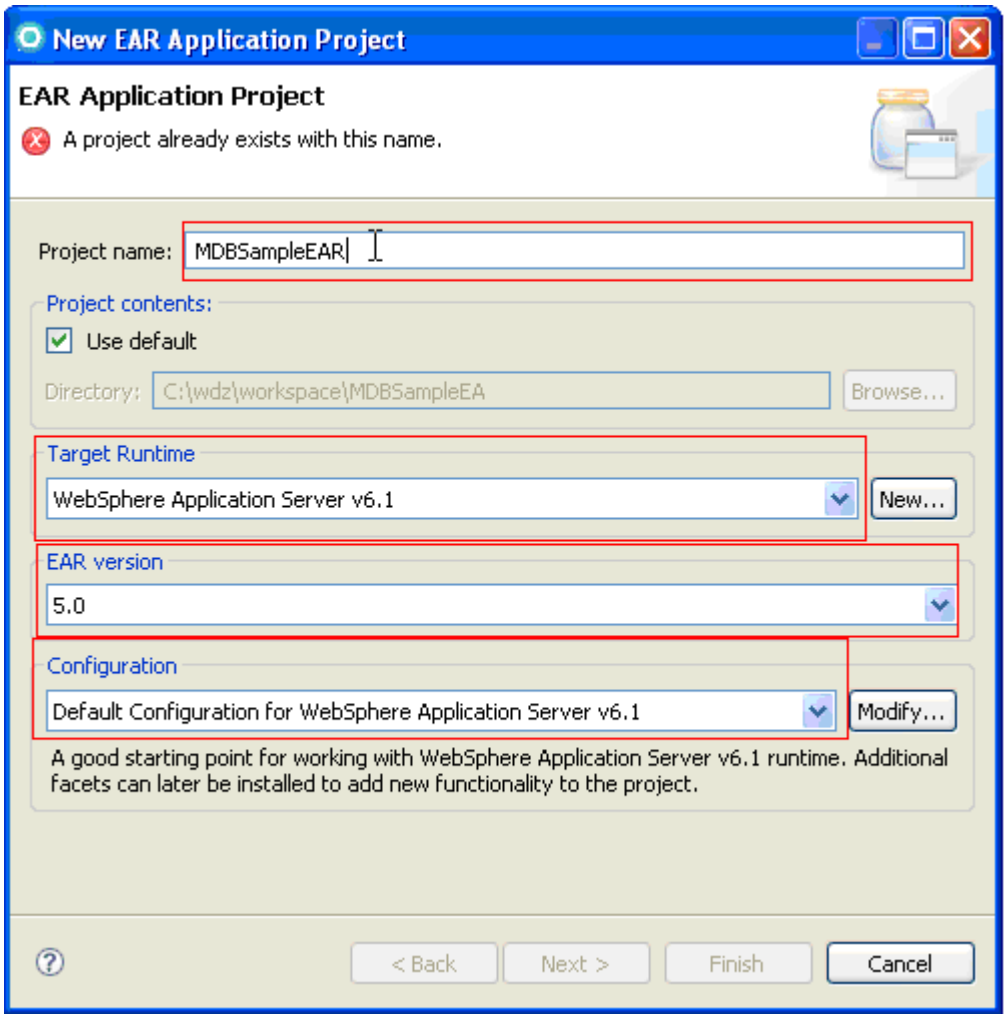


Abbildung 3.2.1-1 Erstellen eines Enterprise Application Projects

Starten Sie RAD7.5 und erstellen Sie ein Enterprise Application Projekt mit dem Namen MDBSampleEAR. Es solle ein EJB Project und ein Dynamic Web Application Projekt enthalten, (siehe Abbildung 3.2.1-1). In dem folgenden Fenster Generate Deployment Descriptor anwählen und 1k auf Finish.

3.2.2 Erstellen eines EJB Projects

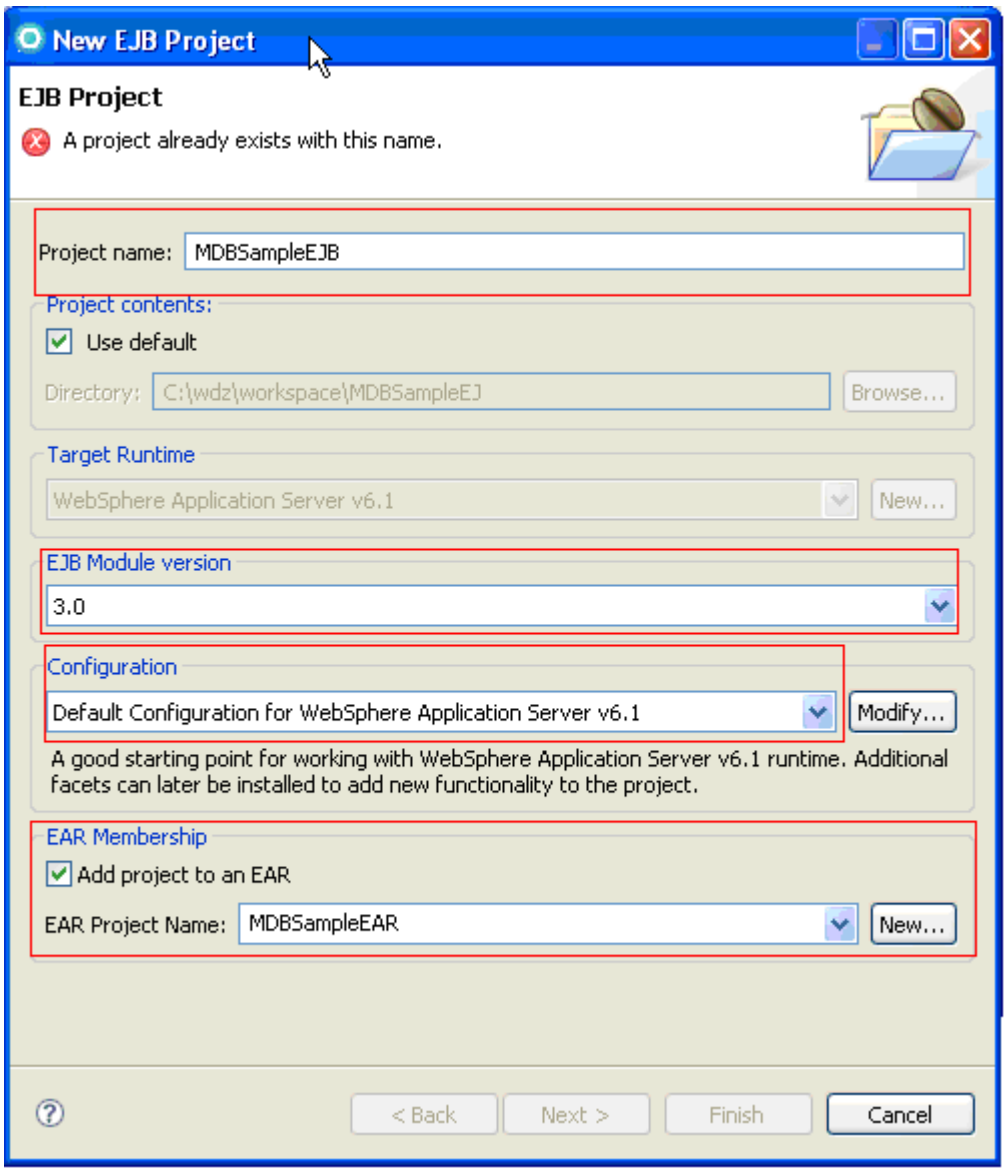


Abbildung 3.2.2-2 Creating an EJB Project

Erstellen Sie ein neues EJB-Projekt mit dem Namen MDBSampleEJB als Teil der MDBSampleEAR Enterprise Application. Wählen Sie 3.0 als die EJB Module Version (Abbildung 3.2.2-1). In dem nächsten Dialog die Option Create an EJB Client Jar module *abwählen* und dann 1k auf Finish.

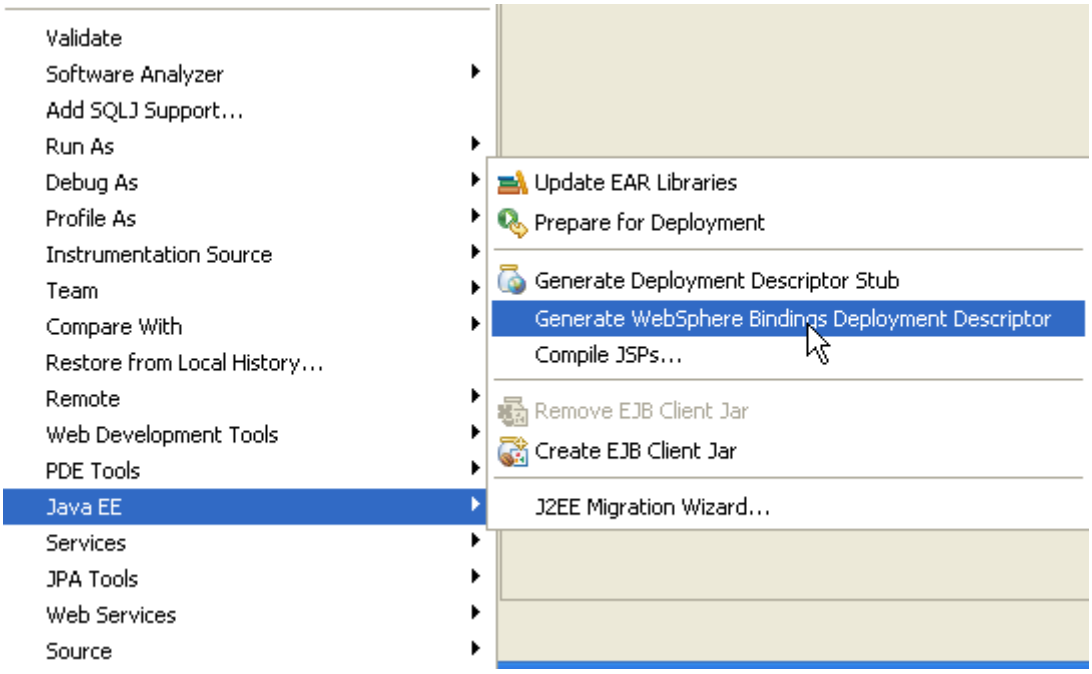


Abbildung 3.2.2-3 Generating ibm-ejb-jar-bnd.xml

1kr auf MDBSampleEJB-Projekt. Selektiere Java EE->Generate WebSphere Bindings Deployment Descriptor um die ibm-ejb-jar-bnd.xml-Datei zu erzeugen (Abbildung 3.2.2-3).

Selbst-Test

- **Stellen die Abbildungen Abbildung 3.2.2-1 bis Abbildung 3.2.2-3 RAD7.5- oder WAS6.1-Fenster dar?**

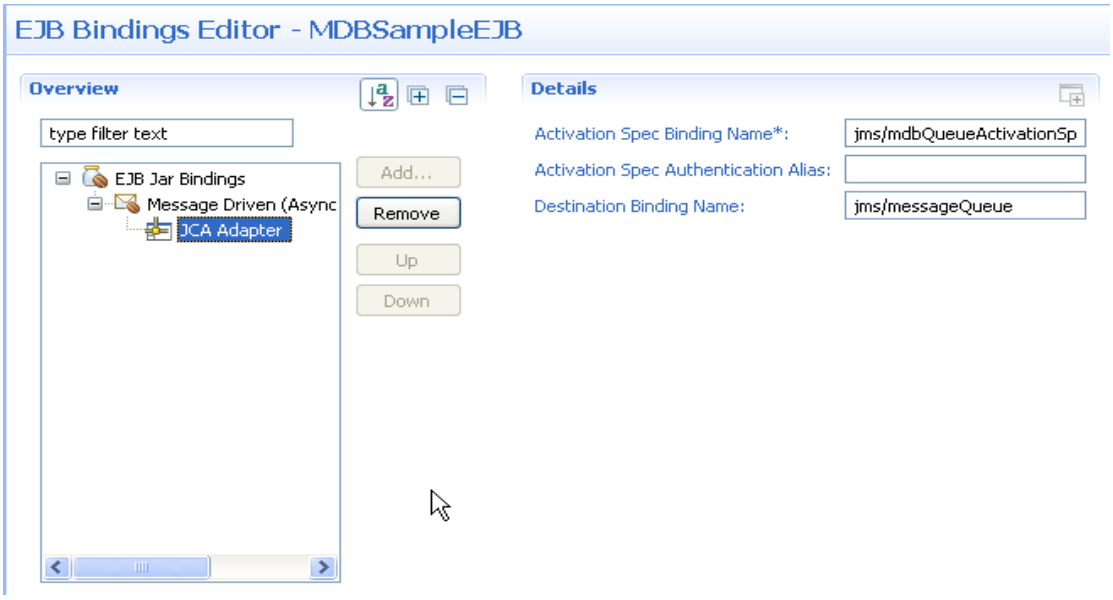


Abbildung 3.2.2-3 MDB Resource Mapping

In dem Design-Tab, können logische Ressourcen erstellt werden, welche die physischen Komponenten abbilden, die vom WAS gemanagt werden.

Klick Add um eine neue Ressource zu erstellen. Wählen Sie Message Driven und 1k auf OK. AsyncMessageConsumerBean als Name eingeben.

Die MDB anwählen, 1k auf Add und JCA Adapter auswählen. Nach dem Hinzufügen des Adapters, jms/mdbQueueActivationSpec als Activation Spec Binding Name eingeben und jms/messageQueue als Destination Binding Name (Abbildung 3.2.2-3).

Verifizieren Sie den erstellten Inhalt in dem Source-Tab. Bestätigen Sie, dass es den folgenden Quellcode entspricht:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar-bnd
	xmlns="http://websphere.ibm.com/xml/ns/javaee"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee

	  http://websphere.ibm.com/xml/ns/javaee/ibm-ejb-jar-
bnd_1_0.xsd"
	version="1.0">

	  <message-driven name="AsyncMessageConsumerBean">
		<jca-adapter activation-spec-binding-
name="jms/mdbQueueActivationSpec"
	  destination-binding-name="jms/messageQueue"/>
		</message-driven>

</ejb-jar-bnd>
```

Erstellen Sie eine Bean-Klasse mit dem Namen AsyncMessageConsumerBean im MDBSampleEJB-Projekt, welche eine Nachricht von der Queue empfängt, und TextMessage handhabt.

```
import java.util.logging.Logger;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.MessageDriven;
import javax.ejb.ActivationConfigProperty;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

/**
 * Message-Driven Bean implementation class for: AsyncMessageConsumerBean
 *
 */
@MessageDriven(activationConfig = {
  @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
       "javax.jms.Queue"),
  @ActivationConfigProperty(propertyName = "destination",
      propertyValue = "jms/messageQueue") })
  // a MDB must implements the MessageListene Interface
public class AsyncMessageConsumerBean implements MessageListener {

  Logger logger = Logger.getLogger(AsyncMessageConsumerBean.class.getName());

   @PostConstruct
   void postConstruct() {
           logger.info("PostConstruct called");
   }

   @PreDestroy
   void preDestroy() {
           logger.info("PreDestroy called");
   }

   /**
    * Default constructor.
    */
   public AsyncMessageConsumerBean() {
           // TODO Auto-generated constructor stub
   }
```

```
    /**
     * @see MessageListener#onMessage(Message)
     */
    public void onMessage(Message message) {
            if (!(message instanceof TextMessage)) {
                    logger.info("received a non-TextMessage message; exiting");
                    System.err.println("received a non-TextMessage message; exiting");
                    return;
            }
            TextMessage textMessage = (TextMessage) message;
            try {
                    System.err.println("AsynchBean Received message: "
                                    + textMessage.getText());
                    } catch (JMSException e) {
                    // we can simply ignore it (print it out to the console) for now
                    e.printStackTrace();
            }
    }
}
```

3.2.3 Erstellen eines Dynamic Web Projects

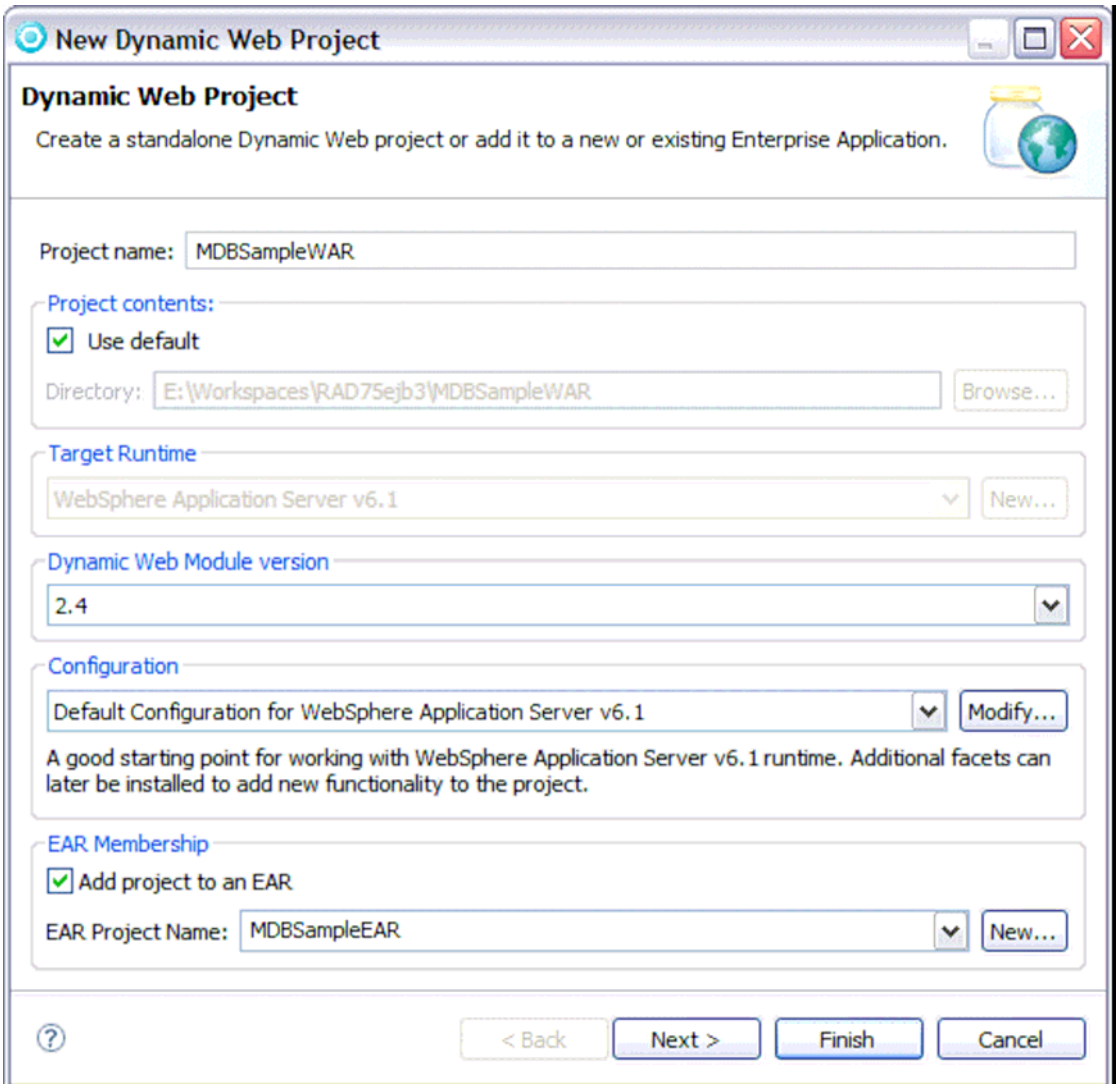


Abbildung 3.2.3-1 Erstellen eines neuen Dynamic Web Projects

Jetzt erstellen Sie ein neues Dynamic Web Project mit dem Namen MDBSampleWAR und fügen es zu der MDBSampleEAR Enterprise Application hinzu. (Abbildung 3.2.3-1).

Öffnen Sie den Web Application Deployment Descriptor.

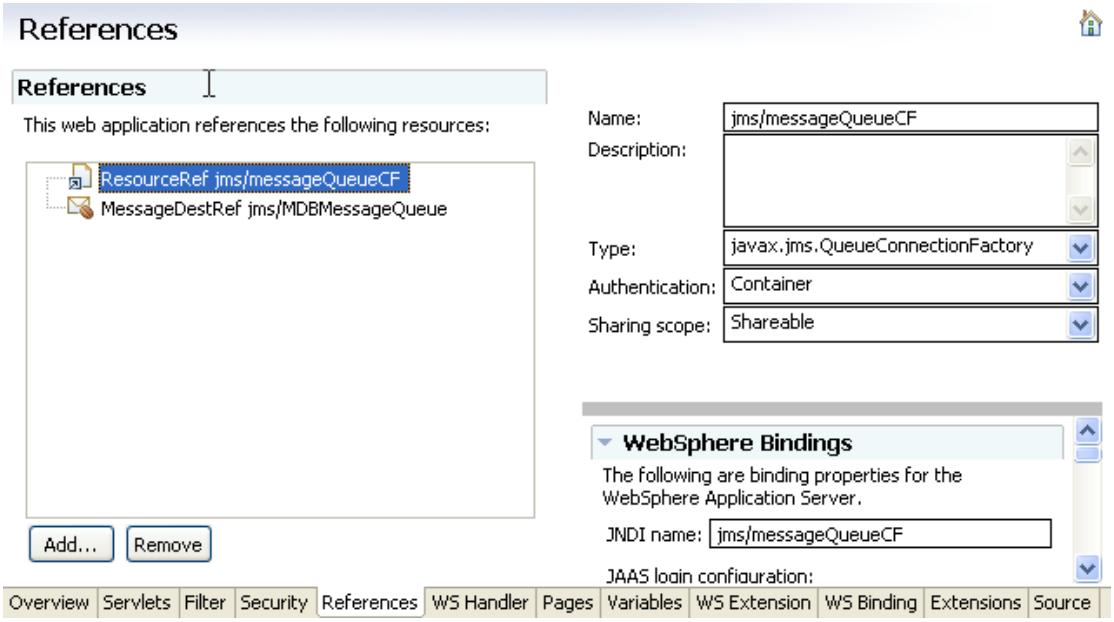


Abbildung 3.2.3-2 Adding a ResourceReference

Auf der References-Seite, 1k auf Add. Selektieren Sie Resource reference und 1k auf Next. jms/messageQueueCF als Namen eingeben. javax.jms.QueueConnectionFactory als Type wählen und Container für Authentication. 1k auf Finish.

Jetzt erscheint die ResourceRef in der Liste. Als JNDI Name jms/messageQueueCF eingeben, unter WebSphere Bindings(Abbildung 3.2.3-2).

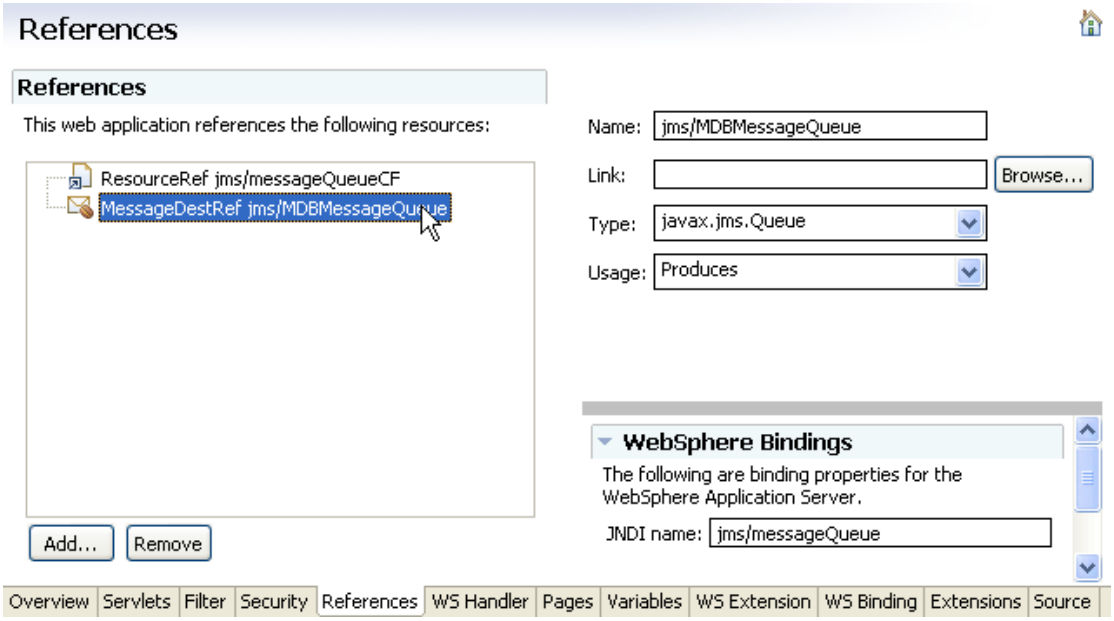


Abbildung 3.2.3-2 Adding a MessageDestinationReference

Klick nochmals Add, Message destination reference auswählen und 1k auf Next. Erstellen Sie eine neue Destination mit dem Namen jms/messageQueue und klick Next. Als Name jms/MDBMessageQueue eingeben, javax.jms.Queue als Type und Produces als usage. 1k auf Finish (Abbildung 3.2.3-3).

Hinweis: **Manchmal funktioniert das GUI-Tool nicht auf Grund von RAD Bugs. Wir müssen manuell den Code "<message-destination-ref> </ message-destination-ref>" zwischen "</ welcome-file-list>" und "</ web-app>" in dem Anwendung Deployment-Deskriptor hinzufügen, um die GUI-Konfiguration zu aktivieren.**

Die MessageDestRef erscheint in der Liste. Als JNDI Name ist jms/messageQueue einzugeben.

Speichern Sie mit Strg+S.

Erstellen Sie ein neues Servlet namens MessageCreaterServlet.java und klicken Sie auf Next. Im nächsten Fenster setzen Sie das URL-Mapping als "/MessageSender". Klicken Sie auf Finish. Dieses Servlet empfängt einen Parameter mit dem Namen "Message" von einer Web-Datei und sendet den Inhalt als TextMessage an die Queue.

Es folgt das Listing des Source Code:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.annotation.Resource;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MessageCreaterServlet extends HttpServlet {
   private static final long serialVersionUID = 18782112020275476 41L;
   // Using the Resource Reference defined in Web Deployment Descript
   @Resource(name = "jms/messageQueueCF")
   QueueConnectionFactory qcf;
   @Resource(name = "jms/MDBMessageQueue")
   Queue queue;

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
          performance(request, response);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse
           response) throws ServletException, IOException {
          performance(request, response);;
   }

   private void performance(HttpServletRequest request, HttpServletResponse
          response) throws ServletException, IOException{
          try {
                 QueueConnection connection = qcf.createQueueConnection();
                 QueueSession session = connection.createQueueSession(false,
                               Session.AUTO_ACKNOWLEDGE);
                 MessageProducer producer = session.createProducer(queue);
                 TextMessage txtMsg = session.createTextMessage();
                 txtMsg.setText(request.getParameter("Message"));
                 producer.send(txtMsg);
                 session.close();
                 connection.close();
                 // for testing purposes
                 PrintWriter out = response.getWriter();
                 out.println("Message is successfully sent.");
                 out.println("QueueConnectionFactory:" +qcf);
                 out.println("QueueConnection: "+ connection);
                 out.println("Queue: " + queue);
                 out.println("MessageProducer: " + producer);
                 out.println("Message Content: " + request.getParameter("Message"));
          } catch (Exception e) {
                 throw new ServletException(e);
          }
   }
}
```

Nachfolgend erstellen wir eine einfache Web Page mit dem Namen SendMessage.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<Form action=MessageSender>
<TABLE width="30%" border="0">
        <TR>
           <TD>Message</TD>
           <TD><INPUT type="text" name="Message"></TD>
        </TR>
</TABLE>
<input type=submit value="Send"/>
</Form>
</body>
</html>
```

3.3 Die Projekte in WAS ausführen und testen

3.3.1 Die Projekte unter WAS ausführen

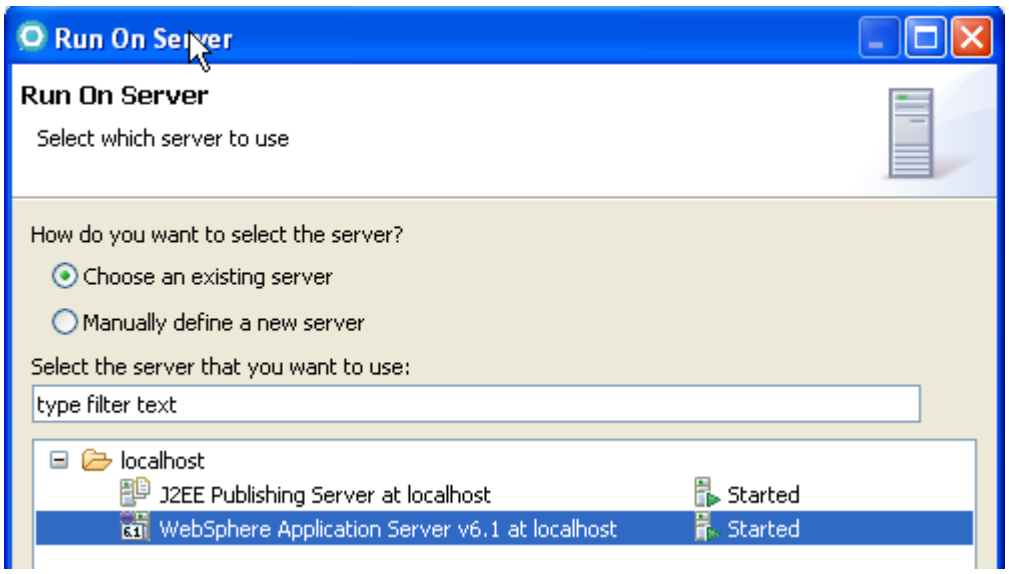


Abbildung 3.3.1

1kr auf das MDBSampleEAR-Projekt, Run As -> Run on Server auswählen, wähle WebSphere Application Server 6.1 at localhost (Abbildung 3.3.1), selektiere die beiden Module MDBSampleEJB und MDBSampleWAR, 1k auf Finish um das Projekt zu starten.

3.3.2 Testen des Projektes

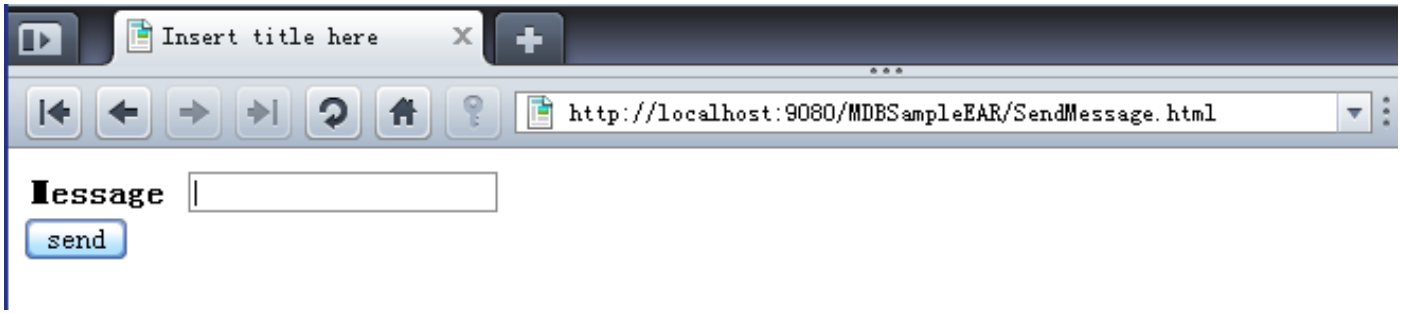


Abbildung 3.3.2-1

Öffnen Sie einen Browser und geben Sie http://localhost:9080/MDBSampleWAR/SendMessage.html in der Addresszeile ein. Die Seite sollte aussehen, wie in Abbildung 3.3.2-1 dargestellt. Geben sie in dem Text Bereich eine beliebige Nachricht ein, z.B. „This is the first message“ und klicken Send.

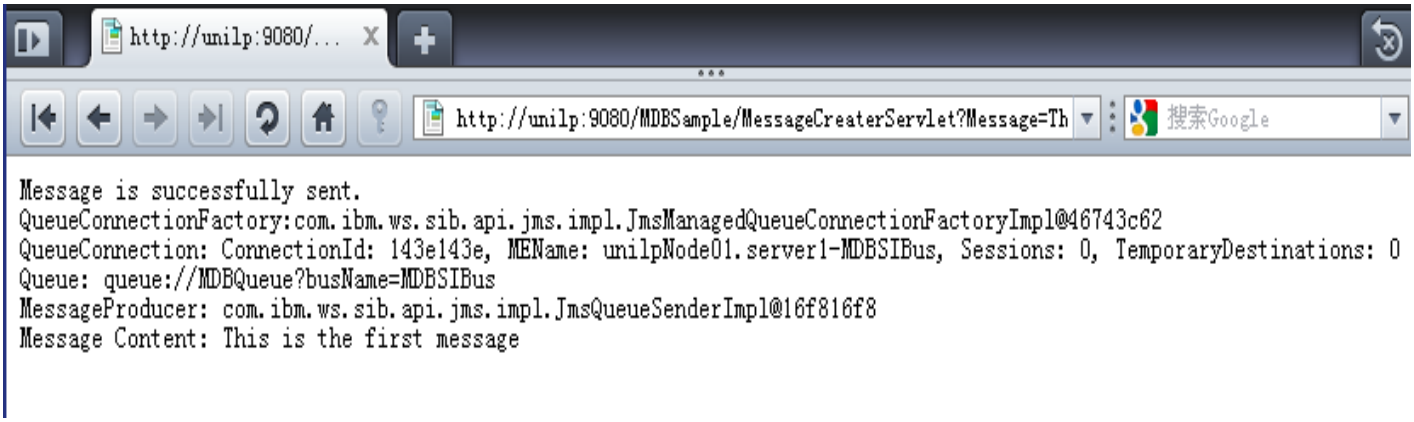


Abbildung 3.3.2-2 Ergebnis

Das Ergebnis sollte aussehen.

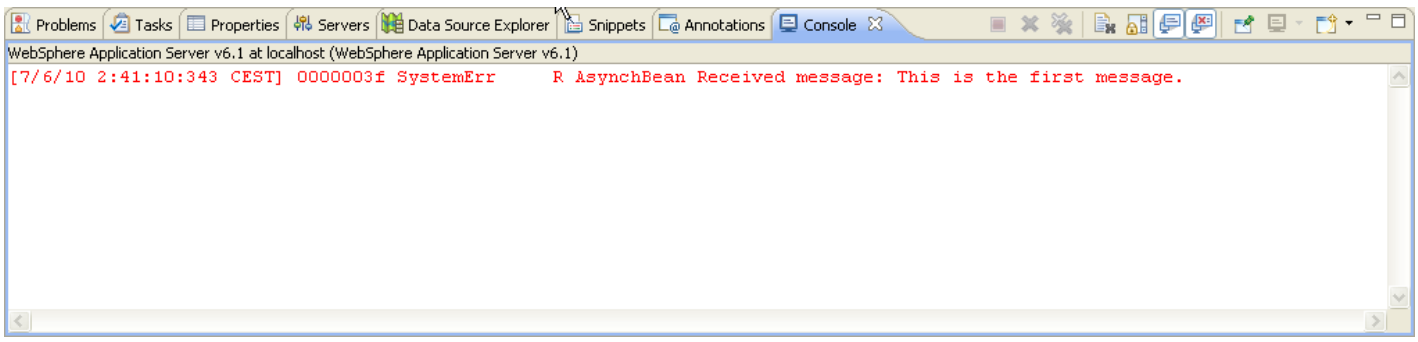


Abbildung 3.3.2-3 Messages in Console

Wechseln Sie zu RD7.5. Das Console-Fenster sollte die in Abbildung 3.3.2-3 gezeigte Nachricht wiedergeben.

Selbst-Test

- **Überlegen Sie sich einige Möglichkeiten: An Stelle nur eine Nachricht in das Eingabefeld zu schreiben, listen Sie 2 – 3 Beispiele, was die Logik in der MDB bewerkstelligen könnte.**

Aufgaben:

1. **Geben Sie in den Message-Text Ihren Namen und ihre PRAK[xxx]-Kennung ein.**
2. **Erstellen Sie anschließend wie in Abbildung 3.3.2-1/2 dargestellt ein Vorher-/Nachher-Screenshot mit Ihren Ergebnissen und laden Sie diese in den Abgabeordner hoch.**