CMP 436/774

Introduction to
JMS (Java Message Service)
MDB (Message Driven Bean)

- **Refereces**
  - Main Reference R1 (chapters 22, 25, 47, 48)

Fall 2012
Department of Mathematics
and Computer Science
Lehman College, CUNY

1

What Is Messaging? (1)

- Messaging is a method of communication between software components or applications.
  - A messaging system is a peer-to-peer facility:
    - A messaging client can send messages to, and receive messages from, any other client.
    - Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages.
- Messaging enables distributed communication that is **loosely coupled**. A component sends a message to a destination, and the recipient can retrieve the message from the destination.
  - The sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender.
  - The sender and the receiver need to know only which message format and which destination to use.

2

What Is Messaging? (2)

- Messaging differs from tightly coupled technologies, such as Remote Method Invocation (RMI), which require an application to know a remote application's methods.
- Messaging also differs from electronic mail (email), which is a method of communication between people or between software applications and people. Messaging is used for communication between software applications or software components.

3

JMS API

- The Java Message Service is a Java API that allows applications to create, send, receive, and read messages.
  - JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations.
  - It also strives to maximize the portability of JMS applications across JMS providers in the same messaging domain.
- The JMS API enables communication that is not only loosely coupled but also
  - Asynchronous: A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
  - Reliable: The JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages.

4

JMS API Work with the Java EE Platform

- When the JMS API was introduced in 1998, its most important purpose was to allow Java applications to access existing messaging-oriented middleware (MOM) systems, such as MQSeries from IBM. Since that time, many vendors have adopted and implemented the JMS API, so a JMS product can now provide a complete messaging capability for an enterprise.
- Beginning with the 1.3 release of the Java EE platform, the JMS API has been an integral part of the platform, and application developers can use messaging with Java EE components.

5

Message Driven Bean (MDB)

- A **message-driven bean** is an enterprise bean that allows Java EE applications to process messages asynchronously.
- Message-driven beans can implement any messaging type. Most commonly, they implement the Java Message Service (JMS) technology.
  - This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events.
  - The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology.
  - Message-driven beans can process JMS messages or other kinds of messages.

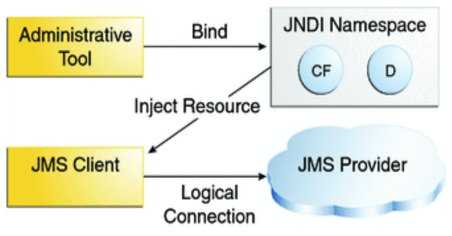
6

JMS Application Architecture

- **A typical JMS application is composed of the following parts.**
  - A JMS provider is a messaging system that implements the JMS interfaces and provides administrative and control features. An implementation of the Java EE platform includes a JMS provider.
  - JMS clients are the programs or components, written in the Java programming language, that produce and consume messages. Any Java EE application component can act as a JMS client.
  - Messages are the objects that communicate information between JMS clients.
  - Administered objects are preconfigured JMS objects created by an administrator for the use of clients. The two kinds of JMS administered objects are destinations and connection factories, which are described in JMS Administered Objects.

JMS Queue and Connection Factory

- The following figure illustrates the way these parts interact.
  - Administrative tools allow you to bind destinations and connection factories into a JNDI namespace.
  - A JMS client can then use resource injection to access the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

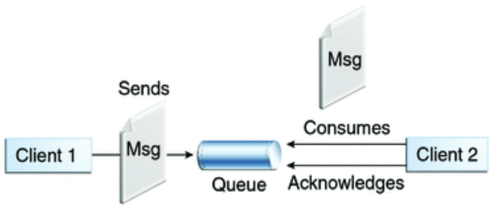


Messaging Domains

- Before the JMS API existed, most messaging products supported either the point-to-point or the publish/subscribe approach to messaging. The JMS specification provides a separate domain for each approach and defines compliance for each domain. A stand-alone JMS provider can implement one or both domains. A Java EE provider must implement both domains.
- In fact, most implementations of the JMS API support both the point-to-point and the publish/subscribe domains, and some JMS clients combine the use of both domains in a single application. In this way, the JMS API has extended the power and flexibility of messaging products.
- JMS provides common interfaces that enable you to use the JMS API in a way that is not specific to either domain.

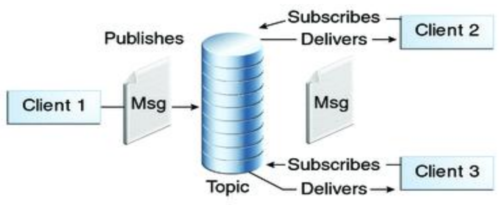
Point to Point (PTP) Messaging Domain

- A **point-to-point** (PTP) product or application is built on the concept of message **queues**, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queues established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.



Publish/Subscribe Messaging Domain

- In a **publish/subscribe** (pub/sub) product or application, clients address messages to a **topic**, which functions somewhat like a bulletin board.
  - Publishers and subscribers are generally anonymous and can dynamically publish or subscribe to the content hierarchy.
  - The system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers.
  - Topics retain messages only as long as it takes to distribute them to current subscribers.



11

Programming with the Common Interfaces

- JMS API allows you to use the same code to send and receive messages under either the PTP or the pub/sub domain.
- The destinations that you use remain domain-specific, and the behavior of the application will depend in part on whether you are using a queue or a topic.
  - However, the code itself can be common to both domains, making your applications flexible and reusable.

12

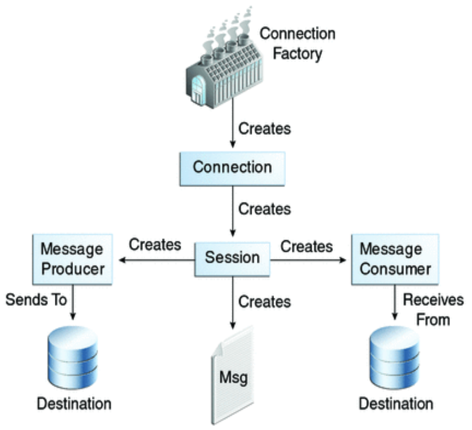
Messaging Consumption

- Messaging products are inherently asynchronous: There is no fundamental timing dependency between the production and the consumption of a message.
- Messages can be consumed in either of two ways:
  - Synchronously: A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
  - Asynchronously: A client can register a message listener with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage method, which acts on the contents of the message.

13

JMS API Programming Model

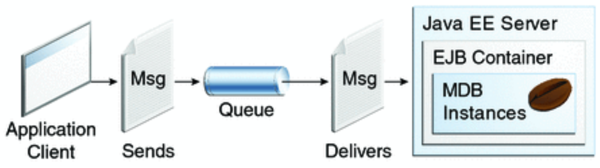
- The basic building blocks of a JMS application consist of
  - Administered objects: connection factories and destinations
  - Connections
  - Sessions
  - Message producers
  - Message consumers
  - Messages



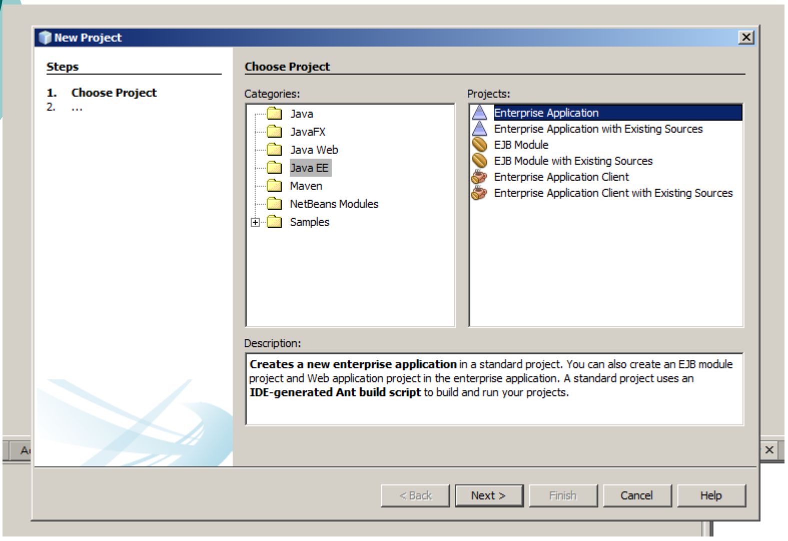
14

Simple JMS Project (JMSProject)

- **JMSProject**
  - The simple message application has the following components:
  - JMSClient: An application client that sends a simple message to a queue
  - Simple MDB EJB MessageReceiver.java : A message-driven bean that asynchronously receives and processes the message that are sent to the queue
  - Message queue myQueue is created as a JMS resource by JMSProject.

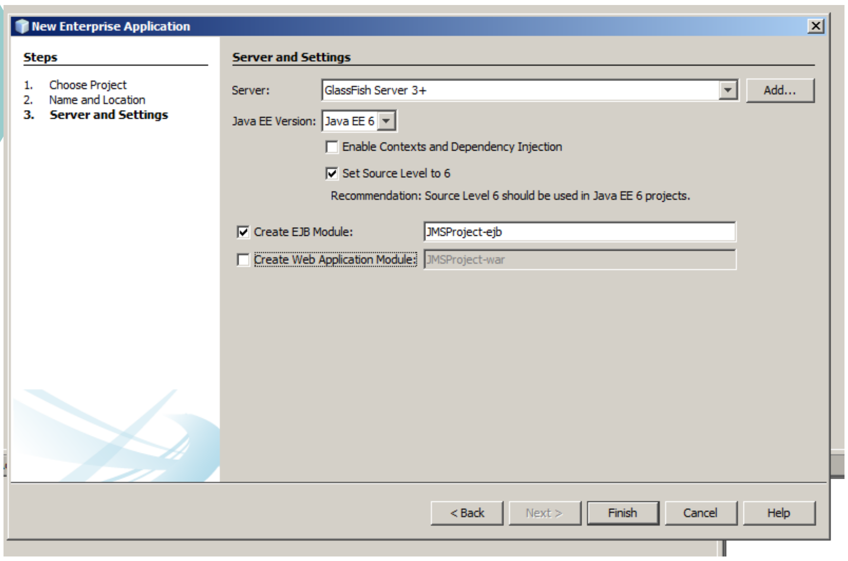


JMS Project –Enterprise App(1)



Creates a new enterprise application in a standard project. You can also create an EJB module project and Web application project in the enterprise application. A standard project uses an **IDE-generated Ant build script** to build and run your projects.

JMS Project –Enterprise App (2)



New Enterprise Application

Steps
1. Choose Project
2. Name and Location
3. **Server and Settings**

Server and Settings

Server: GlassFish Server 3+ [Add...]

Java EE Version: Java EE 6

☐ Enable Contexts and Dependency Injection

☑ Set Source Level to 6

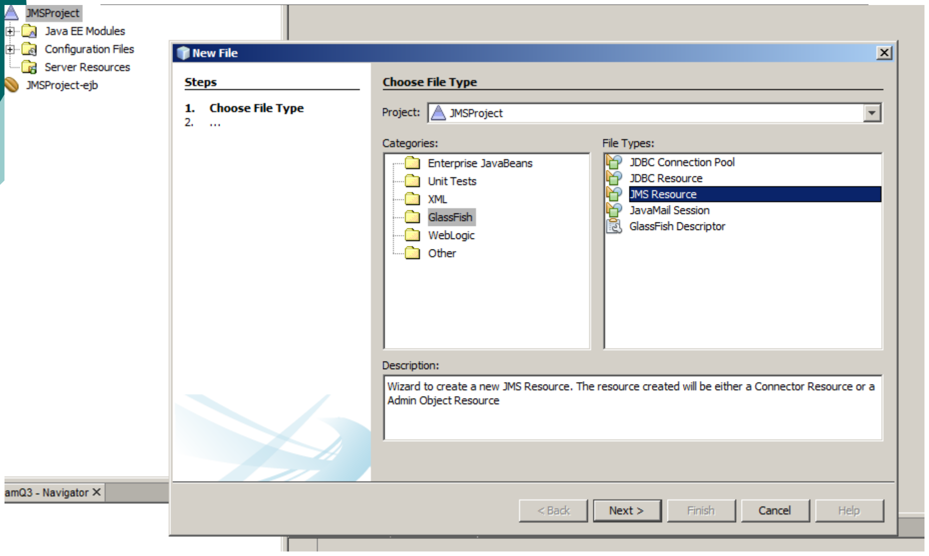
Recommendation: Source Level 6 should be used in Java EE 6 projects.

☑ Create EJB Module: JMSProject-ejb

☐ Create Web Application Module: JMSProject-war

17

Creating Message Queue



New File

Steps
1. **Choose File Type**
2. ...

Choose File Type

Project: JMSProject

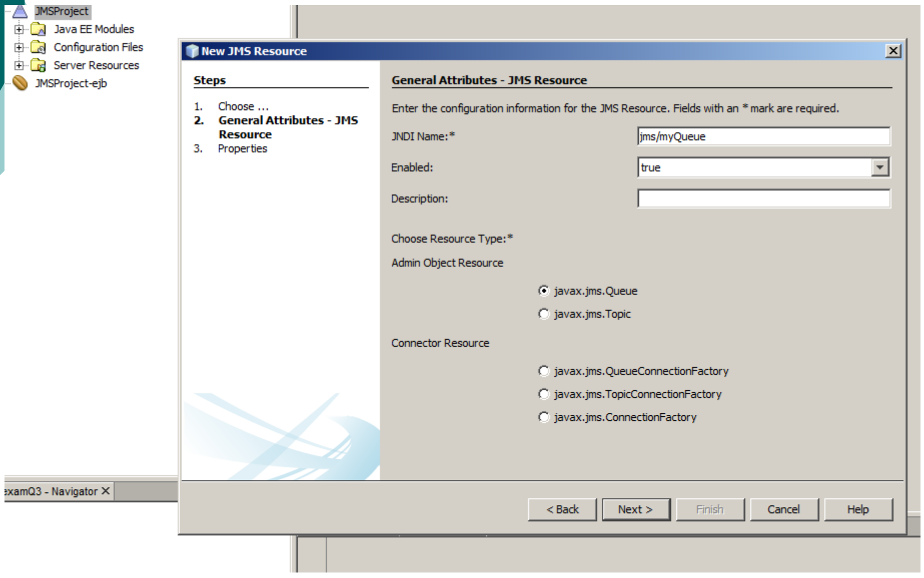
Categories:	File Types:
Enterprise JavaBeans	JDBC Connection Pool
Unit Tests	JDBC Resource
XML	JMS Resource
GlassFish	JavaMail Session
WebLogic	GlassFish Descriptor
Other	

Description:

Wizard to create a new JMS Resource. The resource created will be either a Connector Resource or a Admin Object Resource

18

Enter JNDI name



Steps

1. Choose ...
2. **General Attributes - JMS Resource**
3. Properties

General Attributes - JMS Resource

Enter the configuration information for the JMS Resource. Fields with an * mark are required.

JNDI Name:* jms/myQueue

Enabled: true

Description:

Choose Resource Type:*

Admin Object Resource

[✓ javax.jms.Queue]

javax.jms.Topic

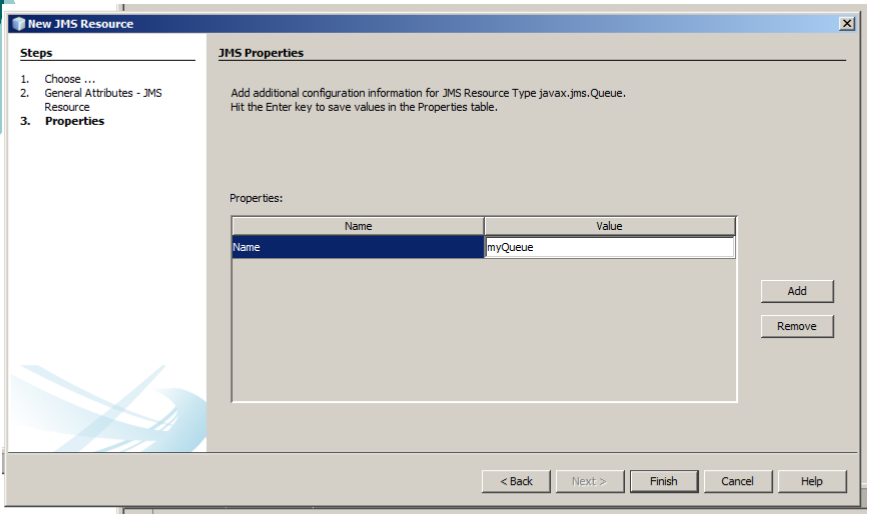
Connector Resource

javax.jms.QueueConnectionFactory

javax.jms.TopicConnectionFactory

javax.jms.ConnectionFactory

JMS Property



Steps

1. Choose ...
2. General Attributes - JMS Resource
3. **Properties**

JMS Properties

Add additional configuration information for JMS Resource Type javax.jms.Queue.
Hit the Enter key to save values in the Properties table.

Properties:

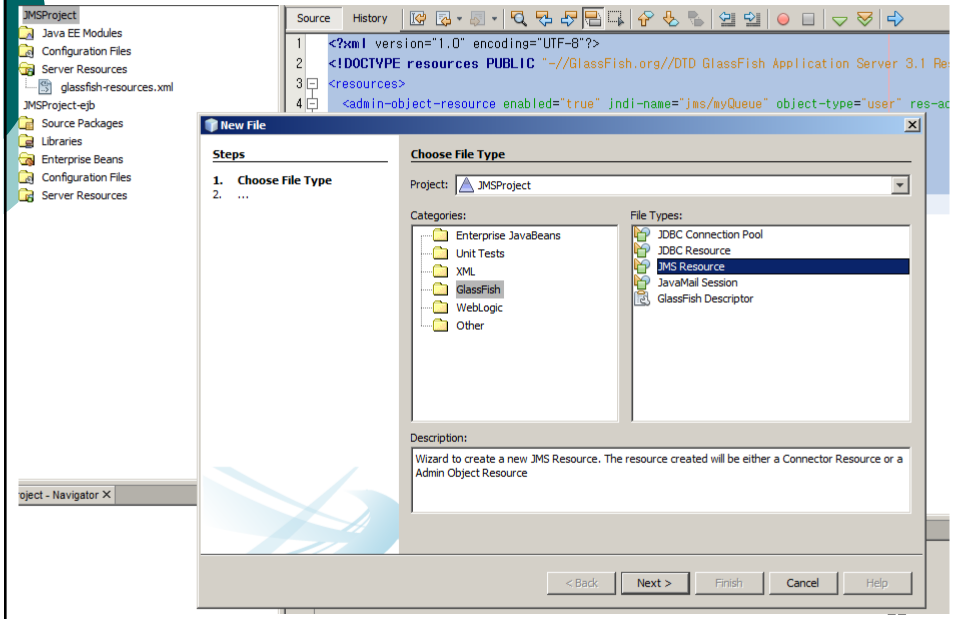
Name	Value
Name	myQueue

glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <admin-object-resource enabled="true" jndi-name="jms/myQueue" object-type="user" res-adapter="jmsra" res-type="javax.jms.Queue">
    <description/>
    <property name="Name" value="myQueue"/>
  </admin-object-resource>
</resources>
```

21

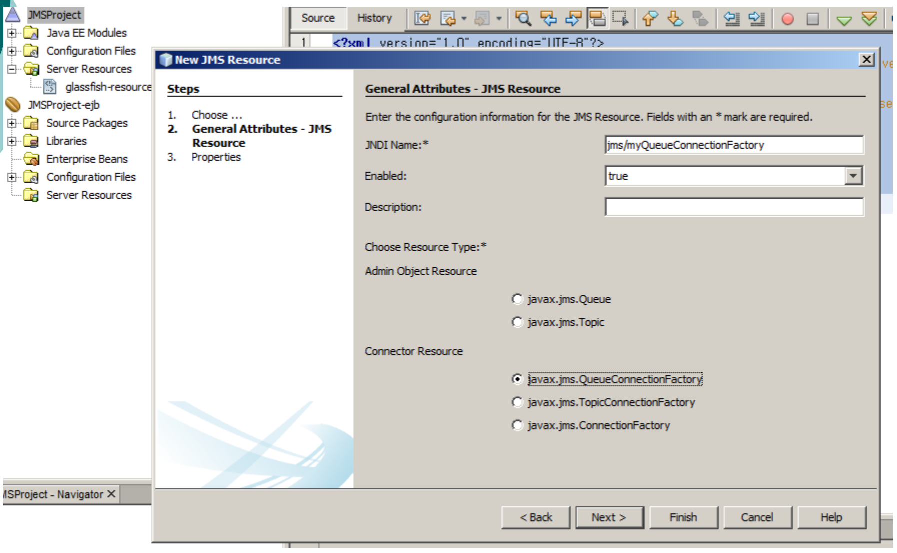
Create JMS Connection Factory(1)



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Re
<resources>
  <admin-object-resource enabled="true" jndi-name="jms/myQueue" object-type="user" res-ad
```

Wizard to create a new JMS Resource. The resource created will be either a Connector Resource or a Admin Object Resource

Create JMS Connection Factory(2)



```
<?xml version="1.0" encoding="UTF-8"?>
```

General Attributes - JMS Resource

Enter the configuration information for the JMS Resource. Fields with an * mark are required.

JNDI Name:*	jms/myQueueConnectionFactory
Enabled:	true
Description:	

Choose Resource Type:*

Admin Object Resource

- ( ) javax.jms.Queue
- ( ) javax.jms.Topic

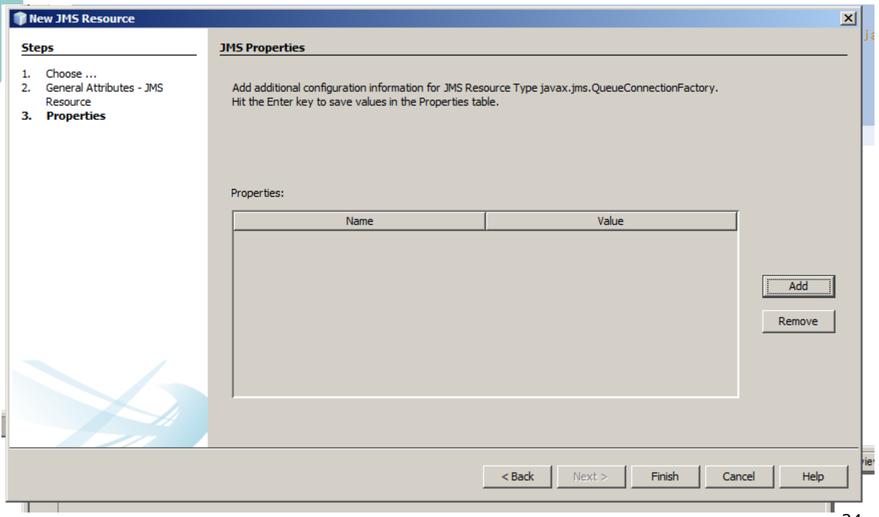
Connector Resource

- [✓ javax.jms.QueueConnectionFactory]
- ( ) javax.jms.TopicConnectionFactory
- ( ) javax.jms.ConnectionFactory

23

Create JMS Connection Factory(3)

- Do not enter properties. Click Finish



JMS Properties

Add additional configuration information for JMS Resource Type javax.jms.QueueConnectionFactory.
Hit the Enter key to save values in the Properties table.

Properties:

Name	Value

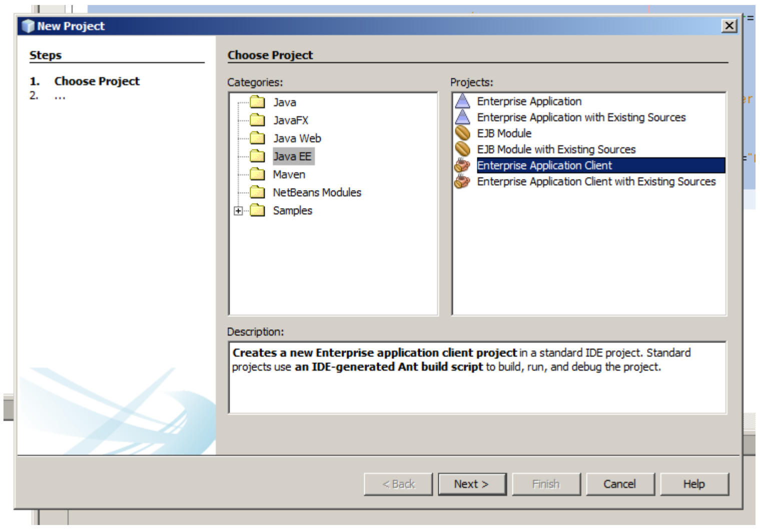
24

glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <admin-object-resource enabled="true" jndi-name="jms/myQueue" object-type="user" res-
adapter="jmsra" res-type="javax.jms.Queue">
    <description/>
    <property name="Name" value="myQueue"/>
  </admin-object-resource>
  <connector-resource enabled="true" jndi-name="jms/myQueueConnectionFactory" object-
type="user" pool-name="jms/myQueueConnectionFactory">
    <description/>
  </connector-resource>
  <connector-connection-pool associate-with-thread="false" connection-creation-retry-
attempts="0" connection-creation-retry-interval-in-seconds="10" connection-definition-
name="javax.jms.QueueConnectionFactory" connection-leak-reclaim="false" connection-leak-
timeout-in-seconds="0" fail-all-connections="false" idle-timeout-in-seconds="300" is-
connection-validation-required="false" lazy-connection-association="false" lazy-connection-
enlistment="false" match-connections="true" max-connection-usage-count="0" max-pool-
size="32" max-wait-time-in-millis="60000" name="jms/myQueueConnectionFactory" ping="false"
pool-resize-quantity="2" pooling="true" resource-adapter-name="jmsra" steady-pool-size="8"
validate-atmost-once-period-in-seconds="0"/>
</resources>
```

25

Enterprise Application Client (1)



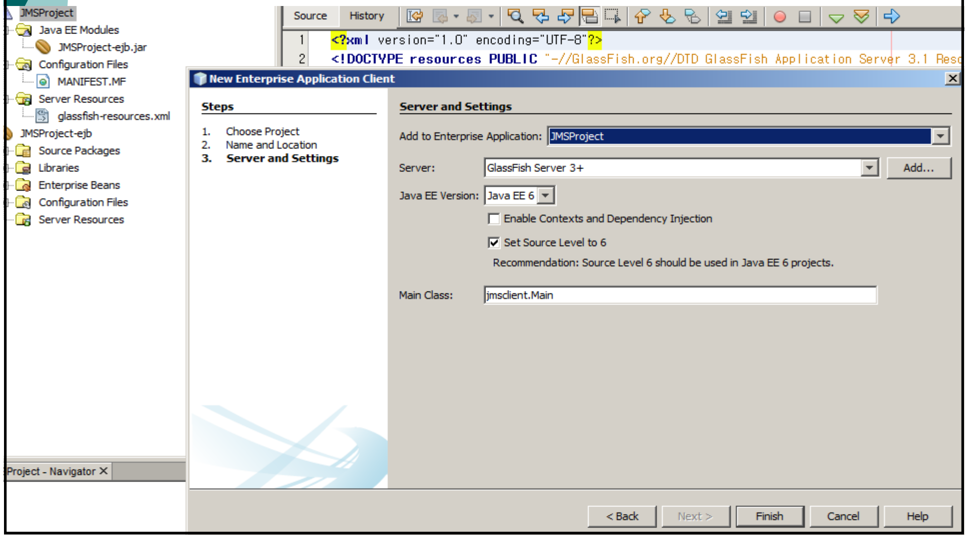
Description:

Creates a new Enterprise application client project in a standard IDE project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug the project.

26

Enterprise Application Client (2)

- Make sure JMSProject is selected (Add to Enterprise Application option)



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Reso
```

New Enterprise Application Client

Steps
1. Choose Project
2. Name and Location
3. Server and Settings

Server and Settings

Add to Enterprise Application: JMSProject

Server: GlassFish Server 3+ Add...

Java EE Version: Java EE 6

☐ Enable Contexts and Dependency Injection

☑ Set Source Level to 6

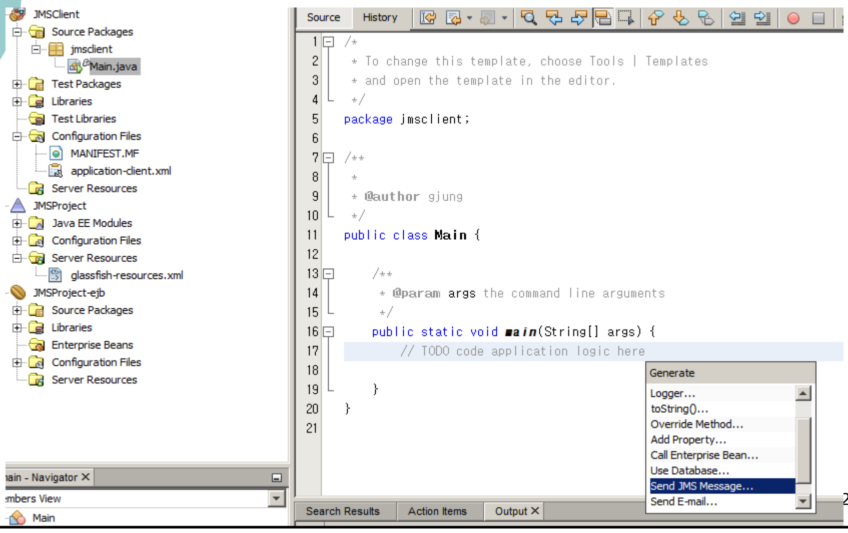
Recommendation: Source Level 6 should be used in Java EE 6 projects.

Main Class: jmsclient.Main

< Back | Next > | Finish | Cancel | Help

Enterprise Application Client (3)

- Right Click Main, select Insert Code then Select Send JMS Message

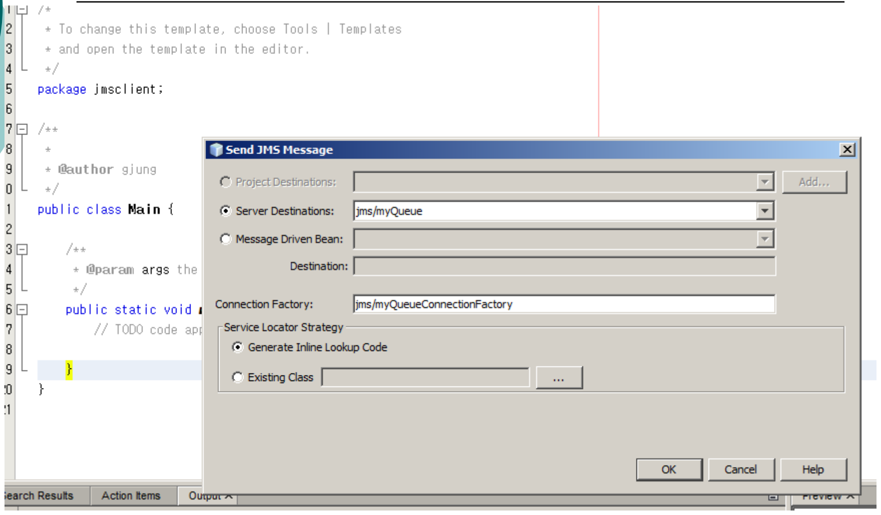


```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package jmsclient;

/**
 *
 * @author gjung
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here

    }
}
```

Generate
Logger...
toString()...
Override Method...
Add Property...
Call Enterprise Bean...
Use Database...
Send JMS Message...
Send E-mail...

28

Enterprise Application Client (4)



```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package jmsclient;

/**
 *
 * @author gjung
 */
public class Main {

    /**
     * @param args the
     */
    public static void 
        // TODO code app
    }
}
```

Send JMS Message

- ( ) Project Destinations:
- (●) Server Destinations: jms/myQueue
- ( ) Message Driven Bean:

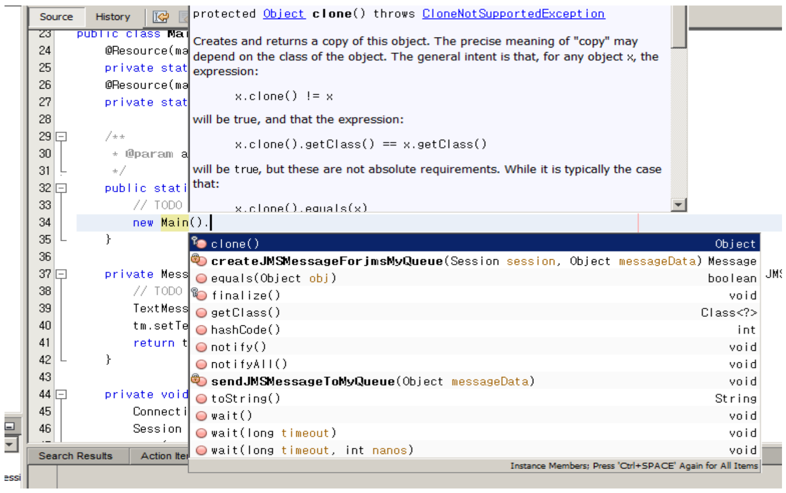
Destination:

Connection Factory: jms/myQueueConnectionFactory

Service Locator Strategy
- (●) Generate Inline Lookup Code
- ( ) Existing Class

29

Enterprise Application Client (5)



```
public class Mai
    @Resource(ma
    private stat
    @Resource(ma
    private stat

    /**
     * @param a
     */
    public stati
        // TODO
        new Main().
    }

    private Mess
        // TODO
        TextMess
        tm.setTe
        return t
    }

    private void
        Connecti
        Session
```

protected Object **clone**() throws CloneNotSupportedException

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that, for any object x, the expression:

```
x.clone() != x
```

will be true, and that the expression:

```
x.clone().getClass() == x.getClass()
```

will be true, but these are not absolute requirements. While it is typically the case that:

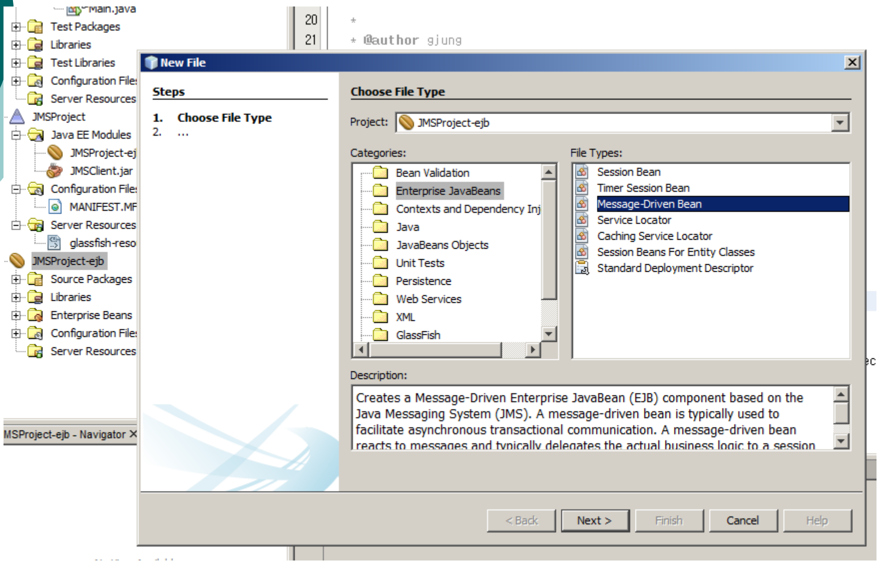
```
x.clone().equals(x)
```

```
clone()                                                           Object
createJMSMessageForjmsMyQueue(Session session, Object messageData) Message
equals(Object obj)                                                boolean
finalize()                                                           void
getClass()                                                      Class<?>
hashCode()                                                           int
notify()                                                            void
notifyAll()                                                         void
sendJMSMessageToMyQueue(Object messageData)                         void
toString()                                                        String
wait()                                                              void
wait(long timeout)                                                  void
wait(long timeout, int nanos)                                       void
```

Instance Members; Press 'Ctrl+SPACE' Again for All Items

30

Message Driven Bean (1)


New File

Steps

1. Choose File Type
2. ...

Choose File Type

Project: JMSProject-ejb

Categories: Bean Validation, Enterprise JavaBeans, Contexts and Dependency Inj, Java, JavaBeans Objects, Unit Tests, Persistence, Web Services, XML, GlassFish

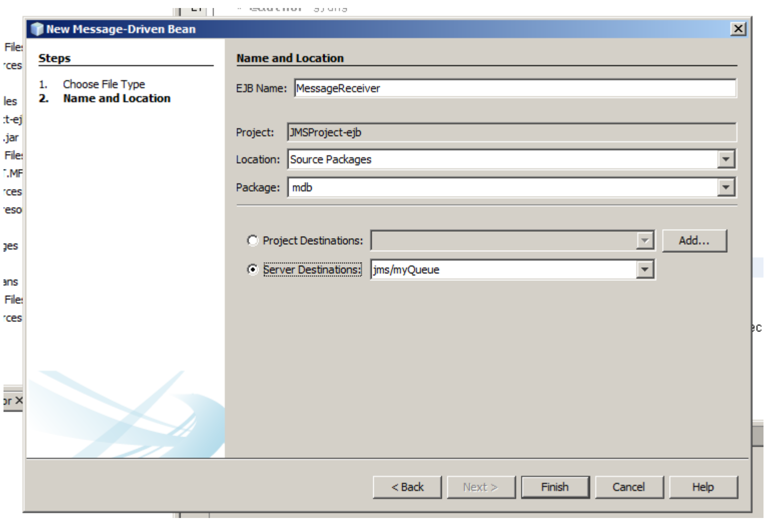
File Types: Session Bean, Timer Session Bean, Message-Driven Bean, Service Locator, Caching Service Locator, Session Beans For Entity Classes, Standard Deployment Descriptor

Description:

Creates a Message-Driven Enterprise JavaBean (EJB) component based on the Java Messaging System (JMS). A message-driven bean is typically used to facilitate asynchronous transactional communication. A message-driven bean reacts to messages and typically delegates the actual business logic to a session

31

Message Driven Bean (2)


New Message-Driven Bean

Steps

1. Choose File Type
2. Name and Location

Name and Location

EJB Name: MessageReceiver

Project: JMSProject-ejb

Location: Source Packages

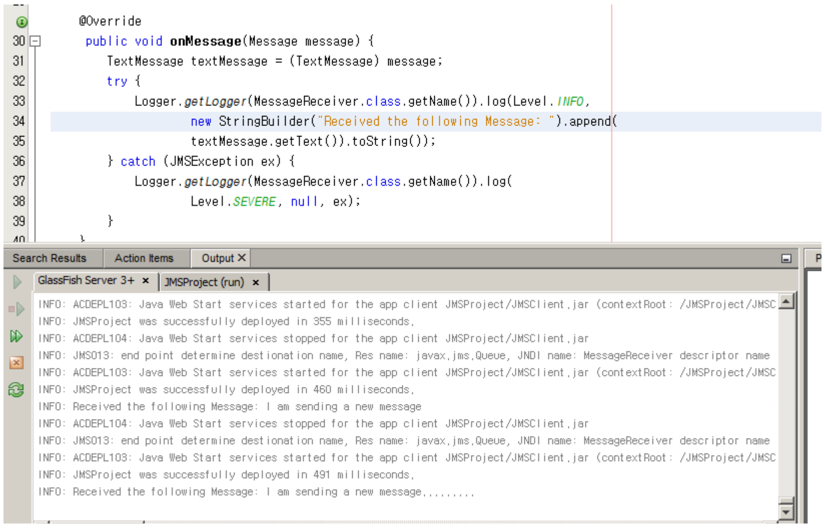
Package: mdb

Project Destinations: [ ]

[✓ Server Destinations: jms/myQueue]

32

Message Driven Bean (3)



```java
    @Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            Logger.getLogger(MessageReceiver.class.getName()).log(Level.INFO,
                    new StringBuilder("Received the following Message: ").append(
                    textMessage.getText()).toString());
        } catch (JMSException ex) {
            Logger.getLogger(MessageReceiver.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }
```

```
INFO: ACDEPL103: Java Web Start services started for the app client JMSProject/JMSClient.jar (contextRoot: /JMSProject/JMSC
INFO: JMSProject was successfully deployed in 355 milliseconds.
INFO: ACDEPL104: Java Web Start services stopped for the app client JMSProject/JMSClient.jar
INFO: JMS013: end point determine destionation name, Res name: javax.jms.Queue, JNDI name: MessageReceiver descriptor name
INFO: ACDEPL103: Java Web Start services started for the app client JMSProject/JMSClient.jar (contextRoot: /JMSProject/JMSC
INFO: JMSProject was successfully deployed in 460 milliseconds.
INFO: Received the following Message: I am sending a new message
INFO: ACDEPL104: Java Web Start services stopped for the app client JMSProject/JMSClient.jar
INFO: JMS013: end point determine destionation name, Res name: javax.jms.Queue, JNDI name: MessageReceiver descriptor name
INFO: ACDEPL103: Java Web Start services started for the app client JMSProject/JMSClient.jar (contextRoot: /JMSProject/JMSC
INFO: JMSProject was successfully deployed in 491 milliseconds.
INFO: Received the following Message: I am sending a new message.........
```