Enterprise JavaBeans™
EJB™ 3.1 Technology
Overview

Kenneth Saks
EJB 3.1 Specification Lead
Agenda

- Introduction
- New functionality
EJB 3.1 Specification

• Goals
  > Continued focus on ease-of-use
  > New features

• JSR (Java Specification Request) 318
  > Expert Group Formed – August 2007
  > Public Draft – October 2008
  > Proposed Final Draft – March 2009
  > Final Specification – December 2009
Ease of Use Improvements

- Optional Local Business Interfaces
- Simplified Packaging
- EJB 3.1 “Lite” API
- Portable JNDI Names
- Simple Component Testing
Session Bean with Local Business Interface

HelloBean Client

```java
@EJB
private Hello h;
...

h.sayHello();
```

```
<<interface>
com.acme.Hello
String sayHello()

com.acme.HelloBean

public String sayHello()
{
  ... }
Session Bean with “No-interface” View

@Stateless
public class HelloBean {

    public String sayHello(String msg) {
        return "Hello " + msg;
    }

}
No-interface View Client

```
@EJB HelloBean h;

...  

h.sayHello("bob");
```
Java™ EE Platform 5 Packaging

**foo.ear**

- **foo_web.war**
  - WEB-INF/web.xml
  - WEB-INF/classes/
    - com/acme/FooServlet.class
  - WEB-INF/classes
    - com/acme/Foo.class
- **foo_ejb.jar**
  - com/acme/FooBean.class
  - com/acme/Foo.class

**foo.ear**

- **lib/foo_common.jar**
  - com/acme/Foo.class
- **foo_web.war**
  - WEB-INF/web.xml
  - WEB-INF/classes/
    - com/acme/FooServlet.class
- **foo_ejb.jar**
  - com/acme/FooBean.class
  - com/acme/Foo.class
Simplified Packaging

foo.war

WEB-INF/classes/com/acme/
FooServlet.class
FooBean.class
EJB 3.1 “Lite”

** Lite **

- Local Session Beans
- CMT / BMT
- Declarative Security
- Interceptors

** Full = Lite + : **

- Message-Driven Beans
- Web Service Endpoints
- 2.x / 3.x Remote view
- RMI-IIOP Interoperability
- Timer Service
- Async method calls
- 2.x Local view
- CMP / BMP Entity Beans

** Web Profile also includes Java Persistence API **
Portable EJB JNDI Names

Each session bean gets the following entries:

• Globally unique name
  \texttt{java:global[/<app-name>]/<module-name>/<ejb-name>}

• Unique name within same application
  \texttt{java:app/<module-name>/<ejb-name>}

• Unique name within defining module
  \texttt{java:module/<ejb-name>
Session Bean

@Stateless

public class HelloBean implements Hello {
    public String sayHello(String msg) {
        return "Hello " + msg;
    }
}

If deployed as hello.jar, JNDI entries are:

java:global/hello/HelloBean
java:app/hello/HelloBean
java:module/HelloBean
EJB Component Testing

• It's too hard to test EJB components, especially Local session beans
  > Forced to go through Remote facade or Web tier
  > Separate processes needed for server and client

• Some support for client-side EJB component execution exists, but...
  > Not present in all implementations
  > No standard behavior for bootstrapping, component discovery, shutdown etc.
Simple Testing : Session Bean

@Stateless
@Local(Bank.class)

public class BankBean implements Bank {

    @PersistenceContext EntityManager accountDB;

    public String createAccount(AcctDetails d)
    {
        ... 
    }

    public void removeAccount(String acctID)
    {
        ... 
    }
public class BankTester {
    public static void main(String[] args) {

        EJBContainer container =
            EJBContainer.createEJBContainer();

        // Acquire Local EJB reference
        Bank bank = (Bank) container.getContext().lookup("java:global/bank/BankBean");

        testAccountCreation(bank);
        container.close();
    }
}
Test Client Execution

% java -classpath bankClient.jar :
    bank.jar :
    javaee.jar :
    <vendor_rt>.jar

    com.acme.BankTester
New Features

• Singletons
• Startup / Shutdown callbacks
• Automatic timer creation / Calendar-based timers
• Asynchronous session bean invocations
Singletons

• New session bean component type
  > Provides easy sharing of state within application
  > Designed for concurrent access
  > One bean instance per singleton component per process

• Lots in common with stateless / stateful beans
  > Client views (Local, Remote, Web Service)
  > CMT / BMT
  > Container services: resource managers, timer service, method authorization, etc.
Simple Singleton

@Singleton

public class SharedBean {

    private SharedData shared;

    @PostConstruct
    private void init() {
        shared = ...;
    }

    public int getXYZ() {
        return shared.xyz;
    }
}
Singleton Client

@Stateless
public class FooBean {

    // Inject reference to Singleton bean
    @EJB
    private SharedBean shared;

    public void foo() {
        int xyz = shared.getXYZ();
        ...
    }
}
Singleton Concurrency Options

- Single threaded (default)
  > For consistency with all existing bean types
- Container Managed Concurrency
  > Control access via method-level locking metadata
- Bean Managed Concurrency
  > All concurrent invocations have access to bean instance
Startup / Shutdown Callbacks

@Singleton
@Startup
public class StartupBean {

    @PostConstruct
    private void onStartup() { ... }

    @PreDestroy
    private void onShutdown() { ... }

}
Calendar Expression Examples

• “The last Thursday in November at 2 p.m.”
  > (hour="14", dayOfMonth="Last Thu", month="Nov")

• “Every day at 3:15 a.m. U.S. Eastern Time”
  > (minute="15", hour="3", timezone="America/New_York")

• “Every twenty seconds”
  > (second="*/20", minute="*", hour="*")
Automatic Timer Creation

@Stateless

public class BankBean {

  @PersistenceContext EntityManager accountDB;
  @Resource javax.mail.Session mailSession;

  // Callback the last day of each month at 8 a.m.

  @Schedule(hour="8", dayOfMonth="Last")
  void sendMonthlyBankStatements() {
    ...
  }
}


Asynchronous Session Bean Invocations

- *Simple* way to add Remote or Local asynchrony to enterprise bean applications
- “Fire and Forget” or async results via `Future<V>`
- Best effort delivery – persistent delivery guarantees are not required by spec
- Available for Stateful, Stateless, Singleton beans
Local Concurrent Computation

@Stateless public class DocBean {

  @PersistenceContext EntityManager resultsDB;
  @EJB DocBean myself;

  public void processDocument(Document document) {
    myself.doAnalysisA(document);
    myself.doAnalysisB(document);
  }

  @Asynchronous public void doAnalysisA(Document d) {...}

  @Asynchronous public void doAnalysisB(Document d) {...}
Asynchronous Results

- Based on `java.util.concurrent.Future`
- Result value is returned via `Future.get()`
  - Also supports `Future.get(long, TimeUnit)`
- Client exception wrapped by `ExecutionException`
  - `getCause()` returns same exception as would have been thrown by a synchronous invocation
Asynchronous Results -- Client

@EJB Processor processor;

Task task = new Task(...);

Future<int> computeTask = processor.compute(task);

...

int result = computeTask.get();
Asynchronous Results

@Stateless
public class ProcessorBean implements Processor {

    @PersistenceContext EntityManager db;

    @Asynchronous
    public Future<int> compute(Task t) {

        // perform computation
        int result = ...;

        return new javax.ejb.AsyncResult<int>(result);
    }
}
Resources
Java EE 6 and GlassFish v3

- Java EE 6 Home
  java.sun.com/javaee
- Java EE 6 Downloads
  java.sun.com/javaee/downloads
- Upcoming Training
  java.sun.com/javaee/support/training
- Sun GlassFish Enterprise Server v3 Home
  www.sun.com/glassfishv3
- Community Page
  glassfish.org
- The Aquarium Blog
  blogs.sun.com/theaquarium
- White Papers/Webinars
  http://www.sun.com/glassfish/resources
Java EE6 Learning Path

**Training**

- Developing Web Applications using JSF Technologies (SL-340-EE6)
- Web Component Development with Servlet and JSP (SL-314-EE6)
- Developing Web Services Using Java Technology (DWS-4050-EE6)
- Developing Secure Java Web Services (DWS-4120-EE6)
- Building Database Driven Applications with JPA (SL-370-EE6)
- Business Component Development with EJB Technology (SL-355-EE6)

**Certifications**

- Sun Certified JSF Developer
- Sun Certified Servlet and JSP Developer
- Sun Certified Web Services Developer
- Sun Certified JPA Developer
- Sun Certified EJB Developer

**Skills**

- Business Component Development with EJB Technology
- Web Services Developer
- Enterprise Web Developer
- Business App Developer
- Web Services Developer

**Courses**

- Sun Certified Java Programmer*
- Developing Applications for the Java EE Platform (FJ-310-EE6)
Updated EJB 3.1 Training & Certification

• Completely updated Training course for EJB 3.1
  > Business Component Development with EJB Technology (SL-355-EE6) – 3 Days

• Includes coverage of:
  > Implement business-tier functionality using EJB technology
  > Best practices and other advanced issues in business component development with EJB technology
  > Integrate an EJB technology-based application using the Java Messaging Service API
  > Transactions, Security and more

• Register your interest!
  > https://dct.sun.com/dct/forms/reg_us_1611_480_0.jsp
Enterprise JavaBeans™
EJB™ 3.1 Technology

Overview

Kenneth Saks
kenneth.saks@sun.com