

At crucial moments  
you want an *IT partner*  
you can trust



# Enterprise JavaBeans 3.0

## Migration Guide

**infoSupport**

Kruisboog 42, 3905 TG, Veenendaal  
Tel. +31 (0)318 - 55 20 20, Fax +31 (0)318 - 55 23 55  
info.nl@infosupport.com

[www.infosupport.com](http://www.infosupport.com)

At crucial moments  
you want an *IT partner*  
you can trust

# Bert Ertman

IT Architect  
Competence Center Java  
Info Support BV

[berte@infosupport.com](mailto:berte@infosupport.com)

Bert is a member of the NL-JUG  
executive committee, the  
official Dutch Java User Group

[bert.ertman@nljug.org](mailto:bert.ertman@nljug.org)



**infoSupport**

Kruisboog 42, 3905 TG, Veenendaal  
Tel. +31 (0)318 - 55 20 20, Fax +31 (0)318 - 55 23 55  
[info.nl@infosupport.com](mailto:info.nl@infosupport.com)

[www.infosupport.com](http://www.infosupport.com)

# Agenda

- Enterprise JavaBeans 3.0 - why should we migrate?
- Migrating Session Beans
- Migrating Client Views
- Migrating Message Driven Beans
- Migrating Entity Beans to new Persistence API
- Migrating POJOs to new Persistence API
- Resources, Q&A

Enterprise JavaBeans 3.0 –  
Why should we migrate?

## What's wrong with EJB 2.x?

- Current EJB spec is too difficult to get started
- Even simple applications need tedious boilerplate
  - Too many classes, interfaces, concepts
  - Complex JNDI lookups
  - Awkward programming model
  - Deployment descriptors
  - Entity Bean anti-patterns
- Hard to implement unit testing
  - Mocking / in-container testing is hard and cumbersome!

## What's new in EJB 3.0?

- Simplified Component Model
  - POJO/POJI-based component definition
    - No required Container interfaces
    - No required Deployment Descriptors
  - Dependency Injection
  - Simple lookups
  - Attract broader range of developers
- Single Java Persistence API
  - It's a separate 'beast'
  - CMP/BMP Entities are no more...
  - Hibernate, JDO, TopLink, EJB 2.x best practices

## Disclaimer ;-)

- Simplification is not feature reduction!
  - Simplifying a programming model does not reduce its functionality. Simplification merely hides the complex logic behind framework code or reusable components. In essence, it shifts complexity from one part of the system that needs explicit management by application developers to another part invisible to most developers.
- Bean Contract now benefits developer rather than Container!

## Why should we migrate?

- Improved maintainability
- Better productivity (lower development costs?)
- Better/more integration with other components/subsystems
- Access to bigger developer pool
- Improved tool support / resources
- Standardization!

## Backward compatibility

- EJB 2.1 and earlier continue to work unchanged in EJB 3 containers
- Spec: 'Retains and improves power of EJB 2.1'
- Manual migration needed?
  - Components can be updated or replaced using EJB 3.0 APIs without affecting existing clients
    - Allows for incremental migration
  - Migrate to new persistence unit?
  - Consider using session façades to control transactional and security behavior

An aerial photograph of a beach with a white rectangular box overlaid on the right side. The box contains the text "Migrating Session Beans". The beach is dark, and the ocean is visible in the lower-left corner.

Migrating Session Beans

## Migrating Session Beans

- Session Beans will become POJOs as described in the Simplified API:
  - Beans don't implement **javax.ejb.SessionBean**
  - Strip out **lifecycle methods** and annotate callbacks that may still be used
  - Beans implement remote/local **business interface(s)**
  - **RemoteExceptions** can be stripped from remote interfaces
  - SFSB: change **ejbCreate()**s to business methods and add removal method annotated with **@Remove**
  - Optionally add **annotations** for security and transactions

## Example: EJB 2.x Session Bean

### EJB 2.x Session Bean – Home Interface

```
public interface BankHome extends javax.ejb.EJBLocalHome {  
    public Bank create() throws CreateException;  
}
```

### EJB 2.x Session Bean – Remote/Local Interface

```
public interface Bank extends javax.ejb.EJBLocalObject {  
    public boolean deposit(double amount, Account account);  
    public boolean transfer(double amount, Account from, Account to);  
    ...  
}
```

### EJB 2.x Session Bean – Bean class

```
public class BankBean implements javax.ejb.SessionBean {  
    public void ejbCreate() {...}  
    public void ejbRemove() {...}  
    public void ejbActivate() {...}  
    public void ejbPassivate() {...}  
    public void setSessionContext(SessionContext ctx) {...}  
  
    public boolean deposit(double amount, Account account) {...}  
    public boolean transfer(double amount, Account from, Account to) {...}  
    ...  
}
```

## Example: EJB 2.x Deployment Descriptor

### EJB 2.x Deployment Descriptor

```
...
<session>
  <ejb-name>BankEJB</ejb-name>
  <local-home>BankHome</local-home>
  <local>Bank</local>
  <ejb-class>com.infosupport.ejb.BankBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Container</transaction-type>
  <resource-ref>
    ...
  </resource-ref>
</session>
...
<assembly-descriptor>
...
</assembly-descriptor>
...
```

## Example: EJB 3.0 Session Bean

### EJB 3.0 Session Bean – Business Interface

```
public interface Bank {  
    public boolean deposit(double amount, Account account);  
    public boolean transfer(double amount, Account from, Account to);  
    ...  
}
```

### EJB 3.0 Session Bean – Bean class

```
@Stateless  
public class BankBean implements Bank {  
    public boolean deposit(double amount, Account account) {...}  
    public boolean transfer(double amount, Account from, Account to) {...}  
    ...  
}
```

## Example: EJB 3.0 Deployment Descriptor

### EJB 3.0 Deployment Descriptor



## How the vök did they do that?

- Configuration By Exception
  - Use of Annotations
    - **@Stateless**, **@Resource**, etc.
  - Use of defaults
    - Transaction type defaults to **CONTAINER**
    - Transaction attribute defaults to **REQUIRED**
    - Security permissions default to **UNCHECKED**
  - Use of **Dependency Injection**

# Migrating Client Views

## Migrating Client Views

- Simplification all the way:
  - Use **instance** directly
  - Do not specify home interface in **ejb-ref**
  - Change home references to use **business interface**
  - **RemoteException / CreateException**, should no longer be explicitly handled
  - SFSB: can use business interface for **create/remove**
  - Optionally replace JNDI lookup code with **dependency injection**
  - **Service Locator** no longer necessary?

## Example: EJB 2.x Client View

### EJB 2.x Client

```
try {  
    Context ctx = new InitialContext();  
  
    BankHome bankHome = (BankHome)  
        ctx.lookup("java:comp/env/ejb/bank");  
  
    Bank bank = bankHome.create();  
    bank.deposit(2500.0, new Account(12345));  
}  
catch (Exception ex) {  
    ...  
}
```

## Example: EJB 3.0 Client View

### EJB 3.0 Client

```
@EJB Bank bank;
```

```
...
```

```
bank.deposit(2500.0, new Account(12345));
```

```
...
```

## Partial Migration?

- Incrementally adopt **Simplified API** features:
  - Make use of **dependency injection** (in XML descriptor) to obtain bean references
  - Use **XML without annotations** to stay on JDK 1.4

## Deployment Descriptors

- Deployment Descriptors are still available as an alternative to annotations:
  - Some developers prefer to **externalize** metadata
  - Can be used to **override** metadata annotations
  - Handle **application-level** metadata (e.g. default interceptors)
  - All **new features** specified with annotations have **XML equivalents**
  - Descriptors can be **sparse** or **full**
  - **Compatible** with EJB 2.x Deployment Descriptors

An aerial photograph of a vast desert landscape, likely a coastal dune area. The terrain is covered in dark sand with several distinct, parallel tracks or paths that curve across the dunes. The lighting creates soft shadows, highlighting the undulating shapes of the sand. The overall color palette is a range of blues and greys, giving it a monochromatic, ethereal feel.

## Migrating Message Driven Beans

## Migrating Message Driven Beans

- Very simple to migrate:
  - New MDBs don't implement **javax.ejb.MessageDrivenBean**
  - Use **annotations** for specifying configuration properties
  - Use **dependency injection** for resources

## Example: Message Driven Bean

### EJB 3.0 Message Driven Bean Example

```
@MessageDriven(mappedName="MDBQueue")
public class MyMDB implements MessageListener {
    public void onMessage(Message msg) {
        // Do something useful with message here...
        ...
    }
}
```

Migrating Entity Beans to  
new Persistence API

## Java Persistence API

- Simplified entity bean programming model
- Improved support for domain modeling
  - Inheritance and polymorphism
- Allow entities to be used outside of the container
  - Enhanced testability!
- Standardize ORM into single Java Persistence API
  - Usable in both Java SE and Java EE
  - Based on best practices from EJB 2.x, Hibernate, JDO, TopLink, etc.
  - Complete query capabilities
    - Includes native SQL query support
- Support for pluggable, third-party persistence providers

## Migrating Entity Beans (1)

- Entities become POJOs as well:
  - Abstract class -> POJO
    - Make class and methods concrete
  - Local/remote **homes** no longer used
  - **ejbCreate()** methods become constructors
  - **ejbRemove()** method disappears
  - **ejbHome()** methods can become static or instance methods on entity
    - leave them the way they are

## Migrating Entity Beans (2)

- Entities become POJOs as well (continued):
  - Convert local/remote interface into **business** interface
  - **getPrimaryKey()** can be a simple accessor method on the bean
  - Allowed to use **this** as a self-reference instead of **getEJBObject()**

## Migrating Entity Beans (3)

- Lifecycle methods:
  - `ejbCreate()` → Constructor(s) or init method(s)
  - `ejbPostCreate()` → Constructor, **@PrePersist** or **@PostPersist**
  - `ejbRemove()` → **@PreRemove**
  - `ejbActivate()` → **@PostLoad**
  - `ejbPassivate()` → ??
  - `ejbSelect()` → Concrete or dynamic query
  - `ejbStore()` → **@PrePersist** or **@PreUpdate**

## Migrating Entity Beans (4)

- Exceptions
  - **CreateException, RemoveException, etc** -> unchecked **EJBException(s)**
- Finders
  - Finders become either **named** or **dynamic** queries
  - EJBQL from DD -> **annotation** or **query**

## Some issues

- What if some entities didn't follow best practices?
  - Non-local entities (remote access)
  - Entity-level transaction demarcation
  - Entity-level method security
- Fortunately, these **bad practices** can no longer be implemented with EJB 3.0 😊
- For migration purposes, **wrap** them into a simplified SFSB
  - Not pretty, and certainly **not recommended!**
  - **Refactor ASAP!**

## So long, Data Transfer Objects!

- Data Transfer Objects
  - **Lightweight** container objects for Entity **data**
  - **Resemble** pretty much the underlying Entity
- No longer needed in EJB 3.0
  - Entities themselves are already DTOs!
- Migration advice:
  - If you have existing DTOs – turn them into entities by adding the logic!

Migrating POJOs to new  
Persistence API

## Migrating Persistent POJOs (1)

- The **GOOD** news:
  - Already are **persistent POJOs**
  - Already **similar** transaction mechanism, session level APIs, etc.
  - Already have **O/R mapping**
- The **BAD** news:
  - Session APIs are **different**
    - Object registration, locking, events, queries, etc.
  - O/R mapping **metadata** is different

## Migrating Persistent POJOs (2)

- Persistence API:
  - **EntityManager** is also available **outside** container
    - Most persistence frameworks will offer EntityManager API (JSR-220 compliant) in addition to proprietary API
- Queries:
  - Most persistence frameworks provide **named queries**
  - **Expose** them as named queries from the **EntityManager**
  - EJB 3 supports **native queries** as well!
  - Can migrate to **EJBQL** query by query

## Migrating Persistent POJOs (3)

- Metadata:
  - By far the largest portion
  - **Manual porting** would be very painful!
  - Migration strategy:
    - Possible to jointly apply annotations as well as vendor specific O/R mappings
    - XML overrides annotations
    - Override at the class level
    - Can migrate mappings a class at a time
  - **Vendor support for mapping metadata formats**

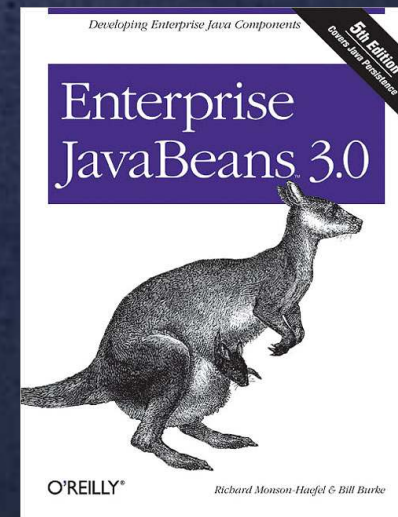
## Migrating Persistent POJOs (4)

- (Vendor specific) features:
  - EJB 3.0 is no kitchen sink spec
    - Too many features just bloat and complicate the specification
    - More vendors => more acceptance => better standard
  - If you rely on a specific vendor feature, isolate it and wrap it in something else
  - As vendor features are prevalent they will eventually become a part of the standard anyway

Resources, Q&A

## Resources

- Enterprise JavaBeans 3.0 (JSR-220):
  - <http://www.jcp.org/en/jsr/detail?id=220>
- Enterprise JavaBeans 3.0, Fifth Edition
  - Richard Monson-Haefel, Bill Burke
  - Fifth edition, May 2006
  - ISBN: 0-596-00978-X



## Shameless self-promotion

- Java Magazine 1, 2006
  - Article: 'Enterprise JavaBeans 3.0 komt eraan!'
- Java Magazine 2, 2006
  - Article: 'JavaOne 2006'
- My Weblog:
  - <http://blogs.infosupport.com/berte>



## SCOOP: Training promotion

- NL-JUG and Info Support will organize and host an EJB 3.0 training in Q3 2006
  - 3 evening sessions
  - Labs!
  - Stay tuned for more information
  - Seats are limited!
- Enrollment starts August 2006

# Q&A