
JavaOne™

JavaOne™

Java EE 7:
What's New in the Java EE Platform

Linda DeMichiel
Java EE 7 Specification Lead

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.

Program Agenda

A look at some of the important new features of Java EE 7

Java EE 7 Themes

- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

Java EE 7 JSRs
New or Updated

- JPA 2.1
- JAX-RS 2.0
- EJB 3.2
- JMS 2.0
- Servlet 3.1
- EL 3.0
- JSF 2.2
- CDI 1.1
- Bean Validation 1.1
- WebSocket 1.0
- JSON 1.0
- Batch Applications 1.0
- Concurrency Utilities 1.0

Java EE 7 Maintenance Releases

- Common Annotations 1.2
- JTA 1.2
- Interceptors 1.2
- Connector 1.7
- JSP 2.3
- JASPIC 1.2
- JACC 1.4
- JavaMail 1.5
- Web Services 1.4

JSON Processing 1.0

- API to parse and generate JSON
- Streaming API (javax.json.stream)
  - Low-level, efficient way to parse/generate JSON
  - Similar to StAX API in XML world
- Object model API (javax.json)
  - Simple, easy to use high-level API
  - Similar to DOM API in XML world

JSON Processing 1.0

Streaming API: Parsing

- Created using
  - Json.createParser(…)
  - Json.createParserFactory().createParser(…)
- Parses JSON in streaming way from input sources

```
Event event = parser.next();   //START_OBJECT
event = parser.next();         // KEY_NAME
event = parser.next();         // VALUE_STRING
```

- Parser state events
  - START_OBJECT, END_OBJECT, START_ARRAY, END_ARRAY, KEY_NAME, VALUE_STRING, VALUE_NUMBER, …

JSON Processing 1.0

Streaming Parser

```
{
    “firstName”: “John”,  “lastName”: “Smith”,  “age”: 25,
    “phoneNumber”: [
          {  “type”: “home”,  “number”:  “212 555-1234” },
          {  “type”: “fax”,  “number”:  “646 555-4567” }
     ]
}
```

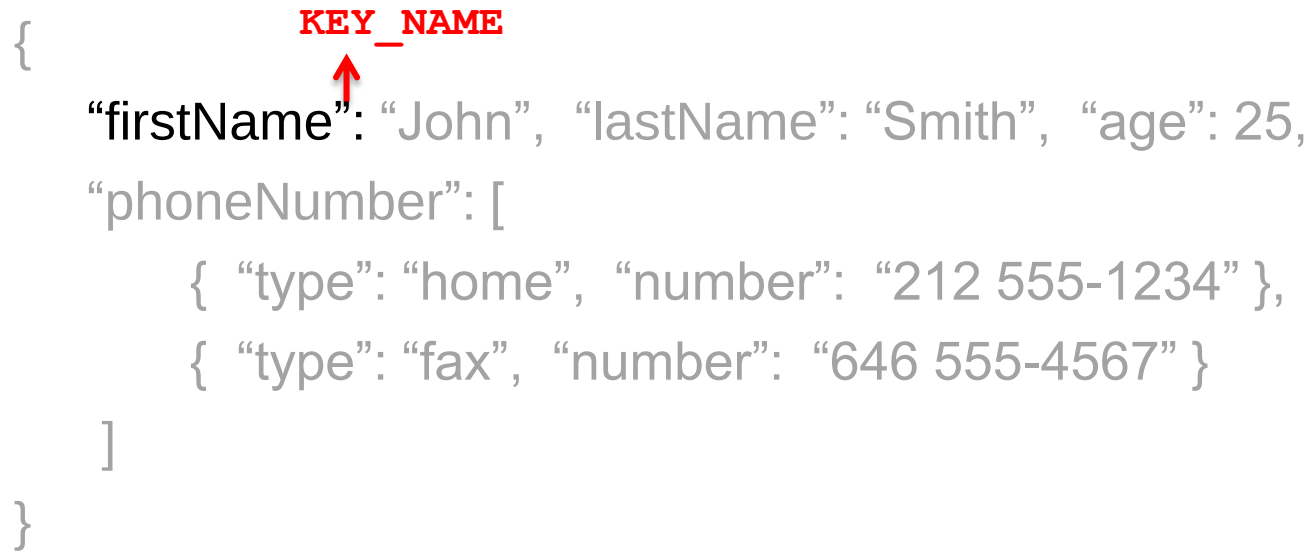
JSON Processing 1.0

Streaming Parser

```
{  START_OBJECT
   “firstName”: “John”,  “lastName”: “Smith”,  “age”: 25,
     “phoneNumber”: [
           {  “type”: “home”,  “number”:  “212 555-1234” },
           {  “type”: “fax”,  “number”:  “646 555-4567” }
      ]
}
```

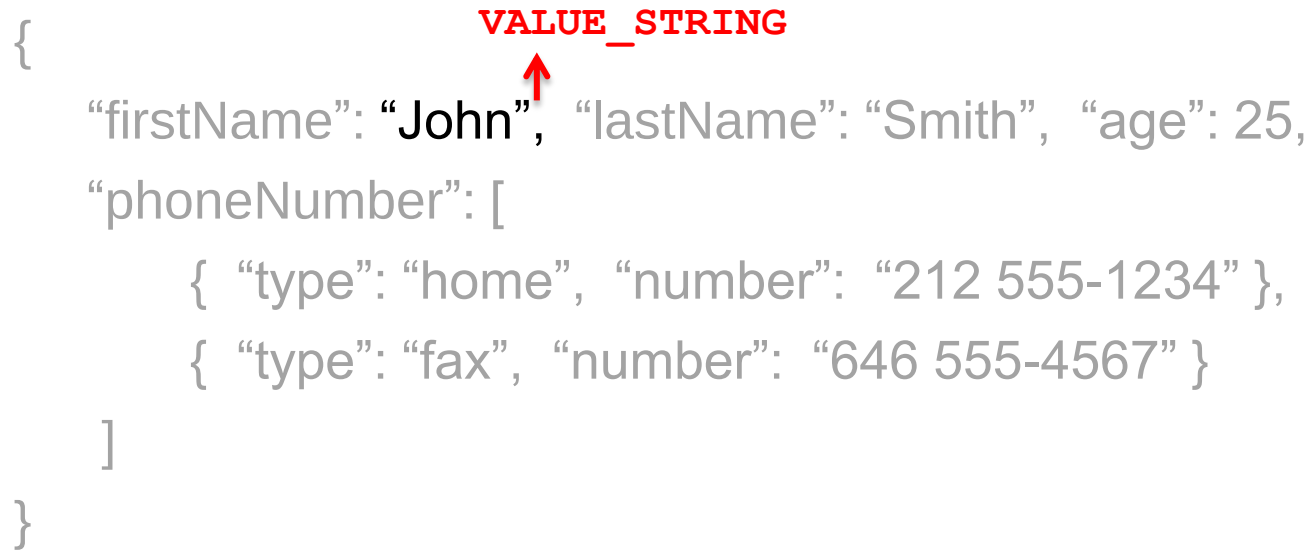
JSON Processing 1.0

Streaming Parser



JSON Processing 1.0

Streaming Parser



JSON Processing 1.0

Streaming Parser

```
{
    “firstName”: “John”,  “lastName”: “Smith”,  “age”: 25,
    “phoneNumber”: [
          {  “type”: “home”,  “number”:  “212 555-1234” },
          {  “type”: “fax”,  “number”:  “646 555-4567” }
     ]
}
```

VALUE_NUMBER

JSON Processing 1.0

Streaming Parser

```
{
    “firstName”: “John”,  “lastName”: “Smith”,  “age”: 25,
    “phoneNumber”: [ ↑START_ARRAY
          {  “type”: “home”,  “number”:  “212 555-1234” },
          {  “type”: “fax”,  “number”:  “646 555-4567” }
     ]
}
```

JSON Processing 1.0

Streaming Parser

```
{
    “firstName”: “John”,  “lastName”: “Smith”,  “age”: 25,
    “phoneNumber”: [
        {  “type”: “home”,  “number”:  “212 555-1234” },
        {  “type”: “fax”,  “number”:  “646 555-4567” }
    ]
}
```

END_ARRAY

JSON Processing 1.0

Using Streaming API to generate JSON

```
JsonGenerator gen = Json.createGenerator…
    .writeStartObject()
      .write("firstName", "John")
      .write("lastName", "Smith")
      .write("age", 25)
      .writeStartArray(“phones”)
         .writeStartObject()
           .write(“type", “home")
           .write(“number”, “222 555-1234”))
         .writeEnd()
         .writeStartObject()   …   .writeEnd()
      .writeEnd()
    .writeEnd();
```

```
{ "firstName": "John",  "lastName": "Smith", "age": 25,
  “phones" : [
     {  “type”:  “home”,  “number”: “ 222 555-1234” },
     {  “type”:  “fax”,  “number”: “ 646 555-4567” }
  ]
}
```

JSON Processing 1.0

Object Model API

- JsonObject/JsonArray – JSON object and array structures
  - JsonString and JsonNumber for string and number values
- JSON builders – build JsonObject and JsonArray
- JsonReader – reads JsonObject and JsonArray
- JsonWriter – writes JsonObject and JsonArray

JSON Processing 1.0

JSON Object-level API

```
JsonObject value = Json.createObjectBuilder()
    .add("firstName", "John")
    .add("lastName", "Smith")
    .add("age", 25)
    .add(“phones", Json.createArrayBuilder()
      .add(Json.createObjectBuilder()
           .add(“type", “home")
           .add(“number”, “222 555-1234”))
      .add(“Json.createObjectBuilder()
           .add(“type", “fax")
           .add(“number”, “646 555-4567”)))
   .build();
```

```
{ "firstName": "John",  "lastName": "Smith",  "age": 25,
   “phones" : [
      {  “type”:  “home”,  “number”: “ 222 555-1234” },
      {  “type”:  “fax”,  “number”: “ 646 555-4567” }
   ]
}
```

Java API for WebSocket 1.0

- Bidirectional full-duplex messaging
  - Over a single TCP connection
- Annotation-based or interface-based programming model
- Server and Client WebSocket Endpoints
  - Annotated: @ServerEndpoint, @ClientEndpoint
  - Programmatic: Endpoint
- Integrated with Java EE web container
- Highly configurable
- Simple packaging and deployment as wars or jars

Java API for WebSocket 1.0

Main API classes

- Endpoint
  - intercepts websocket lifecycle events
- MessageHandler
  - handles incoming messages for endpoint
- Session
  - represents the active conversation
- RemoteEndpoint
  - represents the other end of the conversation

WebSocket Endpoints

Programmatic API

```
public class MyClient extends Endpoint {
    public void onOpen(Session session, EndpointConfig ec) {
        session.addMessageHandler(new MessageHandler.Whole<String>() {
            public void onMessage(String text) {
                System.out.println("Message came from the server : " + message”);
            }
         }
        session.getBasicRemote().sendText("Hello!");
    }

    public void onClose(Session session, CloseReason closeReason) {
        super.onClose(session, closeReason);
    }
}
```

WebSocket Endpoints as POJOs

Annotated client endpoint

```
@ClientEndpoint
public class MyClient {
    @OnOpen public void onOpen(Session session) {
        session.getBasicRemote().sendText(“Hello”);
    }
    @OnMessage public void onMessage(String text, Session session)  {
        System.out.println("Message came from the server : " + message);
    }
}
```

WebSocket Endpoints as POJOs

Annotated server endpoint

```
@ServerEndpoint("/chat")   public class ChatServer {

    static Set<Session> peers = Collections.synchronizedSet(…);

    @OnOpen public void onOpen(Session peer) {
        peers.add(peer);
    }
    @OnMessage public void  onMessage (String message, Session client) {
        for (Session peer : peers) {
            peer.getBasicRemote().sendObject(message);
        }
     }
    @OnClose public void onClose(Session peer) {
        peers.remove(peer);
    }
}
```

Java API for RESTful Web Services (JAX-RS) 2.0

- Client API
- Filters and Interceptors
- Asynchronous Processing
- Hypermedia
- Validation

JAX-RS 1.1

Previous Client API

```
URL url = new URL("http://. . ./atm/balance");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setDoInput(true);
conn.setDoOutput(false);
conn.setRequestMethod("GET");
BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
String line;
while ((line = br.readLine()) != null) {
 //. . .
}
```

JAX-RS 2.0

JAX-RS 2.0 Client API

```
Client client = ClientFactory.newClient();

String name= client.target("http://.../orders/{orderId}/customer")
      .resolveTemplate(“orderId”, “10”)
      .request()
      .get(String.class);
```

JAX-RS 2.0

JAX-RS 2.0 Client API

```
Client client = ClientFactory.newClient();

WebTarget  target = client.target("http://.../orders/{orderId}/customer")
        .register(LoggingFilter.class);

String name =  target.resolveTemplate(“orderId”, “10”)
     .request()
     .get(String.class);
```

JAX-RS 2.0

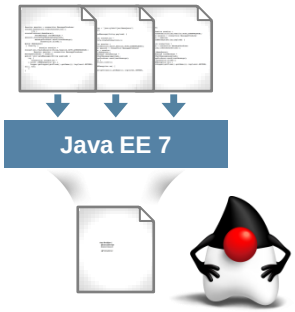
Bean Validation

```
@Path("/")
class MyResourceClass {

    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public void registerUser(
        @NotNull @FormParam("firstName") String firstName,
        @NotNull @FormParam("lastName") String lastName,
        @Email @FormParam("email") String email) {
        ... }
    …
}
```

Alignment and Simplification of Managed Beans

Cohesive, integrated model



- CDI is core component model
- CDI enabled by default
- Expanded use of CDI Interceptors
  - Transactional interceptors
  - Method-level validation interceptors
- New CDI scopes: @TransactionScoped, @FlowScoped

Managed Bean Alignment

Expanded use of CDI

- CDI injection and CDI interceptors apply to all Java EE components and related managed classes when CDI is enabled
- CDI is enabled by default in implicit bean archives
  - Use of CDI bean-defining annotations (@SessionScoped, @Dependent,…) and session beans result in implicit bean archives
    - Library jars, EJB jars, WEB-INF/classes
  - beans.xml not required

Transactional Interceptors

Annotations and semantics defined in JTA 1.2

```
@Inherited
@InterceptorBinding
@Target({TYPE, METHOD}) @Retention(RUNTIME)
public @interface Transactional {
    TxType value() default TxType.REQUIRED;
    Class[] rollbackOn() default {};
    Class[] dontRollbackOn() default {};
}

@Transactional(rollbackOn={SQLException.class},
                        dontRollbackOn={SQLWarning.class})
 public class ShoppingCart {…}
```

Bean Validation 1.1
Method-level Validation

Via CDI Interceptors

```
@Stateless
public class OrderService {
  …
   @ValidOrder
   public Order placeOrder(
      @NotNull String productName,
      @Max(10) int quantity,
      @NotNull String customerName,
      @Address String customerAddress) {
     …
  }
}
```

Interceptor Ordering

Well-defined priority ordering

- Interceptor.Priority.PLATFORM_BEFORE = 0
  - Platform-defined interceptors to be executed at beginning of interceptor chain
  - Transactional interceptors: Interceptor.Priority.PLATFORM_BEFORE+200
- Interceptor.Priority.LIBRARY_BEFORE = 1000
  - Intended for use by extension libraries
- Interceptor.Priority.APPLICATION = 2000
  - Intended for application-defined interceptors
- Interceptor.Priority.LIBRARY_AFTER = 3000
- Interceptor.Priority.PLATFORM_AFTER = 4000
  - Bean Validation defined interceptors: Interceptor.Priority.PLATFORM_AFTER+800

Resource Definition Metadata

- Specifies resources needed by application
  - Enhances configurability in Java EE 7 apps
  - Facilitates provisioning in cloud environments
  - Java EE 6 introduced DataSourceDefinition

```
@DataSourceDefinition (
    name="java:app/jdbc/myDB",
    className="oracle.jdbc.pool.OracleDataSource",
    isolationLevel=TRANSACTION_REPEATABLE_READ,
    initialPoolSize=5)
@Stateless public class MySessionBean {
     @Resource(lookup="java:app/jdbc/myDB")  DataSource my DB;
     …
 }
```

Resource Definition Metadata

- Java EE 7 adds:
  - JMSConnectionFactoryDefinition
  - JMSDestinationDefinition
  - MailSessionDefinition
  - ConnectionFactoryDefinition
  - AdministeredObjectDefinition

Default Resources

Preconfigured resources for use by application

- JDBC/JPA: java:comp/DefaultDataSource
- JMS: java:comp/DefaultJMSConnectionFactory
- Concurrency Utilities:
  - java: comp/DefaultManagedExecutorService
  - java: comp/DefaultManagedScheduledExecutorService
  - java: comp/DefaultManagedThreadFactory
  - java: comp/DefaultManagedContextService

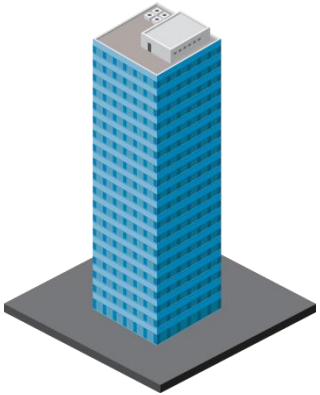
Simplification through Pruning

- Process defined in Java SE 6
  - Platform version N defines feature as “Proposed Optional”
  - Platform version N+1 determines whether to make feature Optional
- Optional APIs as of Java EE 7
  - EJB Entity Beans (CMP, BMP, EJB QL)
  - JAX-RPC
  - JAXR
  - Deployment (JSR 88)

Java Message Service 2.0

New JMS Simplified API targeted at ease of development

- Less code, less boilerplate
- Fewer objects to manage
- Increased developer productivity
- “Classic API” has also been improved



JMS 2.0

Simplifications include….

- New JMSContext interface
- Use of CDI injection; new TransactionScope
- AutoCloseable JMSContext, Connection, Session, …
- Use of runtime exceptions
- Method chaining on JMSProducer
- Simplified message sending

JMS 1.1

Sending a message in the JMS 1.1 “classic” API

```
@Resource(lookup = “java:global/jms/myConnectionFactory”)
ConnectionFactory connectionFactory;

@Resource(lookup = “java:global/jms/myQueue”)
Queue queue;

public void sendMessage(String text) {
  try {
        Connection connection = connectionFactory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(queue);
        TextMessage textMessage = session.createTextMessage(text);
        messageProducer.send(textMessage);
   } finally {
        connection.close();
   }  catch (JMSException ex)  {…}
}
     ..
```


JMS 2.0

Sending a message using Simplified API and JMSContext

```
@Resource(lookup = “java:global/jms/myConnectionFactory”)
ConnectionFactory connectionFactory;

@Resource(lookup = “java:global/jms/myQueue”)
Queue queue;

public void sendMessage(String text) {
  try (JMSContext context =  connectionFactory.createContext();) {
    context.createProducer().send(queue, text);
   } catch (JMSRuntimeException ex) {
     …
   }
}
```

JMS 2.0

Even simpler….

```
@Inject
@JMSConnectionFactory(““java:global/jms/myConnectionFactory”)
JMSContext context;

@Resource(lookup = “java:global/jms/myQueue”)
Queue queue;

public void sendMessage(String text) {
    context.createProducer().send(queue, text);
}
```

JMS 2.0

Even simpler still….

```
@Inject
JMSContext context;

@Resource(lookup = “java:global/jms/myQueue”)
Queue queue;

public void sendMessage(String text) {
    context.createProducer().send(queue, text);
}
```

Concurrency Utilities for Java EE 1.0

Provides asynchronous capabilities to Java EE components

- Extension of Java SE Concurrency Utilities API
- Provides managed objects for submitting tasks and obtaining managed threads
  - ManagedExecutorService
  - ManagedScheduledExecutorService
  - ManagedThreadFactory
  - ContextService

Concurrency Utilities for Java EE 1.0

```
public class AccountTask implements Callable {
  ...
  public AccountInfo call() {
  // task logic
  }
  ...
}

//in calling component:
   @Resource
   ManagedExecutorService mes;
   ...
   Future<AccountInfo> acctFuture = mes.submit(new AccountTask(...));
   AccountInfo accountInfo = acctFuture.get(); // Wait for the results.
   …
   // Process the results
   ...
```

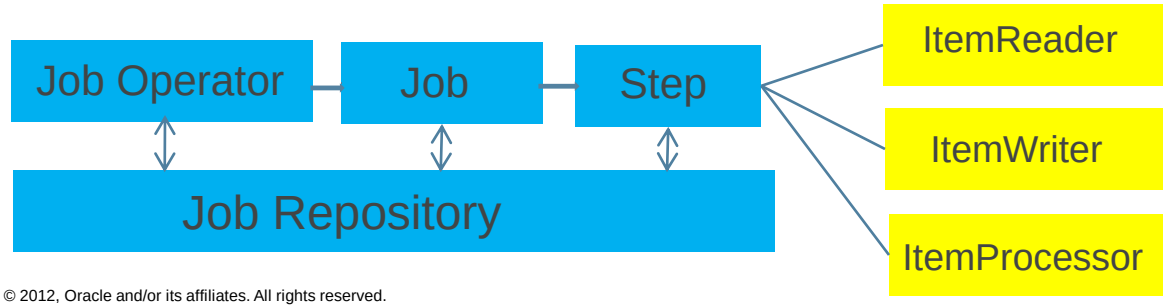
Batch Applications for the Java Platform 1.0

- Designed for non-interactive, bulk-oriented and long-running tasks
- Sequential, parallel, and/or decision-based batch execution
- Processing styles
  - Item-oriented (“chunked”)
  - Task-oriented (“batchlet”)

Batch 1.0

Key concepts

- Job: entire batch process
  - Defined through XML Job Specification Language
- Step: independent, sequential phase of a job
- JobOperator: interface for managing job processing
- JobRepository: information about past and present jobs



Batch 1.0

Job steps

- Chunked step: Item-oriented processing
  - ItemReader/ItemProcessor/ItemWriter pattern
  - Configurable checkpointing and transactions
- Batchlet: Task-oriented processing
  - Roll-your-own batch pattern
  - Runs to completion and exits
- Job can include both types of steps

Batch 1.0

Job specification language

```
<job id="myJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
   <step id="step1" next="step2">
      <chunk item-count="3">
         <reader ref="myItemReader"></reader>
         <processor ref="myItemProcessor"></processor>
         <writer ref="myItemWriter"></writer>
      </chunk>
   </step>
   <step id="step2"  next=“step3”>
      <batchlet ref="myBatchlet"/>
   </step>
 <step id="step3" >
      <chunk item-count="3">
         <reader ref="myOtherItemReader"></reader>
         <processor ref="myOtherItemProcessor"></processor>
         <writer ref="myOtherItemWriter"></writer>
      </chunk>
   </step>
</job>
```

Java Persistence API 2.1

Schema generation

- Generation of database tables, indexes, constraints, etc.
- Designed for flexibility
  - Scenarios: (iterative) prototyping; production; provisioning environments
  - Generate from object/relational metadata (annotations and/or XML)
  - Generate from bundled SQL DDL scripts; also SQL load scripts
  - Generate directly into database
  - Generate into SQL DDL scripts
- Process controlled by metadata or runtime properties

JPA 2.1

Schema generation into scripts

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
   <persistence-unit name=“samplePU" transaction-type="JTA">
      <jta-data-source>jdbc/mypu</jta-data-source>
      <properties>
         <property name="javax.persistence.schema-generation.database.action" value="none" />
         <property name="javax.persistence.schema-generation.scripts.action" value=“drop-and-create"/>
         <property name="javax.persistence.schema-generation.scripts.create-target" value="file:///c:/temp/create.sql"/>
         <property name="javax.persistence.schema-generation.scripts.drop-target" value="file:///c:/temp/drop.sql"/>
      </properties>
  </persistence-unit>
</persistence>
```

JPA 2.1

Schema generation from scripts

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
   <persistence-unit name="samplePU" transaction-type="JTA">
      <jta-data-source>jdbc/mypu</jta-data-source>
      <properties>
         <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
         <property name="javax.persistence.schema-generation.create-source" value="script"/>
         <property name="javax.persistence.schema-generation.drop-source" value="script"/>
         <property name="javax.persistence.schema-generation.create-script-source" value="META-INF/create.sql"/>
         <property name="javax.persistence.schema-generation.drop-script-source" value="META-INF/drop.sql"/>
         <property name="javax.persistence.sql-load-script-source" value="META-INF/load.sql"/>
      </properties>
   </persistence-unit>
</persistence>
```

Java EE 7 Summary

- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

Transparency in JSR processes

All Java EE JSRs run with high level of transparency

- java.net used for all Oracle-led JSRs
  - http://java.net/projects/javaee-spec/pages/Home
- Publicly viewable Expert Group mailing archives
- Users observer lists get copies of all Expert Group emails
- Public download areas, JIRAs
- Wikis, source repositories, etc. at the group discretion
- Commitment to JCP 2.8/2.9 processes

Meet the Java EE Specleads

Tomorrow afternoon

BOF 2795 “Meet the Java EE Specification Leads”

Tuesday, 4:30-5:30

Parc 55 – Cyril Magnin I

DOWNLOAD

Java EE 7 SDK

oracle.com/javaee

GlassFish 4.0

Full Platform or Web Profile

glassfish.org

JavaOne ORACLE

MAKE THE FUTURE JAVA

JavaOne™

ORACLE®


JavaOne™