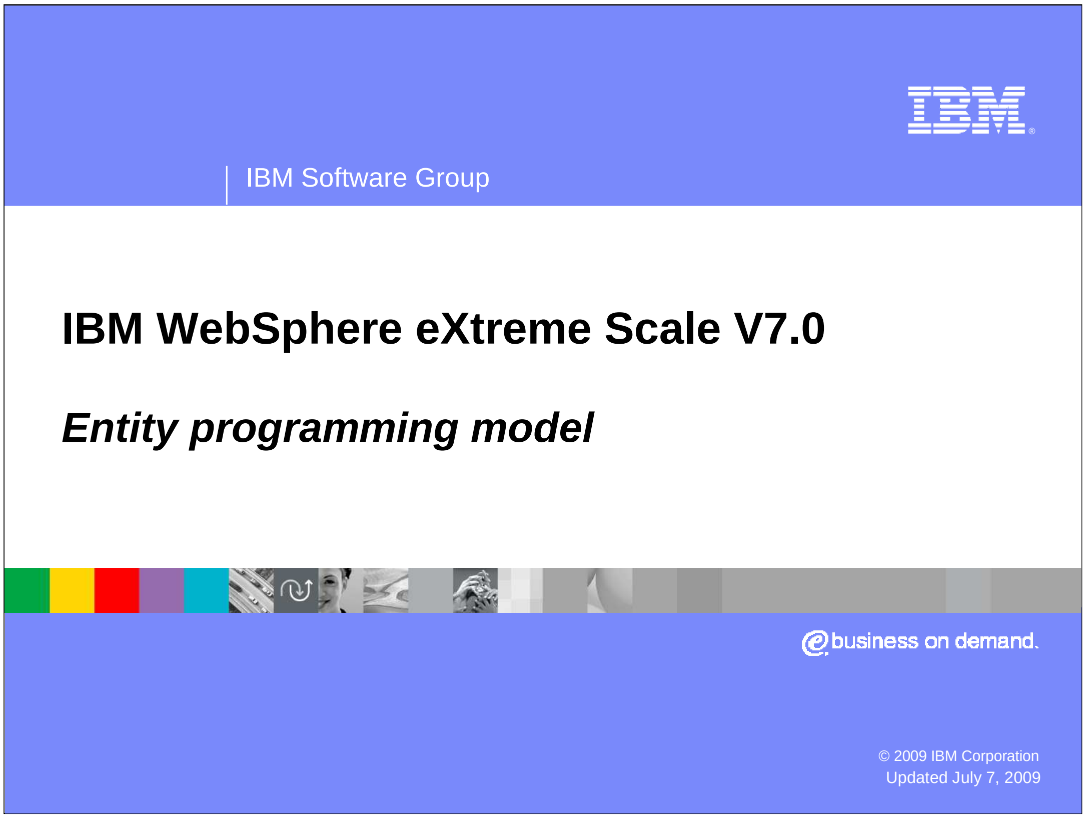


IBM Software Group

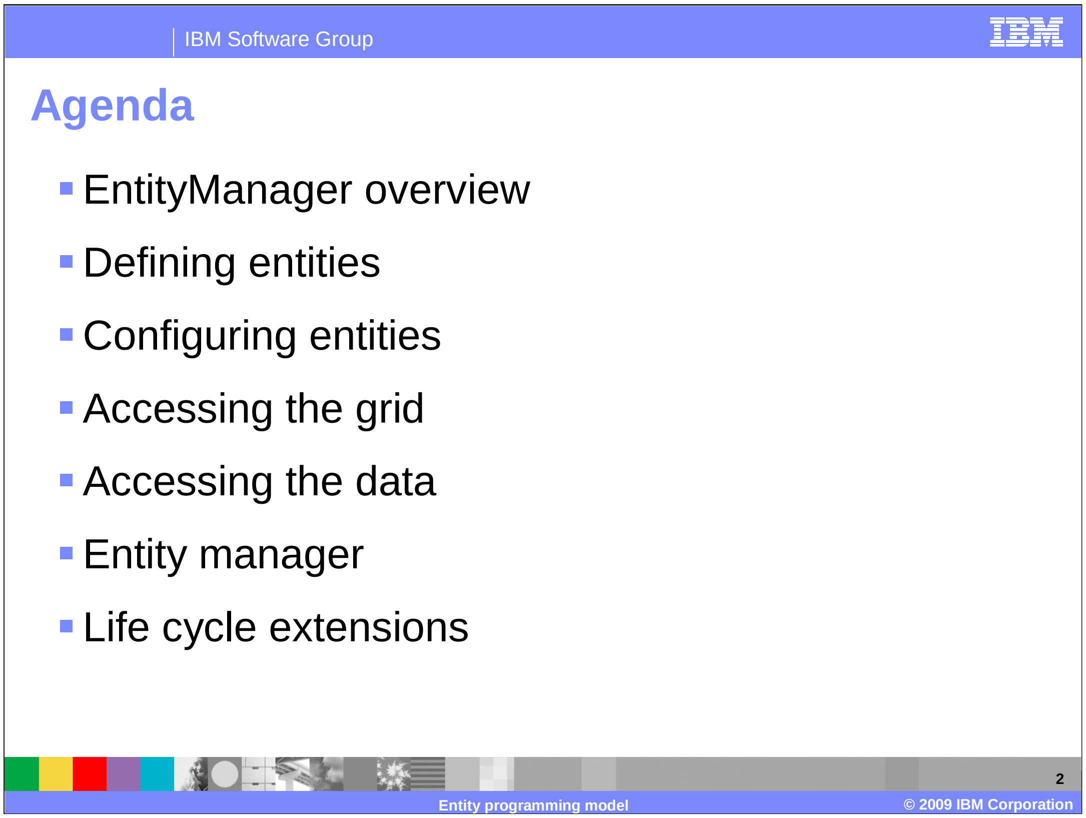
IBM WebSphere eXtreme Scale V7.0

Entity programming model

© 2009 IBM Corporation

Updated July 7, 2009

WebSphere® eXtreme Scale provides an entity-based interface for managing data in the grid as an alternative to the Map-based API.

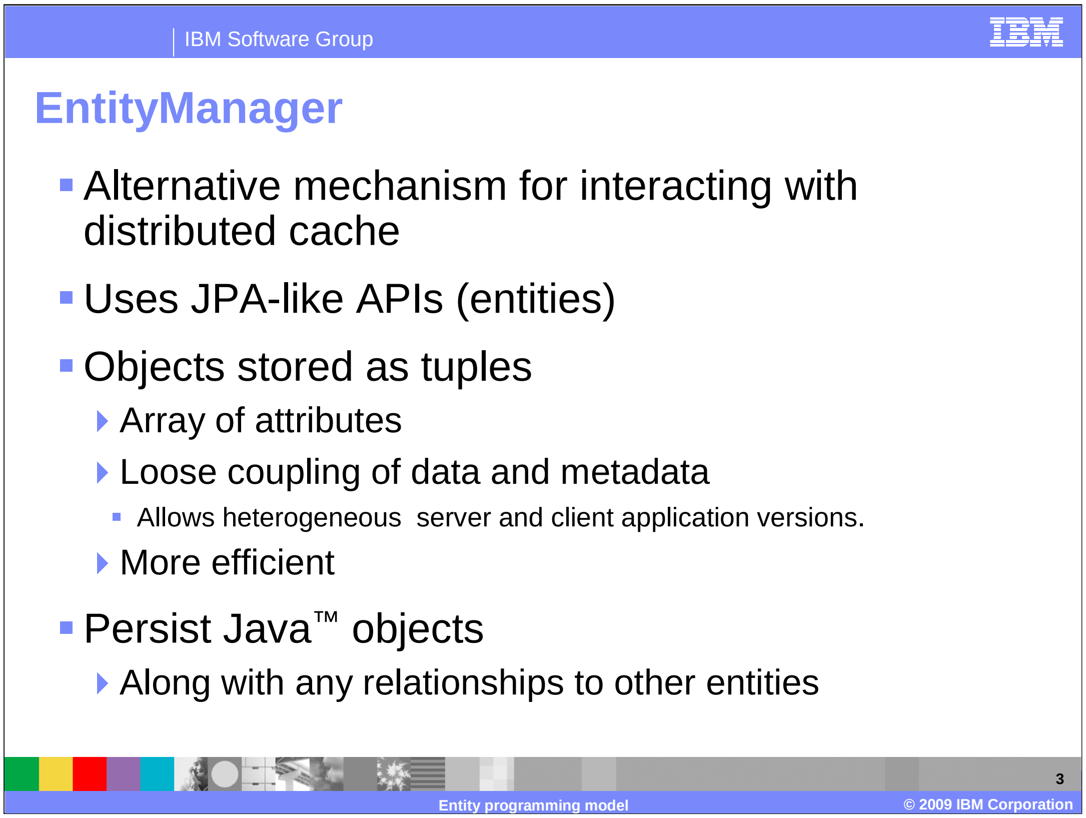


Agenda

- EntityManager overview
- Defining entities
- Configuring entities
- Accessing the grid
- Accessing the data
- Entity manager
- Life cycle extensions

© 2009 IBM Corporation

This presentation will introduce entities, and how to create and configure them for eXtreme Scale. It will then cover how a client program can interact with the eXtreme Scale entity manager. Finally it will provide an overview of extension points that allow an application to be notified when significant life cycle events occur on an entity.



IBM Software Group

EntityManager

- Alternative mechanism for interacting with distributed cache
- Uses JPA-like APIs (entities)
- Objects stored as tuples
  - Array of attributes
  - Loose coupling of data and metadata
    - Allows heterogeneous server and client application versions.
  - More efficient
- Persist Java™ objects
  - Along with any relationships to other entities

3

Entity programming model

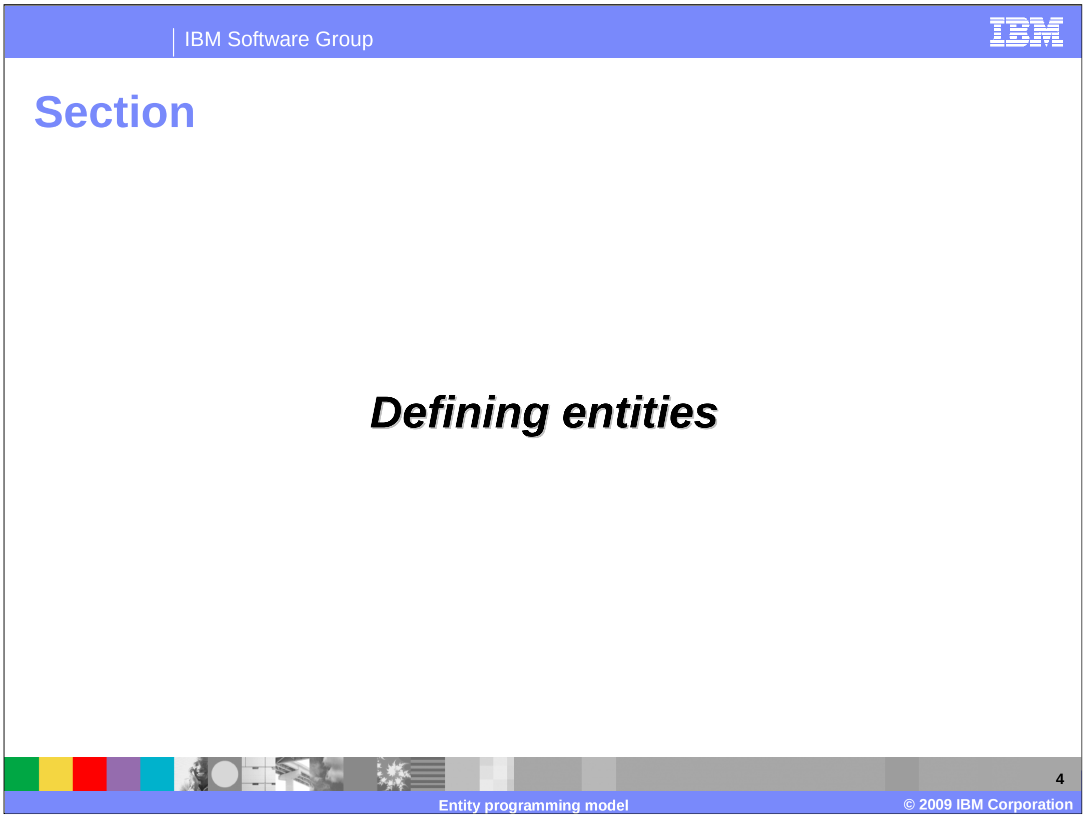
© 2009 IBM Corporation

WebSphere eXtreme Scale provides two ways to store data; a map based interface that resembles a Java map, and an Entity based API. The EntityManager API is similar to other Java object persistence technologies such as Java Persistence API (JPA) and Hibernate, in that it synchronizes a graph of managed Java object instances with the persistent store. In this case, the persistent store is an eXtreme Scale grid.

Objects stored using the EntityManager are converted to entities. As with the map-based interface, the EntityManager inteface uses a key and a value. But unlike the map-based interface (which stores both key and value as Java Objects) the EntityManager interface stores the key and value as tuples. A tuple is essentially an array of primitive attributes. The EntityManager stores metadata describing attribute names and types for the entities it manages. Since data is not stored as objects, entities provide more flexibility in client interaction – clients at different maintenance levels can access objects without common object version problems.

Entities are also more efficient than maps. Objects are expensive from a performance point of view. Object serialization is very inefficient without custom application coding, copying an object is expensive, and reflecting arbitrary attributes from an object is expensive.

Entities can have relationships to other entities, described using metadata. The EntityManager maintains these relationships automatically. When an entity is persisted, the entire object graph is stored in the grid. Similarly, when an entity is retrieved from the map, all related entities are also retrieved.

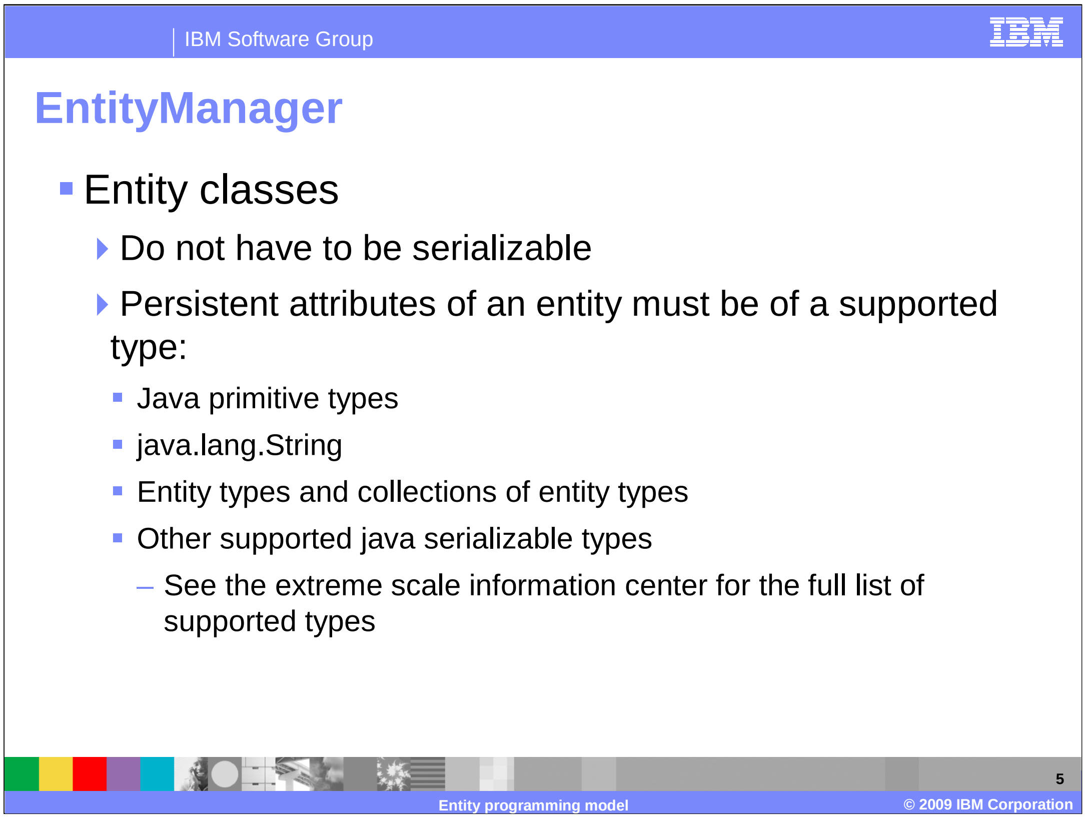


Section

Defining entities

© 2009 IBM Corporation

This section will describe how to define entities.

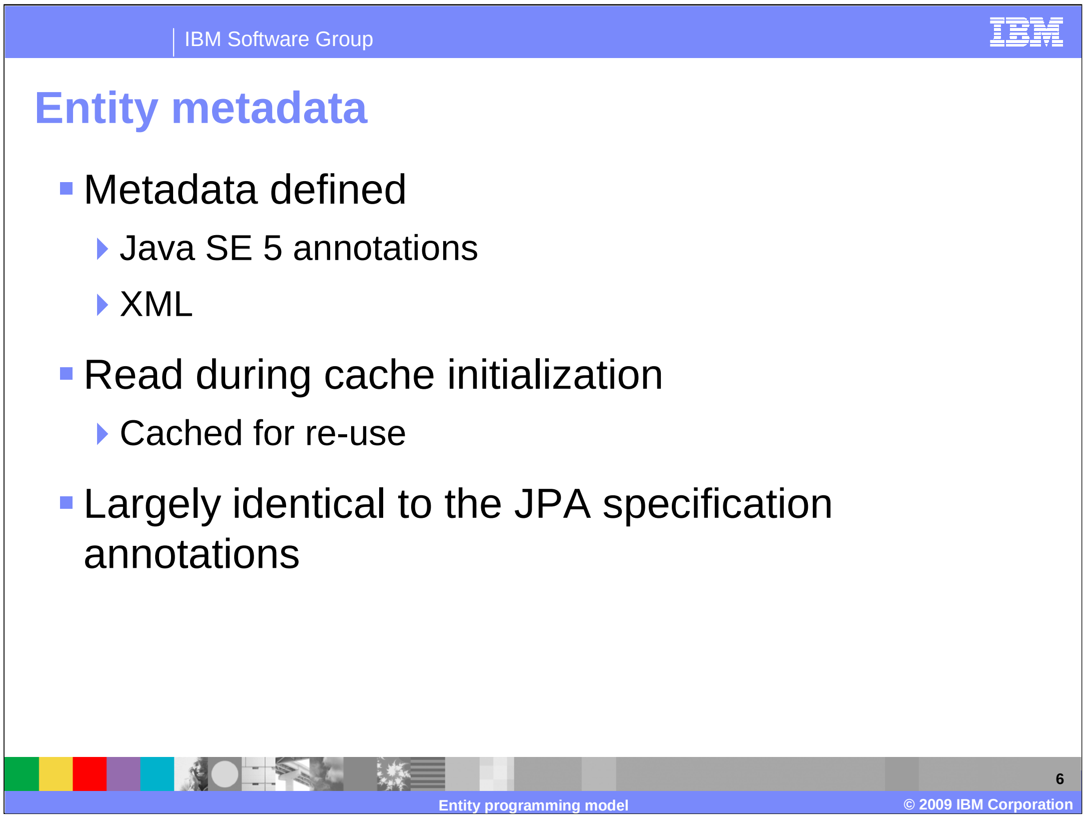


EntityManager

- Entity classes
  - Do not have to be serializable
  - Persistent attributes of an entity must be of a supported type:
    - Java primitive types
    - java.lang.String
    - Entity types and collections of entity types
    - Other supported java serializable types
      - See the extreme scale information center for the full list of supported types

© 2009 IBM Corporation

Unlike objects stored in an ObjectMap, entities do not have to be serializable. Persistent attributes, however, must be Java primitives, supported serializable types, entities, or collections of entities. The WebSphere eXtreme Scale information center lists the full set of supported attribute types.

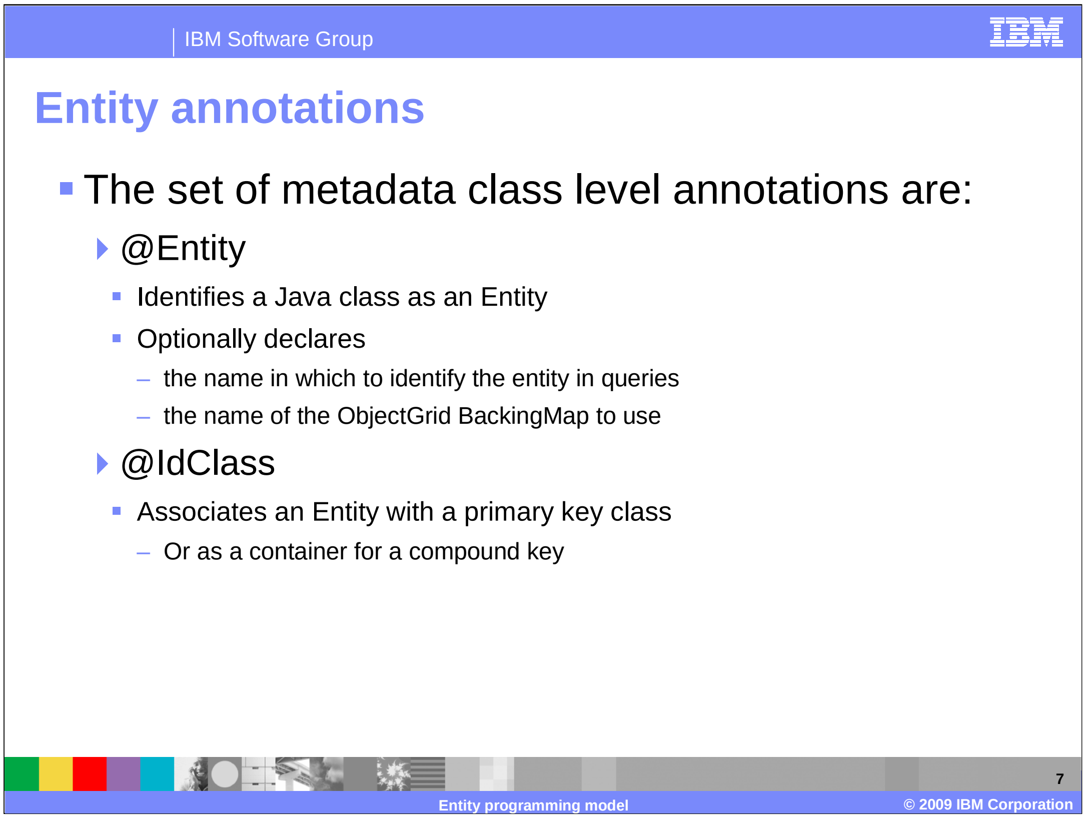


Entity metadata

- Metadata defined
  - Java SE 5 annotations
  - XML
- Read during cache initialization
  - Cached for re-use
- Largely identical to the JPA specification annotations

© 2009 IBM Corporation

The metadata for an entity is automatically discovered using either an entity descriptor XML file or annotated Java classes. Entity descriptor XML definition and supported annotations are largely identical to the JPA specification annotations.



IBM Software Group

Entity annotations

- The set of metadata class level annotations are:
  - @Entity
    - Identifies a Java class as an Entity
    - Optionally declares
      - the name in which to identify the entity in queries
      - the name of the ObjectGrid BackingMap to use
  - @IdClass
    - Associates an Entity with a primary key class
      - Or as a container for a compound key

7

Entity programming model

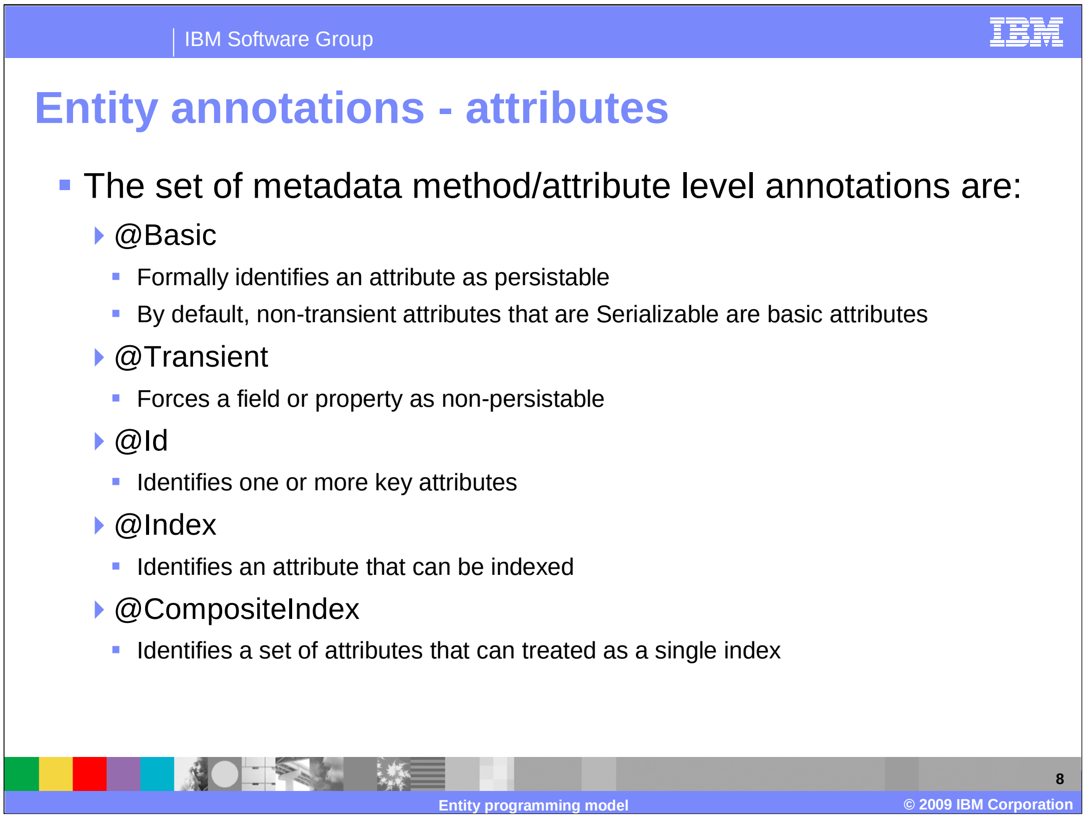
© 2009 IBM Corporation

Entities are identified by associating various metadata with a Java class. The easiest way to specify this metadata is by Java 5 annotations.

Entity classes must include the @Entity annotation. An entity must have a Public or Protected no-argument constructor and must be a top-level class.

All entities in a grid must have a unique name and type. If using annotations, the default entity name is the simple (short) name of the class. This can be overridden using the “name” attribute of the @Entity annotation.

The @IdClass annotation can be used to specify a composite primary key class that is mapped to multiple attributes of an entity. The names of the attributes in the primary key class, and the primary key of the entity, must correspond and their types must be the same. The @Id annotation must also be applied to the corresponding attributes of the entity.



IBM Software Group

IBM

Entity annotations - attributes

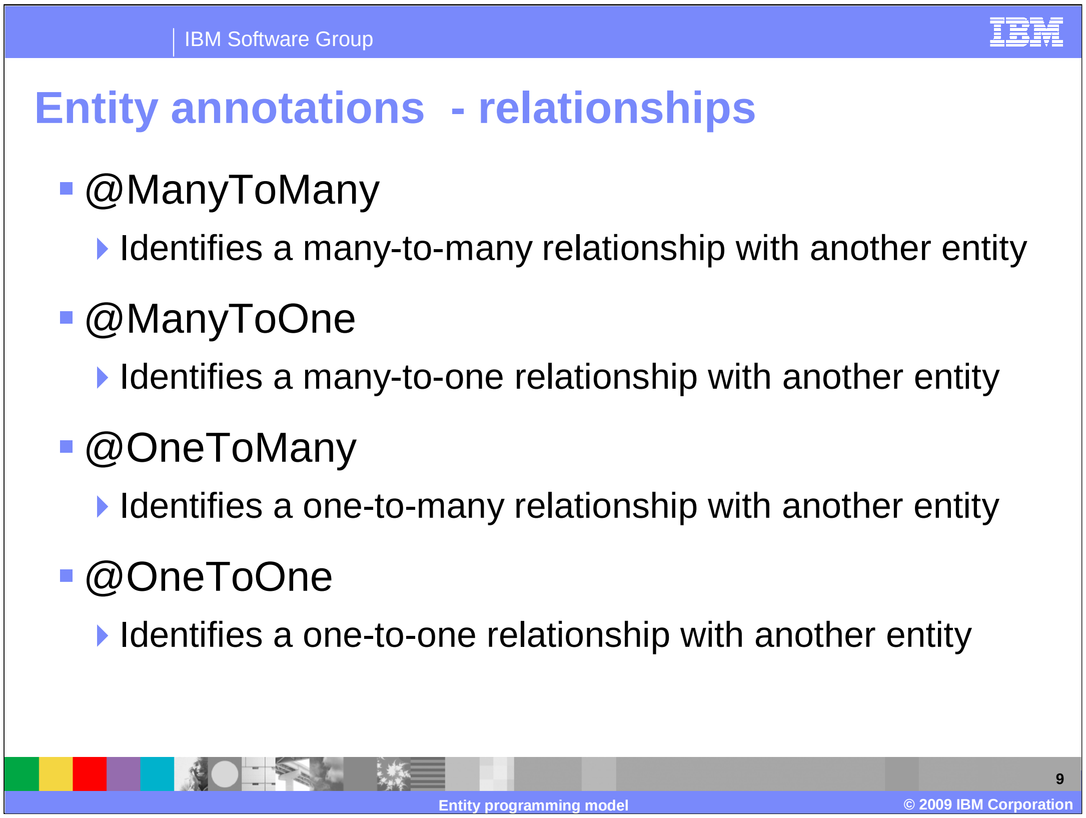
- The set of metadata method/attribute level annotations are:
  - @Basic
    - Formally identifies an attribute as persistable
    - By default, non-transient attributes that are Serializable are basic attributes
  - @Transient
    - Forces a field or property as non-persistable
  - @Id
    - Identifies one or more key attributes
  - @Index
    - Identifies an attribute that can be indexed
  - @CompositeIndex
    - Identifies a set of attributes that can treated as a single index

8

Entity programming model

© 2009 IBM Corporation

The EntityManager provides annotations to identify basic persistable attributes and non-persistable (or transient) attributes. The @Id annotation specifies the attribute (or attributes) that uniquely identify the entity in the grid. You can also create indexes on single attributes or collections of attributes. Index attributes are implicitly persistable. Indexes allow the application to find objects by a specific value or a range of values.

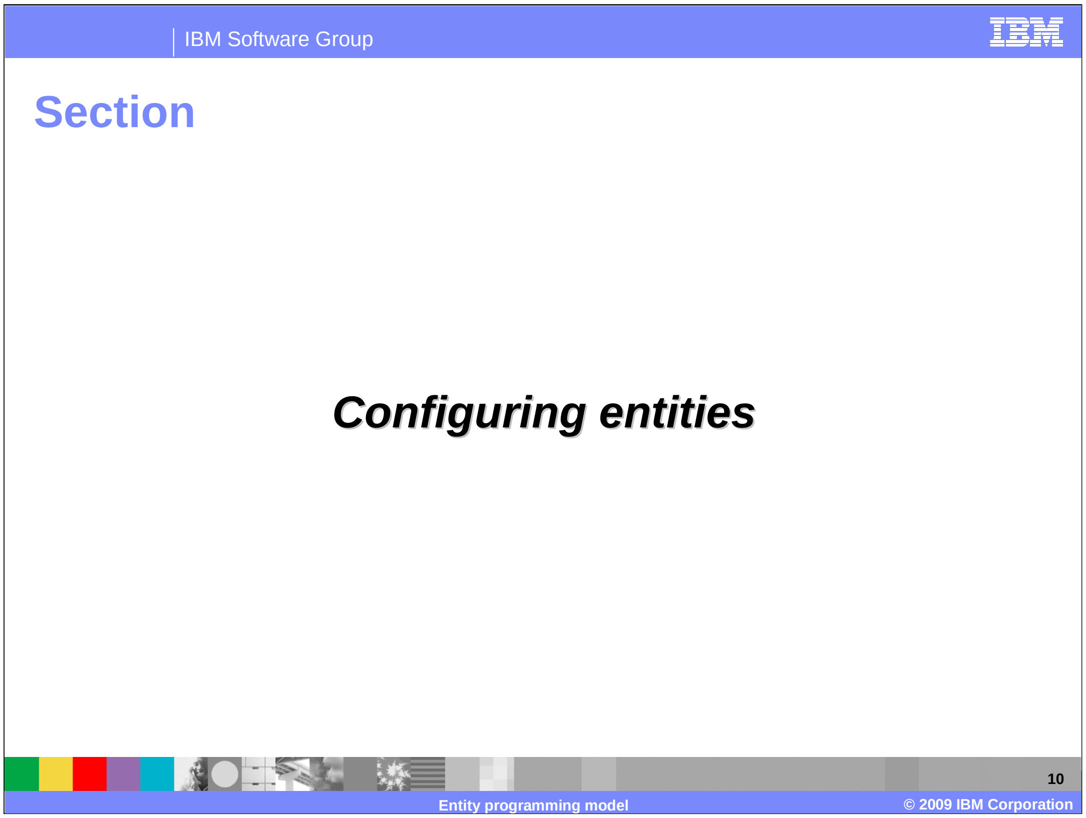


Entity annotations - relationships

- @ManyToMany
  - Identifies a many-to-many relationship with another entity
- @ManyToOne
  - Identifies a many-to-one relationship with another entity
- @OneToMany
  - Identifies a one-to-many relationship with another entity
- @OneToOne
  - Identifies a one-to-one relationship with another entity

© 2009 IBM Corporation

Entities can have relationships to other entities. The eXtreme Scale EntityManager supports one-to-one, many-to-one, one-to-many and many-to-many relationships. When an object is persisted, the EntityManager will recursively follow the defined relationships and persist the related entities.

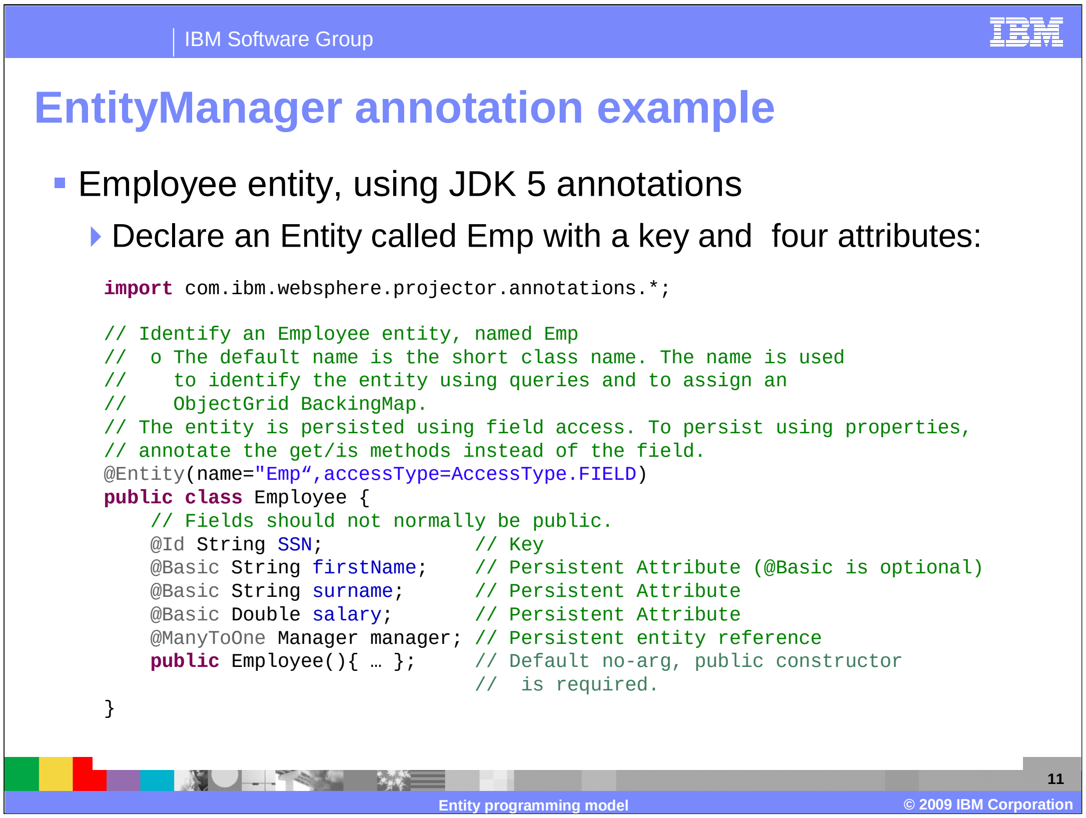


Section

Configuring entities

© 2009 IBM Corporation

This section will present an example entity defined using annotations, and the same entity configured through an entity descriptor XML file.



EntityManager annotation example

- Employee entity, using JDK 5 annotations
  - Declare an Entity called Emp with a key and four attributes:

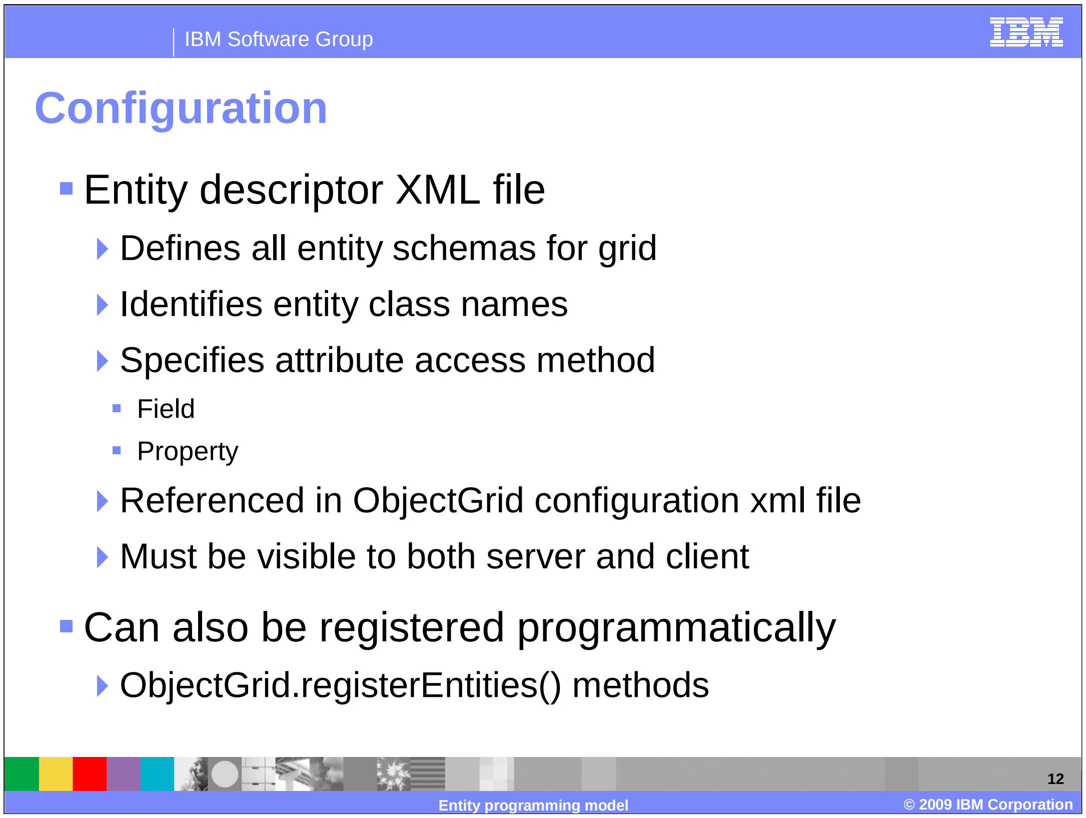
```
import com.ibm.websphere.projector.annotations.*;

// Identify an Employee entity, named Emp
//  o The default name is the short class name. The name is used
//    to identify the entity using queries and to assign an
//    ObjectGrid BackingMap.
// The entity is persisted using field access. To persist using properties,
// annotate the get/is methods instead of the field.
@Entity(name="Emp“,accessType=AccessType.FIELD)
public class Employee {
    // Fields should not normally be public.
    @Id String SSN;             // Key
    @Basic String firstName;    // Persistent Attribute (@Basic is optional)
    @Basic String surname;      // Persistent Attribute
    @Basic Double salary;       // Persistent Attribute
    @ManyToOne Manager manager; // Persistent entity reference
    public Employee(){ … };     // Default no-arg, public constructor
                                //  is required.
}
```

This example shows an employee-type entity defined by Java annotations. Note the @Entity annotation before the “class” definition. This entity includes social security number as a unique identifier, several persistable fields, and a many-to-one relationship to a manager-type entity indicating many employees to one manager. This example uses field-level access.

An entity can specify property-level access instead of field-level access. Property-access entities must adhere to the JavaBeans signature conventions for read and write properties. Methods that do not follow JavaBeans conventions, or have the Transient annotation on the getter method, are ignored. For a property of type T, there must be a getter method: T getProperty() and a setter method: void setProperty(T). For Boolean types, the getter method can be expressed as Boolean isProperty(). Persistent properties must not have the static modifier.

The import of com.ibm.websphere.projector.annotations.* is critical to allow the compiler to process the annotations.

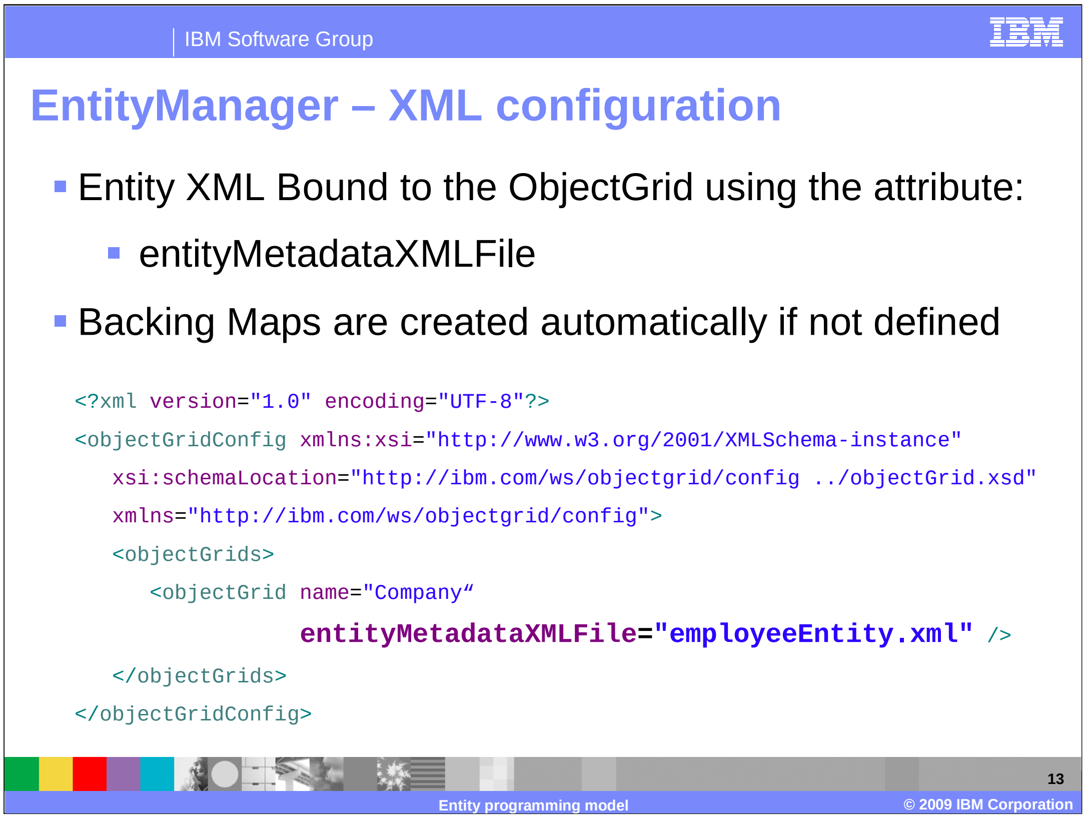


Configuration

- Entity descriptor XML file
  - Defines all entity schemas for grid
  - Identifies entity class names
  - Specifies attribute access method
    - Field
    - Property
  - Referenced in ObjectGrid configuration xml file
  - Must be visible to both server and client
- Can also be registered programmatically
  - ObjectGrid.registerEntities() methods

© 2009 IBM Corporation

Each eXtreme Scale grid that contains entities must have an entity descriptor XML file which tells the grid which objects (and associated maps) should be treated as entities. An entity schema is a set of entities and the relationships between them.



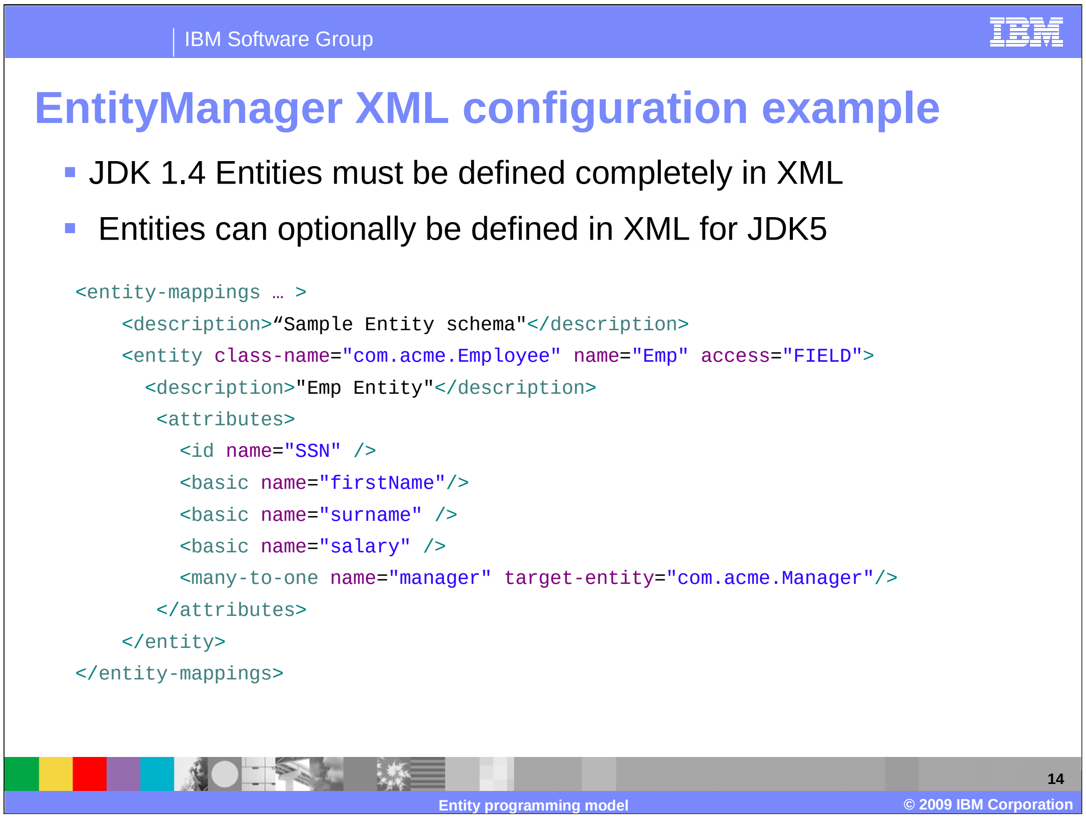
EntityManager – XML configuration

- Entity XML Bound to the ObjectGrid using the attribute:
  - entityMetadataXMLFile
- Backing Maps are created automatically if not defined

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
   xmlns="http://ibm.com/ws/objectgrid/config">
   <objectGrids>
      <objectGrid name="Company“
                  entityMetadataXMLFile="employeeEntity.xml" />
   </objectGrids>
</objectGridConfig>
```

© 2009 IBM Corporation

The entity descriptor XML file must be referenced in the ObjectGrid definition XML file. The ObjectGrid definition file can include explicit definitions for the entity maps. If not, the maps are created automatically based on the entity metadata.



EntityManager XML configuration example

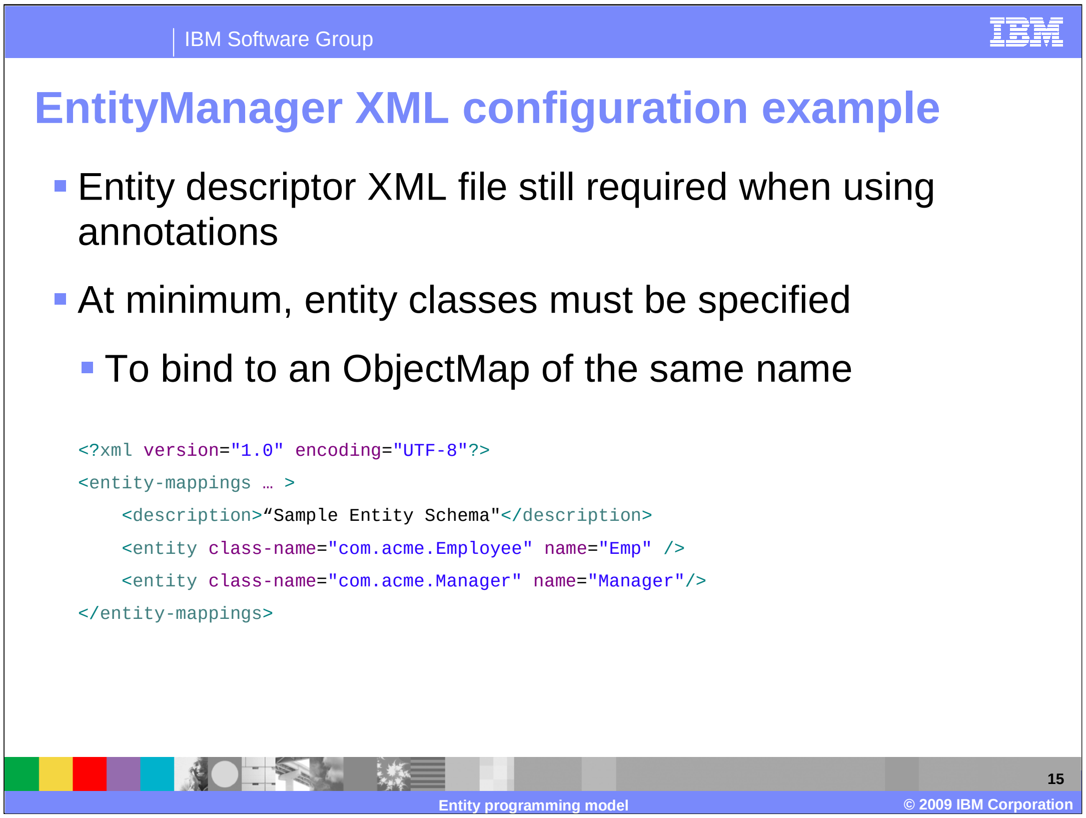
- JDK 1.4 Entities must be defined completely in XML
- Entities can optionally be defined in XML for JDK5

```
<entity-mappings … >
    <description>“Sample Entity schema"</description>
    <entity class-name="com.acme.Employee" name="Emp" access="FIELD">
      <description>"Emp Entity"</description>
       <attributes>
         <id name="SSN" />
         <basic name="firstName"/>
         <basic name="surname" />
         <basic name="salary" />
         <many-to-one name="manager" target-entity="com.acme.Manager"/>
       </attributes>
    </entity>
</entity-mappings>
```

© 2009 IBM Corporation

Java 1.4 does not support annotations. If you are using this version of Java you must specify the entity metadata in the entity descriptor XML file.

This is an example of an entity descriptor XML file for the employee-type entity. Note the XML entries correspond to the annotations in the earlier example.



IBM Software Group

EntityManager XML configuration example

- Entity descriptor XML file still required when using annotations
- At minimum, entity classes must be specified
  - To bind to an ObjectMap of the same name

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings … >
    <description>“Sample Entity Schema"</description>
    <entity class-name="com.acme.Employee" name="Emp" />
    <entity class-name="com.acme.Manager" name="Manager"/>
</entity-mappings>
```

© 2009 IBM Corporation

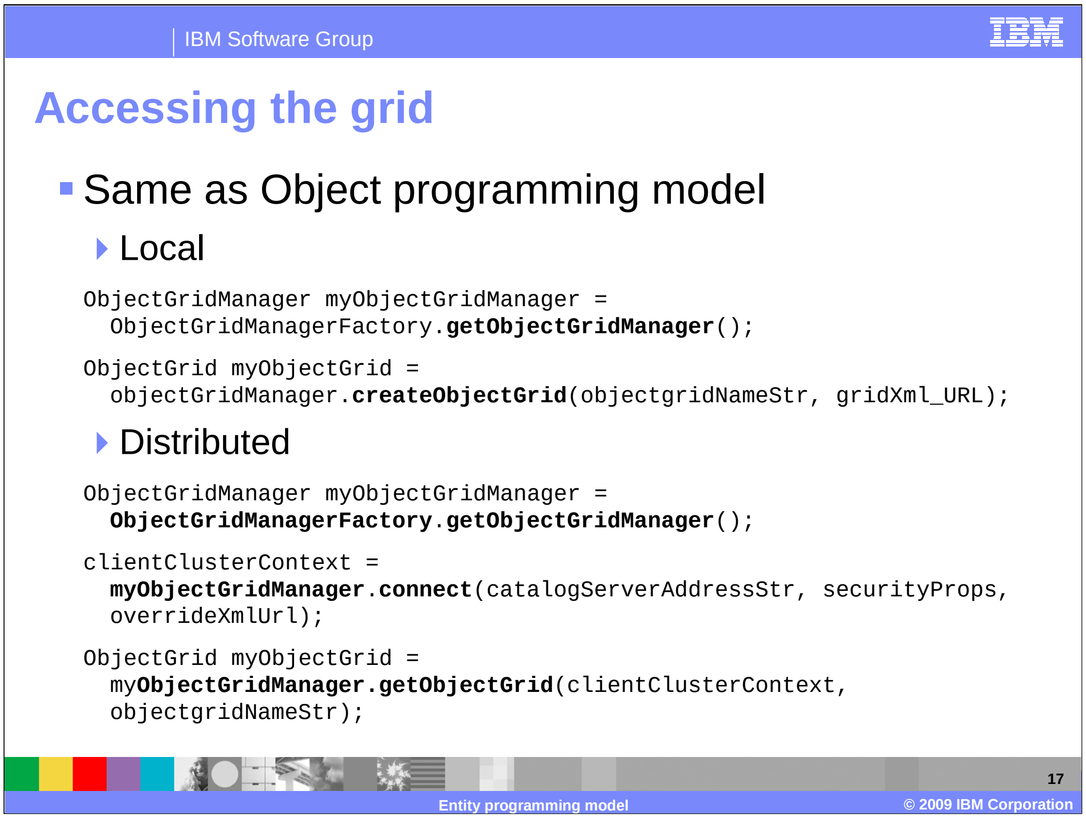
An entity descriptor XML file is required even if using annotations. A minimal entity descriptor XML file lists the entity classes. These classes must include the @Entity annotation.

Section

Accessing the grid

© 2009 IBM Corporation

Once entities have been defined and the grid configured, applications can access the entities through the grid.



Accessing the grid

- Same as Object programming model
  - Local

```
ObjectGridManager myObjectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid myObjectGrid =
  objectGridManager.createObjectGrid(objectgridNameStr, gridXml_URL);
```

  - Distributed

```
ObjectGridManager myObjectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();

clientClusterContext =
  myObjectGridManager.connect(catalogServerAddressStr, securityProps,
  overrideXmlUrl);

ObjectGrid myObjectGrid =
  myObjectGridManager.getObjectGrid(clientClusterContext,
  objectgridNameStr);
```

© 2009 IBM Corporation

First, an application must create or obtain a reference to the grid. This is described in more detail in the core programming concepts presentation.

IBM Software Group

Section

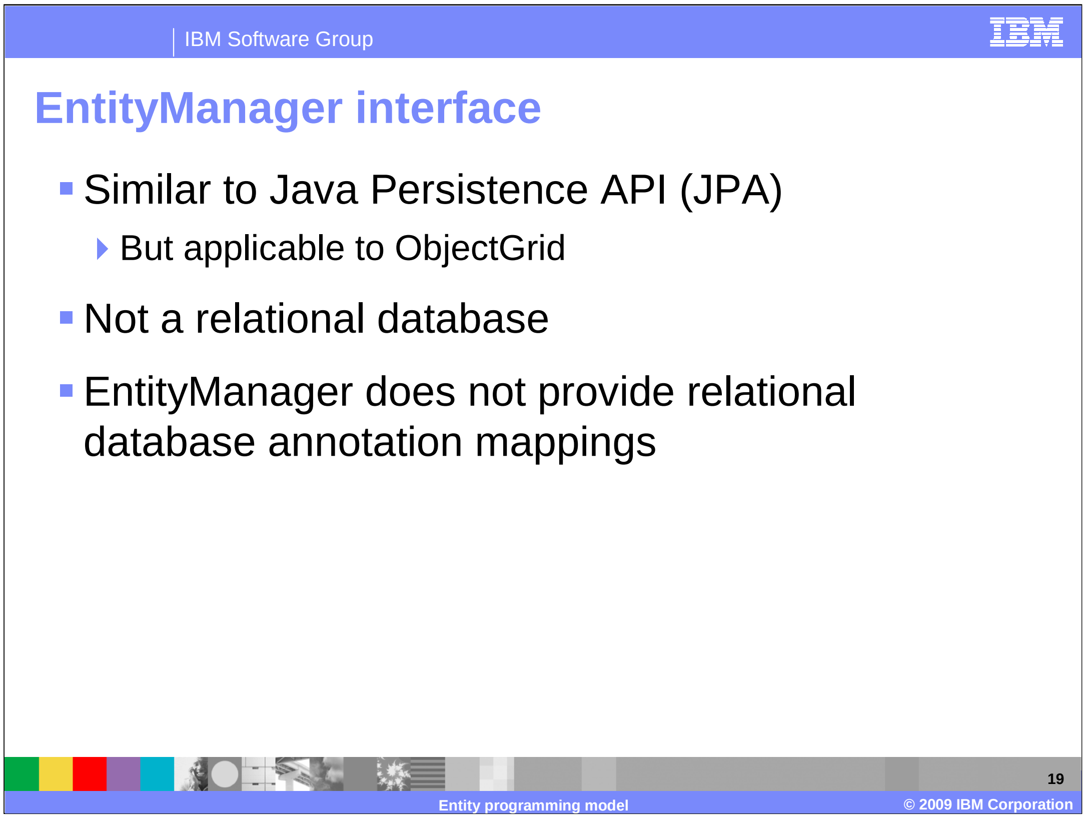
Accessing the data

18

Entity programming model

© 2009 IBM Corporation

Once a program has a reference to the grid it can begin retrieving entities from and adding entities to the grid.

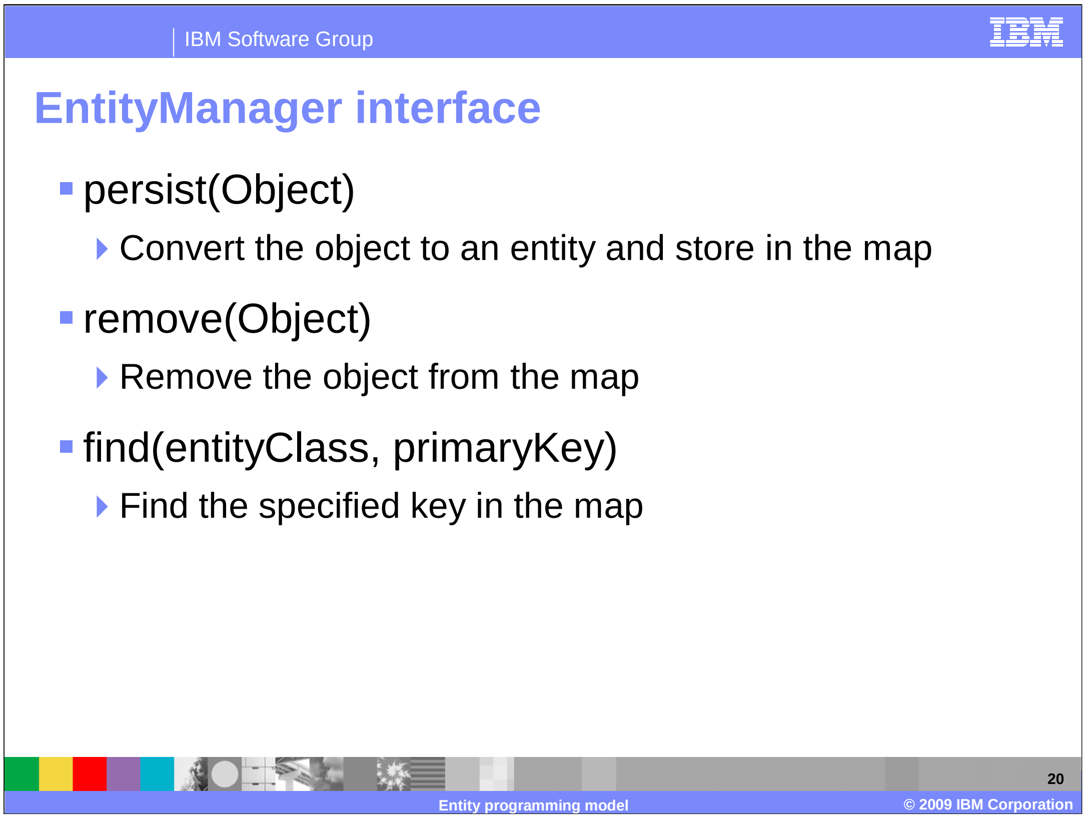


EntityManager interface

- Similar to Java Persistence API (JPA)
  - But applicable to ObjectGrid
- Not a relational database
- EntityManager does not provide relational database annotation mappings

© 2009 IBM Corporation

This is accomplished through the EntityManager interface. The eXtreme Scale EntityManager interface is similar to the Java Persistence API. EntityManager is an object store, not a relational database. WebSphere eXtreme Scale does not provide object-relational mapping tools or relational database mapping annotation.



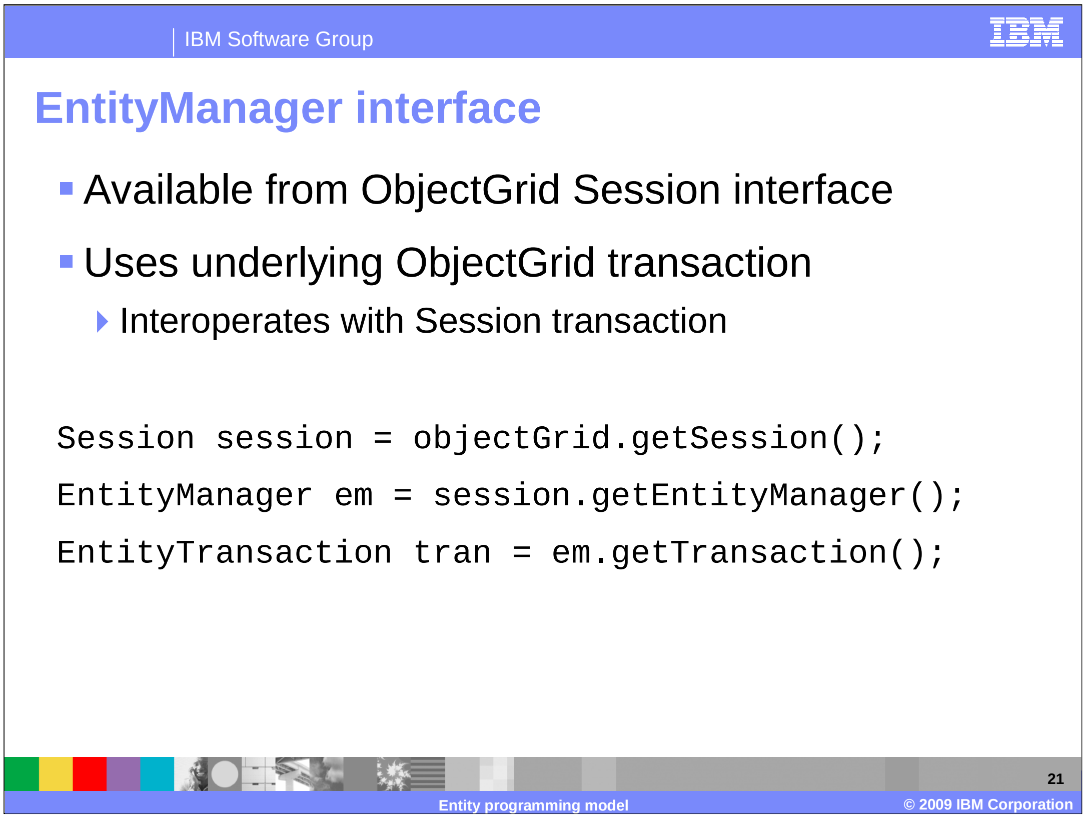
EntityManager interface

- persist(Object)
  - Convert the object to an entity and store in the map
- remove(Object)
  - Remove the object from the map
- find(entityClass, primaryKey)
  - Find the specified key in the map

© 2009 IBM Corporation

The EntityManager provides “find” and “persist” methods that correspond to ObjectMap “get” and “put.”

The “find” method requires a class parameter, which the EntityManager maps to a BackingMap through the entity descriptor XML file.


IBM Software Group

EntityManager interface

- Available from ObjectGrid Session interface
- Uses underlying ObjectGrid transaction
  - Interoperates with Session transaction

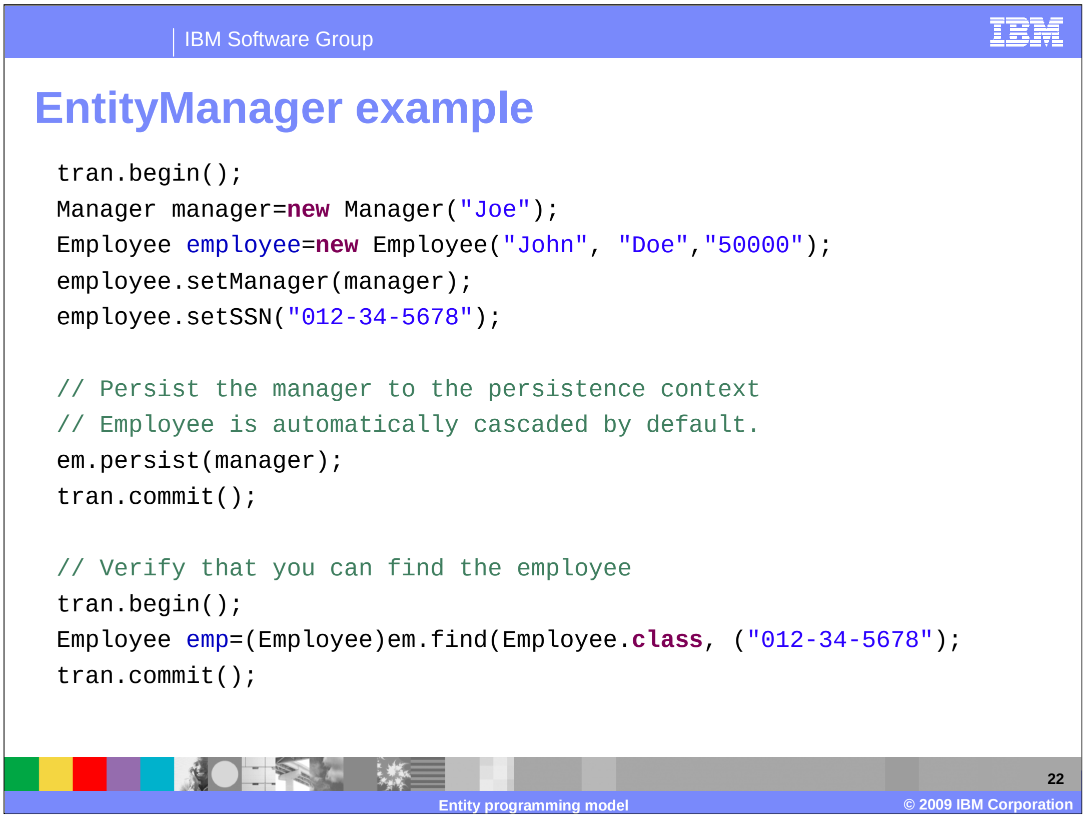
```
Session session = objectGrid.getSession();
EntityManager em = session.getEntityManager();
EntityTransaction tran = em.getTransaction();
```

21

Entity programming model

© 2009 IBM Corporation

The application accesses the EntityManager through the ObjectGrid session. Once the application has an EntityManger, it can retrieve an entity transaction, and begin operating on entities in the map.



EntityManager example

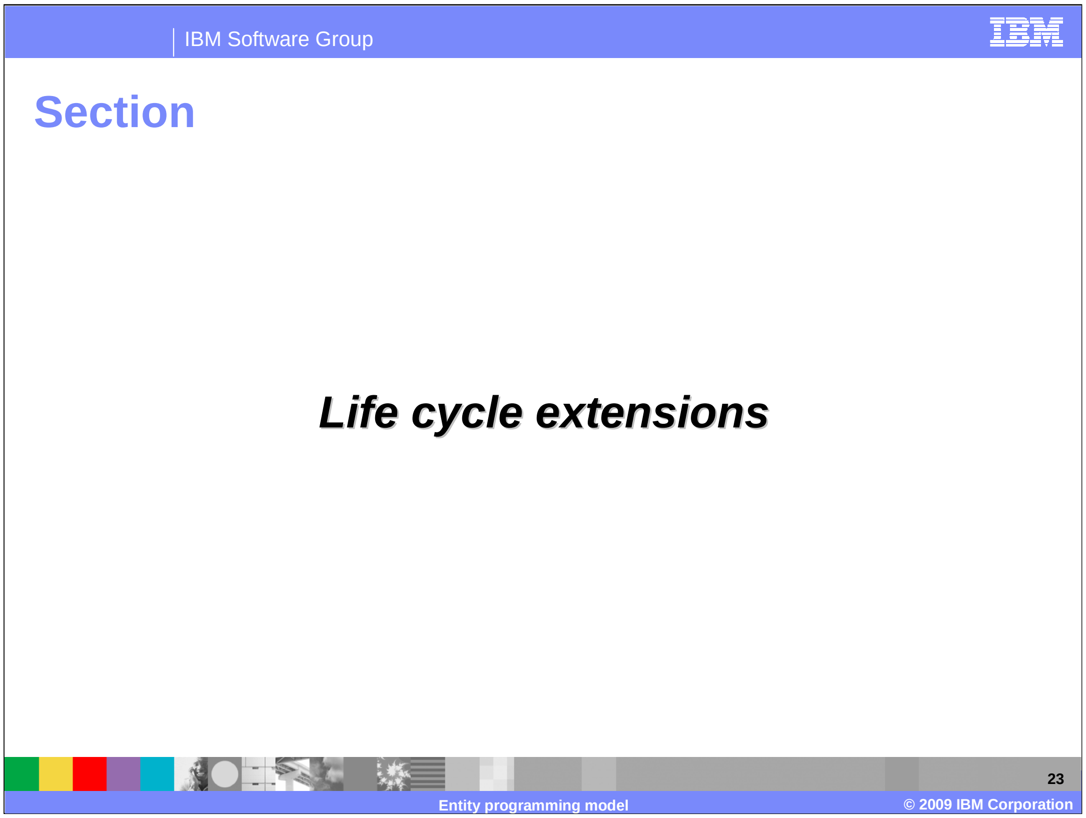
```
tran.begin();
Manager manager=new Manager("Joe");
Employee employee=new Employee("John", "Doe","50000");
employee.setManager(manager);
employee.setSSN("012-34-5678");

// Persist the manager to the persistence context
// Employee is automatically cascaded by default.
em.persist(manager);
tran.commit();

// Verify that you can find the employee
tran.begin();
Employee emp=(Employee)em.find(Employee.class, ("012-34-5678");
tran.commit();
```

© 2009 IBM Corporation

This example creates an employee and that employee’s manager. The employee is the “root” object. When the application persists the employee, the EntityManager will implicitly persist the manager also. Employee and Manager entities are persisted in separate maps, but the relationship is maintained.



IBM Software Group

Section

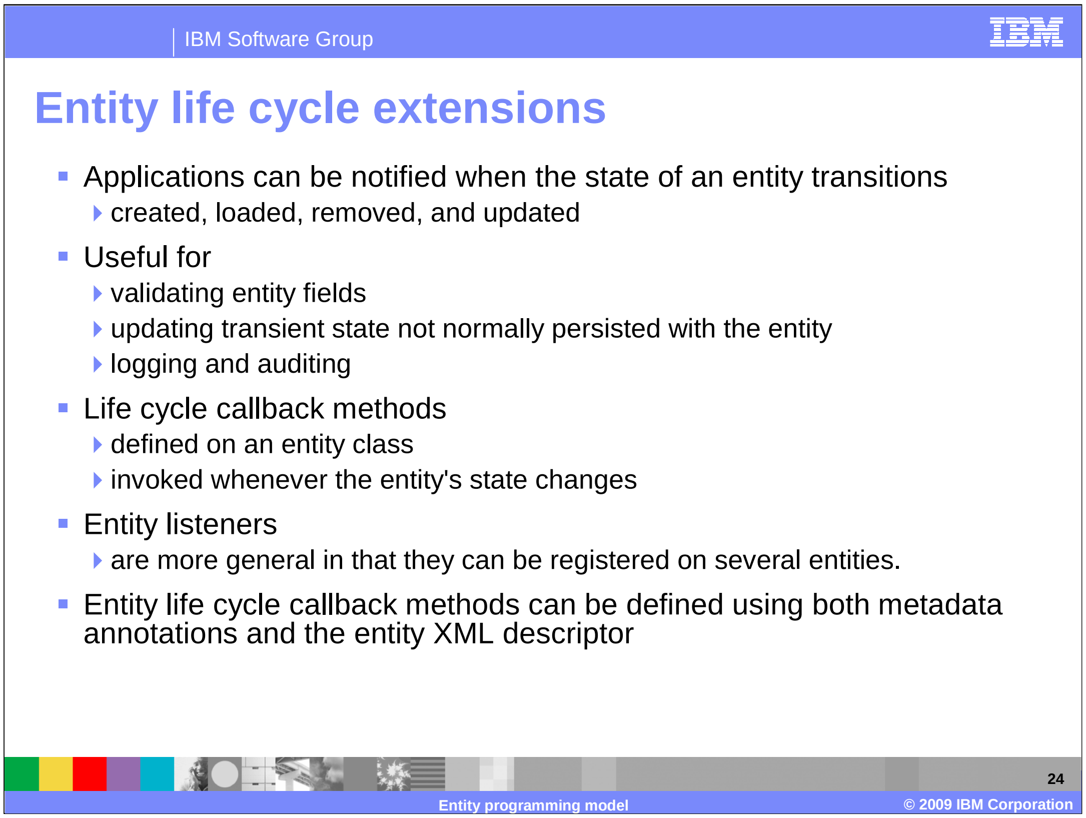
Life cycle extensions

23

Entity programming model

© 2009 IBM Corporation

The EntityManager also provides extension points that allow an application to be notified when significant life cycle events occur on an entity.



IBM Software Group

Entity life cycle extensions

- Applications can be notified when the state of an entity transitions
  - created, loaded, removed, and updated
- Useful for
  - validating entity fields
  - updating transient state not normally persisted with the entity
  - logging and auditing
- Life cycle callback methods
  - defined on an entity class
  - invoked whenever the entity's state changes
- Entity listeners
  - are more general in that they can be registered on several entities.
- Entity life cycle callback methods can be defined using both metadata annotations and the entity XML descriptor

24

Entity programming model

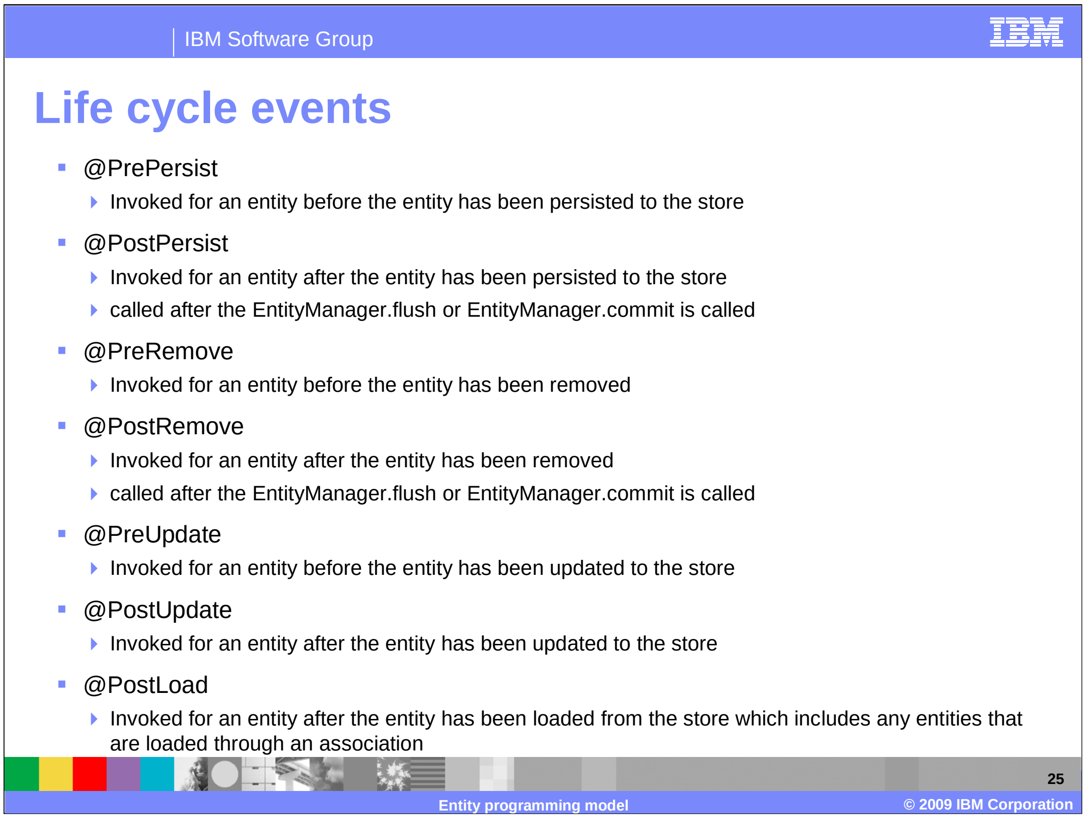
© 2009 IBM Corporation

Applications can be notified when the state of an entity transitions from state to state. ObjectGrid provides two callback mechanisms for state change events: life cycle callbacks and entity listeners.

The application defines life cycle callback methods on an entity class. The EntityManager invokes these callback methods whenever the entity’s state changes. Such methods allow an entity to validate its state before persisting or update transient state that is not normally persisted with the entity.

An entity listener class is a plain, non-entity, class that implements one or more entity life cycle callback methods. An entity listener can be associated with multiple entities. Entity listeners are useful for general purpose auditing or logging applications.

Both life cycle callback and entity listeners can be defined using metadata annotations or an entity metadata XML descriptor file.



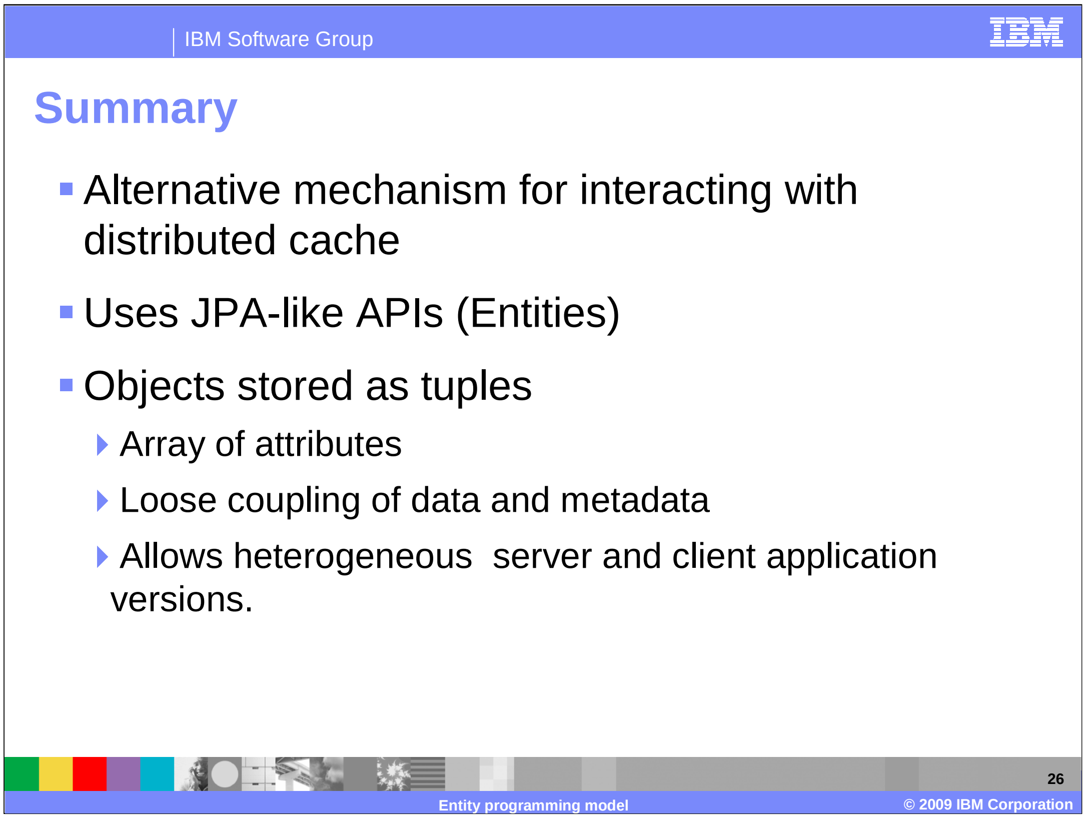
Life cycle events

- @PrePersist
  - Invoked for an entity before the entity has been persisted to the store
- @PostPersist
  - Invoked for an entity after the entity has been persisted to the store
  - called after the EntityManager.flush or EntityManager.commit is called
- @PreRemove
  - Invoked for an entity before the entity has been removed
- @PostRemove
  - Invoked for an entity after the entity has been removed
  - called after the EntityManager.flush or EntityManager.commit is called
- @PreUpdate
  - Invoked for an entity before the entity has been updated to the store
- @PostUpdate
  - Invoked for an entity after the entity has been updated to the store
- @PostLoad
  - Invoked for an entity after the entity has been loaded from the store which includes any entities that are loaded through an association

© 2009 IBM Corporation

Both life cycle callback and entity listeners receive the same set of life cycle notifications as shown here. State changes include storing or replacing the object in the grid, removing the entity from the grid or loading the entity into the grid.

The Java methods implementing the individual callbacks can have any name. The configuration maps events to method. For instance, an application can include a method called calculateTransientData that is called for PostLoad life cycle events.



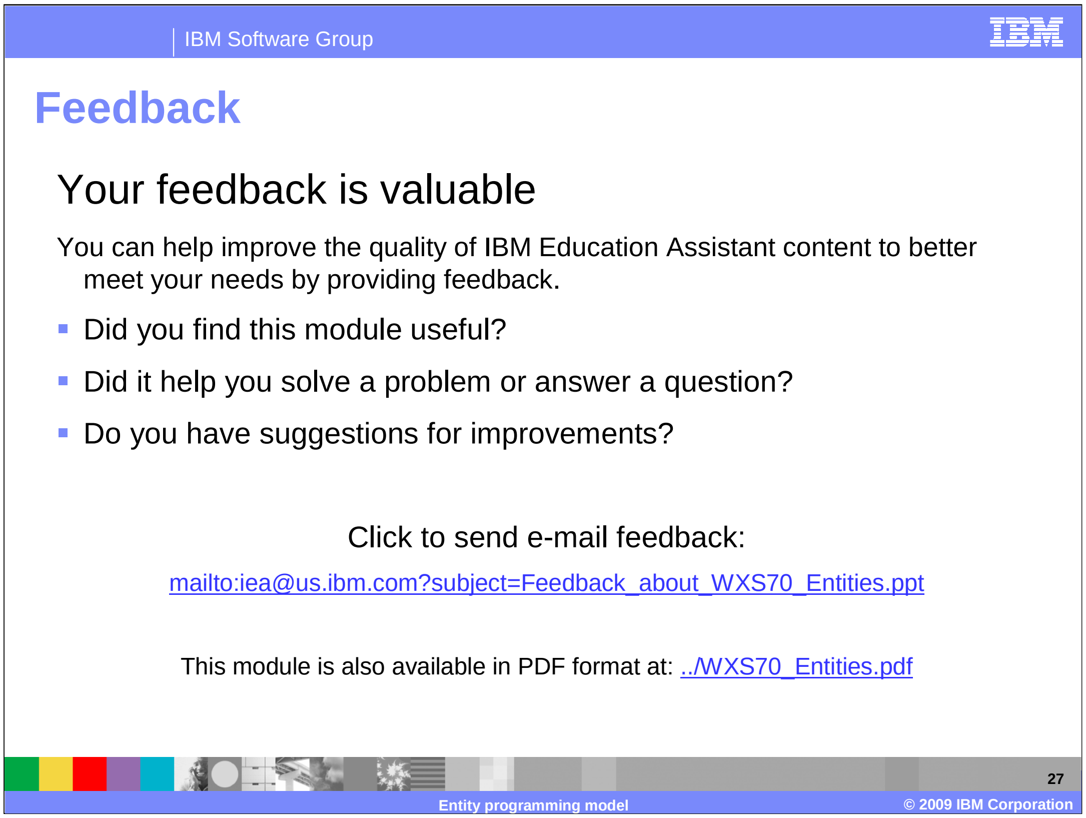
Summary

- Alternative mechanism for interacting with distributed cache
- Uses JPA-like APIs (Entities)
- Objects stored as tuples
  - Array of attributes
  - Loose coupling of data and metadata
  - Allows heterogeneous server and client application versions.

WebSphere eXtreme Scale provides two ways to store data; a map based interface that resembles a Java Map, and an Entity based API. The EntityManager API is similar to other Java object persistence technologies such as Java Persistence API (JPA) and Hibernate, in that it synchronizes a graph of managed Java object instances with the persistent store. In this case, the persistent store is an eXtreme Scale grid.

Entities are stored as tuples, an array of primitive attributes, rather than objects. Metadata describes attribute names and types for entities in the grid.

The eXtreme Scale EntityManager provides a powerful, efficient interface for storing and retrieving data from the grid.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

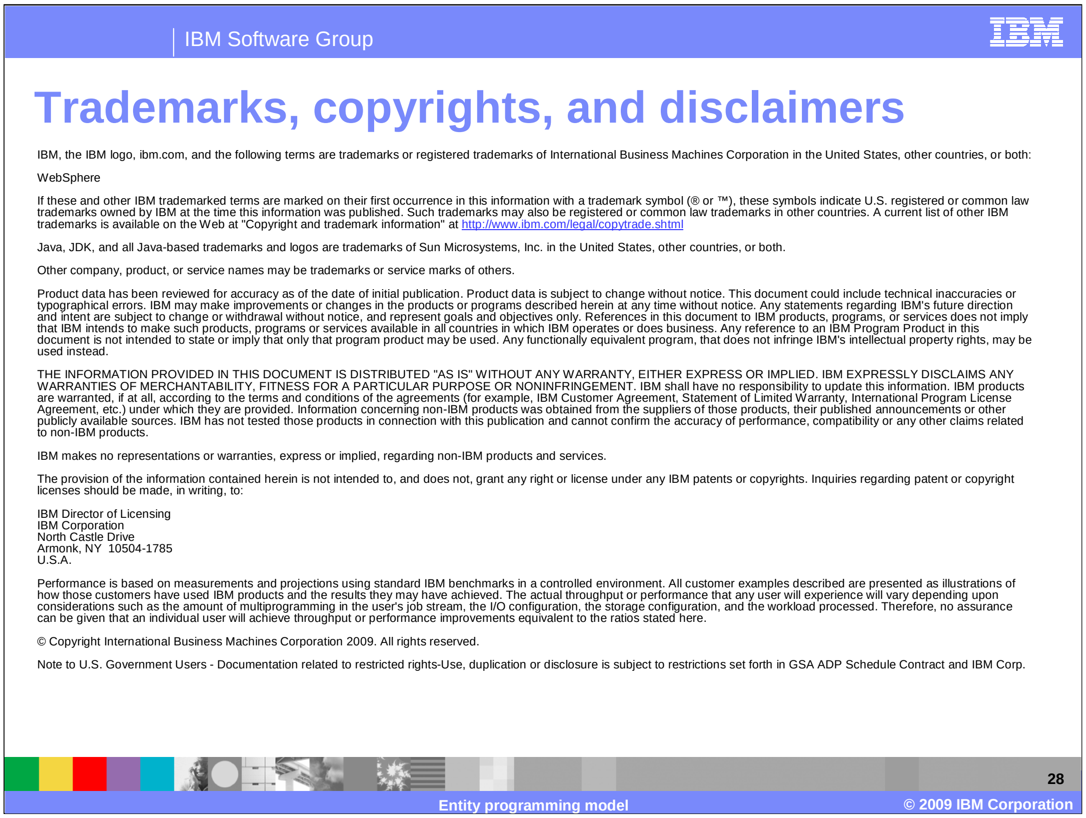
Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WXS70_Entities.ppt

This module is also available in PDF format at: ../WXS70_Entities.pdf

© 2009 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Java, JDK, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.