

Creating String-Based Queries

To create a string-based query, specify the attribute names of entity classes directly as strings, rather than the attributes of the metamodel class. For example, this query finds all `Pet` entities where the value of the `name` attribute is `Fido`:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where(cb.equal(pet.get("name"), "Fido"));
...
```

The name of the attribute is specified as a string. This query is the equivalent of the following metamodel query:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where(cb.equal(pet.get(Pet_.name), "Fido"));
```

Note - Type mismatch errors in string-based queries won't appear until the code is executed at runtime, unlike in the above metamodel query, where type mismatches will be caught at compile time.

Joins are specified in the same way:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
Join<Owner, Address> address = pet.join("owners").join("addresses");
...
```

All the conditional expressions, method expressions, path navigation methods, and result restriction methods used in metamodel queries can be used in string-based queries. In each case, the attributes are specified using strings. For example, here is a string-based query that uses the `in` expression:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where(pet.get("color").in("brown", "black"));
```

Here is a string-based query that orders the results in descending order by date:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.select(pet);
cq.orderBy(cb.desc(pet.get("birthday")));
```