

## Contents

- [Using Composite Keys](#)
  - [Usage](#)
  - [Configuration](#)
  - [Object Retrieval](#)
  - [Object Method Generation](#)
  - [Example](#)

## Using Composite Keys

### Usage

The composite key specification is used when you are using composite keys within your database, and need to specify to Transfer what the unique identifier is for each object.

Normally this would be done by the `<id>` element, however, this may not be possible due to the use of composite keys.

### Configuration

The XML configuration for composite keys uses a `<compositeid>` element to replace the usual `<id>` element

This looks like:

```
<compositeid>
  <property name= "propertyName" />
  <manytoone name= "manytooneName" />
  <parentonetomany class= "classOfParentOneToMany" />
</compositeid>
```

Where -

- The *name* attribute of the *property* element matches the *name* element of a configured *property* element within the *object*
- The *name* attribute of the *manytoone* element matches the *name* element of a configured *manytoone* element within the *object*
- The *class* attribute found on the *parentonetomany* is the class name of a *object* that has a *onetomany* composite element that points to the current *object* configuration.

Any number of these elements inside *compositeid* can be used more than once, or not all, to build the unique identifier for this *object*, based on it's wide columns and foreign key values.

Which one you use will depend on your object model configuration.

### Object Retrieval

It is expected that the majority of composite id object retrieval is done via foreign key relationships, however to retrieve a specific object, the following extension to the 'get' method is available.

`getTransfer().get()` allows either Strings, or Structs through the *key* argument, depending on if the ID is composite or not.

When the type of ID for the object is an element of type `<compositeid>` *key* argument will require a Struct with key-value pairs.

For either *property* or *manytoone* composite ID elements, the *name* attribute is used as the key in the struct, and the property or foreign key value is provided as the value respectively.

For a *parentonetomany* composite id element, the key will be 'parent' + the Object name of the class that is specified. (For example, if you have a class of 'user.User', the Object name would be 'User'), and the value is the foreign key that maps to this composite element on the object.

Again this may be confusing, but it mimics how it works on the TransferObject, which means you won't get clashes between *property* names and *Object* names in the Struct.

If a key is not present within the Struct provided to `compositeID`, the value NULL will be used when querying for the Object.

### Object Method Generation


TransferObjects with a *compositeid* element have the method 'getCompositeID()' which will return a pipe (|) delimited string containing all the values of the composite IDs configured against the object. If no value is present, an empty string will be returned for that specific value.

### Example

Taken from the tPetmarket sample application (see [Example Code](#)).

```
<object name= "ShoppingCartItem" table= "user_shoppingCart" decorator= "petmarket.com.user.ShoppingCartItem" >
  <-- use a composite key comprised of User,Item,created. Including the date
  prevents an issue with a user purchasing the same item more than once.-->
  <compositeid>
    <manytoone name= "User" />
    <manytoone name= "Item" />
    <property name= "created" />
  </compositeid>
```

```
</compositeid>
<manytoone name="User" >
  <link column="userid" to="user.User" />
</manytoone>
<manytoone name="Item" >
  <link column="itemid" to="pets.Item" />
</manytoone>
<property name="created" type="date" column="dateCreated" />
<property name="quantity" type="numeric" />
</object>
```

 Categories:

- [Configuration](#)