

Key Paths



In this paper...

Topic	See page
Introduction	1
What is a Key Path?	1
Example	2
Automatic Key Maintenance	3
Why Is An Inverse Required For Each Key Path Component?	3
Resolving 1099 Exceptions	4
Setting Key Path References to Null	5
Summary	5

Introduction

This document discusses the use of key paths in a JADE system. It focuses on those cases where a key path dictionary participates as an end point in a bidirectional relationship. In such cases, JADE will automatically maintain the dictionary when any of the key path components are updated. This document covers some of the issues you should consider when using key paths in your JADE applications.

What Is A Key Path?

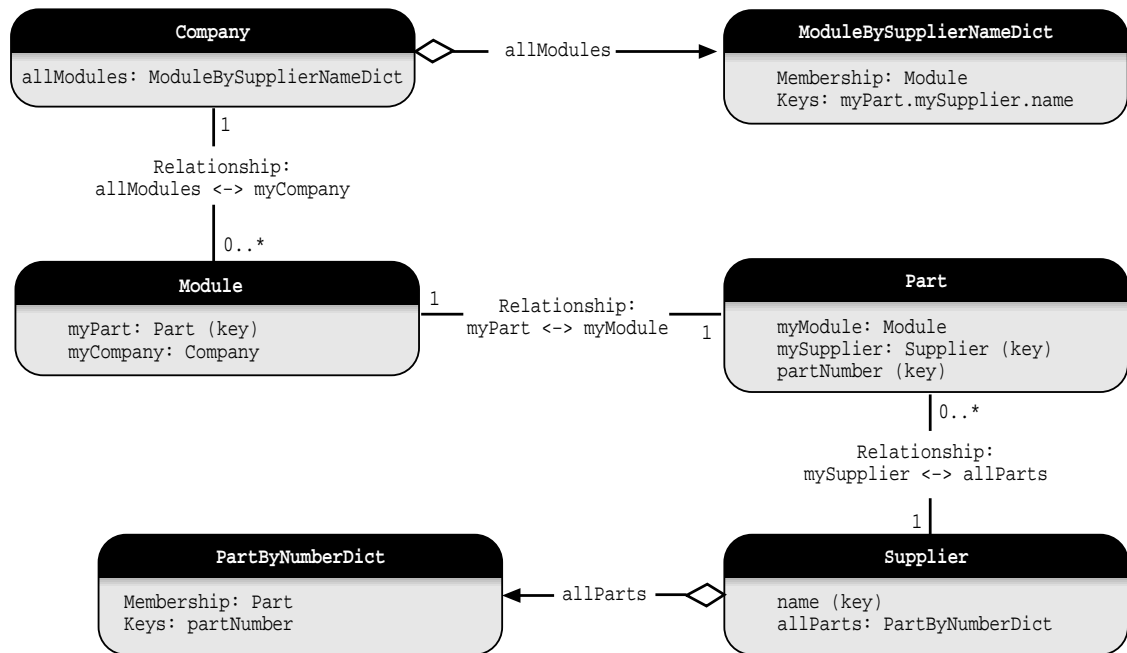
A key path is a mechanism that allows you to define an automatically maintained dictionary key that is not an embedded property of the members of the dictionary, but is instead derived from the member objects.

When you define a key path, you specify a chain of references starting from the member class and finishing at an end point.

At run time, the references are traversed to arrive at the end point that yields the key value. In the following diagram, **myPart.mySupplier.name** is the key path of the **ModuleBySupplierNameDict** dictionary.

Example

The discussion about key paths that follows uses the example shown in the following diagram.



In this example, each company has a dictionary of all of its modules (`allModules` of type `ModuleBySupplierNameDict`).

This dictionary is keyed by the supplier name of the module's part using a key path; that is, `myPart.mySupplier.name`. Each module knows the company to which it belongs by means of `myCompany` (which is the inverse of

`Company::allModules`), and the part that it contains by means of `myPart`.

Modules are used to add additional semantics to parts. Each part knows its associated module by means of `myModule` (which is the inverse of `Module::myPart`), and its supplier by means of `mySupplier`. Each supplier knows all of the parts that it supplies by means of `allParts` (which is the inverse of `Part::mySupplier`).

Automatic Key Maintenance

Dictionary keys are automatically maintained when the dictionary has one or more explicit inverse references; that is, when it participates as an end point in a bidirectional relationship. For example, if **allModules** and **allParts** did not have their respective inverses **myCompany** and **mySupplier**, automatic key maintenance would not be performed for them.

When a dictionary property participates in a relationship (as **allModules** and **allParts** do in this example), if any of the non-key path keys are changed, the dictionary is automatically updated to reflect the new key values.

Automatic key maintenance is also performed for key path keys. This means that when a module has been added to **allModules**, if any component of the key path **myPart.mySupplier.name** is changed, **allModules** is automatically updated to reflect the new key value. Again, this automatic key maintenance is performed only because **allModules** participates in a relationship (that is, it has **myCompany** as an inverse).

Why Is An Inverse Required For Each Key Path Component?

To perform automatic key maintenance for key paths, JADE requires each class on which a key path property is defined to have an inverse back to the prior key path property. This is required so that if any point along a key path is changed, JADE can find its way back to the related collection or collections to update key values.

This requirement means that **myPart** must have the inverse **myModule** and **mySupplier** must have the

inverse **allParts** in the **myPart.mySupplier.name** key path. If either of these inverses were missing, the first attempt to use the **ModuleBySupplierNameDict** collection in a relationship (in this case **allModules**) would result in a 1099 exception (*Key path component does not have an inverse to its prior component*) at run time.

Resolving 1099 Exceptions

If your application encounters 1099 exceptions, the following options are available to you:

- Add the required inverses. In the above example, you would add the **myModule** inverse to **myPart** and the **allParts** inverse to **mySupplier**. If inverses must be added to a system that is already deployed, a database reorganization may be required. Ensure that you take a full backup of your system before applying any schema modifications.
- Avoid using the key path collection in a relationship. In the above example, if you removed the **myCompany** inverse of **allModules**, then **allModules** no longer participates in a relationship. As such, no automatic key maintenance is performed and the inverses back along the key path chain are not required. If inverses must be removed in a system that is already deployed, a database reorganization may be required. Ensure that you take a full backup of your system before applying any schema modifications.

Note: Because no automatic key maintenance is performed, it is your application's responsibility to ensure that keys are not changed once an object has been added to a key path collection. If the keys are changed, your application must properly manage the maintenance of the collection members.

- In the [JadeClient] section of the JADE initialization file on both your client and your server, add the following entry:

```
AllowKeyPathsWithoutInverses=true
```

Setting this parameter to **true** tells JADE not to raise an exception if it encounters a key path without reverse inverses, and not to perform automatic key maintenance for key path properties *other than the first property in the key path*. Changes to the first property in a key path can always update related dictionaries, as this property must always have an inverse in order for the key path dictionary to be participating in the relationship.

Note: With this initialization file parameter set to **true**, it is your application's responsibility to ensure that keys are not changed after an object has been added to a key path collection. If the keys are changed, your application must properly manage the maintenance of the collection members.

For example, when a module has been added to **allModules** and you want to change a property on its key path, your application must take care of removing the module from **allModules** before the change, changing the key properties, and then adding the module to **allModules** again. This process is performed automatically by JADE if you elect to use automatic key path maintenance.

Setting Key Path References to Null

When using key paths, take care when setting intermediate references in the key path to null (either directly by assigning to the property, or indirectly by deleting the referenced object). For example, assume that **allModules** contains five different modules, with each module related to a different part and with each part supplied by a different supplier, as listed in the following table.

Module	myPart	myPart.mySupplier	myPart.mySupplier.name (key value)
module1	part1	supplier1	"Supplier One"
module2	part2	supplier2	"Supplier Two"
module3	part3	supplier3	"Supplier Three"
module4	part4	supplier4	"Supplier Four"
module5	part5	supplier5	"Supplier Five"

Imagine that for the first Part instance (**part1**), you set its supplier reference to null or you delete the supplier (which implicitly sets **mySupplier** to null because of the **Supplier::allParts** inverse). **mySupplier** is part of the key path and it is being updated, so **allModules** must automatically be updated to change the key value of the entry for **module1**. Because **myPart.mySupplier** is now null, meaning that **myPart.mySupplier.name** cannot be evaluated, the key value in the dictionary for **module1** is set to null.

Now imagine that you set **mySupplier** on the second Part instance (**part2**) to null. The same thing happens, only it now results in a duplicate key value in **allModules**, because **module1** already exists with a null key. This will be a problem only if **allModules** does not allow duplicates.

Summary

- For key path dictionaries that implement the end point of a relationship, ensure that each reference in the key path has at least one inverse back to the prior reference in the key path.
- When working with key paths and no-duplicate dictionaries, ensure that key path references are null for one member object at a time only.
- For any dictionary that does not participate in a relationship (regardless of whether or not it has key paths), automatic key maintenance is not performed.

Such dictionaries are often referred to as *manually maintained* dictionaries. For manually maintained dictionaries, it is your application's responsibility to ensure that keys are not changed once an object has been added to the dictionary; or if the keys are changed, your application must maintain the dictionary members. This is one of the reasons why it is recommended that, certainly in your persistent object model, you define inverses for your dictionary properties.