Registering Dependent Keys

There are many situations in which the value of one property depends on that of one or more other attributes in another object. If the value of one attribute changes, then the value of the derived property should also be flagged for change. How you ensure that key-value observing notifications are posted for these dependent properties depends on the cardinality of the relationship.

To-one Relationships

To trigger notifications automatically for a to-one relationship you should either override `keyPathsForValuesAffectingValueForKey:` or implement a suitable method that follows the pattern it defines for registering dependent keys.

For example, the full name of a person is dependent on both the first and last names. A method that returns the full name could be written as follows:

```
- (NSString *)fullName {
    return [NSString stringWithFormat:@"%@ %@",firstName, lastName];
}
```

An application observing the `fullName` property must be notified when either the `firstName` or `lastName` properties change, as they affect the value of the property.

One solution is to override `keyPathsForValuesAffectingValueForKey:` specifying that the `fullName` property of a person is dependent on the `lastName` and `firstName` properties. Listing 1 shows an example implementation of such a dependency:

Listing 1 Example implementation of `keyPathsForValuesAffectingValueForKey:`

```
+ (NSSet *)keyPathsForValuesAffectingValueForKey:(NSString *)key {

    NSSet *keyPaths = [super keyPathsForValuesAffectingValueForKey:key];

    if ([key isEqualToString:@"fullName"]) {
        NSArray *affectingKeys = @[@"lastName", @"firstName"];
        keyPaths = [keyPaths setByAddingObjectsFromArray:affectingKeys];
    }
    return keyPaths;
}
```

Your override should typically invoke super and return a set that includes any members in the set that result from doing that (so as not to interfere with overrides of this method in superclasses).

You can also achieve the same result by implementing a class method that follows the naming convention `keyPathsForValuesAffecting<Key>`, where `<Key>` is the name of the attribute (first letter capitalized) that is dependent on the values. Using this pattern the code in Listing 1 could be rewritten as a class method named `keyPathsForValuesAffectingFullName` as shown in Listing 2.

Listing 2 Example implementation of the `keyPathsForValuesAffecting<Key>` naming convention

```
+ (NSSet *)keyPathsForValuesAffectingFullName {
```

```
    return [NSSet setWithObjects:@"lastName", @"firstName", nil];
}
```

You can't override the `keyPathsForValuesAffectingValueForKey:` method when you add a computed property to an existing class using a category, because you're not supposed to override methods in categories. In that case, implement a matching `keyPathsForValuesAffecting<Key>` class method to take advantage of this mechanism.

Note: You cannot set up dependencies on to-many relationships by implementing `keyPathsForValuesAffectingValueForKey:`. Instead, you must observe the appropriate attribute of each of the objects in the to-many collection and respond to changes in their values by updating the dependent key yourself. The following section shows a strategy for dealing with this situation.

To-many Relationships

The `keyPathsForValuesAffectingValueForKey:` method does not support key-paths that include a to-many relationship. For example, suppose you have a Department object with a to-many relationship (`employees`) to a Employee, and Employee has a salary attribute. You might want the Department object have a `totalSalary` attribute that is dependent upon the salaries of all the Employees in the relationship. You can not do this with, for example, `keyPathsForValuesAffectingTotalSalary` and returning `employees.salary` as a key.

There are two possible solutions in both situations:

1. You can use key-value observing to register the parent (in this example, Department) as an observer of the relevant attribute of all the children (Employees in this example). You must add and remove the parent as an observer as child objects are added to and removed from the relationship (see Registering for Key-Value Observing). In the `observeValueForKeyPath:ofObject:change:context:` method you update the dependent value in response to changes, as illustrated in the following code fragment:

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:
(NSDictionary *)change context:(void *)context {


    if (context == totalSalaryContext) {
        [self updateTotalSalary];
    }
    else
    // deal with other observations and/or invoke super...
}


- (void)updateTotalSalary {
    [self setTotalSalary:[self valueForKeyPath:@"employees.@sum.salary"]];
}


- (void)setTotalSalary:(NSNumber *)newTotalSalary {


    if (totalSalary != newTotalSalary) {
        [self willChangeValueForKey:@"totalSalary"];
        _totalSalary = newTotalSalary;
        [self didChangeValueForKey:@"totalSalary"];
```

```
    }
}

- (NSNumber *)totalSalary {
    return _totalSalary;
}
```

2. If you're using Core Data, you can register the parent with the application's notification center as an observer of its managed object context. The parent should respond to relevant change notifications posted by the children in a manner similar to that for key-value observing.
