Introduction to Key-Value Observing Programming Guide

Key-value observing is a mechanism that allows objects to be notified of changes to specified properties of other objects.

Important: In order to understand key-value observing, you must first understand key-value coding.

At a Glance

Key-value observing provides a mechanism that allows objects to be notified of changes to specific properties of other objects. It is particularly useful for communication between model and controller layers in an application. (In OS X, the controller layer binding technology relies heavily on key-value observing.) A controller object typically observes properties of model objects, and a view object observes properties of model objects through a controller. In addition, however, a model object may observe other model objects (usually to determine when a dependent value changes) or even itself (again to determine when a dependent value changes).

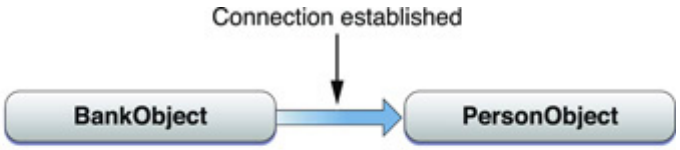
You can observe properties including simple attributes, to-one relationships, and to-many relationships. Observers of to-many relationships are informed of the type of change made—as well as which objects are involved in the change.

There are three steps to setting up an observer of a property. Understanding these three steps provides a clear illustration of how KVO works.

1. First, see whether you have a scenario where key-value observing could be beneficial, for example, an object that needs to be notified when any changes are made to a specific property in another object.

   For example, a `PersonObject` will want to be aware of any changes made to their `accountBalance` in the `BankObject`.

2. The `PersonObject` must register as an observer of the `BankObject`'s `accountBalance` property by sending an `addObserver:forKeyPath:options:context:` message.

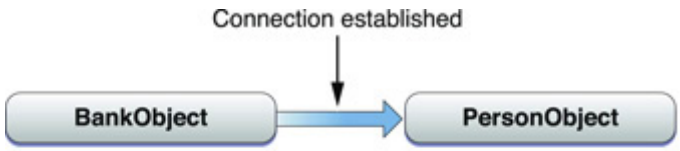



```
[bankInstance addObserver:personInstance
    forKeyPath:@"accountBalance"
       options:NSKeyValueObservingOptionNew
          context:NULL];
```

Note: The `addObserver:forKeyPath:options:context:` method establishes a connection between the instances of the objects that you specify. A connection is not

established between the two classes, but rather between the two specified instances of the objects.

3. In order to respond to change notifications, the observer must implement the `observeValueForKeyPath:ofObject:change:context:` method. This method implementation defines how the observer responds to change notifications. It is in this method that you can customize your response to a change in one of the observed properties.

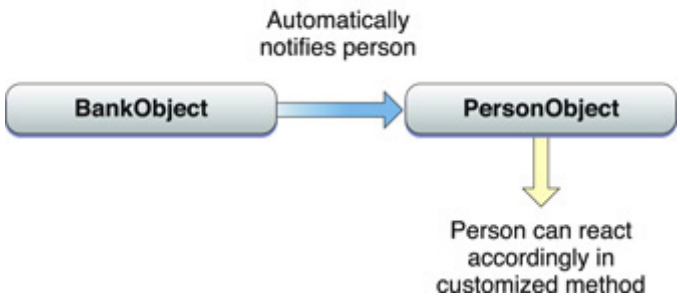



```
-(void) observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
        change:(NSDictionary *)change
            context:(void *)context
{

//custom implementation
//be sure to call the superclass' implementation
//if the superclass implements it

}
```

Registering for Key-Value Observing describes how to register and receive observation notifications.

4. The `observeValueForKeyPath:ofObject:change:context:` method is automatically invoked when the value of an observed property is changed in a KVO-compliant manner, or if a key upon which it depends is changed.

```
[bankInstance setAccountBalance:50];
```



Registering Dependent Keys explains how to specify that the value of a key is dependent on the value of another key.

KVO's primary benefit is that you don't have to implement your own scheme to send notifications every time a property changes. Its well-defined infrastructure has framework-level support that makes it easy to adopt—typically you do not have to add any code to your project. In addition, the infrastructure is already full-featured, which makes it easy to support multiple observers for a single property, as well as dependent values.

KVO Compliance describes the difference between automatic and manual key-value observing, and how to implement both.

Unlike notifications that use `NSNotificationCenter`, there is no central object that provides change notification for all observers. Instead, notifications are sent directly to the observing objects when changes are made. `NSObject` provides this base implementation of key-value observing, and you should rarely need to override these methods.

Key-Value Observing Implementation Details describes how key-value observing is implemented.