Accessor Search Patterns

On This Page

The default implementation of the `NSKeyValueCoding` protocol provided by `NSObject` maps key-based accessor calls to an object's underlying properties using a clearly defined set of rules. These protocol methods use a key parameter to search their own object instance for accessors, instance variables, and related methods that follow certain naming conventions. Although you rarely modify this default search, it can be helpful to understand how it works, both for tracing the behavior of key-value coded objects, and for making your own objects compliant.

> NOTE
>
> The descriptions in this section use <key> or <Key> as a placeholder for the key string that appears as a parameter in one of the key-value coding protocol methods, which is then used by that method as part of a secondary method call or variable name lookup. The mapped property name obeys the placeholder's case. For example, for the getters <key> and `is`<Key>, the property named hidden maps to `hidden` and `isHidden`.

Search Pattern for the Basic Getter

The default implementation of `valueForKey:`, given a `key` parameter as input, carries out the following procedure, operating from within the class instance receiving the `valueForKey:` call.

1. Search the instance for the first accessor method found with a name like `get`<Key>, <key>, `is`<Key>, or `_`<key>, in that order. If found, invoke it and proceed to step 5 with the result. Otherwise proceed to the next step.
2. If no simple accessor method is found, search the instance for methods whose names match the patterns `countOf`<Key> and `objectIn`<Key>`AtIndex:` (corresponding to the primitive methods defined by the `NSArray` class) and <key>`AtIndexes:` (corresponding to the `NSArray` method `objectsAtIndexes:`).

   If the first of these and at least one of the other two is found, create a collection proxy object that responds to all `NSArray` methods and return that. Otherwise, proceed to step 3.

   The proxy object subsequently converts any `NSArray` messages it receives to some combination of `countOf`<Key>, `objectIn`<Key>`AtIndex:`, and <key>`AtIndexes:` messages to the key-value coding compliant object that created it. If the original object also implements an optional method with a name like `get`<Key>`:range:`, the proxy object uses that as well, when appropriate. In effect, the proxy object working together with the key-value coding compliant object allows the underlying property to behave as if it were an `NSArray`, even if it is not.
3. If no simple accessor method or group of array access methods is found, look for a triple of methods named `countOf`<Key>, `enumeratorOf`<Key>, and `memberOf`<Key>`:` (corresponding to the primitive methods defined by the `NSSet` class).

   If all three methods are found, create a collection proxy object that responds to all `NSSet` methods and return that. Otherwise, proceed to step 4.

   This proxy object subsequently converts any `NSSet` message it receives into some combination of `countOf`<Key>, `enumeratorOf`<Key>, and `memberOf`<Key>`:` messages to the object that created it. In effect, the proxy object working together with the key-value coding compliant object allows the underlying property to behave as if it were an `NSSet`, even if it is not.
4. If no simple accessor method or group of collection access methods is found, and if the receiver's class method `accessInstanceVariablesDirectly` returns `YES`, search for an instance variable named `_`<key>, `_is`<Key>, <key>, or `is`<Key>, in that order. If found, directly obtain the value of the instance variable and proceed to step 5. Otherwise, proceed to step 6.
5. If the retrieved property value is an object pointer, simply return the result.

   If the value is a scalar type supported by `NSNumber`, store it in an `NSNumber` instance and return that.

   If the result is a scalar type not supported by NSNumber, convert to an `NSValue` object and return that.
6. If all else fails, invoke `valueForUndefinedKey:`. This raises an exception by default, but a subclass of `NSObject` may provide key-specific behavior.

Search Pattern for the Basic Setter

The default implementation of `setValue:forKey:`, given `key` and `value` parameters as input, attempts to set a property named `key` to `value` (or, for non-object properties, the unwrapped version of `value`, as described in

Representing Non-Object Values) inside the object receiving the call, using the following procedure:

1. Look for the first accessor named `set<Key>:` or `_set<Key>`, in that order. If found, invoke it with the input value (or unwrapped value, as needed) and finish.
2. If no simp
   look for an instance variable with a name like `_<key>`, `_is<Key>`, `<key>`, or `is<Key>`, in that order. If found, set the variable directly with the input value (or unwrapped value) and finish.
3. Upon finding no accessor or instance variable, invoke `setValue:forUndefinedKey:`. This raises an exception by default, but a subclass of `NSObject` may provide key-specific behavior.

On This Page

Search Pattern for Mutable Arrays

The default implementation of `mutableArrayValueForKey:`, given a `key` parameter as input, returns a mutable proxy array for a property named `key` inside the object receiving the accessor call, using the following procedure:

1. Look for a pair of methods with names like `insertObject:in<Key>AtIndex:` and `removeObjectFrom<Key>AtIndex:` (corresponding to the `NSMutableArray` primitive methods `insertObject:atIndex:` and `removeObjectAtIndex:` respectively), or methods with names like `insert<Key>:atIndexes:` and `remove<Key>AtIndexes:` (corresponding to the `NSMutableArrayinsertObjects:atIndexes:` and `removeObjectsAtIndexes:` methods).

   If the object has at least one insertion method and at least one removal method, return a proxy object that responds to `NSMutableArray` messages by sending some combination of `insertObject:in<Key>AtIndex:`, `removeObjectFrom<Key>AtIndex:`, `insert<Key>:atIndexes:`, and `remove<Key>AtIndexes:` messages to the original receiver of `mutableArrayValueForKey:`.

   When the object receiving a `mutableArrayValueForKey:` message also implements an optional replace object method with a name like `replaceObjectIn<Key>AtIndex:withObject:` or `replace<Key>AtIndexes:with<Key>:`, the proxy object utilizes those as well when appropriate for best performance.
2. If the object does not have the mutable array methods, look instead for an accessor method whose name matches the pattern `set<Key>:`. In this case, return a proxy object that responds to `NSMutableArray` messages by issuing a `set<Key>:` message to the original receiver of `mutableArrayValueForKey:`.

   > NOTE
   > The mechanism described in this step is much less efficient than that of the previous step, because it may involve repeatedly creating new collection objects instead of modifying an existing one. Therefore, you should generally avoid it when designing your own key-value coding compliant objects.

3. If neither the mutable array methods, nor the accessor are found, and if the receiver's class responds `YES` to `accessInstanceVariablesDirectly`, search for an instance variable with a name like `_<key>` or `<key>`, in that order.

   If such an instance variable is found, return a proxy object that forwards each `NSMutableArray` message it receives to the instance variable's value, which typically is an instance of `NSMutableArray` or one of its subclasses.
4. If all else fails, return a mutable collection proxy object that issues a `setValue:forUndefinedKey:` message to the original receiver of the `mutableArrayValueForKey:` message whenever it receives an `NSMutableArray` message.

   The default implementation of setValue:forUndefinedKey: raises an `NSUndefinedKeyException`, but subclasses may override this behavior.

Search Pattern for Mutable Ordered Sets

The default implementation of `mutableOrderedSetValueForKey:` recognizes the same simple accessor methods and ordered set accessor methods as `valueForKey:` (see Default Search Pattern for the Basic Getter), and follows the same direct instance variable access policies, but always returns a mutable collection proxy object instead of the immutable collection that `valueForKey:` returns. In addition, it does the following:

1. Search for methods with names like `insertObject:in<Key>AtIndex:` and `removeObjectFrom<Key>AtIndex:` (corresponding to the two most primitive methods defined by the

NSMutableOrderedSet class), and also `insert<Key>:atIndexes:` and `remove<Key>AtIndexes:` (corresponding to `insertObjects:atIndexes:` and `removeObjectsAtIndexes:`).

If at least one insertion method and at least one removal method are found, the returned proxy object sends sor
`insert<Key>:atIndexes:`, and `remove<Key>AtIndexes:` messages to the original receiver of the `mutableOrderedSetValueForKey:` message when it receives `NSMutableOrderedSet` messages.

The proxy object also makes use of methods with names like `replaceObjectIn<Key>AtIndex:withObject:` or `replace<Key>AtIndexes:with<Key>:` when they exist in the original object.

2. If the mutable set methods are not found, search for an accessor method with a name like `set<Key>:`. In this case, the returned proxy object sends a `set<Key>:` message to the original receiver of `mutableOrderedSetValueForKey:` every time it receives a `NSMutableOrderedSet` message.

> NOTE
> The mechanism described in this step is much less efficient than that of the previous step, because it may involve repeatedly creating new collection objects instead of modifying an existing one. Therefore, you should generally avoid it when designing your own key-value coding compliant objects.

3. If neither the mutable set messages nor the accessor are found, and if the receiver's `accessInstanceVariablesDirectly` class method returns `YES`, search for an instance variable with a name like `_<key>` or `<key>`, in that order. If such an instance variable is found, the returned proxy object forwards any `NSMutableOrderedSet` messages it receives to the instance variable's value, which is typically an instance of `NSMutableOrderedSet` or one of its subclasses.
4. If all else fails, the returned proxy object sends a `setValue:forUndefinedKey:` message to the original receiver of `mutableOrderedSetValueForKey:` whenever it receives a mutable set message.

   The default implementation of `setValue:forUndefinedKey:` raises an `NSUndefinedKeyException`, but objects may override this behavior.

Search Pattern for Mutable Sets

The default implementation of `mutableSetValueForKey:`, given a `key` parameter as input, returns a mutable proxy set for an array property named `key` inside the object receiving the accessor call, using the following procedure:

1. Search for methods with names like `add<Key>Object:` and `remove<Key>Object:` (corresponding to the `NSMutableSet` primitive methods `addObject:` and `removeObject:` respectively) and also `add<Key>:` and `remove<Key>:` (corresponding to `NSMutableSet` methods `unionSet:` and `minusSet:`). If at least one addition method and at least one removal method are found, return a proxy object that sends some combination of `add<Key>Object:`, `remove<Key>Object:`, `add<Key>:`, and `remove<Key>:` messages to the original receiver of `mutableSetValueForKey:` for each `NSMutableSet` message it receives.

   The proxy object also makes use of the methods with a name like `intersect<Key>:` or `set<Key>:` for best performance, if they are available.
2. If the receiver of the `mutableSetValueForKey:` call is a managed object, the search pattern does not continue as it would for non-managed objects. See Managed Object Accessor Methods in *Core Data Programming Guide* for more information.
3. If the mutable set methods are not found, and if the object is not a managed object, search for an accessor method with a name like `set<Key>:`. If such a method is found, the returned proxy object sends a `set<Key>:` message to the original receiver of `mutableSetValueForKey:` for each `NSMutableSet` message it receives.

> NOTE
> The mechanism described in this step is much less efficient than that of the first step, because it may involve repeatedly creating new collection objects instead of modifying an existing one. Therefore, you should generally avoid it when designing your own key-value coding compliant objects.

4. If the mutable set methods and the accessor method are not found, and if the `accessInstanceVariablesDirectly` class method returns `YES`, search for an instance variable with a name like `_<key>` or `<key>`, in that order. If such an instance variable is found, the proxy object forwards each `NSMutableSet` message it receives to the instance variable's value, which is typically an instance of `NSMutableSet` or one of its subclasses.

5. If all else fails, the returned proxy object responds to any `NSMutableSet` message it receives by sending a `setValue:forUndefinedKey:` message to the original receiver of `mutableSetValueForKey:`.

On This Page