About Key-Value Coding

On This Page

Key-value coding is a mechanism enabled by the `NSKeyValueCoding` informal protocol that objects adopt to provide indirect access to their properties. When an object is key-value coding compliant, its properties are addressable via string parameters through a concise, uniform messaging interface. This indirect access mechanism supplements the direct access afforded by instance variables and their associated accessor methods.

You typically use accessor methods to gain access to an object's properties. A get accessor (or getter) returns the value of a property. A set accessor (or setter) sets the value of a property. In Objective-C, you can also directly access a property's underlying instance variable. Accessing an object property in any of these ways is straightforward, but requires calling on a property-specific method or variable name. As the list of properties grows or changes, so also must the code which accesses these properties. In contrast, a key-value coding compliant object provides a simple messaging interface that is consistent across all of its properties.

Key-value coding is a fundamental concept that underlies many other Cocoa technologies, such as key-value observing, Cocoa bindings, Core Data, and AppleScript-ability. Key-value coding can also help to simplify your code in some cases.

Using Key-Value Coding Compliant Objects

Objects typically adopt key-value coding when they inherit from `NSObject` (directly or indirectly), which both adopts the `NSKeyValueCoding` protocol and provides a default implementation for the essential methods. Such an object enables other objects, through a compact messaging interface, to do the following:

- **Access object properties.** The protocol specifies methods, such as the generic getter `valueForKey:` and the generic setter `setValue:forKey:`, for accessing object properties by their name, or key, parameterized as a string. The default implementation of these and related methods use the key to locate and interact with the underlying data, as described in Accessing Object Properties.
- **Manipulate collection properties.** The default implementations of the access methods work with an object's collection properties (such as `NSArray` objects) just like any other property. In addition, if an object defines collection accessor methods for a property, it enables key-value access to the contents of the collection. This is often more efficient than direct access and allows you to work with custom collection objects through a standardized interface, as described in Accessing Collection Properties.
- **Invoke collection operators on collection objects.** When accessing a collection property in a key-value coding compliant object, you can insert a *collection operator* into the key string, as described in Using Collection Operators. Collection operators instruct the default `NSKeyValueCoding` getter implementation to take an action on the collection, and then return either a new, filtered version of the collection, or a single value representing some characteristic of the collection.
- **Access non-object properties.** The default implementation of the protocol detects non-object properties, including both scalars and structures, and automatically wraps and unwraps them as objects for use on the protocol interface, as described in Representing Non-Object Values. In addition, the protocol declares a method allowing a compliant object to provide a suitable action for the case when a `nil` value is set on a non-object property through the key-value coding interface.
- **Access properties by key path.** When you have a hierarchy of key-value coding compliant objects, you can use key path based method calls to drill down, getting or setting a value deep within the hierarchy using a single call.

Adopting Key-Value Coding for an Object

In order to make your own objects key-value coding compliant, you ensure that they adopt the `NSKeyValueCoding` informal protocol and implement the corresponding methods, such as `valueForKey:` as a generic getter and `setValue:forKey:` as a generic setter. Fortunately, as described above, `NSObject` adopts this protocol and provides default implementations for these and other essential methods. Therefore, if you derive your objects from `NSObject` (or any of its many subclasses), much of the work is already done for you.

In order for the default methods to do their work, you ensure your object's accessor methods and instance variables adhere to certain well-defined patterns. This allows the default implementation to find your object's properties in response to key-value coded messages. You then optionally extend and customize key-value coding by providing methods for validation and for handling certain special cases.

Key-Value Coding with Swift

Swift objects that inherit from `NSObject` or one of its subclasses are key-value coding compliant for their properties by default. Whereas in Objective-C, a property's accessors and instance variables must follow certain patterns,
many of the protocol's features are either not relevant or are better handled using native Swift constructs or techniques that do not exist in Objective-C. For example, because all Swift properties are objects, you never exercise the default implementation's special handling of non-object properties.

On This Page

Therefore, while the key-value coding protocol methods translate straightforwardly to Swift, this guide focuses primarily on Objective-C, where you need to do more to ensure compliance, and where key-value coding is often most useful. Situations that call for a significantly different approach in Swift are noted throughout the guide.

For more information about using Swift with Cocoa technologies, read *Using Swift with Cocoa and Objective-C (Swift 3.0.1)*. For a complete description of Swift, read *The Swift Programming Language (Swift 3.0.1)*.

Other Cocoa Technologies Rely on Key-Value Coding

An object that is key-value coding compliant can participate in a wide range of Cocoa technologies that depend upon this kind of access, including:

- **Key-value observing.** This mechanism enables objects to register for asynchronous notifications driven by changes in another object's properties, as described in *Key-Value Observing Programming Guide*.
- **Cocoa bindings.** This collection of technologies fully implement a Model-View-Controller paradigm, where models encapsulate application data, views display and edit that data, and controllers mediate between the two. Read *Cocoa Bindings Programming Topics* to learn more about Cocoa Bindings.
- **Core Data.** This framework provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence. You can read about Core Data in *Core Data Programming Guide*.
- **AppleScript.** This scripting language enables direct control of scriptable apps and of many parts of macOS. Cocoa's scripting support takes advantage of key-value coding to get and set information in scriptable objects. The methods in the `NSScriptKeyValueCoding` informal protocol provide additional capabilities for working with key-value coding, including getting and setting key values by index in multi-value keys and coercing (or converting) a key-value to the appropriate data type. *AppleScript Overview* provides a high-level overview of AppleScript and its related technologies.