User Defaults Programming Topics

Data Management: Preference Settings

2007-10-31

Apple Inc.
© 2001, 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Cocoa, Mac, Mac OS, Objective-C, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction to User Defaults 7

Who Should Read This Document 7
Organization of This Document 7

Defaults Domains 9

Domains 9
Argument Domain 10
Application Domain 10
Global Domain 10
Languages Domain 10
Registration Domain 11

Using NSUserDefaults 13

Setting a Default in the NSRegistrationDomain 13
Allowing the User to Specify a Different Default Behavior 13
Using the Default Value to Determine Behavior 14
Synchronizing an NSUserDefaults Object with the Defaults Database 14
Using CFPreferences 15

User Defaults and Bindings 17

What Is NSUserDefaultsController? 17
The Shared User Defaults Controller 17
Binding to the Shared User Defaults Controller 18
initialValues Versus NSUserDefaults registerDefaults: 18
Search Order for Defaults Values 19
Programmatically Accessing NSUserDefaultsController Values 19

Storing NSColor in User Defaults 21

Extending NSUserDefaults to support NSColor 21
Establishing Bindings Between Colors and User Defaults 22

Document Revision History 25

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Figures and Listings

User Defaults and Bindings 17

Listing 1 Binding the userName defaults key to an NSTextField programmatically 18
Listing 2 Changing the initial values of the sharedUserDefaultsController instance 18

Storing NSColor in User Defaults 21

Figure 1 Establishing a binding between an `NSColor` value and user defaults 23
Listing 1 Storing an NSColor instance in user defaults 21
Listing 2 Retrieving an NSColor instance from user defaults 21
Listing 3 Contents of NSUserDefaults myColorSupport category .h file 21
Listing 4 Contents of NSUserDefaults myColorSupport category .m file 22
Listing 5 Establishing a binding between an `NSColor` property and `NSUserDefaultsController` 24

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Introduction to User Defaults

This programming topic describes the programmatic interface for interacting with the Mac OS X user preferences system—also known as the user defaults system—using Cocoa. Preference settings let you offer ways for users to customize the appearance or behavior of your software. The user defaults system lets you access and manage user preferences. You can use the defaults system to provide reasonable initial values for application settings, as well as save and retrieve the user's own preference selections across sessions.

The `NSUserDefaults` class only supports the storage of objects that can be serialized to property lists. This limitation would seem to exclude many kinds of objects, such as `NSColor` and `NSFont` objects, from the user default system. But if objects conform to the `NSCoding` protocol they can be archived to `NSData` objects, which are property list–compatible objects. For information on how to do this, see "Storing NSColor in User Defaults (page 21)"; although this article focuses on `NSColor` objects, the procedure can be applied to any object that can be archived.

`NSUserDefaults` does not currently support per-host preferences. To do this, you must use *Preferences Utilities Reference*. However, `NSUserDefaults` correctly reads per-host preferences, so you can safely mix `CFPreferences` code with `NSUserDefaults`.

Who Should Read This Document

You should read this document to understand the programmatic interface for interacting with the Mac OS X user defaults system using Cocoa.

Organization of This Document

This programming topic contains the following articles:

- "Defaults Domains" (page 9) describes the various defaults domain groupings.
- "Using NSUserDefaults" (page 13) describes how to create and save user defaults.
- User Defaults and Bindings (page 17) describes the role of `NSUserDefaultsController` and how it works with `NSUserDefaults`.
- Storing NSColor in User Defaults (page 21) describes how to store colors in an application's user defaults.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Defaults Domains

Defaults are grouped in domains. For example, there's a domain for application-specific defaults and another for system-wide defaults that apply to all applications. All defaults are stored and accessed per user. Defaults that affect all users are not provided for.

Domains

Defaults are grouped in domains. Each domain has a name by which it's identified and stores defaults as key-value pairs in an `NSDictionary` object. Each default is made up of three components:

- The domain in which the default is stored
- The name by which the default is identified (an `NSString` object)
- The default's value, which can be any property-list object (`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`)

A domain is either persistent or volatile. Persistent domains are permanent and last past the life of the user defaults object. Persistent domains are stored in a user's defaults database. If you use a user defaults object (an instance of `NSUserDefaults` or `NSUserDefaultsController`) to make a change to a default in a persistent domain, the changes are saved in the user's defaults database automatically. On the other hand, volatile domains last only as long as the user defaults object exists; they aren't saved in the user's defaults database. The standard domains are:

Domain	State
`NSArgumentDomain`	volatile
Application (Identified by the application's identifier)	persistent
`NSGlobalDomain`	persistent
Languages (Identified by the language names)	volatile
`NSRegistrationDomain`	volatile

A search for the value of a given default proceeds through the domains in an `NSUserDefaults` object's search list. Only domains in the search list are searched. The standard search list contains the domains from the table above, in the order listed. A search ends when the default is found. Thus, if multiple domains contain the same default, only the domain nearest the beginning of the search list provides the default's value.

The following sections describe the purpose of each of the domains.

 © 2001, 2007 Apple Inc. All Rights Reserved.

Argument Domain

Default values can be set from command line arguments (if you start the application from the command line) as well as from a user's defaults database. Default values set from the command line go in the NSArgumentDomain. They are set on the command line by preceding the default name with a hyphen and following it with a value. For example, the following command launches Xcode and sets Xcode's IndexOnOpen default to NO:

```
localhost> Xcode.app/Contents/MacOS/Xcode -IndexOnOpen NO
```

Defaults set from the command line temporarily override values from a user's defaults database. In the example above, Xcode won't automatically index projects even if the user's IndexOnOpen preference is set to YES in the defaults database.

Application Domain

The application domain contains application-specific defaults that are read from a user's defaults database. The application domain is identified by the bundle identifier of the application. When you save a user's defaults in your application, this is typically the domain to which the values are saved. Values themselves are stored in a file managed by the system. Currently, they are stored in $HOME/Library/Preferences/<ApplicationIdentifier>.plist; it may be useful to inspect this file for debugging purposes, but you should not modify this file directly—instead you should use the appropriate API provided by the user defaults objects.

Global Domain

The global domain contains defaults that are read from a user's defaults database and are applicable to all applications that a user runs. Many Application Kit and Foundation objects use default values from the NSGlobalDomain. For example, NSRulerView objects automatically use a user's preferred measurement units, as stored in the user's defaults database under the key AppleMeasurementUnits. Consequently, ruler views in all applications use the user's preferred measurement units—unless an application overrides the default by creating an AppleMeasurementUnits default in its application domain. Another NSGlobalDomain default, under the key AppleLanguages, allows users to specify a preference of languages as an array of strings. For example, a user could specify English as the preferred language, followed by Spanish, French, German, Italian, and Swedish.

Languages Domain

If a user has a value for the AppleLanguages default, then NSUserDefaults records language-specific default values in domains identified by the language name. The language specific domains contain defaults for a locale. Certain classes from the Foundation Framework (such as NSCalendarDate, NSDate, and NSTimeZone, NSString, and NSScanner) use locale defaults to modify their behavior. For example, when you request an NSString representation of an NSCalendarDate, the NSCalendarDate object looks at the locale to determine what the months and the days of the week are named in your preferred language.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Registration Domain

The registration domain is a set of application-provided defaults that are used unless a user overrides them (the “default defaults” or “factory settings”). For example, the first time you run Xcode, there isn’t an `IndexOnOpen` value saved in your defaults database. Consequently, Xcode registers a default value for `IndexOnOpen` in the `NSRegistrationDomain` as a “catch all” value. Xcode can thereafter assume that an `NSUserDefaults` object always has a value to return for the default, simplifying the use of user defaults.

You set `NSRegistrationDomain` defaults programmatically with the method `registerDefaults:`.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Using NSUserDefaults

Typically, you use the NSUserDefaults class by invoking the standardUserDefaults class method to get an NSUserDefaults object. This method returns a global NSUserDefaults object with a search list already initialized. Use the objectForKey: and setObject:forKey: methods to get and set default values. Note that a default's value can be only property list objects: NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary (for more on property lists, see *Property List Programming Guide*).

The following sections discuss various aspects of using the NSUserDefaults class.

Setting a Default in the NSRegistrationDomain

An application can set values for all its defaults in the NSRegistrationDomain. If users specify a different preference in their defaults database, the users' preferences override the values from the NSRegistrationDomain. An NSUserDefaults object only uses values from the NSRegistrationDomain when a user hasn't specified a different preference. So, you need to decide whether or not your application should delete backup files by default.

To register the application's default behavior, you get the application's shared instance of NSUserDefaults and register default values with it. A good place to do this is in the initialize method of the class that uses the default. The following example registers the value "YES" for the default named "DeleteBackup".

```
+ (void)initialize{

    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSDictionary *appDefaults = [NSDictionary
        dictionaryWithObject:@"YES" forKey:@"DeleteBackup"];

    [defaults registerDefaults:appDefaults];
}
```

The initialize message is sent to each class before it receives any other message, ensuring that the application's defaults are set before the application needs to read them.

Allowing the User to Specify a Different Default Behavior

To allow users to specify a different default behavior for deleting backups, you must provide an interface in which they can express their preference. Most applications provide a Preferences panel for this purpose. When your application detects that a user has specified a new preference, it should save it in the shared instance of NSUserDefaults.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

For example, assume that your application has an instance variable called *deleteBackupButton* that is an outlet to an NSButton, and that users toggle this button's state to indicate whether or not the application should delete its backup files. You could use the following code to update the user's value for the DeleteBackup default:

```
if ([deleteBackupButton state]) {
    // The user wants to delete backup files.
    [[NSUserDefaults standardUserDefaults]
        setObject:@"YES" forKey:@"DeleteBackup"];
} else {
    // The user doesn't want to delete backup files.
    [[NSUserDefaults standardUserDefaults]
        setObject:@"NO" forKey:@"DeleteBackup"];
}
```

After determining the button's state, `setObject:forKey:` is used to set the value of the specified default in the application domain.

You don't have to use a Preferences panel to manage all defaults. For example, an `NSWindow` object can store its placement in the user defaults system, so that it appears in the same location the next time the user starts the application.

Using the Default Value to Determine Behavior

To find out whether or not to delete a backup file, you can use the following statement:

```
[[NSUserDefaults standardUserDefaults] boolForKey:@"DeleteBackup"];
```

As a convenience, `NSUserDefaults` provides `boolForKey:`, `floatForKey:`, and so on. Recall that a default's value can be only an NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary. `boolForKey:` and similarly named methods attempt to get the value for the specified default and interpret it as a different data type.

Synchronizing an NSUserDefaults Object with the Defaults Database

Since other applications (and the user) can write to a defaults database, the database and an `NSUserDefaults` object might not agree on the value of a given default at all times.

You can update the defaults database with an `NSUserDefaults` object's new values and update the `NSUserDefaults` object with any changes that have been made to the database using the `synchronize` method.

On Mac OS X v10.5 and later, in applications in which a run-loop is present, `synchronize` is automatically invoked at periodic intervals. Consequently, you might synchronize before exiting a process, but otherwise you shouldn't need to.

 © 2001, 2007 Apple Inc. All Rights Reserved.

Using CFPreferences

Since CFPreferences currently has some features not yet supported in `NSUserDefaults`, you may want to use CFPreferences to perform some of your defaults operations. For example, CFPreferences supports per-host preferences, and `NSUserDefaults` currently does not.

For more information about CFPreferences see the Core Foundation Programming Topic *Preferences Programming Topics for Core Foundation*.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

User Defaults and Bindings

Many applications provide a preferences window that allows the user to customize an application's settings. NSUserDefaultsController provides a layer on top of NSUserDefaults and allows you to bind attributes of user interface items to the corresponding key in an application's user defaults.

What Is NSUserDefaultsController?

NSUserDefaultsController is a concrete subclass of NSController that implements a bindings-compatible interface to NSUserDefaults. Properties of an instance of NSUserDefaultsController are bound to user interface items to access and modify values stored using NSUserDefaults.

NSUserDefaultsController is typically used when implementing your application's preference window interface, or when you can bind a user interface item directly to a default value. NSUserDefaults remains the primary programmatic interface to your application's default values for the rest of your application.

By default NSUserDefaultsController immediately applies any changes made to its properties. It can be configured so that changes are not applied until it receives an `applyChanges:` message, allowing the preferences dialog to support an Apply button. NSUserDefaultsController also supports reverting to the last applied set of values, using the `revert:` method.

NSUserDefaultsController also allows you to provide a dictionary of factory defaults that can be used to reset the user configurable values for your application, usually done in response to a user clicking a Revert to Factory Defaults button.

The Shared User Defaults Controller

NSUserDefaultsController provides a shared instance of itself via the class method `sharedUserDefaultsController`. This shared instance uses the NSUserDefaults instance returned by the method `standardUserDefaults` as its model, has no initial values, and immediately applies changes made through its bindings.

Care must be taken that changes to the settings of the shared user defaults controller are made before any nib files containing bindings to the shared controller are loaded. To ensure that these changes are made before any nib files are loaded, they are often implemented in the `initialize` class method of the application delegate, or in your preferences window controller.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Binding to the Shared User Defaults Controller

The shared NSUserDefaultsController is always available as a bindable controller in the Bindings Info window in Interface Builder. When establishing a binding to a user default, set the Controller Key to `values`, and the Model Key Path to the key of the default.

Creating bindings programmatically requires that you retrieve the shared user defaults controller using the NSUserDefaultsController class method `sharedUserDefaultsController`. You then provide that object as the *`observableController`* to the `bind:toObject:withKeyPath:options:` method.

The example in Listing 1 establishes a binding between an NSTextField (`theTextField`) and the `userName` default using the shared user defaults controller.

Listing 1 Binding the userName defaults key to an NSTextField programmatically

```
 [theTextField bind:@"value"
             toObject:[NSUserDefaultsController sharedUserDefaultsController]
          withKeyPath:@"values.userName"
              options:[NSDictionary dictionaryWithObject:[NSNumber
numberWithBool:YES]

forKey:@"NSContinuouslyUpdatesValue"]];
```

initialValues Versus NSUserDefaults registerDefaults:

The initial values dictionary allows you to provide a means to reset the user configurable default values to the factory defaults. Typically these values represent a subset of the defaults that your application registers using the NSUserDefaults method `registerDefaults:`.

Calling the NSUserDefaultsController method `setInitialValues:` should not be considered a replacement for registering your application's preference defaults using NSUserDefault's `registerDefaults:` method.

The example in Listing 2 loads the default values from a file in the application wrapper, registers those values with NSUserDefaults, and then registers a subset of the values as the initial values of the shared user defaults controller. The `setupDefaults` method would be called from your application delegate's `initialize` class method.

Listing 2 Changing the initial values of the sharedUserDefaultsController instance

```
+ (void)setupDefaults
{
    NSString *userDefaultsValuesPath;
    NSDictionary *userDefaultsValuesDict;
    NSDictionary *initialValuesDict;
    NSArray *resettableUserDefaultsKeys;

    // load the default values for the user defaults
   userDefaultsValuesPath=[[NSBundle mainBundle] pathForResource:@"UserDefaults"
                               ofType:@"plist"];
    userDefaultsValuesDict=[NSDictionary
dictionaryWithContentsOfFile:userDefaultsValuesPath];
```


2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

```
    // set them in the standard user defaults
    [[NSUserDefaults standardUserDefaults]
registerDefaults:userDefaultsValuesDict];

    // if your application supports resetting a subset of the defaults to
    // factory values, you should set those values
    // in the shared user defaults controller
    resettableUserDefaultsKeys=[NSArray
arrayWithObjects:@"Value1",@"Value2",@"Value3",nil];
    initialValuesDict=[userDefaultsValuesDict
dictionaryWithValuesForKeys:resettableUserDefaultsKeys];

    // Set the initial values in the shared user defaults controller
    [[NSUserDefaultsController sharedUserDefaultsController]
setInitialValues:initialValuesDict];
}
```

Search Order for Defaults Values

When a method that is key-value coding compliant attempts to get a value for a key from an NSUserDefaultsController the following search pattern is used:

1. The value of a corresponding key in `values`
2. The value of a corresponding key in the NSUserDefaults instance returned by the NSUserDefaultsController method `defaults`.
3. The value of a corresponding key in the initial values dictionary

If no corresponding value is found, `nil` is returned.

The search path is somewhat different when you retrieve the result directly from the NSUserDefaults instance associated with the NSUserDefaultsController. In that case, any unapplied values in the NSUserDefaultsController, as well as the values in the initial values dictionary are ignored.

Programmatically Accessing NSUserDefaultsController Values

Although NSUserDefaults should remain your primary programmatic interface to the user defaults, some circumstances require that you get and set the default values contained in an NSUserDefaultsController instance directly. For example, when implementing portions of your preferences window that don't directly interact with an existing binding, such as setting a font or choosing a directory path.

The NSUserDefaultsController method `values` returns a KVC-compliant object that is used to access these default values. To get the value of a default, use the `valueForKey:` method.

```
[[theDefaultsController values] valueForKey:@"userName"];
```

Similarly, to set a value for a default, use `setValue:forKey:`.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

```
[[theDefaultsController values] setValue:newUserName
                                   forKey:@"userName"];
```

The NSUserDefaultsController automatically provides notification of the value change to any established bindings for that key path.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Storing NSColor in User Defaults

It is often desirable to store the value of an NSColor instance in an application's user defaults. However, NSUserDefaults only supports the storage of objects that can be represented in an property list.

The solution is to use object archiving to write the NSColor instance data to an NSData instance and then store that as the default as shown in Listing 1. This is often done in an application life-cycle exit point such as the `applicationShouldTerminate:` delegation method.

Listing 1 Storing an NSColor instance in user defaults

```
// store the value in aColor in user defaults
// as the value for key aKey
NSData *theData=[NSArchiver archivedDataWithRootObject:aColor];
[[NSUserDefaults standardUserDefaults] setObject:theData forKey:aKey];
```

To read the value back from NSUserDefaults an application retrieves the NSData instance for the required key and unarchives the NSColor instance. The example in Listing 2 demonstrates retrieving the color. This is often done in an application life-cycle entry point such as `awakeFromNib`.

Listing 2 Retrieving an NSColor instance from user defaults

```
// read the value of the user default with key aKey
// and return it in aColor
NSColor * aColor =nil;
NSData *theData=[[NSUserDefaults standardUserDefaults] dataForKey:aKey];
if (theData != nil)
    aColor =(NSColor *)[NSUnarchiver unarchiveObjectWithData:theData];
```

Extending NSUserDefaults to support NSColor

It's possible to take advantage of the support for categories in Objective-C to add NSColor support to the existing NSUserDefaults class, without subclassing.

The example code in Listing 3 and Listing 4 shows an implementation of such a category. The method `setColor:forKey:` in archives the specified color to an NSData instance and stores it in the user defaults using the specified key. The method `colorForKey:` retrieves the NSData instance specified by the key, and then unarchives an instance of NSColor using the data.

Listing 3 Contents of NSUserDefaults myColorSupport category .h file

```
#import <Foundation/Foundation.h>

@interface NSUserDefaults(myColorSupport)
- (void)setColor:(NSColor *)aColor forKey:(NSString *)aKey;
- (NSColor *)colorForKey:(NSString *)aKey;
@end
```


2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Listing 4 Contents of NSUserDefaults myColorSupport category .m file

```
#import "NSUserDefaults+myColorSupport.h"

@implementation NSUserDefaults(myColorSupport)

- (void)setColor:(NSColor *)aColor forKey:(NSString *)aKey
{
    NSData *theData=[NSArchiver archivedDataWithRootObject:aColor];
    [self setObject:theData forKey:aKey];
}

- (NSColor *)colorForKey:(NSString *)aKey
{
    NSColor *theColor=nil;
    NSData *theData=[self dataForKey:aKey];
    if (theData != nil)
        theColor=(NSColor *)[NSUnarchiver unarchiveObjectWithData:theData];
    return theColor;
}

@end
```

Important: There is some risk in implementing a category with method names that are common enough that Apple could use them in the future. An alternative would be to use prefixes that Apple would not use, for example, `my_colorForKey:`.

Establishing Bindings Between Colors and User Defaults

You can easily establish a binding between a user-interface object whose value is a color (that is, an `NSColor` object) and user defaults. When the user chooses a color preference for something in an application, the binding preserves and restores the preference across successive launches of the application.

To effect the binding, use a ready-made instance of the `NSUnarchiveFromDataTransformerName` value transformer in Interface Builder. An `NSValueTransformer` object converts an object value typically in two directions: between the form in which it is displayed and the form in which it is stored. The `NSUnarchiveFromDataTransformerName` value transformer works by archiving an `NSColor` object in an `NSData` object and then, on the other side of the binding, unarchiving the color object from the data object. For this value transformation to work, the archived object must implement the `NSCoding` protocol using sequential archiving—which `NSColor` does.

An `NSColorWell` instance is a user-interface object whose value is a `NSColor` object. You can drag the color-well object from the Controls palette of Interface Builder onto a view. To establish the binding between this object and user defaults, complete the following steps:

1. With the color well still selected, open the Bindings pane of the Inspector and expose the **value** binding.
2. From the “Bind to” pop-up menu choose Shared User Defaults.

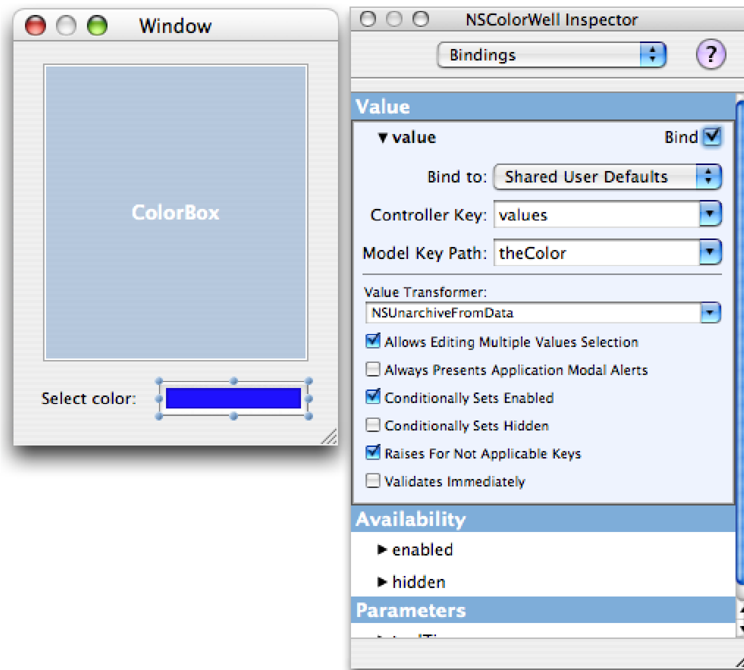
   This action adds an instance of `NSUserDefaultsController` (“Shared Defaults”) to the nib file window.
3. Keep the Controller Key field as `values` but in the Model Key Path field specify a name under which to save the color object (`theColor`, in this example).

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

4. From the Value Transformer combo box select (or enter) `NSUnarchiveFromData`.

When you're finished, your setup in Interface Builder should look similar to that in Figure 1.

Figure 1 Establishing a binding between an `NSColor` value and user defaults



If at this point you save your nib file and build your project, you can launch the application, change the color in the color well, quit the application, and then relaunch. The color in the color well is what it was when you last changed it.

Although the foregoing procedure establishes a binding between an `NSColor` value of a view and user defaults, it does not propagate changes in that value to other objects in the application. You can do that by explicitly setting the color to the restored default when the application launches and, thereafter, by having the first responder handle the `changeColor:` message whenever the user changes the color. But you can also use bindings so that any change in color value is propagated both to user defaults and applied to a custom view in the application. This requires you to complete the following steps:

- Declare an `NSColor` property of the custom view class.
- Expose this property as a binding (`exposeBinding:`); do this in the class method `initialize`.
- In the setter method for the property, send `setNeedsDisplay:` (or `setNeedsDisplayInRect:`) to `self` after the new color is retained; this forces the view to redraw itself in the new color.
- Define a controller object that acts as application delegate. When the application finishes launching, this object establishes a binding between the custom view's `NSColor` property and the property of the `NSUserDefaultsController` object bound to the color well.

  See Listing 5 for an example of this final step.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Listing 5 Establishing a binding between an `NSColor` property and `NSUserDefaultsController`

```
@implementation AppDelegate
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    [theColorBox bind:@"backgroundColor" toObject:[NSUserDefaultsController
sharedUserDefaultsController]
        withKeyPath:@"values.theColor"
        options:[NSDictionary dictionaryWithObject:NSUnarchiveFromDataTransformerName
forKey:NSValueTransformerNameBindingOption]];
}
@end
```


2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *User Defaults Programming Topics*.

Date	Notes
2007-10-31	Updated information about periodic autosave behavior.
2007-01-08	Corrected typos and capitalization mistakes.
2006-11-07	Added overview of procedure for storing non-property-list objects in user defaults, and linked to related article.
2006-09-05	Made small additions to the content. Changed title from "User Defaults."
	Expanded explanation of user defaults in introduction.
	Noted requirement that a default's value must be a property list value at the beginning of the Using Defaults (page 13) article.
2005-08-11	Included an article that describes the use of NSUserDefaultsController. Corrected minor typographical errors.
2004-02-03	Added article Storing NSColor in User Defaults (page 21).
2003-05-09	Linked to the Core Foundation Preferences Programming Topic, which was also incorrectly named.
2003-01-13	Added link in limitations area to CFPreferences. Corrected class name in Defaults Domains Concept.
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.

2007-10-31 | © 2001, 2007 Apple Inc. All Rights Reserved.