# Value Transformer Programming Guide

**Data Management**

**2007-04-06**

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



Simultaneously published in the United States and Canada.

# Contents

# Listings

# Introduction to Value Transformers

*Value Transformers* describes the `NSValueTransformer` class, the built-in value transformers and how to write your own subclasses. Value transformers are primarily used by Controller Layer bindings.

## Who Should Read This Document

You should read this document to gain an understanding of the use of value transformers in applications that use Cocoa bindings. You are expected to be familiar with the basics of Cocoa development, including the Objective-C language and memory management.

## Organization of This Document

This programming topic includes the following articles:

- "Role of Value Transformers" (page 9) provides an overview of value transformers.
- "Available Value Transformers" (page 11) describes the value transformers provided by the Foundation framework.
- "Writing a Custom Value Transformer" (page 15) describes how to implement a value transformer subclass.
- "Registering a Value Transformer" (page 13) describes how to register your own value transformers.

# Role of Value Transformers

Value transformer classes are used to transform the value of an object in some manner. This is particularly useful when using Cocoa bindings and creating a binding between a model property of a controller and a user interface element, or another controller object. By using the built-in value transformers, or creating custom value transformers, you can further reduce the amount of glue code required by your application.

For example, it's often necessary to disable a user interface element if a model property is a `nil` value. Instead of writing a method that returns `YES` if the property is `nil`, you specify that the binding use a "is not nil" transformer. The transformer acts as the "middleman", providing a `YES` value to the user interface element if the property is `nil`.

Value transformation is done immediately before a value is passed to a user interface element's `setObjectValue:` method. Likewise, the reverse transformation is applied before the value in the user interface is set in the model. See Bindings Message Flow in *Cocoa Bindings Programming Topics* for a detailed description of when value transformers are applied in the context of Cocoa bindings.

All value transformers are subclasses of `NSValueTransformer`. In addition to providing the abstract methods for subclasses, the `NSValueTransformer` class maintains a mapping of value transformer names and the corresponding value transformer objects. This name is used in Interface Builder to specify the value transformer that is used for a binding. You register instances of your custom value transformers in order to expose them, allowing them to be used by Cocoa bindings in Interface Builder.

A value transformer can be reversible, able to convert a value to a new value, and back again. A reversible transformer can be thought of as "read-write", it transforms the original property value, but will also return any changes made to the transformed value. A non-reversible transformer is "read-only", only able to reflect changes in the original property.

# Available Value Transformers

In addition to providing a mechanism for registering your own value transformers, there are several built-in transformers provided by `NSValueTransformer`.

The built-in transformers provide facilities for negating boolean values, testing for `nil` or non `nil` values, and archiving and unarchiving values into `NSData` instances.

## NSNegateBooleanTransformerName

The `NSNegateBooleanTransformerName` value transformer returns an instance of `NSNumber` containing a boolean value. The returned value is the boolean negation of the original value and is reversible.

This value transformer is useful in enabling or disabling user interface elements, as well as setting the values of checkboxes and radio buttons.

## NSIsNilTransformerName

The `NSIsNilTransformerName` value transformer returns an instance of `NSNumber` containing a boolean value. The returned value is `YES` if the original value is `nil`, otherwise the returned value is `NO`. This value transformer is not reversible.

This value transformer is often used to enable or disable user interface elements.

## NSIsNotNilTransformerName

The `NSIsNotNilTransformerName` value transformer returns an instance of `NSNumber` containing a boolean value. The returned value is `YES` if the original value is not `nil`, otherwise the returned value is `NO`. This value transformer is not reversible.

This value transformer is often used to enable or disable user interface elements.

## NSUnarchiveFromDataTransformerName

The `NSUnarchiveFromDataTransformerName` transformer returns an object created by attempting to unarchive the data in the `NSData` object passed as the value. The reverse transformation returns an `NSData` instance created by archiving the value.

An object must implement the `NSCoding` protocol using sequential archiving in order to be unarchived and archived with this transformer.

This transformer is primarily used with instances of `NSUserDefaultsController`. This transformer allows your application to store objects in the user defaults that are not natively supported, for example, `NSColor` objects.

## NSKeyedUnarchiveFromDataTransformerName

The `NSKeyedUnarchiveFromDataTransformerName` transformer returns an object created by attempting to unarchive the data in the `NSData` object passed as the value. The reverse transformation returns an `NSData` instance created by archiving the value.

This transformer differs from the `NSUnarchiveFromDataTransformerName` transformer in that the object must implement the `NSCoding` protocol using *keyed archiving*, rather than sequential archiving..

This transformer is primarily used with instances of `NSUserDefaultsController`. This transformer allows your application to store objects in the user defaults that are not natively supported, for example, `NSColor` objects.

# Registering a Value Transformer

When creating bindings in Interface Builder, you can specify the name of a value transformer that is used as a "middleman". In order for your custom value transformer objects to be used in this manner, they must first be registered by name.

## Registering a Custom Value Transformer

The `NSValueTransformer` class maintains a mapping of value transformer names, and the corresponding value transformer object. Rather than registering a subclass, individual instances of the `NSValueTransformer` subclasses are registered. This allows a value transformer that provides a generalized functionality to be registered multiple times, with different parameters, for different names. For example, you could write a MultiplicationTransformer and specify the number that is used as the multiplier when an instance is initialized. Separate instances could be registered as "MultiplyByTwoTransformer", "MultiplyByTenTransformer", and so on.

The example in Listing 1 registers an instances of the Fahrenheit to Celsius value transformer created in "Writing a Custom Value Transformer" (page 15) using the name "FahrenheitToCelsiusTransformer".

**Listing 1** Registering the Fahrenheit to Celsius value transformer

```
FahrenheitToCelsiusTransformer *fToCTransformer;

// create an autoreleased instance of our value transformer
fToCTransformer = [[[FahrenheitToCelsiusTransformer alloc] init]
                                                    autorelease];

// register it with the name that we refer to it with
[NSValueTransformer setValueTransformer:fToCTransformer
                                forName:@"FahrenheitToCelsiusTransformer"];
```

Value transformers are typically registered by an application's delegate class, in response to receiving a `initialize:` class message. This allows registration to occur early in the application startup process, providing access to the value transformers as nib files load.

## Availability in Interface Builder

Your `NSValueTransformer` subclasses are not automatically listed in the Interface Builder bindings inspector. When inspecting a binding you can enter the name that the value transformer is registered with, but the functionality will not be present in Interface Builder's test mode. When your application is compiled and run the transformer will be used.

# Writing a Custom Value Transformer

The Foundation framework provides several built-in value transformers. You create your own custom value transformers by subclassing `NSValueTransformer`.

An `NSValueTransformer` subclass must, at a minimum, implement the `transformedValueClass`, `allowsReverseTransformation` and `transformedValue:` methods. If your custom value transformer supports reverse transformations, you must also implement the `reverseTransformedValue:` method.

As an example, we'll create an `NSValueTransformer` subclass, `FahrenheitToCelsiusTransformer`, that converts Fahrenheit temperatures to the Celsius scale. This value transformer is also reversible, able to convert Celsius temperatures back to the Fahrenheit scale.

## Declaring the Returned Value Class

A value transformer subclass must implement the `transformedValueClass` class method. This method returns the class of the object that the `transformedValue:` method returns.

The `FahrenheitToCelsiusTransformer` class returns an NSNumber, as shown in Listing 1.

**Listing 1** Fahrenheit to Celsius `transformedValueClass` implementation

```
+ (Class)transformedValueClass
{
    return [NSNumber class];
}
```

## Allowing Reverse Transformations

`NSValueTransformer` subclasses must also implement the `allowsReverseTransformation` class method. The subclass implementation should return `YES` if the value transformer is reversible.

The Fahrenheit to Celsius value transformer is reversible, so the `allowsReverseTransformation` implementation returns `YES`, as shown in Listing 2.

**Listing 2** Fahrenheit to Celsius `allowsReverseTransformation` implementation

```
+ (BOOL)allowsReverseTransformation
{
    return YES;
}
```

# Transforming a Value

The `transformedValue:` method implements the actual value transformation. It's passed the object to transform, and returns the result of the transformation. The result must be an instance of the class returned by `transformedValueClass`.

For maximum flexibility, an implementation of `transformedValue:` should be prepared to handle a variety of different classes as the value. The Fahrenheit to Celsius transformer can handle values of both `NSString` and `NSNumber` classes, by using the `doubleValue` method to convert the value to a scalar.

The result that is returned when the value is `nil` is dependent on what the value transformer is attempting to do. The Fahrenheit to Celsius implementation of `transformedValue:`, shown in Listing 3, returns `nil` in this case.

**Listing 3** Fahrenheit to Celsius `transformedValue` implementation

```
- (id)transformedValue:(id)value
{
    float fahrenheitInputValue;
    float celsiusOutputValue;

    if (value == nil) return nil;

    // Attempt to get a reasonable value from the
    // value object.
    if ([value respondsToSelector: @selector(floatValue)]) {
    // handles NSString and NSNumber
        fahrenheitInputValue = [value floatValue];
    } else {
        [NSException raise: NSInternalInconsistencyException
                    format: @"Value (%@) does not respond to -floatValue.",
        [value class]];
    }

    // calculate Celsius value
    celsiusOutputValue = (5.0/9.0)*(fahrenheitInputValue - 32.0);

    return [NSNumber numberWithFloat: celsiusOutputValue];
}
```

# Reverse Transforming a Value

If an `NSValueTransformer` subclass supports reverse transformations, it must implement the `reverseTransformedValue:` method.

Care should be taken when implementing reversible value transformers to ensure that the reversal does not result in a loss of accuracy. In many cases, passing the result of `transformedValue:` to `reverseTransformedValue:` should return an object with the same value as the original object.

The Fahrenheit to Celsius implementation of `reverseTransformedValue:` is shown in Listing 4. The only significant difference between this and the `transformedValue:` implementation is the temperature conversion formula.

**Listing 4** Fahrenheit to Celsius `reverseTransformedValue` implementation

```
- (id)reverseTransformedValue:(id)value
{
    float celsiusInputValue;
    float fahrenheitOutputValue;

    if (value == nil) return nil;

    // Attempt to get a reasonable value from the
    // value object.
    if ([value respondsToSelector: @selector(floatValue)]) {
    // handles NSString and NSNumber
        celsiusInputValue = [value floatValue];
    } else {
        [NSException raise: NSInternalInconsistencyException
                    format: @"Value (%@) does not respond to -floatValue.",
        [value class]];
    }

    // calculate Fahrenheit value
    fahrenheitOutputValue = ((9.0/5.0) * celsiusInputValue) + 32.0;

    return [NSNumber numberWithDouble: fahrenheitOutputValue];
}
```

# Document Revision History

This table describes the changes to *Value Transformer Programming Guide*.

| Date | Notes |
| --- | --- |
| 2007-04-06 | Added information on new keyed unarchiving transformer. |
| 2007-02-08 | Corrected typos. |
| 2006-03-08 | Clarified the role of the transformedValueClass method. |
| 2005-08-11 | Noted that NSUnarchiveFromData expects objects to provide serial archiving support. |
| 2004-08-31 | Corrected minor typos in "Role of Value Transformers" (page 9). |
| | Clarified the developer's responsibility when using `NSUnarchiveFromDataTransformerName` in "Available Value Transformers" (page 11). |
| 2004-03-30 | Minor edits to the source code in "Writing a Custom Value Transformer" (page 15). |
| 2004-03-10 | Revised suggested value transformer initialization method in "Registering a Value Transformer" (page 13). |
| 2003-08-18 | Initial publication of *Value Transformers*. |