

**mitp**

Ferhat Ayaz • Jörn Hameister • Erik Meißner

# Cocoa Kochbuch

Zahlreiche Rezepte und Beispiele  
für den sofortigen Einsatz

Applikationen, Frameworks, XML,  
Internet, Kommunikation, Agenten

Persistenz, Xcode Hacks, PDF-Export,  
Drawings

# Die Image Browser View

In diesem Kapitel wird die `IKImageBrowserView` des `ImageKit` beschrieben. Dazu werden wir eine Applikation `ImageBrowserViewDemo` entwickeln, mit der Bilder in Form von Thumbnails angezeigt und nach verschiedenen Arten gruppiert werden können. In der Anwendung werden die importierten Bilder exemplarisch anhand des Anfangsbuchstabens des Dateinamens gruppiert.

In der Beispielapplikation werden folgende Themen vorgestellt:

- Drag&Drop von Bildern in die `IKImageBrowserView`
- das Zoomen von Bildern
- das Entfernen von Bildern aus der `IKImageBrowserView`
- das Gruppieren von Bildern nach definierten Kriterien
- das Umschalten von Gruppierungsarten
- das Setzen von Farben
- das Ein- und Ausschalten von Bildzusatzinformationen
- das Brennen von CDs und DVDs

Die fertige Beispielapplikation wird beide Gruppierungsarten unterstützen, die von der `IKImageBrowserView` angeboten werden. Abbildung 2.1 zeigt die Gruppierungsart `IKGroupBezelStyle`. Bei dieser Gruppierungsart wird jede Gruppe von einer Art Rahmen umrundet, der farbig hinterlegt ist und die Bilder einer Gruppe zusammenfasst. Durch einen Doppelklick auf das Gruppierungssymbol, in dem Beispiel der Buchstabe, lässt sich eine Gruppe ein- und ausklappen.

Abbildung 2.2 zeigt die Gruppierungsart `IKGroupDisclosureStyle`. Zwischen den Gruppierungsarten wird in der Beispielapplikation über eine `NSCheckBox` gewechselt. Auf der rechten Seite befindet sich ein `NSSlider`, über den der Zoom-Faktor geregelt werden kann.

Die Gruppierungsart `IKGroupDisclosureStyle` fasst die Bilder einer Gruppe unter einem Dreieck zusammen, das sich ein- und ausklappen lässt.

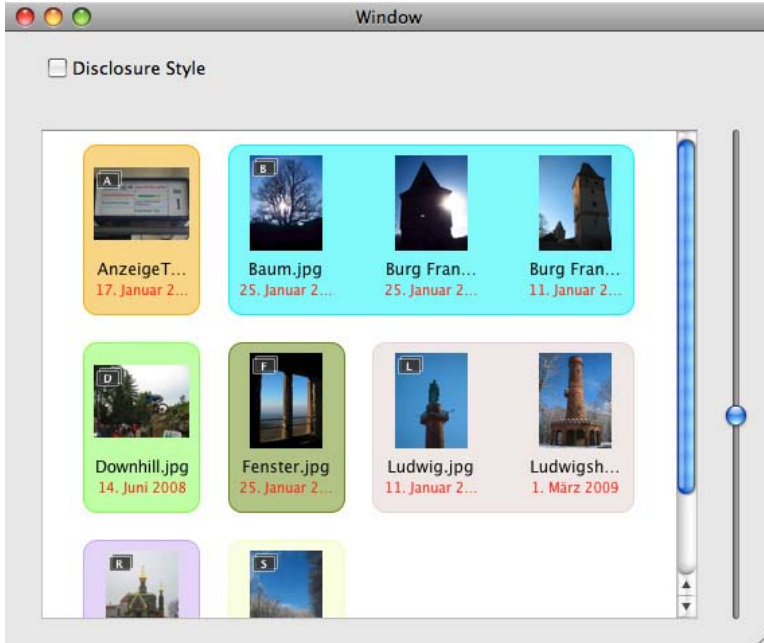


Abb. 2.1: Gruppierungsart Bezel Style

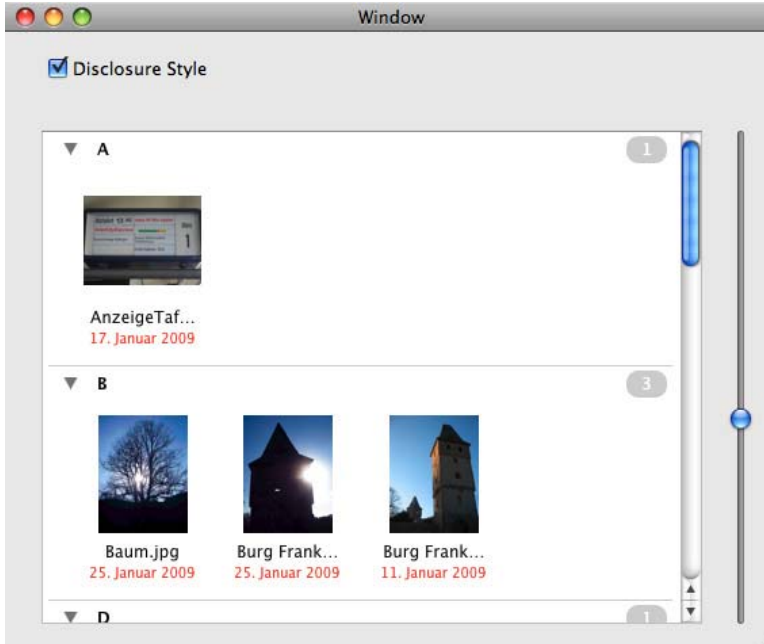


Abb. 2.2: Gruppierungsart Disclosure Style

Das `ImageKit` enthält eine Reihe weiterer interessanter Features, die es sich anzusehen lohnt. Im `ImageKit` finden sich eine ganze Reihe von Möglichkeiten mit Bildern zu arbeiten, das heißt, Bilder zu laden, zu editieren, anzuzeigen und zu speichern. Viele Features und Darstellungsmöglichkeiten, die man von `iPhoto` kennt, sind mit diesem Framework implementiert worden.

Ein sehr guter Einstiegspunkt ist das Dokument *Image Kit Programming Guide*, das unter diesem Titel auf der Apple-Seite zu finden ist.

In den Rezepten dieses Kapitels wird nur eine Funktionalität des `ImageKits` vorgestellt, die sich auf das Gruppieren von Bildern bezieht.

Außerdem werden in diesem Kapitel das `DiscRecording.framework` und das `DiscRecordingUI.framework` behandelt. Mit diesen beiden Frameworks lassen sich Daten auf CDs und DVDs brennen.

## 2.1 Image Browser View-Basisrezept

In diesem Basisrezept erklären wir, wie die `IKImageBrowserView` verwendet werden kann, um eine Vorschau von Bildern zu ermöglichen. In diesem Basisrezept werden die anzuzeigenden Bilder mittels `Drag&Drop` in die View gezogen. Um ein solches Verhalten zu implementieren, bietet das Quartz-Framework das `ImageKit`.

Damit die `IKImageBrowserView` verwendet werden kann, muss das Quartz-Framework in das Projekt integriert werden.

Als Erstes wird eine neue Cocoa-Applikation erstellt. Dann wird in den Ordner `OTHER FRAMEWORKS` mit einem Rechtsklick über `ADD` das Framework hinzugefügt. Das Quartz-Framework ist im Ordner `System/Library` zu finden. Es werden die Pakete `Quartz.framework` und `QuartzCore.framework` selektiert und hinzugefügt.

Die Vorschaubilder werden in Objekten vom Typ `ImageItem` verwaltet. Diese Objekte speichern verschiedene Informationen zu jedem Bild. Beispielsweise den Pfad zu der Bilddatei, den Namen des Bildes, seine Größe und das Datum der Datei.

Die Header-Datei sieht folgendermaßen aus:

```
#import <Cocoa/Cocoa.h>
#import <Quartz/Quartz.h>

@interface ImageItem : NSObject {
    NSString *imagePath;
    NSString *filename;
}
```

```

    NSNumber *filesize;
    NSDate *filedate;
}

@property (readwrite, copy) NSString *imagePath;
@property (readwrite, copy) NSString *filename;
@property (readwrite, retain) NSNumber *filesize;
@property (readwrite, retain) NSDate *filedate;

@end

```

Die Klasse selbst implementiert die definierten Methoden und das Protokoll von `IKImageBrowserItem`. Dieses Protokoll dient dazu, dass die eigentliche Image View mit dem Bildobjekt umgehen kann und es richtig angezeigt wird.

Damit die Bilder diesem Protokoll entsprechen, müssen folgende Methoden implementiert werden:

- (NSString \*) imageRepresentationType
- (id) imageRepresentation
- (NSString \*) imageUID

Die erste Methode liefert einen Repräsentationstyp zurück. In dem Beispiel ist dies der Typ `IKImageBrowserPathRepresentationType`, der angibt, dass das Bild über einen Dateipfad in die Anwendung gelangt.

Die zweite Methode liefert die Repräsentation zurück. Für `IKImageBrowserPathRepresentationType` ist das der Dateipfad zu dem Bild.

Die dritte Methode sorgt dafür, dass eine eindeutige UID zurückgeliefert wird. In der Beispiellapplikation wird der Dateipfad verwendet.

Die Klasse, die das Protokoll implementiert, sieht wie folgt aus:

```

#import "ImageItem.h"

@implementation ImageItem
@synthesize imagePath;
@synthesize filename;
@synthesize filedate;
@synthesize filesize;

- (id) initWithPath:(NSString*)newPath
{
    self = [super init];
    if (self != nil) {

```

```
[self setImagePath:newPath];

    self.filename = [newPath lastPathComponent]
}
return self;
}

- (void) dealloc
{
    self.imagePath = nil;
    self.filename = nil;
    self.filedate = nil;
    self.filesize = nil;
    [super dealloc];
}

// Bereitstellen der fuer ein Bild benoetigten
// Informationen
// - imageUID
// - imageRepresentationType
// - imageRepresentation
- (NSString *) imageRepresentationType
{
    return IKImageBrowserPathRepresentationType;
}

- (id) imageRepresentation
{
    return imagePath;
}

- (NSString *) imageUID
{
    return imagePath;
}

@end
```

Neben dem Protokoll wird eine `init`-Methode und eine `dealloc`-Methode zur Verfügung gestellt.

In der Klasse `ImageBrowserViewController` befindet sich die Implementierung für den Controller. Diese Klasse erweitert `NSWindowController`.

**Hinweis**

Xcode bietet die Möglichkeit, eine solche Klasse beim Erstellen direkt anzulegen. Es muss dann *Objective-C NSWindowController Class* ausgewählt werden.

Die Header-Datei der Klasse definiert ein `IBOutlet` für den `ImageBrowser` und ein `NSMutableArray` für die anzuzeigenden Bilder.

```
#import <Cocoa/Cocoa.h>
#import <Quartz/Quartz.h>
#import "ImageItem.h"

@interface ImageBrowserViewController : NSWindowController
{
    IBOutlet id imageBrowserView;
    NSMutableArray *importedImages;
}
@end
```

In der Klasse selbst findet die Initialisierung des Arrays in der Methode `init` statt. Außerdem wird das Delegate für die Drag&Drop-Funktionalität in der Methode `awakeFromNib` gesetzt.

```
- (id)init {
    self = [super init];
    if (self != nil) {
        importedImages = [[NSMutableArray alloc] init];
    }
    return self;
}

- (void) awakeFromNib
{
    [imageBrowserView setAnimates:YES];

    // Unterstütze Drag and Drop
    [imageBrowserView setDraggingDestinationDelegate:self];
}
```

Das Freigeben des Speichers wird in der `dealloc`-Methode durchgeführt.

```
- (void) dealloc
{
    [importedImages release];
}
```

```
[super dealloc];
}
```

Damit Drag&Drop funktioniert, muss das entsprechende Protokoll zu der Klasse hinzugefügt werden. Beschrieben wird dieses informelle Protokoll in dem Dokument *NSDraggingDestination Protocol Reference*. Es sind folgende drei Methoden zu implementieren.

- (NSDragOperation)draggingEntered:(id <NSDraggingInfo>)sender
- (NSDragOperation)draggingUpdated:(id <NSDraggingInfo>)sender
- (BOOL) performDragOperation:(id <NSDraggingInfo>)sender

Die ersten beiden Methoden liefern die Drag-Operation zurück. Die wichtigsten Operationen sind Kopieren, Löschen und Verschieben. Weitere Operationen sind in der Dokumentation zu dem Protokoll zu finden und sollen hier nicht weiter behandelt werden.

```
//
// Drag & Drop
//
// NSDraggingDestination Protocol Reference
//

- (BOOL) performDragOperation:(id <NSDraggingInfo>)sender
{
    NSArray *dragAndDropFiles = [[sender draggingPasteboard]
                                  propertyListForType:NSFileNamesPboardType];
    for (NSString *filePath in dragAndDropFiles) {
        BOOL isADirectory = NO;
        if([[NSFileManager defaultManager]
            fileExistsAtPath:filePath
            isDirectory:&isADirectory] ] && !isADirectory) {
            ImageItem *p =
                [[ImageItem alloc] initWithPath:filePath];
            [importedImages addObject:p];
            [p release];
        }
    }
    [imageView reloadData];

    return YES;
}
```

```

- (NSDragOperation)draggingEntered:(id <NSDraggingInfo>)sender
{
    return NSDragOperationCopy;
}

- (NSDragOperation)draggingUpdated:(id <NSDraggingInfo>)sender
{
    return NSDragOperationCopy;
}

```

In der Methode `performDragOperation` werden zuerst die Pfade zu den Bildern vom Pasteboard geholt und dann für jeden Pfad, sofern es sich um kein Verzeichnis handelt, ein Objekt vom Typ `ImageItem` erzeugt und im Array `importedImages` abgelegt. Danach wird an den `ImageBrowser` eine Nachricht geschickt, dass sich die Daten geändert haben. Dieser weiß dadurch, dass er sich neu zeichnen muss.

Die Methoden `draggingEntered` und `draggingUpdated` dienen dazu, dem Sender mitzuteilen, dass er eine Kopie von dem gedraggten Objekt machen soll. Dies wird durch die Konstante `NSDragOperationCopy` erreicht.

### Wichtig

In dem Beispiel wurde von dem anzuzeigenden Bild eine Kopie erstellt. In dem Dokument *NSDraggingInfo Protocol Reference* sind unter *Dragging Operations* weitere Schlüsselwörter definiert, die beispielsweise ein Verschieben (`NSDragOperationMove`) oder Verlinken (`NSDragOperationLink`) ermöglichen.

Als Letztes müssen noch zwei Methoden des *IKImageBrowserDataSource Protocols* implementiert werden, die dafür sorgen, dass der `ImageBrowser` die Anzahl der Bilder kennt und weiß, welches Bild an welcher Position angezeigt werden soll.

```

//
// IKImageBrowserDataSource Protocol Reference
//
// Bereitstellen von Informationen ueber Items
// (Required)
// - numberOfItemsInImageBrowser:
// - imageBrowser:itemAtIndex:
//
- (int) numberOfItemsInImageBrowser:(IKImageBrowserView *) view
{
    return [importedImages count];
}

```

```
}  
  
- (id) imageBrowser:(IKImageBrowserView *) view itemAtIndex:(int) index  
{  
    return [importedImages objectAtIndex:index];  
}
```

Die Methode `numberOfItemsInImageBrowser` bestimmt die Anzahl der Bilder im Array `importedImages` und gibt diese Zahl zurück. Bei der Methode `imageBrowser:itemAtIndex:` wird überprüft, welches Bild sich an welcher Position befindet. In der Beispielanwendung reicht es aus, das entsprechende Bild direkt aus dem Array zu holen und an den Sender der Nachricht zurückzugeben.

Nach dem Speichern und Kompilieren kann die `ImageBrowserViewDemo` gestartet werden. Wenn man nun Bilddateien aus dem Finder in die View zieht, werden diese als Vorschaubilder angezeigt. Mit einem Doppelklick lassen sie sich in einer externen Applikation öffnen.

## Hinweis

Ein sehr nützliches Dokument zum Thema *ImageKit* ist unter dem Namen *Image Kit Programming Guide* auf der Apple-Seite zu finden.

## 2.2 Zoomen

In diesem Rezept erklären wir, wie die `ImageBrowserViewDemo` um eine Zoom-Funktionalität erweitert wird. Abbildung 2.3 zeigt links ein Fenster, in dem der Zoomfaktor sehr klein gewählt wurde. Im rechten Teil der Abbildung 2.3 ist der Zoomfaktor erheblich größer.

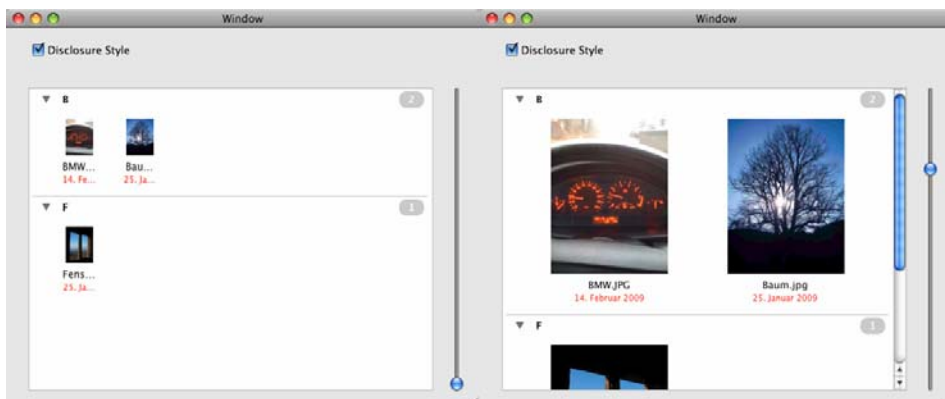


Abb. 2.3: Mit und ohne Zoom

Die Vorschaubilder, die in der *Image Browser View* des *ImageKit* angezeigt werden, sind teilweise zu klein, um zu erkennen, was dargestellt wird. Deshalb wäre es nützlich, eine Möglichkeit zum Vergrößern der Bilder zu haben.

In dem Rezept wird beschrieben, wie man die Möglichkeit zum Vergrößern in die *ImageBrowserViewDemo* einbaut.

Um das Vergrößern zu ermöglichen, muss nur eine einzige Methode in der Klasse *ImageBrowserViewController* hinzugefügt werden:

```
- (IBAction) zoomSliderDidChange:(id)sender
{
    [imageView setZoomValue:[sender floatValue]];
    [imageView setNeedsDisplay:YES];
}
```

Außerdem muss dafür eine *IBAction* in der Header-Datei definiert werden, die mit dem *NSSlider* verbunden wird, der den Zoom steuert.

Im Interface Builder wird ein vertikaler Slider (*NSSlider*) ergänzt, um das Zoomen zu ermöglichen. Der Slider befindet sich in der Library unter *INPUTS & VALUES*. Für den Slider müssen im Inspector noch die Values gesetzt werden. *MINIMUM* wird auf 0, *MAXIMUM* auf 1.0 und *CURRENT* auf 0.5 gesetzt. Nachdem der Slider durch ein Binding mit dem *ImageBrowserViewController* verbunden wurde, kann nach dem Übersetzen der Applikation, mit Hilfe des Sliders, die Größe der Vorschaubilder verändert werden.

## 2.3 Entfernen von Bildern

Die Vorschaubilder, die in der *Image Browser View* des *ImageKit* mittels Drag&Drop importiert wurden, sollen bei Bedarf wieder entfernt werden können.

Zum Löschen muss die Methode *removeItemsAtIndex* des *IKImageBrowserDataSource*-Protokolls implementiert werden.

```
- (void) imageBrowser:(IKImageBrowserView *) view removeItemsAtIndexes:
(NSIndexSet *) indexes
{
    [importedImages removeObjectAtIndexes:indexes];
    [imageView reloadData];
}
```

Die Methode entfernt einfach das Bild anhand des übergebenen Index aus dem Array mit den Bildern und schickt eine Nachricht an die *Image Browser View*, dass sich die angezeigten Daten geändert haben.

Nachdem die Methode eingefügt und die Applikation übersetzt wurde, ist ein Löschen von Bildern über die Löschen-Taste möglich.

## 2.4 Gruppieren von Bildern

Dieser Abschnitt beschreibt, wie die *ImageBrowserViewDemo* so erweitert wird, dass die Bilder nach dem Laden gruppiert sind. Der besseren Übersichtlichkeit wegen kann es sinnvoll sein, die anzuzeigenden Bilder nach bestimmten Kriterien zu gruppieren. In diesem Beispiel werden die Bilder anhand der Anfangsbuchstaben des Dateinamens angeordnet. Es wäre natürlich auch eine Gruppierung nach Dateigröße, Datum oder Ähnlichem denkbar und auch möglich.

Unter Gruppieren versteht man das Zusammenfassen von Objekten, bei denen die Gruppierungsbedingung übereinstimmt. Im Fall der Bilder ist die Gruppierungsbedingung der erste Buchstabe des Dateinamens. Für jeden Buchstaben, der bei den angezeigten Bildern vorkommt, wird eine Gruppe angelegt.

Diese Gruppen werden in dem `NSMutableArray` `groupArray` gespeichert, das in der Header-Datei der Klasse `ImageBrowserViewController` deklariert wird. Die Initialisierung findet in der Methode `init` der Klasse statt.

```
- (id)init {
    self = [super init];
    if (self != nil) {
        importedImages = [[NSMutableArray alloc] init];
        groupArray = [[NSMutableArray alloc] init];
    }
    return self;
}
```

Damit das Gruppieren von Bildern unterstützt wird, müssen zwei Methoden des *ImageBrowserDataSource*-Protokolls implementiert werden.

Die eine Methode liefert die Anzahl der Gruppen und die andere ein `NSDictionary` mit den Eigenschaften einer Gruppe, die sich an einer bestimmten Position befindet, zurück.

```
//
// UIImagePickerController Reference
//
// Bereitstellen von Informationen ueber Gruppen
// (Optional)
// - numberOfGroupsInImageBrowser:
// - imageBrowser:groupAtIndex:
```

```

- (NSDictionary *) imageBrowser:(IKImageBrowserView *)
  aBrowser groupAtIndex:(NSUInteger) index
{
    return [groupArray objectAtIndex:index];
}

- (NSUInteger) numberOfGroupsInImageBrowser:
  (IKImageBrowserView *) aBrowser
{
    return [groupArray count];
}

```

Die Gruppeneigenschaften werden in einer Methode `groupImages` bestimmt.

```

- (void) groupImages
{
    int numberOfImages =
        [groupArray count];

    // Sortieren der Bilder
    NSSortDescriptor *sortDesc =
        [[[NSSortDescriptor alloc]
         initWithKey:@"filename" ascending:YES] autorelease];
    [importedImages sortUsingDescriptors:
     [[NSArray alloc] arrayWithObject:sortDesc]];

    int rangeStart = 0;
    UniChar currentChar;
    UniChar firstFilenameChar;
    ImageItem *image = [importedImages objectAtIndex:0];
    currentChar = [[image filename] characterAtIndex:0];
    int i=1;
    for (;i<numberOfImages;i++)
    {
        ImageItem *image = [importedImages objectAtIndex:i];
        firstFilenameChar = [[image filename] characterAtIndex:0];
        if(firstFilenameChar!=currentChar) {
            [self createNewGroupForChar:
             [NSString stringWithFormat:@"%c",currentChar]
             start:rangeStart length: i-rangeStart];

            rangeStart = i;
        }
    }
}

```

```
        currentChar = firstFilenameChar;
        i--;
    }
}
[self createNewGroupForChar:
    [NSString stringWithFormat:@"%c", currentChar]
    start:rangeStart length: i-rangeStart];
}
```

Als Erstes wird die Anzahl der Bilder bestimmt. Danach werden die Bilder anhand ihres Namens sortiert. In einer `for`-Schleife wird über das Array mit den Bildern iteriert. Dabei wird für jeden neuen Anfangsbuchstaben die Methode `createNewGroupForChar:start:length:` aufgerufen.

Die Methode, die die Gruppen erzeugt, sieht folgendermaßen aus:

```
- (void) createNewGroupForChar:(NSString*)name
    start:(int)rangeStart length:(int)rangeLength
{
    NSRange range = NSMakeRange(rangeStart, rangeLength);
    NSDictionary *newGroup =
        [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:range],
            IKImageBrowserGroupRangeKey,
            [NSNumber numberWithInt:groupingStyle],
            IKImageBrowserGroupStyleKey,
            name,
            IKImageBrowserGroupTitleKey,
            nil];
    [groupArray addObject:newGroup];
}
```

In dieser Methode wird für jede Gruppe ein `NSDictionary` erzeugt, das für das Aussehen verantwortlich ist. Das `NSDictionary` enthält eine Reihe von Key-Value-Paaren, die jeweils aus einem Schlüssel und einem Wert bestehen.

Als Erstes wird über den Schlüssel `IKImageBrowserGroupRangeKey` festgelegt, welche Bilder aus dem Array `importedImages` zu dieser Gruppe gehören. Es wird dazu in einem `NSRange`-Objekt festgelegt, bei welchem Index die Gruppe beginnt und wie viele Elemente sie enthält.

Mit dem Schlüssel `IKImageBrowserGroupStyleKey` wird die Gruppierungsart angegeben, das heißt, es wird ein Objekt vom Typ `NSNumber` erzeugt, das entweder den Wert `IKGroupDisclosureStyle` oder den Wert `IKGroupBezelStyle` hat.

Der Name der Gruppe wird über den Schlüssel `IKImageBrowserGroupTitleKey` festgelegt und erhält als Wert einfach einen `NSString`.

### Hinweis

Weitere mögliche Schlüssel sind in der Klassenreferenz der Image Browser View (*IKImageBrowserView Class Reference*) unter Group-Konstanten (*Group keys*) zu finden.

Da das Gruppieren nach dem Drag&Drop erfolgen soll, muss in der Funktion `performDragOperation` der Aufruf `groupImages` ergänzt werden.

```
- (BOOL) performDragOperation:(id <NSDraggingInfo>)sender
{
    ...

    [self groupImages];

    [imageView reloadData];
    return YES;
}
```

Nun wird eine Gruppierung der Bilder direkt nach dem Import durchgeführt.

## 2.5 Gruppierungsarten

In diesem Rezept wird die Gruppierungsfunktionalität dahingehend erweitert, dass sich mittels einer `NSCheckBox` die Gruppierungsart umschalten lässt.

Das `ImageKit` bietet zwei Darstellungsmöglichkeiten für Gruppen in der `BrowserView`:

- `IKGroupDisclosureStyle` und
- `IKGroupBezelStyle`

Damit in dem Beispiel zwischen den beiden Darstellungsmöglichkeiten umgeschaltet werden kann, benötigen wir die Variable `groupingStyle`, um den aktuellen Zustand speichern zu können. Diese wird in der Methode `awakeFromNib` ergänzt.

```
- (void) awakeFromNib
{
```

```
    groupingStyle = IKGroepDisclosureStyle;  
    //...  
}
```

Damit die gewünschte Gruppierungsart bei der Anzeige der Bilder verwendet wird, muss sie in der Methode `createNewGroupForChar:start:length:` gesetzt werden.

Dazu muss über den Schlüssel `IKImageBrowserGroupStyleKey` ein Wert gesetzt werden, der die Art der Gruppierung bestimmt. Der Wert wird in Form eines `NSNumber`-Objekts übergeben.

```
- (void) createNewGroupForChar:(NSString*)name  
    start:(int)rangeStart length:(int)rangeLength  
{  
    ...  
    NSDictionary *newGroup =  
    [NSDictionary dictionaryWithObjectsAndKeys:  
    [NSNumber numberWithInt:groupingStyle],  
    IKImageBrowserGroupStyleKey,  
    name, IKImageBrowserGroupTitleKey,  
    nil];  
  
    //...  
}
```

Zum Umschalten zwischen den Gruppierungsarten wird eine `IBAction` in der Header-Datei der Klasse `ImageBrowserViewController` ergänzt. In der Implementierung der Methode wird die Gruppierungsart anhand des Zustandes der `NSCheckBox` geändert.

```
- (IBAction) groupingChanged:(id)sender  
{  
    [groupArray removeAllObjects];  
  
    int state = [sender state];  
    if(state == NSOnState) {  
        groupingStyle = IKGroepDisclosureStyle;  
    }  
    else {  
        groupingStyle = IKGroepBezelStyle;  
    }  
}
```

```
[self groupImages];  
[imageView reloadData];  
}
```

Es wird also festgestellt, welchen Zustand die `NSCheckBox` (`sender`) hat, und der `groupingStyle` wird umgesetzt. Danach wird der *Image Browser View* mitgeteilt, dass sie sich neu zeichnen soll.

Damit die Methode `groupingChanged` aufgerufen wird, muss im Interface Builder eine `NSCheckBox` ergänzt werden, die an die `IBAction` gebunden wird.

Wenn jetzt in der Applikation der Haken in der Checkbox gesetzt oder entfernt wird, dann wird auch die Gruppierungsart geändert.

## 2.6 Setzen von Farben

In diesem Abschnitt wird erklärt, wie man die Hintergrundfarbe ändert, wenn die Gruppierungsart `IKGroupBezelStyle` gewählt wurde, das heißt, man möchte beispielsweise für jede neue Gruppe eine andere Hintergrundfarbe benutzen.

Damit für jede Gruppe eine neue Hintergrundfarbe benutzt wird, muss in dem `NSDictionary` für die Gruppe für den Schlüssel `IKImageBrowserGroupBackgroundColorKey` ein Wert gesetzt werden und die Methode muss um einen Parameter erweitert werden.

```
- (void) createNewGroupForChar:(NSString*)name  
    start:(int)rangeStart length:(int)rangeLength  
    color:(NSColor*)groupColor  
{  
    ...  
    NSDictionary *newGroup =  
        [NSDictionary dictionaryWithObjectsAndKeys:  
        [NSValue valueWithRange:range],  
            IKImageBrowserGroupRangeKey,  
        [NSNumber numberWithInt:groupingStyle],  
            IKImageBrowserGroupStyleKey,  
        name, IKImageBrowserGroupTitleKey,  
        groupColor, IKImageBrowserGroupBackgroundColorKey,  
        nil];  
    ...  
}
```

Der Methode `createNewGroupForChar:start:length:color:` wird ein Objekt vom Typ `NSColor` übergeben. Dieses Objekt wird in der Methode `groupImages` folgendermaßen erzeugt:

```
- (void) groupImages
{
    ...
    UniChar firstFilenameChar;

    NSColor* groupColor;

    ...
    for (;i<numberOfImages;i++)
    {
        groupColor =
        [NSColor colorWithCalibratedRed: rand()*0.000000005
                                     green: rand()*0.000000005
                                     blue: rand()*0.000000005
                                     alpha:1.0];

        ...

        if(firstFilenameChar!=currentChar) {
            [self createNewGroupForChar:
             [NSString stringWithFormat:@"%c",currentChar]
             start:rangeStart length: i-rangeStart
             color: groupColor];
        }
    }
    [self createNewGroupForChar:
     [NSString stringWithFormat:@"%c",currentChar]
     start:rangeStart length: i-rangeStart
     color: groupColor];
}
```

Man sieht, dass einfach nur ein `NSColor`-Objekt erzeugt wird und die Rot-, Grün- und Blau-Werte mittels der Zufallsgeneratorfunktion `rand()` variiert werden. Das erzeugte Objekt wird dann als Parameter an die Methode übergeben, die die Gruppe erzeugt.

## 2.7 Einblenden von Bildinformationen

In diesem Rezept wird beschrieben, wie die angezeigten Bilder um Zusatzinformationen ergänzt und wie das Aussehen angepasst werden kann.

Die *Image Browser View* bietet die Möglichkeit, folgende *Styles* zu aktivieren:

- Titel (*Title*)
- Untertitel (*Subtitle*)
- Outline (*Outlined*)
- Schatten (*Shadowed*)

Zum Ein- und Ausblenden gibt es zwei Möglichkeiten:

- Im Interface Builder ein Häkchen setzen
- Im Programmcode aktivieren

Im Interface Builder muss die *Image Browser View* selektiert werden und im Inspector das Attributes Pane ausgewählt werden. Dort können unter STYLE mit dem Häkchen die Titel ein- und ausgeblendet werden.

Im Programmcode macht man dies am besten in der *awakeFromNib*-Methode über den Aufruf von `setCellStyleMask`: der Klasse `IKImageBrowserView`. Als Parameter wird eine Bit-Maske übergeben.

```
// IKCellStyleNone
// IKCellStyleShadowed
// IKCellStyleOutlined
// IKCellStyleTitled
// IKCellStyleSubtitled
[imageBrowserView setCellStyleMask:IKCellStyleTitled
                  | IKCellStyleSubtitled];
```

Es sind fünf Werte möglich, die mit einem logischen ODER verknüpft werden. In dem Beispiel werden der Titel und die Untertitel aktiviert.

Will man beispielsweise die Farbe oder den Font der Bildunterschriften anpassen, lässt sich über den Wert `IKImageBrowserCellsSubtitleAttributesKey` ein `NSDictionary` auslesen, in dem die Farbwerte und der Font umgestellt werden können. Auch dies sollte in der Methode *awakeFromNib* gemacht werden.

```
NSMutableDictionary *attrs =
    [[imageBrowserView
     valueForKey:@"IKImageBrowserCellsSubtitleAttributesKey"]
```

```
mutableCopy] autorelease];

[attrrs setValue:[NSColor redColor] forKey:@"NSColor"];

[imageBrowserView setValue:
    attrrs forKey:@"IKImageBrowserCellsSubtitleAttributesKey"];
```

In dem Beispiel wird das `NSDictionary` ausgelesen und eine Kopie erstellt. Dann wird der Farbwert auf Rot gesetzt und das veränderte `NSDictionary` wird wieder an die `IKImageBrowserView` übergeben.

Dies ist jetzt nur ein Beispiel dafür, was alles mit dem Image Kit möglich ist. Im Dokument *IKImageBrowserView Class Reference* findet man unter *View Options Keys* weitere Schlüssel, mit denen man mittels KVC Werte in der `IKImageBrowserView` anpassen kann.

Aus der Apple-Dokumentation ist oft nicht ersichtlich, welche Werte gesetzt werden können. Deshalb sollte man sich die Werte mit den angegebenen Schlüsseln einfach auslesen und im Debugger ansehen.

### Hinweis

Ist beispielsweise in dem `NSDictionary`, das mit dem Schlüssel `IKImageBrowserCellsTitleAttributesKey` abgefragt wurde, ein Objekt vom Typ `NSFont` enthalten, dann kann dieses auch umgesetzt werden.

Nachdem wir die Titel und die Untertitel eingeblendet haben, müssen wir noch die Klasse `ImageItem` erweitern, damit die von uns gewünschten Werte angezeigt werden.

Dazu bietet das *IKImageBrowserItem Protocol* mehrere Funktionen an. Im folgenden Listing sind zwei dieser Methoden implementiert.

```
- (NSString *) imageTitle
{
    return filename;
}

- (NSString *) imageSubtitle
{
    NSDateFormatter *dateFormatter =
        [[[NSDateFormatter alloc] init] autorelease];
    [dateFormatter setDateStyle:NSDateFormatterLongStyle];
```

```
[dateFormatter setTimeStyle:NSDateFormatterNoStyle];

return [dateFormatter stringFromDate:filedate];
}
```

Die erste Methode liefert als Titel einfach den Dateinamen zurück. Die zweite Methode ist für den Untertitel verantwortlich, der etwas kleiner als der Titel angezeigt wird. In dem Beispiel wird das Dateidatum zurückgeliefert.

## 2.8 Kopieren von Dateien

Hier wird erklärt, wie man Dateien mit dem `NSFileManager` von einer Stelle im Dateisystem der Festplatte an eine andere Stelle kopiert. Diese Funktionalität werden wir später vor dem Brennen der CD benötigen, um alle zu brennenden Bilder in einem Ordner auf der Festplatte abzulegen.

Deshalb definieren wir die Methode `burnImagesToCD:(id)sender` als `IBAction`:

```
- (IBAction) burnImagesToCD:(id)sender
{
    NSLog(@"Burn images %d to CD", [importedImages count]);
    NSString *burnPath = @"/tmp/BurnFolder";

    // Anlegen eines Verzeichnisses
    [[NSFileManager defaultManager]
     createDirectoryAtPath:burnPath attributes:nil];

    ImageItem *item;
    NSError *error;
    for(item in importedImages)
    {
        NSLog(@"Filename: %@", [item imagePath]);

        NSString * destination =
            [burnPath stringByAppendingPathComponent: filename];
        NSLog(destination);

        // Kopieren der Bilddatei
        if(![[NSFileManager defaultManager]
            copyItemAtPath:[item imagePath]
            toPath:destination error:&error])
        {
            NSAlert *alertPanel = [NSAlert alertWithError:error];
        }
    }
}
```

```
(void) [alertView runModal];  
    }  
    }  
}
```

Wir legen einen Ordner fest, in dem alle zu brennenden Bilder abgelegt werden. Dann iterieren wir über alle Bilder und kopieren jedes einzelne Bild von der Position `imagePath` nach `destination`. Falls beim Kopieren ein Problem auftritt, wird ein Dialogfenster geöffnet, das eine Fehlerbeschreibung anzeigt. Es kann beispielsweise vorkommen, dass schon ein Bild mit dem gleichen Namen in dem Zielverzeichnis liegt.

## 2.9 Brennen von CDs und DVDs

Das Brennen von CDs und DVDs wird mit den Frameworks `DiscRecording.framework` und `DiscRecordingUI.framework` erledigt. Das `DiscRecording.framework` dient der Aufbereitung der zu brennenden Daten und führt die Brennoperation durch. Das `DiscRecordingUI.framework` stellt mehrere `NSPanels` zur Verfügung, die vor und beim Brennen angezeigt werden.

Um die in Rezept 2.8 in einen Ordner kopierten Bilder zu brennen, erweitern wir die Methode – (`IBAction`) `burnImagesToCD:(id)sender`.

```
- (IBAction) burnImagesToCD:(id)sender  
{  
    ...  
    DRTrack *track =  
        [DRTrack trackForRootFolder:  
         [DRFolder folderWithPath:burnPath]];  
  
    DRBurnSetupPanel *bsp = [DRBurnSetupPanel setupPanel];  
  
    int status = [bsp runSetupPanel];  
    if ( status == NSOKButton ) {  
        DRBurn *burn = [bsp burnObject];  
        DRBurnProgressPanel *bpp =  
            [DRBurnProgressPanel progressPanel];  
        // Brennen wird gestartet  
        [bpp beginProgressPanelForBurn: burn layout:track];  
    }  
}
```

Als Erstes wird ein `DRTrack` erstellt, der auf den Ordner `burnPath` gesetzt wird. Dies sind die Daten, die auf die CD oder DVD gebrannt werden sollen.

Als Nächstes wird das `DRBurnSetupPanel` erstellt, in dem der Benutzer verschiedene Einstellungen vornehmen kann. Wenn dieses Panel mit dem OK-Button geschlossen wird, wird der Brenn-Prozess gestartet. Dazu wird ein `DRBurnProgressPanel` erstellt, das die Daten übergeben bekommt.

Wenn man die Methode in diesem Zustand verwendet, ist es nicht möglich, die CD so zu brennen, dass später weitere Daten hinzugefügt werden können. Das Häkchen `LEAVE DISC APPENDABLE` ist immer ausgegraut. Um diese Checkbox zu aktivieren, muss folgende Zeile eingefügt werden:

```
// Aktivieren der Checkbox 'Leave disc appendable'
[bsp setCanSelectAppendableMedia:YES];
```

Auf die gleiche Weise kann eine Checkbox eingeblendet werden, die es ermöglicht, den Brennvorgang zu simulieren.

```
// Simulieren des Brennvorgangs ermöglichen
[bsp setCanSelectTestBurn:YES];
```

Außerdem lässt sich der Titel des OK-Buttons nach den eigenen Bedürfnissen anpassen.

```
// Titel des Buttons in 'Bilder brennen' umbenennen
[bsp setDefaultButtonText:@"Bilder brennen"];
```

Nach diesen Änderungen sieht das `DRBurnSetupPanel` folgendermaßen aus:

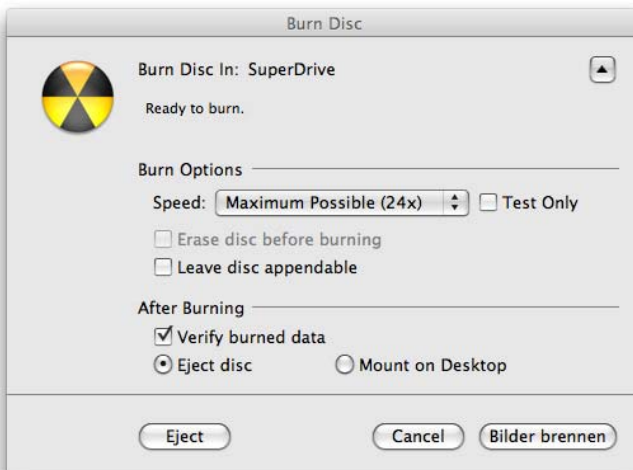


Abb. 2.4: `DRBurnSetupPanel`

## 2.10 Kategorie für das DRBurnSetupPanel

Bei vielen Aktionen kann es nützlich sein, auf Aktionen im DRBurnSetupPanel zu reagieren. Deshalb existiert das DRSetupPanelDelegate. Diese Kategorie von NSObject stellt eine Reihe nützlicher Methoden zur Verfügung, die wir benutzen, um Informationen über das Laufwerk und die eingelegte CD zu bekommen.

Die Kategorie stellt folgende Methoden zur Verfügung:

- `setupPanel:deviceContainsSuitableMedia:promptString:`
- `setupPanel:deviceCouldBeTarget:`
- `setupPanelDeviceSelectionChanged:`
- `setupPanelShouldHandleMediaReservations:`

In diesem Rezept werden wir nur die Methode `setupPanel:deviceContainsSuitableMedia:promptString:` verwenden, um die Informationen zu dem CD-Brenner und der CD auszulesen.

### Wichtig

Weitere Informationen und Beschreibungen zu dieser Kategorie findet man unter dem Stichwort *DRSetupPanelDelegate* in der Apple-Dokumentation.

In der folgenden Methode der Kategorie fragen wir die Informationen und den Status des im DRBurnSetupPanel ausgewählten Brenners ab.

```
- (BOOL)setupPanel:(DRSetupPanel*)aPanel
    deviceContainsSuitableMedia:(DRDevice*)device
    promptString:(NSString**)prompt
{
    NSDictionary *info = [device info];
    NSDictionary *status = [device status];

    for (NSString *key in info)
        NSLog(@"Key = %@, Value = %@",
              key,
              [info objectForKey: key]);
    for (NSString *key in status)
        NSLog(@"Key = %@, Value = %@",
              key,
              [status objectForKey: key]);
}
return YES;
}
```

Anschließend geben wir alle Keys und deren zugehörigen Values der `NSDictionary` auf der Konsole aus. Auf diese Weise kommt man auf einfache Art an eine Reihe von Informationen über den Brenner und die eingelegte CD.

### Hinweis

Ein `NSDictionary` lässt sich auch direkt mit `NSLog` ausgeben. Allerdings ist es manchmal sinnvoll, eine eigene Schleife dafür zu erstellen, wenn die Formatierung nicht den eigenen Vorstellungen entspricht.

Mit diesen Informationen lässt sich in der Methode beispielsweise überprüfen, ob eine normale CD-R eingelegt ist. Man stelle sich vor, dass verhindert werden soll, dass auf die teureren CD-RWs oder auf DVDs gebrannt werden soll. Der Code, um dies zu verhindern, sieht folgendermaßen aus:

```
// Teste, ob eine CD vom Typ CDR eingelegt ist
NSString *mediaType =
    [[status objectForKey:DRDeviceMediaInfoKey]
    objectForKey:DRDeviceMediaTypeKey];
if ([mediaType isEqualToString:DRDeviceMediaTypeCDR] == NO)
{
    return NO;
}
```

Dieser Code wird unter den `while`-Schleifen vor der `return`-Anweisung eingefügt.

Damit die Methoden dieser Kategorie aufgerufen werden, muss ein Delegate für das `DRBurnSetupPanel` in der Methode `burnImagesToCD` gesetzt werden.

```
[bsp setDelegate:self];
```

## 2.11 Loggen von Fehlern beim Brennen

In diesem Rezept wird erklärt, wie eine Log-Ausgabe erzeugt wird, die einen aufgetretenen Fehler beim Brennen einer CD protokolliert. Dazu verwenden wir eine Kategorie, die für das `DRBurnProgressPanel` existiert.

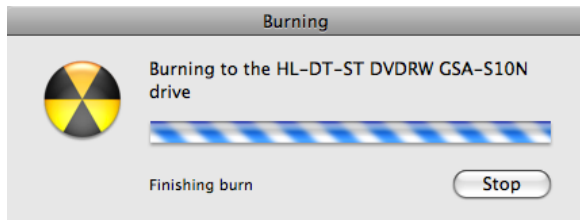


Abb. 2.5: DRBurnProgressPanel

Die Kategorie kann dafür verwendet werden, um vor und nach dem Brennvorgang Aktionen auszuführen.

Dafür werden folgende Methoden angeboten:

- `burnProgressPanel:burnDidFinish:`
- `burnProgressPanelDidFinish:`
- `burnProgressPanelWillBegin:`

Wir verwenden in dem Rezept nur die Methode `burnProgressPanel:burnDidFinish:`, um einen eventuellen Fehler anzufragen:

```
- (BOOL) burnProgressPanel:(DRBurnProgressPanel*)theBurnPanel
burnDidFinish:(DRBurn*)burn
{
    NSDictionary* burnStatus = [burn status];
    NSString* state=[burnStatus objectForKey:DRStatusStateKey];

    if ([state isEqualToString:DRStatusStateFailed])
    {
        NSLog(@"Burning of images failed!");
    }
    return YES;
}
```

In dem Beispielcode wird der Status des Brennvorgangs abgefragt und überprüft, ob ein Fehler aufgetreten ist.

Auf ähnliche Weise ist es möglich, Hinweise einzublenden, bevor das `DRBurnProgressPanel` sich öffnen wird, beispielsweise um den Benutzer darauf hinzuweisen, dass er die Applikation während des Brennvorgangs nicht schließen soll.

Damit die Methoden der Kategorie aufgerufen werden, muss ein Delegate in der Methode `burnImagesToCD` gesetzt werden (siehe Abschnitt 2.9).

```
[bpp setDelegate:self];
```

## 2.12 System-Icons verwenden

In diesem Rezept wird erklärt, wie die System-Icons mit Hilfe des Icon-Service angesprochen und verwendet werden können. Wir wollen ein Icon innerhalb eines `NSButton` anzeigen, der in die `ImageBrowser`-Applikation integriert werden soll und das Brennen anstößt.

Oft stellt sich die Frage, wo die System-Icons von Apple zu finden sind und wie man sie in eigenen Applikationen verwendet. Apple hat zu diesem Zweck im `Carbon`-Framework die Klassen `Icons` und `IconsCore` zur Verfügung gestellt.

Der folgende Code-Schnipsel zeigt, wie man die System-Icons anspricht und verwendet.

```
UIImage * burnIcon =  
    [[NSWorkspace sharedWorkspace]  
    iconForFileType: NSFileTypeForHFSTypeCode(kBurningIcon)];
```

Der Code sorgt dafür, dass die Variable `burnIcon` mit einem `UIImage` initialisiert wird. Das Image ist in diesem Fall das Burn-Symbol.

### Wichtig

#### Verfügbare Icons

Eine Liste mit verfügbaren Icons findet man in dem Dokument: *Icon Services und Utilities Reference* auf der Apple-Developer-Seite unter:

<http://developer.apple.com/documentation/Carbon/reference/IconServices/Reference/reference.html>

Damit wir das Icon in einem `NSButton` verwenden können, muss ein `IBOutlet` erstellt werden. Dieses wird im *Interface Builder* mit dem Button der View verknüpft.

In der `awakeFromNib`-Methode der Klasse `ImageBrowserViewController` setzen wir das Image.

```
[burnButton setImage:burnIcon];
```

Die fertige Oberfläche mit dem neuen Button ist in *Abbildung 2.6* zu sehen.

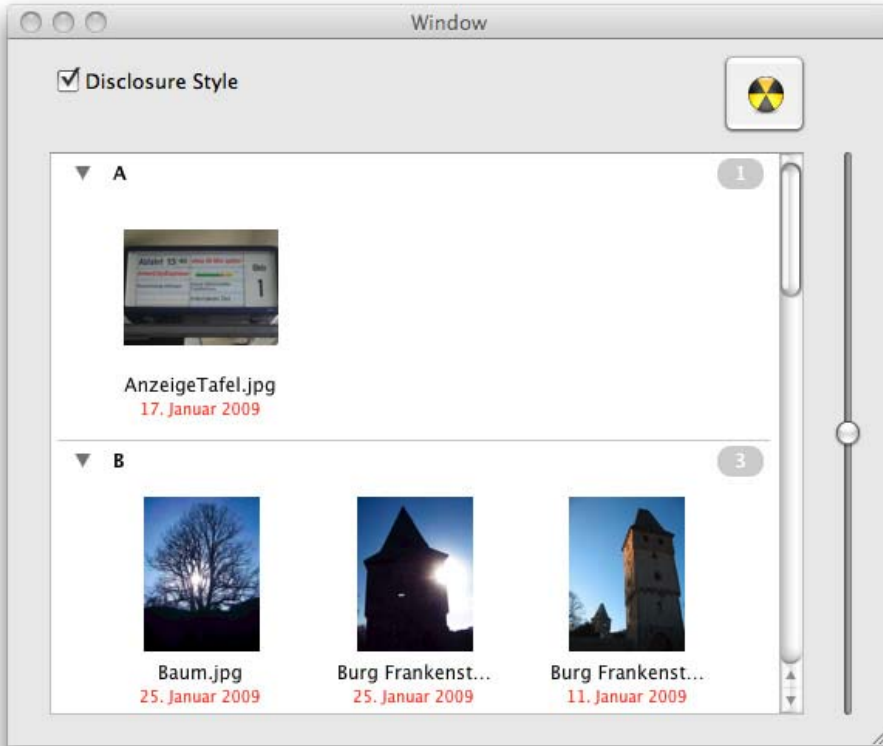


Abb. 2.6: Button mit System-Icon