```
/* MiscSwapContentsController.m
 *
 * A very simple class for controlling swapAble views. A subclass is a must
 * to easily work with the MiscSwapView classes.
 *
 * For more interface-info see the header file. In depth information
 * can be found here in the source-code.
 *
 * Written by:          Thomas Engel
 * Created:             24.01.1994 (Copyleft)
 * Last Modified:       25.09.1994
      Changes are in black bold.
 */

//#import "MiscSwapContentsController.h"
#import "MiscSwapView.h"
#import "MiscSwapContentsController.h"
```

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

// Defined in order to do archiving and versioning properly, so
// if the archiving changes we will be able to read all versions.

#define MISC_SCC_VERSION 0
#define MISC_SCC_CLASSNAME "MiscSwapContentsController"


// Declarations of private methods that need not be in the
// header file.

@interface MiscSwapContentsController (PrivateMethods)

- _dispatchToDelegate: (SEL)message;
- _dispatchToDelegate: (SEL)message with: anObject;
```

```
@end


@implementation MiscSwapContentsController

+ initialize
{
	// Sets the version of this class for archiving purposes.

	if (self == [MiscSwapContentsController class])
		[self setVersion: MISC_SCC_VERSION];

	return self;
}

- init
{
```

```
        [super init];

        swapView = nil;
        view = nil;
        trigger = nil;
        triggerTag = 0;
        delegate = nil;

        return self;
}

- setSwapView:aView
{
        // Here we set our swapView. This objects is the right place to
        // register ourselves for the swapping. But only if we have a trigger..
        // otherwise we have to wait for a awakeFromNib message.
        // Sorry we can only register for one swapView at a time. So if we had
```

```
	// a swapView before..lets say good bye.

	if( swapView ) [swapView removeController:self];

	swapView = aView;

	// I removed the necessity that the trigger has to be set to something
	// before adding ourselves to the swapview's list of controllers. The
	// swapView class was also changed so it would treat controllers with
	// no trigger in the correct way, since you can depend upon tags and
	// not use any controllers.
	// You also do not need awakeFromNib anymore.

	[swapView addController: self];

	return self;
}
```

```
- swapView
{
	if (swapView != nil)
		return swapView;

	// If we aren't pointing to a swapview maybe our delegate
	// knows something we don't. I don't really know why you
	// would have the delegate know the swapView, but it is
	// here for consistency.
	return [self _dispatchToDelegate: @selector(swapView)];

}

- setView:aView
{
	view = aView;
```

```
        return self;
}

- view
{
        if (view != nil)
                return view;

        // This way you can have views that are all in seperate nibs and
        // have the delegate get them when they are needed.
        return [self _dispatchToDelegate: @selector(view)];
}

- setTrigger:anObject
{
        // The trigger is the object we are related to. By default we try to set
        // the triggerTag according to that object.
```

```
	// Activating the trigger object (or an object with this tag) will cause
	// us to swap in.

	trigger = anObject;

	if( [trigger respondsTo:@selector(tag)] )
			[self setTriggerTag:[trigger tag]];
	else	[self setTriggerTag:0];

	return self;
}

- trigger
{
	return trigger;
}
```

```
- setTriggerTag:(int)tag
{
	// Sets the tag we will be activated for.
	// Working with tags frees us from having to know what typ of object caused
	// the action (TextCell,ButtonCell,Matrix or what ever..) as long as the
	// tags are handled the right way.

	triggerTag = tag;
	return self;
}

- (int)triggerTag
{
	return triggerTag;
}
```

// As an alternative to subclassing this controller to "control" the rest of

// the UI objects on the view, you can make it a seperate class and connect
// it to the delegate, so it knows when it will be swapped in and out.

```
- delegate
{
	return delegate;
}

- setDelegate: aDelegate
{
	delegate = aDelegate;
	return self;
}


/*
 * These revert/ok msg are send to setup/save the contents of a view.
```

```
 * Only the revert msg is invoked by default. If you have to store some
 * data when swapping out implement a [self ok:self] msg iside willSwapOut.
 * Ok should be used to save the changes made and revert should init the
 * view to show the current settings.
 */

- ok:sender
{
	[self _dispatchToDelegate: @selector(ok:) with: sender];
	return self;
}

- revert:sender
{
	[self _dispatchToDelegate: @selector(revert:) with: sender];
	return self;
}
```

```
/*
 * These messages we will get from our swapView. ThatÂs how we can
 * recognize that maybe some things have to be written to the defaults
 * database or something has to be updated etc.
 * You should override them in you subclass.
 * They are no actionMethods because we always know who our swapCtrl. is.
 * So sender is not needed here.
 */

- willSwapIn
{
        [self revert:self];
        [self _dispatchToDelegate: @selector(willSwapViewIn:) with: self];
        return self;
}
```

```
- willSwapOut
{
	[self _dispatchToDelegate: @selector(willSwapViewOut:) with: self];
	return self;
}

- didSwapIn
{
	[self _dispatchToDelegate: @selector(didSwapViewIn:) with: self];
	return self;
}

- didSwapOut
{
	[self _dispatchToDelegate: @selector(didSwapViewOut:) with: self];
	return self;
}
```

```
- read: (NXTypedStream *)stream
{
  int  version;

        [super read: stream];
        version = NXTypedStreamClassVersion(stream, MISC_SCC_CLASSNAME);

        switch (version)
        {
                case 0:
                        swapView = NXReadObject (stream);
                        view = NXReadObject (stream);
                        trigger = NXReadObject (stream);
                        delegate = NXReadObject (stream);
```

```
                NXReadType (stream, "i", &triggerTag);
                break;
        default:
                break;
    }

    return self;
}

- write: (NXTypedStream *)stream
{
    [super write: stream];
    NXWriteObjectReference (stream, swapView);
    NXWriteObjectReference (stream, view);
    NXWriteObjectReference (stream, trigger);
    NXWriteObjectReference (stream, delegate);
    NXWriteType (stream, "i", &triggerTag);
```

```
	return self;
}

@end


@implementation MiscSwapContentsController (PrivateMethods)

- _dispatchToDelegate: (SEL)message
{
	if (delegate && [delegate respondsTo: message])
		return [delegate perform: message];

	// No delegate or doesn't respond to message.
	return nil;
}
```

```
- _dispatchToDelegate: (SEL)message with: anObject
{
        if (delegate && [delegate respondsTo: message])
                return [delegate perform: message with: anObject];

        return nil;
}

@end


/*
 * History: 25.09.94 Added archiving (read:, write:, +initialize).
 *                                  Added a delegate so this class could be on a palette
```

```
*                   and would not require subclassing. Also took away
*                   the awakeFromNib method since you can now register a
*                   controller without having a trigger set.
*
*           14.02.94 Changed the classes name to MiscSwapContentsController
*                   from the MiscSwapSubviewController becauses it fits better
*                   to what it really is.
*
*           24.01.94 Made it a Misc object and changed it to work with the
*                   new Misc stuff.
*
*           09.01.94 Added the ok/revert stuff.
*
*           08.01.94 Derived from my old swapViewdelegate. The code+methods
*                   were cleaned up a bit.
*
*
```

* Bugs: - no read/write; **(not anymore)**
*
* - Maybe I should do more responds to or class checking.. hmm ??
*
* - Not a bug but: I don't love the double registry made by awakeFromNib
* and setSwapView. It works but it's not elegant. **(not anymore)**
*/