```
/* MiscSwapView_ByObject.m
 *
 * This is a MiscSwapView category. It can handle swapping of different
 * contentViews (controlled by MiscSwapContentsController's) into ourself by
 * comparing trigger objects or tags (trigger tags and controller tags).
 * More 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

 My changes and comments are in bold black.
 */

#import "MiscSwapView.h"
#import "MiscSwapContentsController.h"
//#import <misckit/misckit.h> // since for MiscKit the headers were moved by Don
```

```
@implementation MiscSwapView(ByObject)


- trigger
{
	return trigger;
}

- swapContentView:sender
{
	// This is the method that has to be triggered by each trigger for a nice
	// swap. It incorporates the delegate and the new&previous viewCtrl.
	// Our delegate gets informed before the viewControllers are.
	// The new controller will get willSwapIn which should cause a revert or
	// any other method causing the contoller to update/init its view. It's
	// the last chance before it will be displayed.
```

```
// We could check wether the controller has really changed or not but
// this might cause a problem when using a inspector because the same
// inspector might show different data.
// Maybe we could and a switch here...right now I'll leave it this way.
// It should work fine.
// Be sure that whenever you send the message you really have some changes
// to show. Otherwise it might be not that perfect.
//
// Anyway. The same view will never swap in twice! SwapView take care of
// that inside setContentView. So you don't have the overhead of a doubled
// drawing time.

id      oldController;

trigger = sender;

if( [delegate respondsTo:@selector(viewWillSwap:)] )
```

```
        [delegate viewWillSwap:self];

oldController = currentController;
currentController = [self findControllerForTrigger: sender];

[oldController willSwapOut];
[currentController willSwapIn];

[self setContentView:[currentController view]];

if( [delegate respondsTo:@selector(viewDidSwap:)] )
        [delegate viewDidSwap:self];

[oldController didSwapOut];
[currentController didSwapIn];

return self;
```

```
}

- addController:sender
{
	// Adding a viewController has to ensure that there is only one controller
	// for one trigger. The controller added last will be the only known
	// controller after adding has finished!
	// To find interferring controllers we use our findController method.
	// This needs the trigger to check and the right tagComparison setting.
	// If you change the tagComparison setting later the results might not be
	// correct!
	// You are now allowed to add a controller that does not have a trigger
	// since you can rely totally on tag comparisons. A controller with a
	// trigger of nil will for sure not match any other trigger, but could
	// match by tag.

	// Since I changed findController to findControllerForTrigger:, you don't
```

```
        // have to save the trigger then restore it.
//      id      oldTrigger;
        id      oldController;

//      oldTrigger = trigger;
//      trigger = [sender trigger];

        // If there already is someone...remove him.!

        // Also do not want the same controller registering more than once. The
        // old way, if a controller changed it's trigger, it would get by
        // findControllerForTrigger: and would be added again.
        [self removeController: sender];

        oldController = [self findControllerForTrigger: [sender trigger] ];
        if( oldController ) [self removeController:oldController];
```

```
        [controllers addObject:sender];

//      trigger = oldTrigger;

        return self;
}

- removeController:sender
{
        // Here we remove  aViewController from the subviewControllers list.
        // ATTENTION: This does not cause the view to disappear when it is the
        // current swapView contents!

        [controllers removeObject:sender];
        return self;
}
```

```
- removeAllControllers
{
	[controllers empty];
	return self;
}

- controllers
{
	return controllers;
}

- contentsController
{
	return currentController;
}

- setTagComparison:(BOOL)flag
```

```
{
	// Should we compare the tags first...and then the objects ?

	tagComparison = flag;
	return self;
}

- (BOOL)doesTagComparison
{
	return tagComparison;
}

- findControllerForTrigger: aTrigger
{
	// This is the basic comparison center. Subclasses of this class should
	// implement only the new findControllerByTag(Object) methods and leave
	// this method untouched. Sometime there is a way of finding a more
```

```
// 'useable' trigger even inside the swapAction method.
// Using this method from addController: to see if there is already a
// a controller for that trigger only works when using triggers as the
// comparison. It does not for tags, since we don't care if someone
// registers all the controllers for the same tag..

id      newController;

newController = nil;

// Used to also check that trigger responded to tag, but that did not
// work when the trigger was nil and we wanted a comparison by tag only.

if( [self doesTagComparison] )
newController = [self findControllerByTag:[aTrigger tag]];
if( !newController ) newController = [self findControllerByObject: aTrigger];
```

```
	return newController;
}

- findControllerByTag:(int)aTag
{
	// Here we try to find the right controller by comparing the tags.
	// Really simple. Well...the trigger has to have a tag different from
	// zero.

	id		aViewController;
	int		i;

	if( aTag == 0 ) return nil;

	aViewController = nil;

	// Ok now lets find out what viewController refers to this tag.
```

```
	for( i=0; i<[controllers count]; i++ )
	{
		if( aTag == [[controllers objectAt:i] triggerTag] )
		{
			aViewController = [controllers objectAt:i];
			break;
		};
	}
	return aViewController;
}

- findControllerByObject:aTrigger
{
	// Here we simple compare the objects. They have to be the SAME...not only
	// similar!!!
```

```
id              aViewController;
int             i;

aViewController = nil;

// Now that we allow nil triggers we must check for them.
if (aTrigger == nil)
        return nil;

for( i=0; i<[controllers count]; i++ )
{
        if( aTrigger == [[controllers objectAt:i] trigger] )
        {
                aViewController = [controllers objectAt:i];
                break;
        };
}
```

```
        return aViewController;
}

@end

/*
 * History: 24.02.94 Made it a MiscswapView Category.
 *
 *                  24.01.94 Made it a subclass of MiscSwapView
 *
 *                  08.01.94 Switched to tagComparison for better reading.
 *                           choosesByTagFirst was not that nice.
 *
 *                  21.12.93 Code transferred from the old swapPopManager and
 *                           some viewController methods added plus the trigger object
 *                           which now stores the object triggering our swap.
 *
```

```
*                   20.12.93 Enlightened the controller to check the tags if they are
*                            set. This helps to localize apps.
*
*                   04.12.93 Added a delegate to this class the enable better
*                            command-key handling.
*
*                   04.11.93 First steps to a general-purpose swapPopManager.
*
*
* Bugs: - I'm not sure about what to free....
*/
```