Mike Ferris - February 5th, 1994

MiscClassVariable

Inherits From: Object

Declared In: misckit/MiscClassVariable.h

Class Description

MiscClassVariable is used to implement class variables that keep separate values for each subclass. Usually to simulate class variables in Objective-C a static variable is declared at the top-level of the implementation block of the class and class methods are provided to get and set this static variable. The result is that all subclasses share the same value unless they explicitly re-implement the class variable again. MiscClassVariable solves this problem. Internally, it uses a hash table keyed on the class' name to store a separate value for each subclass.

Instead of declaring a static variable of the type you want in your implementation, you will declare a static

MiscClassVariable pointer. The **+initialize** method should allocate and initialize this object. Then the get/set methods should call the MiscClassVariable methods **-getObjectForClass:** and **-setObject:forClass:** to do their work. Finally, the **+startUnloading** method should free the object. See the MiscNibController class for an example of the use of this object. Warning: the current implementation of MiscClassVariable can only store object values.

The MiscClassVariable can be configured at init time to free its values when it is freed. The default is not to free its values.

This class does not have **-read:** and **-write:** methods. The reason for this is that I don't think it needs them. MiscClassVariable instances are meant to be used with static top-level variables of other classes, not as parts of instances of other classes. They should never need archiving. Of course I'm probably wrong...

...for example, you could use a MiscClassVariable to fudge extra instance variables for use by Objective-C categories of existing objects. The trick to doing this is to first name each instance of an object explicity and then use a cast to cast *self* as a Class when using the get/set methods. Overriding ±**read:** and ±**write:** in a category would be a problem, however, so this solution is far from ideal.

Instance Variables

HashTable	***ht**;
Bool	**doesFreeValues**;

ht	The hash table used to store the values.
doesFreeValues	Whether the values should be automatically freed when the object is..

Method Types

Initializing the class	+ initialize
Initializing instances	- init - initDoesFreeValues: - free
Managing the values	- setObject:forClass: - getObjectForClass:

Class Methods

initialize
+ **initialize**

Sets the class version number.

Instance Methods

free
± **free**

This method frees the hash table used to contain the class variables. It first frees all the object values if the instance is supposed to free its values.

See also: ± **init**, ± **initDoesFreeValues:**

getObjectForClass:
± **getObjectForClass:**(Class)*class*

Looks up the value (if any) for the given class and returns it. Returns nil if no value has been set for the given class.

See also: ± **setObject:forClass:**

init:
± **init**

Calls ±**initDoesFreeValues:** with **NO** as its argument.

See also: ± **initDoesFreeValues:**, ± **free**

initDoesFreeValues:

± **initDoesFreeValues:**(BOOL)*flag*

This is the designated initializer for the class. It justs sets things up (the hash table in particular).

See also: ± **init**, ± **free**

setObject:forClass:

± **setObject:***obj* **forClass:**(Class)*class*

Sets the value for the given class to *obj*. If the receiver is supposed to free its values, the old value (if any) is freed and this method returns nil. If the receiver doesn't free its values, the old value is returned if it existed; otherwise nil is returned.

See also: ± **getObjectForClass:**