Object (MiscObjectRecycler)

Declared In: <misckit/MiscObjectRecycler.h>

Category Description

This category allows for more efficient management of objects which go in and out of scope quickly. Because freeing and allocating memory dynamically can be rather slow, the object recycler's job is to keep track of ªfreedº objects until they are needed again. Then, the object is re-initialized and put back into action. Use of the recycler for classes such as MiscColor and MiscString could provide fairly large boosts in performance for applications which use those classes extensively.

To effectively use the object recycler, when you need a new object, send a **;MiscObjectRecycler.rtf;newFromRecycler;¬+newFromRecycler** message to the appropriate class. For example, `[MiscString newFromRecycler]` will obtain a recycled MiscString and re-initialize it for you. When you are finished with an object, simply send it the **;MiscObjectRecycler.rtf;recycle;¬±recycle** message instead of ±**free** and it will be sent to the recycler to wait until it is needed again.

If a particular class needs finer control over how it is re-initialized, simply override **;MiscObjectRecycler.rtf;firstInitObject:;¬+firstInitObject:** and **;MiscObjectRecycler.rtf;reInitObject:;¬+reInitObject:** to do things the right way. Normally a simple ±**init** message is sent. If you need access to the recycler for a particular class, it is obtained with either **;MiscObjectRecycler.rtf;recycler;¬+recycler**, **;MiscObjectRecycler.rtf;recyclerForClass:;¬+recyclerForCla**

ss:, or **;MiscObjectRecycler.rtf;recyclerForClassName:;¬+recyclerForClassName:**.

There is one very important caveat when using the object recycler: do **not** recycle an object more than once! If you do this, strange things will occur due to objects suddenly re-initializing themselves while in use. The best way to avoid this is to make sure you set all pointers to an object to nil when you recycle it. If you tend to be scatterbrained, and feel that the recycler should keep track of this for you, recompile the source with `MISC_SLOW_BUT_SAFE` defined. (It is commented out in the source file; just uncomment it and recompile.) If you never have very many objects in the recycler at any given time, you won't take much of a performance hit. By defining this, however, you change the algorithm for recycling objects from O(1) to O(n) where *n* is the number of objects in the recycler.

Method Types

Controlling how objects are re-initialized:	**;MiscObjectRecycler.rtf;firstInitObject:;¬+** firstInitObject: **;MiscObjectRecycler.rtf;reInitObject:;¬+** reInitObject:
Obtaining a Recycled Object:	**;MiscObjectRecycler.rtf;newFromRecycler;¬+** newFromRecycler
Obtaining a Specific Recycler:	**;MiscObjectRecycler.rtf;recycler;¬+** recycler **;MiscObjectRecycler.rtf;recyclerForClass:;¬+** recyclerForClass: **;MiscObjectRecycler.rtf;recyclerForClassName:;¬+** recyclerForClassName:
Recycling an Object:	**;MiscObjectRecycler.rtf;recycle;¬-** recycle

Class Methods

firstInitObject:;¬ firstInitObject:
+ **firstInitObject:***anObject*

This method is used to initialize a newly created object which was created if the recycler was empty. This method by default sends ±**init** to the object in question; if initialization should take place differently, then this

method should be overridden to do the right thing. Returns whatever sending ±**init** to *anObject* would return.

See also: ;MiscObjectRecycler.rtf;reInitObject:;¬+reInitObject: and
;MiscObjectRecycler.rtf;newFromRecycler;¬+newFromRecycler:

newFromRecycler;¬newFromRecycler
+ **newFromRecycler**

Removes an object from the receiving class' recycler and re-initializes it, returning the object. If the recycler is empty, a new instance is allocated and initialized. The +**reInitObject** and +**firstInitObject** methods are used to perform the respective initializations. Whatever those methods return is returned by +**newFromRecycler**. Normally this is the id of the recycled (or new) object, but may occasionally be nil, depending upon the class.

See also: ;MiscObjectRecycler.rtf;recycle;¬±recycle,
;MiscObjectRecycler.rtf;reInitObject:;¬+reInitObject: and
;MiscObjectRecycler.rtf;firstInitObject:;¬+firstInitObject:

reInitObject:;¬reInitObject:
+ **reInitObject:***anObject*

This method is used to re-initialize an object after removing it from the recycler. This method by default sends ±**init** to the object in question; if re-initialization should take place differently, then this method should be overridden to do the right thing. Returns whatever sending ±**init** to *anObject* would return.

See also: ;MiscObjectRecycler.rtf;firstInitObject:;¬+firstInitObject: and
;MiscObjectRecycler.rtf;newFromRecycler;¬+newFromRecycler:

recycler;¬recycler
+ **recycler**

Returns the recycler (a List object) for the receiving class object.

See also: **;MiscObjectRecycler.rtf;recyclerForClass:;¬+recyclerForClass:** and **;MiscObjectRecycler.rtf;recyclerForClassName:;¬+recyclerForClassName:**

recyclerForClass:;¬recyclerForClass:

+ **recyclerForClass:***aClass*

Returns the recycler (a List object) for the class *aClass*.

See also: **;MiscObjectRecycler.rtf;recycler;¬+recycler** and **;MiscObjectRecycler.rtf;recyclerForClassName:;¬+recyclerForClassName:**

recyclerForClassName:;¬recyclerForClassName:

+ **recyclerForClassName:**(const char *)*className*

Returns the recycler (a List object) for the class named *className*.

See also: **;MiscObjectRecycler.rtf;recycler;¬+recycler** and **;MiscObjectRecycler.rtf;recyclerForClass:;¬+recyclerForClass:**

Instance Methods

recycle;¬recycle

- **recycle**

Places the receiver into the recycler. No messages should be sent to the receiver after this message until after the recycler has re-initialized and released it. Returns nil. The safest invocation of this method (and ±**free** as well) looks like this:

```
anObject = [anObject recycle];
```

See also: **;MiscObjectRecycler.rtf;newFromRecycler;¬+newFromRecycler:**