SharpMedia Caching Design

© 2007 Žiga Osolin – SharpMedia project, document version: DRAFT

Table Of Contents

[About 2](#_Toc167628795)

[Goals 2](#_Toc167628796)

[Interfaces 2](#_Toc167628797)

[ICache<T> 2](#_Toc167628798)

[ICacheable 2](#_Toc167628799)

[IEvaluator<T> 3](#_Toc167628800)

[Implementation Notes 4](#_Toc167628801)

[Usage Cases 4](#_Toc167628802)

# About

Caching is a key to improve performance of access to resources. It is expected that commonly used resource will be needed many times and is thus put in cache. This document outlines the caching as a programmatic procedure to keep certain object referenced even when they are not needed by the process (but *may* be needed in future).

# Goals

The cache library should be:

* Generic, applicable to all problems that needed data caching;
* Compatible with GC;
* Fast;
* Configurable (with many different evaluators, least-recently used being the most common);
* Multi-layered;
* Type-safe.

Unresolved:

* Multi-layer caches;
* Additional evaluator types.

# Interfaces

The following interfaces are part of caching:

## ICache<T>

An **ICache** is a cache controller object. This object exposes methods for explicit eviction, additions of new cacheable objects and searches for objects. The type **T** is the search key, used when adding new elements, removing them or searching for them. A key-cacheable pair always stays the same. The only way to change it is using the rebind method, which assigns new key to cacheable element. A key **must** be globally unique.

Cache must be updated. Only when updated can the data be freed. Usually, updating is done automatically by subscribing a delegate to **ThreadPool**, using a **System.Threading.Timer** class.

## ICacheable

This interface extends **IDisposable** interface and implies that any cacheable element can be evicted, e.g. released from cache. After this is called, no-one can gain reference to object through **ICache<T>**. A cacheable has a special method, **Touch** that touches the cacheable, e.g. signals that it is being used. A special event is also introduced to be fired on touching (it also updates ICache). On evicting, this event must be cleared.

A cacheable is **not** automatically touched when it is found by cache; this is because it may not be used. If it will definitely be touched, use **ICache<T>.FindAndTouch**.

An ICacheable can be in the following states:

* Normal;
* Evicted;
* Disposed.

The following rules apply:

1. ICachable can only reside in one cache;
2. On eviction, cache goes from normal to evicted state. If already in disposed state, this is ignored;
3. Evicted resource can be disposed or used in any other way but it is not a member of cache anymore (touch has no effect);
4. ICacheable goes to normal state if added to new cache (previous state must be evicted);
5. Disposed objects can go to any other state only through their own logic.

## IEvaluator<T>

Evaluator is a base class for all evaluators. It has the following methods:

* **Touch**, issued when resource is touched;
* **Update**, issued when cache issues an update of cacheable’s score;
* **Data**, issued when new data is added to evaluator (this is per-evaluator data), can return null.

The following evaluators must be implemented:

* LRU (least recently used), adds score when touched and subtracts score when updated, based on time interval;
* Use once evaluator (automatically evicts from cache on first update);
* …

# Implementation Notes

# Usage Cases