EOF-Using EOF-Delegates and Notifications

EO Change Notifications

Pierre Bernard

To invalidate cached versions of derived values, you might want to use this:

```
/** Utility class. Provides for a way to watch for changes in EOs. The main use of this would be to
  * be able to safely keep cached/computed/derived values that get cleared once out of sync.
  *
  * @author bernard
  * @version ChangeNotificationCenter.java,v 1.1 2003/03/11 16:26:59 bernard Exp
**/
public class ChangeNotificationCenter
{
  // Public class constants

  /** Name of the posted notification
   */
  public static final String OBJECT_CHANGED = "MBSObjectChanged";

  /** Possible type of change a notification is posted for
   */
  public static final String INVALIDATION = "Invalidation";

  /** Possible type of change a notification is posted for
   */
  public static final String DELETION = "Deletion";

  /** Possible type of change a notification is posted for
   */
  public static final String UPDATE = "Update";

  // Private class constants

  /** Key in the posted notification's userInfo dictionary.
   */
  private static final String OBJECT = "MBSObject";

  /** Key in the posted notification's userInfo dictionary.
   */
  private static final String TYPE = "MBSType";

  /** Selector used for calling the watcher object upon a change
   */
  private static final NSSelector watchedObjectChangedSelector =
    new NSSelector("watchedObjectChanged", new Class[] { NSNotification.class });

  // Private class variables

  /** Reference holding the shared singleton instance.
   */
  private static ChangeNotificationCenter defaultCenter = null;

  // Constructor

  /** Singleton class
   */
  private ChangeNotificationCenter()
  {
    super();

    Class[] notificationArray = new Class[] { NSNotification.class };
    NSSelector objectsChangedIInEditingContextSelector =
      new NSSelector("objectsChangedInEditingContext", notificationArray);
```

```
    NSNotificationCenter.defaultCenter().addObserver(
      this,
      objectsChangedIInEditingContextSelector,
      EOEditingContext.ObjectsChangedInEditingContextNotification,
      null);
  }

  // Public insatnce methods

  /** Method called when a change notification is received.
   *
   * The notification is split and redispatched for all updated objects.
   */
  public void objectsChangedInEditingContext(NSNotification notification)
  {
    NSArray updated = (NSArray) notification.userInfo().objectForKey(EOObjectStore.UpdatedKey);

    if (updated != null)
    {
      int count = updated.count();

      for (int i = 0; i < count; i++)
      {
        Object object = updated.objectAtIndex(i);
        NSDictionary userInfo =
          new NSDictionary(new Object[] { object, UPDATE }, new Object[] { OBJECT, TYPE });
        NSNotificationCenter.defaultCenter().postNotification(OBJECT_CHANGED, object, userInfo);
      }
    }

    NSArray invalidated = (NSArray) notification.userInfo().objectForKey(EOObjectStore.InvalidatedKey);

    if (invalidated != null)
    {
      int count = invalidated.count();

      for (int i = 0; i < count; i++)
      {
        Object object = invalidated.objectAtIndex(i);
        NSDictionary userInfo =
          new NSDictionary(new Object[] { object, INVALIDATION }, new Object[] { OBJECT, TYPE });
        NSNotificationCenter.defaultCenter().postNotification(OBJECT_CHANGED, object, userInfo);
      }
    }

    NSArray deleted = (NSArray) notification.userInfo().objectForKey(EOObjectStore.DeletedKey);

    if (deleted != null)
    {
      int count = deleted.count();

      for (int i = 0; i < count; i++)
      {
        Object object = deleted.objectAtIndex(i);
        NSDictionary userInfo = new NSDictionary(new Object[] { object, DELETION }, new Object[] { OBJECT, TYPE
});
        NSNotificationCenter.defaultCenter().postNotification(OBJECT_CHANGED, object, userInfo);
      }
    }
  }

  /** Method to be called when one creates a cached value that depends on the attributes of a given object.
   *
   * Upon registration the object is watched and the watchedObjectChanged() is called once it changes, thus
providing an
   * oppurtunity to invalidate the cached value.
   *
   * @param watcher the watching object which should unregister before disposing
   * @param object the object to watch
   * @see #unregisterCacheDependancy
   * @see #unregisterCacheDependancies
```

```
   */
  public void registerCacheDependancy(ObserverInterface watcher, EOEnterpriseObject object)
  {
    NSNotificationCenter.defaultCenter().addObserver(watcher, watchedObjectChangedSelector, OBJECT_CHANGED,
object);
  }

  /** Method to be called when one abandons or clears a cached value that depended on attributes of a given
object.
   *
   * @param watcher the watching object
   * @param object the object to watch
   * @see #registerCacheDependancy
   */
  public void unregisterCacheDependancy(ObserverInterface watcher, Object object)
  {
    NSNotificationCenter.defaultCenter().removeObserver(watcher, OBJECT_CHANGED, object);
  }

  /** Unregisters all of the callers dependancies. To be used sparingly as it may break unknown but required
   * dependancies. Usually called when disposing the calling object.
   *
   * @param watcher the watching object
   * @see #registerCacheDependancy
   * @see #unregisterCacheDependancy
   */
  public void unregisterCacheDependancies(ObserverInterface watcher)
  {
    NSNotificationCenter.defaultCenter().removeObserver(watcher, OBJECT_CHANGED, null);
  }

  // Public class methods

  /** Gets or lazily instantiates the shared instance of the ChangeNotificationCenter
   *
   * @return the unique instance
   */
  public static ChangeNotificationCenter defaultCenter()
  {
    if (defaultCenter == null)
    {
      createDefaultCenter();
    }

    return defaultCenter;
  }

  /** Utility method. Allows for retrieving the changed object from a notification
   * that results of registering interest in a change.
   *
   * @param notification the received notification
   * @return the object that triggered the notification
   */
  public static EOEnterpriseObject objectFromNotification(NSNotification notification)
  {
    return (EOEnterpriseObject) notification.userInfo().objectForKey(OBJECT);
  }

  /** Utility method. Allows for retrieving the type of change that ocurred from a notification
   * that results of registering interest in a change.
   *
   * @param notification the received notification
   * @return the type as one of the class constants defined in ChangeNotificationCenter
   */
  public static String typeFromNotification(NSNotification notification)
  {
    return (String) notification.userInfo().objectForKey(TYPE);
  }

  // Private class methods
```

```
  /** Creates the singleton instance ensuring there is no existing instance.
   */
  private static synchronized void createDefaultCenter()
  {
    if (defaultCenter == null)
    {
      defaultCenter = new ChangeNotificationCenter();
    }
  }

  // Inner interface definition

  /** Interface to be implemented by objects that need to listen to changes.
   *
   * CAVEAT: in most cases it is recommended to not directly implement the interface,
   * but rather create an inner class that implements the interface or better yet
   * extends the default implementation. Indeed if an object (e.g. an EO) which
   * participates in an inheritance hierarchy is used as receiver for notifications,
   * registering or unregistering might break functionality in a parent class.
   */
  public static interface ObserverInterface
  {
    /** Method called when an object on which locally cached values depend is modified.
     *
     * This hook is provided as an opportunity to clear locally cached values. It's a good idea not to recreate
     * the cached values immediately, but on an as-needed basis. You should refrain from changing persisted EO
     * attributes from within this method as this might kick off another chain of notifications.
     *
     * Once the caches cleared, it would be a very good idea to unregister from further notifications until
     * the cache is recreated.
     *
     * @see lu.bcl.enterprise.entity.ChangeNotificationCenter#objectFromNotification
     * @see lu.bcl.enterprise.entity.ChangeNotificationCenter#registerCacheDependancy
     * @see lu.bcl.enterprise.entity.ChangeNotificationCenter#unRegisterCacheDependancy
     */
    public void watchedObjectChanged(NSNotification notification);
  }

  // Inner class

  /** Convenience class for implementing ObserverInterface.
   *
   * The recommended way of registering for change notificications is to create an inner
   * class extending this one.
   */
  public abstract static class Observer implements ObserverInterface
  {
    /** Method to be called by subclasses when the create a cached value that depends on the attributes of a
given object.
     *
     * Upon registration the object is watched and the clearCachedValues() is called once it changes, thus
providing an
     * oppurtunity to invalidate the cached value.
     *
     * You need to register for each object your locally cached values depend on. E.g. if you build a cached
value from
     * attributes of EOs in a too-many relationship, your cache depends on each of the objects in the
relationship as well
     * as on the source of the realtionship.
     *
     * N.B. Extend the dispose() method to unregister any dependancy you might be registered for.
     *
     * @param object the object to watch
     * @see #unregisterCacheDependancy
     * @see #clearCachedValues
     */
    protected void registerCacheDependancy(EOEnterpriseObject object)
    {
      ChangeNotificationCenter.defaultCenter().registerCacheDependancy(this, object);
    }
```

```
    /** Method to be called by subclasses when they abandon or clear a cached value that depended on attributes
     * of a given object.
     *
     * @param object the object to watch
     * @see #registerCacheDependancy
     * @see #clearCachedValues
     */
    protected void unregisterCacheDependancy(EOEnterpriseObject object)
    {
      ChangeNotificationCenter.defaultCenter().unregisterCacheDependancy(this, object);
    }

    /** Unregisters all of the callers dependancies. To be used sparingly as it may break unknown but required
     * dependancies. Usually called when disposing the calling object.
     *
     * @see #registerCacheDependancy
     * @see #unregisterCacheDependancy
     */
    public void unregisterCacheDependancies()
    {
      ChangeNotificationCenter.defaultCenter().unregisterCacheDependancies(this);
    }

    /** Overridden to unregister all cache dependancies
     */
    public void finalize() throws Throwable
    {
      ChangeNotificationCenter.defaultCenter().unregisterCacheDependancies(this);

      super.finalize();
    }
  }
}
```