

Unit Of Work

Overview

The **Unit Of Work** pattern is a common pattern in enterprise application architectures.

It can be used for implementing:

- transparent persistence
- replication
- clustering/caching
- transaction demarcation
- etc.

It implements the concept of an "application transaction" that keeps track of:

- *dirty* objects
- *new* objects
- *removed* objects

It records the state of the objects, by tracking field changes, as it is before the transaction is started and will then rollback the state in memory upon rollback of the transaction. It allows adding additional behavior at *commit*, *rollback* etc., for example store all new, update all dirty and remove all removed objects in a persistent storage.

Fowler defines the Unit Of Work pattern like this in his book [Patterns of Enterprise Application Architecture](#):

"Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems."

This implementation implements the following transaction semantics:

- *TX SUPPORTS*
- *TX MANDATORY*
- *TX NEVER*
- *TX REQUIRES*
- *TX REQUIRES NEW*

UnitOfWorkListener

The user can implement listeners that allows him to trigger additional behaviour on specific events, such as *begin*, *commit*, *rollback*, *dispose* etc.

All listeners must implement that `org.codehaus aware.unitofwork.UnitOfWorkListener` interface.

The Unit Of Work will not commit until all listeners have agreed on that. If for example on listener have set the transaction to *rollbackOnly* then the whole Unit Of Work transaction will rollback.

```
public interface UnitOfWorkListener {  
    /**  
     * Initializes the UnitOfWork listener
```

by passing in the UnitOfWork for this listener.

```
    *
    * @param unitOfWork the UnitOfWork
instance
    */
    void initialize(UnitOfWork unitOfWork);

    /**
    * Checks if the UnitOfWork is set to
rollback only
    *
    * @return boolean
    */
    boolean isRollbackOnly();

    /**
    * Is invoked when the UnitOfWork is
started.
    */
    void doBegin();

    /**
    * Is invoked when the UnitOfWork is
committed.
    */
    void doCommit();

    /**
    * Is invoked right before the
UnitOfWork is committed.
```

```
    */
    void doPreCommit();

    /**
     * Is invoked right after the UnitOfWork
    is committed.
     */
    void doPostCommit();

    /**
     * Is invoked when the UnitOfWork is
    aborted.
     */
    void doRollback();

    /**
     * Is invoked when the UnitOfWork is
    aborted.
     */
    void doDispose();

    /**
     * Adds an object to the ID map.
     *
     * @param obj the object to add to the
    ID map
     */
    void addToIdMap(Object obj);

    /**
     * Removes an object from the ID map.
```

*

* @param obj the object to remove from
the ID map

```
    */  
    void removeFromIdMap(Object obj);  
}
```

We also provide a convenience abstract adapter class that you can use if you are only interested in implementing a couple of methods and don't want to have empty method definitions. Just extend the `org.codehaus.aware.unitofwork.UnitOfWorkAdapter` class.

Definition

The listeners are added to the Unit Of Work in the `initialize` method:

```
UnitOfWork.initialize(new  
    Class[] {HibernateUnitOfWorkListener.class,  
    JtaUnitOfWorkListener.class});
```

But in practice it is usually best to define this information outside the code, in a configuration file. Such as in the [HibernateUnitOfWorkListener](#) implementation, which uses the [SpringAspectContainer](#) to configure the UnitOfWork.