# Business Objects With Updatable Collections 2

This page describes the process for correctly implementing a Business Object with updatable collections.  An updatable collection is a collection of a BO that will persist changes made to the collection when the parent BO is saved.

For this page, the word "delete" will mean the deletion of a collection element from the database (i.e. running a SQL DELETE statement).  The word "remove" will mean the removal of an element from a collection on a browser.  For example, when a user is presented with transactional document with a collection of items, the user may choose to remove an item before saving a document.  Note that the button on the document may be marked with the word "delete", but this differentiation of terminology will make this discussion terse.  In other words, "remove" refers to an act of the user, and "delete" refers to a database action.  Removal from the form should result in deletion from the database.

Because of Rice's request-based architecture, when a form is submitted from the user to the server, we are only able to determine which elements are on the form.  In particular, by looking at the request parameters alone, we can only know which elements are in a collection and are unable to determine whether the user removed an element from a collection.  Because of this, OJB natively takes care of updating or adding items to a collection, but is unable to automatically determine which elements to delete from the database.

To implement this functionality, Rice retrieves the BO from the database and compares it with the BO from the form. It deletes any element BOs that exist in the database BO and not in the form BO (because it presumably has been removed). This page describes the process to enable OJB deletion of removed elements from a collection for a particular BO.

This process consists of two steps:

1. Defining which updatable collections have deletable elements
2. Implementing removed element deletion code if necessary

## Defining which updatable collections have deletable elements

`org.kuali.core.bo.PersistableBusinessObject.buildListOfDeletionAwareLists()` should be implemented for all business objects that require deletion of removed elements.

In a nutshell, this method returns a **`List of Collection`** of a BO. It returns a list of references to the actual collections storing data. Given a BO from the form to persist, it retrieves the corresponding BO from the database (based on primary key). Compares the collections returned by `buildListOfDeletionAwareLists()`, and deletes any elements contained within the DB-based BO and not in the form-based BO.

For each BO class, its implementation of `buildListOfDeletionAwareLists()` should return a list of equal size, and each element of the list must represent the same property of the BO every time it is called. Note that this does not imply that all BOs (regardless of class) must return a list with the same size. It implies that, for a particular BO class and primary key combination, it always has to return a list of the same size and matching collections.

The following is an example implementation for accounting documents, which are business objects. Most accounting documents contain 3 updatable collections. Two of them, the source and target accounting line collections, must be able to delete accounting lines from the collection. Most of them also contain a list of General Ledger Pending Entries. These are dynamically generated by the accounting framework and not directly editable by users, so we don't need to enable deletion of removed elements.

**From AccountingDocumentBase**

```
public List buildListOfDeletionAwareLists() {
    List managedLists = super.buildListOfDeletionAwareLists();

    managedLists.add(getSourceAccountingLines());
    managedLists.add(getTargetAccountingLines());

    return managedLists;
}
```

This implementation says that the source and target accounting line lists should delete elements that are removed for a particular business object. Unless overridden, note that calling this method will always return a list of 2 collections, the first element is the source accounting lines collection, and the second is the target accounting lines collection.

By default, the `buildListOfDeletionAwareLists()` will only be called upon the top level BO. If you are calling save on a BO, then you must define `buildListOfDeletionAwareLists()` on that BO, even if the deletable collection is a property of a reference object. In this case, the top-level BO would return a list of collections that include the collections of its reference object.

**Deleting Elements Reference Collections**

```
public List buildListOfDeletionAwareLists() {
    List managedLists = super.buildListOfDeletionAwareLists();

    managedLists.add(getReferenceBusinessObject().getSomeCollection());

    return managedLists;
}
```

This method does not easily handle collections nested within collections. To do so, consult `PurApOjbCollectionHelper` to see how PURAP documents accomplish this.

## Implementing removed element deletion code if necessary

If the BO containing deletable collections will be persisted using a Spring bean of types `BusinessObjectDaoOjb` or `DocumentDaoOjb`, then your work is done. DAOs should only be called by services. For higher level code (e.g. struts actions), use `BusinessObjectService` or `DocumentService`, which delegate to these DAOs. There may be other services that use these DAOs as well.

However, if it is not possible to directly or indirectly use these DAOs, then the developer should implement a new DAO.

The steps for doing so are as follows:

1. Have the DAO class implement `org.kuali.core.util.OjbCollectionAware`
2. In the method that saves/updates the business object (e.g. save), retrieve the corresponding business object from the database with the same primary key values.
3. Retrieve the `org.kuali.core.util.OjbCollectionHelper` bean from the Spring bean factory. In rice, use `KNSServiceLocator.getOjbCollectionHelper()` and in KFS or Rice client code, use `SpringContext.getBean(OjbCollectionHelper.class)`.
4. Call `org.kuali.core.util.OjbCollectionHelper.processCollections(OjbCollectionAware theDAO, PersistableBusinessObject boFromForm, PersistableBusinessObject boFromDatabase)`
5. Store the BO from the form as normal (typically by calling `getPersistenceBrokerTemplate().store(document)`).

Example from `DocumentDaoOjb`:

```
    public void save(Document document) throws DataAccessException {
        Document retrievedDocument =
findByDocumentHeaderId(document.getClass(),document.getDocumentNumber());
        KNSServiceLocator.getOjbCollectionHelper().processCollections(this, document,
retrievedDocument);
        this.getPersistenceBrokerTemplate().store(document);
    }
```