Process for updating orders in ATG commerce code (Doc ID 1362812.1)

Modified: 14-Feb-2013 Type: HOWTO

In this Document

APPLIES TO:

Oracle ATG Web Commerce - Version 2006.3 and later
Information in this document applies to any platform.

GOAL

How to Update orders in ATG commerce code.

FIX

When you are using the Order object with synchronization and transactions, there is a specific usage pattern that is critical to follow. Not following the expected pattern can lead to unnecessary ConcurrentUpdateExceptions, InvalidVersionExceptions, and deadlocks. The following sequence must be strictly adhered to in your code:

1. Obtain local-lock on profile ID.
2. Begin Transaction
3. Synchronize on Order
4. Perform ALL modifications to the order object.
5. Call OrderManager.updateOrder.
6. End Synchronization
7. End Transaction.
8. Release local-lock on profile ID.

Steps 1, 2, 7, 8 are done for you in the beforeSet() and afterSet() methods for ATG form handlers where order updates are expected. These include form handlers that extend PurchaseProcessFormHandler and OrderModifierFormHandler (deprecated). If your code accesses/modifies the order outside of a PurchaseProcessFormHandler, it will likely need to obtain the local-lock manually. The lock fetching can be done using the TransactionLockService.

So, if you have extended an ATG form handler based on PurchaseProcessFormHandler, and have written custom code in a handleXXX() method that updates an order, your code should look like:

```
synchronized( order )
{
  // Do order updates
  orderManager.updateOrder( order );
}
```

If you have written custom code updating an order outside of a PurchaseProcessFormHandler (e.g. CouponFormHandler, droplet, pipeline servlet, fulfillment-related), your code should look like:

```
ClientLockManager lockManager = getLocalLockManager(); // Should be configured as
/atg/commerce/order/LocalLockManager
boolean acquireLock = false;
try
```

```
{
  acquireLock = !lockManager.hasWriteLock( profileId, Thread.currentThread() );
  if ( acquireLock )
    lockManager.acquireWriteLock( profileId, Thread.currentThread() );

  TransactionDemarcation td = new TransactionDemarcation();
  td.begin( transactionManager );
  boolean shouldRollback = false;
  try
  {
    synchronized( order )
    {
      // do order updates
      orderManager.updateOrder( order );
    }
  }
  catch ( ... e )
  {
    shouldRollback = true;
    throw e;
  }
  finally
  {
    try
    {
      td.end( shouldRollback );
    }
    catch ( Throwable th )
    {
      logError( th );
    }
  }
}
finally
{
  try
  {
    if ( acquireLock )
      lockManager.releaseWriteLock( profileId, Thread.currentThread(), true );
  }
  catch( Throwable th )
  {
    logError( th );
  }
}
```

This pattern is only useful to prevent ConcurrentUpdateExceptions, InvalidVersionExceptions, and deadlocks when multiple threads attempt to update the same order on the *same* ATG instance. This should be adequate for most situations on a commerce site since session stickiness will confine updates to the same order to the same ATG instance.

Legacy ATG Community DOC-1888

REFERENCES

NOTE:1362761.1 - Managing Transactions Involving the ATG Commerce Order Object