# Analyse details

## Class AbstractEntityTuplizer

- the method setPropertyValues(Object entity, Object[] values) ist called 3 times until values[17] contains the HistoryContext ([17] "HIST_de.icongmbh.dope.flow.bo.TicketIF" (id=237)
  )

```
    @Override
    public void setPropertyValues(Object entity, Object[] values) throws
HibernateException {
        boolean setAll = !entityMetamodel.hasLazyProperties();

        final SessionFactoryImplementor factory = getFactory();
        for ( int j = 0; j < entityMetamodel.getPropertySpan(); j++ ) {
            if ( setAll || values[j] !=
LazyPropertyInitializer.UNFETCHED_PROPERTY ) {
                setters[j].set( entity, values[j], factory );
            }
        }
    }
```

## Call Stack

- AbsObject.setHistoryContext(final String historyContext)

```
   this.historyContext = historyContext;
```

- AbstractEntityTuplizer.setPropertyValues(Object entity, Object[] values)
- PojoEntityTuplizer.setPropertyValues(Object entity, Object[] values)
- AbstractEntityPersister.setPropertyValues(Object object, Object[] values)
- TwoPhaseLoad.initializeEntityFromEntityEntryLoadedState(final Object entity,final EntityEntry entityEntry,final boolean readOnly,final SharedSessionContractImplementor session,final,PreLoadEvent preLoadEvent,final Iterable preLoadEventListeners)

```
        final Object[] hydratedState = entityEntry.getLoadedState();
        persister.setPropertyValues( entity, hydratedState );  <---------
hydratedState  == values[]
```

- TwoPhaseLoad.initializeEntity(final Object entity,final boolean readOnly,final SharedSessionContractImplementor session,final PreLoadEvent preLoadEvent,final Iterable preLoadEventListeners,final EntityResolver entityResolver)

```
        final PersistenceContext persistenceContext =
session.getPersistenceContextInternal(); <-------------- session???
        final EntityEntry entityEntry = persistenceContext.getEntry( entity );
        if ( entityEntry == null ) {
            throw new AssertionFailure( "possible non-threadsafe access to the
session" );
        }
        initializeEntityEntryLoadedState( entity, entityEntry, session,
entityResolver );
        initializeEntityFromEntityEntryLoadedState( entity, entityEntry,
readOnly, session, preLoadEvent, preLoadEventListeners );
```

- TwoPhaseLoad.initializeEntity(final Object entity, final boolean readOnly,final SharedSessionContractImplementor session,final PreLoadEvent preLoadEvent,final Iterable preLoadEventListeners)

```
{
        initializeEntity( entity, readOnly, session, preLoadEvent,
preLoadEventListeners, EntityResolver.DEFAULT );
    }
```

- AbstractRowReader.performTwoPhaseLoad(PreLoadEvent preLoadEvent,ResultSetProcessingContextImpl context,List hydratedEntityRegistrations)

```
        final SharedSessionContractImplementor session = context.getSession();
<------------------------- context???
        for ( HydratedEntityRegistration registration :
hydratedEntityRegistrations ) {
            TwoPhaseLoad.initializeEntity(
                    registration.getInstance(),
                    context.isReadOnly(),
                    session,
                    preLoadEvent,
                    listeners
            );
        }
```

- AbstractRowReader.finishUp(ResultSetProcessingContextImpl context, List afterLoadActionList)

```
    performTwoPhaseLoad( preLoadEvent, context, hydratedEntityRegistrations );
```

- ResultSetProcessrImpl.extractResults(ResultSet resultSet,final SharedSessionContractImplementor session,QueryParameters queryParameters,NamedParameterContext namedParameterContext,boolean returnProxies,boolean readOnly,ResultTransformer forcedResultTransformer,List afterLoadActionList)

```
    final ResultSetProcessingContextImpl context =
createResultSetProcessingContext(
        resultSet,
        session,
        queryParameters,
        namedParameterContext,
        returnProxies,
        readOnly
    );
    rowReader.finishUp( context, afterLoadActionList );
```

# Trace

## Class AbsObject

public void setHistoryContext(final String historyContext) {
this.historyContext = historyContext; <---------- ist Null!!
}

## class AbstractEntityTuplizer

```
   public void setPropertyValues(Object entity, Object[] values) throws
HibernateException {
    boolean setAll = !entityMetamodel.hasLazyProperties();

    final SessionFactoryImplementor factory = getFactory();
    for ( int j = 0; j < entityMetamodel.getPropertySpan(); j++ ) {
        if ( setAll || values[j] != LazyPropertyInitializer.UNFETCHED_PROPERTY
) {
            setters[j].set( entity, values[j], factory ); <--------------------
--------------------------- values[j] ???
        }
    }
   }
```

```
no method return value
this    PojoEntityTuplizer  (id=148)
entity  Ticket  (id=256)
    additionalDatas HashSet<E>  (id=273)
    administrationState "smartadmin" (id=189)
    audit   Audit  (id=274)
    currentTask null
    currentTask null
    currentTaskOid  null
    deleted false
    externalID  null
    extraInfo   null
    historyContext  null
    mergedObjects   HashSet<E>  (id=275)
    oid ObjectUID  (id=196)
    predecessor null
    prio    2147483647
    processState    ProcessState  (id=200)
    processType null
    processWorkbasket   null
    releaseDateObjects  HashSet<E>  (id=276)
    requirement null
    stagingTimestamp    null
    supervisor  null
    tenantId    "DOPiX" (id=206)
    testAnnotations HashSet<E>  (id=277)
    transitionCounter   0
    version null
values  Object[22]  (id=257)
    [0] Integer  (id=192)
    [1] Audit  (id=270)
    [2] Boolean  (id=271)
    [3] null
    [4] null
    [5] "saved ticket" (id=203)
    [6] null
    [7] Integer  (id=192)
    [8] "DOPiX" (id=206)
    [9] null
    [10]    null
    [11]    null
    [12]    Integer  (id=272)
    [13]    null
    [14]    null
    [15]    null
    [16]    "smartadmin" (id=189)
    [17]    "HIST_de.icongmbh.dope.flow.bo.TicketIF" (id=278)   <--------------
--------------- HistoryContext
    [18]    PersistentSet  (id=279)
    [19]    PersistentSet  (id=280)
    [20]    PersistentSet  (id=292)
    [21]    PersistentSet  (id=293)
```

```
setAll  true
```

## class PojoEntityTuplizer

```
    @Override
    public void setPropertyValues(Object entity, Object[] values) throws
HibernateException {
        if ( !getEntityMetamodel().hasLazyProperties() && optimizer != null &&
optimizer.getAccessOptimizer() != null ) {
            setPropertyValuesWithOptimizer( entity, values );
        }
        else {
            super.setPropertyValues( entity, values ); <----------------------
-------------------------
        }
    }
```

```
this    PojoEntityTuplizer  (id=148)
entity  Ticket  (id=256)
    additionalDatas HashSet<E>  (id=273)
    administrationState "smartadmin" (id=189)
    audit   Audit  (id=274)
    currentTask null
    currentTask null
    currentTaskOid  null
    deleted false
    externalID  null
    extraInfo   null
    historyContext  null    <-----------------------------------------
    mergedObjects   HashSet<E>  (id=275)
    oid ObjectUID  (id=196)
    predecessor null
    prio    2147483647
    processState    ProcessState  (id=200)
    processType null
    processWorkbasket   null
    releaseDateObjects  HashSet<E>  (id=276)
    requirement null
    stagingTimestamp    null
    supervisor  null
    tenantId    "DOPiX" (id=206)
    testAnnotations HashSet<E>  (id=277)
    transitionCounter   0
    version null
values  Object[22]  (id=257)
    [0] Integer  (id=192)
    [1] Audit  (id=270)
```

```
    [2] Boolean  (id=271)
    [3] null
    [4] null
    [5] "saved ticket" (id=203)
    [6] null
    [7] Integer  (id=192)
    [8] "DOPiX" (id=206)
    [9] null
    [10]    null
    [11]    null
    [12]    Integer  (id=272)
    [13]    null
    [14]    null
    [15]    null
    [16]    "smartadmin" (id=189)
    [17]    "HIST_de.icongmbh.dope.flow.bo.TicketIF" (id=278)   <--------------
----------------------------
    [18]    PersistentSet  (id=279)
    [19]    PersistentSet  (id=280)
    [20]    PersistentSet  (id=292)
    [21]    PersistentSet  (id=293)
```

## class AbstractEntityPersister

```
    public void setPropertyValues(Object object, Object[] values) {
        getEntityTuplizer().setPropertyValues( object, values ); <-------------
-------------
    }
```

## class AbstractSaveEventListener

```
    /**
     * Performs all the actual work needed to save an entity (well to get the
save moved to
     * the execution queue).
     *
     * @param entity The entity to be saved
     * @param key The id to be used for saving the entity (or null, in the case
of identity columns)
     * @param persister The entity's persister instance.
     * @param useIdentityColumn Should an identity column be used for id
generation?
     * @param anything Generally cascade-specific information.
     * @param source The session which is the source of the current event.
     * @param requiresImmediateIdAccess Is access to the identifier required
immediately
```

```
     * after the completion of the save?  persist(), for example, does not
require this...
     *
     * @return The id used to save the entity; may be null depending on the
     *         type of id generator used and the requiresImmediateIdAccess
value
     */
    protected Serializable performSaveOrReplicate(
            Object entity,
            EntityKey key,
            EntityPersister persister, <-----------------------------------
???
            boolean useIdentityColumn,
            Object anything,
            EventSource source,
            boolean requiresImmediateIdAccess) {

        Serializable id = key == null ? null : key.getIdentifier();

        boolean inTrx = source.isTransactionInProgress();
        boolean shouldDelayIdentityInserts = !inTrx &&
!requiresImmediateIdAccess;
        final PersistenceContext persistenceContext =
source.getPersistenceContextInternal();

        // Put a placeholder in entries, so we don't recurse back and try to
save() the
        // same object again. QUESTION: should this be done before onSave() is
called?
        // likewise, should it be done before onUpdate()?
        EntityEntry original = persistenceContext.addEntry(
                entity,
                Status.SAVING,
                null,
                null,
                id,
                null,
                LockMode.WRITE,
                useIdentityColumn,
                persister, <-------------------------------------------------
-------
                false
        );

        cascadeBeforeSave( source, persister, entity, anything );

        Object[] values = persister.getPropertyValuesToInsert( entity,
getMergeMap( anything ), source ); <------------------------------
        Type[] types = persister.getPropertyTypes();

        boolean substitute = substituteValuesIfNecessary( entity, id, values,
persister, source );
```

```
        if ( persister.hasCollections() ) {
            substitute = visitCollectionsBeforeSave( entity, id, values, types,
source ) || substitute;
        }

        if ( substitute ) {
            persister.setPropertyValues( entity, values );  <------------------
-------------------------------------------------
        }

        TypeHelper.deepCopy(
                values,
                types,
                persister.getPropertyUpdateability(),
                values,
                source
        );

        AbstractEntityInsertAction insert = addInsertAction(
                values, id, entity, persister, useIdentityColumn, source,
shouldDelayIdentityInserts
        );

        // postpone initializing id in case the insert has non-nullable
transient dependencies
        // that are not resolved until cascadeAfterSave() is executed
        cascadeAfterSave( source, persister, entity, anything );
        if ( useIdentityColumn && insert.isEarlyInsert() ) {
            if ( !EntityIdentityInsertAction.class.isInstance( insert ) ) {
                throw new IllegalStateException(
                        "Insert should be using an identity column, but action
is of unexpected type: " +
                                insert.getClass().getName()
                );
            }
            id = ((EntityIdentityInsertAction) insert).getGeneratedId();

            insert.handleNaturalIdPostSaveNotifications( id );
        }

        EntityEntry newEntry = persistenceContext.getEntry( entity );

        if ( newEntry != original ) {
            EntityEntryExtraState extraState = newEntry.getExtraState(
EntityEntryExtraState.class );
            if ( extraState == null ) {
                newEntry.addExtraState( original.getExtraState(
EntityEntryExtraState.class ) );
            }
        }

        return id;
```

```
    }
```

## class AbstractSaveEventListener

```
    /**
     * Prepares the save call by checking the session caches for a pre-existing
     * entity and performing any lifecycle callbacks.
     *
     * @param entity The entity to be saved.
     * @param id The id by which to save the entity.
     * @param persister The entity's persister instance.
     * @param useIdentityColumn Is an identity column being used?
     * @param anything Generally cascade-specific information.
     * @param source The session from which the event originated.
     * @param requiresImmediateIdAccess does the event context require
     * access to the identifier immediately after execution of this method (if
     * not, post-insert style id generators may be postponed if we are outside
     * a transaction).
     *
     * @return The id used to save the entity; may be null depending on the
     *         type of id generator used and the requiresImmediateIdAccess
value
     */
    protected Serializable performSave(
            Object entity,
            Serializable id,
            EntityPersister persister, <--------------------------------???
            boolean useIdentityColumn,
            Object anything,
            EventSource source,
            boolean requiresImmediateIdAccess) {

        if ( LOG.isTraceEnabled() ) {
            LOG.tracev( "Saving {0}", MessageHelper.infoString( persister, id,
source.getFactory() ) );
        }

        final EntityKey key;
        if ( !useIdentityColumn ) {
            key = source.generateEntityKey( id, persister );
            final PersistenceContext persistenceContext =
source.getPersistenceContextInternal();
            Object old = persistenceContext.getEntity( key );
            if ( old != null ) {
                if ( persistenceContext.getEntry( old ).getStatus() ==
Status.DELETED ) {
                    source.forceFlush( persistenceContext.getEntry( old ) );
                }
                else {
```

```
                    throw new NonUniqueObjectException( id,
persister.getEntityName() );
                }
            }
            persister.setIdentifier( entity, id, source );
        }
        else {
            key = null;
        }

        if ( invokeSaveLifecycle( entity, persister, source ) ) {
            return id; //EARLY EXIT
        }

        return performSaveOrReplicate(
                entity,
                key,
                persister,<------------------------------
                useIdentityColumn,
                anything,
                source,
                requiresImmediateIdAccess
        );
    }
```

```
    protected Serializable saveWithGeneratedId(
            Object entity,
            String entityName,
            Object anything,
            EventSource source,
            boolean requiresImmediateIdAccess) {
        callbackRegistry.preCreate( entity );

        ManagedTypeHelper.processIfSelfDirtinessTracker( entity,
SelfDirtinessTracker::$$_hibernate_clearDirtyAttributes );

        EntityPersister persister = source.getEntityPersister( entityName,
entity ); <----------------------------------------
        Serializable generatedId = persister.getIdentifierGenerator().generate(
source, entity );
        if ( generatedId == null ) {
            throw new IdentifierGenerationException( "null id generated for:" +
entity.getClass() );
        }
        else if ( generatedId ==
IdentifierGeneratorHelper.SHORT_CIRCUIT_INDICATOR ) {
            return source.getIdentifier( entity );
        }
        else if ( generatedId ==
```

```
IdentifierGeneratorHelper.POST_INSERT_INDICATOR ) {
            return performSave( entity, null, persister, true, anything,
source, requiresImmediateIdAccess );
        }
        else {
            // TODO: define toString()s for generators
            if ( LOG.isDebugEnabled() ) {
                LOG.debugf(
                        "Generated identifier: %s, using strategy: %s",
                        persister.getIdentifierType().toLoggableString(
generatedId, source.getFactory() ),
                        persister.getIdentifierGenerator().getClass().getName()
                );
            }

            return performSave( entity, generatedId, persister, false,
anything, source, true );
        }
    }
```

## class SessionImpl

```
    @Override
    public EntityPersister getEntityPersister(final String entityName, final
Object object) {
        checkOpenOrWaitingForAutoClose();
        if ( entityName == null ) {
            return getFactory().getMetamodel().entityPersister(
guessEntityName( object ) );
        }
        else {
            // try block is a hack around fact that currently tuplizers are not
            // given the opportunity to resolve a subclass entity name.  this
            // allows the (we assume custom) interceptor the ability to
            // influence this decision if we were not able to based on the
            // given entityName
            try {
                return getFactory().getMetamodel().entityPersister( entityName
).getSubclassEntityPersister( object, getFactory() ); <-----------------------
---------------
            }
            catch (HibernateException e) {
                try {
                    return getEntityPersister( null, object );
                }
                catch (HibernateException e2) {
                    throw e;
                }
            }
```

```
        }
    }
```

```
no method return value
this    SessionImpl  (id=117)
entityName  "de.icongmbh.dope.flow.bo.TicketIF" (id=154)
    coder   0
    hash    576080636
    hashIsZero  false
    value   (id=201)
object  Ticket  (id=134)
    additionalDatas HashSet<E>  (id=163)
    administrationState "smartadmin" (id=171)
    audit   Audit  (id=172)
    currentTask null
    currentTask null
    currentTaskOid  null
    deleted false
    externalID  null
    extraInfo   null
    historyContext  null
    mergedObjects   HashSet<E>  (id=174)
    oid null
    predecessor null
    prio    2147483647
    processState    ProcessState  (id=175)
    processType "saved ticket" (id=179)
    processWorkbasket   null
    releaseDateObjects  HashSet<E>  (id=181)
    requirement null
    stagingTimestamp    null
    supervisor  null
    tenantId    "DOPiX" (id=184)
    testAnnotations HashSet<E>  (id=186)
    transitionCounter   0
    version null
```

## AbstractSharedSessionContract

```
    @Override
    public SessionFactoryImplementor getFactory() {
        return factory;
    }
```

```
no method return value
this    SessionImpl  (id=117)
    actionQueue ActionQueue  (id=178)
    autoClear   false
    autoClose   false
    autoJoinTransactions    true
    cacheMode   CacheMode  (id=182)
    cacheTransactionSync    NoCachingTransactionSynchronizationImpl  (id=185)
    closed  false
    connectionHandlingMode  PhysicalConnectionHandlingMode  (id=190)
    criteriaCompiler    null
    currentHibernateTransaction TransactionImpl  (id=192)
    dontFlushFromFind   0
    entityNameResolver  CoordinatingEntityNameResolver  (id=197)
    exceptionConverter  null
    factory SessionFactoryImpl  (id=205)
        cacheAccess DisabledCaching  (id=254)
        criteriaBuilder CriteriaBuilderImpl  (id=261)
        currentSessionContext   ThreadLocalSessionContext  (id=265)
        defaultSessionOpenOptions   SessionFactoryImpl$SessionBuilderImpl<T>
(id=269)
        defaultStatelessOptions SessionFactoryImpl$StatelessSessionBuilderImpl
(id=274)
        delayedDropAction   SchemaDropperImpl$DelayedDropActionImpl  (id=277)
        eventEngine EventEngine  (id=280)
        fastSessionServices FastSessionServices  (id=216)
        fetchProfiles   HashMap<K,V>  (id=282)
        filters HashMap<K,V>  (id=286)
        identifierGenerators    HashMap<K,V>  (id=287)
        jdbcServices    JdbcServicesImpl  (id=288)
        jpaPersistenceUnitUtil  PersistenceUnitUtilImpl  (id=293)
        metamodel   MetamodelImpl  (id=297)
        name    null
        namedQueryRepository    NamedQueryRepository  (id=302)
        observer    SessionFactoryObserverChain  (id=304)
        persistenceContextType  null
        properties  HashMap<K,V>  (id=307)
        queryPlanCache  QueryPlanCache  (id=308)
        serviceRegistry SessionFactoryServiceRegistryImpl  (id=310)
        sessionFactoryOptions   SessionFactoryOptionsBuilder  (id=318)
        settings    Settings  (id=321)
        sqlFunctionRegistry SQLFunctionRegistry  (id=323)
        sqlStringGenerationContext  SqlStringGenerationContextImpl  (id=325)
        statistics  StatisticsImpl  (id=328)
        status  SessionFactoryImpl$Status  (id=333)
        synchronizationType null
        temporarySessionOpenOptions SessionFactoryImpl$SessionBuilderImpl<T>
(id=335)
        typeHelper  TypeLocatorImpl  (id=336)
        uuid    "f29f4ec9-0d72-4e60-864d-f7e264c463d9" (id=339)
    fastSessionServices FastSessionServices  (id=216)
    flushMode   FlushMode  (id=218)
```

```
    interceptor AuditDelegateInterceptor  (id=220)
    isEnforcingFetchGraph   false
    isTransactionCoordinatorShared  false
    jdbcBatchSize   null
    jdbcConnectionAccess    NonContextualJdbcConnectionAccess  (id=225)
    jdbcCoordinator JdbcCoordinatorImpl  (id=228)
    jdbcSessionContext  JdbcSessionContextImpl  (id=233)
    jdbcTimeZone    null
    loadEvent   null
    loadQueryInfluencers    LoadQueryInfluencers  (id=236)
    lobHelper   null
    lockOptions null
    persistenceContext  StatefulPersistenceContext  (id=238)
    properties  null
    queryParametersValidationEnabled    true
    sessionEventsManager    SessionEventListenerManagerImpl  (id=241)
    sessionIdentifier   null
    tenantIdentifier    null
    transactionCoordinator  JdbcResourceLocalTransactionCoordinatorImpl
(id=245)
    transactionObserver null
    waitingForAutoClose false
```

## Class RegisteredObject ????

public RegisteredObject(final RegisteredObjectIF registeredObject) {
super(registeredObject.getTicketID(), UNKNOWN_OBJECT_UID, Type.REGISTERED_OBJECT,
UNKNOWN_OBJECT_VERSION);
this.tenantId = registeredObject.getTenantId();
this.regObjID = registeredObject.getRegObjID();
this.actionType = registeredObject.getActionType();
this.objectLastModified = new Date(System.currentTimeMillis());
setHistoryContext(registeredObject.getHistoryContext()); <----------
---
}

{
hibernate.format_sql=false,
hibernate.hbm2ddl.auto=create-drop,
hibernate.jdbc.use_streams_for_binary=true,
hibernate.interceptor=de.icongmbh.org.hibernate.RevisionSecureInterceptor,
hibernate.c3p0.max_size=20,
hibernate.c3p0.idle_test_period=3000,
hibernate.dialect=de.icongmbh.org.hibernate.dialect.HSQLDialect,
hibernate.ejb.event.flush-entity=de.icongmbh.org.hibernate.event.MarkAsDeletedEventListener,
hibernate.c3p0.min_size=5,
hibernate.ejb.event.pre-update=de.icongmbh.org.hibernate.event.KeepHistoryEventListener ,

de.icongmbh.org.hibernate.event.SetTaskUIDToTicketListener,
hibernate.connection.url=jdbc:hsqldb:mem:TEST_DOPE_FLOW_DATABASE,
hibernate.show_sql=false,
hibernate.generate_statistics=false,
hibernate.order_updates=true,
hibernate.ejb.event.pre-delete=de.icongmbh.org.hibernate.event.MarkAsDeletedEventListener,
hibernate.current_session_context_class=thread,
hibernate.jdbc.batch_size=0,
hibernate.use_custom_identifier_generator_factory=true,
hibernate.connection.ClientUser=DOPE/Flow Core - TEST,
hibernate.connection.driver_class=org.hsqldb.jdbc.JDBCDriver,
hibernate.use_sql_comments=true,
hibernate.ejb.event.post-load=de.icongmbh.org.hibernate.event.ReleaseDateNameInitializeListener,
hibernate.c3p0.max_statements=50,
hibernate.connection.ApplicationName=DOPE/Flow Hibernate DBM,
hibernate.cache.provider_class=org.hibernate.cache.NoCacheProvider,
hibernate.mapping.resource=de/icongmbh/dope/flow/bo/BaseTypes.hbm.xml,
de/icongmbh/dope/flow/bo/ErrorMessage.hbm.xml,
de/icongmbh/dope/flow/bo/BasicTicket.hbm.xml,
de/icongmbh/dope/flow/bo/Ticket.hbm.xml,
de/icongmbh/dope/flow/bo/task/FlowTask.hbm.xml,
de/icongmbh/dope/flow/bo/AttachedObject.hbm.xml,
de/icongmbh/dope/flow/bo/ObjectDataContainer.hbm.xml,
de/icongmbh/dope/flow/bo/ValuedObjectDataContainer.hbm.xml,
de/icongmbh/dope/flow/bo/ObjectConfiguration.hbm.xml,
de/icongmbh/dope/flow/bo/attribute/CustomAttributeDefinition.hbm.xml,
de/icongmbh/dope/flow/bo/attribute/CustomAttributeValue.hbm.xml,
de/icongmbh/dope/flow/bo/release/ReleaseDateValue.hbm.xml,
de/icongmbh/dope/flow/bo/release/ReleaseDateName.hbm.xml,
de/icongmbh/dope/flow/bo/ProtocolTaskInfo.hbm.xml,
de/icongmbh/dope/flow/bo/TestAnnotation.hbm.xml,
de/icongmbh/dope/flow/bo/AdditionalData.hbm.xml,
de/icongmbh/dope/flow/bo/TestObjects.hbm.xml,
de/icongmbh/dope/flow/bo/TestHistoryObjects.hbm.xml,
de/icongmbh/dope/flow/bo/MergedObject.hbm.xml,
de/icongmbh/dope/flow/bo/RegisteredObjectConflict.hbm.xml,
hibernate.ejb.event.pre-insert=de.icongmbh.org.hibernate.event.SetTaskUIDToTicketListener,
hibernate.c3p0.timeout=300, hibernate.connection.pool_size=2,
hibernate.default_batch_fetch_size=16}