Clean up FeatureEvents

Motivation:	We need feature events to draw changes as they occur, and keep caches in sync.
Contact:	~jeichar
Tracker:	http://jira.codehaus.org/browse/GEOT-1607
Tagline:	What changed where

This page represents the **current** plan; for discussion please check the tracker link above.

Problem

FeatureEvents have long been something missed in the GeoTools library; the uDig project has hacked a few DataStores into submission way back in GeoTools 2.2. Now that we have updated to GeoTools trunk it is time to talk about what we have learned and see what can be cleaned up. Conversely FeatureEvents have not been implemented by the various DataStore implementations; in part that is probably due to a lack of clarity and available test cases; we need to take implementation experience into account and provide an event notification system that **ca n** be implemented.

There are several "feature event" notification things around:

- FeatureCollection add/remove CollectionListener
  - CollectionEvent tracks FEATURES_ADDED / FEATURES_REMOVED / FEATURES_CHANGED and holds an array of changed features
- FeatureSource add/remove FeatureListener
  - FeatureEvent tracks FEATURES_ADDED / FEATURES_REMOVED / FEATURES_CHANGED and holds **bounds**

There are two other input reference points around the functionality of the "commit" step:

- The FIDs created by addFeatures are often temporary; until the commit() method is called. WFSDataStore has taken to maintaining a history of all temporary FeatureIds handed out, it creates a mapping from temporary Id to a real Id during the commit, and then uses this mapping to preprocess all incoming FidFilters into a form understood by the web feature server. Wow.
- The revised datastore api figured out for GeoAPI forced the commit step to return the set of added FeatureIds in order to handle the same problem
- DataStoreEvents were added as part of one of the many catalog proposal in the hopes of noticing when the bounds were changed on the data store.
- During the code sprint a lot of the ReferencedEnvelope changes were centred on building up the bounds for a FeatureEvent; this code looked really troubled to start with and the result had been cut and pasted everywhere.
- FeatureCollection is gradually being punted in the direction of a data result rather than a read/write API
- FeatureListenerManager already gathers up the hard work of handling events on different transactions; it is a utility class used by DataStore implementors as far as I know nobody has implemented events without using this class.
- The feature added/removed/changed events are documented to be performed at each FeatureWriter.next(), some implementors are using FeatureWriter.write().
- It would be easier for listeners if the changes were batched up and sent once. On FeatureWriter.close() and on FeatureStore.add/remove/modifyFeatures. That is a single event shall be thrown once the

add/remove/modifyFeatures method did its work.

Distractions, Observations and Complaints:

- GR: I'd actually would like to get rid of FeatureWriter as a first class citizen.
- GR: DataStore.getFeatureWriter(...) is quite missplaced, if should ever have to exist, it should be part of the FeatureStore interface

Solution

- For the listener (ie uDig or a cache implementation) the FeatureEvent will capture a Filter and optional bounds. This is more efficient than an array of SimpleFeature (which would prove fatal if we ever did it). We will provide a NullObject for the bounding box that represents everything (to prevent client code from continuing to do null checks) on bounds.
- For the data store implementor we will make the FeatureEventImpl object a lot smarter; it will have methods for addFeature(SimpleFeature) so that you no longer have to manage ReferenceEnvelope madness yourself. You can either make these events and fire them on every change; or go through your loop and build up an event to fire at the end.
- For the data store implementor we will make the FeatureEventImpl do whatever is needed for your "commit" cycle; as long as the event is keeping a Filter we should be able to handle the case where a Id filter of FeatureId is updated as actual identifiers are acquired during commit. Since FeatureId is an **object** rather than a String we can do this without violating encapsulations. This change is limited to the `FeatureListenerManager` class and may get better over time; we don't need to get it perfect right off to have a benifit.
- For the data store implementor we will ask them to fire an event on FeatureWriter.close(); rather than next() or write(). They can create a FeatureEvent at the start of their FeatureWriter and add to it as they iterate.
- For the data store implementor we will ask for a single event for each add / remove / modifyFeatures FeatureStore call (most do this already).

Status

The window for feedback on this proposal has closed, the proposal has been accepted by default.

Initial discussion as part of an IRC meeting was mostly in agreement. Justin made the interesting observation that only one of the changes is significant to client code:

- Change from FeatureWriter.next() notification to FeatureWriter.close()
- Gabriel asked that the Set used for batch notification of be a weak hash set

Voting is open:

- Andrea Aime +0
- Ian Turton +0
- Justin Deoliveira +0
- Jody Garnett +1
- Martin Desruisseaux +0
- Simone Giannecchini +0

Community support:

- Gabriel Roldán +1

dynamictasklist: task list macros declared inside wiki-markup macros are not supported

Tasks

This section is used to make sure your proposal is complete (did you remember documentation?) and has enough paid or volunteer time lined up to be a success

	no progress	✅	done	❎	impeded	⚠️	lack mandate /funds/time	❓	volunteer needed

1. ✅ Deprecate CollectionListener / CollectionEvent - asking people to listen to the FeatureSource instead
2. ✅ Introduce RefernecedEnvelope.EVERYTHING
3. ✅ Add Filter to FeatureEvent
4. ✅ Add Filter and Bounds "building" methods to FeatureEvent(update: Implemented as BatchFeatureEvent)
5. ✅ Update the DataStore Tutorial section on FeatureEvents based on the ArcSDE experience
6. ⚠️ Provide an example of using a listening with FeatureEvents in the user guide

API Changes

BEFORE

describe subject is being changed

```
class ReferencedEnvelope {
     ....
}
class FeatureEvent {
     ....
}
```

AFTER

Deprecate code that is orphaned (maybe done in GeoTools 2.4?):

```
interface FeatureCollection {
    /**
     * @deprecated Please use
FeatureSource.addFeatureListener
     */
    void addListener(CollectionListener
listener) throws NullPointerException;

    /**
     * @deprecated Please use
FeatureSource.removeFeatureListener
     */
    void removeListener(CollectionListener
listener) throws NullPointerException;
}
```

Expand ReferencedEnvelope to include the definition of everything:

```
class ReferencedEnvelope {
     static final RefernecedEnvelope
EVERYTHING = new RefernecedEnvelope(){
          ... over ride methods as needed
this is an infinite envelope of any CRS ...
     }
}
```

Add helper methods to FeatureEvent so DataStore implementors do not have to work so hard to build up a good change notification:

```
interface FeatureEvent {
```

```
     int FEATURES_ADDED = 1;
     int ADDED = 1;

     int FEATURES_CHANGED = 0;
     int CHANGED = 0;

     int FEATURES_REMOVED = -1;
     int REMOVED = -1;

     public Filter getFilter();
     public ReferencedEnvelope getBounds();
}

class FeatureEventImpl {

     private ReferencedEnvelope bounds;

     /** Filter here is the new part; the
point of this proposal */
     private Filter filter;

     public Filter getFilter();

     /** Allow event to grab the bounds and
Fid *before* anything is modified or removed
*/
     public void before( SimpleFeature
feature );

     /** Indicate the provided feature was
modified */
```

```
     public void modify( SimpleFeature
feature );

     /** Indicate the provided feature was
added */
     public void add( SimpleFeature feature
);

     /** Indicate the provided feature was
removed */
     public void remove( SimpleFeature
feature );
}
```

Subclass to provide batch notification on Commit / Rollback. This is used by FeatureListenerManager to report a bit more detail on transaction commit() and rollback(). Right now these changes show up as a change event with no known bounds.

```
class BatchFeatureEvent {
     public static in COMMIT = 5;
     public static in ROLLBACK = 6;

     /**
      * This is a weak hash set as we don't
need to track
      * changes on indentifiers that are not
being used
      * by the client to track selection.
      */
     WeakHashSet<FeatureId> fids;

     /** Will often return Filter.INCLUDE */
```

```
     public Filter getFilter();

     /** Will often return everything */
     public RefernecedEnvelope getBounds();

     /**
      * Indicate the provided feature was
added.
      * Will be use to update internal state
of bounds and
      * fids.
      */
     public void add( FetaureEvent change );

     /**
      * Used to update any FeatureId during
a commit.
      */
     public void replaceFid( String tempFid,
String actualFid );
```

```
     public void WeakHashSet<FetaureId>
getCreatedFeatureIds();
}
```

With the following changes to javadocs documenting expected behaviour:

```
interface FeatureWriter {
   /**
    * Advance to the next feature - notice
no event fired
    */
   SimpleFeature next();
   /**
    * Modify (or add) the current feature -
notice no event fired
    */
   void write();
   /**
    * We are finished with this
FeatureWrtier; event fired summarizing the
changes made.
    */
   void close();
}
interface FeatureStore extends FeatureSource
{
     /**
      * Will issue a single FeatureEvent.
      */
     Set<String>
addFeatures(FeatureCollection collection)
```

```
throws IOException;
    /**
     * Will issue a single FeatureEvent.
     */
    void removeFeatures(Filter filter)
throws IOException;
    /**
     * Will issue a single FeatureEvent.
     */
    void modifyFeatures(AttributeDescriptor
type, Object value, Filter filter)
        throws IOException;
    ...
}
class FeatureListenerManager {
     class BatchListener implements
FeatureListener, Transaction.State {
          private BatchFeatureEvent batch;
          void setTransaction(Transaction
transaction){
                batch = transaction == null
? null : new BatchFeatureEvent( transaction
);
          }
          void changed(FeatureEvent
featureEvent){
               if( batch != null )
batch.add( featureEvent );
          }
          /** Fire batch event to
Transaction.AUTO_COMMIT */
```

```
        public synchronized void commit()
throws IOException{
        }
        /** Fire batch event to
Transaction.AUTO_COMMIT */
        public synchronized void
rollback() throws IOException{
        }
```

```
      }
      ...
}
```

We also need to allow our FeatureIdImpl to be "updated" during the commit:

```
public class FeatureIdImpl implements
FeatureId {
 String fid;
 String origionalFid;
        ...
 public void setID( String id ){
  if( origionalFid == null ){
   origionalFid = fid;
   fid = id;
  }
  else {
   throw new IllegalStateException("You can
only assign a real id once during a
commit");
  }
 }
        ...
}
```

Documentation Changes

There has been little documentation thus far around the behaviour of events during editing; so in that sense our work is easy.

- Upgrade to 2.5 - Capture the BEFORE / AFTER section
- Home - add a page for feature events
- Home - update the datastore tutorial