

FeatureCollection cleanup

Motivation:	FeatureCollection is not a java.util.Collection
Contact:	Jody Garnett,Michael Bedward,Andrea Aime
Tracker:	https://jira.codehaus.org/browse/GEOT-4181
Tagline:	remove methods that assume in-memory model

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

- [History and Motivation](#)
- [Bring back FeatureResult](#)
 - [Remove the CollectionListener](#)
 - [DefaultFeatureCollection](#)
 - [Remove Iterator Methods](#)
 - [Remove Collection Methods](#)
 - [getSchema and getDescriptor\(\)](#)
 - [FeatureCollection](#)
 - [DataUtilities](#)
 - [FeatureCollection Implementations](#)
- [Status](#)
 - [Tasks](#)
 - [Documentation Changes](#)
 - [Upgrade Instructions](#)
 - [FeatureCollection Add](#)
 - [FeatureCollection Iterator](#)
 - [FeatureCollection close method](#)

Children:

History and Motivation

FeatureCollection needs to lighten up to be happy; the original FeatureCollection implementation was a java.util.Collection offering iterator based access to features stored in memory. When the data access layer was rewritten for DataStore the concept of a FeatureResults was introduced which could be thought of as a "prepared statement" (able to fetch results each time it was asked). This is a nice light weight "streaming" model for data access that does not force us to load the data into memory.

The intention was to treat it a bit more like a Java ResultSet (so several iterators could be open against the same pagged results). But the road to hell is paved with good intentions.

And the FeatureResults interface was in for some hell: [2.1.x Bring Back FeatureCollection](#) was a last ditch community lead request to bring back the FeatureCollection api and allow iterators to be used (thus not existing current code). The compromise that was reached was to ask users to FeatureCollection.close(Iterator) each iterator after use; so that we could still offer a nice streaming based approach to data access.

For GeoTools 2.5.x we made the migration to Java 5 and we could no longer allow FeatureCollection to implement java.util.Collection because of the new "for each" syntax did not offer a callback for us to close our iterators. Justin started [Focus FeatureSource around FeatureReader and FeatureWriter](#), however all we had time for was the bare

minimum to make Java 5 not a fatal opportunity for client code to leak memory.

Bring back FeatureResult

In the spirit of [Bring Back FeatureCollection](#) here is our last minuet 2.7.x request to complete the journey; formally recognise that FeatureCollection is not a java.util.Collection and turf Iterator access (and other concepts that mirror in memory storage).

Remove the CollectionListener

First up is the add/remove listeners; a recent code review shows that they are only implemented by the new JDBCNG FeatureCollection (because justin is apparently very responsible). Due to everyone else not being responsible we can safely remove these since no client code could of depended on being sent an event.

```
addListener(CollectionListener)
removeListener(CollectionListener)
```

Note: These listeners have been deprecated as part of [Clean up FeatureEvents](#) since this proposal was written.

DefaultFeatureCollection

FeatureCollections.newCollection

Currently FeatureCollections.newCollection returns a SimpleFeatureCollection. Many code examples populate this feature collection using the add method.

```
SimpleFeatureCollection collection =
FeatureCollections.newCollection();
collection.add( feature ); // <-- WILL NOT
WORK!
```

The most expidient fix is to create a new DefaultFeatureCollection directly (this is the implementation returned by newCollection() above).

```
DefaultFeatureCollection collection = new
DefaultFeatureCollection();
collection.add( feature );
```

DefaultFeatureCollection is a FeatureCollection that implements java.util.Collection . This is the current migration path for those wishing to have an in memory collection.

DataUtilities.collection

Asking for your feature collection to be "loaded" into a FeatureCollection is the currently documented way of avoiding all this streaming nonsense and loading your data into memory.

```
SimpleFeatureCollection collection =
DataUtilities.collection( featureCollection
);
```

If we find a lot of users go on to call **add** we can change the DataUtilities.collection methods to explicitly return a DefaultFeatureCollectionOption.

Alternatives

We may also wish to consider a different implementation for DefaultFeatureCollection:

- MemoryFeatureCollection and TreeSetFeatureCollection are copies of DefaultFeatureCollection
- ListFeatureCollection - is newer and does not protect against duplicates, but may be faster in day to day use?
- SpatialIndexFeatureCollection - cannot be modified once made; not a suitable replacement

Remove Iterator Methods

The following methods cover the care and feeding of Iterators:

```
void close(FeatureIterator<SimpleFeature>)
Iterator<SimpleFeature> iterator()
void close(Iterator<SimpleFeature>);
void purge()
```

We are removing the close(FeatureIterator) method as well as it already has a serviceable FeatureIterator.close() method (the method was only provided to ease transition to FeatureIterator).

Remove Collection Methods

The following methods are used to update and modify the contents of an in memory Collection and do not fit with the "Prepared Statement" model of a FeatureCollection.

```
// modification of in memory feature
collections
    // (recommend using featuresource instead)
    boolean add(SimpleFeature)
    boolean addAll(Collection<? extends
SimpleFeature>)
    boolean addAll(FeatureCollection<? extends
SimpleFeatureType, ? extends SimpleFeature>)
    boolean clear()
    boolean remove(Object)
    boolean removeAll(Collection<?>)
    boolean retainAll(Collection<?>)
```

getSchema and getDescriptor()

This is a long standing hole in the API, proposal written in response to questions from Ben: [FeatureCollection Descriptor for FeatureMembers](#)

It sounds like this is just an additional method:

```
AttributeDescriptor getDescriptor();
FeatureType getSchema();
```

Consider making the API consistent with a getType() method? Does not seem worth the bother.

Tasks:

- Added in GeoTools 9.0.x

FeatureCollection

Here is what SimpleFeatureCollection looks like after the change:

```
public interface SimpleFeatureCollection
extends
```

```
FeatureCollection<SimpleFeatureType, SimpleFeature> {
    // feature access - close when done!
    SimpleFeatureIterator features();

    // feature access with out the loop
    void accepts(FeatureVisitor,
ProgressListener);

    // metadata
    String getID();
    SimpleFeatureType getSchema();
    AttributeDescriptor getDescriptor(); //
pending prosal

    // sub query
    SimpleFeatureCollection
subCollection(Filter);
    SimpleFeatureCollection sort(SortBy);
    boolean contains(Object);
    boolean containsAll(Collection<?>);

    // summary information
    ReferencedEnvelope getBounds();
    boolean isEmpty();
    int size();

    // convert to array
```

```
Object[] toArray();
<O> O[] toArray(O[]);
}
```

Points raised during discussion:

- toArray methods are there on the off chance the implementation can provide a quick implementation? Is this another good intension? Probably. Could be replaced with a DataUtilities method
- subCollection & sort functionality have been covered by Query these days; and could be removed. Alternative: subCollection(Query) would be more useful.
- contains and containsAll are interesting; would almost expect look up by FeatureId instead
- getID() is left from a time when FeatureCollection implemented Feature; it is still used when encoding
- getDescriptor is technically accurate; however we are used to working with FeatureType
- isEmpty() earns its keep; size() == 0 is not a replacement as you need to count the whole set
- with that in mind containsAll falls into the same idea; how to prevent a multi pass operation
- accepts earns its keep for [Jody Garnett](#), [Andrea Aime](#) wants the javadocs cleaned up so it is obvious that the feature being visited is a Flyweight (ie the same object with just the values changing inside each time)
- Minimal baseline: minimum that is functional (if we want more we need to justify it):

```
public interface SimpleFeatureCollection
extends
FeatureCollection<SimpleFeatureType, SimpleFe
ature> {
    SimpleFeatureIterator features();
    String getID();
    SimpleFeatureType getSchema();
    ReferencedEnvelope getBounds();
    boolean isEmpty();
    int size();
}
```

DataUtilities

During the review process many additional utility methods were added to DataUtilities in order to ensure all FeatureCollections handled things in a similar fashion.

Specific changes to functionality:

- visit(FeatureVisitor, Progress) now throws an IOException in the case of problems, rather than just reporting

via the supplied progress

Here are the new DataUtilities methods:

```
DataUtilities.visit( FeatureCollection,
FeatureVisitor, ProgerssListener )
DataUtilities.bounds( FeatureCollection )
<-- already existed
DataUtilities.bounds( FeatureIterator )
DataUtilities.count( FeatureCollection )
DataUtilities.count( FeatureIterator )
DataUtilities.close( Iterator )
DataUtilities.first( SimpleFeatureCollection
): SimpleFeature
DataUtilities.first( FeatureCollection<?,F>
): F
DataUtilities.list( FeatureCollection<?,F>
): List<F> // Already existed
DataUtilities.list( FeatureCollection<?,F>,
int ): List<F>
DataUtilities.iterator( FeatureIterator ):
Iterator // also Closable
DataUtilities.collectionCast(
FeatureCollection ): Collection
```

FeatureCollection Implementations

Base FeatureCollection classes available for extending:

- **AbstractFeatureCollection** - requires an **openIterator** method and attempts to track all the outstanding iterators and clean up after them with a **purge** method.
- **BaseFeatureCollection / BaseSimpleFeatureCollection** - requires a **features()** method, all other methods are implemented by calling **features()**, recommend supply **size()** and **bounds()** methods.
 - **ProcessingCollection / SimpleProcessingCollection** - requires **features()**, **getBounds()**, **size()**, and **buildTargetFeatureType()**

- **DecoratingFeatureCollection / DecoratingSimpleFeatureCollection** - each method calls through to an internal delegate FeatureCollection
- **AdaptorFeatureCollection** - copy of AbstractFeatureCollection used as a base class by uDig (should replace with BaseFeatureCollection)
- **DataFeatureCollection** - requires a **reader()** method and implementation of **FeatureReader**, also **getSchema()**, **getBounds()**, **getCount()** and **collection()**

Working FeatureCollection implementations for day to day use in your own code:

- **DefaultFeatureCollection** - uses a TreeSet sorted by FeatureId internally
- **ListFeatureCollection**
- **SpatialIndexFeatureCollection**
- **TreeSetFeatureCollection**

Implementation specific FeatureCollection classes:

- **MemoryFeatureCollection** - used internally by MemoryDataStore storing Features in a TreeSet sorted by FeatureId
- **SubFeatureCollection** - uses a Filter to reduce the number of features returned
 - **SubFeatureList** - tries to implement SortBy using a List<FeatureId> with determined order
- **ContentFeatureCollection** - delegates most work to ContentFeatureSource for the win
- **DefaultVersionedFeatureCollection** - does not appear complete
- **SimpleFeatureCollectionBridge** - wraps a FeatureCollection<SimpleFeatureType,SimpleFeature> up as a SimpleFeatureCollection

Status

This proposal is the subject of a grass roots revolution on the geotools-devel list; indeed the revolution is on going and will not be televised.

Voting has not started yet:

- [Andrea Aime](#) +0
- [Ben Caradoc-Davies](#) +0
- [Christian Mueller](#) +0
- [Ian Turton](#) +0
- [Justin Deoliveira](#) +1
- [Jody Garnett](#) +1
- [Michael Bedward](#) +0
- [Simone Giannecchini](#) +0

Tasks

1. JG: Deprecate methods for 8.0 release
 - FeatureCollection Deprecate Collection methods
 - FeatureCollection Deprecate CollectionListener (and methods)
 - FeatureCollection Deprecate FeatureCollection Iterator methods
2. JG: Changes for 9.0 master
 - FeatureCollection Remove Collection methods
 - FeatureCollection Remove CollectionListener (and methods)
 - FeatureCollection Remove FeatureCollection Iterator methods
 - Deprecate FeatureCollections factory
3. Fix support classes (⚠️ used to mark anything that did not go smoothly)
 - DataFeatureCollection remains based on FeatureReader

- AbstractFeatureCollection will remain based on iterator()
 - ForceCoordinateSystemFeatureResults
 - MemoryFeatureCollection
 - ReprojectFeatureResults
 - TransformFeatureCollection
 - BaseFeatureCollection created to be based on features()
 - SubCollection migrated to BaseFeatureCollection
 - SortFeatureList migrated to BaseFeatureCollection
 - AdaptorFeatureCollection - base class used by uDig
 - ContentFeatureCollection
 - DecoratingSimpleFeatureCollection
 - FilteringSimpleFeatureCollection: migrated to FilteringFeatureIterator (from FilteringIterator)
 - MaxSimpleFeatureCollection: migrated to MaxFeaturesSimpleFeatureIterator (from MaxFeaturesIterator)
 - ReprojectingFeatureCollection: migrated to ReprojectingFeatureIterator (from ReprojectingIterator)
 - ReTypingFeatureCollection: migrated to ReTypingFeatureIterator (from ReTypingIterator)
 - DefaultFeatureCollection - remains based on TreeSet
 - GMLFeatureCollection: subclass of DefaultFeatureCollection
 - MemoryFeatureCollection: copy of DefaultFeatureCollection
 - TreeSetFeatureCollection: copy of DefaultFeatureCollection changed to no longer implement Collection
 - DefaultVersionedFeatureCollection: add is now protected and use during init to populate the collection
 - PropertyValueCollection - this is not a FeatureCollection it was being used to return Collection<Attribute> I have changed it to return a Feature (containing the expected Attribute) but cannot untangle the Encoder well enough to make the test pass.
4. Fix geotools modules (notes on any thing interesting, also check commit logs)
- In general pretty smooth transition
 - app-schema makes "dangerous" use of feature collection support classes (that internally assume SimpleFeature)
 - StreamingRenderer required code duplication to handle Collection.iterator() and FeatureCollection.features()
 - Many cases where try/finally was used to ensure featureIterator.close() was called
 - PropertyValueCollection and ValueCollectionTypeBindingTest is broken as is conflicted between FeatureCollection and Collection<Attribute>
 - Thanks to Justin for redoing the EMF bindings to support a Collection<Attribute> so this could be handled
5. Documentation - see Documentation Changes below
6. Integration party
- a. JG: Stage work on branch: https://github.com/geotools/geotools/tree/featurecollection_cleanup
 - b. JG: Submit pull request: <https://github.com/geotools/geotools/pull/44>
 - c. JG: deploy 9.0-M0-SNAPSHOT as a migration target
 - d. JD: GeoServer pull request: <https://github.com/geoserver/geoserver/pull/61>
 - e. JG: uDig pull request: <https://github.com/uDig/udig-platform/pull/161>

Documentation Changes

The following documentation needs to be updated:

- Fix up the [upgrade](#) instructions
- Update [feature tutorial](#) to use ListFeatureCollection (to preserve order)
- Update [feature collection examples](#)

Upgrade Instructions

These examples are taken from [upgrade.rst](#) (included in the pull request).

1. `add(Feature)`
 - Use `DefaultFeatureCollection` so `Collection.add(feature)` is visible
 - Change `FeatureCollections.newCollection()` to `new DefaultFeatureCollection()`
2. `FeatureCollection.close(Iterator)`
 - Preferred: use `FeatureIterator.close()`
 - Use: `((Closeable)iterator).close()`
3. Handy `FeatureCollections` that implement `java.util.Collection`
 - `ListFeatureCollection` (delegates to `List`)
 - `DefaultFeatureCollection` (the original based on a `TreeSet` sorted by `FeatureId`)
 - `MemoryFeatureCollection` (a copy of `DefaultFeatureCollection` for use by `MemoryDataStore`)

FeatureCollection Add

With the `FeatureCollection.add` method being removed, you will need to use an explicit instance that supports adding content.

BEFORE:

```
SimpleFeatureCollection features =
FeatureCollections.newCollection();

for( SimpleFeature feature : list ){
    features.add( feature );
}
```

AFTER:

```
DefaultFeatureCollection features = new
DefaultFeatureCollection();
for( SimpleFeature feature : list ){
    features.add( feature );
}
```

ALTERNATE (will throw exception if FeatureCollection does not implement java.util.Collection)

```
Collection<SimpleFeature> collection =
DataUtilities.collectionCast(
featureCollection );
collection.addAll( list );
```

ALTERNATE DETAIL:

```
SimpleFeatureCollection features =
FeatureCollections.newCollection();
if( features instanceof Collection ){
    Collection<SimpleFeature> collection =
(Collection) features;
    collection.addAll( list );
}
else {
    throw new
IllegalStateException("FeatureCollections
configured with immutable implementation");
}
```

FeatureCollection Iterator

The deprecated FeatureCollection.iterator() method is no longer available, please use FeatureCollection.features() as shown below.

BEFORE:

```
Iterator i=featureCollection.iterator();
try {
    while( i.hasNext(); ){
        SimpleFeature feature = i.next();
        ...
    }
}
finally {
    featureCollection.close( i );
}
```

AFTER:

```
FeatureIterator
i=featureCollection.features();
try {
    while( i.hasNext(); ){
        SimpleFeature feature = i.next();
        ...
    }
}
finally {
    i.close();
}
```

JAVA7 (using try-with-resource):

```
try ( FeatureIterator
i=featureCollection.features() ){
    while( i.hasNext() ){
        SimpleFeature feature = i.next();
        ...
    }
}
```

FeatureCollection close method

We have made FeatureCollection implement closable (for Java 7 try-with-resource compatibility). This also provides an excellent replacement for FeatureCollection.close(Iterator).

For other code that relied on FeatureCollection.close(Iterator) to clean up after things, please make sure your Iterator implements **Closeable**.

BEFORE:

```
Iterator iterator = collection.iterator();
try {
    ...
} finally {
    if (collection instanceof
SimpleFeatureCollection) {
        ((SimpleFeatureCollection)
collection).close(iterator);
    }
}
```

AFTER:

```

Iterator iterator = collection.iterator();
try {
    ...
} finally {
    DataUtilities.close( iterator, collection
);
}

```

DETAIL:

```

Iterator iterator = collection.iterator();
try {
    ...
} finally {
    if (iterator instanceof Closeable) {
        try {
            ((Closeable)iterator).close();
        }
        catch( IOException e){
            Logger log = Logger.getLogger(
collection.getClass().getPackage().toString(
) );
            log.log(Level.FINE,
e.getMessage(), e );
        }
    }
}

```

JAVA7: using try-with-resource syntax for iterators that implement Closeable

```
try ( FeatureIterator  
i=featureCollection.features()) {  
    ...  
}
```