

# Random Data Access

## Goal

The uDig project (an OpenGIS based gt2 client w/ WFS+WMS support) is expected to provide an influx of JUMP developers used to an Array based in memory Feature model.

Our streaming approach will threaten to isolate them, and in general iterators are "harder" to quickly hack with then arrays. We do **know** we are stuck with them - iterators (or visitor) is required when dealing with large data sets.

## Solution

I think we can get 90% of the way there with convenience methods. Similar to getBounds(), getCount() the idea is to provide for common operations we want to optimize.

The idea is that users can keep fids rather than array indexes (requires a simple getFeature/setFeature API). The other mode of use I see people using is bounding box queries, providing optimized access to based on BBox would solve a lot of woes.

## Proposed API Changes

So FeatureSource would add the following methods:

- Set getFids() - return set of Feature IDs
- Feature getFeature( fid ) - grab a Feature from the Feature Source
- FeatureResults getFeatures( Envelope bbox ) - convenience method for spatial query
- FeatureReader reader( Envelope bbox ) - convenience method for spatial query
- Set getFids( Envelope bbox ) - convenience, low memory, method for spatial query? (jmacgill)

And FeatureStore would add the following methods:

- void setFeature( fid, Feature ) - may be able to remove
- FeatureWriter writer( Envelope bbox ) - convenience method for write access

And an extension to FeatureResults that handles writing .... FeatureSet (peer of result set):

- writer() - grab a writer for the FeatureSet

## Expected Benefit

With the above api we can provide optimized implementations based on random file or database result set access while maintaining iterator based implementations.

Any GIS based data solution worth it's salt provides spatial indexing allowing the BBox based queries to really move. I would also assume that the choice of FID aligns well with something that is quickish (like row based access for shapefile, or primary key for a database).

## Aside: Weither FeatureResults?

The split between FeatureResults and FeatureSource is blurring on me as people try and construct FeatureSources that are limited by a Query. If this continues we may need to smuch them together (in which case we lose the visiability of optimiations), or rename FeatureResults to FeatureView to make it more obviously useful.

## Feedback

IanS: so for a ds which doesn't provide random access, you can iterate off a file or use an in-memory cache (taking the appropriate performance hit)

## Status Update - December 15th

### Research

- [RandomAccess](#)
- [Learning From Code: Fast Random Access](#)
- [Faster List Iterator with RandomAccess Interface](#)

### Isolating Resource Maintenance

I have set up ResourceCollection and AbstractResourceCollection to be the following:

- iterator() - updated javadocs
- close( iterator ) - return resources used by the provided iterator
- purge() - close all open iterators

Use is as expected:

```
Iterator iterator = collection.iterator();
    try {
        while( iterator.hasNext();){
            Feature feature = (Feature)
iterator.hasNext();
            System.out.println(
feature.getID() );
        }
    }
    finally {
        collection.close( iterator );
    }
```

And purge is used to clean up after others:

```
Collections.sort( collection );  
collection.purge();
```

All of this is in the Javadocs.

## Feedback on DataFeatureCollection

Please note as it stands right now the existing implementation of DataFeatureCollection is a second class citizen, many of the traditional collection methods are still just stubbed. By extending DataFeatureCollection to extend AbstractResourceCollection everything will just work, but this is not something I will have time for ... if there are any volunteers it would be a great review of my work.

Also implemented two FeatureIterators directly wrapping FeatureWriter and FeatureReader respectively. Change made to DataFeatureCollection.

## Setting things Up

As it stands I am subclassing AbstractResourceCollection to implement ....

- MemoryFeatureCollection

In addition the following will be set up:

- AbstractResourceList - support for resource based lists
- AbstractFeatureList - implementing FeatureList
- SubFeatureCollection - produced by featureCollection.subCollection( Filter ), chaining expected
- OrderedFeatureList - captures the "Natural" ordering of a FeatureCollection
- SortedFeatureList - produced by featureCollection.sort( SortBy ), chaining expected

I have "gasp" introduced two constants SortBy.NATURAL and SortBy.REVERSE that intended to reveal the "Natural Order" (or reverse it) of the datastore, this is the order that is most likely based on FID or a database Key, and thus most likely to be **fast**.

As for capturing the FeatureList ... most likely implemented as a TreeMap by FID, order based on the ID so random access but O(lg N) rather than O(C). I may consider tag teaming a TreeSet of FID with a HashMap on the grounds that Shapefile would be better able to reuse the code.

All other sorted orders will be on permutations of a copy of the TreeSet of FIDS, shapefile with support for attributes indexes can produce their own SortedFeatureList ...

The final bit of the puzzle is this - FeatureCollection ISA Feature, and that has some consequences. Rather than rewrite the same boiler plate code seen duplicated in DataFeatureCollection and DefaultFeatureCollection I am isolating it out in to FeatureState, derived collections (both sorted and sub) will make use of an alternate implementation that delegates attribute handling up to the original, while maintain their own cached idea of size and bounds (cache to be managed using using FeatureEvents).

Finally the existing implementations of DefaultFeatureCollection and DataFeatureCollection advertise themselves as AbstractFeatureCollection to the XML world, and offer up a single attribute (in index 0) which is a normal List containing all of their contents.

## The Future of FeatureState

Aka stuff I won't be able to work on right now - but I would like to adjust FeatureState at a later time to:

- support custom attributes on FeatureCollection
- refer to their contents as "featureMembers" or "featureMember" can't remember which, and return the result as the FeatureList with natural ordering ...
- ensure attribute "boundedBy" is available with the cached value of Bounds
- ensure attribute "name" and "description" are also available and are allowed to be null

This would have consequences for the XML writing code, we have not seen FeatureCollections with attributes thus far, and we reference Feature.getBounds() directly, rather than relying on it to be picked up as a normal Attribute.

As you can see by isolating these decisions to FeatureState we will have less work patching up FeatureCollection later.