

# Feature paging and Query capabilities

<b>Motivation:</b>	Allow efficient paged access to features
<b>Contact:</b>	<a href="#">Gabriel Roldán</a>
<b>Tracker:</b>	<a href="http://jira.codehaus.org/browse/GEOT-1794">http://jira.codehaus.org/browse/GEOT-1794</a>
<b>Tagline:</b>	Access a million of features... one page at a time and efficiently too...

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

- [Motivation](#)
- [Requirements](#)
- [API Changes](#)
  - [Status](#)
- [Tasks](#)
- [Documentation Changes](#)

## Motivation

GeoTools nowadays allows for streaming feature access, meaning that it's possible to efficiently access a very big amount of data with low memory consumption. Yet, for various applications, this is not enough. The ability to page thru the results with different feature iterators, at different times, allows for efficient long lived interactions with the datastore. Examples:

- showing all the feature attributes as a table in a GUI application. Loading all of them at once and storing into memory is not advisable, keeping the reader open does not help since the user may want to scroll both forwards and backwards
- web application showing the feature contents with the classic Google like paged results. Each web request needs to use a different datastore connection, and again, the user can page in both directions.

## Requirements

*Paging requires the following:*

1. a way to specify an offset and a page size
2. a stable output order
3. an efficient paging implementation

Point number 2 is required to make sure the same features are returned given the same page and offset and that the same feature does not appear in two different pages (excluding data changes, if features are deleted or inserted there is no way to keep results consistent). Given that ordering against fields is not always possible, this raises the request of being able to sort against the Feature ID (the only sortable element that's guaranteed to be there) should the user did not ask for a different sorting.

Point number 1 and 3 require some changes in the `Query` interface so that the data store can natively perform the paged request. An naive implementation could do without it, but it would be very inefficient: supposing sorting is supported, we could just open an iterator, skip the first `offset` rows, and then return `limit` rows out.

Of course, if the data set is very big, this is not efficient, as it requires uselessly reading way too many features.

Yet, adding new fields to the Query interface further raises the bar for the implementation of a data store, a bar that's already proven to be too high: various data stores do not correctly implement reprojection or CRS forcing. In order to sustainably extend the Query interface data stores should be allowed to not support parts of it. This requires a concept of Query capabilities.

Query capabilities should be returned by FeatureSource, as opposed to DataStore. The rationale is that they may be FeatureType dependent. Think for example of an aggregating data store, that lumps together different data sources (directory data store is an example). This kind of datastore will have different capabilities depending on what the origin data store can do.

## API Changes

Query interface will sport a new property:

```
interface Query {  
  
    ...  
  
    /**  
     * An indication to skip  
     * offset features before starting  
     * to return features. Cannot be negative,  
     * default value is 0  
     */  
    long getStartIndex();  
}
```

Using a combination of `offset` and the already available `maxFeatures` allows to completely support the typical DBMS support for offset/limit, see [the PostgreSQL case](#) for an example.

FeatureSource will sport a new property as well:

```
interface FeatureSource {  
  
    ...  
  
    /**  
     * Returns the query capabilities for this
```

FeatureSource

```
    */
    QueryCapabilities getQueryCapabilities();
}

/**
 * This is the minimal Query capabilities we
 could come up in order to reliably support
 paging.
 * Yet, the need for a more complete set of
 capabilities is well known and a new
 proposal should be done
 * in order to define the complete set of
 capabilities a FeatureSource should
 advertise.
 */
public abstract class QueryCapabilities {

    /**
     * Is offset supported.
     * A value of true implies ability to have
 a consistent sort order.
     * At least SortBy.NATURAL_ORDER shall be
 supported, and be
     * the default order if a Query with
 offset but no SortBy is issued.
     */
    public boolean isOffsetSupported(){
        return false;
    }
}
```

```
/**
 * Returns whether a list of properties
 can be used as SortBy keys. May include
 current feature type
 * properties as well as @id for sorting
 on the Feature ID.
 * Note, however, that ability to sort by
 the fature id does not necessarily implies
 the same ordering
 * than SortBy.NATURAL_ORDER, though its
 probable they match for datastores where the
 feature id is built
 * up from a primary key.
 */
boolean supportsSorting(SortBy[]){
    return false;
}

/**
 * Returns the list of filters natively
 supported by the underlying storage.
 * Every other filter will be emulated in
 memory (and thus will not enjoy any
 acceleration).
 */
//TODO: consider adding it in the near
future
//FilterCapabilities
getFilterCapabilities();

/**
```

```
    * True if this feature source supports
reprojection
    */
    //TODO: consider adding it in the near
future
    //boolean isReprojectionSupported();

/**
 *
 */
//TODO: consider adding it in the near
future
//boolean isCRSForcingSupported();

/**
 * Can you think of other capabilities you
```

want to be added?

```
    */  
}
```

As a rule, paging is supported only if sorting is supported on at least one `SortBy.NATURAL_ORDER`, and `isOffsetSupported()` returns true.

The following conditions will cause an exception to be thrown:

- query used with offset, but no sort key is supported
- query specifies one or more unsupported sort keys

On the other side, if there are supported sort keys, and offset is supported, then the following should return the expected results:

- query uses offset and eventually maxFeatures, and query does not specify a `SortOrder`, but `QueryCapabilities.isOffsetSupported()` is true, in which case the `NATURAL_ORDER` is applied.
- query uses offset and eventually maxFeatures, and sorting uses one of the supported sort keys

Example usage, trying to extract features in positions between 101 and 110 (included), for a paging ui table:

```
FeatureSource fs = ...;  
    QueryCapabilities caps =  
fs.getQueryCapabilities();  
    DefaultQuery query = new  
DefaultQuery(fs.getFeatureType().getName().g  
etLocalPart());  
    FeatureCollection fc;  
  
    SortBy[] sortBy = new  
SortBy[] {filterFac.sort("@id"),  
SortOrder.ASCENDING});  
    if(caps.supportsSorting(sortBy) {  
        if( caps.isOffsetSupported() ){  
            //optimal route, both offset and  
sorting by the needed field is available  
            query.setOffset(100);  
            query.setMaxFeatures(10);
```

```

        query.setSortBy(sortBy);
        fc = fs.getFeatures(query);
    }else {
        // ops... will have to workaround...
        at least ds provides consistent sort order
        query.setMaxFeatures(110);
        fc = new
        SkippingFeatureCollection(fs.getFeatures(que
        ry), 100);
    }
    }else{
        //no way, can't even sort by the
        required field
    }

```

...

```

    public SkippingFeatureCollection
    implements FeatureCollection {
        /**
         * Given the original feature
        collection, skips <code>skipped</code>
         * elements and then starts returning
        values
         */
        public
        SkippingFeatureCollection(FeatureCollection

```

```
wrapped, long skipped);  
    ...  
}
```

This example shows how a client can use the native offset support, if available, or alternatively roll your own (an alternative approach would be to give up and throw an exception).

## Status

This proposal has been voted on and is being implemented

Voting has not started yet:

- [Andrea Aime](#) +1
- [Ian Turton](#)
- [Justin Deoliveira](#) +1
- [Jody Garnett](#) +1
- [Martin Desruisseaux](#)
- [Simone Giannecchini](#) +0

## 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/ti me		voluntee r needed
--	----------------	--	------	--	---------	--	------------------------------------	--	----------------------

## Documentation Changes

- Developers Guide will need a section on handling QueryCapabilities and Query.offset
- [Data Module](#) from the [Module matrix](#)
- [Home](#)