

ResourceId

Contact:	full name
Tracker:	http://jira.codehaus.org/browse/GEOT-3921
Tagline:	who what where when

- [Description](#)
 - [ResourceId Proposal](#)
 - [Code Examples](#)
 - [Implementation](#)
- [Status](#)
- [Tasks](#)
- [Alternatives Considered](#)
 - [Specification provided by Filter 2.0 and WFS 2.0 Resource ID](#)
 - [Original ResourceId Patch \(jdeolive\)](#)
 - [Just use FeatureId \(jody\)](#)
 - [Clean Data / Query Split \(grolan\)](#)

Children:

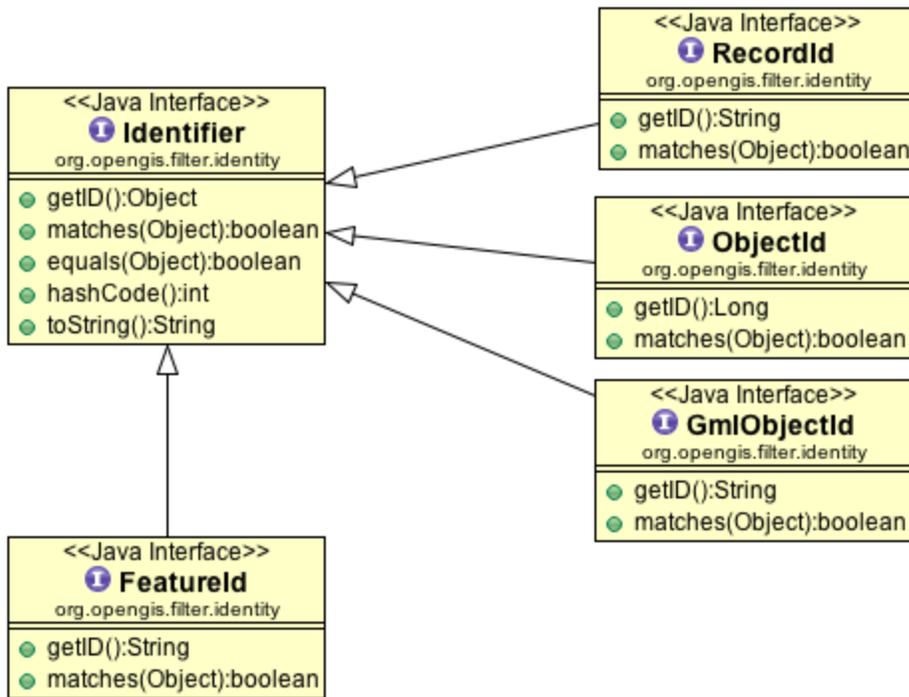
References:

- OGC 09-0261r1 OpenGIS Filter 2.0 Encoding Standard

Description

As part of the WFS 2.0 specification we are offered a greater ability to interact with revisions for of a Feature. This concept is captured as a **ResourceId** as part of this proposal.

Current GeoTools FeatureId



The GeoTools FeatureId comes from an earlier version of these concepts as shown:

- **Id** this represents the concept of an abstract identifier with equals, identity and toString based around a single
- **FeatureId** this is our main concrete implementation which many code just assumes (with a cast)
- **ObjectId** this was drawn from an example in an earlier specification and used to ensure our class breakdown was correct and did not assume **FeatureId**

Note that **FeatureId** and **ObjectId** do not appear anywhere in the WFS 2.0 and Filter 2.0 specifications; these concepts have been orphaned and/or renamed.

ResourceId Proposal

This proposal maintains the clean separation of Data Model / Query Model / MetaData:

Data Model:

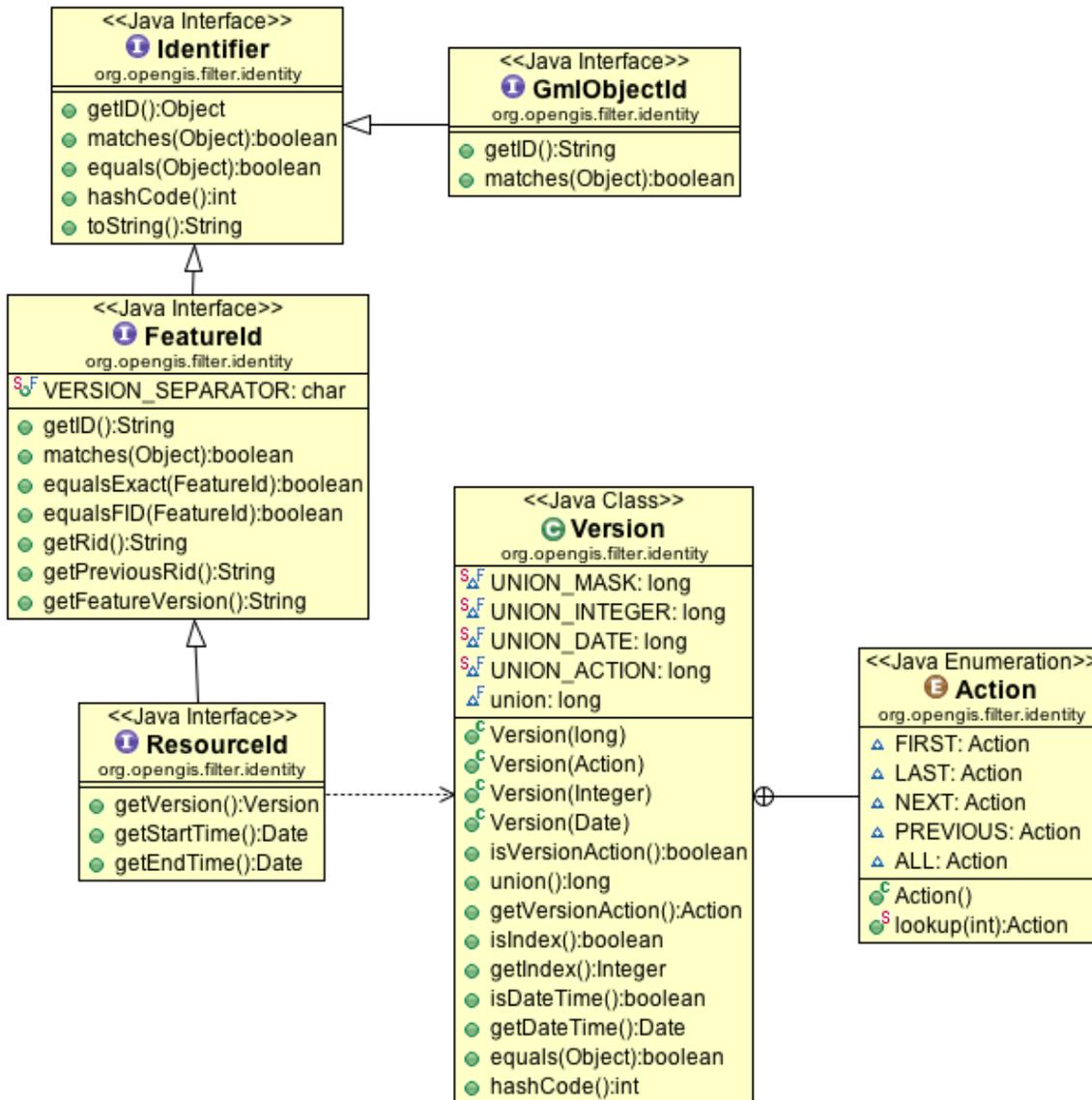
- **FeatureId** updated with additional version information

Query Model:

- **ResourceId** updated to allow a specific record in the history of a Feature to be obtained
- **Query** is updated to allow the use of version
- No changes or extensions to **FeatureSource** are required by this proposal

Metadata:

- No indication of the valid revision or time range is provided as part of the **FeatureType** description or **Info** classes
- **QueryCapabilities** indicates support for **version**; this is your clue that the datastore has a sense of History
- You may determine the time range must be determined by hand using "FIRST", "LAST" and comparing the dates attached to these records



The following "syntactic sugar" methods are added to **Query** after the existing getVersion and setVersion:

```

class Query {
    ...
    /**
     * From WFS Spec: The version attribute
     is included in order to
     * accommodate systems that support
     feature versioning. A value of {@linkplain
     #ALL}
     * indicates that all versions of a
  
```

feature should be fetched. Otherwise

- * an integer, n, can be specified to return the n th version of a

- * feature. The version numbers start at '1' which is the oldest version.

- * If a version value larger than the largest version is specified then

- * the latest version is return. The default action shall be for the query

- * to return the latest version. Systems that do not support versioning

- * can ignore the parameter and return the only version that they have.

*

- * @return the version of the feature to return, or null for latest.

*/

```
public String getVersion();
```

```
/**
```

- * Set the version of features to retrieve where this is supported by the

- * data source being queried.

- * @param version

- * @see #getVersion() getVersion() for explanation

- * @since 2.4

*/

```
public void setVersion(String version);
```

```
// Syntactic Sugar
```

```
public void setVersion( int index );
```

```
public void setVersion( Date date );  
public void setVersion( Version.Action  
action );  
public void setVersion( Date startTime,  
Date endTime );  
public void setVersion( ResourceId
```

```
history )
    ...
}
```

Code Examples

Checking if version is supported:

```
if( queryCapabilities.isVersionSupported()
){
    ...
}
```

Direct use of **ResourceId** during a Filter Id query:

```
Set<FeatureId> selectedIds = new
HashSet<FeatureId>();

    // defaults to latest record for
CITY.123
    selectedIds.add(
ff.featureId("CITY.123") );

    // grab the previous record for CITY.123
for comparison
    selectedIds.add(
ff.resourceId("CITY.123",VersionAction.PREVI
OUS) );

    // grab city size in the 1930s for
historical comparison
```

```
    DateFormat dfm = new
SimpleDateFormat("yyyy-MM-dd");
    Date startDate =
dfm.parse("1930-01-01");
    Date endDate = dfm.parse("1940-01-01");
    selectedIds.add(
ff.resourceId("CITY.123", startDate, endDate
) );

    // grab a specific record by version
    selectedIds.add(
ff.resourceId("CITY.123","AH874C9814F9") );

    // grab a specific record by Date
    selectedIds.add(
ff.resourceId("CITY.123",dfm.parse("1983-04-
11")) );

    Filter filter = ff.id( selectedIds );
```

```
SimpleFeatureCollection collection =  
featureSource.getFeatures( filter );
```

Finding the complete history for a record:

```
Filter filter = ff.id(  
ff.resourceId("CITY.123",VersionAction.ALL)  
);  
SimpleFeatureCollection collection =  
featureSource.getFeatures( filter );
```

Grabbing the history for an area::

```
Filter filter = ff.bbox(  
ff.property("the_geom"), ff.literal(  
envelope ) );  
Query query = new Query( "CITY", filter  
);  
query.setVersion("ALL");  
  
SimpleFeatureCollection collection =  
featureSource.getFeatures( query );
```

As shown above the Query version information can be provided (it defaults to null indicating that the "LAST" record should be returned). The constants described by VersionAction are supported with the useful ones being FIRST, LAST and ALL. These values are considered to be "the default" applied is applied to any normal filter elements including bbox.

An interesting wrinkle is that this gives us a second way to pull out the complete history for a record.

```
Filter filter = ff.id(
ff.featureId("city.123") );
    Query query = new Query( "CITY", filter
);
    query.setVersion("ALL");
    query.setPropertyNames(Query.NO_NAMES);
// only return fids

    SimpleFeatureCollection collection =
featureSource.getFeatures( query );
```

In this case the Query version range of "ALL" is used as the default when understanding the FeatureId reference "city.123".

The results are shown in an extension of the property datastore format that allows a compound "fid|rid|start|end" information.

```
city.123|AH874C9814F9|1930-02-27|1935-11-07=  
city.123|D9134BCE9348|1935-11-07|1941-05-23=  
city.123|9274DF937364|1941-05-23|1955-02-14=  
city.123|A3412349274D|1955-02-14|1974-11-21=  
city.123|8EF783894362|1974-11-21|1983-04-11=  
city.123|736235648323|1983-04-11|1993-09-27=  
city.123|E8E779CD0789|1993-09-27|2007-08-13=  
  city.123|83656C84AB12|2007-08-13|  
=  
=
```

(In this case no attributes are returned given the provided query)

This same technique can be used to determine history for a dataset:

```
Query query = new Query( "CITY" );  
  query.setVersion("ALL");  
  query.setPropertyNames(Query.NO_NAMES);  
  // only return fids  
  SimpleFeatureCollection collection =  
  featureSource.getFeatures( query );
```

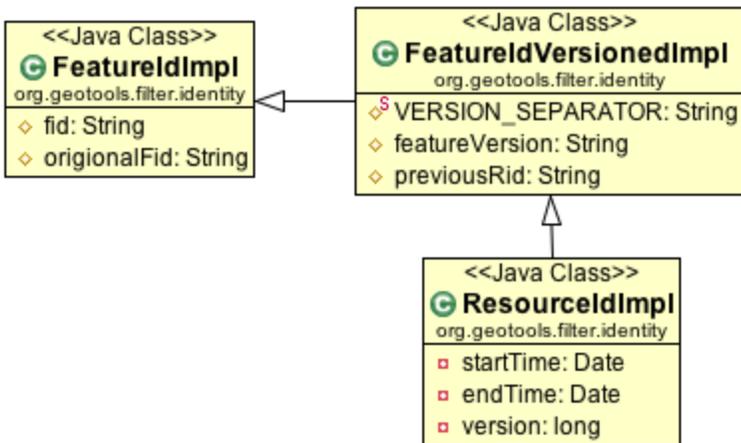
The returned FeatureCollection consists of FeatureIds; with the natural ordering sorting the results into ChangeSets:

```

city.123 | AH874C9814F9 | 1930-02-27 | 1935-11-07=
city.093 | AH874C9814F9 | 1930-02-27 | 1935-11-07=
city.926 | AH874C9814F9 | 1930-02-27 | 1935-11-07=
city.246 | AH874C9814F9 | 1930-02-27 | 1935-11-07=
city.123 | D9134BCE9348 | 1935-11-07 | 1941-05-23=
city.246 | D9134BCE9348 | 1935-11-07 | 1941-05-23=
...

```

Implementation



The following is added to FilterFactory:

```

interface FilterFactory {
    ..
    // Identity
    FeatureId featureId(String id);
    GmlObjectId gmlObjectId(String id);

    // (Not required as it is only used by
    DataStore Implementations)
    // FeatureId featureId(String fid,
    String featureVersion, String previousRid);

    // Query
    ResourceId resourceId(String fid, String
    featureVersion, Version version );
    ResourceId resourceId(String fid, Date
    startTime, Date endTime);
    ...
}
interface FilterFactory2 {
    Id id( FeatureId ...fids);
}

```

Status

This proposal is under discussion; with work slated for Oct 24th.

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

Community support:

- grolan +1
- mleslie +1 (with patch)

Tasks

	no progress	✓	done	✗	impeded	⚠	lack mandate /funds/time	?	volunteer needed
--	-------------	---	------	---	---------	---	--------------------------	---	------------------

- ✓ Review options with community
 - ✓ ResourceId option
 - ✓ FeatureId option
 - ✓ compromise and combine
- ✓ ResourceID implementation
 - ✓ Interface and Factory Change
 - ✓ Initial implementation
 - ✓ Hook up to FES2 bindings for parser / encoder
 - ✓ Review Patch (thanks mleslie)
- ✓ Evaluation impact on downstream code
- ✓ <http://jira.codehaus.org/browse/GEOT-3921>
 - ✓ Review impact on GeoServer WFS2 work
 - ✓ Review impact on GeoGit project
- ✓ Documentation update
 - ✓ Update the user guide

Nice to have:

- ⚠ Update PropertyDataStore reference implementation
 - <https://jira.codehaus.org/browse/GEOT-3939>
 - Update QueryCapabilities
 - Update FeatureId parsing / generation
 - Update resource id matching code
- Update PropertyDataStore tutorial

Alternatives Considered

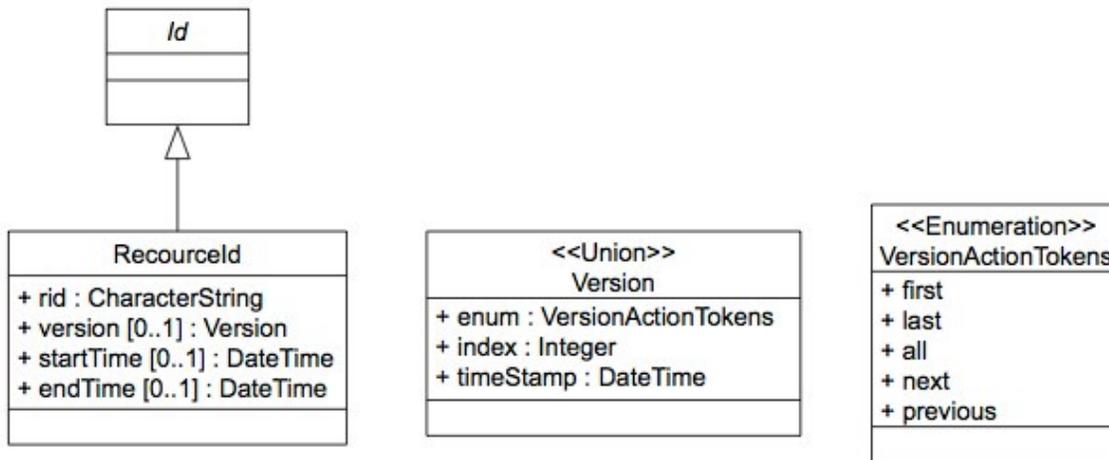
The following options have been considered during the course of this proposal.

Specification provided by Filter 2.0 and WFS 2.0 Resource ID

Filter 2.0

The Filter 2.0 specification defines the concept of ResourceID:

- ResourceId: Implements the ResourceId operator with the rid parameter to allow predicates to be written that allow a specific resource to be queried.
- Version Navigation: Implements ResourceId operator with the parameters that allow versions of resources to be queried (version, startTime, endTime)
- Formally the WFS 2.0 specification allows you to only "select" one of Filter, ResourceID or BBox at a time. This restriction is lifted for the GeoTools implementation of Filter.



(Figure 9 - ResourceId from OGC 09-026r1 OpenGIS Filter Encoding 2.0 Encoding Standard)

The Filter 2.0 standard in **7.11.1 Object identifiers** outlined a few key concepts in relation to Figure 9:

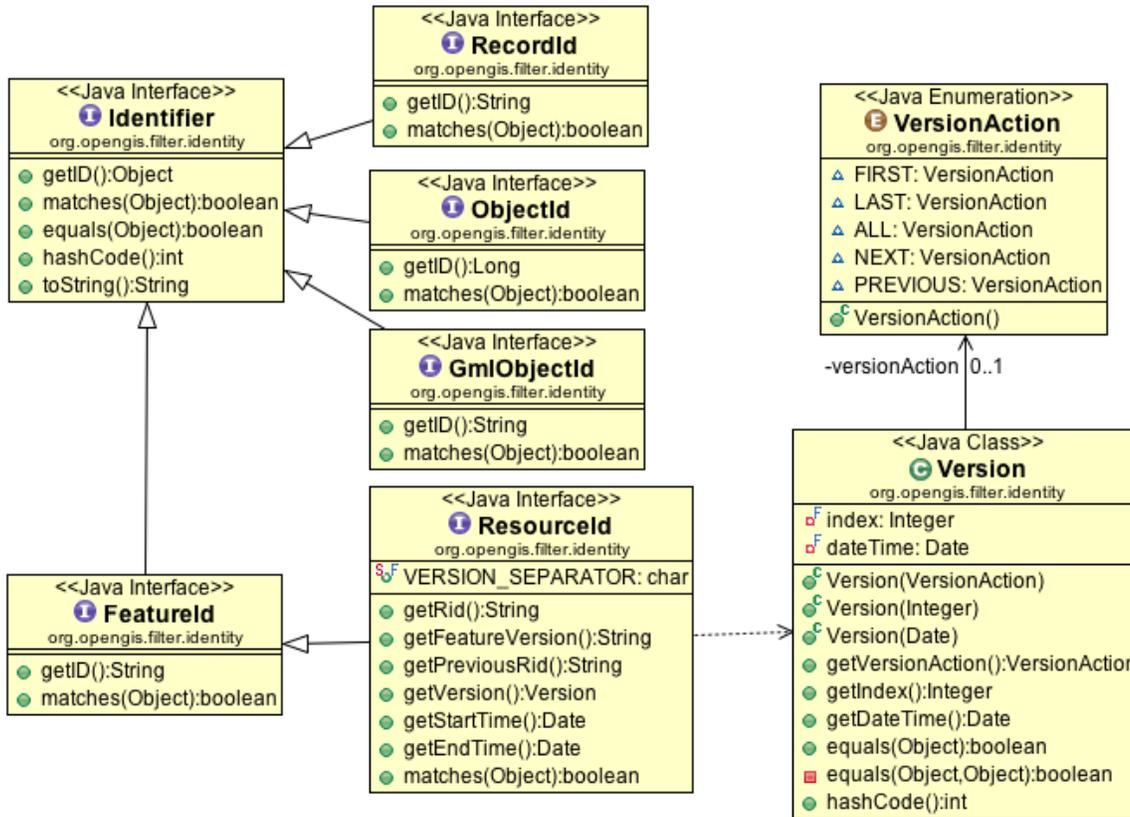
- **Id** - forms an "AbstractIdType" allowing different kinds of identifiers to be slotted in
- **ResourceId** is the only example provided by Filter 2.0 - it makes use of rid, version and a startTime/endTime combo
- **Version** is a small data object containing an index; timestamp and an indication of "versionAction" defining use (FIRST, LAST, ALL, NEXT, PREVIOUS)

WFS 2.0

The WFS 2.0 specification is a bit kinder; section 7.2 Resource Identifiers provides an outline of expectations:

- "Resource identifiers are not intended to associate WFS resources with real world objects and the values do not have to be meaningful outside the scope of a web feature service instance."
- "TYPENAMES parameter may be omitted because each feature instance can be identified by its resource identifier"

Original ResourceId Patch (jdeolive)



The first option is to make ResourceId a straight extension of FeatureId.

- This has been implemented by Justin as and is available as a patch
- Take appropriate measures to prevent the confusion of FeatureId and ResourceId when parsing / encoding

Pros: This is a clear reflection of the Filter 2.0 ResourceID schema

Cons: Duplication of concept, hard to compare FeatureID to ResourceId (reporting back from Mark Leslie who has been working on GeoGit)

ResourceId definition:

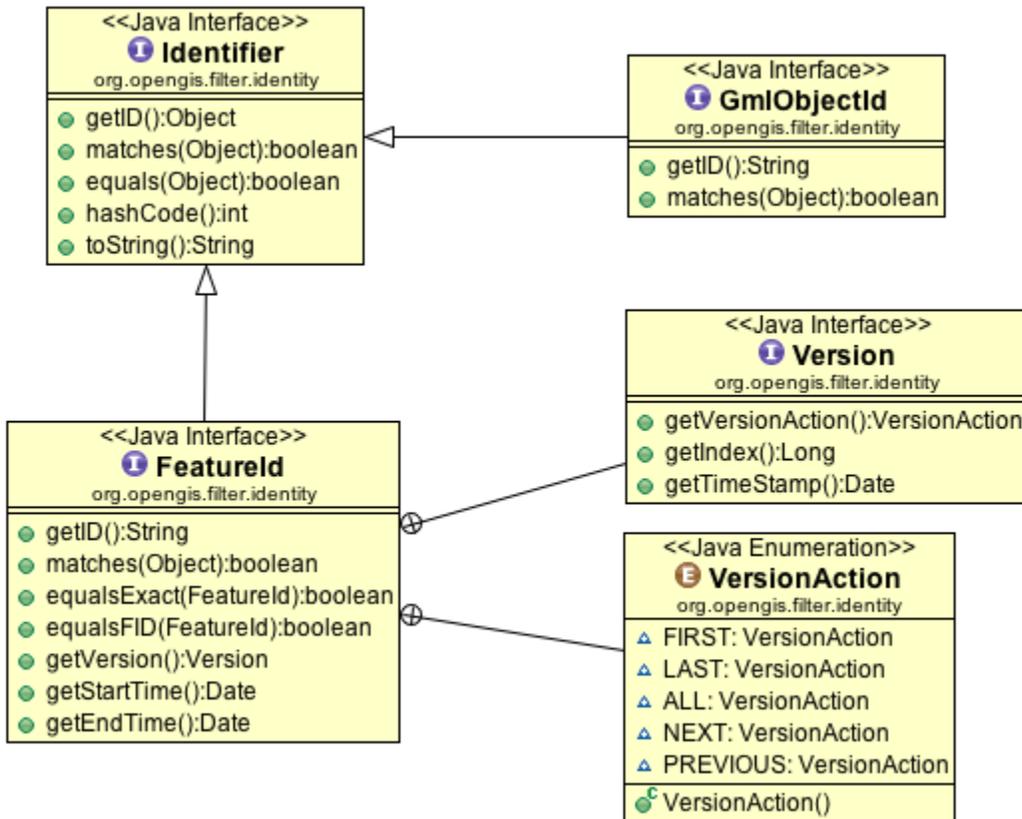
```
@XmlElement("FeatureId")
public interface ResourceId extends
FeatureId {
    public static final char
VERSION_SEPARATOR = '@';
    String getRid();
    String getFeatureVersion();
    String getPreviousRid();
    Version getVersion();
    Date getStartTime();
    Date getEndTime();
    boolean matches(Object resource);
}
public final class Version {
    private final VersionAction
versionAction;
    private final Integer index;
    private final Date dateTime;

    public Version(final VersionAction
action);
    public Version(final Integer index);
    public Version(final Date dateTime);
    public VersionAction getVersionAction();
    public Integer getIndex();
    public Date getDateTime();
}
public enum VersionAction { FIRST, LAST,
ALL, NEXT, PREVIOUS; }
```

From ResourceIdTypeBinding parse:

```
ResourceId resourceId =
factory.resourceId(fid, featureVersion,
previousRid, version, startTime, endTime);
```

Just use FeatureId (jody)



This option is proposed by Jody as a simplification of the current statue of play.

1. Remove **ObjectId** and **RecordId** as they are not used in our case base (it is too bad that GmlObjectId did not make use of ObjectId when it was added)
 - Assume the specification body is out to annoy us (by introducing a new form of Identifier each release)
2. Rebel as we value stability over the latest ever changing OGC standard
 - Update FeatureId interface with the new concepts fid plus version and a startTime/endTime
 - Ensure the javadocs indicate this is a representation of ResourceId from WFS 2.0
3. Update FilterFactory to use an efficient implementation as appropriate
 - FeatureIdImpl when only a fid string is provided (with placeholder values returned for version, startTime / endTime)
 - FeatureIdResourceImpl when a fid and version, startTime / endTime is provided
4. Carefully consider the equals implementations so the above are compatible
 - **equalsExact** compares rid and version range (field by field equals)
 - **equalsFid** compares just the feature id

- Bonus
 - `getTypeName()` determination of tupe name from a resource id (in order to cut down on the hacks)

Pros: Does not break client code, Easier to use

Cons: Not implemented, addresses more that ResourceId (such as equalsExact and getTypeName())

```
public interface FeatureId extends
Identifier {
    String getID();
    boolean matches(Object feature);
    boolean equalsExact( FeatureId
featureId);
    boolean equalsFID( FeatureId fatureId);
    Version getVersion();
    Date getStartTime();
    Date getEndTime();
    public interface Version {
        VersionAction getVersionAction();
        Long getIndex();
        Date getTimeStamp();
    }
    public enum VersionAction
{FIRST, LAST, ALL, NEXT, PREVIOUS}
}
```

Code example based on ResourceIdTypeBinding parse:

```
Set<FeatureId> selectedIds = new
HashSet<FeatureId>();

// defaults to latest record for
CITY.123
selectedIds.add(
```

```
ff.featureId("CITY.123") );

    // grab the previous record for CITY.123
for comparison
    selectedIds.add(
ff.featureId("CITY.123",VersionAction.PREVI
US) );

    // grab city size in the 1930s for
historical comparison
    DateFormat dfm = new
SimpleDateFormat("yyyy-MM-dd");
    Date startDate =
dfm.parse("1930-01-01");
    Date endDate = dfm.parse("1940-01-01");
    selectedIds.add(
ff.featureId("CITY.123", startDate, endDate
) );

    // grab a specific record by version
    selectedIds.add(
ff.featureId("CITY.123","AH874C9814F9") );

    // grab a specific record by Date
    selectedIds.add(
ff.featureId("CITY.123",dfm.parse("1983-04-1
1")) );

    Filter filter = ff.id( selectedIds );
```

```
SimpleFeatureCollection collection =  
featureSource.getFeatures( filter );
```

Finding the complete history for a record:

```
Set<ResourceId> selectedIds = new  
HashSet<ResourceId>( );  
    selectedIds.add(  
ff.resourceId( "CITY.123", VersionAction.ALL)  
);  
    Filter filter = ff.id( selectedIds );  
  
    SimpleFeatureCollection collection =  
featureSource.getFeatures( filter );
```

Grabbing the history for an area::

```
Filter filter = ff.bbox(  
ff.property( "the_geom" ), ff.literal(  
envelope ) );  
    Query query = new Query( "CITY", filter  
);  
    query.setVersion( "ALL" );  
  
    SimpleFeatureCollection collection =  
featureSource.getFeatures( query );
```

As shown above the Query version information can be provided (it defaults to null indicating that the "LAST" record should be returned). The constants described by VersionAction are supported with the useful ones being FIRST, LAST and ALL. These values are considered to be "the default" applied is applied to any normal filter elements including bbox.

An interesting wrinkle is that this gives us a second way to pull out the complete history for a record.

```
Filter filter = ff.id(
ff.featureId("city.123") );
    Query query = new Query( "CITY", filter
);
    query.setVersion("ALL");
    query.setPropertyNames(Query.NO_NAMES);
// only return fids

    SimpleFeatureCollection collection =
featureSource.getFeatures( query );
```

In this case the Query version range of "ALL" is used as the default when understanding the FeatureId reference "city.123".

The results are shown in an extension of the property datastore format that allows a compound "fid|rid|start|end" information.

```
city.123|AH874C9814F9|1930-02-27|1935-11-07=  
city.123|D9134BCE9348|1935-11-07|1941-05-23=  
city.123|9274DF937364|1941-05-23|1955-02-14=  
city.123|A3412349274D|1955-02-14|1974-11-21=  
city.123|8EF783894362|1974-11-21|1983-04-11=  
city.123|736235648323|1983-04-11|1993-09-27=  
city.123|E8E779CD0789|1993-09-27|2007-08-13=  
    city.123|83656C84AB12|2007-08-13|  
=  
=
```

(In this case no attributes are returned given the provided query)

This same technique can be used to determine history for a dataset:

```
Query query = new Query( "CITY" );  
    query.setVersion("ALL");  
    query.setPropertyNames(Query.NO_NAMES);  
// only return fids  
    SimpleFeatureCollection collection =  
featureSource.getFeatures( query );
```

The returned FeatureCollection consists of FeatureIds; with the natural ordering sorting the results into ChangeSets:

```
city.123 | AH874C9814F9 | 1930-02-27 | 1935-11-07=  
city.093 | AH874C9814F9 | 1930-02-27 | 1935-11-07=  
city.926 | AH874C9814F9 | 1930-02-27 | 1935-11-07=  
city.246 | AH874C9814F9 | 1930-02-27 | 1935-11-07=  
city.123 | D9134BCE9348 | 1935-11-07 | 1941-05-23=  
city.246 | D9134BCE9348 | 1935-11-07 | 1941-05-23=  
...
```

Clean Data / Query Split (groidan)

Gabriel has proposed a clean split between:

- Data Model using **FeatureId** to describe each record / feature returned
- Query using **ResourceId** to carefully request a date range or access to the next / previous record in a sequence

This solution is largely the same as the previous one; it is slightly more complicated in that there are two concepts (FeatureId and ResourceId) however it is more explicit with its separation of concerns for greater clarity.

```

interface FeatureId extends Identifier {
    String getID();
    String getVersion();
    /**
     * @return <ID>[@<version>]
     */
    String rid();
}
interface ResourceId extends Identifier {
    String getID();
    boolean matches(Object feature);
    Version getVersion();

    Date getStartTime();
    Date getEndTime();

    public interface Version {
        VersionAction getVersionAction();
        Long getIndex();
        Date getTimeStamp();
    }
    public enum VersionAction
{FIRST, LAST, ALL, NEXT, PREVIOUS}
}

```

Code example based on ResourceIdTypeBinding parse:

```

Set<ResourceId> selectedIds = new
HashSet<ResourceId>();

```

```
// defaults to latest record for
CITY.123
    selectedIds.add(
ff.resourceId("CITY.123") );

    // grab the previous record for CITY.123
for comparison
    selectedIds.add(
ff.resourceId("CITY.123",VersionAction.PREVI
OUS) );

    // grab city size in the 1930s for
historical comparison
    DateFormat dfm = new
SimpleDateFormat("yyyy-MM-dd");
    Date startDate =
dfm.parse("1930-01-01");
    Date endDate = dfm.parse("1940-01-01");
    selectedIds.add(
ff.resourceId("CITY.123", startDate, endDate
) );

    // grab a specific record by version
    selectedIds.add(
ff.resourceId("CITY.123",876123586793245687)
);

    // grab a specific record by Date
    selectedIds.add(
ff.resourceId("CITY.123",dfm.parse("1983-04-
11")) );
```

```
Filter filter = ff.id( selectedIds );
```

```
SimpleFeatureCollection collection =  
featureSource.getFeatures( filter );
```

Finding the complete history for a record:

```
Set<ResourceId> selectedIds = new  
HashSet<ResourceId>( );  
    selectedIds.add(  
ff.resourceId( "CITY.123", VersionAction.ALL)  
);  
    Filter filter = ff.id( selectedIds );  
  
    SimpleFeatureCollection collection =  
featureSource.getFeatures( filter );
```

Grabbing the history for an area::

```
// not supported by this proposal
```