Support GetGMLObject

Motivation:	Support GetGMLObject operation of WFS 1.1
Contact:	Justin Deoliveira
Tracker:	http://jira.codehaus.org/browse/GEOT-XXXX
Tagline:	

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

Status

This proposal is closed and has been implemented.

Voting has not started yet:

	Vote	Alternative 1	Alternative 2
Andrea Aime			
Ian Turton			
Justin Deoliveira	+1	✓	
Jody Garnett	+1	✓	
Martin Desruisseaux			
Simone Giannecchini			

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

Description

WFS 1.1 comes with the addition of a new operation called **GetGmlObject**. This is relatively simple in nature. Given the identifier of a "GML object" (be it a Feature or a Geometry) return that object.

The following is an example of a GetGmlObject request in which an individual feature with the id "PrimitiveGeoFeature.1" is being requested:

```
<wfs:GetGmlObject xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
    version="1.1.0" service="WFS">
    <ogc:GmlObjectId>PrimtiveGeoFeature.1</ogc:GmlObjectId>
</wfs:GetGmlObject>
```

The response to this request is the feature encoded in gml:

```
<sf:PrimitiveGeoFeature xmlns:sf="http://cite.opengeossptial.org/gmlsf"
xmlns:gml="http://www.opengis.net/gml" gml:id="PrimitiveGeoFeature.1">
  <gml:name>name-01</gml:name>
  <gml:description>description-01</description>
  <sf:pointProperty>
    <gml:Point xmlns:gml="http://www.opengis.net/gml" gml:id="point.1">
      <gml:pos>12 34</gml:pos>
    </gml:Point>
  </sf:pointProperty>
  <sf:intProperty>1</sf:intProperty>
  ...
</sf:PrimitiveGeoFeature>
```

The GetGmlObject operation can also be used to retreive individual geometries. The following is an example in which an individual point with the id "point.1" is being requested:

```
<wfs:GetGmlObject xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
    version="1.1.0" service="WFS">
    <ogc:GmlObjectId>point.1</ogc:GmlObjectId>
</wfs:GetGmlObject>
```

The response is the point encoded in gml:

```
<gml:Point xmlns:gml="http://www.opengis.net/gml" gml:id="point.1">
  <gml:pos>12 34</gml:pos>
</gml:Point>
```

Providing support in Geotools for this operation is broken out into two alternatives which will be explained in greater detail in the following sections. The first alternative requires a DataStore api extension which will provide additional datastore api for querying specifically for an object. The second alternative requires is implemented entirely with existing api and hints.

Proposal

Alternative 1: Extending the DataStore API

As stated above, the first alternative is centered around an extension to the DataStore api. An additional interface called `GmlObjectStore` is added:

```
/**
 * Interface providing lookup operations for gml objects.
 * <p>
 * This interface may be implemented by data stores to provide an additional operation

 * for looking object a "gml object" directly. A gml object is typically a feature
 * or a geometry.
 * </p>
 */
public interface GmlObjectStore {

    /**
     * Looks up an object by its gml id.
     * <p>
     * This method returns <code>null</code> if no such object exists.
     * </p>
     * @param id The id of the object, must not be <code>null</code>.
     * @param hints Hints to use while querying
     *
     * @return The gml object, or <code>null</code> if one could not be found
     * matching the specified id.
     *
     * @throws IOException Any I/O errors that occur.
     */
    Object getGmlObject( GmlObjectId it, Hints hints ) throws IOException;
}
```

The interface adds a single method which takes a `GmlObjectId` (from the filter model), and produces an object. The object being a `Feature`, or a `Geometry`.

The idea is that a DataStore which is capable of providing these lookup operations can implement this additional interface. A client wishing to use this functionality must do an instanceof check against the interface.

As an example consider the implementation of the GetGmlObject operation in GEoServer:

```
class GetGmlObject {

  ...

  Object run( GetGmlObjectType request ) {

     //look up the datastore
     DataStore dataStore = findDataStore( request );

     if ( dataStore instanceof GmlObjectAware ) {
         GmlObjectAware gmlObjectAwareDataStore = (GmlObjectAware) dataStore;

         //get the id from the request
         GmlObjectId id = request.getGmlObjectId();

         //get the object
         return gmlObjectAwareDataStore.getGmlObject( id  );
     }
     else {
          //data store does not support hte operation
          throw new WFSException( "DataStore: " + dataStore + " does not support
GetGmlObject" );
     }
  }
}
```

Alternative 2: Hints

An alternative to extending the datastore api is to use a combination of the old api and the hint system. A hint called "GML_OBJECT_ID" is added:

```
class Hints {

  ...
  /**
   * The gml id of an object in a datastore query.
   * <p>
   * This maps directly to a GmlObjectId element in a wfs query.
   * </p>
   */
  public static final Hints.Key GML_OBJECT_ID = new Key( GmlObjectId.class );

  ...
}
```

A client must supply this hint to a data store via a `Query` object. Again consider the implementation of the GetGmlObject operation in GeoServer:

```
class GetGmlObject {

  ...

  Object run( GetGmlObjectType request ) {

     //look up the datastore
     DataStore dataStore = findDataStore( request );

     //build a query
     Query query = new DefaultQuery();
     ...

     //set the gmlObjectId hint
     GmlObjectId id = request.getGmlObjectId();
     query.getHints().put( Hints.GML_OBJECT_ID, id );

     ...
  }
}
```

At this point it is up to the particular data store implementation to honor the hint passed in. Now this is where things get a big fuzzy. In the case where a particular Feature is being requested the answer is simple. Just return a single feature:

```
  //execute the query
  FeatureReader reader = dataStore.getFeatureReader( query, Transaction.AUTO_COMMIT );
  try {
    if ( reader.hasNext() ) {
        //return the feature
        return reader.next();
    }
    else {
      throw new WFSException( "No such object found for id: " + id );
    }
  }
  finally {
    reader.close();
  }
```

However, what about the case where a particular Geometry is being requested? The existing data store api limits us to returning features. So for this case we must wrap the geometry in a feature.

```
//execute the query
FeatureReader reader = dataStore.getFeatureReader( query, Transaction.AUTO_COMMIT );
try {
  if ( reader.hasNext() ) {
      //read the feature
      SimpleFeature feature = reader.next();

      //return the geometry
      return feature.getDefaultGeometry();
  }
  else {
    throw new WFSException( "No such object found for id: " + id );
  }
}
finally {
  reader.close();
}
```

It gets a bit more complicated in that the client code must know which type of object corresponds to the requested id, since it needs to know if it should unwrap the feature or not. A possible solution to this is to use "user data" for the data store to report back the type of the object:

```
//execute the query
FeatureReader reader = dataStore.getFeatureReader( query, Transaction.AUTO_COMMIT );
try {
  if ( reader.hasNext() ) {
      //read the feature
      SimpleFeature feature = reader.next();

      //check the gml object type
      String type = feature.getUserData().get( "gmlObjectType" );
      if ( "Geometry".equals( type ) ) {
          //unwrap
          return feature.getDefaultGeometry();
      }

      //return the feature itself
      return feature;
  }
  else {
    throw new WFSException( "No such object found for id: " + id );
  }
}
finally {
  reader.close();
}
```

API Changes

Alternative 1

Addition of the `GmlObjectAware` interface above.

Alternative 2

Addition of the `GML_OBJECT_ID` hint.