deegree Web Feature Service v2.2

lat/lon GmbH
Aennchenstr. 19
53177 Bonn
Germany
Tel ++49 - 228 - 184 96-0
Fax ++49 - 228 - 184 96-29
info@lat-lon.de
www.lat-lon.de

Dept. of Geography
Bonn University
Meckenheimer Allee 166
53115 Bonn

Tel. ++49 228 732098

Change log

Date	*Description*	*Author*
2007-01-09	Update using new formatting style	Markus Müller
2007-05-22	Update on new release and demo data	Hanko Rubach
2007-05-24	Update of Philosopher example installation	Hanko Rubach
2008-01-07	updated Chapter “Setting up WFS-Gazetteer (WFS-g)” as ISO will be supported; explained XSLT inFilter and outFilter	Hanko Rubach
2008-04-15	Update to v2.2	Judit Mays

Table of Contents

Index of Tables

Illustration Index

1 Introduction

deegree is a Java Framework offering the main building blocks for Spatial Data Infrastructures (SDIs). Its entire architecture is developed using standards of the Open Geospatial Consortium (OGC) and ISO Technical Committee 211 – Geographic information / Geoinformatics (ISO/TC 211). deegree encompasses OGC Web Services as well as clients. deegree is Free Software protected by the GNU Lesser General Public License (GNU LGPL) and is accessible at http://www.deegree.org.

deegree2 is the new release of deegree supporting a number of features that deegree1 was not able to handle. This documentation describes setup and configuration of the deegree Web Feature Service (WFS), an implementation of OGC's Web Feature Service Implementation Specification 1.1.0.

deegree's WFS is able to serve vector and attribute data from different data sources (backends) and deliver it to any client that is able to perform WFS compliant HTTP-GET or POST requests.

Currently supported backends are[1]:

- PostgreSQL / PostGIS
- Oracle (Spatial / Locator)
- arbitrary SQL databases that allow JDBC-connections - using deegree's own quadtree implementation as a spatial index
- ESRI Shapefiles
- ArcSDE

The following WFS requests are supported[2]:

- GetCapabilities
- DescribeFeatureType
- GetFeature
- Transaction
- GetFeatureWithLock
- LockFeature

[1] see chapter 4 for details

[2] see appendix A for details

Besides the WFS, deegree comprises a number of additional services and clients. A complete list of deegree components can be found at:

`http://www.lat-lon.de` → Products

Downloads of packaged deegree components can be found at:

`http://www.deegree.org` → Download

deegree's Web Feature Service offers great flexibility regarding it's configuration and output formats. It works with a wide range of physical data sources and server environments. The configuration of WFS is similar to the configuration of other deegree Web Services and requires the editing of XML files which controls the functionality of the server.

The web services of deegree are realized as Java modules controlled by one central servlet (the “dispatcher”). This servlet has to be deployed to the respective web server/servlet engine. Most of the common web servers support servlet technology, thus making deegree a universal product. The Apache-Tomcat 5.5 Servlet-Engine is recommended due to its widespread use and its status as an open-source product.

2 Download / Installation

2.1 Prerequisites

For deegree2.2 Web Feature Service to run you need:

- Java (JRE or JSDK) version 1.5.x
- Tomcat 5.5.x

For installation of these components refer to the corresponding documentation at java.sun.com and tomcat.apache.org.

2.2 deegree Web Feature Service release

deegree Web Feature Service can be downloaded from http://www.deegree.org. The release is packed as a WAR-archive. Simply put this file into your `$TOMCAT_HOME$/webapps` directory and (re-)start Tomcat. The installation of deegree WFS is already done with this.

Note: It is also possible to extract the WAR archive into another place of your computer and direct Tomcat to this place. Because of this possibility, in the remainder of this document, the directory you extracted the files to is referred to as `$wfs_home$` (=`$TOMCAT_HOME$/webapps/deegree-wfs` in the standard case).

Your `$wfs_home$` will contain the following structure:

directory	**Content**
./WEB-INF	Required by Tomcat, containing all libraries, configuration- and data-files
./WEB-INF/conf/scripts	SQL scripts for creating and filling example database
./WEB-INF/conf/tools	tools for generating featuretype definitions, etc.
./WEB-INF/conf/wfs	WFS configuration files
./WEB-INF/data/	Example Shapefiles

Table 1: Directory structure of the WFS release

2.3 Testing the installation

After deploying the deegree-wfs into the tomcat, you should be able to start the following link:

http://localhost:8080/deegree-wfs/

As a first test it is possible e.g. to request a GetCapabilities on the WFS. Over the 'Test WFS' link you can check DescribeFeature. The most interesting 'tool' is the generic client which offers the opportunaty to send any request to the WFS. You find some predefined ones (demo_... .xml) in 'example' dropdown menue which work right from the beginning with the demo datasets.

Please keep in mind, that this demo WFS is preconfigured with some simple datasets (shape, hsqldb). Additional you find instructions on how to set up a complex gml application schema (philosopher example) or a WFS-G (gazetteer) using a postgis database. Therefore some example requests won't work until you adapted the configuration.

3 Architecture

Figure 1 shows the overall architecture of the modules involved in the deegree WFS. The HTTP interface is realized by a servlet that has to be registered into a servlet engine like Tomcat or Jetty. The servlet chooses a handler class depending on the incoming request that delegates it to the responsible service (in this case the WFService). Depending on the requested feature type(s) the WFService decides which datastore is responsible for handling it. The central configuration document of the deegree WFS is based on a OGC WFS 1.1.0 capabilities document (Appendix B) plus a few custom tags.

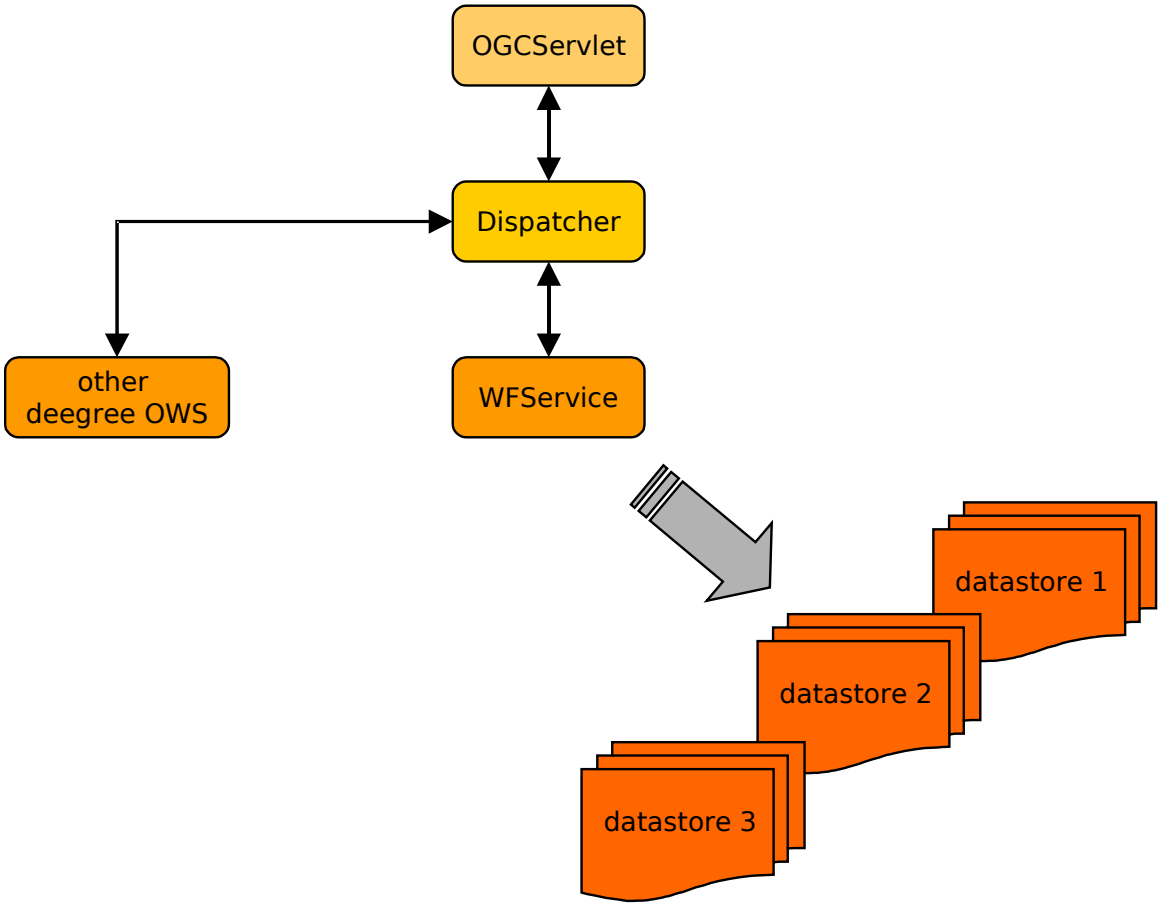

Figure 1: deegree WFS architecture overview

4 Basic configuration

4.1 Structure of the configuration files

The following figure shows the relationships between the different configuration files that determine the behaviour of the deegree WFS.

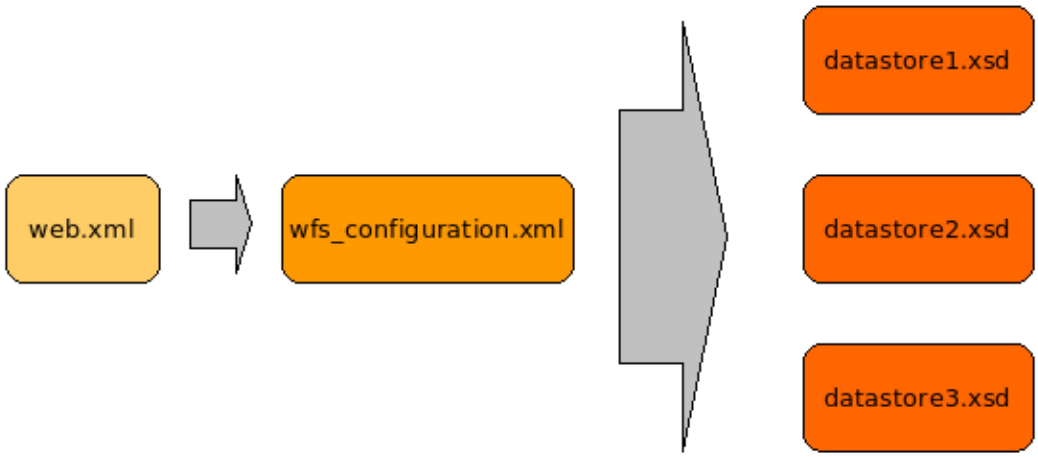

Figure 2: Files affecting the deegree WFS configuration

The WFS-configuration/capabilities document and datastore configuration files will be described in detail later.

4.2 The deegree WFS configuration document

The format of the central configuration document of the deegree WFS is based on the OGC WFS 1.1.0 capabilities document plus a few custom elements. Most important is the `<deegreewfs:deegreeParams>` section, which is the first element inside the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:WFS_Capabilities xmlns:deegree="http://www.deegree.org/wfs"
  xmlns:ows="http://www.opengis.net/ows" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wfs="http://www.opengis.net/wfs" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1.0" updateSequence="0">
  <!-- ================================================================= -->
  <!-- DEEGREE PARAMETERS                                                -->
  <!-- ================================================================= -->
  <deegree:deegreeParams>
    <!-- mandatory; used as the default URL for omitted DCP-elements (in the
    OperationMetadata section) -->
    <deegree:DefaultOnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:type="simple"
```

```
        xlink:href="http://localhost:8080/deegree-wfs/services" />
    <!-- optional; default = 100 (MB); cache size available for storing feature
    instances in memory, not implemented yet -->
    <deegree:CacheSize>250</deegree:CacheSize>
    <!-- optional; default = 30 (seconds); maximum time allowed for the execution
    of a request  -->
    <deegree:RequestTimeLimit>120</deegree:RequestTimeLimit>
    <!-- optional; default = same directory as configuration; list of directories
    to be scanned for featuretypes/datastores to be served by the WFS -->
    <deegree:DataDirectoryList>
       <deegree:DataDirectory>featuretypes</deegree:DataDirectory>
    </deegree:DataDirectoryList>
    <!-- optional; default = directory from system property "java.io.tmpdir" -->
    <!-- <deegree:LockManagerDirectory>/tmp</deegree:LockManagerDirectory> -->
  </deegree:deegreeParams>
…
```

Within the `deegreewfs:deegreeParams` element, the `DefaultOnlineResource` is defined first. It is used as the default value for omitted DCP-elements (in the OperationMetadata section) – so you don't have to adapt the URL of the service at serveral points in the document. The `deegreewfs:CacheSize` element allows to set the size of cache available for storing feature instances in memory. ***This feature is not implemented yet***. To avoid that the service can be blocked altogether by extremely time-consuming requests, a maximum time limit for request processing can be defined using the `deegreewfs:RequestTimeLimit` element. If the request processing exceeds this limit a timeout exception will be returned instead of the requested result. Internally deegree WFS will terminate all processes participating in processing the request. Very important is the list of directories defined by the `deegreewfs:DataDirectoryList` element. deegree WFS will search all directories defined here for datastore configuration files. `<deegreewfs:LockManager Directory>` sets the directory where information about currently locked features is stored; if omitted, the directory specified in the system property "java.io.tmpdir" is used.

Right behind the `deegreewfs:deegreeParams` section, sections for service identification and provider description follow.

```
…
  <!-- ====================================================================== -->
  <!-- SERVICE IDENTIFICATION SECTION                                         -->
  <!-- ====================================================================== -->
  <ows:ServiceIdentification>
    <ows:ServiceType>WFS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.1.0</ows:ServiceTypeVersion>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>None</ows:AccessConstraints>
  </ows:ServiceIdentification>
  <!-- ====================================================================== -->
  <!-- SERVICE PROVIDER SECTION                                               -->
  <!-- ====================================================================== -->
  <ows:ServiceProvider>
    <ows:ProviderName>lat/lon GmbH</ows:ProviderName>
    <ows:ProviderSite xlink:href="http://www.lat-lon.de" />
    <ows:ServiceContact>
      <ows:IndividualName>Markus Schneider</ows:IndividualName>
      <ows:PositionName>deegree WFS core developer</ows:PositionName>
```

```
      <ows:ContactInfo>
        <ows:Phone>
          <ows:Voice>+49 228 184960</ows:Voice>
          <ows:Facsimile>+49 228 1849629</ows:Facsimile>
        </ows:Phone>
        <ows:Address>
          <ows:DeliveryPoint>Aennchenstr. 19</ows:DeliveryPoint>
          <ows:City>Bonn</ows:City>
          <ows:AdministrativeArea>Northrhine-Westfalia</ows:AdministrativeArea>
          <ows:PostalCode>53177</ows:PostalCode>
          <ows:Country>Germany</ows:Country>
          <ows:ElectronicMailAddress>schneider@lat-
          lon.de</ows:ElectronicMailAddress>
        </ows:Address>
        <ows:OnlineResource xlink:href="http://localhost:8080/deegree-wfs/services"
        />
        <ows:HoursOfService>24x7</ows:HoursOfService>
        <ows:ContactInstructions>Don't call us. We'll call
        you.</ows:ContactInstructions>
      </ows:ContactInfo>
      <ows:Role>PointOfContact</ows:Role>
    </ows:ServiceContact>
  </ows:ServiceProvider>
…
```

No extensions have been made to these sections, for a detailed description of all elements please refer to the OGC WFS specification.

Afterwards, the definition of operations available through the WFS instance follows. Currently, the deegree WFS implementation supports the operations `GetCapabilities`, `DescribeFeatureType`, `GetFeature`, `Transaction`, `LockFeature` and `GetFeatureWithLock`. The support for the `Transaction` operation depends on the datastore backend (see appendix A for details).

```
…
  <!-- ====================================================================== -->
  <!-- OPERATIONS METADATA SECTION                                            -->
  <!-- ====================================================================== -->
  <ows:OperationsMetadata>
    <ows:Operation name="GetCapabilities">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="AcceptVersions">
        <ows:Value>1.1.0</ows:Value>
        <!-- Just Version 1.1.0 is supported -->
        <!-- <ows:Value>1.0.0</ows:Value>-->
      </ows:Parameter>
      <ows:Parameter name="AcceptFormats">
        <ows:Value>text/xml</ows:Value>
      </ows:Parameter>
      <ows:Parameter name="Sections">
        <ows:Value>ServiceIdentification</ows:Value>
        <ows:Value>ServiceProvider</ows:Value>
        <ows:Value>OperationsMetadata</ows:Value>
        <ows:Value>FeatureTypeList</ows:Value>
        <ows:Value>ServesGMLObjectTypeList</ows:Value>
        <ows:Value>SupportsGMLObjectTypeList</ows:Value>
        <ows:Value>Filter_Capabilities</ows:Value>
      </ows:Parameter>
    </ows:Operation>
    <ows:Operation name="DescribeFeatureType">
```

```
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="outputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
      </ows:Parameter>
    </ows:Operation>
    <ows:Operation name="GetFeature">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="resultType">
        <ows:Value>results</ows:Value>
        <ows:Value>hits</ows:Value>
      </ows:Parameter>
      <ows:Parameter name="outputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
      </ows:Parameter>
    </ows:Operation>
    <ows:Operation name="Transaction">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="inputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
      </ows:Parameter>
      <ows:Parameter name="idgen">
        <ows:Value>GenerateNew</ows:Value>
        <ows:Value>UseExisting</ows:Value>
        <!--  <ows:Value>ReplaceDuplicate</ows:Value> -->
      </ows:Parameter>
      <ows:Parameter name="releaseAction">
        <ows:Value>ALL</ows:Value>
        <!--  <ows:Value>SOME</ows:Value> -->
      </ows:Parameter>
    </ows:Operation>
    <!--  -->
    <ows:Operation name="LockFeature" />
    <ows:Operation name="GetFeatureWithLock" />
    <!-- Sets default CRS   -->
    <ows:Parameter name="srsName">
      <ows:Value>EPSG:4326</ows:Value>
    </ows:Parameter>
    <!-- Maximum number of features, which the WFS will return to the client  -->
    <ows:Constraint name="DefaultMaxFeatures">
      <ows:Value>15000</ows:Value>
    </ows:Constraint>
    <!-- time in minutes until locked features expire; not evaluated yet -->
    <ows:Constraint name="DefaultLockExpiry">
      <ows:Value>5</ows:Value>
    </ows:Constraint>
  </ows:OperationsMetadata>
...
```

Even though no additional elements are introduced by deegree here (actually the DCP-element can be left out), you should consider that an additional valid outputFormat value for the GetFeature operation besides `text/xml; subtype=gml/3.1.1` is `FEATURECOLLECTION`. This indicates that features can be accessed in a binary format representing a serialized instance of `org.deegree.model.feature.FeatureCollection`. This can be used by clients implemented using the deegree framework. E.g. the deegree WMS acts as a client on top of the deegree WFS and uses this format since it is significantly faster than text based GML formats.

The `wfs:FeatureTypeList` section lists all the features types available through a deegree WFS instance. As the WFS searches for feature type definitions automatically on startup (see above) this element can be empty:

```
…
<!-- ================================================================ -->
<!-- FEATURE TYPE LIST SECTION                                        -->
<!-- ================================================================ -->
  <!-- The wfs:FeatureTypeList can ususally be left empty, as it gets filled with
information automatically by evaluation of the defined featuretype definitons
located in <deegree:DataDirectory>s. It has to be filled, if you wish to offer
featuretypes in different SRSs. There still has to be a featuretype definition in
the <deegree:DataDirectory>. You can also define deegreewfs:inFilter and
deegreewfs:outFilter attributs to the <wfs:OutputFormats> to change GML format.
Refer documentation for this topic.  -->
<wfs:FeatureTypeList/>
…
```

In this case, all available feature types and required meta information will just be extracted from the datastore configuration files found in the defined `<DataDirectories>`. Alternatively, you can add feature types manually to this list (note that the defined feature types must also match the respective feature type definitions from the datastore configuration files). This is useful if meta information for a feature type can not be adequately extracted automatically from a datastore configuration (e.g. a feature type shall support more than its original CRS):

```
…
<wfs:FeatureType xmlns:bo="http://www.BlueOx.org/BlueOx">
    <wfs:Name>bo:WoodsType</wfs:Name>
    <wfs:Title>The Great Northern Forest</wfs:Title>
    <wfs:Abstract>Describes the arborial diversity of the Great
     Northern Forest.</wfs:Abstract>
    <ows:Keywords>
        <ows:Keyword>forest</ows:Keyword>
        <ows:Keyword>north</ows:Keyword>
        <ows:Keyword>woods</ows:Keyword>
        <ows:Keyword>arborial</ows:Keyword>
        <ows:Keyword>diversity</ows:Keyword>
    </ows:Keywords>
    <wfs:DefaultSRS>EPSG:62696405</wfs:DefaultSRS>
    <wfs:OtherSRS>EPSG:32615</wfs:OtherSRS>
    <wfs:OtherSRS>EPSG:32616</wfs:OtherSRS>
    <wfs:OtherSRS>EPSG:32617</wfs:OtherSRS>
    <wfs:OtherSRS>EPSG:32618</wfs:OtherSRS>
    <wfs:OutputFormats>
        <wfs:Format deegreewfs:inFilter="xsl/inTransform.xsl"
        deegreewfs:outFilter="xsl/outTransform.xsl"
        deegreewfs:schemaLocation="xsl/europe.xsd">text/xml;
        subtype=gml/3.1.1</wfs:Format>
    </wfs:OutputFormats>
    <ows:WGS84BoundingBox>
        <ows:LowerCorner>-180.0 -90.0</ows:LowerCorner>
        <ows:UpperCorner>180.0 90.0</ows:UpperCorner>
    </ows:WGS84BoundingBox>
</wfs:FeatureType>
…
```

The `OutputFormat` element has been extended by three additional attributes. These give you the option to apply XSL-processing to requests and responses.

- **deegreewfs:outFilter**: XSL-script for transforming the GML document that will be produced as output
- **deegreewfs:inFilter**: XSL-script for transforming requests to match the internal (default) GML schema
- **deegreewfs:schemaLocation**: location of the XML schema describing the XML-documents produced by the outFilter-script

After definition of the `FeatureTypeList` the capabilities for filters (in `GetFeature` and `Transaction` requests) follow. This includes a list of geometry types supported as operands as well as a list of spatial and logical operators supported. A detailed description is given in the OGC WFS 1.1.0 specification. In most cases you won't have to edit this section and can safely use the default section from the deegree example WFS configuration:

```
...
  <!-- ================================================================ -->
  <!-- FILTER CAPABILITIES SECTION                                      -->
  <!-- ================================================================ -->
  <ogc:Filter_Capabilities>
    <!-- Don't change <ogc:Filter_Capabilities>, its client information only -->
    <ogc:Spatial_Capabilities>
      <ogc:GeometryOperands>
        <ogc:GeometryOperand>gml:Envelope</ogc:GeometryOperand>
        <ogc:GeometryOperand>gml:Point</ogc:GeometryOperand>
        <ogc:GeometryOperand>gml:LineString</ogc:GeometryOperand>
        <ogc:GeometryOperand>gml:Polygon</ogc:GeometryOperand>
      </ogc:GeometryOperands>
      <ogc:SpatialOperators>
        <ogc:SpatialOperator name="BBOX" />
        <ogc:SpatialOperator name="Equals" />
        <ogc:SpatialOperator name="Disjoint" />
        <ogc:SpatialOperator name="Intersects" />
        <ogc:SpatialOperator name="Touches" />
        <ogc:SpatialOperator name="Crosses" />
        <ogc:SpatialOperator name="Within" />
        <ogc:SpatialOperator name="Contains" />
        <ogc:SpatialOperator name="Overlaps" />
        <ogc:SpatialOperator name="Beyond" />
      </ogc:SpatialOperators>
    </ogc:Spatial_Capabilities>
    <ogc:Scalar_Capabilities>
      <ogc:LogicalOperators />
      <ogc:ComparisonOperators>
        <ogc:ComparisonOperator>LessThan</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>GreaterThan</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>LessThanEqualTo</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>GreaterThanEqualTo</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>EqualTo</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>Like</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>Between</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>NullCheck</ogc:ComparisonOperator>
      </ogc:ComparisonOperators>
      <ogc:ArithmeticOperators>
        <ogc:SimpleArithmetic />
      </ogc:ArithmeticOperators>
    </ogc:Scalar_Capabilities>
    <ogc:Id_Capabilities>
```

```
      <ogc:EID />
      <ogc:FID />
    </ogc:Id_Capabilities>
  </ogc:Filter_Capabilities>
</wfs:WFS_Capabilities>
```

4.3 Principles of deegree WFS datastore configuration

Data sources must be assigned to feature types in order to be servable by the WFS. A datastore is configured by specifying a GML application schema that defines the structure of the served feature types plus access and mapping information to the physical data source.

On startup, the deegree WFS scans certain directories for `.xsd` files. The feature types defined in these “annotated GML application schemas” are backed by the respective data source and served by the WFS.

The most complex and sophisticated part of configuring deegree WFS is to create datastore configurations. This is the process of mapping feature types (and their properties) to the entities of a physical data source (e.g. the tables of an SQL database). These basic concepts apply:

- The configuration of a data store has to be a valid GML3 application schema document. The mapping information is contained in annotation elements.
- There is a 1:1 relation between feature types and the entities of a datasource (table), but it is possible to map properties of a feature type to a related table. This allows, for example, properties that have a cardinality > 1.
- Feature types can be complex, i.e. one or more properties can be defined to contain features themselves. This allows complex hierarchies of feature types that may even be recursive (a feature type can contain features of the same type as sub features).

4.4 Simple datastore that accesses a single database table

In this example, we are going to map the `COUNTRY` table to a GML application schema containing the `Country` feature type. The table name should be UPPERCASE and the feature type names should be in UpperCamelCase. The properties and property types should be in lowerCamelCase (this is a proposed convention, but it is not mandatory).

To set up philosopher database please follow the next steps.

1. To install the philosopher example, please create a spatially enabled and UTF8 encoded POSTGIS or Oracle database called 'deegreetest' and create a

user 'deegreetest' with a password 'deegreetest'. Please refer the according documentations of the backends to do so; e.g.
http://www.postgis.org/documentation/
(a sample script under wfs_home/WEB-INF/conf/tools/
01_create_spatial_database.sh/.bat can give you some advice)

2. Afterwards run the appropriate scripts you find under wfs_home/WEB-INF/conf/scripts/philosopher/ to create the philosopher example tables. Postgis example: `psql -d deegreetest -U deegreetest -f create_philosopher.sql`

3. Copy the appropriate featuretype definition from wfs_home/WEB-INF/conf/wfs/featuretypes/examplefeaturetypes/philosopher/ into wfs_home/WEB-INF/conf/wfs/featuretypes. In case PostgreSQL/Postgis does not run on the same machine as the wfs (localhost) and port 5432 adapt the .xsd to the location and setting of the database. Afterwards restart tomcat or the service via the tomcat manager.

4. Fill the database by sending a WFS-transaction request. You find the request under: http://localhost:8080/deegree-wfs/client/client.html
Go to the Example Philosopher dropdown menu and select in the Request dropdown menue Transaction-> complex.xml. Submit the request by pressing 'Send'. If a successful wfs:Transaction response appears in the bottom part of the browser window, everything went fine and the WFS is ready to use.

The `Country` feature type has the following properties:

- (1) id (primary key)
- (1) name
- (1) geom

Example of a `Country` instance:

```
<app:Country gml:id="COUNTRY_1">
  <app:id>1</app:id>
  <app:name>France</app:name>
  <app:geom> <!-- GML geometry skipped for brevity --> </app:geom>
</app:Country>
```

The application schema we are aiming for consists of just one feature type, `Country` with 3 properties (2 simple + 1 spatial):

- (1) id:integer
- (1) name:string
- (1) geom:geometry

The `Country` feature type is defined by this XSD-fragment:

```
<xsd:element name="Country" substitutionGroup="gml:_Feature">
 <xsd:annotation>
   <xsd:appinfo>
     <deegreewfs:table>COUNTRY</deegreewfs:table>
     <deegreewfs:gmlId prefix="COUNTRY_">
       <deegreewfs:MappingField field="ID" type="INTEGER"/>
       <deegreewfs:IdGenerator type="DB_SEQ">
         <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
       </deegreewfs:IdGenerator>
     </deegreewfs:gmlId>
   </xsd:appinfo>
 </xsd:annotation>
 <xsd:complexType>
   <xsd:complexContent>
     <xsd:extension base="gml:AbstractFeatureType">
       <xsd:sequence>
         <xsd:element name="id" type="xsd:integer">
           <xsd:annotation>
             <xsd:appinfo>
               <deegreewfs:Content>
                 <deegreewfs:MappingField field="ID" type="INTEGER"/>
               </deegreewfs:Content>
             </xsd:appinfo>
           </xsd:annotation>
         </xsd:element>
         <xsd:element name="name" type="xsd:string">
           <xsd:annotation>
             <xsd:appinfo>
               <deegreewfs:Content>
                 <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
               </deegreewfs:Content>
             </xsd:appinfo>
           </xsd:annotation>
         </xsd:element>
         <xsd:element name="geom" type="gml:GeometryPropertyType">
           <xsd:annotation>
             <xsd:appinfo>
               <deegreewfs:Content>
                 <deegreewfs:MappingField field="GEOM" type="GEOMETRY"/>
               </deegreewfs:Content>
             </xsd:appinfo>
           </xsd:annotation>
         </xsd:element>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```

Please note that the element declaration for each feature type must be substitutable by `gml:_Feature`, using `substitutionGroup="gml:_Feature"`. This is necessary for the feature type definition to be valid. Unless a feature type is valid, it will not be recognized by deegree WFS (for detailed description on defining a valid GML feature type / application schema please see GML 3.1.1 specification or have a look at: Ron Lake et al. (2004): Geography Mark-Up Language, John Wiley & Sons Ltd.

The content model for the feature type `Country` is defined using the 'inline' style, i.e. the complex type is defined inside the element. Alternatively, the reference to a complex type definition can be made by using the `type` attribute of the element declaration.

The complex type declaration that defines the content model of the feature type must have an annotation section that binds a table to it and defines the key columns to be used:

```
<xsd:element name="Country" substitutionGroup="gml:_Feature">
 <xsd:annotation>
   <xsd:appinfo>
     <deegreewfs:table>COUNTRY</deegreewfs:table>
     <deegreewfs:gmlId prefix="COUNTRY_">
       <deegreewfs:MappingField field="ID" type="INTEGER"/>
       <deegreewfs:IdGenerator type="DB_SEQ">
         <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
       </deegreewfs:IdGenerator>
     </deegreewfs:gmlId>
        <deegreewfs:visible>true</deegreewfs:visible>
         <deegreewfs:transaction update="true" delete="true" insert="true"/>
   </xsd:appinfo>
 </xsd:annotation>
 <xsd:complexType>
   <xsd:complexContent>
...
```

The `deegreewfs:table` element declares the associated table, and the key columns are defined inside the `deegreewfs:gmlId` element. The keys are used to build the GML ids (also known as feature ids) of the generated features. By definition a GML id must not start with a number. The `prefix`-attribute can be used to prepend a user defined string prefix to the key fields. The `deegreewfs:MappingField` elements define the table columns to use to retrieve the id values, in this case only one column is needed. This is not a restriction, a user defined number of columns (and therefore `deegreewfs:MappingField` elements) may be retrieved. `<deegreewfs:IdGenerator type="...">` sets which ID generation method is to be used. The method types are:

```
<deegreewfs:IdGenerator type="DB_MAX">
  <deegreewfs:param name="table">PHILOSOPHER</deegreewfs:param>
  <deegreewfs:param name="column">ID</deegreewfs:param>
</deegreewfs:IdGenerator>
```

or

```
<deegreewfs:IdGenerator type="DB_SEQ">
  <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
</deegreewfs:IdGenerator>
```

or

```
<deegreewfs:IdGenerator type="UUID">
  <deegreewfs:param name="macAddress">0x000F0934545A</deegreewfs:param>
</deegreewfs:IdGenerator>
```

The 0x indicates hexadecimal writing. It's followed by the MAC adress disregarding ':'.

`<deegreewfs:transaction ... />` specifies, which WFS transaction requests are enabled.

If mapping a geometry field of a PostGIS or Oracle database one optionally can define an additional attribute (srs) for the mappingField to define the internal srs-code used by the data source (which is not identical to the defaultSRS of the featuretype). In the example above no further information about the geometry field is given so deegree assumes no SRS has been defined within the connected datasource. This is identical to setting srs="-1". If a SRS code has been defined for a geometry field within the datasource the srs-Attribute must be set accordingly:

```
<xsd:element name="geom" type="gml:GeometryPropertyType">
 <xsd:annotation>
   <xsd:appinfo>
     <deegreewfs:Content>
       <deegreewfs:MappingField field="GEOM" type="GEOMETRY" srs="4326"/>
     </deegreewfs:Content>
   </xsd:appinfo>
 </xsd:annotation>
</xsd:element>
```

This enables the WFS to map the defaultSRS (e.g. EPSG:4326) to the internal srs code of the datasource (in this case 4326).

The next part of the feature type declaration looks like this:

```
<xsd:complexContent>
 <xsd:extension base="gml:AbstractFeatureType">
   <xsd:sequence>
   ...
   </xsd:sequence>
 </xsd:extension>
</xsd:complexContent>
```

The `xsd:complexContent` element is necessary to mark the beginning of the content model for a feature type, the declaration *must* use the „XSD extension" mechanism. The type that is extended must be `gml:AbstractFeatureType` or a type (which must be defined in the same schema document) that derives `gml:AbstractFeatureType`.

The remaining part defines the properties of the feature type and how their values are mapped to the table columns:

```
 <xsd:element name="id" type="xsd:integer">
   <xsd:annotation>
     <xsd:appinfo>
       <deegreewfs:Content>
         <deegreewfs:MappingField field="ID" type="INTEGER"/>
       </deegreewfs:Content>
     </xsd:appinfo>
   </xsd:annotation>
 </xsd:element>
 <xsd:element name="name" type="xsd:string">
```

```
    <xsd:annotation>
      <xsd:appinfo>
        <deegreewfs:Content>
          <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
        </deegreewfs:Content>
      </xsd:appinfo>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name="geom" type="gml:GeometryPropertyType">
    <xsd:annotation>
      <xsd:appinfo>
        <deegreewfs:Content>
          <deegreewfs:MappingField field="GEOM" type="GEOMETRY"/>
        </deegreewfs:Content>
      </xsd:appinfo>
    </xsd:annotation>
  </xsd:element>
```

The `id` property is the primary key in this context and is used to uniquely identify the features. The `name` property contains the name of the country and the `geom` property the geometry. To set the geometry property it is mandatory to set the `type` to `gml:GeometryPropertyType` (see above). The annotation element is used to provide the WFS with the mapping information for each element (=property) definition. The `deegreewfs:Content` element encapsulates this information. Because each of the three properties is simple or spatial and is stored in the base table of the feature type ('COUNTRY'), a simple `deegreewfs:MappingField` element is sufficient to tell the WFS in which column the value for a property is found. The name of the GML property does not have to be the same as the name of the table field that is mapped to it (see name – countryName in the example).

4.5 Complex datastore that accesses several database tables

In this example, we're going to map 5 tables to a GML application schema with 3 feature types. The relational schema uses foreign keys and join tables. All table names are in UPPERCASE, feature and feature type names are in UpperCamelCase, properties and property types are in lowerCamelCase.

You have already set up the required database in 4.4. So just redo the step 4 but use the Transaction -> complex.xml insert request to fill the database with the needed features. Existing features will be updated.

The application realm is historical philosophers. Our relational model provides a (dead) philosopher with the following properties / relations:

- A `PHILOSOPHER` has:
  - (1) id (primary key)
  - (1) name
  - (1) sex

- (0...n) subjects (modelled using SUBJECT table)
- (1) date of birth
- (1) place of birth (foreign key to PLACE table)
- (1) date of death
- (1) place of death (foreign key to PLACE table)
- (0...n) friends (modelled using IS_FRIEND_OF table)

- A `PLACE` (town, city, ..) has:
  - (1) id (primary key)
  - (1) name
  - (1) country (foreign key to COUNTRY)
- A `COUNTRY` has:
  - (1) id (primary key)
  - (1) name
  - (1) geometry

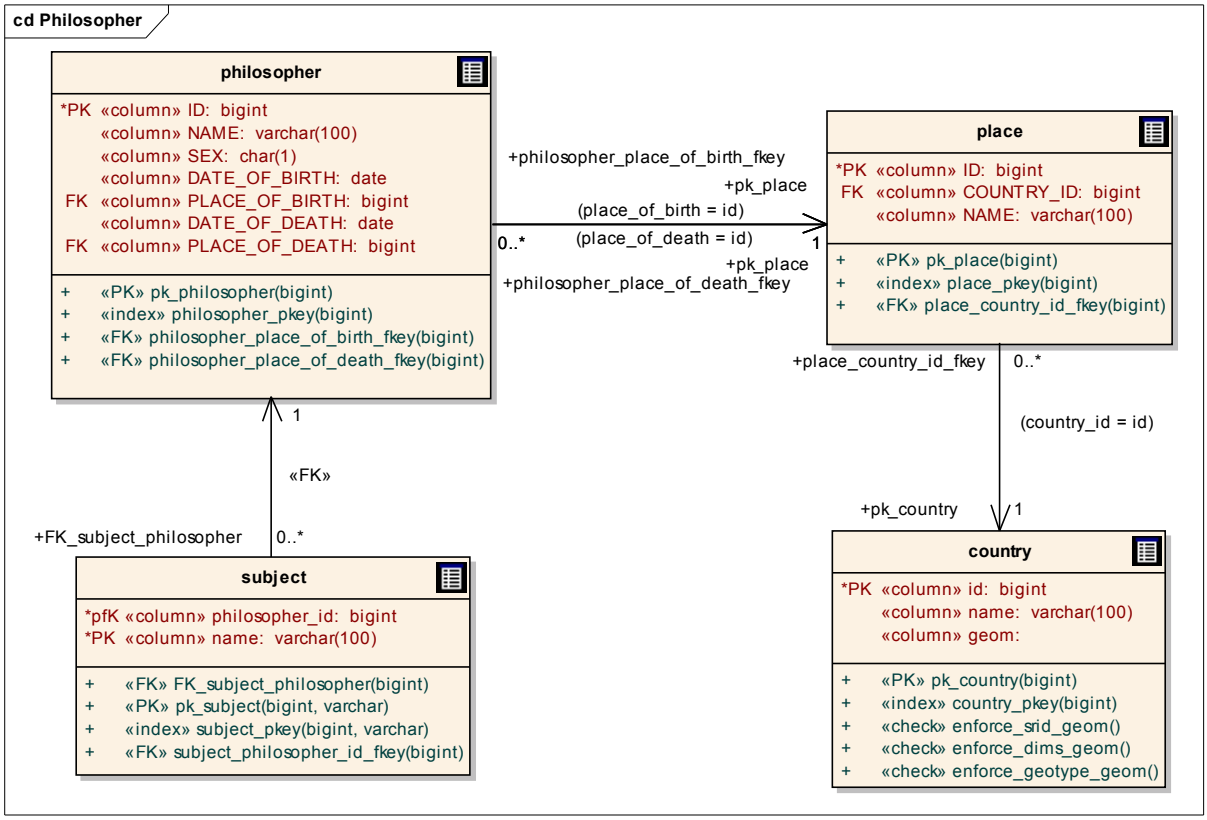

Figure 3: Relational schema for philosopher example (misses IS_FRIEND_OF table)

Before we define the structure and mapping of our feature type formally using XML Schema, we present how a `Philosopher` instance looks like:

```
<app:Philosopher gml:id="PHILOSOPHER_6">
  <app:id>6</app:id>
  <app:name>Jean-Paul Sartre</app:name>
  <app:sex>m</app:sex>
  <app:subject>existentialism</app:subject>
  <app:subject>freedom</app:subject>
  <app:dateOfBirth>1905-6-2</app:dateOfBirth>
  <app:placeOfBirth>
    <app:Place gml:id="PLACE_6">
      <app:id>6</app:id>
      <app:name>Paris</app:name>
      <app:country>
        <app:Country gml:id="COUNTRY_1">
          <app:id>1</app:id>
          <app:name>France</app:name>
          <app:geom> <!-- GML geometry skipped--> </app:geom>
        </app:Country>
      </app:country>
    </app:Place>
  </app:placeOfBirth>
  <app:dateOfDeath>1980-4-15T00:00:00:</app:dateOfDeath>
  <app:placeOfDeath>
    <app:Place gml:id="PLACE_6">
      <app:id>6</app:id>
      <app:name>Paris</app:name>
      <app:country>
```

```
        <app:Country gml:id="COUNTRY_1">
          <app:id>1</app:id>
          <app:name>France</app:name>
          <app:geom> <!-- GML geometry skipped --> </app:geom>
        </app:Country>
      </app:country>
    </app:Place>
  </app:placeOfDeath>
  <app:friend>
    <app:Philosopher gml:id="PHILOSOPHER_7">
    <!-- friend Philosopher instance skipped -->
    </app:Philosopher>
  </app:placeOfDeath>
</app:Philosopher>
```

The GML application schema that we're aiming at consists of three feature types:

- `Philosopher`, has 9 properties (6 simple + 3 complex)
  - (1) id:string
  - (1) name:string
  - (1) sex:string
  - (0..n) subject:string
  - (1) dateOfBirth:date
  - (1) placeOfBirth, containing a feature of type Place
  - (1) dateOfDeath:date
  - (1) placeOfDeath, containing a feature of type Place
  - (0..n) friend, each containing a feature of type Philosopher
- `Place`, has 2 properties (1 simple + 1 complex)
  - (1) name:string
  - (1) country, containing a feature of type Country
- `Country`, has 2 properties (1 simple + 1 spatial)
  - (1) name:string
  - (1) geom:geometry

We'll start bottom-up, because the last feature type is much simpler than our root feature type `Philosopher`.

The `Country` feature type is defined by this XSD-fragment:

```
<element name="Country" substitutionGroup="gml:_Feature">
  <!-- anonymous types are allowed as well -->
  <complexType>
    <annotation>
      <appinfo>
        <wfs:table>COUNTRY</wfs:table>
        <wfs:gmlId prefix="COUNTRY_">
          <wfs:MappingField field="ID" type="INTEGER"/>
        </wfs:gmlId>
      </appinfo>
    </annotation>
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element name="name" type="xsd:string">
            <annotation>
              <appinfo>
                <wfs:Content>
                  <wfs:MappingField field="NAME" type="VARCHAR"/>
                </wfs:Content>
              </appinfo>
            </annotation>
          </element>
          <element name="geom" type="gml:GeometryPropertyType">
            <annotation>
              <appinfo>
                <wfs:Content>
                  <wfs:MappingField field="geom" type="GEOMETRY"/>
                </wfs:Content>
              </appinfo>
            </annotation>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

Please note that the element declaration for each feature type must be substitutable by `gml:_Feature`, for example using `substitutionGroup="gml:_Feature"`. Otherwise, it is not a GML feature type definition and therefore won't be recognized by the WFS.

The content model for the feature type `Country` is defined using the 'inline' style, i.e. the complex type is defined inside the element declaration's element. Alternatively, you can reference a complex type definition by using the 'type' attribute of the element declaration.

The complex type declaration which defines the content model of the feature type must have an annotation section that binds a table to it and defines the key columns to be used:

```
<element name="Country" substitutionGroup="gml:_Feature">
  <!-- anonymous types are allowed as well -->
  <complexType>
    <annotation>
      <appinfo>
        <wfs:table>COUNTRY</wfs:table>
        <wfs:gmlId prefix="COUNTRY_">
          <wfs:MappingField field="ID" type="INTEGER"/>
```

```
        </wfs:gmlId>
      </appinfo>
    </annotation>
      <complexContent>
...
```

The `deegreewfs:table` element declares the associated table, and the key columns are defined inside the `deegreewfs:gmlId` element. The keys are used to build the GML ids (formerly known as feature ids) of the generated features. By definition a GML id may not start with a number. Use the `prefix`-attribute to prepend a string to the key fields as you like. The `deegreewfs:MappingField` elements define the table columns to use for retrieving the id values, in this case only one column is needed, but you could use as many columns (and therefore `deegreewfs:MappingField` elements) as you want.

The next part of the feature type declaration looks like this:

```
<complexContent>
  <extension base="gml:AbstractFeatureType">
    <sequence>
    ...
    </sequence>
  </extension>
</complexContent>
```

The `xsd:complexContent` element is necessary to define the content model for a feature type, the declaration must use the „XSD extension“ mechanism. The type that is extended must be `gml:AbstractFeatureType` or a type (defined in the same schema document) that derives `gml:AbstractFeatureType`.

The remaining part defines the properties of the feature and how their values are fetched from the database:

```
<element name="name" type="xsd:string">
  <annotation>
    <appinfo>
      <wfs:Content>
        <wfs:MappingField field="NAME" type="VARCHAR"/>
      </wfs:Content>
    </appinfo>
  </annotation>
</element>
<element name="geom" type="gml:GeometryPropertyType">
  <annotation>
    <appinfo>
      <wfs:Content>
        <wfs:MappingField field="GEOM" type="GEOMETRY"/>
      </wfs:Content>
    </appinfo>
  </annotation>
</element>
```

The definition of the two properties `name` and `geom` should be quite self-explanatory (at least if you are used to XML Schema). If you want to define a geometry property, you have to set `type` to `gml:GeometryPropertyType`, as one can see in

the last element declaration. Again, the annotation element is used to provide the WFS with the mapping information for each element (=property) definition. The `wfs:Content` element encapsulates this information. Because each of the three properties is simple or spatial and is stored in the base table of the feature type (`COUNTRY`), a simple `wfs:MappingField` element is sufficient to tell the WFS in which column the value for a property is found.

Next, we're going to look at the definition of the feature type `Place` which – in contrast to the `Country` feature type contains a complex valued property (namely `country`):

```
<!-- ============================================================================ -->
<element name="Place" type="app:PlaceType" substitutionGroup="gml:_Feature"/>
<!-- ============================================================================ -->
<complexType name="PlaceType">
  <annotation>
    <appinfo>
      <wfs:table>PLACE</wfs:table>
      <wfs:gmlId prefix="PLACE_">
        <wfs:MappingField field="ID" type="INTEGER"/>
      </wfs:gmlId>
    </appinfo>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="id" type="xsd:string" nillable="false">
          <annotation>
            <appinfo>
              <wfs:Content>
                <wfs:MappingField field="ID" type="INTEGER"/>
              </wfs:Content>
            </appinfo>
          </annotation>
        </element>
        <element name="name" type="xsd:string">
          <annotation>
            <appinfo>
              <wfs:Content>
                <wfs:MappingField field="NAME" type="VARCHAR"/>
              </wfs:Content>
            </appinfo>
          </annotation>
        </element>
        <element name="country" type="gml:FeaturePropertyType">
          <annotation>
            <appinfo>
              <wfs:Content type="app:Country">
                <wfs:Relation>
                  <wfs:From>
                    <wfs:MappingField field="COUNTRY_ID" type="INTEGER"/>
                  </wfs:From>
                  <wfs:To>
                    <wfs:MappingField field="ID" type="INTEGER" table="COUNTRY"/>
                  </wfs:To>
                </wfs:Relation>
              </wfs:Content>
            </appinfo>
          </annotation>
        </element>
      </sequence>
```

```
    </extension>
  </complexContent>
</complexType>
```

The feature type definition starts just like the last one, but this time, it does not use the 'inline' variant to define the content model, but defines a named complex type 'PlaceType':

```
<!-- ====================================================================== -->
<element name="Place" type="app:PlaceType" substitutionGroup="gml:_Feature"/>
<!-- ====================================================================== -->
<complexType name="PlaceType">
  <annotation>
    <appinfo>
      <wfs:table>PLACE</wfs:table>
      <wfs:gmlId prefix="PLACE_">
        <wfs:MappingField field="ID" type="INTEGER"/>
      </wfs:gmlId>
    </appinfo>
  </annotation>
  <complexContent>
    <extension base="gml:AbstractFeatureType">
...
```

The two ways are equivalent, but if you use a named complexType here, another complexType definition can extend this complexType. The rest of the feature type definition's epilogue is identical to the former examples; the annotation contains the binding of a table to the feature type model and declares how to build GML-Ids from one table row.
Now for the model of the feature type and the promised complex valued property:

```
<complexContent>
  <extension base="gml:AbstractFeatureType">
    <sequence>
      <element name="name" type="xsd:string">
       ...
      </element>
      <element name="country" type="gml:FeaturePropertyType">
        <annotation>
          <appinfo>
            <wfs:Content type="app:Country">
              <wfs:Relation>
                <wfs:From>
                  <wfs:MappingField field="COUNTRY_ID" type="INTEGER"/>
                </wfs:From>
                <wfs:To>
                  <wfs:MappingField field="ID" type="INTEGER" table="COUNTRY"/>
                </wfs:To>
              </wfs:Relation>
            </wfs:Content>
          </appinfo>
        </annotation>
      </element>
    </sequence>
  </extension>
</complexContent>
```

The definition of the `name` property is nothing new. Let's focus on the definition of element (=property) `country`. As this element does not contain primitive or spatial

values, but `Country`-elements, its content must be declared (according to GML practices) as `type="gml:FeaturePropertyType"`. This is already a correct GML feature property definition, but the WFS needs to know where the data for this property comes from. Again, the `xsd:annotation` element comes to the rescue. Because we need to tell the WFS which features are to be put here, the `deegreewfs:Content` element has the `type` attribute set to `app:Country`. If you wonder about the `app:` prefix: this is because our whole example schema definition operates on the target namespace `'app=http://www.deegree.org/app'` So now, the WFS knows that the feature property `country` should contain the features from type definition `Country`. But it still needs information on how `PLACE` (table from `Place` feature type annotation) relates to `COUNTRY` (table from `Country`). This is done by one or more `wfs:Relation elements`. The `deegreewfs:From` element contains the join fields in the origin table (`PLACE`), the `deegreewfs:To` element lists the corresponding fields in the target table (`COUNTRY`)[3].
Last, but not least, we're ready to face some new constructs in the definition of the root feature type 'Philosopher':

```
    <!-- simple (string) valued property 'subject', located in related table -->
<xsd:element name="subject" type="xsd:string" minOccurs="0" maxOccurs="unbounded">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
        <deegreewfs:Relation>
          <deegreewfs:From>
            <deegreewfs:MappingField field="ID" type="INTEGER"/>
          </deegreewfs:From>
          <deegreewfs:To fk="true">
            <deegreewfs:MappingField field="PHILOSOPHER_ID" type="INTEGER"
            table="SUBJECT"/>
          </deegreewfs:To>
        </deegreewfs:Relation>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

The property `subject` may occur none or several times. Therefore `minOccurs` is set to `0` and `maxOccurs` is `unbounded` in the element declaration. The relational schema holds the subject information in a related table (`SUBJECT`) which has a foreign key to the `PHILOSOPHER` table. The `deegreewfs:Relation` element reflects this as it sets `fk` to `true`. As stated above, it tells *how to get from one table (`PHILOSOPHER`) to another table (`SUBJECT`)*. And because `PHILOSOPHER` -> `SUBJECT` has 0...* multiplicity, one row in the `PHILOSOPHER` table has an arbitrary number of associated `SUBJECT` rows. In the feature-property perspective, this maps to: one feature instance of type '`Philosopher`' having an arbitrary number of '`subject`' properties. Note that the `deegreewfs:MappingField` element therefore does not

[3]In SQL this clause would look like: 'WHERE PLACE.COUNTRY_ID = COUNTRY.ID'

refer to the table of the feature (PHILOSOPHER), but to the target table of the last deegreewfs:Relation element in the property annotation.
Another new aspect of the Philosopher feature type is that it has two relations to the Place featureType':

```
<element name="placeOfBirth" type="gml:FeaturePropertyType">
  <annotation>
    <appinfo>
      <wfs:Content type="app:Place">
        <wfs:Relation>
          <wfs:From>
            <wfs:MappingField field="PLACE_OF_BIRTH" type="INTEGER"/>
          </wfs:From>
          <wfs:To>
            <wfs:MappingField field="ID" type="INTEGER" table="PLACE"/>
          </wfs:To>
        </wfs:Relation>
      </wfs:Content>
    </appinfo>
  </annotation>
</element>
...
<element name="placeOfDeath" type="gml:FeaturePropertyType">
  <annotation>
    <appinfo>
      <wfs:Content type="app:Place">
        <wfs:Relation>
          <wfs:From>
            <wfs:MappingField field="PLACE_OF_DEATH" type="INTEGER"/>
          </wfs:From>
          <wfs:To>
            <wfs:MappingField field="ID" type="INTEGER" table="PLACE"/>
          </wfs:To>
        </wfs:Relation>
      </wfs:Content>
    </appinfo>
  </annotation>
</element>
```

Both properties are feature properties (they even contain features of the same type, Place). The only difference are their names and the foreign key field: for placeOfDeath, the column PLACE_OF_DEATH is used to find the corresponding Place features, for placeOfBirth, this is PLACE_OF_BIRTH.

Finally, the friend property shows how to use join tables:

```
<xsd:element name="friend" type="gml:FeaturePropertyType" minOccurs="0"
maxOccurs="unbounded">
 <xsd:annotation>
   <xsd:appinfo>
     <deegreewfs:Content type="app:Philosopher">
       <deegreewfs:Relation>
         <deegreewfs:From>
           <deegreewfs:MappingField field="ID" type="INTEGER"/>
         </deegreewfs:From>
         <deegreewfs:To fk="true">
           <deegreewfs:MappingField field="PHILOSOPHER1_ID" type="INTEGER"
table="IS_FRIEND_OF"/>
         </deegreewfs:To>
```

```
        </deegreewfs:Relation>
        <deegreewfs:Relation>
          <deegreewfs:From fk="true">
            <deegreewfs:MappingField field="PHILOSOPHER2_ID" type="INTEGER"/>
          </deegreewfs:From>
          <deegreewfs:To>
            <deegreewfs:MappingField field="ID" type="INTEGER" table="PHILOSOPHER"/>
          </deegreewfs:To>
        </deegreewfs:Relation>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

It is a feature property that contains `Philosopher` instances and that may occur an arbitrary number of times. The join table `IS_FRIEND_OF` is used to model this relation. Again, please read this like this: in order to find the corresponding friend Philosopher instances for a given Philosopher A, use the value of A's `ID` field to determine the corresponding rows in table `IS_FRIEND_OF` (all rows where `A.ID` equals `IS_FRIEND_OF.PHILOSOPHER1_ID`). For these rows, determine all rows in table `PHILOSOPHER` where `IS_FRIEND_OF.PHILOSOPHER1_ID` equals `PHILOSOPHER.ID`. You might wonder, how cyclic features are represented, e.g. Philosopher A is friend with Philosopher B and Philosopher B is friend with Philosopher A. In this case, the friend property of Philosopher B will not contain the Philosopher A feature – this would lead to an endless recursion – but will specify the feature id of Philosopher A in `xlink:href` attribute of the friend property.

Bringing all together the `Philosopher` type definition looks like this:

```
<!-- ============================================================================ -->
 <xsd:element name="Philosopher" type="app:PhilosopherType"
substitutionGroup="gml:_Feature">
   <!-- annotation of each element (=feature type definition) contains associated
table name and gml:id definition -->
   <xsd:annotation>
     <xsd:appinfo>
       <deegreewfs:table>PHILOSOPHER</deegreewfs:table>
       <deegreewfs:gmlId prefix="PHILOSOPHER_">
         <deegreewfs:MappingField field="ID" type="INTEGER"/>
         <deegreewfs:IdGenerator type="DB_MAX">
           <deegreewfs:param name="table">PHILOSOPHER</deegreewfs:param>
           <deegreewfs:param name="column">ID</deegreewfs:param>
         </deegreewfs:IdGenerator>
       </deegreewfs:gmlId>
       <deegreewfs:transaction update="true" delete="true" insert="true"/>
     </xsd:appinfo>
   </xsd:annotation>
 </xsd:element>
 <!-- ============================================================================ -->
 <xsd:complexType name="PhilosopherType">
   <xsd:complexContent>
     <xsd:extension base="gml:AbstractFeatureType">
       <xsd:sequence>
         <!-- simple (integer) valued property 'id' (feature id without prefix) -->
         <xsd:element name="id" type="xsd:integer">
           <xsd:annotation>
             <xsd:appinfo>
```

```
            <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
            <deegreewfs:Content>
              <deegreewfs:MappingField field="ID" type="INTEGER"/>
            </deegreewfs:Content>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <!-- simple (string) valued property 'name' -->
      <xsd:element name="name" type="xsd:string">
        <xsd:annotation>
          <xsd:appinfo>
            <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
            <deegreewfs:Content>
              <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
            </deegreewfs:Content>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <!-- simple (string) valued property 'sex' -->
      <xsd:element name="sex" type="xsd:string">
        <xsd:annotation>
          <xsd:appinfo>
            <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
            <deegreewfs:Content>
              <deegreewfs:MappingField field="SEX" type="CHAR"/>
            </deegreewfs:Content>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <!-- simple (string) valued property 'subject', located in related table
-->
      <xsd:element name="subject" type="xsd:string" minOccurs="0"
maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:appinfo>
            <deegreewfs:Content>
              <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
              <deegreewfs:Relation>
                <deegreewfs:From>
                  <deegreewfs:MappingField field="ID" type="INTEGER"/>
                </deegreewfs:From>
                <deegreewfs:To fk="true">
                  <deegreewfs:MappingField field="PHILOSOPHER_ID" type="INTEGER"
table="SUBJECT"/>
                </deegreewfs:To>
              </deegreewfs:Relation>
            </deegreewfs:Content>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <!-- simple (date) valued property 'dateOfBirth' -->
      <xsd:element name="dateOfBirth" type="xsd:date">
        <xsd:annotation>
          <xsd:appinfo>
            <deegreewfs:Content>
              <deegreewfs:MappingField field="DATE_OF_BIRTH" type="DATE"/>
            </deegreewfs:Content>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <!-- complex valued property 'placeOfBirth' -->
      <xsd:element name="placeOfBirth" type="gml:FeaturePropertyType">
        <xsd:annotation>
          <xsd:appinfo>
            <deegreewfs:Content type="app:Place">
```

```
                <deegreewfs:Relation>
                  <deegreewfs:From fk="true">
                    <deegreewfs:MappingField field="PLACE_OF_BIRTH"
type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- simple (date) valued property 'dateOfDeath' -->
        <xsd:element name="dateOfDeath" type="xsd:date">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="DATE_OF_DEATH" type="DATE"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- complex valued property 'placeOfDeath' -->
        <xsd:element name="placeOfDeath" type="gml:FeaturePropertyType">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content type="app:Place">
                <deegreewfs:Relation>
                  <deegreewfs:From fk="true">
                    <deegreewfs:MappingField field="PLACE_OF_DEATH"
type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- complex valued property 'friend' (recursive) -->
        <xsd:element name="friend" type="gml:FeaturePropertyType" minOccurs="0"
maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content type="app:Philosopher">
                <deegreewfs:Relation>
                  <deegreewfs:From>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To fk="true">
                    <deegreewfs:MappingField field="PHILOSOPHER1_ID"
type="INTEGER" table="IS_FRIEND_OF"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
                <deegreewfs:Relation>
                  <deegreewfs:From fk="true">
                    <deegreewfs:MappingField field="PHILOSOPHER2_ID"
type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To>
                    <deegreewfs:MappingField field="ID" type="INTEGER"
table="PHILOSOPHER"/>
                  </deegreewfs:To>
```

```
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Notice that definition of the `gml:Id` is made within an annotation section of the `Philosopher` element and *not* within the type definition. The reason for this is that maybe then one `xsd:element` definition can use the same `xsd:complexType` definition but use different field(s) for creating a unique Id.

5 Supported datasources (backend types)

In chapter and we explained how to map one or more tables of a physical datasource to the elements of a logical output schema. The way these mappings are encoded is not dependent on the type of datasource (except that shapefile based datasources are not able to support relations between tables). Specific for each type of datasource is the connection part of the configuration where the datastore administrator defines the required parameters for establishing access to the underlying physical datasource (usually a relational database).

5.1 Common parameters

The connection information is defined in the first `xsd:annotation/xsd:appinfo` section embedded in the root element of the schema document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"..."/>
...
<!-- configuration of the persistence backend to be used -->
<xsd:annotation>
  <xsd:appinfo>
    <deegreewfs:Prefix>app</deegreewfs:Prefix>
    <deegreewfs:Backend>POSTGIS</deegreewfs:Backend>
    <deegreewfs:DefaultSRS>EPSG:4326</deegreewfs:DefaultSRS>
    <deegreewfs:SuppressXLinkOutput>false</deegreewfs:SuppressXLinkOutput>
    <dgjdbc:JDBCConnection>
      <dgjdbc:Driver>org.postgresql.Driver</dgjdbc:Driver>
      <dgjdbc:Url>jdbc:postgresql://localhost:5432/deegreetest</dgjdbc:Url>
      <dgjdbc:User>deegreetest</dgjdbc:User>
      <dgjdbc:Password>deegreetest</dgjdbc:Password>
      <dgjdbc:SecurityConstraints/>
      <dgjdbc:Encoding>iso-8859-1</dgjdbc:Encoding>
    </dgjdbc:JDBCConnection>
  </xsd:appinfo>
</xsd:annotation>
...
```

The example shows a connection configuration for a PostGIS database. Common to all kinds of datastores are the first four elements:

- `<deegreewfs:Prefix>` sets the namespace prefix that is used to qualify feature and property names in responses to `GetFeature` requests
- `<deegreewfs:Backend>` sets the backend type; valid type codes are `POSTGIS`, `ORACLE`, `GENERICSQL`, `SHAPE` and `SDE`. Further backend types will be added in the future.
- `<deegreewfs:DefaultSRS>` sets the default EPSG code of the coordinate reference system for all geometry properties of the application schema; it can be overwritten in a geometry property definition

- `<deegreewfs:SuppressXLinkOutput>` can be used to prohibit the use of xlinked feature properties in the response to GetFeature-requests. Generally, a feature (more precise: a certain gml:Id) may only be present once in the output (otherwise the document is formally invalid). However, if it occurs a second time in the result feature collection (which is very well possible), it must be referenced using an `xlink`-attribute. If a client cannot cope with `xlinks`, one can set `deegreewfs:SuppressXLinkOutput` to 'true'. Note that this is generally discouraged, as it may force the WFS to produce invalid GML sometimes. In some cases (if a feature contains itself as a subfeature on some level), it is not possible to generate the output at all (in this case an error messages will be generated). If in doubt, simply leave out `deegreewfs:SuppressXLinkOutput`, which will enable the use of xlinks in the output (identical to setting `SuppressXLinkOutput` to `'false'`).

All other elements are specific to concrete backend types.

5.2 PostGIS-based datastores

SQL based datastores use a `<dgjdbc:JDBCConnection>` element to define the required connection parameters. Most important are the definitions of the JDBC driver to be used, the connection URL, the user name to be used for establishing connections and the user's password. Notice that the definition of a user and the password is mandatory; both elements must be supplied. The `<dgjdbc:SecurityConstraints>` element has no effect at the moment and is defined for usage in a future deegree release. The optional parameter `<dgjdbc:Encoding>` defines the character encoding to be used for establishing the JDBC connection. The interpretation of this element depends on the underlying database. E.g. PostGIS enables the definition of another character encoding for a JDBC connection than used within the connected database.

```
<!-- configuration of the persistence backend to be used -->
<xsd:annotation>
  <xsd:appinfo>
    <deegreewfs:Prefix>app</deegreewfs:Prefix>
    <deegreewfs:Backend>POSTGIS</deegreewfs:Backend>
    <deegreewfs:DefaultSRS>EPSG:4326</deegreewfs:DefaultSRS>
    <dgjdbc:JDBCConnection>
      <dgjdbc:Driver>org.postgresql.Driver</dgjdbc:Driver>
      <dgjdbc:Url>jdbc:postgresql://localhost:5432/databasename</dgjdbc:Url>
      <dgjdbc:User>USer</dgjdbc:User>
      <dgjdbc:Password>passord</dgjdbc:Password>
      <dgjdbc:SecurityConstraints/>
      <dgjdbc:Encoding>UTF-8</dgjdbc:Encoding>
    </dgjdbc:JDBCConnection>
    <deegreewfs:SuppressXLinkOutput>false</deegreewfs:SuppressXLinkOutput>
  </xsd:appinfo>
</xsd:annotation>
```

To relieve you from setting up the datastore (featuretype definition) totally manually you can use either the following example command

(Windows)

```
java -classpath .;..\..\classes;..\..\lib\postgresql-8.0-
311.jdbc3.jar;..\..\lib\log4j-1.2.9.jar;..\..\lib\deegree2.jar
org.deegree.tools.datastore.DBSchemaToDatastoreConf -tables sgid024_springs
-user deegreetest -password 'deegreetest' -driver org.postgresql.Driver
-url  jdbc:postgresql://localhost:5432/deegreetest -output
../../conf/wms/featuretypes/sgid024_springs.xsd
```

(Linux)

```
java -classpath .:../../classes:../../lib/postgresql-8.0-
311.jdbc3.jar:../../lib/log4j-1.2.9.jar:../../lib/deegree2.jar
org.deegree.tools.datastore.DBSchemaToDatastoreConf -tables sgid024_springs
-user deegreetest -password 'deegreetest' -driver org.postgresql.Driver
-url  jdbc:postgresql://localhost:5432/deegreetest -output
../../conf/wfs/featuretypes/sgid024_springs.xsd
```

or the script under wfs_home/WEB-INF/conf/tools/03_DBtoFeatureTypeDef.sh/ .bat, which both need to be adapted to your system paths.

The automatically generated *.xsd file needs some adaption afterwards as for example the <deegreewfs:DefaultSRS> and the srs= attribute.

```
<deegreewfs:MappingField field="GEOM" type="GEOMETRY" srs="4326"/>
```

Please refer the appendix D for a full datastore (featuretype definition) configuration file.

5.3 Oracle-based datastores

Oracle database of version 10 and higher supports the concept of workspaces (see Oracle documentation for details). The deegree WFS configuration supports definition of a Oracle workspace to be used when connecting a database. For this, the `<dgjdbc:Workspace>` element can be used to specify the name of the target workspace. At the moment, it is not possible to change the workspace during runtime.

```
  <xsd:annotation>
  <xsd:appinfo>
    <deegreewfs:Prefix>app</deegreewfs:Prefix>
    <deegreewfs:Backend>ORACLE</deegreewfs:Backend>
    <deegreewfs:DefaultSRS>EPSG:31466</deegreewfs:DefaultSRS>
    <JDBCConnection xmlns="http://www.deegree.org/jdbc">
      <Driver>oracle.jdbc.OracleDriver</Driver>
      <Url>jdbc:oracle:thin:@ORACLE:PORT:SID</Url>
      <User>USER</User>
      <Password>PASS</Password>
      <SecurityConstraints/>
      <Encoding>iso-8859-1</Encoding>
    </JDBCConnection>
    <deegreewfs:SuppressXLinkOutput>true</deegreewfs:SuppressXLinkOutput>
  </xsd:appinfo>
</xsd:annotation>
```

Please refer the appendix E for a full datastore (featuretype definition) configuration file.

5.4 GenericSQL-based datastores

The GenericSQL datastore enables managing of geospatial data in standard SQL databases like MSSQLServer or IBM DB2 (without spatial extension). For this, special metadata tables must be created that store the spatial index of a column containing geometric objects. All this, creating the metadata tables, importing data from a shape file and creating the spatial index can be done by the `GenericSQLShapeImporter` utility program available in deegree.

For creating a new spatially indexed table the `GenericSQLShapeImporter` requires the following parameters:

-driver: JDBC database driver to connect the target database (e.g. com.microsoft.jdbc.sqlserver.SQLServerDriver or org.hsqldb.jdbcDriver)
-url: JDBC database connection information (e.g. jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=myDB or jdbc:hsqldb:${docRoot}../../data/hsqldb/administra ['${docRoot}' must be set as in this example])
-user: user name to connect the database
-password: user's password to connect the database
-indexName: name of the index to be created; it must be unique and a valid database table name
-table: name of the table to be created to store the data
-owner: owner of tables specified by -indexName and -table (just to be set if different from the database user)
-maxTreeDepth: maximum depht of the created quadtree (shall be between 3 and 8; optional - default = 6)
-shapeFile: name of the shape file to be imported (without extension: e.g. c:/data/shapes/myShapeFile)
-idType: determines if the database type for storing object IDs shall be INTEGER or UUID (default = UUID). Using INTEGER as ID type is significantly faster.
-h or ***-?*** forces printing help

(Windows)

```
java -classpath
.;..\..\classes;..\..\lib\msbase.jar;..\..\lib\mssqlserver.jar;..\..\lib\ms
util.jar;..\..\lib\log4j-
1.2.9.jar;..\..\lib\jai_core.jar;..\..\lib\hsqldb.jar;..\..\lib\deegree2.ja
r org.deegree.tools.shape.GenericSQLShapeImporter
```

(Linux)

```
java -classpath
.:../../classes:../../lib/msbase.jar:../../lib/mssqlserver.jar:../../lib/ms
```

```
util.jar:../../lib/log4j-
1.2.9.jar:../../lib/jai_core.jar:../../lib/hsqldb.jar:../../lib/deegree2.ja
r org.deegree.tools.shape.GenericSQLShapeImporter
```

To insert a shape into an already existing geospatially enabled table, the parameters must be used:

-driver: JDBC database driver to connect the target database (e.g. com.microsoft.jdbc.sqlserver.SQLServerDriver)
-url: JDBC database connection information (e.g. jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=myDB)
-user: user name to connect the database
-password: user's password to connect the database in
-table: name of table the index shall be created for
-owner: owner of the table (optional, database user will be used if set to null)
-shapeFile: name of the shape that to be imported into the database

To relieve you from setting up the datastore (featuretype definition) totally manually you can use the following example command running for the wfs_home/WEB-INF/conf/tools/:

(Windows)

```
java -classpath
.;..\..\classes;..\..\lib\msbase.jar;..\..\lib\mssqlserver.jar;..\..\lib\ms
util.jar;..\..\lib\log4j-
1.2.9.jar;..\..\lib\jai_core.jar;..\..\lib\hsqldb.jar;..\..\lib\deegree2.ja
r org.deegree.tools.datastore.DBSchemaToDatastoreConf -tables
sgid024_stateboundary -user username -password 'password' -driver
com.microsoft.jdbc.sqlserver.SQLServerDriver -url
jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=myDB -output
..\..\conf\wfs\featuretypes\sgid024_stateboundary_mssql.xsd
```

The automatically generated *.xsd file needs some adaption afterwards. The <deegreewfs:DefaultSRS> must be specified.

GenericSQL-based datastores have the limitation that if using a spatial query against them, the BoundingBox of the requested area must be on top level of the used filter expression. If other constraints are been used they must be combined using an <And> operation. If no BoundingBox is defined on top level no spatial indexing will be used no matter if other spatial constraints are present. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetFeature outputFormat="text/xml; subtype=gml/3.1.1"
xmlns:wfs="http://www.opengis.net/wfs" xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc">
    <wfs:Query typeName="Europe">
        <ogc:Filter>
            <ogc:And>
```

```
                <ogc:BBOX>
                    <PropertyName>/Europe/Border</PropertyName>
                    <gml:Box>
                        <gml:coord>
                            <gml:X>1</gml:X>
                            <gml:Y>40</gml:Y>
                        </gml:coord>
                        <gml:coord>
                            <gml:X>12</gml:X>
                            <gml:Y>56</gml:Y>
                        </gml:coord>
                    </gml:Box>
                </ogc:BBOX>
                <ogc:PropertyIsLessThan>
                    <ogc:PropertyName>/Europe/CountryArea</ogc:PropertyName>
                    <ogc:Literal>50000</ogc:Literal>
                </ogc:PropertyIsLessThan>
            </ogc:And>
        </ogc:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

Please refer the appendix F for a full HSQLDB datastore (featuretype definition) configuration file.

5.5 Shapefile-based datastores

Here's a snippet from an example datastore configuration that serves data from a shapefile:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"..."/>
...
<xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Prefix>app</deegreewfs:Prefix>
      <deegreewfs:Backend>SHAPE</deegreewfs:Backend>
      <deegreewfs:File>../../../data/utah/vector/SGID024_Springs</deegreewfs:File>
      <deegreewfs:DefaultSRS>EPSG:26912</deegreewfs:DefaultSRS>
    </xsd:appinfo>
</xsd:annotation>
...
```

The `deegreewfs:File` element specifies the name of the shapefile to be used. Please note that the extension (.shp) must be omitted. See appendix C for the complete example. The most noteworthy thing about mapping shapefiles is the handling of geometry properties:

```
...
<xsd:element name="geom" type="gml:GeometryPropertyType">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="GEOM" type="GEOMETRY"/>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
...
```

The geometry field of a shapefile is referenced using `field="GEOM"`.

To relieve you from setting up the datastore (featuretype definition) totally manually you can use either the following example command (Windows)

```
java -classpath .;..\..\classes;..\..\lib\log4j-
1.2.9.jar;..\..\lib\deegree2.jar
org.deegree.tools.datastore.DBSchemaToDatastoreConf -driver SHAPE
-tables ..\..\data\utah\vector\SGID024_Springs -output
..\..\conf\wms\featuretypes\SGID024_Springs_shp.xsd
```

(Linux)

```
java -classpath .:../../classes:../../lib/log4j-
1.2.9.jar:../../lib/deegree2.jar
org.deegree.tools.datastore.DBSchemaToDatastoreConf -driver SHAPE
-tables ../../data/SGID024_Springs -output
../../conf/wms/featuretypes/SGID024_Springs_shp.xsd
```

or the script under wfs_home/WEB-INF/conf/tools/ShapetoFeatureTypeDef.sh/ .bat, which both need to be adapted to your system paths. -table specifies the location of the shape file.

The automatically generated *.xsd file needs some adaption afterwards. The <deegreewfs:DefaultSRS> and the <deegreewfs:File> need to be set/adapted – relative or absolut file paths are accepted where absolute file paths should start with a /// . Relative paths should start at the datastore.xsd file location. Please notice that Umlaute or spaces in shape field names should be avoided.

Please refer the appendix C for a full datastore (featuretype definition) configuration file.

5.6 ArcSDE-based datastores

If you have a specific questions according ArcSDE datasources please use the deegree mailing lists under http://www.deegree.org.

6 Advanced configuration

6.1 Manual Tomcat integration

The location of deegree's libraries and the central deegree configuration file wfs_configuration.xml should be registered with the Servlet Engine (in this case Apache Tomcat 5.5). Tomcat offers several possibilities to register and configure web contexts.

The easiest way to register deegree web services with Tomcat is to copy the `deegree-wfs.war` file to the $TOMCAT_HOME$/webapps directory. You can do this either with running or stopped tomcat. If the tomcat is started afterwards, the application should be automatically deployed. Tomcat will unpack the deegree-wfs.war file (which is nothing more than a .zip file) to the webapps directory automatically. The name of the .war sets the name of the service address:

http://localhost:8080/**deegree-wfs**

If you want to do the Tomcat installation process manually use the steps described in the following.

Unpack the deegree-wfs.war to a directory (e.g. c:/deegree/webapps/deegree-wfs) of your choice.

Afterwards Tomcat needs information about the root directory of the WFS. The easiest way is to create a XML-file in the directory $TOMCAT_HOME$/conf/Catalina/localhost, named the same as the service e.g. deegree-wfs.xml, and fill it with the following information

```
<Context docBase="c:/deegree/webapps/deegree-wfs" path="/deegree-wfs">
</Context>
```

where the docBase attribute reflects the physical location of the deegree service in the file system and the path attribute describes the virtual location of the main directory of the deegree web service. In the example, the root directory of the service is accessible at http://my.server.domain/deegree-wfs/. For further information have a look at the Tomcat documentation included with the installation.

The name of the deegree-service directory is arbitrary whereas Tomcat definitely looks for a subdirectory WEB-INF (in capital letters – even on a Windows system) in the root directory. You will find this directory after tomcat automatically unpacked the war archive. Here the Deployment-Descriptor (web.xml) is located, which is analysed by Tomcat to identify the servlet(s) belonging to the application, their names, the parameters that are delivered to the servlet(s) and information about the existing access restrictions.

Before starting deegree WFS the following dataset entry in web.xml is essential:

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>deegree 2.2</display-name>
  <description>deegree 2.2 OWS</description>
  <servlet>
    <servlet-name>owservice</servlet-name>
    <servlet-class>org.deegree.enterprise.servlet.OGCServletController</servlet-
class>

    <init-param>
      <param-name>services</param-name>
      <param-value>wfs</param-value>
      <description>
        list of supported services, e.g.: wfs,wms (comma separated) allways use
        lowercase
      </description>
    </init-param>

    <!-- WFS INITIALIZING PARAMETERS -->
    <init-param>
      <param-name>wfs.handler</param-name>
      <param-value>org.deegree.enterprise.servlet.WFSHandler</param-value>
    </init-param>
    <init-param>
      <param-name>wfs.config</param-name>
      <param-value>WEB-INF/conf/wfs/wfs_configuration.xml</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>owservice</servlet-name>
    <url-pattern>/services</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>/index.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>500</error-code>
    <location>/error.jsp</location>
  </error-page>

  <error-page>
    <exception-type>org.deegree.ogcwebservices.OGCWebServiceException</exception-
type>
    <location>/error.jsp</location>
  </error-page>

</web-app>
```

The name of the servlet and of the java-class representing the servlet should be indicated in the <servlet> tags. The servlet-name can be user defined, but care should be taken that the same name that is defined here is also used in the servlet-mapping. The servlet is located in the deegree2.jar library.

The tag <init-param> defines parameters that are analyzed by the servlet, while initializing. The transferred parameters are

- 'services': The value of this parameter contains a comma separated list of OWS that will be made available through the context. In the example only a 'wms' is defined to be available (other possible values at the moment are: wfs, wcs, sos, wpvs and csw).
- For each service listed in the 'service' init-param a handler class and a configuration file must be referenced.
- The name of the init-param for defining the handler starts with the service name ('wfs' in the example) followed by '.handler'. The value of this parameter is the name of the handler class to be used. It is possible to write a different class for this and reference it accordingly. As default ' org.deegree.enterprise.servlet.WFSHandler' should be used.
- The name of the init-param for defining the main configuration file of a service also starts with the service name followed by '.config'. Notice that you can use a relative path to the configuration file starting at the WEB-INF directory of the context.

If you want to make more than one service available through a servlet context, web.xml looks like this (the example defines a 'wms' as well as a 'wfs' and if you uncomment the WCS section even this one is accessible):

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>deegree 2.2</display-name>
  <description>deegree 2.2 OWS</description>
  <servlet>
    <servlet-name>owservice</servlet-name>
    <servlet-class>org.deegree.enterprise.servlet.OGCServletController</servlet-
    class>

    <init-param>
      <param-name>services</param-name>
      <param-value>wms,wfs</param-value>
      <description>
        list of supported services, e.g.: wfs,wms,wcs (comma separated) allways use
        lowercase
      </description>
    </init-param>

    <!-- WMS INITIALIZING PARAMETERS -->
    <init-param>
      <param-name>wms.handler</param-name>
      <param-value>org.deegree.enterprise.servlet.WMSHandler</param-value>
    </init-param>
```

```
    <init-param>
      <param-name>wms.config</param-name>
      <param-value>WEB-INF/conf/wms/wms_configuration.xml</param-value>
    </init-param>

    <!-- WFS INITIALIZING PARAMETERS -->
    <init-param>
      <param-name>wfs.handler</param-name>
      <param-value>org.deegree.enterprise.servlet.WFSHandler</param-value>
    </init-param>
    <init-param>
      <param-name>wfs.config</param-name>
      <param-value>WEB-INF/conf/wfs/LOCALWFS_configuration.xml</param-value>
    </init-param>

    <!-- WCS INITIALIZING PARAMETERS -->
    <!--
    <init-param>
      <param-name>wcs.handler</param-name>
      <param-value>org.deegree.enterprise.servlet.WCSHandler</param-value>
    </init-param>
    <init-param>
      <param-name>wcs.config</param-name>
      <param-value>WEB-INF/conf/wcs/LOCALWCS_capabilities.xml</param-value>
    </init-param>
    -->
        <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>owservice</servlet-name>
    <url-pattern>/services</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>/index.jsp</welcome-file>
  </welcome-file-list>
  <error-page>
    <error-code>500</error-code>
    <location>/error.jsp</location>
  </error-page>
  <error-page>
    <exception-type>org.deegree.ogcwebservices.OGCWebServiceException</exception-
    type>
    <location>/error.jsp</location>
  </error-page>
</web-app>
```

In case you wish to run more than on deegree service, it es recommended to use the deegree-wms.war as this configuration already includes the services wfs,wcs and wms.

The tag <servlet-mapping> defines the alias name for the servlet. It is not necessary that the <servlet-name> and <url-pattern> are identical. <url-pattern> is the name for the parameter the servlet will be called through (part of the base-URL of the service that all requests have to use). Combined with the path settings in $TOMCAT_HOME$/conf/Catalina/localhost/deegree-wfs.xml and respectively the name of the .war which you deployed in the $TOMCAT_HOME$/webapps (in our example deegree-wfs) you should be able to point to your WFS (OWS) via the following URL: http://my.server.domain/deegree-wfs/services?

6.2 Virtual property types

Usually, a property of a feature is mapped to a field (e.g. a table column) of a physical datasource. Virtual properties however, have their values defined by:

- constant values given in the application schema
- calls to SQL-functions that the database backend provides
- calls to Java functions (not implemented yet)

Please note that this functionality is currently restricted to SQL-based backends (`ORACLE`, `POSTGIS`, `GENERICSQL`).

To WFS clients, virtual properties should behave very much like regular properties. They can be queried and they can even be used as sort criteria. In transactions, the values of virtual properties are ignored (because it is usually not possible to store them).

This is the basic skeleton for all deegree WFS property definitions (virtual + conventional):

```
<xsd:element name="nameOfTheProperty" type="xsd:typeOfTheProperty" minOccurs="..."
maxOccurs="..." >
   <xsd:annotation>
      <xsd:appinfo>
         <deegreewfs:Content>
            <!-- here comes the mapping, i.e. the source of the property values -->
         </deegreewfs:Content>
      </xsd:appinfo>
   </xsd:annotation>
</xsd:element>
```

Constants

Here is an example of a property definition that assigns a constant value:

```
<!-- simple (string) valued property 'dataOrigin',  mapped to constant -->
<xsd:element name="dataOrigin" type="xsd:string" minOccurs="0">
   <xsd:annotation>
      <xsd:appinfo>
         <deegreewfs:Content>
            <deegreewfs:Constant>Imported from shapefile.</deegreewfs:Constant>
         </deegreewfs:Content>
      </xsd:appinfo>
   </xsd:annotation>
</xsd:element>
```

This means that every generated `dataOrigin` property of the corresponding feature type will have the value `Imported from shapefile`.

SQL-functions

Here is an example of a property definition that uses a simple SQL-function call to determine a property value. The generated properties of this type will contain the uppercased content of the `NAME` column:

```
<!-- simple (string) valued property 'upperName', mapped to SQL function call -->
<xsd:element name="upperName" type="xsd:string" minOccurs="0">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:SQLFunctionCall call="UPPER($1)" type="VARCHAR">
          <deegreewfs:FunctionParam>
            <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
          </deegreewfs:FunctionParam>
        </deegreewfs:SQLFunctionCall>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

The `deegreewfs:SQLFunctionCall` element tells the WFS that the property value has to be computed by an SQL-function call. The `call` attribute contains the SQL-function call (in the native syntax of the used SQL database). Note that parameters to the call have to be specified using placeholders (`$1`, `$2`, `$3`, ...). During the generation of the SQL statement, the datastore replaces each placeholder with the column specified in the respective `deegreewfs:FunctionParam` element: `$1` will be replaced by the column specified in the first FunctionParam element, `$2` will be replaced by the column specified in the second FunctionParam element and so on. The generated SQL-fragment for the example SQL-function call would look like this:

`...UPPER(`*`TABLEALIAS`*`.NAME)....`

It would also be possible, to specify the necessary columns *inline*, i.e. To specify no `deegreewfs:FunctionParam` at all and write the column names directly into the `call` attribute:

```
<!-- simple (string) valued property 'upperName', mapped to SQL function call -->
<xsd:element name="upperName" type="xsd:string" minOccurs="0">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:SQLFunctionCall call="UPPER(NAME)" type="VARCHAR"/>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

However, it is not recommended to do it like this! The problem with this approach is that the WFS cannot correctly qualify the column name (by specifying the correct alias for the table). As the deegree WFS generates quite complex SQL statements sometimes with lots of tables involved (even the same table may be used in one statement with different aliases), this can lead to ambiguities. It may happen that the SQL database can not determine which table is meant. Just d*on't use inline columns and you're on the safe side!*

Here are some more examples for possible SQL-function calls. This maps the area value of a geometry to a property named `area`:

```
<!-- simple valued property 'area', mapped to SQL function call -->
<xsd:element name="area" type="xsd:float" minOccurs="0">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:SQLFunctionCall call="area($1)" type="FLOAT">
          <deegreewfs:FunctionParam>
            <deegreewfs:MappingField field="GEOM" type="GEOMETRY" srs="4326"/>
          </deegreewfs:FunctionParam>
        </deegreewfs:SQLFunctionCall>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

The final example shows an SQL-function call with two parameters:

```
<!-- simple valued property 'queryBBOXOverlap', mapped to SQL function call -->
<xsd:element name="queryBBOXOverlap" type="xsd:float" minOccurs="0">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:SQLFunctionCall call="area(intersection($1, $2)) / (area($2)  /
100)" type="FLOAT">
          <deegreewfs:FunctionParam>
            <deegreewfs:SpecialContent>$QUERY.BBOX</deegreewfs:SpecialContent>
          </deegreewfs:FunctionParam>
          <deegreewfs:FunctionParam>
            <deegreewfs:MappingField field="GEOM" type="GEOMETRY" srs="4326"/>
          </deegreewfs:FunctionParam>
        </deegreewfs:SQLFunctionCall>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

It defines a property that contains the percentage of intersection of the geometry column `GEOM` with the bounding box of the query (`$QUERY.BBOX`). This is very special, as it uses a dynamic value from the currently processed GetFeature-query to compute the property value. The query bounding box is referenced by the `deegreewfs:SpecialContent` element. The idea behind the `deegreewfs:SpecialContent` element is that it provides a flexible and extensible means to access certain parameters of the WFS environment, e.g. the bbox of the

current GetFeature-query. Currently the only special content parameter known to the deegree WFS is `$QUERY.BBOX`, but this may change in the future.

6.3 Virtual output formats

Virtual output formats allow to use different schemas for the communication with WFS clients and for storage. In other words, requests (GetFeature + Transaction) to the WFS may be transformed before they are processed by the datastore and responses from the datastore are transformed again to match the requested output format. The transformations are performed by XSL-scripts.

In order to define a virtual format, the feature type in question has to be listed explicitly in the `wfs:FeatureTypeList` section of the `wfs_configuration.xml` document (see chapter 4.2.). Here's an example:

```
<wfs:FeatureTypeList xmlns:app="http://www.deegree.org/app">
  <wfs:FeatureType>
    <wfs:Name>app:Philosopher</wfs:Name>
    <wfs:Title>European philosopher</wfs:Title>
    <wfs:Abstract>Describes famous European philosophers</wfs:Abstract>
    <ows:Keywords>
      <ows:Keyword>philosopher</ows:Keyword>
    </ows:Keywords>
    <wfs:DefaultSRS>EPSG:4326</wfs:DefaultSRS>
    <wfs:OutputFormats>
      <wfs:Format>text/xml; subtype=gml/3.1.1</wfs:Format>
      <wfs:Format deegreewfs:outFilter="xslt/output.xsl"
deegreewfs:inFilter="xslt/input.xsl"
deegreewfs:schemaLocation="xslt/myformat.xsd">text/xml;
subtype=myformat/1.0</wfs:Format>
    </wfs:OutputFormats>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180 -90</ows:LowerCorner>
      <ows:UpperCorner>180 90</ows:UpperCorner>
    </ows:WGS84BoundingBox>
  </wfs:FeatureType>
</wfs:FeatureTypeList>
```

This defines a feature type named `app:Philosopher` with two output formats:

- `text/xml; subtype=gml/3.1.1` (default, must always be present)
- `text/xml; subtype=myformat/1.0` (additional virtual format)

Consider the following request:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature version="1.1.0" outputFormat="text/xml; subtype=myformat/1.0"
xmlns:wfs="http://www.opengis.net/wfs" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.opengis.net/wfs
http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
  <wfs:Query xmlns:app="http://www.deegree.org/app" typeName="app:Philosopher"/>
</wfs:GetFeature>
```

As the `outputFormat` attribute specifies the virtual format `text/xml; subtype=myformat/1.0`, the WFS knows that the request has to be transformed before and after processing it. It looks up the `deegreewfs:inFilter` attribute for the format and applies the specified XSL-script (`xslt/input.xsl` in this case) to the GetFeature-request. The transformed request is then processed as usual, except that the response is transformed also before it is passed back to the client. The response transformation is performed by the script specified in the `deegreewfs:outFilter` attribute (`xslt/output.xsl` in this case).

Transaction requests with features given in virtual formats must be transformed as well. The use of other formats than `text/xml; subtype=gml/3.1.1` is not defined by the WFS specification and the deegree WFS only offers some simple heuristics to determine whether XSL-processing (using the script from the `deegreewfs:inFilter` attribute) must be applied: The name of the first affected feature in the Transaction is determined. If it has a virtual format, the XSL-script specified in the corresponding `deegreewfs:inFilter` attribute is used to transform the transaction request. The transaction response is never transformed. We know that this strategy may lead to non-standard behaviour of the WFS, so use with caution.

If a DescribeFeatureType request with a virtual format is performed by the WFS, it responds with the schema document specified in the `deegreewfs:schemaLocation` attribute.

6.4 Coordinate transformation

The deegree WFS supports the querying of features in coordinate reference systems other than the native CRS of the stored features.

In order to allow the querying of other CRS, you have to add the additional CRS to the corresponding FeatureType definition in the `wfs:FeatureTypeList` section of the WFS configuration document. Here's an example:

```
...
<wfs:FeatureTypeList xmlns:app="http://www.deegree.org/app">
  <wfs:FeatureType>
    <wfs:Name>app:Philosopher</wfs:Name>
    <wfs:Title>European philosopher</wfs:Title>
    <wfs:Abstract>Describes famous European philosophers</wfs:Abstract>
```

```
    <ows:Keywords>
      <ows:Keyword>philosopher</ows:Keyword>
    </ows:Keywords>
    <wfs:DefaultSRS>EPSG:4326</wfs:DefaultSRS>
    <wfs:OtherSRS>EPSG:32615</wfs:OtherSRS>
    <wfs:OtherSRS>EPSG:32616</wfs:OtherSRS>
    <wfs:OtherSRS>EPSG:32617</wfs:OtherSRS>
    <wfs:OtherSRS>EPSG:32618</wfs:OtherSRS>
    <wfs:OutputFormats>
      <wfs:Format>text/xml; subtype=gml/3.1.1</wfs:Format>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180 -90</ows:LowerCorner>
      <ows:UpperCorner>180 90</ows:UpperCorner>
    </ows:WGS84BoundingBox>
  </wfs:FeatureType>
</wfs:FeatureTypeList>
...
```

In this example, the feature type may also be queried in the coordinate reference systems that are specified in the `wfs:OtherSRS` elements: in this case EPSG:32615-EPSG:32618.

Please note that the coordinate transformation slows the processing of the requests (in contrast to querying features in their native CRS).

6.5 Composite keys

Sometimes, a relational schema contains composite keys, i.e. one column of a table is not sufficient to uniquely identify a table row. Such schemas can be mapped to features served by the deegree WFS, but the following restrictions apply:

- only read access is possible (no Transactions)
- only PostGISDatastore and OracleDatastore support it (at the moment)

Here's an example of a feature type definition that uses three columns that constitute the primary key (and thus the feature id):

```
<xsd:element name="MyFeature" type="app:MyFeatureType"
substitutionGroup="gml:_Feature">
 <!-- annotation of each element (=feature type definition) contains associated
table name and gml:id definition -->
 <xsd:annotation>
   <xsd:appinfo>
     <deegreewfs:table>TABLE1</deegreewfs:table>
     <deegreewfs:gmlId prefix="MYFEATURE_">
       <deegreewfs:MappingField field="ID_PART1" type="INTEGER"/>
       <!-- define more columns that constitute the feature id-->
       <deegreewfs:MappingField field="ID_PART2" type="INTEGER"/>
       <deegreewfs:MappingField field="ID_PART3" type="INTEGER"/>
     </deegreewfs:gmlId>
   </xsd:appinfo>
 </xsd:annotation>
</xsd:element>
```

It's also possible to use composite keys in `deegreewfs:Relation` elements:

```
<deegreewfs:Relation>
 <deegreewfs:From>
   <deegreewfs:MappingField field="ID1" type="INTEGER" table="TABLE1"/>
   <deegreewfs:MappingField field="ID2" type="INTEGER" table="TABLE1"/>
 </deegreewfs:From>
 <deegreewfs:To>
   <deegreewfs:MappingField field="FK_ID1" type="INTEGER" table="TABLE2"/>
   <deegreewfs:MappingField field="FK_ID2" type="INTEGER" table="TABLE2"/>
 </deegreewfs:To>
</deegreewfs:Relation>
```

This would correspond to a JOIN condition of the following form:

```
..TABLE1.ID1 = TABLE2.ID1 AND TABLE1.ID2 = TABLE2.ID2...
```

6.6 Setting up WFS-Gazetteer (WFS-g)

With this demo you are able to set up a deegree2 gazetteer (WFS-g) according to the OGC best practices paper (Gazetteer Service - Application Profile of the Web Feature Service Implementation Specification) - containing two FeatureTypes covering the state of Utah.

1. If you have already set up the 'deegreetest' database skip to No. 2. Else please create a spatially enabled and UTF8 encoded POSTGIS or Oracle database called 'deegreetest' and create a user 'deegreetest'. Please refer the according documentations of the backends to do so; e.g. http://www.postgis.org/documentation/ . (A sample script under wfs_home/WEB-INF/conf/tools/01_create_spatial_database.sh can give you some advice)

2. You find two shape files containing the required data for WFS-g demo in the directory WMS_HOME/WEB-INF/data/
   SGID100_CountyBoundaries_edited.shp
   SGID024_Municipalities2004_edited.shp
   Script wfs_home/WEB-INF/conf/scripts/gazetteer/01_create_tables_index_ready.bat/.sh sums up as well the insert of shape files into database as the sql statements 02_*.sql to 05_*.sql for creating the tables. Please adapt the script to your system paths and run it. Script 05_create_tables_for_ft_si_gazeteer_metadata.sql creates the tables needed for the FeatureType SI_Gazetteer. In case you set up a gazetteer with your own data you are able to adapt the metadata in this file.

3. Now the database is ready for use but the WFS still needs some adaption. Just copy the two featuretype definitions under wfs_home/WEB-

INF/conf/wfs/featuretypes/examplefeaturetypes/gazetteer into wfs_home/WEB-INF/conf/wfs/featuretypes/ and restart tomcat.

4. Usually no FeatureType declaration has to be done in the wfs_home/WEB-INF/conf/wfs/wfs_configuration.xml as the FeatureTypeList gets filled automatically. As this gazetteer example requires XSL-transformation, a declaration IS needed to reference the XSLT and schema files. For that reason please uncomment the three gazetteer FeatureTypes.

5. You are now ready to use the WFS-gazetteer.
   Go to your internet browser and start the generic client:
   http://localhost:8080/deegree-wfs/client/client.html Check out the WFS-g with the example requests.
   All gazetteer requests start with Gaz_*.

The gazetteer demo makes use of XSL-transformation for requests and responses where the principal can be transferred to other use cases too. In order to do so, some adaptions have to be made in the following files:

1. wfs_home/WEB-INF/conf/wfs/wfs_configuration.xml
   The path to outFilter, inFilter and schemaLocation need to be specified.

```
<wfs:FeatureType>
        <wfs:Name>iso19112:SI_Gazetteer</wfs:Name>
        <wfs:Title/>
        <wfs:DefaultSRS>EPSG:26912</wfs:DefaultSRS>
        <wfs:Operations>
          <wfs:Operation>Query</wfs:Operation>
        </wfs:Operations>
        <wfs:OutputFormats>
<wfs:Format deegree:outFilter="./xslt/outfilter_main.xsl"
deegree:inFilter="./xslt/infilter_main.xsl"
deegree:schemaLocation="./schemas/iso19112.xsd">text/xml;
subtype=gml/3.1.1</wfs:Format>
          <!-- wfs:Format>text/xml; subtype=gml/3.1.1</wfs:Format-->
        </wfs:OutputFormats>
        <ows:WGS84BoundingBox>
          <ows:LowerCorner>-180 -90</ows:LowerCorner>
          <ows:UpperCorner>180 90</ows:UpperCorner>
        </ows:WGS84BoundingBox>
    </wfs:FeatureType>
```

2. outfilter_main.xsl, infilter_main.xsl and the the schema must be configured. If you want to log the transformation for debug purposes please uncomment the following entry for the class WFSHandler in the log4j.properties (wfs_home/WEB-INF/classes/)
   #log4j.logger.org.deegree.enterprise.servlet.WFSHandler=DEBUG.
   Afterwards the transformed results will be written to the apache tomcat temp directory (standard is $TOMCAT_HOME$/temp)

3. wfs_home/WEB-INF/classes/de/latlon/gazetteer/
   request_gazetteer.properties
   All requestable xpath expressions must be mapped to be transformed to the WFS internal schema.

Appendix A: Supported WFS-requests

The OGC Web Feature Service Implementation Specification 1.1 defines 7 WFS requests and their encoding as either KVP (key-value-pair) or XML requests:

- GetCapabilities
- DescribeFeatureType
- GetFeature
- Transaction
- GetFeatureWithLock
- LockFeature
- GetGmlObject
-

Here's the current status (and the limitations) for each request type in the deegree WFS implementation:

GetCapabilities

- should be implemented 100%
- XML: no known additional issues
- KVP: no known additional issues

DescribeFeatureType

- deegree WFS does not provide a correct response when virtual *and* concrete feature types are queried in the same request
- KVP: no known additional issues
- XML: no known additional issues

GetFeature

- features served by the SHAPE backend are not sorted correctly

- querying of virtual *and* concrete feature types in the same request will not provide a valid result
- XML: no known additional issues
- KVP: BBOX-parameter does not work - use the FILTER-parameter with BBOX constraint as a workaround

Transaction

- only supported by datastore backends that support transactions: currently these are only the `ORACLE` and `POSTGIS` backends
- if a WFS serves virtual *and* concrete feature types, the preprocessing XSL-script is always applied to transaction requests
- XML: no known additional issues
- KVP: not implemented (specification only defines Delete-operations for KVP anyway)

GetFeatureWithLock

- only supported by datastore backends that support transactions: currently these are only the `ORACLE` and `POSTGIS` backends

LockFeature

- only supported by datastore backends that support transactions: currently these are only the `ORACLE` and `POSTGIS` backends

GetGmlObject

- currently not implemented

Appendix B: Example WFS configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:WFS_Capabilities xmlns:deegree="http://www.deegree.org/wfs"
  xmlns:ows="http://www.opengis.net/ows" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wfs="http://www.opengis.net/wfs" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1.0" updateSequence="0">
  <!-- ================================================================== -->
  <!-- DEEGREE PARAMETERS                                                 -->
  <!-- ================================================================== -->
  <deegree:deegreeParams>
    <!-- mandatory; used as the default URL for omitted DCP-elements (in the
    OperationMetadata section) -->
    <deegree:DefaultOnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:type="simple"
          xlink:href="http://localhost:8080/deegree-wfs/services" />
    <!-- optional; default = 100 (MB); cache size available for storing feature
    instances in memory, not implemented yet -->
    <deegree:CacheSize>250</deegree:CacheSize>
    <!-- optional; default = 30 (seconds); maximum time allowed for the execution of
    a request  -->
    <deegree:RequestTimeLimit>120</deegree:RequestTimeLimit>
    <!-- optional; default = same directory as configuration; list of directories to
    be scanned for featuretypes/datastores to be served by the WFS -->
    <deegree:DataDirectoryList>
    <deegree:DataDirectory>featuretypes</deegree:DataDirectory>
    </deegree:DataDirectoryList>
    <!-- optional; default = directory from system property "java.io.tmpdir" -->
        <!-- <deegree:LockManagerDirectory>/tmp</deegree:LockManagerDirectory> -->
  </deegree:deegreeParams>
  <!-- ================================================================== -->
  <!-- SERVICE IDENTIFICATION SECTION                                     -->
  <!-- ================================================================== -->
  <ows:ServiceIdentification>
    <ows:ServiceType>WFS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.1.0</ows:ServiceTypeVersion>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>None</ows:AccessConstraints>
  </ows:ServiceIdentification>
  <!-- ================================================================== -->
  <!-- SERVICE PROVIDER SECTION                                           -->
  <!-- ================================================================== -->
  <ows:ServiceProvider>
    <ows:ProviderName>lat/lon GmbH</ows:ProviderName>
    <ows:ProviderSite xlink:href="http://www.lat-lon.de" />
    <ows:ServiceContact>
      <ows:IndividualName>Markus Schneider</ows:IndividualName>
      <ows:PositionName>deegree WFS core developer</ows:PositionName>
      <ows:ContactInfo>
        <ows:Phone>
          <ows:Voice>+49 228 184960</ows:Voice>
          <ows:Facsimile>+49 228 1849629</ows:Facsimile>
        </ows:Phone>
        <ows:Address>
          <ows:DeliveryPoint>Aennchenstr. 19</ows:DeliveryPoint>
          <ows:City>Bonn</ows:City>
          <ows:AdministrativeArea>Northrhine-Westfalia</ows:AdministrativeArea>
          <ows:PostalCode>53177</ows:PostalCode>
          <ows:Country>Germany</ows:Country>
          <ows:ElectronicMailAddress>schneider@lat-lon.de
          </ows:ElectronicMailAddress>
        </ows:Address>
        <ows:OnlineResource xlink:href="http://localhost:8080/deegree-wfs/services"
        />
        <ows:HoursOfService>24x7</ows:HoursOfService>
```

```
        <ows:ContactInstructions>Don't call us. We'll call you.
        </ows:ContactInstructions>
      </ows:ContactInfo>
      <ows:Role>PointOfContact</ows:Role>
    </ows:ServiceContact>
  </ows:ServiceProvider>
  <!-- ================================================================== -->
  <!-- OPERATIONS METADATA SECTION                                        -->
  <!-- ================================================================== -->
  <ows:OperationsMetadata>
    <ows:Operation name="GetCapabilities">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="AcceptVersions">
        <ows:Value>1.1.0</ows:Value>
        <!-- Just Version 1.1.0 is supported -->
        <!-- <ows:Value>1.0.0</ows:Value>-->
      </ows:Parameter>
      <ows:Parameter name="AcceptFormats">
        <ows:Value>text/xml</ows:Value>
      </ows:Parameter>
      <ows:Parameter name="Sections">
        <ows:Value>ServiceIdentification</ows:Value>
        <ows:Value>ServiceProvider</ows:Value>
        <ows:Value>OperationsMetadata</ows:Value>
        <ows:Value>FeatureTypeList</ows:Value>
        <ows:Value>ServesGMLObjectTypeList</ows:Value>
        <ows:Value>SupportsGMLObjectTypeList</ows:Value>
        <ows:Value>Filter_Capabilities</ows:Value>
      </ows:Parameter>
    </ows:Operation>
    <ows:Operation name="DescribeFeatureType">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="outputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
      </ows:Parameter>
    </ows:Operation>
    <ows:Operation name="GetFeature">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="resultType">
        <ows:Value>results</ows:Value>
        <ows:Value>hits</ows:Value>
      </ows:Parameter>
      <ows:Parameter name="outputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
      </ows:Parameter>
    </ows:Operation>
    <ows:Operation name="Transaction">
      <!-- ows:DCP element omitted -> filled automatically with
      DefaultOnlineResource value -->
      <ows:Parameter name="inputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
      </ows:Parameter>
      <ows:Parameter name="idgen">
        <ows:Value>GenerateNew</ows:Value>
        <ows:Value>UseExisting</ows:Value>
        <!--  <ows:Value>ReplaceDuplicate</ows:Value> -->
      </ows:Parameter>
      <ows:Parameter name="releaseAction">
        <ows:Value>ALL</ows:Value>
        <!--  <ows:Value>SOME</ows:Value> -->
      </ows:Parameter>
    </ows:Operation>
```

```
    <!--  -->
    <ows:Operation name="LockFeature" />
    <ows:Operation name="GetFeatureWithLock" />
    <!-- Sets default CRS   -->
    <ows:Parameter name="srsName">
      <ows:Value>EPSG:4326</ows:Value>
    </ows:Parameter>
    <!-- Maximum number of features, which the WFS will return to the client  -->
    <ows:Constraint name="DefaultMaxFeatures">
      <ows:Value>15000</ows:Value>
    </ows:Constraint>
    <!-- time in minutes until locked features expire; not evaluated yet -->
    <ows:Constraint name="DefaultLockExpiry">
      <ows:Value>5</ows:Value>
    </ows:Constraint>
  </ows:OperationsMetadata>
  <!-- ================================================================== -->
  <!-- FEATURE TYPE LIST SECTION                                          -->
  <!-- ================================================================== -->
  <!-- The wfs:FeatureTypeList can ususally be left empty, as it gets filled with
  information automatically by evaluation of the defined featuretype definitons
  located in <deegree:DataDirectory>s. It has to be filled, if you wish to offer
  featuretypes in different SRSs. There still has to be a featuretype definition in
  the <deegree:DataDirectory>  -->
  <wfs:FeatureTypeList xmlns:app="http://www.deegree.org/app">
    <wfs:FeatureType>
      <wfs:Name>app:Springs</wfs:Name>
      <wfs:Title>Spring in Utah</wfs:Title>
      <wfs:Abstract>All Spring in Utah</wfs:Abstract>
      <ows:Keywords>
        <ows:Keyword>Springs</ows:Keyword>
      </ows:Keywords>
      <wfs:DefaultSRS>EPSG:26912</wfs:DefaultSRS>
      <wfs:OtherSRS>EPSG:4326</wfs:OtherSRS>
      <wfs:OutputFormats>
        <wfs:Format>text/xml; subtype=gml/3.1.1</wfs:Format>
      </wfs:OutputFormats>
      <ows:WGS84BoundingBox>
        <ows:LowerCorner>-180 -90</ows:LowerCorner>
        <ows:UpperCorner>180 90</ows:UpperCorner>
      </ows:WGS84BoundingBox>
    </wfs:FeatureType>

  </wfs:FeatureTypeList>
  <!-- ================================================================== -->
  <!-- FILTER CAPABILITIES SECTION                                        -->
  <!-- ================================================================== -->
  <ogc:Filter_Capabilities>
    <!-- Don't change <ogc:Filter_Capabilities>, its client information only -->
    <ogc:Spatial_Capabilities>
      <ogc:GeometryOperands>
        <ogc:GeometryOperand>gml:Envelope</ogc:GeometryOperand>
        <ogc:GeometryOperand>gml:Point</ogc:GeometryOperand>
        <ogc:GeometryOperand>gml:LineString</ogc:GeometryOperand>
        <ogc:GeometryOperand>gml:Polygon</ogc:GeometryOperand>
      </ogc:GeometryOperands>
      <ogc:SpatialOperators>
        <ogc:SpatialOperator name="BBOX" />
        <ogc:SpatialOperator name="Equals" />
        <ogc:SpatialOperator name="Disjoint" />
        <ogc:SpatialOperator name="Intersects" />
        <ogc:SpatialOperator name="Touches" />
        <ogc:SpatialOperator name="Crosses" />
        <ogc:SpatialOperator name="Within" />
        <ogc:SpatialOperator name="Contains" />
```

```
        <ogc:SpatialOperator name="Overlaps" />
        <ogc:SpatialOperator name="Beyond" />
      </ogc:SpatialOperators>
    </ogc:Spatial_Capabilities>
    <ogc:Scalar_Capabilities>
      <ogc:LogicalOperators />
      <ogc:ComparisonOperators>
        <ogc:ComparisonOperator>LessThan</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>GreaterThan</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>LessThanEqualTo</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>GreaterThanEqualTo</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>EqualTo</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>Like</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>Between</ogc:ComparisonOperator>
        <ogc:ComparisonOperator>NullCheck</ogc:ComparisonOperator>
      </ogc:ComparisonOperators>
      <ogc:ArithmeticOperators>
        <ogc:SimpleArithmetic />
      </ogc:ArithmeticOperators>
    </ogc:Scalar_Capabilities>
    <ogc:Id_Capabilities>
      <ogc:EID />
      <ogc:FID />
    </ogc:Id_Capabilities>
  </ogc:Filter_Capabilities>
</wfs:WFS_Capabilities>
```

Appendix C: Example datastore config (SHAPE)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:deegreewfs="http://www.deegree.org/wfs"
xmlns:dgjdbc="http://www.deegree.org/jdbc" xmlns:app="http://www.deegree.org/app"
targetNamespace="http://www.deegree.org/app" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
 <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/geometryAggregates.xsd"/>
 <xsd:import namespace="http://www.opengis.net/wfs"
schemaLocation="http://schemas.opengis.net/wfs/1.1.0/wfs.xsd"/>
 <!-- configuration of the persistence backend to be used -->
 <xsd:annotation>
   <xsd:appinfo>
     <deegreewfs:Prefix>app</deegreewfs:Prefix>
     <deegreewfs:Backend>SHAPE</deegreewfs:Backend>
     <deegreewfs:DefaultSRS>EPSG:4326</deegreewfs:DefaultSRS>
     <deegreewfs:File>BasicPolygons</deegreewfs:File>
   </xsd:appinfo>
 </xsd:annotation>
 <!-- ============================================================== -->
 <xsd:element name="BasicPolygons" type="app:BasicPolygonsType"
substitutionGroup="gml:_Feature"/>
 <xsd:complexType name="BasicPolygonsType">
   <xsd:annotation>
     <xsd:appinfo>
       <deegreewfs:table>BasicPolygons</deegreewfs:table>
       <deegreewfs:gmlId prefix="POLY_">
         <deegreewfs:MappingField field="FID" type="INTEGER"/>
       </deegreewfs:gmlId>
     </xsd:appinfo>
   </xsd:annotation>
   <xsd:complexContent>
     <xsd:extension base="gml:AbstractFeatureType">
       <xsd:sequence>
         <xsd:element name="id" type="xsd:integer">
           <xsd:annotation>
             <xsd:appinfo>
               <deegreewfs:Content>
                 <deegreewfs:MappingField field="ID" type="INTEGER"/>
               </deegreewfs:Content>
             </xsd:appinfo>
           </xsd:annotation>
         </xsd:element>
         <xsd:element name="type" type="xsd:integer">
           <xsd:annotation>
             <xsd:appinfo>
               <deegreewfs:Content>
                 <deegreewfs:MappingField field="TYP" type="INTEGER"/>
               </deegreewfs:Content>
             </xsd:appinfo>
           </xsd:annotation>
         </xsd:element>
         <xsd:element name="name" type="xsd:string">
           <xsd:annotation>
             <xsd:appinfo>
               <deegreewfs:Content>
                 <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
               </deegreewfs:Content>
             </xsd:appinfo>
```

```
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="geom" type="gml:GeometryPropertyType">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="GEOM" type="GEOMETRY"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <!-- ========================================================================== -->
</xsd:schema>
```

Appendix D: Example datastore config (POSTGIS)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:deegreewfs="http://www.deegree.org/wfs"
xmlns:dgjdbc="http://www.deegree.org/jdbc" xmlns:app="http://www.deegree.org/app"
targetNamespace="http://www.deegree.org/app" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
  <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/geometryAggregates.xsd"/>
  <!-- configuration of the persistence backend to be used -->
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Prefix>app</deegreewfs:Prefix>
      <deegreewfs:Backend>POSTGIS</deegreewfs:Backend>
      <deegreewfs:DefaultSRS>EPSG:4326</deegreewfs:DefaultSRS>
      <dgjdbc:JDBCConnection>
        <dgjdbc:Driver>org.postgresql.Driver</dgjdbc:Driver>
        <dgjdbc:Url>jdbc:postgresql://localhost:5432/deegreetest</dgjdbc:Url>
        <dgjdbc:User>deegreetest</dgjdbc:User>
        <dgjdbc:Password>deegreetest</dgjdbc:Password>
        <dgjdbc:SecurityConstraints/>
        <dgjdbc:Encoding>UTF-8</dgjdbc:Encoding>
      </dgjdbc:JDBCConnection>
      <deegreewfs:SuppressXLinkOutput>false</deegreewfs:SuppressXLinkOutput>
    </xsd:appinfo>
  </xsd:annotation>
  <!-- ========================================================================= -->
  <xsd:element name="Philosopher" type="app:PhilosopherType"
substitutionGroup="gml:_Feature">
    <xsd:annotation>
      <xsd:appinfo>
        <deegreewfs:table>PHILOSOPHER</deegreewfs:table>
        <deegreewfs:gmlId prefix="PHILOSOPHER_">
          <deegreewfs:MappingField field="ID" type="INTEGER"/>
          <deegreewfs:IdGenerator type="DB_SEQ">
            <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
          </deegreewfs:IdGenerator>
          <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
        </deegreewfs:gmlId>
        <deegreewfs:visible>true</deegreewfs:visible>
        <deegreewfs:transaction update="true" delete="true" insert="true"/>
      </xsd:appinfo>
    </xsd:annotation>
    <!-- annotation of each element (=feature type definition) contains associated
    table name and gml:id definition -->
  </xsd:element>
  <!-- ========================================================================= -->
  <xsd:complexType name="PhilosopherType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <!-- simple (integer) valued property 'id' (feature id without prefix)-->
          <xsd:element name="id" type="xsd:integer">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field="ID" type="INTEGER"/>
                </deegreewfs:Content>
              </xsd:appinfo>
```

```
  </xsd:annotation>
</xsd:element>
<!-- simple (string) valued property 'name' -->
<xsd:element name="name" type="xsd:string">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- simple (string) valued property 'sex' -->
<xsd:element name="sex" type="xsd:string">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="SEX" type="CHAR"/>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- simple (string) valued property 'subject', located in related table
-->
<xsd:element name="subject" type="xsd:string" minOccurs="0"
maxOccurs="unbounded">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
        <deegreewfs:Relation>
          <deegreewfs:From>
            <deegreewfs:MappingField field="ID" type="INTEGER"/>
          </deegreewfs:From>
          <deegreewfs:To fk="true">
            <deegreewfs:MappingField field="PHILOSOPHER_ID" type="INTEGER"
            table="SUBJECT"/>
          </deegreewfs:To>
        </deegreewfs:Relation>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- complex valued property 'isAuthorOf' -->
<xsd:element name="isAuthorOf" type="gml:FeaturePropertyType"
minOccurs="0" maxOccurs="unbounded">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      <deegreewfs:Content type="app:Book">
        <deegreewfs:Relation>
          <deegreewfs:From>
            <deegreewfs:MappingField field="ID" type="INTEGER"/>
          </deegreewfs:From>
          <deegreewfs:To fk="true">
            <deegreewfs:MappingField field="PHILOSOPHER_ID"
            type="INTEGER"/>
          </deegreewfs:To>
        </deegreewfs:Relation>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

```
<!-- simple (date) valued property 'dateOfBirth' -->
<xsd:element name="dateOfBirth" type="xsd:date">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="DATE_OF_BIRTH" type="DATE"/>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- complex valued property 'placeOfBirth' -->
<xsd:element name="placeOfBirth" type="gml:FeaturePropertyType">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      <deegreewfs:Content type="app:Place">
        <deegreewfs:Relation>
          <deegreewfs:From fk="true">
            <deegreewfs:MappingField field="PLACE_OF_BIRTH"
            type="INTEGER"/>
          </deegreewfs:From>
          <deegreewfs:To>
            <deegreewfs:MappingField field="ID" type="INTEGER"/>
          </deegreewfs:To>
        </deegreewfs:Relation>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- simple (date) valued property 'dateOfDeath' -->
<xsd:element name="dateOfDeath" type="xsd:date">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      <deegreewfs:Content>
        <deegreewfs:MappingField field="DATE_OF_DEATH" type="DATE"/>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- complex valued property 'placeOfDeath' -->
<xsd:element name="placeOfDeath" type="gml:FeaturePropertyType">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      <deegreewfs:Content type="app:Place">
        <deegreewfs:Relation>
          <deegreewfs:From fk="true">
            <deegreewfs:MappingField field="PLACE_OF_DEATH"
            type="INTEGER"/>
          </deegreewfs:From>
          <deegreewfs:To>
            <deegreewfs:MappingField field="ID" type="INTEGER"/>
          </deegreewfs:To>
        </deegreewfs:Relation>
      </deegreewfs:Content>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- complex valued property 'friend' (recursive) -->
<xsd:element name="friend" type="gml:FeaturePropertyType" minOccurs="0"
maxOccurs="unbounded">
  <xsd:annotation>
    <xsd:appinfo>
```

```
            <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
            <deegreewfs:Content type="app:Philosopher">
              <deegreewfs:Relation>
                <deegreewfs:From>
                  <deegreewfs:MappingField field="ID" type="INTEGER"/>
                </deegreewfs:From>
                <deegreewfs:To fk="true">
                  <deegreewfs:MappingField field="PHILOSOPHER1_ID"
                  type="INTEGER" table="IS_FRIEND_OF"/>
                </deegreewfs:To>
              </deegreewfs:Relation>
              <deegreewfs:Relation>
                <deegreewfs:From fk="true">
                  <deegreewfs:MappingField field="PHILOSOPHER2_ID"
                  type="INTEGER"/>
                </deegreewfs:From>
                <deegreewfs:To>
                  <deegreewfs:MappingField field="ID" type="INTEGER"
                  table="PHILOSOPHER"/>
                </deegreewfs:To>
              </deegreewfs:Relation>
            </deegreewfs:Content>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<!-- ========================================================================= -->
<xsd:element name="Book" type="app:BookType" substitutionGroup="gml:_Feature">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:table>BOOK</deegreewfs:table>
      <deegreewfs:gmlId prefix="BOOK_">
        <deegreewfs:MappingField field="ID" type="INTEGER"/>
        <deegreewfs:IdGenerator type="DB_SEQ">
          <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
        </deegreewfs:IdGenerator>
        <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
      </deegreewfs:gmlId>
      <deegreewfs:visible>false</deegreewfs:visible>
      <deegreewfs:transaction update="true" delete="true" insert="true"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- ========================================================================= -->
<xsd:complexType name="BookType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <!-- simple (string) valued property 'title' -->
        <xsd:element name="title" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="TITLE" type="VARCHAR"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- simple (date) valued property 'publicationDate' -->
        <xsd:element name="publicationDate" type="xsd:date" minOccurs="0">
          <xsd:annotation>
```

```
            <xsd:appinfo>
              <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="PUB_DATE" type="DATE"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- ============================================================================ -->
<xsd:element name="Place" type="app:PlaceType" substitutionGroup="gml:_Feature">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:table>PLACE</deegreewfs:table>
      <deegreewfs:gmlId prefix="PLACE_">
        <deegreewfs:MappingField field="ID" type="INTEGER"/>
        <deegreewfs:IdGenerator type="DB_SEQ">
          <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
        </deegreewfs:IdGenerator>
        <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      </deegreewfs:gmlId>
      <deegreewfs:visible>false</deegreewfs:visible>
      <deegreewfs:transaction update="true" delete="true" insert="true"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- ============================================================================ -->
<xsd:complexType name="PlaceType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <!-- simple (string) valued property 'name' -->
        <xsd:element name="name" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- complex valued property 'country' -->
        <xsd:element name="country" type="gml:FeaturePropertyType" minOccurs="0">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
              <deegreewfs:Content type="app:Country">
                <deegreewfs:Relation>
                  <deegreewfs:From fk="true">
                    <deegreewfs:MappingField field="COUNTRY_ID" type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
```

```
  </xsd:complexContent>
</xsd:complexType>
<!-- ========================================================================== -->
<xsd:element name="Country" substitutionGroup="gml:_Feature">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:table>COUNTRY</deegreewfs:table>
      <deegreewfs:gmlId prefix="COUNTRY_">
        <deegreewfs:MappingField field="ID" type="INTEGER"/>
        <deegreewfs:IdGenerator type="DB_SEQ">
          <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
        </deegreewfs:IdGenerator>
        <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
      </deegreewfs:gmlId>
      <deegreewfs:visible>true</deegreewfs:visible>
      <deegreewfs:transaction update="true" delete="true" insert="true"/>
    </xsd:appinfo>
  </xsd:annotation>
  <!-- anonymous (inline) complex type definitions are allowed as well -->
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <!-- simple (string) valued property 'name' -->
          <xsd:element name="name" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- simple (string) valued property 'upperName', mapped to SQL
          function call -->
          <xsd:element name="upperName" type="xsd:string" minOccurs="0">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:SQLFunctionCall call="UPPER($1)" type="VARCHAR">
                    <deegreewfs:FunctionParam>
                      <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
                    </deegreewfs:FunctionParam>
                  </deegreewfs:SQLFunctionCall>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- simple (string) valued property 'dataOrigin',  mapped to constant
          -->
          <xsd:element name="dataOrigin" type="xsd:string" minOccurs="0">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:Constant>Imported from
                  shapefile.</deegreewfs:Constant>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- simple valued property 'area', mapped to SQL function call -->
          <xsd:element name="area" type="xsd:float" minOccurs="0">
```

```
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:SQLFunctionCall call="area($1)" type="FLOAT">
                    <deegreewfs:FunctionParam>
                      <deegreewfs:MappingField field="GEOM" type="GEOMETRY"
                      srs="4326"/>
                    </deegreewfs:FunctionParam>
                  </deegreewfs:SQLFunctionCall>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- simple valued property 'queryBBOXOverlap', mapped to SQL function
          call -->
          <xsd:element name="queryBBOXOverlap" type="xsd:float" minOccurs="0">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:SQLFunctionCall call="area(intersection($1, $2)) /
                  (area($2)  / 100)" type="FLOAT">
                    <deegreewfs:FunctionParam>
                      <deegreewfs:SpecialContent>$QUERY.BBOX</deegreewfs:SpecialC
                      ontent>
                    </deegreewfs:FunctionParam>
                    <deegreewfs:FunctionParam>
                      <deegreewfs:MappingField field="GEOM" type="GEOMETRY"
                      srs="4326"/>
                    </deegreewfs:FunctionParam>
                  </deegreewfs:SQLFunctionCall>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- geometry property 'geom' -->
          <xsd:element name="geom" type="gml:GeometryPropertyType">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:SRS>EPSG:4326</deegreewfs:SRS>
                  <deegreewfs:MappingField field="GEOM" type="GEOMETRY"
                  srs="4326"/>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<!-- ========================================================================= -->
</xsd:schema>
```

Appendix E: Example datastore config (ORACLE)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:deegreewfs="http://www.deegree.org/wfs"
xmlns:dgjdbc="http://www.deegree.org/jdbc" xmlns:app="http://www.deegree.org/app"
targetNamespace="http://www.deegree.org/app" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
  <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/geometryAggregates.xsd"/>
  <xsd:import namespace="http://www.opengis.net/wfs"
schemaLocation="http://schemas.opengis.net/wfs/1.1.0/wfs.xsd"/>
  <!-- configuration of the persistence backend to be used -->
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Prefix>app</deegreewfs:Prefix>
      <deegreewfs:Backend>ORACLE</deegreewfs:Backend>
      <deegreewfs:DefaultSRS>EPSG:4326</deegreewfs:DefaultSRS>
      <dgjdbc:JDBCConnection>
        <dgjdbc:Driver>oracle.jdbc.OracleDriver</dgjdbc:Driver>
        <dgjdbc:Url>jdbc:oracle:thin:@localhost:1521:deegreetest</dgjdbc:Url>
        <dgjdbc:User>deegreetest</dgjdbc:User>
        <dgjdbc:Password>deegreetest</dgjdbc:Password>
        <dgjdbc:SecurityConstraints/>
        <dgjdbc:Encoding>iso-8859-1</dgjdbc:Encoding>
      </dgjdbc:JDBCConnection>
      <deegreewfs:SuppressXLinkOutput>false</deegreewfs:SuppressXLinkOutput>
    </xsd:appinfo>
  </xsd:annotation>
  <!-- ============================================================== -->
  <xsd:element name="Philosopher" type="app:PhilosopherType"
substitutionGroup="gml:_Feature">
    <!-- annotation of each element (=feature type definition) contains associated
table name and gml:id definition -->
    <xsd:annotation>
      <xsd:appinfo>
        <deegreewfs:table>PHILOSOPHER</deegreewfs:table>
        <deegreewfs:gmlId prefix="PHILOSOPHER_">
          <deegreewfs:MappingField field="ID" type="INTEGER"/>
          <deegreewfs:IdGenerator type="DB_SEQ">
            <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
          </deegreewfs:IdGenerator>
        </deegreewfs:gmlId>
        <deegreewfs:transaction update="true" delete="true" insert="true"/>
      </xsd:appinfo>
    </xsd:annotation>
  </xsd:element>
  <!-- ============================================================== -->
  <xsd:complexType name="PhilosopherType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <!-- simple (integer) valued property 'id' (feature id without prefix) --
>
          <xsd:element name="id" type="xsd:integer">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field="ID" type="INTEGER"/>
                </deegreewfs:Content>
```

```
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- simple (string) valued property 'name' -->
        <xsd:element name="name" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- simple (string) valued property 'sex' -->
        <xsd:element name="sex" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="SEX" type="CHAR"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- simple (string) valued property 'subject', located in related table
-->
        <xsd:element name="subject" type="xsd:string" minOccurs="0"
maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
                <deegreewfs:Relation>
                  <deegreewfs:From>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To fk="true">
                    <deegreewfs:MappingField field="PHILOSOPHER_ID"
type="INTEGER" table="SUBJECT"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- complex valued property 'isAuthorOf' -->
        <xsd:element name="isAuthorOf" type="gml:FeaturePropertyType"
minOccurs="0" maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:IdentityPart>false</deegreewfs:IdentityPart>
              <deegreewfs:Content type="app:Book">
                <deegreewfs:Relation>
                  <deegreewfs:From>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To fk="true">
                    <deegreewfs:MappingField field="PHILOSOPHER_ID"
type="INTEGER"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
```

```
          </xsd:element>
          <!-- simple (date) valued property 'dateOfBirth' -->
          <xsd:element name="dateOfBirth" type="xsd:date">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field="DATE_OF_BIRTH" type="DATE"/>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- complex valued property 'placeOfBirth' -->
          <xsd:element name="placeOfBirth" type="gml:FeaturePropertyType">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content type="app:Place">
                  <deegreewfs:Relation>
                    <deegreewfs:From fk="true">
                      <deegreewfs:MappingField field="PLACE_OF_BIRTH"
type="INTEGER"/>
                    </deegreewfs:From>
                    <deegreewfs:To>
                      <deegreewfs:MappingField field="ID" type="INTEGER"/>
                    </deegreewfs:To>
                  </deegreewfs:Relation>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- simple (date) valued property 'dateOfDeath' -->
          <xsd:element name="dateOfDeath" type="xsd:date">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field="DATE_OF_DEATH" type="DATE"/>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- complex valued property 'placeOfDeath' -->
          <xsd:element name="placeOfDeath" type="gml:FeaturePropertyType">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content type="app:Place">
                  <deegreewfs:Relation>
                    <deegreewfs:From fk="true">
                      <deegreewfs:MappingField field="PLACE_OF_DEATH"
type="INTEGER"/>
                    </deegreewfs:From>
                    <deegreewfs:To>
                      <deegreewfs:MappingField field="ID" type="INTEGER"/>
                    </deegreewfs:To>
                  </deegreewfs:Relation>
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <!-- complex valued property 'friend' (recursive) -->
          <xsd:element name="friend" type="gml:FeaturePropertyType" minOccurs="0"
maxOccurs="unbounded">
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content type="app:Philosopher">
                  <deegreewfs:Relation>
                    <deegreewfs:From>
```

```
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To fk="true">
                    <deegreewfs:MappingField field="PHILOSOPHER1_ID"
type="INTEGER" table="IS_FRIEND_OF"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
                <deegreewfs:Relation>
                  <deegreewfs:From fk="true">
                    <deegreewfs:MappingField field="PHILOSOPHER2_ID"
type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To>
                    <deegreewfs:MappingField field="ID" type="INTEGER"
table="PHILOSOPHER"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- ============================================================================ -->
<xsd:element name="Book" type="app:BookType" substitutionGroup="gml:_Feature">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:table>BOOK</deegreewfs:table>
      <deegreewfs:gmlId prefix="BOOK_">
        <deegreewfs:MappingField field="ID" type="INTEGER"/>
        <deegreewfs:IdGenerator type="DB_SEQ">
          <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
        </deegreewfs:IdGenerator>
        <deegreewfs:IdentityPart>true</deegreewfs:IdentityPart>
      </deegreewfs:gmlId>
      <deegreewfs:visible>false</deegreewfs:visible>
      <deegreewfs:transaction update="true" delete="true" insert="true"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- ============================================================================ -->
<xsd:complexType name="BookType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <!-- simple (string) valued property 'title' -->
        <xsd:element name="title" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="TITLE" type="VARCHAR"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- simple (date) valued property 'publicationDate' -->
        <xsd:element name="publicationDate" type="xsd:date" minOccurs="0">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="PUB_DATE" type="DATE"/>
              </deegreewfs:Content>
            </xsd:appinfo>
```

```
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- ========================================================================== -->
<xsd:element name="Place" type="app:PlaceType" substitutionGroup="gml:_Feature">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:table>PLACE</deegreewfs:table>
      <deegreewfs:gmlId prefix="PLACE_">
        <deegreewfs:MappingField field="ID" type="INTEGER"/>
        <deegreewfs:IdGenerator type="DB_SEQ">
          <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
        </deegreewfs:IdGenerator>
      </deegreewfs:gmlId>
      <deegreewfs:visible>false</deegreewfs:visible>
      <deegreewfs:transaction update="true" delete="true" insert="true"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
<!-- ========================================================================== -->
<xsd:complexType name="PlaceType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <!-- simple (string) valued property 'name' -->
        <xsd:element name="name" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <!-- complex valued property 'country' -->
        <xsd:element name="country" type="gml:FeaturePropertyType" minOccurs="0">
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content type="app:Country">
                <deegreewfs:Relation>
                  <deegreewfs:From fk="true">
                    <deegreewfs:MappingField field="COUNTRY_ID" type="INTEGER"/>
                  </deegreewfs:From>
                  <deegreewfs:To>
                    <deegreewfs:MappingField field="ID" type="INTEGER"/>
                  </deegreewfs:To>
                </deegreewfs:Relation>
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- ========================================================================== -->
<xsd:element name="Country" substitutionGroup="gml:_Feature">
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:table>COUNTRY</deegreewfs:table>
      <deegreewfs:gmlId prefix="COUNTRY_">
        <deegreewfs:MappingField field="ID" type="INTEGER"/>
```

```
          <deegreewfs:IdGenerator type="DB_SEQ">
            <deegreewfs:param name="sequence">FID_seq</deegreewfs:param>
          </deegreewfs:IdGenerator>
        </deegreewfs:gmlId>
        <deegreewfs:visible>true</deegreewfs:visible>
        <deegreewfs:transaction update="true" delete="true" insert="true"/>
      </xsd:appinfo>
    </xsd:annotation>
    <!-- anonymous (inline) complex type definitions are allowed as well -->
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="gml:AbstractFeatureType">
          <xsd:sequence>
            <!-- simple (string) valued property 'name' -->
            <xsd:element name="name" type="xsd:string">
              <xsd:annotation>
                <xsd:appinfo>
                  <deegreewfs:Content>
                    <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
                  </deegreewfs:Content>
                </xsd:appinfo>
              </xsd:annotation>
            </xsd:element>
            <!-- simple (string) valued property 'upperName', mapped to SQL
function call -->
            <xsd:element name="upperName" type="xsd:string" minOccurs="0">
              <xsd:annotation>
                <xsd:appinfo>
                  <deegreewfs:Content>
                    <deegreewfs:SQLFunctionCall call="UPPER($1)" type="VARCHAR">
                      <deegreewfs:FunctionParam>
                        <deegreewfs:MappingField field="NAME" type="VARCHAR"/>
                      </deegreewfs:FunctionParam>
                    </deegreewfs:SQLFunctionCall>
                  </deegreewfs:Content>
                </xsd:appinfo>
              </xsd:annotation>
            </xsd:element>
            <!-- simple (string) valued property 'dataOrigin',  mapped to constant
-->
            <xsd:element name="dataOrigin" type="xsd:string" minOccurs="0">
              <xsd:annotation>
                <xsd:appinfo>
                  <deegreewfs:Content>
                    <deegreewfs:Constant>Imported from
shapefile.</deegreewfs:Constant>
                  </deegreewfs:Content>
                </xsd:appinfo>
              </xsd:annotation>
            </xsd:element>
            <!-- geometry property 'geom' -->
            <xsd:element name="geom" type="gml:MultiPolygonPropertyType"
minOccurs="0">
              <xsd:annotation>
                <xsd:appinfo>
                  <deegreewfs:Content>
                    <deegreewfs:SRS>EPSG:4326</deegreewfs:SRS>
                    <deegreewfs:MappingField field="GEOM" type="GEOMETRY"
srs="8192"/>
                  </deegreewfs:Content>
                </xsd:appinfo>
              </xsd:annotation>
            </xsd:element>
          </xsd:sequence>
        </xsd:extension>
```

```
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
  <!-- ========================================================================== -->
</xsd:schema>
```

Appendix F: Example datastore config (GENERICSQL: example HSQLDB)

```
<xsd:schema targetNamespace="http://www.deegree.org/app"
xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:deegreewfs="http://www.deegree.org/wfs"
  xmlns:ogc="http://www.opengis.net/ogc" xmlns:app="http://www.deegree.org/app"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:import namespace="http://www.opengis.net/gml"
    schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd" />
  <xsd:import namespace="http://www.opengis.net/gml"
    schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/geometryAggregates.xs
d" />
  <!-- configuration for the persistence backend to be used -->
  <xsd:annotation>
    <xsd:appinfo>
      <deegreewfs:Prefix>app</deegreewfs:Prefix>
      <deegreewfs:Backend>GENERICSQL</deegreewfs:Backend>
      <deegreewfs:DefaultSRS>EPSG:26912</deegreewfs:DefaultSRS>
      <JDBCConnection xmlns="http://www.deegree.org/jdbc">
        <Driver>org.hsqldb.jdbcDriver</Driver>
        <Url>jdbc:hsqldb:../../../data/hsqldb/administration</Url>
        <User>sa</User>
        <Password></Password>
        <SecurityConstraints />
        <Encoding>iso-8859-1</Encoding>
      </JDBCConnection>
    </xsd:appinfo>
  </xsd:annotation><!--
================================================================ -->
  <xsd:element name='StateBoundary' type='app:StateBoundaryType'
substitutionGroup="gml:_Feature">
    <xsd:annotation>
      <xsd:appinfo>
        <deegreewfs:table>SGID024_STATEBOUNDARY</deegreewfs:table>
        <deegreewfs:gmlId prefix="ID_">
          <deegreewfs:MappingField field='FEATURE_ID' type="VARCHAR" />
        </deegreewfs:gmlId>
      </xsd:appinfo>
    </xsd:annotation>
  </xsd:element>
  <!-- ================================================================ -->
  <xsd:complexType name='StateBoundaryType'>
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name='feature_id' type='xsd:integer'>
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field='FEATURE_ID' type='INTEGER' />
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name='objectid' type='xsd:integer'>
            <xsd:annotation>
              <xsd:appinfo>
                <deegreewfs:Content>
                  <deegreewfs:MappingField field='OBJECTID' type='INTEGER' />
                </deegreewfs:Content>
              </xsd:appinfo>
            </xsd:annotation>
```

```
        </xsd:element>
        <xsd:element name='state' type='xsd:string'>
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field='STATE' type='VARCHAR' />
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name='shape_area' type='xsd:float'>
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field='SHAPE_AREA' type='FLOAT' />
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name='shape_len' type='xsd:float'>
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field='SHAPE_LEN' type='FLOAT' />
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name='geometry' type='gml:GeometryPropertyType'>
          <xsd:annotation>
            <xsd:appinfo>
              <deegreewfs:Content>
                <deegreewfs:MappingField field='GEOMETRY' type='GEOMETRY' srs="-1"
                />
              </deegreewfs:Content>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:schema>
```