

deegree Web Perspective View 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 1662152

53115 Bonn

Tel. ++49 228 732098

Change log

Date	Description	Author
2008-01-16	Changes and fixes for demo release v2.1 stable	Hanko Rubach
2008-04-15	Update to v2.2	Judit Mays

Table of Content

1 Introduction.....	5
2 Get deegree WPVS demo up and running.....	8
2.1 Pre-requisites.....	8
2.2 Download.....	9
2.3 Tomcat integration.....	9
2.3.1 Easy installation into Apache Tomcat.....	9
2.3.2 Pointing Tomcat explicitly to application home directory	9
2.4 Testing the installation.....	12
3 Configuring the WPVS.....	14
3.1 Architecture.....	14
3.2 Configuration file.....	14
3.3 Service Parameters.....	15
3.4 Datasets.....	16
3.5 DataSources.....	19
3.5.1 A WMS DataSource.....	21
3.5.2 A WCS DataSource.....	22
3.5.3 A WFS DataSource.....	23
3.6 Elevation Models.....	25
3.6.1 An example: WCS-based Elevation Model.....	25
4 Fine-tuning the WPVS.....	27
4.1 Deegree parameters.....	27
4.2 Rendering options.....	30
4.3 Material properties.....	31
5 Different ways to install deegree2 WPVS into Tomcat	32
5.1 Manually installing the demo in Tomcat.....	32
5.2 Installing a single deegree.jar.....	32
5.2.1 Tomcat configuration files.....	32
5.3 Installing the latest WPVS from SVN.....	35

6 Extras.....	36
6.1 adding the optional Get3DFeatureInfo-Request.....	36
6.2 teaching the WFS the Keyhole Markup Language (KML).....	37
Appendix A Third party libraries.....	39
Appendix B Complete deegree2 WPVS configuration....	42
Appendix C Rendering configuration options.....	51
Appendix D Material properties file.....	54

1 Introduction

deegree is a Java Framework that offers 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 accessible at www.deegree.org.

The deegree2 framework is the new release of deegree and includes number of features as well as new services. It is written in the Java 5 language. This documentation describes setup and configuration of one of these new services, the deegree2 *Web Perspective View Service* (WPVS), in deegree (1) known as the Web Terrain Service (WTS). It is a partial implementation of OGC's *Web Perspective View Service* Implementation Specification 1.0.0 which is in the draft stadium since 20th October 2005.

Besides a WPVS, 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

A WPVS is a web service which is able to generate perspective terrain views, e.g. images of a certain terrain/area (possibly containing three dimensional objects, e.g. building or trees) rendered from a requested viewpoint. For this purposes the WPVS needs to process and render different kinds of geospatial data which is retrieved from different pre-configured datasets. These datasets can be remote OGC web services or locally installed deegree 2 web services.

With deegree's WPVS it is easy to display vector based- as well as raster data from different storage formats and deliver it to any client that is able to perform a HTTP GET or POST request (at the moment, only GET requests are supported). The currently supported storage formats are:

- PostgreSQL / PostGIS
- Oracle (Spatial / Locator)
- databases allowing JDBC-connections
- ESRI Shapefiles

(all provided by the deegree WFS)

- GML2 and GML3 provided by a OGC WFS 1.1.0
- JPEG, GIF, PNG, BMP, TIFF and GeoTIFF images (provided by the deegree WCS)
- OGC WMS 1.1.0/1.1.1/1.3.0

This document is divided into different chapters which help you to setup and understand a deegree WPVS. First we attempt to give you a start on how to install and integrate the WPVS (and eventually other deegree web services) into a tomcat context in chapter 2 “Get deegree WPVS demo up and running”. The discussion is fairly general, so if setting up a web service within a tomcat context is nothing new to you, you might want to skip this chapter and jump right into chapter 3 “Configuring the WPVS”.

In chapter 3 “Configuring the WPVS” the actual configuration of the deegree2 WPVS is described and illustrated. The configuration of WPVS is similar to that of any other deegree 2 web service and requires editing of different XML files. We will see that the WPVS is highly configurable which allows for a great flexibility on the one hand, but makes it harder for a novice to understand the different configuration possibilities. But no fear, if you understand the underlying mechanism it is fairly straight forward to configure a deegree2 WPVS.

The next chapter “Fine-tuning the WPVS”, will briefly describe the advanced options of configuring the deegree2 WPVS. If you are well known to the basics of configuration and are looking to get the best out of your hardware/data this chapter might be a good place to start as an introduction.

The last chapter 5 Different ways to install deegree2 WPVS into Tomcat, will deal with some additional (advanced) deploying techniques into your tomcat. If you don't have the demo and just want to install your own deegree2 WPVS this might be a good place to look.

In order to use the deegree2 WPVS you must install some extra (third party) libraries which are described in Appendix A “Third party libraries”. Because the WPVS will not run without those libraries, it is strongly recommended that you have a look at this appendix first.

VERY IMPORTANT: There are two very important issues you must resolve before you can use the deegree2 WPVS:

- You must install the java 3D api binary, which can be downloaded from <http://java.sun.com/products/java-media/3D/downloads/>
- Make sure your graphiccard driver / gpu supports hardware 3D rendering

Finally some complete deegree2 WPVS configurations are presented in the Appendices B “Complete deegree2 WPVS configuration“, C “Rendering configuration options“ and D “Material properties file“.

2 Get deegree WPVS demo up and running

This chapter will give you a step by step introduction on how to install the deegree2 WPVS demo into a web server / servlet engine.

The web services of deegree2 are realized as Java modules controlled by one central servlet (a “dispatcher”). This servlet (and the Java modules) has to be integrated into your web server/servlet engine, this process is often addressed to as “deploying”. Most of the common web servers support servlet technology, making it possible to install deegree2 on a variety of web servers.

Although deegree2 can be installed on different web servers we recommend using the Apache-Tomcat 5.5 Servlet-Engine, due to its widespread use and its status as an open-source product¹. Furthermore, all deegree2 development is done with Tomcat 5.5, therefore you can be quite sure that if you're using a tomcat 5.5 and your webapp (the deegree2 WPVS) doesn't work, the problem is likely to be a configuration error and not an incompatibility error.

If you're using another web server / servlet engine please check it's documentation on how to deploy web services on it.

2.1 Pre-requisites

It is assumed that all necessary third party libraries are installed on your system. If you've not installed those libraries yet, please take a look at Appendix A Third party libraries for a complete list.

It is also assumed that you have a jdk (version 5 or higher) installed on your system and you have some standard tools like a plain text/xml editor and an unzip program.

VERY IMPORTANT: There are two very important issues you must resolve before you can use the deegree2 WPVS:

- Install the java 3D api, which can be downloaded from <http://java.sun.com/products/java-media/3D/downloads/>
- Make sure your graphics drive / gpu supports hardware 3D rendering.

¹ Several tests have shown that deegree2 web services also run with Tomcat 4.x and 5.0.x.

2.2 Download

Visit <http://www.deegree.org> and download the WPVS zip file including a Web Perspective View Server package, a simple client and this documentation.

You can also checkout the latest deegree version from SVN (follow the instructions on the homepage). For more information on installing a WPVS from SVN please read chapter 5.3 "Installing the latest WPVS from SVN" (not recommended as this is the developing version).

2.3 Tomcat integration

First download and install a current release of Tomcat, you'll also need the JDK 5 for this, the directory you've installed your tomcat in is traditionally called **\$CATALINA_HOME**.

The next thing we have to do, is registering the deegree2 WPVS with your Tomcat, this means tomcat needs to be aware of the deegree's libraries (**\$WPVS_DIRECTORY**), the handler (servlet) and the **WPVS-configuration.xml** file. Tomcat offers different mechanisms to achieve this.

This chapter will first explain the necessary steps to install the deegree-WPVS demo, which means you possess a deegree-wpvs.war.

2.3.1 Easy installation into Apache Tomcat

The easiest way to register deegree web services with Tomcat is to copy the **deegree-wvps.war** into the 'webapps' directory located in \$catalina_home\$. Afterwards restart Tomcat and the application will install automatically. The .war file, which is nothing more than a .zip file will be unzipped into a new directory (named the same as the .war; in our case deegree-wpvs) under \$catalina_home\$/webapps. This directory will, in the following, be referenced as \$WPVS_home\$. Tomcat will be automatically pointed to the configuration files as it searches for a web.xml file in the WEB-INF directory.

2.3.2 Pointing Tomcat explicitly to application home directory

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

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

Afterwards Tomcat needs information about the root directory of the WPVS. 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-wpvs.xml, and fill it with the following information

```
<Context docBase="c:/deegree/webapps/deegree-wpvs" path="/deegree-wpvs">
</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-wpvs/>. 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>wms,wpvs,wfs,wcs</param-value>
      <description>list of supported services, e.g.: wfs,wms (comma
separated)</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_1_3_0_configuration.xml</param-value>
  </init-param>
  <!-- WPVS INITIALIZING PARAMETERS -->
  <init-param>
    <param-name>wpvs.handler</param-name>
```

```

    <param-value>org.deegree.enterprise.servlet.WPVSHandler</param-value>
  </init-param>
  <init-param>
    <param-name>wpvs.config</param-name>
    <param-value>WEB-INF/conf/wpvs/wpvs_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/wfs_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>wfs.config</param-name>
    <param-value>WEB-INF/conf/wcs/wcs_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.html</welcome-file>
</welcome-file-list>
</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 a 'wpvs', 'wms', 'wfs' and 'wcs' is defined to be available (other possible values at the moment are e.g.: sos or 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.WPVSHandler' 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.

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-wpvs.xml` and respectively the name of the .war which you deployed in the `$TOMCAT_HOME$/webapps` (in our example `deegree-wpvs`) you should be able to point to your WFS (OWS) via the following URL: `http://my.server.domain/deegree-wpvs/services?`

2.4 Testing the installation

If everything went fine, deegree WPVS should be running by now with the sample data sets. Try the following request in your web browser:

```
http://localhost:8080/deegree-  
wpvs/services?SERVICE=WPVS&VERSION=1.0.0&REQUEST=GetCapabilities
```

This should provide a valid XML Capabilities file (if your browser asks for an application to open the file, try your browser again or any text editor). If you have installed the WPVS-demo or checked out the deegree sources from the CVS you are able to do just a little more though, check out the following GetView request, it will show you a nice 3D-View of Salt-lake city with some buildings:

```
http://localhost:8080/deegree-  
wpvs/services?BACKGROUND=cirrus&ELEVATIONMODEL=saltlakedem&CRS=EPSG:26912&YAW=0&REQ  
UEST=GetView&OUTPUTFORMAT=image/png&DATETIME=2007-03-  
21T12:00:00&PITCH=20&DISTANCE=1550&VERSION=1.0.0&AOV=60&ROLL=0&POI=424994.4,4513359  
.9,1550&BOUNDINGBOX=424585.3,4512973.8,425403.5,4513746.0&SCALE=1.0&SPLITTER=BBOX&H  
EIGHT=600&DATASETS=Utah_Overview,satelite_images&STYLES=default&FARCLIPPINGPLANE=11  
000&WIDTH=800&EXCEPTIONFORMAT=INIMAGE
```

As this demo comes with a very simple example client, you can afterwards check, if this one works properly, too. Requesting this

`http://localhost:8080/deegree-wpvs`

url with your browser should start a page with a link to the client.

If all of this worked fine, your deegree WPVS and client is running and you can now add your own data. But before that, let's first have a brief look at the architecture of deegree WPVS.

3 Configuring the WPVS

Although the deegree2 WPVS comes with some standard data of Salt-Lake city, you probably want to be able to get a perspective view of your own data. This chapter will explain the first step configuration possibilities of the WPVS and how you can make it aware of your data. Before jumping into the details, an overview of the architecture of the WPVS will be given, to let you understand the the different elements of a deegree2 WPVS configuration file.

3.1 Architecture

The WPVS (like all deegree2 web services) has a central configuration file, in which you define all parameters for controlling the web service. This file is an xml file which, includes the parameters mentioned in the OGC WPVS 1.0.0 specification for the Capabilities document, such as datasets and elevation models, but also additional parameters that are specific to deegree, as, for example, descriptions of data sources, copyright image and server settings. Figure 3.2 (on page 15) will give you an overview of the interaction between the wpvs-configuration elements and the wpvs.

3.2 Configuration file

The configuration file (located at `$WPVS_home$/WEB-INF/conf/wpvs/wpvs_configuration.xml`) is, at the same time, a basis from which the WPVS Capabilities will be generated. Because of this, you'll find in the configuration two types of XML elements: Those bound to OGC namespaces and those to the deegree2 WPVS namespace (e.g. *deegreewpvs*).

In the following, some of the basic elements of the configuration file will be described. Appendix A includes the complete configuration document. A more detailed description of the other elements is done in Chapter 4 Fine-tuning the WPVS.

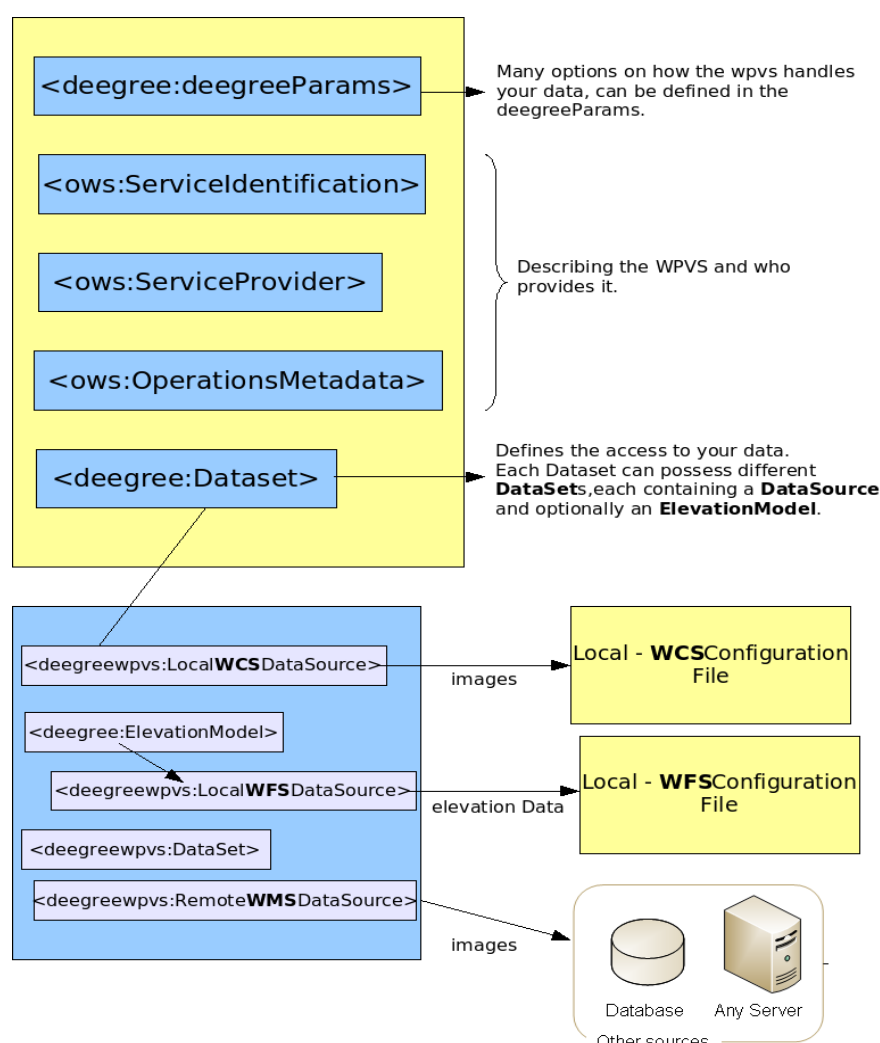


Figure 1: Schematic view of the Configuration of a Degree2 WPVS.

After the root element, some degree-WPVS-specific parameters are defined. These values can be used to fine tune your wpvs to the data you will be providing. The possible values are described in Chapter 4.1 Degree parameters.

3.3 Service Parameters

The following elements in the configuration file (`<ows:ServiceIdentification>`, `<ows:ServiceProvider>` and `<ows:OperationsMetadata>`) are used to describe the service metadata. The elements are adopted from the WPVS 1.0.0 specification (OGC Commons) and therefore not described in this document. Most of the names of these elements are self explaining, for a description and possible values of these elements please read the WPVS-Specification, or take a look at the configuration file in Appendix B Complete degree2 WPVS configuration

3.4 Datasets

After the deegree specific parameters and the dataproviders metadata elements the most important element is defined: `<deegreewpvs:Dataset>`. A Dataset can be understood as an encapsulation of the data which your service provides. There are two different kinds of `<deegreewpvs:Dataset>` elements in the deegree configuration:

- **Structural datasets**, define the layout of the WPVS
- **Data providing datasets**, define where and how data will be requested from different services.

Although a little confusing, there is a distinct difference in the two, the first kind of dataset is only accessible for the administrator of the WPVS to define the layout of the Service, whereas the other dataset is handed to requesting clients, so they can use it to request data, e.g. to insert in GetView – requests. In this chapter we will only discuss the second kind of datasets.

Each “data providing” `<deegreewpvs:Dataset>` must contain at least one `<deegreewpvs:DataSource>` (this can be understood as a link to a datasource). These datasources are only known to the administrator of the WPVS and will not be visible to any requesting clients.

The same way a `<deegreewpvs:Dataset>` can be understood as an encapsulation of data, a `<deegreewpvs:DataSource>` can be understood as a link to the location and the description of the capabilities of another webservice (e.g. a WCS, WMS or a WFS) which the WPVS will query to receive its data from. Such data can be images, height values or 3D-objects. How these datasources are defined and which kind of datasources are available in deegree is described further below.

Another very important Element inside a (data providing or structural) Dataset is the `<deegreewpvs:ElevationModel>` element. Like a dataset it encapsulates a datasource, with the difference, that a datasource underneath an `<deegreewpvs:ElevationModel>` always returns **elevationdata** (e.g. height values) in contrast to a datasource which lies directly underneath a (data providing) `<deegreewpvs:Dataset>`, which always will provide **geographic data** in form of images or 3D-Objects.

More specific, *the deegree 2 WPVS can display elevation data coming from vector sources (WFS) and or raster sources (WCS) as well as textural raster data (from a WMS or WCS), and three-dimensional object data (e.g. buildings) (from a WFS)*

Some elements in the `<deegreewpvs:Dataset>` are not specifically defined by the WPVS 1.0.0 specification. This is true for the `<deegreewpvs:DataSource>` elements and the `<deegreewpvs:ElevationModel>`

To sum up, (also compare figure 3.2) data providing Datasets can encapsulate following data:

- Own data (referenced over a datasource):
 - a) Texture from a WMS and/or WCS, for texturing the TIN
 - b) Three-dimensional objects from a WFS
- Elevation data (referenced over an elevationmodel):
 - a) As Point, Line, or Polygon (all 3D) from a WFS
 - b) A raster (GeoTiff only)

A Dataset may or may not contain other (nested) Datasets. A typical Dataset has some optional and some mandatory elements. The following xml fragment shows all mandatory elements inside a `<deegreewpvs:Dataset>`:

```
<deegreewpvs:Dataset>
  <deegreewpvs:Name>satellite_images</deegreewpvs:Name>
  <deegreewpvs:Title>Metropolis Satellite photos</deegreewpvs:Title>
  <deegreewpvs:Format>image/tiff</deegreewpvs:Format>
  <ows:WGS84BoundingBox crs="urn:ogc:def:crs:OGC:2:84" dimensions="2">
    <ows:LowerCorner>9.99 53.55</ows:LowerCorner>
    <ows:UpperCorner>10.03 53.569</ows:UpperCorner>
  </ows:WGS84BoundingBox>
  <deegreewpvs:Identifier>second_dataset_identifier</deegreewpvs:Identifier>
  <deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominator>
  <deegreewpvs:MaximumScaleDenominator>7000</deegreewpvs:MaximumScaleDenominator>
```

This could be a top (structural) Dataset, under which all other Datasets with actual data are defined. To point out the difference between a structural and an data providing Dataset, the same Dataset is displayed with an added datasource (`<deegreewpvs:LocalWCSDatasource>`) which gets it's data from a local wcs:

```
<deegreewpvs:Dataset>
  <deegreewpvs:Name>satellite_images</deegreewpvs:Name>
  <deegreewpvs:Title>Metropolis Satellite photos</deegreewpvs:Title>
  <deegreewpvs:Format>image/tiff</deegreewpvs:Format>
```

```

<ows:WGS84BoundingBox crs="urn:ogc:def:crs:OGC:2:84" dimensions="2">
  <ows:LowerCorner>9.99 53.55</ows:LowerCorner>
  <ows:UpperCorner>10.03 53.569</ows:UpperCorner>
</ows:WGS84BoundingBox>
<deegreewpvs:Identifier>second_dataset_identifier</deegreewpvs:Identifier>
<deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominator>
<deegreewpvs:MaximumScaleDenominator>7000</deegreewpvs:MaximumScaleDenominator>
<!-- Defining a datasource which will return raster data -->
<deegreewpvs:LocalWCSDataSource>
  <deegreewpvs:Name>Metropolis_photos</deegreewpvs:Name>
  <deegreewpvs:OWSCapabilities>
    <deegreewpvs:OnlineResource xlink:type="simple" xlink:href="./wcs/wcs_configuration.xml"/>
  </deegreewpvs:OWSCapabilities>
  <deegreewpvs:MinimumScaleDenominator>
    0.1
  </deegreewpvs:MinimumScaleDenominator>
  <deegreewpvs:MaximumScaleDenominator>
    1000000
  </deegreewpvs:MaximumScaleDenominator>
  <deegreewpvs:FilterCondition>
    <deegreewpvs:WCSRequest>
      <![CDATA[VERSION=1.0.0&coverage=luftbild&TRANSPARENT=TRUE&FORMAT=tif&EXCEPTIONS=application/vnd.ogc.se_xml&crs=EPSG:31467]]>
    </deegreewpvs:WCSRequest>
  </deegreewpvs:FilterCondition>
</deegreewpvs:LocalWCSDataSource>
</deegreewpvs:Dataset>

```

It is often useful to create a new `<deegreewpvs:Dataset>` with each new `<deegreewpvs:DataSource>`, this way a client can make an accurate decision of which datasets are to be requested. Consider the following example: a WPVS only provides one dataset containing three different datasources, a) a `WFSDDataSource` to display buildings 2) a `WMSDataSource` to show orthographic photos and 3) a `WMSDataSource` to show topographic maps of an area. Now a client can only request this one dataset and must always see the building the maps and the orthographic images although he might only be interested in the maps. Therefore it would be better to create three (nested) datasets each one containing it's own datasource (that is, a `WFSDDataSource`, a `WMSDataSource` and another `WMSDataSource`), this way the client can select the one it wants to see. In the following section we described the different types of `<deegreewpvs:DataSource>` and their uses.

Inside the `<deegreewpvs:Dataset>` element the following metadata elements are mandatory:

- `<deegreewpvs:Name>`: The name by which a dataset shall be requested, it is good practice to choose unique names.
- `<deegreewpvs:Title>`: The title to be represented to a client to pick from
- `<deegreewpvs:Format>`: Mimetype describing the returned object of this dataset
- `<ows:WGS84BoundingBox crs="urn:ogc:def:crs:OGC:2:84" dimensions="2">`: The boundingbox in wgs84 for which this dataset is valid, it must contain following two child elements:
 - `<ows:LowerCorner>`: the wgs84 coordinates of the lowerleft corner of the boundingbox

- `<ows:UpperCorner>`: the wgs84 coordinates of the upperright corner of the boundingbox
- `<deegreewpvs:Identifier>`: This identifier is specified by the WPVSreference, it should be unique in the total configuration. In the future clients may address this identifier to request the associated Dataset.
- `<deegreewpvs:MinimumScaleDenominator>`: Defining the minimum scale of the data for which this dataset is valid.
- `<deegreewpvs:MaximumScaleDenominator>`: Defining the maximum scale of the data for which this dataset is valid.

Except for these mandatory elements some optional child elements can be defined inside the `<deegreewpvs:Dataset>` element too. For the exact definitions and their possible values take a look at the complete configuration file displayed in Appendix B Complete deegree2 WPVS configuration.

3.5 DataSources

As mentioned above each data providing `<deegreewpvs:Dataset>` Element must also contain at least one `<deegreewpvs:DataSource>`. Although this is the most common use of DataSources, they are also found in `<deegreewpvs:ElevationModel>` elements, see Chapter 3.6 Elevation Models for an introduction of elevationmodels.

A `<deegreewpvs:DataSource>` can be understood as a link to the location and the description of the capabilities of another web service (e.g. a WCS, WMS or a WFS) which the WPVS will query to receive data. Datasources are only used internally by the WPVS and are never presented to requesting clients. Deegree2 can handle two different classes of datasources:

- **Local** DataSources: which can be configured to access your data via the deegree Framework. This means you don't have to set up a webservice inside a tomcat, but only need to supply a configuration file (or more exact an url embedded inside an `<deegreewpvs:OnlineResource>` element which points to the (local) configuration file), which describes the capabilities of your Local DataSource. Three different local datasources can be defined:
 - Local**WMS**DataSource, providing raster data (as images), which can only be used as textures².

²e.g. some image on a 3D-Object, usually the 3D terrain representation of the requested area.

- Local**WCS**DataSource, providing raster data (as images), which can be used as height-values or textures.
- Local**WFS**DataSource, providing vector data embedded in xml. This data can be used as height-values or as 3D-Objects like buildings (e.g. city-GML)
- **Remote** DataSources: provides (among other things) a link to another Web Service (running inside a servlet engine) which can be contacted to receive data. Don't let the *remote* keyword confuse you, it only means the service has to be contacted via the http-protocol, but the service might as well run on your local machine. Deegree2 WPVS only knows two different remote datasources:
 - Remote**WMS**DataSource, providing raster data (as images), which can only be used as textures.
 - Remote**WFS**DataSource, providing vector data embedded in xml. This data can be used as height-values or as 3D-Objects like buildings (e.g. city-GML)

All `<deegreewpvs:DataSource>` elements have some (mandatory) common elements, which are exemplified in the following xmlfragment. This datasource defines a local datasource using data from a WCS:

```
<deegreewpvs:LocalWCSDataSource>
  <deegreewpvs:Name>Metropolis_photos</deegreewpvs:Name>
  <deegreewpvs:OWSCapabilities>
    <deegreewpvs:OnlineResource xlink:type="simple" xlink:href="./wcs/wcs_configuration.xml"/>
  </deegreewpvs:OWSCapabilities>
  <deegreewpvs:MinimumScaleDenominator>
    0.1
  </deegreewpvs:MinimumScaleDenominator>
  <deegreewpvs:MaximumScaleDenominator>
    1000000
  </deegreewpvs:MaximumScaleDenominator>
  <deegreewpvs:FilterCondition>
    <!--datasource specific-->
  </deegreewpvs:FilterCondition>
</deegreewpvs:LocalWCSDataSource>
```

A `<deegreewpvs:Name>` element defines an identifier for the given DataSource, although this name is only known to the administrator (and is not provided to a client) it is good practice to make this name unique.

The `<deegreewpvs:OWSCapabilities>` element defines the link to the configuration document of the DataSource. For `<deegreewpvs:Local***DataSource>` this is a (relative or absolute) link pointing to a configuration file on the local filesystem. for

`<deegreewpvs:Remote***DataSource>` this is a link to a GetCapabilities-request on a Webserver, this can be a webservice running on your local machine (<http://localhost/>) or any other server. In the example above, the configuration can be found in the same file system, and is referenced as a relative path. For a remote DataSource you'd put the HTTP request pointing to the GetCapabilities request (resulting in an xml capabilities file) of the service in question.

The `</deegreewpvs:MinimumScaleDenominator>` and the `<deegreewpvs:MaximumScaleDenominator>` elements are equal to the elements of the `<deegreewpvs:Dataset>`. Resolution ranges may overlap or have gaps. They also need not match that of the parent Dataset. Note here the difference between those: The resolution range for a Dataset is an informative element, which is going to be shown in the capabilities of the WPVS. On the other hand, a resolution range of a data source is a configuration parameter, which only the service administrator has access to.

The `<deegreewpvs:FilterCondition>` element can define different child element. For a DataSource which handles raster data (a WMS or a WCS) this element defines parts of a wcs/wms request, for a DataSource handling vector data (WFS) this element defines a `<wfs:query>`. In the following the possible DataSources are shortly explained.

3.5.1 A WMS DataSource

A typical WMS DataSource is shown below:

```
<deegreewpvs:LocalWMSDataSource>
  <deegreewpvs:Name>simple_wms</deegreewpvs:Name>
  <deegreewpvs:OWSCapabilities>
    <deegreewpvs:OnlineResource xlink:type="simple"
                                xlink:href="../../wms/wms_configuration.xml" />
  </deegreewpvs:OWSCapabilities>
  <deegreewpvs:MinimumScaleDenominator>
    0
  </deegreewpvs:MinimumScaleDenominator>
  <deegreewpvs:MaximumScaleDenominator>
    5000000
  </deegreewpvs:MaximumScaleDenominator>
  <deegreewpvs:FilterCondition>
    <deegreewpvs:WMSRequest>
      <![CDATA[VERSION=1.1.1&LAYERS=LandUse,Roads&TRANSPARENT=true&FORMAT=image/png&EXCEPTIONS
=application/vnd.ogc.se_inimage&BGCOLOR=0xffffffff]]>
    </deegreewpvs:WMSRequest>
  </deegreewpvs:FilterCondition>
  <deegreewpvs:TransparentColor>
    <deegreewpvs:Color>#ffffff</deegreewpvs:Color>
  </deegreewpvs:TransparentColor>
</deegreewpvs:LocalWMSDataSource>
```

Note that the Capabilities (OnlineResource element of OWSCapabilities) point to a file in the same file system. Note that this is, strictly speaking, not really the WMS

capabilities, but the deegree WMS configuration. (You know from reading the deegree WMS documentation, that a WMS configuration looks very much like the WMS capabilities itself.)

The `<deegreewpvs:FilterCondition>` element includes a partial WMS request (defined inside the `<![CDATA[]]>` element) that will be used for the GetMap request. In the example above we are asking the GetMap to draw the layers 'LandUse' and 'Roads'. These layers must be defined by the WMS and it's your task to make sure the request is valid. (The WPVS will fill in the blanks, by adding missing parameters, e.g. width, height, bounding box, etc.)

Note the `<deegreewpvs:TransparentColors>` element, which define which colors in the GetMap image will be set to transparent. In the example above, all black pixels will be made fully transparent. This allows you to filter out pixels, making the texture associated with this service partially transparent.

A RemoteWMSDataSource looks just like a local one. The only difference is in the definition of the Capabilities:

```
<deegreewpvs:OWSCapabilities>
  <deegreewpvs:OnlineResource xlink:type="simple"
    xlink:href="http://localhost:8080/deegree-wms/services?
      service=WMS&request=GetCapabilities&version=1.1.1" />
</deegreewpvs:OWSCapabilities>
```

This now points to the Capabilities document of the given webservice (localhost:8080) which is defined inside the href attribute. Note the different use of the ampersands inside the `<![CDATA[]]>` element and the `<deegreewpvs:OnlineResource>`.

3.5.2 A WCS DataSource

A WCS DataSource is almost every aspect identical to a WMS DataSource. The difference lies in the WCSRequest element of FilterCondition. This is obviously a fragment of a WCS request.

```
<deegreewpvs:LocalWCSDataSource>
  <deegreewpvs:Name>simple_wms</deegreewpvs:Name>
  <deegreewpvs:OWSCapabilities>
    <deegreewpvs:OnlineResource xlink:type="simple"
      xlink:href=" ../wms/config.xml" />
  </deegreewpvs:OWSCapabilities>
  <deegreewpvs:MinimumScaleDenominator>
    0
  </deegreewpvs:MinimumScaleDenominator>
  <deegreewpvs:MaximumScaleDenominator>
    5000000
  </deegreewpvs:MaximumScaleDenominator>
  <deegreewpvs:FilterCondition>
    <deegreewpvs:WCSRequest>
      <![CDATA[VERSION=1.0.0&coverage=city_layout&TRANSPARENT=TRUE&
```

```

        FORMAT=png&EXCEPTIONS=application/vnd.ogc.se_xml&crs=EPSG:31466]]>
    </deegreewpvs:WCSRequest>
</deegreewpvs:FilterCondition>
<deegreewpvs:TransparentColor>
    <deegreewpvs:Color>#ffffff</deegreewpvs:Color>
</deegreewpvs:TransparentColor>
</deegreewpvs:LocalWMSDataSource>

```

3.5.3 A WFS DataSource

A WFSDataSource has great similarities with the above mentioned Datasources, there are however some differences. They can be explained by the nature of the data the different services return. It might therefore be useful to shortly describe the response of a WFS-GetFeature request.

The first difference between the services is, that the WFS returns vector data, while the other two services (WMS and WCS) return raster data. The result of a GetFeature request is a FeatureCollection in which different features are embedded. The common representation for features is XML. Inside these Features, GeometryProperties contain the “raw points” as geometry elements. An example for a feature, could be a building or a 3D label, but also height values as measure points of a terrain.

For the WPVS to use a WFSDataSource, it needs to tell the latter, the name of the GeometryProperties it wants to visualize. This can be done by defining a new element `<deegreewpvs:GeometryProperty>`. It contains an `xpath` expression which is used to select the GeometryProperty of the Features. This expression should be a qualified name, it's therefore up to you to make sure, that the prefix is bound to a namespace. The following XML-fragment is an example of a local WFS DataSource. In this example the `xpath` expression is `app:geometry`, with `app:` bound to the namespace `http://www.deegree.org/app` in the root element of the xml-document with the following assignment `xmlns:app="http://www.deegree.org/app"`.

```

<deegreewpvs:LocalWFSDataSource>
  <deegreewpvs:Name>buildings of metropolis</deegreewpvs:Name>
  <deegreewpvs:OWSCapabilities>
    <deegreewpvs:OnlineResource xlink:href="./wfs/wfs_configuration.xml" />
  </deegreewpvs:OWSCapabilities>
  <deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominator>
  <deegreewpvs:MaximumScaleDenominator>99992</deegreewpvs:MaximumScaleDenominator>
  <deegreewpvs:MaxFeatures>150</deegreewpvs:MaxFeatures>
  <deegreewpvs:GeometryProperty>app:geometry</deegreewpvs:GeometryProperty>
  <deegreewpvs:ValidArea>
    <gml:Polygon srsName="EPSG:31466">
      <gml:outerBoundaryIs>
        <gml:LinearRing>
          <gml:coordinates cs="," decimal="." ts=" " >
            3565687.9,5935846.7
            3568308.4,5935846.7
            3568308.4,5937998.9
            3565687.9,5937998.9
          </gml:coordinates>
        </gml:LinearRing>
      </gml:outerBoundaryIs>
    </gml:Polygon>
  </deegreewpvs:ValidArea>
</deegreewpvs:LocalWFSDataSource>

```

```

        3565687.9,5935846.7
      </gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
</gml:Polygon>
</deegreewpvs:ValidArea>
</deegreewpvs:LocalWFSDataSource>

```

Note the similarities to other data sources: name, capabilities, scale range. The differences are explained below. Also note that a RemoteWFSDataSource is in every aspect similar to a local one.

Apart from the `<deegreewpvs:GeometryProperty>` element, the given LocalWFSDataSource example also introduces a new optional element which can be inserted into all DataSources (that is: remote and local, WMS, WCS and WFS) it is called: `<deegreewpvs:ValidArea>`. With this element you have the possibility of defining in which area the WPVS might find the data of the given DataSource. This prevents the WPVS from sending requests to the DataSource, if the ValidArea does not intersect with the visible area. Inside this element three different definitions are valid, either a `<gml:Polygon>` (displayed in this example), `<gml:Surface>` or `<ows:BoundingBox>`. For the exact definitions of these elements please take a look at the gml definitions and/or the ows definition.

There is yet one more element `<deegreewpvs:MaxFeatures>`, which cannot be added to the other two DataSources. It is a common use case, that you have different wfs-datasources, but you might not want to receive maxFeatures (defined in the wfs-configuration.xml) from each and every one of them. This element will allow you to configure the number of features for each of your wfs-datasources.

Although not displayed here, a `<deegreewpvs:FilterCondition>` element may be defined into a WFSDataSource too³. If it is defined, it is the skeleton of a WFS GetFeature query. The most important attributes here are 'typeName', which is also found in GetFeature requests, the srsName, which must match that of your data, and the namespace (xmlns:app) of your FeatureType. For more information on how to write `<wfs:Query>` take a look at your wfs documentation. For completion an example is given below:

```

<deegreewpvs:FilterCondition>
  <wfs:Query handle="String"
    typeName=""
    featureVersion="String"
    srsName="EPSG:31466"
    xmlns:app="http://www.deegree.org/app">
    <wfs:PropertyName>app:geometry</wfs:PropertyName>
    <ogc:Filter>
      <ogc:FeatureId fid="some_id" />
    </ogc:Filter>
  </wfs:Query>
</deegreewpvs:FilterCondition>

```

³In a WFSDataSource the `<deegreewpvs:FilterCondition>` element is optional and not mandatory as it is in the WMS and WCSDatasources.

```

    </ogc:Filter>
  </wfs:Query>
</deegreewpvs:FilterCondition>

```

3.6 Elevation Models

Elevation models can be understood as a representation of the height values of a given area. As stated above the WPVS can handle two different representation of these height values, raster data and vector data (in form of measure points). To handle this data another element should be defined beneath a `<deegreewpvs:DataSet>` element in the configurationfile it is called: `<deegreewpvs:ElevationModel>`. It is comparable to a data providing `<deegreewpvs:DataSet>` element, it too encapsulates the data (which is used to create the terrain), it has a name, and must have at least one Datasource.

An elevation model element may have any number of data sources. However, the only valid types of datasources for elevation models are WFS and WCS. For a WCS data source, the returned image type must be of mimetype GeoTiff. Apart from this, datasource elements are not different to those described above. We now turn our attention to describing the two different elevationmodels with the different types of data sources.

3.6.1 An example: WCS-based Elevation Model

Although we say here “WCS-based” elevation model, no restrictions apply as to what type of data sources combination you may have in an elevation model. (Remember: WMS DataSources are not valid sources for elevation models.) You may combine a WFS and a WFC DataSource for, say, providing a coarse elevation model based on WFS points, and a high-resolution one based on a WCS. In practice, however, you'd only have either a WFS- or a WCS-based elevation model.

```

<deegreewpvs:ElevationModel>
  <deegreewpvs:Name>dgm</deegreewpvs:Name>
  <deegreewpvs:LocalWCSDataSource>
    <deegreewpvs:Name>DEM-Model of Metropolis</deegreewpvs:Name>
    <deegreewpvs:OWSCapabilities>
      <deegreewpvs:OnlineResource xlink:type="simple" xlink:href="./wcs/config.xml"/>
    </deegreewpvs:OWSCapabilities>
    <deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominator>
    <deegreewpvs:MaximumScaleDenominator>1000</deegreewpvs:MaximumScaleDenominator>
    <deegreewpvs:FilterCondition>
      <deegreewpvs:WCSRequest>
        <![CDATA[VERSION=1.0.0&coverage=dgm&TRANSPARENT=TRUE&FORMAT=GeoTiff&EXCEPTIONS=application/vnd.ogc.se_xml&crs=EPSG:31467&response_CRS=EPSG:31467]]>
      </deegreewpvs:WCSRequest>
    </deegreewpvs:FilterCondition>
    <deegreewpvs:TransparentColor>
      <deegreewpvs:Color>#000000</deegreewpvs:Color>
    </deegreewpvs:TransparentColor>
  </deegreewpvs:LocalWCSDataSource>
</deegreewpvs:ElevationModel>

```

```
</deegreewpvs:TransparentColor>  
</deegreewpvs:LocalWCSDatasource>  
</deegreewpvs:ElevationModel>
```

There are no real surprises here. There are two things to bare in mind with elevationmodels, first the value of the `<deegreewpvs:Name>` element should be unique under all defined elevationmodels and secondly the returned image type of the WCS Datasource must be GeoTiff.

You now should be able to understand most of the configuration in the example file given in Appendix B Complete deegree2 WPVS configuration, and hopefully add your own DataSets to it. There are however a lot more options to configure, with which the behavior of the deegree2 WPVS is adopted to your specific needs. These will be discussed in the next chapter.

4 Fine-tuning the WPVS

We have so far described how you can configure your WPVS to show a perspective image in response to a GetView-request. The result might be quite satisfactory, but under most circumstances not good enough for you. The degree2 WPVS has a way to fine tune your service appropriate to the data you want to present to a client. With the degree2 WPVS it is possible for you to fine tune some settings according to your data and hardware. These possibilities are discussed in the following subsection.

Although we've said, all configuration is done in only one xml file, this isn't quite true. There are some other files (*java property files*), in which you can define the the rendering behavior of the java3D engine, as well as setting the colors of vector data geometries coming from a WFS. Before we take a look at those, we first describe the degree parameters in the normal xml configuration file.

4.1 Degree parameters

The degree parameters element contains only child elements, which are specific for the degree2 WPVS. The majority of these elements supply the WPVS with some kind of default value if the particular value isn't known or given. The responsible element in the configuration file is located directly under the root element and is called: `<degreewpvs:degreeParams>`. An example of a typical layout of this element is shown in the following xml code fragment:

```
<degreewpvs:degreeParams>
  <degreewpvs:DefaultOnlineResource
    xlink:href="http://localhost:8080/degree-wpvs/services"/>
  <degreewpvs:CacheSize>100</degreewpvs:CacheSize>
  <degreewpvs:RequestTimeLimit>60</degreewpvs:RequestTimeLimit>
  <degreewpvs:ViewQuality>0.95</degreewpvs:ViewQuality>
  <degreewpvs:MaxViewWidth>1200</degreewpvs:MaxViewWidth>
  <degreewpvs:MaxViewHeight>1000</degreewpvs:MaxViewHeight>
  <degreewpvs:DefaultSplitter>QUAD</degreewpvs:DefaultSplitter>
  <degreewpvs:RequestQualityPreferred>
    false
  </degreewpvs:RequestQualityPreferred>
  <degreewpvs:MaxTextureDimension>8029</degreewpvs:MaxTextureDimension>
  <degreewpvs:QuadMergeCount>10</degreewpvs:QuadMergeCount>
  <degreewpvs:Copyright>
    <!-- Text>Copyright 2006 lat/lon GmbH </Text -->
    <degreewpvs:ImageURL xlink:href=" ../images/degree_alpha_unaliased.png"
      watermark="false" />
  </degreewpvs:Copyright>
  <degreewpvs:RequestsMaximumFarClippingPlane>
    15000
  </degreewpvs:RequestsMaximumFarClippingPlane>
  <degreewpvs:NearClippingPlane>2</degreewpvs:NearClippingPlane>
  <degreewpvs:MinimalTerrainHeight>55</degreewpvs:MinimalTerrainHeight>
  <degreewpvs:MinimalWCSElevationModelResolution>
    25
  </degreewpvs:MinimalWCSElevationModelResolution>
  <degreewpvs:ExtendRequestPercentage>1</degreewpvs:ExtendRequestPercentage>
```

```
<deegreewpvs:BackgroundList>
  <deegreewpvs:Background name="cloudy" href="images/background/cloudy.jpg" />
  <deegreewpvs:Background name="sunset" href="images/background/sunset.jpg" />
</deegreewpvs:BackgroundList>
</deegreewpvs:degreeParams>
```

The `<deegreewpvs:DefaultOnlineResource/>` is the URL by which the WPVS operations can be invoked. This parameter can be overwritten by the URLs defined in the request-definitions of the datasources (explained in the previous chapter).

The `<deegreewpvs:CacheSize>` parameter defines the size of the cache available to deegree-WPVS in megabyte (this does not affect the cache for data sources). This parameter is optional, its default value is 100 MB. (As of 2006-04-04, the cache is not activated.)

By `<deegreewpvs:RequestTimeLimit>` the maximum time span is defined after which a request has to be processed. If this value is exceeded the processing is canceled and an exception will be thrown. Its default value is 15 seconds.

Afterwards a `<deegreewpvs:ViewQuality>` parameter is defined, by which the quality of the created view can be controlled. Values for this parameter are between 0 (lowest quality) and 1 (best quality) with the default value being 0.95. This parameter is only used for image formats supporting different quality values (e.g. jpeg).

The following two parameters limit the maximum size of the response image that can be requested via a GetView request. It has to be considered that requesting large maps is very taxing on a server as the processing cost increases with the square of the border length of a map. For example, the creation a map having 1000x1000 pixel takes four times the memory than creating a map with size 500x500 pixel. Default is 1000 pixel for width and height.

The `<deegreewpvs:DefaultSplitter>` can be used to define a default splitter mode if no splitter was given inside a GetView request. A splitter is used to partially cut up the view frustum (which can be understood as the viewable area) into different resolutionstrips. This splitting is useful to create smaller individual request to the configured and requested datasets/elevationmodels and therefore acquiring more detailed images. The possible values for this element are QUAD or BBOX. A BBOX splitter doesn't actually cut up the viewable area, instead the bounding box of the requested area is sent to the datasets and elevationmodels, resulting in a poorly detailed response image. The default value for this element is QUAD.

Relevant to the `<deegreewpvs:DefaultSplitter>` element, is the following `<deegreewpvs:RequestQualityPreferred>` element. If the defaultsplitter is set

to `QUAD`, the value of this element decides the number of (resolution) stripes that are created. A value of `true`, results in a very detailed response image, but a (much) longer processing time. Setting this value to `false` however, will result in a lesser quality response image but a faster processing time. If no value is given the default is `false`.

In some cases it can be very useful to set the size of a texture. For example if a specific server cannot respond images larger then 1024 pixels. The `<deegreewpvs:MaxTextureDimension>` element defines this request size. It is however (due to the nature of the wpvs) only possible to set one value for all `DataSources`. If this value is omitted or set higher then the size of the maximum texture size your `gpu`⁴ can handle, the latter size will be used.

Although the quattree-splitter is pretty good in calculatin the request envelopes. After it calculated these bboxes, it tries to merge them so fewer requests have to be made, hence resulting in a quicker processing of the data. Tests have proven however, that in some rare cases, it is useful not to merge quads. For this purpose the `<deegreewpvs:QuadMergeCount>` element can be used. We advise to set the value relatively low, it defaults to 10.

With the `<deegreewpvs:Copyright>` parameter it is possible to write a copyright remark in the lower left corner of each image. A copyright note can be either a text fragment or a (absolute or relative) reference to an image (e.g. `c:/images/copyright.gif`). For better effect, the image should be set to half-transparent. In case no Copyright parameter is supplied no output is produced.

The `<deegreewpvs:RequestsMaximumFarClippingPlane>` element gives you the opportunity to decide how far a requesting client can see. For very steep request, the maximum distance a `GetView` request should be able to see will lie somewhere in infinity, resulting in an immense amount of data to be requested. This parameter eliminates this behaviour, but be careful; setting a too small value will result in poor response images.

The `<deegreewpvs:NearClippingPlane>` element gives you the opportunity to decide how near a requesting client can see. If you set this value to high, the client will not see the near perimeter, if you set it to low e.g. much smaller then `maxFarClippingPlane*0.001` some strange stripes in your images might be appear. This value defaults to 2.

⁴gpu: Graphics Processing Unit, the processor on the 3d-video-board in your computer.

For some data it is not necessary to have an elevation model, or the datasource defined in the elevationmodel doesn't always respond properly. In those cases it is useful to be able to define a minimal height value, for which you know the area you're presenting will never go beneath. This value can be defined in the `<deegreewpvs:MinimalTerrainHeight>`.

The next element `<deegreewpvs:MinimalWCSElevationModelResolution>` defines a way to set the minimal resolution a wcs based ElevationModel can handle. This is not to be confused with the `<deegreewpvs:MaximumScaleDenominator>` which defines a scale the datasource is valid for. In some cases it is more convenient to be able to define the minimal resolution of a wcs-request, resulting in smaller requests. This is particularly handy if the elevation model (build on a wcs-response) shows "rice-field" tableaux; setting this value to the appropriate resolution will resolve this problem.

With a WCS-DataSource as Elevation model, it can happen in a terrain with lots of mountains, that the requests don't perfectly overlap. Resulting in voids in the terrain. This phenomenon can be reduced by requesting a little larger Area as strictly been calculated. The `<deegreewpvs:ExtendRequestPercentage>` element lets you decide the percentage of extension for each datasource-request. A good value lies somewhere around '3'. Be careful though, if you are using a WFS-DataSource for the terrain don't use this option.

The last parameter you can set is a list of background images which can be presented to the client. The `<deegreewpvs:Background>` elements inside the `<deegreewpvs:BackgroundList>` element define the locations of images, which can be rendered in the background of the scene.

4.2 Rendering options

All the options discussed in the previous chapter are very useful to define a certain behaviour of the deegree2 WPVS as a service. But so far nothing can be configured for the actual rendering process. To accommodate you to get the most beautiful images out of your hardware, another file can be edited. This file is a *java properties* file with different key value pairs defined inside it. You can find a copy of default used values in Appendix C Rendering configuration options.

If you want to change any of these values, just put a copy of this file into the root directory of your webapp with the name `rendering_options.properties`. The WPVS will first try to read this location and (if nothing was found) will use the one inside the deegree package. While a lot of combinations are possible and quite some

background knowledge is necessary to understand the values in this file, we will not try to describe them, but leave it to the interested reader to do some self investigation. The key value pairs are all very well documented though and they can be used to get first impression what the different parameters are able to do. So if you're not content with the result of your response image (e.g. it's too dark, bright etc.) take a look in this file and try adjusting any or some of these parameters.

4.3 Material properties

There is one final possibility to adjust the outcome of the rendered GetView-response image. Again it is a `java.properties` file, but it now only is valid for the 3D-Objects coming from a WFS-DataSource underneath a dataset element. The file is called `material.properties` and can be used to set the visual behaviour of the geometries which are created out of a GetFeature response from a WFS. Once again a complete file is given in Appendix D Material properties file, and if you want to adjust these values you can (again) place a copy (with the adjusted values) in the root directory of your web application with the name `material.properties`. The values in this file will only be used if the CityGML Geometry (which is probably the Geometry inside the Features) has no value defined for parameter.

Most of the parameters in this property file take a three color attributes in `r g b` (red, green, blue), which means you can put in values between 0 and 1, each value separated with a blank space in between. So for example if you want to make all the surfaces of your 3D-Objects that do not supply a `diffusecolor` attribute themselves, will show up green you would adjust the `diffusecolor` value inside the properties file as follows:

```
diffusecolor=0 1 0
```

5 Different ways to install deegree2 WPVS into Tomcat

In this chapter some additional ways (other than using the manager) are described to install your deegree2 WPVS into a tomcat servlet engine.

5.1 Manually installing the demo in Tomcat

If you want to install the deegree2 WPVS demo into Tomcat manually, the first thing you have to do is to tell Tomcat about the root directory of your WPVS (and other deegree web services if installed in the same server context). This has to be done by first unzipping the deegree-wpvs.war file to a location (e.g. a directory called deegree-wpvs) of your choice and afterwards creating an additional .xml file (for example deegree-wpvs.xml) in **\$CATALINA_HOME/conf/Catalina/localhost/** with the following `<Context>` tag in it (remember **\$CATALINA_HOME** is the install directory of your Apache Tomcat):

```
<Context path="/deegree-wpvs"
        docBase="$WPVS_DIRECTORY/deegree-wpvs"
        crossContext="false"
        debug="0" reloadable="false"/>
```

The `path` attribute describes the virtual location (the *context path*) of the deegree web service. In the example, the service is accessible at <http://localhost:8080/deegree-wpvs/>. The `docBase` attribute reflects the physical location of the deegree service in the file system. The other attributes and their values are essential for good performance.

5.2 Installing a single deegree.jar

As stated before there are several ways to install a web application into the Tomcat servlet engine, if you just possess the deegree.jar some work is coming up to you. Let us first take a look at the Configuration file you are going to have to create, and after that take a look at the two deploy possibilities of the Tomcat manager and the manual deployment.

5.2.1 Tomcat configuration files

If you want to build a web application from scratch, the first thing you have to do, is to create a directory (let's call it **\$DEEGREE_WEBAPP**) in which your web application will reside. In this directory Tomcat will look for a (mandatory) subdirectory **WEB-INF/** (in capital letters – even on a Windows system), so we are going to have to create this directory too. In **WEB-INF** a *Deployment-Descriptor* with the filename **web.xml** is located. This file is analyzed 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. The description of all allowed values is out of the scope of this document, but the (for the degree-WPVS) necessary values are listed in the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <servlet>
    <servlet-name>services</servlet-name>
    <servlet-class>
      org.degree.enterprise.servlet.OGCServletController
    </servlet-class>
    <init-param>
      <param-name>services</param-name>
      <param-value>wpvs</param-value>
    </init-param>
    <init-param>
      <param-name>WPVS.handler</param-name>
      <param-value>org.degree.enterprise.servlet.WPVSHandler</param-value>
    </init-param>
    <init-param>
      <param-name>WPVS.config</param-name>
      <param-value>WEB-INF/conf/wpvs/wpvs_configuration.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>services</servlet-name>
    <url-pattern>/services</url-pattern>
  </servlet-mapping>
</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>` tag.

The servlet 'org.degree.enterprise.servlet.OGCServletController' is located inside the degree.jar.

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 WPVS is defined to be available (other possible values at the moment could be: WFS, WMS, WCS, SOS and CS-W).
- For each service listed in the 'service' `init-param` a handler class and a configuration file must be referenced.
- The `<param-name>` of the `<init-param>` for defining the handler starts with the service name (WPVS in the example) followed by '.handler'. The `<param-value>` of this parameter is the name of the handler class to be used. It is possible to

write different classes for this purpose and reference them accordingly. As a default 'org.deegree.enterprise.servlet.WPVSHandler' (which is available inside the deegree2.jar) could be used.

- The `<param-name>` of the `<init-param>` for defining the location of the WPVS-configuration file (which has nothing to do with the Tomcat configuration files) 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.
- The tag `<servlet-mapping>` defines the alias name for the servlet. It is not necessary for the `<servlet-name>` and `<url-pattern>` to be 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), in our example this will be `http://localhost:8080/deegree-wpvs/services?`

The next step is to create the directory for the WPVS-configuration, at exactly the location you've defined in the **web.xml** file. Because we've defined `<param-value>WEB-INF/conf/wpvs/wpvs_configuration.xml</param-value>` our WPVS configuration file must be located into the directory **\$DEEGREE_WEBAPP/WEB-INF/conf/wpvs/**. We won't worry about the layout of this file now, but we'll get back to it in Chapter 3 Configuring the WPVS.

The last directory we have to create is the directory where the Tomcat will look for the classes. This directory is called **lib**, and it also resides under WEB-INF. You should copy the deegree2.jar (and all the libraries it depends on) into this directory.

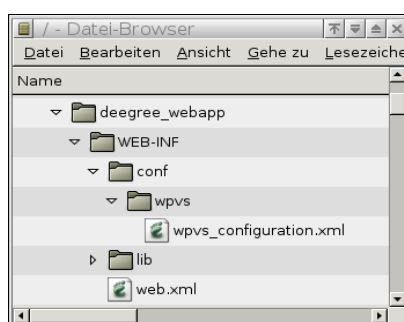


Abbildung 2: Directory tree of the deegree web application

This leaves you with following directorystructure:

You are now ready to deploy your deegree-WPVS to your Tomcat. Again you can use the two ways described above (see and Manually installing the demo in Tomcat),

with the only difference that you just specify your sole directory **\$DEEGREE_WEBAPP/** instead of the file **\$WPVS_DIRECTORY/deegree.war**.

5.3 Installing the latest WPVS from SVN

If you don't have deegree2 WPVS demo or a deegree.jar yet, there is a third way to deploy a deegree2 WPVS to your Tomcat; you can use the sources from the SVN repository. This documentation will not describe how to checkout deegree from SVN⁵ but solely the deploy mechanism available to you. This mechanism uses the ant tool, which can be downloaded from <http://ant.apache.org/>. In order for this tool to work with your tomcat installation, you are going to have to install the extra package apache-tomcat-5.5-deployer.

If you've checkout the current deegree sources, (let's say in the directory **\$DEEGREE_SRC**), there are two things you have to do, first of all edit the file **\$DEEGREE_SRC/webapps/deegree/WEB-INF/web.xml** so that it contains only those element tags needed for the WPVS (for an example and explanation see the chapter Tomcat configuration files).

After that you must open the file **\$DEEGREE_SRC/build.properties** with your favorite text/xml editor. In this file locate the following lines and adopt them to your system, (the values are described in the previous chapters). Only those lines you should edit are shown here, you should leave the others unchanged.

```
# Tomcat 5.5 home (needed for catalina-ant.jar)
tomcat.home=$CATALINA_HOME
tomcat.host=localhost
tomcat.port=8080
tomcat.username=tomcat
tomcat.password=tomcat
```

You are done editing files and ready to deploy your wpvs into your Tomcat. To do this, open a command line (in windows click "start->run" and type 'cmd.exe') and now change the working directory (with the cd command) to **\$DEEGREE_SRC**. If all is well you can enter the following command: "ant deploy", which will use your Tomcat manager to install the deegree2 WPVS into your web engine.

⁵The checkout procedure can be found on the homepage <http://www.deegree.org>

6 Extras

6.1 adding the optional Get3DFeatureInfo-Request

This feature is still under construction!

As in the GetFeatureInfo-Request of the WMS, one can also query informations of three-dimensional objects displayed by the deegreeWPVS. This operation is not described in the draft version of the WPVS specification, but it is additionally implemented in the deegree WPVS. To use this optional method one has to adapt the configuration file of the WPVS:

First you have to register the Get3DFeatureInfo operation - insert the following xml fragment in the operation metadata section:

```
<ows:Operation name="Get3DFeatureInfo">
  <ows:DCP>
    <ows:HTTP>
      <ows:Get xlink:href="http://127.0.0.1:8080/deegree-wpvs/services">
        <ows:Constraint name="operation_dcp_http_get_constraint_name_String">
          <ows:AllowedValues>
            <ows:Value>operation_dcp_http_get_constraint_owsValue_String</ows:Value>
          </ows:AllowedValues>
          <ows:DefaultValue>String</ows:DefaultValue>
        </ows:Constraint>
      </ows:Get>
    </ows:HTTP>
  </ows:DCP>
  <ows:Parameter name="Get3DFeatureInfoOperation_parameter_name_String">
    <ows:AllowedValues>
      <ows:Value>operation_parameter_owsValue_String</ows:Value>
    </ows:AllowedValues>
    <ows:DefaultValue>String</ows:DefaultValue>
  </ows:Parameter>
  <ows:Constraint name="operation_constrain_name_String">
    <ows:AllowedValues>
      <ows:Value>operation_constrain_owsValue_String</ows:Value>
    </ows:AllowedValues>
    <ows:DefaultValue>String</ows:DefaultValue>
  </ows:Constraint>
</ows:Operation>
```

Depending on your installation you maybe have to match the attribute `xlink:href` in the `Get` element. Then make sure that the Dataset(s) you want to query have the attribute `queryable="1"`. Otherwise you get no access to the associated features.

Now the Get3DFeatureInfo-Request can be used. The parameter needed to this operation are described as follows:

- YAW, PITCH, ROLL, AOV, DISTANCE, STYLES, WIDTH, HEIGHT, CRS, BOUNDINGBOX, OUTPUTFORMAT, EXCEPTIONS, DATASETS, POI: general information of the view

- QUERY_DATASETS: names the dataset(s) to query
- FEATURE_COUNT: sets the maximal number of features contained in the response
- I, J: specifies the coordinates (pixel) of the point of interest in the view
- DEPTH: limits the distance of the geometry of the request

6.2 teaching the WFS the Keyhole Markup Language (KML)

The Keyhole Markup Language (KML) is a file format for exchanging of three-dimensional geographical data. It is used i.e. by GoogleEarth. The deegree WFS used by a deegree WPVS as WFSDataSource can return features in KML. Only adjust the configuration file of the WFSDataSource. The format (text/xml; subtype=kml) in the `OutputFormat` element must be added in the feature type list section. This element has been extended by three additional attributes, providing the option to apply XSL-processing to responses.

```
<wfs:Format deegree:outFilter="xsl/wpvs2kml.xsl"
deegree:inFilter="xsl/dummy.xsl"
deegree:schemaLocation="http://code.google.com/apis/kml/schema/kml21.xsd">
text/xml; subtype=kml</wfs:Format>
```

- The **deegree:wfs:outFilter** must be set to the XSL-script wpvs2kml.xsl. It transforms the document that will be produced as output in KML.
- The **deegree:wfs:inFilter** must be set to the XSL-script dummy.xsl. It is necessary for transforming requests to match the internal (default) GML schema.
- The **deegree:wfs:schemaLocation** shows the location of the schema describing the KML-documents produced by the outFilter-script.

Make sure that the WFS is running in your tomcat. Chapter 5.2.1, Tomcat configuration files, describe how to register a service. In the list of the available services 'wfs' must be noticed. Furthermore the following parameters must be set:

```
<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_Capabilities.xml</param-value>
</init-param>
```

The parameter `wfs.config` must point at the configuration file of the WFS. Having restarted the Tomcat you can set the parameter `outputFormat` in your GetFeature-Requests to *text/xml; subtype=kml*.

Appendix A Third party libraries

deegree uses and redistributes the following third party libraries:

JTS - Java Topology Suite:

<http://www.vividsolutions.com/jts/jtshome.htm>

The JTS Topology Suite is an API of 2D spatial predicates and functions. It has the following design goals:

- JTS conforms to the Simple Features Specification for SQL published by the Open GIS Consortium
- JTS provides a complete, consistent, robust implementation of fundamental 2D spatial algorithms
- JTS is fast enough for production use
- JTS is written in 100% pure Java™

JTS is open source (under the LGPL license)

PostgreSQL/Postgis JDBC Driver

PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS), much like ESRI's SDE or Oracle's Spatial extension. PostGIS follows the OpenGIS "Simple Features Specification for SQL" and will be submitted for conformance testing at version 1.0.

PostGIS has been developed by Refractions Research Inc as a research project in open source spatial database technology. PostGIS is released under the GNU General Public License. We intend to continue development as time and resources permit. Our list of future projects includes enhanced technology for data loading and dumping, user interface tools for direct data access and manipulation, and support for advanced topologies at the server side, such as coverages, networks, and surfaces.

ACME Java - Software

<http://www.acme.com/java/software/>

All of this is completely free for any use, educational commercial or whatever. I do have to earn my grocery money, though. I'm available for consulting, and would be very interested in work that builds on these Java utilities. And of course if your company wants to sponsor ACME Labs with equipment or large cash donations, that would be ok too!

This software is written for JDK1.0.2, but with one exception it also compiles under JDK1.1. You'll get a lot of warnings about deprecated methods, but it'll work. The only exception is StubToolkit, where I had to use some of the internal sun.* classes.

Apache Batik

<http://xml.apache.org/batik/index.html>

Batik is a Java(tm) technology based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG) format for various purposes, such as viewing, generation or manipulation.

- Applications of Batik
- The SVG Specification
- What is Batik's Implementation Status?
- Downloading the Batik distribution (source and binary)
- Examples of projects and products using Batik

The project's ambition is to give developers a set of core modules which can be used together or individually to support specific SVG solutions. Examples of modules are the SVG Parser, the SVG Generator and the SVG DOM. Another ambition for the Batik project is to make it highly extensible (for example, Batik allows the developer to handle custom SVG tags). Even though the goal of the project is to provide a set of core modules, one of the deliverables is a full fledged SVG browser implementation which validates the various modules and their inter-operability.

Jakarta Commons

<http://jakarta.apache.org/commons/index.html>

The Commons is a Jakarta subproject focused on all aspects of reusable Java components.

The Jakarta Commons project is composed of two parts:

- The Commons Proper - A repository of reusable Java components.
- The Commons Sandbox - A workspace for Java component development.

Apache log4j

Apache log4j is a logging api provided by the apache project.

Appendix B Complete deegree2 WPVS configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<deegreewpvs:WPVS_Configuration xmlns:deegreewpvs="http://www.deegree.org/wpvs"
  xmlns:gml="http://www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows"
  xmlns:wpvs="http://www.opengis.net/wpvs"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:app="http://www.deegree.org/app" xmlns:wfs="http://www.opengis.net/wfs"
  version="1.0.0"
  updateSequence="String">
  <!-- ===== -->
  <!-- DEEGREE PARAMS SECTION -->
  <!-- ===== -->
  <deegreewpvs:deegreeParams>
    <deegreewpvs:DefaultOnlineResource xlink:href="http://localhost:8080/deegree-
wpvs/services" />
    <deegreewpvs:CacheSize>100</deegreewpvs:CacheSize>
    <!-- The time a request has, before it gets terminated -->
    <deegreewpvs:RequestTimeLimit>60</deegreewpvs:RequestTimeLimit>
    <!-- for jpeg (in the future tiff) encoding -->
    <deegreewpvs:ViewQuality>0.95</deegreewpvs:ViewQuality>
    <!-- maximum view width that can be requested. default = 1200 -->
    <deegreewpvs:MaxViewWidth>1200</deegreewpvs:MaxViewWidth>
    <!-- maximum view height that can be requested. default = 1000 -->
    <deegreewpvs:MaxViewHeight>1000</deegreewpvs:MaxViewHeight>
    <!-- what kind of splitter to use if the no splitter is requested (can be BBOX
or QUAD) -->
    <deegreewpvs:DefaultSplitter>QUAD</deegreewpvs:DefaultSplitter>
    <!-- which is printed in the lowerleft corner of the GetView (image) response
-->
    <deegreewpvs:Copyright>
      <!-- Text>Copyright 2006 lat/lon GmbH </Text -->
      <deegreewpvs:ImageURL xlink:href="./images/copyright/logo-
deegree_alpha_unaliassed.png"
        watermark="false" />
    </deegreewpvs:Copyright>
    <!-- If the splitter is QUAD should it generate lots of request (e.g higher
quality) or fewer
requests (e.g. lower Quality but quicker processing time)-->
    <deegreewpvs:RequestQualityPreferred>>false</deegreewpvs:RequestQualityPreferred>
  <
    <!-- What is the maximum requestable far clippingplane in meters-->
    <deegreewpvs:RequestsMaximumFarClippingPlane>150000</deegreewpvs:RequestsMaximu
mFarClippingPlane>
    <!-- The minimalheight of the terrain, if no fitting dgm is found all points
will be set to this
height, defaults to 0-->
    <deegreewpvs:MinimalTerrainHeight>1455</deegreewpvs:MinimalTerrainHeight>
    <!-- If a wcs is used for the elevationmodel, the following value defines the
minimal resolution
in meters, which means a wcs-request can not have a smaller resolution as
this value. This
helps removing typical "ricefield" forms in the landscape. -->
    <deegreewpvs:MinimalWCSElevationModelResolution>
      40
    </deegreewpvs:MinimalWCSElevationModelResolution>
    <!-- If a wcs is used for the elevationmodel, it is often possible to see a
little rim between
the request tiles. This value defines a percentage to add to the request,
resulting in
overlapping request thus removing the space between tiles. valid values are 0
<= x <= 100 -->
    <deegreewpvs:ExtendRequestPercentage>6</deegreewpvs:ExtendRequestPercentage>
    <!-- A list of the images a client can request to be rendered in the background
-->
```

```

<deegreewpvs:BackgroundList>
  <deegreewpvs:Background name="cloudy" href="images/background/cloudy.jpg" />
  <deegreewpvs:Background name="cirrus" href="images/background/cirrus.jpg" />
  <deegreewpvs:Background name="sunset" href="images/background/sunset.jpg" />
</deegreewpvs:BackgroundList>
</deegreewpvs:deegreeParams>
<!-- ===== -->
<!-- SERVICE IDENTIFICATION SECTION -->
<!-- ===== -->
<ows:ServiceIdentification>
  <ows:Title>WPVS for Salt Lake City, Utah -USA</ows:Title>
  <ows:Abstract>
    A WPVS which creates a perspective view of the Downtown region of Salt Lake
    City, Utah - USA
  </ows:Abstract>
  <ows:Keywords>
    <ows:Keyword>WPVS</ows:Keyword>
    <ows:Type>String</ows:Type>
  </ows:Keywords>
  <ows:ServiceType>WPVS</ows:ServiceType>
  <ows:ServiceTypeVersion>1.0.0</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.some.com" />
  <ows:ServiceContact>
    <ows:IndividualName>Rutger Bezema</ows:IndividualName>
    <ows:PositionName>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>bezema@lat-lon.de</ows:ElectronicMailAddress>
      </ows:Address>
      <ows:OnlineResource xlink:href="http://www.lat-lon.de" />
      <ows:HoursOfService>24x7</ows:HoursOfService>
      <ows:ContactInstructions>Write an email, or call</ows:ContactInstructions>
    </ows:ContactInfo>
    <ows:Role>PointOfContact</ows:Role>
  </ows:ServiceContact>
</ows:ServiceProvider>
<!-- ===== -->
<!-- OPERATIONS METADATA SECTION -->
<!-- ===== -->
<ows:OperationsMetadata>
  <ows:Operation name="GetCapabilities">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8080/deegree-wpvs/services">
          <ows:Constraint name="operation_dcp_http_get_constraint_name_String">
            <ows:AllowedValues>
              <ows:Value>operation_dcp_http_get_constraint_owsValue_String</ows:V
              alue>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
</ows:OperationsMetadata>

```

```

        </ows:AllowedValues>
        <ows:DefaultValue>String</ows:DefaultValue>
    </ows:Constraint>
</ows:Get>
</ows:HTTP>
</ows:DCP>
<ows:Parameter name="GetCapabilitiesOperation_parameter_name_String">
    <ows:AllowedValues>
        <ows:Value>operation_parameter_owsValue_String</ows:Value>
    </ows:AllowedValues>
    <ows:DefaultValue>String</ows:DefaultValue>
</ows:Parameter>
<ows:Constraint name="operation_constrain_name_String">
    <ows:AllowedValues>
        <ows:Value>operation_constrain_owsValue_String</ows:Value>
    </ows:AllowedValues>
    <ows:DefaultValue>String</ows:DefaultValue>
</ows:Constraint>
</ows:Operation>
<ows:Operation name="GetView">
    <ows:DCP>
        <ows:HTTP>
            <ows:Get xlink:href="http://localhost:8080/deegree-wpvs/services">
                <ows:Constraint name="String">
                    <ows:AllowedValues>
                        <ows:Value>String</ows:Value>
                    </ows:AllowedValues>
                    <ows:DefaultValue>String</ows:DefaultValue>
                </ows:Constraint>
            </ows:Get>
        </ows:HTTP>
    </ows:DCP>
    <ows:Parameter name="GetViewOperation_parameter_name_String">
        <ows:AllowedValues>
            <ows:Value>String</ows:Value>
        </ows:AllowedValues>
        <ows:DefaultValue>String</ows:DefaultValue>
    </ows:Parameter>
    <ows:Constraint name="String">
        <ows:AllowedValues>
            <ows:Value>String</ows:Value>
        </ows:AllowedValues>
        <ows:DefaultValue>String</ows:DefaultValue>
    </ows:Constraint>
</ows:Operation>
<ows:Operation name="Get3DFeatureInfo">
    <ows:DCP>
        <ows:HTTP>
            <ows:Get xlink:href="http://localhost:8080/deegree-wpvs/services">
                <ows:Constraint name="String">
                    <ows:AllowedValues>
                        <ows:Value>String</ows:Value>
                    </ows:AllowedValues>
                    <ows:DefaultValue>String</ows:DefaultValue>
                </ows:Constraint>
            </ows:Get>
        </ows:HTTP>
    </ows:DCP>
    <ows:Parameter name="Get3DFeatureInfoOperation_parameter_name_String">
        <ows:AllowedValues>
            <ows:Value>String</ows:Value>
        </ows:AllowedValues>
        <ows:DefaultValue>String</ows:DefaultValue>
    </ows:Parameter>
    <ows:Constraint name="String">

```

```

    <ows:AllowedValues>
      <ows:Value>String</ows:Value>
    </ows:AllowedValues>
    <ows:DefaultValue>String</ows:DefaultValue>
  </ows:Constraint>
</ows:Operation>

<ows:Parameter name="String">
  <ows:AllowedValues>
    <ows:Value>String</ows:Value>
  </ows:AllowedValues>
  <ows:DefaultValue>String</ows:DefaultValue>
</ows:Parameter>
<ows:Constraint name="String">
  <ows:AllowedValues>
    <ows:Value>String</ows:Value>
  </ows:AllowedValues>
  <ows:DefaultValue>String</ows:DefaultValue>
</ows:Constraint>
<ows:ExtendedCapabilities>Text</ows:ExtendedCapabilities>
</ows:OperationsMetadata>
<!-- ===== -->
<!--   DATASET SECTION   -->
<!-- ===== -->
<deegreepvs:Dataset queryable="0" opaque="0" noSubsets="0" fixedWidth="0"
fixedHeight="0">
  <!-- The name by which a Dataset shall be requested, the specification is not
clear about the real
    requestable element, therefor deegree uses the (optional) Name element to
find datasets for a
    given request. If the dataset is not a structural one the name is thus
mandatory (in deegree),
    in this toplayer dataset the name could be omitted.
  -->
  <deegreepvs:Name>WPVS Top Layer</deegreepvs:Name>
  <!-- the Title element will be represented to a client to pick from, not to be
confused with
    the identifier element, which will be given in a GetView request. -->
  <deegreepvs:Title>A human readable name for this dataset</deegreepvs:Title>
  <deegreepvs:Abstract>
    This Dataset is the root of all datasets, the values here will be used to
fill in optional
    values in it's children. This abstract element is optional.
  </deegreepvs:Abstract>
  <!-- Keywords: Optional Unordered list of one or more commonly used or
formalised word(s) or phrase(s) used to describe the subject.-->
  <ows:Keywords>
    <!-- mandatory element -->
    <ows:Keyword>Cool view</ows:Keyword>
    <!-- optional element describing in which namespace/dictionary etc. the
keywords can be found-->
    <ows:Type codeSpace="http://optional_codespace_attribute/">englisch slang
dictionary</ows:Type>
  </ows:Keywords>
  <!-- The root dataset must define at least one coordinate system, the other
datasets will use this crs if no other is given-->
  <deegreepvs:CRS>EPSG:4326</deegreepvs:CRS>
  <deegreepvs:CRS>EPSG:26912</deegreepvs:CRS>
  <!-- Mandatory element, describing the returned object of this dataset, e.g.
text/xml or image/png -->
  <deegreepvs:Format>image/png</deegreepvs:Format>
  <!-- Mandotory element -->
  <ows:WGS84BoundingBox crs="urn:ogc:def:crs:OGC:2:84" dimensions="2">
    <ows:LowerCorner>-111.95352895979279 40.724274423103856</ows:LowerCorner>
    <ows:UpperCorner>-111.88392802793884 40.871304682151964</ows:UpperCorner>
  </ows:WGS84BoundingBox>
</deegreepvs:Dataset>

```

```

</ows:WGS84BoundingBox>
<!-- Optional element BoundingBox can be used to precise the valid bbox for
the supported crs's -->
<ows:BoundingBox crs="EPSG:26912" dimensions="2">
  <ows:LowerCorner>424585.3 4512973.8</ows:LowerCorner>
  <ows:UpperCorner>425403.5 4513746.0</ows:UpperCorner>
</ows:BoundingBox>
<!-- Optional element dimension can be used to describe different dimension
inside a dataset,
comparable to the dimension element of a wms-layer.
Mandatory attributes are: name and units,
Optional attributes: unitSymbol (a string),
default (a string),
multipleValues (a boolean),
nearestValue (a boolean),
current (a boolean)
****Attention, deegree WPVS can parse, but not handle this element at the
moment**** -->
<wpvs:Dimension name="time" units="hour" unitSymbol="h" default="23"
multipleValues="false"
nearestValue="true" current="true">
  A time dimension of this dataset
</wpvs:Dimension>
<!-- Optional element DataProvider can be used to describe the owner of this
dataset -->
<wpvs:DataProvider>
  <wpvs:ProviderName>Optional name of this provider</wpvs:ProviderName>
  <wpvs:ProviderSite
xlink:href="http://Optional_internetsite_of_this_provider.com" />
  <!-- optional logo which can be displayed to a user, optional attributes width
and height > 0 -->
  <wpvs:LogoURL width="400" height="400">
    <!-- mandatory element format -->
    <wpvs:Format>mime/type</wpvs:Format>
    <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
  </wpvs:LogoURL>
</wpvs:DataProvider>
<!-- mandatory element Identifier which clients will use to refer this dataset
to, unambiguous/unique for all datasets -->
<deegreewpvs:Identifier codeSpace="http://www.optional_attribute.com">
  first_dataset_identifier
</deegreewpvs:Identifier>
<!-- optional element metadata, can be used for detailed, standardized metadata
about the data underneath a particular dataset -->
<wpvs:MetaData
type="The type attribute indicates the standard to which the metadata
complies">
  <!-- mandatory element format -->
  <wpvs:Format>mime/type</wpvs:Format>
  <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
</wpvs:MetaData>
<!-- optional element DatasetReference A wpvs Server may use DatasetReference
to offer a link to the underlying data represented
by a particular dataset -->
<wpvs:DatasetReference>
  <!-- mandatory element format -->
  <wpvs:Format>mime/type</wpvs:Format>
  <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
</wpvs:DatasetReference>
<!-- optional element FeatureListReference A wpvs Server may use
FeatureListReference to point to a list of the
features represented in a dataset -->
<wpvs:FeatureListReference>
  <!-- mandatory element format -->
  <wpvs:Format>mime/type</wpvs:Format>

```

```

    <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
</wpvs:FeatureListReference>
<!-- optional element style, lists the name by which a style is requested and
a
human-readable title for pick lists, optionally (and ideally)
provides a human-readable description, and optionally gives a style
URL.-->
<wpvs:Style>
  <wpvs:Name>the human readable mandatory name</wpvs:Name>
  <wpvs:Title>the unique mandatory style title</wpvs:Title>
  <wpvs:Abstract>describing the syle mandatory</wpvs:Abstract>
  <ows:Keywords>
    <ows:Keyword>Cool Style</ows:Keyword>
    <ows:Type codeSpace="http://optional_codespace_attribute/">
      englisch slang dictionary
    </ows:Type>
  </ows:Keywords>
  <wpvs:Identifier codeSpace="http://www.optional_attribute.com">
    mandatory_unique_style_identifiier
  </wpvs:Identifier>
  <!-- optional list of element LegendURL (attributes width and height are
mandatory), A WPVS Server may use zero or more LegendURL elements to provide an
image(s) of a legend relevant to each Style of a Dataset -->
  <wpvs:LegendURL width="400" height="400">
    <!-- mandatory element format -->
    <wpvs:Format>mime/type</wpvs:Format>
    <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
  </wpvs:LegendURL>
  <!-- Optional list of element StyleSheeetURL provides symbology information
for each Style of a Dataset -->
  <wpvs:StyleSheetURL>
    <!-- mandatory element format -->
    <wpvs:Format>mime/type</wpvs:Format>
    <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
  </wpvs:StyleSheetURL>
  <!-- Optional list of element StyleURL. A WPVS Server may use StyleURL to
offer more information about the
data or symbology underlying a particular Style -->
  <wpvs:StyleURL>
    <!-- mandatory element format -->
    <wpvs:Format>mime/type</wpvs:Format>
    <wpvs:OnlineResource xlink:href="http://optinal_online_location.com" />
  </wpvs:StyleURL>
</wpvs:Style>
<!-- mandatory element MinimumScaleDenominator 0 < value <=
maximumScaleDenominator-->
  <deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominator>
<!-- mandatory element MaximumScaleDenominator value >=
minimumScaleDenominator-->
  <deegreewpvs:MaximumScaleDenominator>55555555</deegreewpvs:MaximumScaleDenomina
tor>
  <!-- ***** -->
  <!-- FIRST CHILD DATASET, -->
  <!-- which uses a localwfs to display some buildings -->
  <!-- ***** -->
  <deegreewpvs:Dataset queryable="1" opaque="0" noSubsets="0" fixedWidth="0"
fixedHeight="0">
    <!-- The schema says name is optional, for deegree the name is only optional
if the dataset is
a structural one (one which will not provide viewable data and thus cannot
be requested by a
GetView-Request). This dataset provides viewable data therefore this name
element is
mandatory -->

```

```

<deegreewpvs:Name>Utah_Overview</deegreewpvs:Name>
<deegreewpvs:Title>Utah WPVS</deegreewpvs:Title>
<deegreewpvs:Format>innerDataset_format_application/x;x=x</deegreewpvs:Format
>
<deegreewpvs:Abstract>
  A dataset which shows some buildings of Salt Lake City, Utah - USA
</deegreewpvs:Abstract>
<deegreewpvs:CRS>EPSG:26912</deegreewpvs:CRS>
<ows:WGS84BoundingBox crs="urn:ogc:def:crs:OGC:2:84" dimensions="2">
  <ows:LowerCorner>-111.95352895979279 40.724274423103856</ows:LowerCorner>
  <ows:UpperCorner>-111.88392802793884 40.771304682151964</ows:UpperCorner>
</ows:WGS84BoundingBox>
<deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominator
>
<deegreewpvs:MaximumScaleDenominator>700000</deegreewpvs:MaximumScaleDenomina
tor>
<deegreewpvs:Identifier>second_dataset_identifier</deegreewpvs:Identifier>
<!-- ***** -->
<!-- A LocalWFSDataSource -->
<!-- ***** -->
<!-- The schema says DataSource is optional, for deegree the DataSource is
only optional if
  the dataset is a structural one (one which will not provide viewable data
and thus cannot be
  requested by a GetView-Request). This dataset provides viewable data
therefore at least one
  DataSource element is mandatory -->
<deegreewpvs:LocalWFSDataSource>
  <!-- The Name element of a deegreewpvs:DataSource is optional. It is only
useful for the
  administrator and finding debug information -->
  <deegreewpvs:Name>app:WPVS</deegreewpvs:Name>
  <!-- The mandatory OWSCapabilities element defines the location of the
capabilities
  document of this DataSource -->
  <deegreewpvs:OWSCapabilities>
    <!-- mandatory OnlineResource element -->
    <deegreewpvs:OnlineResource xlink:type="simple"
      xlink:href="file:../wfs/wfs_configuration.xml" />
  </deegreewpvs:OWSCapabilities>
  <!-- Each DataSource must have a MinimumScaleDenominator -->
  <deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominat
or>
  <!-- Each DataSource must have a MaximumScaleDenominator -->
  <deegreewpvs:MaximumScaleDenominator>700000</deegreewpvs:MaximumScaleDenomi
nator>
  <!-- Optional GeometryProperty defining the location of the Geometry inside
a Feature,
  though a string is valid, this should be a qualified name -->
  <deegreewpvs:GeometryProperty>app:geometry</deegreewpvs:GeometryProperty>
</deegreewpvs:LocalWFSDataSource>
</deegreewpvs:Dataset>

<!-- ***** -->
<!-- Second CHILD DATASET, -->
<!-- which shows only the mandatory elements -->
<!-- ***** -->
<deegreewpvs:Dataset queryable="0" opaque="0" noSubsets="0" fixedWidth="0"
fixedHeight="0">
  <deegreewpvs:Name>satellite_images</deegreewpvs:Name>
  <deegreewpvs:Title>SaltLake Satellite photos</deegreewpvs:Title>
  <deegreewpvs:Format>image/jpg</deegreewpvs:Format>
  <deegreewpvs:CRS>EPSG:26912</deegreewpvs:CRS>
  <ows:WGS84BoundingBox crs="urn:ogc:def:crs:OGC:2:84" dimensions="2">
    <ows:LowerCorner>-111.94721360400128 40.719021928792515</ows:LowerCorner>

```

```

    <ows:UpperCorner>-111.80644038553243 40.82820182642026</ows:UpperCorner>
  </ows:WGS84BoundingBox>
  <deegreepvs:Identifier>third_dataset_identifier</deegreepvs:Identifier>
  <deegreepvs:MinimumScaleDenominator>0.1</deegreepvs:MinimumScaleDenominator>
<
  <deegreepvs:MaximumScaleDenominator>700000</deegreepvs:MaximumScaleDenominator>

  <!-- ***** -->
  <!-- A LocalWCFSDataSource -->
  <!-- ***** -->
  <deegreepvs:LocalWCFSDataSource>
    <deegreepvs:Name>satlakesatellite</deegreepvs:Name>
    <deegreepvs:OWSCapabilities>
      <deegreepvs:OnlineResource xlink:type="simple"
xlink:href="file:../wcs/wcs_configuration.xml" />
    </deegreepvs:OWSCapabilities>
    <!-- optional ValidArea element defines an area a datasource is valid for,
if no ValidArea
      is given, the surrounding datasets WGS84BoundingBox is used. Valid
Geometry definitions are
      gml:Surface, gml:Polygon or ows:BoundingBox
      <deegreepvs:ValidArea>
        <gml:Polygon srsName="EPSG:31466">
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates cs="," decimal="." ts=" ">
3565687.9,5935846.7 3568308.4,5935846.7 3568308.4,5937998.9
3565687.9,5937998.9
              </gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </deegreepvs:ValidArea>-->
    <deegreepvs:MinimumScaleDenominator>0.1</deegreepvs:MinimumScaleDenominator>
or>
    <deegreepvs:MaximumScaleDenominator>1000000</deegreepvs:MaximumScaleDenominator>
  <!-- Mandatory FilterCondition element, for a CoverageDataSourceType this
element defines
      parts of a wcs/wms request, for a WFSDataSourceType this element defines
a wfs:query -->
    <deegreepvs:FilterCondition>
      <!-- Mandatory Request element of type WCSRequest, WMSRequest or
wfs:Query -->
      <deegreepvs:WCSRequest>
        <![CDATA[VERSION=1.0.0&coverage=satlakesatellite&TRANSPARENT=TRUE&FORMAT=jpg&EXCEPTIONS=application/vnd.ogc.se_xml&crs=EPSG:26912]]>
      </deegreepvs:WCSRequest>
    </deegreepvs:FilterCondition>
    <!-- Optional TransparentColors element defines a color in a
CoverageRequest (wcs/wms)
      which should be transparent, for the ElevationModel element this element
has no effect -->
    <deegreepvs:TransparentColors>
      <!-- Mandatory Color(s) element(s), defining color(s) in hex or oct
String format,
      which should be transparent -->
      <deegreepvs:Color>0xffffffff</deegreepvs:Color>
      <deegreepvs:Color>0x777777</deegreepvs:Color>
    </deegreepvs:TransparentColors>
  </deegreepvs:LocalWCFSDataSource>
</deegreepvs:Dataset>

```

```

<!-- optional element ElevationModel, The elevation model is valid for all
child datasets if no
other is given, if no elevationmodel is given at all the wpvs uses
deegreewpvs:MinimalTerrainHeight or 0 -->
<deegreewpvs:ElevationModel>
  <!-- Mandatory Name element with which this ElevationModel shall be requested
-->
  <deegreewpvs:Name>saltlakedem</deegreewpvs:Name>
  <!-- some datasources which can be one or more of LocalWCSDataSource,
RemoteWCSDataSource, LocalWFSDDataSource or RemoteWFSDDataSource -->
  <deegreewpvs:LocalWCSDataSource>
    <deegreewpvs:Name>DEM-Model of Salt Lake City</deegreewpvs:Name>
    <deegreewpvs:OWSCapabilities>
      <deegreewpvs:OnlineResource xlink:type="simple"
xlink:href="file:../wcs/wcs_configuration.xml" />
    </deegreewpvs:OWSCapabilities>
    <deegreewpvs:MinimumScaleDenominator>0.1</deegreewpvs:MinimumScaleDenominat
or>
    <deegreewpvs:MaximumScaleDenominator>1000000</deegreewpvs:MaximumScaleDenom
inator>
    <deegreewpvs:FilterCondition>
      <deegreewpvs:WCSRequest>
        <![CDATA[VERSION=1.0.0&coverage=saltlakedem&TRANSPARENT=TRUE&FORMAT=Geo
Tiff&EXCEPTIONS=application/vnd.ogc.se_xml&crs=EPSG:26912&response_CRS=EPSG:26912]]
>
      </deegreewpvs:WCSRequest>
    </deegreewpvs:FilterCondition>
    <deegreewpvs:TransparentColor>
      <deegreewpvs:Color>#000000</deegreewpvs:Color>
    </deegreewpvs:TransparentColor>
  </deegreewpvs:LocalWCSDataSource>
</deegreewpvs:ElevationModel>
<!-- The schema says DataSource is optional, for deegree the DataSource element
is only
optional if the dataset is a structural one (one which will not provide
viewable data and
thus cannot be requested by a GetView-Request). This (top layer) dataset
provides no
viewable data therefore no DataSource element is given -->
</deegreewpvs:Dataset>
</deegreewpvs:WPVS_Configuration>

```

Appendix C Rendering configuration options

```
#####
# these values can be used to adjust the rendering of terrains and textures
#####
#blendingfunction, defines the way the terrain primitive colors and textures are
#rendered onto each other
## allowed values are (C'=outgoing color, C=Color, t=target, a=alpha):
# 1) MODULATE -> C' = C Ct (Default)
# 2) BLEND -> C'rgb = Crgb*(1 - Ctrgb) + Cbrgb*Ctrgb and C'a = Ca Ct
# 3) DECAL -> C'rgb = Crgb*(1 - Cta) + Ctrgb*Cta and C'a = C
# 4) COMBINE -> further values need to be set(below)
#####
blend_function=MODULATE
#####
#blending_color, (Applies only to blend_function=BLEND) defines the constant color
#to blend with
## allowed values are
# 1) any one of the rgba parameters between >=0 and <=1. (default= 0,0,0,0)
#####
blend_color=0,0,0,0
#####
#combinefunction, defines the way the terrain primitive colors and textures are
#rendered onto each other
##allowed values are:
# 1) COMBINE_REPLACE -> C' = C0
# 2) COMBINE_MODULATE -> C' = C0 C1 (Default)
# 3) COMBINE_ADD -> C' = C0 + C1
# 4) COMBINE_ADD_SIGNED -> C' = C0 + C1 - 0.5
# 5) COMBINE_SUBTRACT -> C' = C0 - C1
# 6) COMBINE_INTERPOLATE -> C' = C0 C2 + C1 (1 - C2)
# 7) COMBINE_DOT3 -> C' = 4 * ( (C0r - 0.5) * (C1r - 0.5) + (C0g - 0.5) * (C1g -
#0.5) + (C0b - 0.5) * (C1b - 0.5))
#####
combine_function_rgb=COMBINE_ADD_SIGNED
combine_function_alpha=COMBINE_MODULATE
#combine_color_source, defines the source for a color operand (the 0, 1 or 2 ) in
#the combine operation
## allowed values are:
# 1) COMBINE_OBJECT_COLOR
# 2) COMBINE_TEXTURE_COLOR (Default)
# 3) COMBINE_PREVIOUS_TEXTURE_UNIT_STATE (Default2)
# 3) COMBINE_CONSTANT_COLOR (Default3)
#####
combine_color_source_rgb_0=COMBINE_TEXTURE_COLOR
combine_color_source_rgb_1=COMBINE_PREVIOUS_TEXTURE_UNIT_STATE
combine_color_source_rgb_2=COMBINE_CONSTANT_COLOR
combine_color_source_alpha_0=COMBINE_TEXTURE_COLOR
combine_color_source_alpha_1=COMBINE_PREVIOUS_TEXTURE_UNIT_STATE
combine_color_source_alpha_2=COMBINE_CONSTANT_COLOR
#combine_color_function_rgb, specifies the function for a color operand in the
#combine operation
## allowed values are:
# 1) COMBINE_SRC_COLOR - the color function is f = Crgb (default)
# 2) COMBINE_ONE_MINUS_SRC_COLOR - the color function is f = (1 - Crgb )
#####
combine_color_function_rgb=COMBINE_SRC_COLOR
#combine_color_function_alpha, specifies the function for a alpha operand in the
#combine operation
## allowed values are:
# 1) COMBINE_SRC_ALPHA - the color function is f = Ca
# 2) COMBINE_ONE_MINUS_SRC_ALPHA - the color function is f = (1 - Ca)
#####
combine_color_function_alpha=COMBINE_SRC_ALPHA
```

```
#combine_scale, specifies the scale factor to be applied to the output color of the
#combine operation.
## allowed values are:
# 1) 1 (default)
# 2) 2
# 3) 4
#####
combine_scale_factor_rgb=1
combine_scale_factor_alpha=1

#####
#shading_model, defines the shading technique used
## allowed values are:
# 1) SHADE_FLAT //for simple shading
# 2) SHADE_GOURAUD //for better shading (Default)
#####
shading_model=SHADE_GOURAUD
#####
#shading_quality, defines the (hinted) shading quality
## allowed values are:
# 1) FASTEST //for simple, not very good shading
# 2) NICEST //for better shading (Default)
#####
shading_quality=NICEST
#####
#terrain_culling, can be used to cull (not draw) back or front facing
#terrainprimitives, resulting in higher rendering speed
## allowed values are:
# 1) BACK //Primitives with backfacing normals are culled
# 2) FRONT //Primitives with frontfacing normals are culled
# 3) NONE //no Primitives are culled (Default)
#####
terrain_culling=NONE
#####
#terrain_backflip, can be used to invert backfacing terrain primitives.
## allowed values are:
# 1) TRUE //the backfacing normals of terrains should be flipped (Default)
# 2) FALSE //the backfacing normals of terrains should be flipped
#####
terrain_backflip=TRUE
#####
#terrain_texture_mipmapping, can be used to create a mipmap of terrain textures
## allowed values are:
# 1) TRUE //create and render mipmapped terrain textures (Default)
# 2) FALSE //don'tcreate and render mipmapped terrain textures
#####
terrain_texture_mipmapping=TRUE
#####
#terrain_texture_perspective_correction, defines the (hinted) texture mapping mode
## allowed values are:
# 1) FASTEST //for fast but simple, not very good mapping
# 2) NICEST //for better mapping (Default)
#####
terrain_texture_perspective_correction=NICEST
#####
#surface_culling, can be used to cull (not draw) back or front facing surfaces,
#resulting in higher rendering speed
## allowed values are:
# 1) BACK //Primitives with backfacing normals are culled
# 2) FRONT //Primitives with frontfacing normals are culled
# 3) NONE //no Primitives are culled (Default)
#####
surface_culling=NONE
#####
```

```
#surface_backflip, can be used to invert backfacing surface primitives (like  
#walls).  
## allowed values are:  
# 1) TRUE //the backfacing normals of surfaces should be flipped (Default)  
# 2) FALSE //the backfacing normals of surfaces should be flipped  
####  
surface_backflip=TRUE
```

Appendix D Material properties file

```
#####  
# these values can be used to adjust the default behaviour of a surface style  
# if no material has been defined in a CityGML object  
shininess=1 0 0  
transparency=0  
ambientintensity=1  
specularcolor=1 1 1  
diffusecolor=1 1 1
```