Oracle® Big Data Spatial and Graph

User's Guide and Reference

Release 1.2

E67958-03

May 2016

Oracle Big Data Spatial and Graph User's Guide and Reference, Release 1.2

E67958-03

Copyright © 2015, 2016, Oracle and/or its affiliates. All rights reserved.

Primary Authors: Chuck Murray, Harihara Subramanian, Donna Carver

Contributors: Bill Beauregard, Hector Briseno, Hassan Chafi, Zazhil Herena, Sungpack Hong, Roberto Infante, Hugo Labra, Gabriela Montiel-Moreno, Siva Ravada, Carlos Reyes, Korbinian Schmid, Jane Tao, Zhe (Alan) Wu

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

2 Using Big Data Spatial and Graph with Spatial Data

3 Configuring Property Graph Support

4 Using Property Graphs in a Big Data Environment

5 Using the In-Memory Analyst

6 Using Multimedia Analytics

A Third-Party Licenses for Bundled Software

B Hive Spatial Functions

Index

List of Figures

List of Tables

Preface

This document provides conceptual and usage information about Oracle Big Data Spatial and Graph, which enables you to create, store, and work with Spatial and Graph vector, raster, and property graph data in a Big Data environment.

Audience

This document is intended for database and application developers in Big Data environments.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents"

- *Oracle Big Data Connectors User's Guide*
- *Oracle Big Data Spatial and Graph Java API Reference for Apache HBase*
- *Oracle Big Data Spatial and Graph Java API Reference for Oracle NoSQL Database*
- *Oracle Big Data Appliance Site Checklists*
- *Oracle Big Data Appliance Owner's Guide*
- *Oracle Big Data Appliance Safety and Compliance Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Big Data Spatial and Graph User's Guide and Reference

This preface describes significant new features and changes in *Oracle Big Data Spatial and Graph User's Guide and Reference* for Oracle Big Data Spatial and Graph Release 1.2.

- Enhanced Information about Installation and Configuration
- Enhanced Information about Spatial Vector Analysis
- Vector Hive Analysis Information Added
- Enhanced Information about the Vector Console
- Expanded Information about Using Property Graphs
- New and Enhanced Java API Reference Material

Enhanced Information about Installation and Configuration

The information about installing and configuring Oracle Big Data Spatial and Graph has been revised and enhanced, especially for Installing and Configuring the Big Data Spatial Image Server.

Enhanced Information about Spatial Vector Analysis

Oracle Big Data Spatial Vector Analysis has been revised and expanded to cover more operations and options.

Vector Hive Analysis Information Added

Oracle Big Data Spatial Vector Hive Analysis describes spatial functions to analyze the data using Hive.

Hive Spatial Functions provides reference information about the available functions.

Enhanced Information about the Vector Console

Using the Oracle Big Data Spatial and Graph Vector Console provides expanded information abuot using the Oracle Big Data Spatial and Graph Vector Console to perform tasks related to spatial indexing and creating and showing thematic maps.

Expanded Information about Using Property Graphs

Using Property Graphs in a Big Data Environment and Using the In-Memory Analyst provide more information and examples. Major additions include:

- New SQL-like declarative language for querying property graph data, including a rich set of graph pattern matching capabilities, For details, see Querying Property Graph Data.
- New distributed mode for the in-memory analyst. The in-memory analyst can now be run in a distributed mode, in which multiple nodes (computers) form a cluster, partition a large property graph across distributed memory, and work together to provide efficient and scalable graph analytics. For details, see Using the In-Memory Analyst in Distributed Mode.
- In-memory analyst support for reading data from HDFS. For details, see Loading Data from HDFS.
- New built-in analytics. More built-in analytics have been added, including the following:
  - Approximate Pagerank is a faster variant of Pagerank that can be used when less precision is acceptable.
  - Weighted Pagerank considers edge weights.
  - Personalized SALSA evaluates the relative importance of nodes with respect to a given set of hub nodes.
  - K-Core computes k-core decomposition of a graph.

  For details, see the *Oracle Big Data Spatial and Graph In-Memory Analyst Java API Reference* (Javadoc).

New and Enhanced Java API Reference Material

The Java API Reference (Javadoc) material for Big Data Spatial and Graph reflects many new packages and classes available with this release. The Big Data Appliance Documentation Library has a "Big Data Spatial and Graph" section with links to the available Java API Reference material.

1

Big Data Spatial and Graph Overview

This chapter provides an overview of Oracle Big Data support for Oracle Spatial and Graph spatial, property graph, and multimedia analytics features.

1.1 About Big Data Spatial and Graph

Oracle Big Data Spatial and Graph delivers advanced spatial and graph analytic capabilities to supported Apache Hadoop and NoSQL Database Big Data platforms.

The spatial features include support for data enrichment of location information, spatial filtering and categorization based on distance and location-based analysis, and spatial data processing for vector and raster processing of digital map, sensor, satellite and aerial imagery values, and APIs for map visualization.

The property graph features support Apache Hadoop HBase and Oracle NoSQL Database for graph operations, indexing, queries, search, and in-memory analytics.

The multimedia analytics features provide a framework for processing video and image data in Apache Hadoop, including built-in face recognition using OpenCV.

1.2 Spatial Features

Spatial location information is a common element of Big Data. Businesses can use spatial data as the basis for associating and linking disparate data sets. Location information can also be used to track and categorize entities based on proximity to another person, place, or object, or on their presence a particular area. Location information can facilitate location-specific offers to customers entering a particular geography, something known as *geo-fencing*. Georeferenced imagery and sensory data can be analyzed for a variety of business benefits.

The Spatial features of Oracle Big Data Special and Graph support those use cases with the following kinds of services.

Vector Services:

- Ability to associate documents and data with names, such as cities or states, or longitude/latitude information in spatial object definitions for a default administrative hierarchy
- Support for text-based 2D and 3D geospatial formats, including GeoJSON files, Shapefiles, GML, and WKT, or you can use the Geospatial Data Abstraction Library (GDAL) to convert popular geospatial encodings such as Oracle SDO_Geometry, ST_Geometry, and other supported formats
- An HTML5-based map client API and a sample console to explore, categorize, and view data in a variety of formats and coordinate systems
- Topological and distance operations: Anyinteract, Inside, Contains, Within Distance, Nearest Neighbor, and others
- Spatial indexing for fast retrieval of data

Raster Services:

- Support for many image file formats supported by GDAL and image files stored in HDFS
- A sample console to view the set of images that are available
- Raster operations, including, subsetting, georeferencing, mosaics, and format conversion

1.3 Property Graph Features

Graphs manage networks of linked data as vertices, edges, and properties of the vertices and edges. Graphs are commonly used to model, store, and analyze relationships found in social networks, cyber security, utilities and telecommunications, life sciences and clinical data, and knowledge networks.

Typical graph analyses encompass graph traversal, recommendations, finding communities and influencers, and pattern matching. Industries including, telecommunications, life sciences and healthcare, security, media and publishing can benefit from graphs.

The property graph features of Oracle Big Data Special and Graph support those use cases with the following capabilities:

- A scalable graph database on Apache HBase and Oracle NoSQL Database
- Developer-based APIs based upon Tinkerpop Blueprints, and Java graph APIs
- Text search and query through integration with Apache Lucene and SolrCloud
- Scripting languages support for Groovy and Python
- A parallel, in-memory graph analytics engine
- A fast, scalable suite of social network analysis functions that include ranking, centrality, recommender, community detection, path finding

- Parallel bulk load and export of property graph data in Oracle-defined flat files format
- Manageability through a Groovy-based console to execute Java and Tinkerpop Gremlin APIs

See also Property Graph Sizing Recommendations

1.3.1 Property Graph Sizing Recommendations

The following are recommendations for property graph installation.

Table 1-1 Property Graph Sizing Recommendations

Graph Size	Recommended Physical Memory to be Dedicated	Recommended Number of CPU Processors
10 to 100M edges	Up to 14 GB RAM	2 to 4 processors, and up to 16 processors for more compute-intensive workloads
100M to 1B edges	14 GB to 100 GB RAM	4 to 12 processors, and up to 16 to 32 processors for more compute-intensive workloads
Over 1B edges	Over 100 GB RAM	12 to 32 processors, or more for especially compute-intensive workloads

1.4 Multimedia Analytics Features

The multimedia analytics feature of Oracle Big Data Spatial and Graph provides a framework for processing video and image data in Apache Hadoop. The framework enables distributed processing of video and image data.

A main use case is performing facial recognition in videos and images.

1.5 Installing Oracle Big Data Spatial and Graph on an Oracle Big Data Appliance

The Mammoth command-line utility for installing and configuring the Oracle Big Data Appliance software also installs the Oracle Big Data Spatial and Graph option, including the spatial, property graph, and multimedia capabilities. You can enable this option during an initial software installation, or afterward using the `bdacli` utility.

To use Oracle NoSQL Database as a graph repository, you must have an Oracle NoSQL Database cluster.

To use Apache HBase as a graph repository, you must have an Apache Hadoop cluster.

See Also:

Oracle Big Data Appliance Owner's Guide for software configuration instructions.

1.6 Installing and Configuring the Big Data Spatial Image Processing Framework

Installing and configuring the Image Processing Framework depends upon the distribution being used.

- The Oracle Big Data Appliance cluster distribution comes with a pre-installed setup, but you must follow few steps in Installing Image Processing Framework for Oracle Big Data Appliance Distribution to get it working.
- For a commodity distribution, follow the instructions in Installing the Image Processing Framework for Other Distributions (Not Oracle Big Data Appliance).

After performing the installation, verify it (see Post-installation Verification of the Image Processing Framework).

1.6.1 Installing Image Processing Framework for Oracle Big Data Appliance Distribution

The Oracle Big Data Appliance distribution comes with a pre-installed configuration. However, perform the following actions to ensure that it works.

- Identify the `ALL_ACCESS_FOLDER` under `/opt/shareddir/spatial`.
- Make the `libproj.so (Proj.4)` Cartographic Projections Library accessible to the users, and copy `libproj.so` to the `gdal/lib` folder under `/opt/oracle/oracle-spatial-graph/spatial/gdal/lib`, as follows:

  ```
  cp libproj.so  /opt/oracle/oracle-spatial-graph/spatial/gdal/lib
  ```

- Provide read and execute permissions for the `libproj.so` library for all users, as follows:

  ```
  chmod 755 /opt/oracle/oracle-spatial-graph/spatial/gdal/lib/libproj.so
  ```

1.6.2 Installing the Image Processing Framework for Other Distributions (Not Oracle Big Data Appliance)

For Big Data Spatial and Graph in environments other than the Big Data Appliance, follow the instructions in this section.

1.6.2.1 Prerequisites for Installing the Image Processing Framework for Other Distributions

- Ensure that `HADOOP_LIB_PATH` is under `/usr/lib/hadoop`. If it is not there, find the path and use it as it your `HADOOP_LIB_PATH`.
- Install NFS.
- Have at least one folder, referred in this document as SHARED_FOLDER, in the Resource Manager node accessible to every Node Manager node through NFS.
- Provide write access to all the users involved in job execution and the yarn users to this SHARED_FOLDER
- Download `oracle-spatial-graph-<version>.x86_64.rpm` from the Oracle e-delivery web site.

- Execute `oracle-spatial-graph-<version>.x86_64.rpm` using the rpm command.
- After rpm executes, verify that a directory structure created at `/opt/oracle/oracle-spatial-graph/spatial` contains these folders: `console`, `examples`, `jlib`, `gdal`, and `tests`. Additionally, `index.html` describes the content, and `HadoopRasterProcessorAPI.zip` contains the Javadoc for the API..

1.6.2.2 Installing the Image Processing Framework for Other Distributions

1. Make the `libproj.so (Proj.4)` Cartographic Projections Library accessible to the users, and copy `libproj.so` to `gdal/lib` under `/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib`, as follows:

   ```
   cp libproj.so  /opt/oracle/oracle-spatial-graph/spatial/gdal/lib
   ```

2. Provide read and execute permissions for the `libproj.so` library for all users, as follows::

   ```
   chmod 755 /opt/oracle/oracle-spatial-graph/spatial/gdal/lib/libproj.so
   ```

3. In the Resource Manager Node, copy the `gdal` data folder under `/opt/oracle/oracle-spatial-graph/spatial/gdal` and `gdalplugins` under `/opt/oracle/oracle-spatial-graph/spatial/gdal` into the SHARED_FOLDER as follows:

   ```
   cp -R /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/
   data SHARED_FOLDER
   ```

4. Create a directory `ALL_ACCESS_FOLDER` under `SHARED_FOLDER` with write access for all users involved in job execution. The `hdfs` user is the one shown here to run tests, but also consider the `yarn` user in the write access because job results are written by this user. Group access may be used to configure this.

   Go to the shared folder.

   ```
   cd SHARED_FOLDER
   ```

   Create a new directory.

   ```
   mkdir ALL_ACCESS_FOLDER
   ```

   Provide write access.

   ```
   chmod 777 ALL_ACCESS_FOLDER
   ```

5. Copy the data folder under `/opt/oracle/oracle-spatial-graph/spatial/demo` into `ALL_ACCESS_FOLDER`.

   ```
   cp -R /opt/oracle/oracle-spatial-graph/spatial/raster/
   examples/data ALL_ACCESS_FOLDER
   ```

6. Provide write access to the `data/xmls` folder as follows (or just ensure that users executing the jobs, including tests and examlpes, have write access):

   ```
   chmod 777 ALL_ACCESS_FOLDER/data/xmls/
   ```

1.6.3 Post-installation Verification of the Image Processing Framework

Several test scripts are provided: one to test the image loading functionality, another to test the image processing functionality, another to test a processing class for slope calculation in a DEM and a map algebra operation, and another to verify the image processing of a single raster with no mosaic process (it includes a user-provided function that calculates hill shade in the mapping phase).. Execute these scripts to verify a successful installation of image processing framework.

To execute the scripts, make sure the current user is the `hdfs` user. Switch to this user before executing the scripts, and make sure you have provided this user with access to write to the necessary directories.

```
sudo su - hdfs
```

1.6.3.1 Image Loading Test Script

This script loads a set of four test rasters into the `ohiftest` folder in HDFS, 3 rasters of byte data type and 3 bands and 1 raster (DEM) of float32 data type and 1 band. No parameters are required for OBDA environments and a single parameter with the $ALL_ACCESS_FOLDER value is required for non-OBDA environments.

Internally, the job creates a split for every raster to load. Split size depends on the block size configuration; for example, if a block size >= 64MB is configured, 4 mappers will run; and as a result the rasters will be loaded in HDFS and a corresponding thumbnail will be created for visualization. An external image editor is required to visualize the thumbnails, and an output path of these thumbnails is provided to the users upon successful completion of the job.

The test script can be found here:

```
/oracle/oracle-spatial-graph/raster/tests/runimageloader.sh
```

For ODBA environments, enter:

```
./runimageloader.sh
```

For non-ODBA environments, enter:

```
./runimageloader.sh ALL_ACCESS_FOLDER
```

Upon successful execution, the message `GENERATED OHIF FILES ARE LOCATED IN HDFS UNDER` is displayed, with the path in HDFS where the files are located (this path depends on the definition of ALL_ACCESS_FOLDER) and a list of the created images and thumbnails on HDFS. The output may include:

```
“THUMBNAILS CREATED ARE:
----------------------------------------------------------------------
total 13532
drwxr-xr-x 2 yarn yarn    4096 Sep  9 13:54 .
drwxr-xr-x 3 yarn yarn    4096 Aug 27 11:29 ..
-rw-r--r-- 1 yarn yarn 3214053 Sep  9 13:54 hawaii.tif.ohif.tif
-rw-r--r-- 1 yarn yarn 3214053 Sep  9 13:54 kahoolawe.tif.ohif.tif
-rw-r--r-- 1 yarn yarn 3214053 Sep  9 13:54 maui.tif.ohif.tif
-rw-r--r-- 1 yarn yarn 4182040 Sep  9 13:54 NapaDEM.tif.ohif.tif
YOU MAY VISUALIZE THUMBNAILS OF THE UPLOADED IMAGES FOR REVIEW FROM THE FOLLOWING
PATH:
```

If the installation and configuration were not successful, then the output is not generated and a message like the following is displayed:

```
NOT ALL THE IMAGES WERE UPLOADED CORRECTLY, CHECK FOR HADOOP LOGS
```

1.6.3.2 Image Processor Test Script (Mosaicking)

This script executes the processor job by setting three source rasters of Hawaii islands and some coordinates that includes all three. The job will create a mosaic based on these coordinates and resulting raster should include the three rasters combined in a single one.

`runimageloader.sh` should be executed as a prerequisite, so that the source rasters exist in HDFS. These are 3 band rasters of byte data type.

No parameters are required for OBDA environments, and a single parameter "-s" with the $ALL_ACCESS_FOLDER value is required for non-OBDA environments.

Additionally, if the output should be stored in HDFS, the "-o" parameters must be used to set the HDFS folder where the mosaic output will be stored.

Internally the job filters the tiles using the coordinates specified in the configuration input, xml, only the required tiles are processed in a mapper and finally in the reduce phase, all of them are put together into the resulting mosaic raster.

The test script can be found here:

```
/oracle/oracle-spatial-graph/raster/tests/runimageprocessor.sh
```

For ODBA environments, enter:

```
./runimageprocessor.sh
```

For non-ODBA environments, enter:

```
./runimageprocessor.sh -s ALL_ACCESS_FOLDER
```

Upon successful execution, the message `EXPECTED OUTPUT FILE IS` is displayed, with the path in HDFS where the files are located (this path depends on the definition of ALL_ACCESS_FOLDER) and a list of the created images and thumbnails on HDFS. The output may include:

```
ALL_ACCESS_FOLDER/processtest/hawaiimosaic.tif
total 9452
drwxrwxrwx 2 hdfs    hdfs    4096 Sep 10 09:12 .
drwxrwxrwx 9 zherena dba     4096 Sep  9 13:50 ..
-rwxrwxrwx 1 yarn    yarn 4741101 Sep 10 09:12 hawaiimosaic.tif

MOSAIC IMAGE GENERATED
----------------------------------------------------------------------
YOU MAY VISUALIZE THE MOSAIC OUTPUT IMAGE FOR REVIEW IN THE FOLLOWING PATH:
ALL_ACCESS_FOLDER/processtest/hawaiimosaic.tif”
```

If the installation and configuration were not successful, then the output is not generated and a message like the following is displayed:

```
MOSAIC WAS NOT SUCCESSFULLY CREATED, CHECK HADOOP LOGS TO REVIEW THE PROBLEM
```

To test the output storage in HDFS, use the following command

For ODBA environments, enter:

```
./runimageprocessor.sh -o hdfstest
```

For non-ODBA environments, enter:

```
./runimageprocessor.sh -s ALL_ACCESS_FOLDER -o hdfstest
```

1.6.3.3 Single-Image Processor Test Script

This script executes the processor job for a single raster, in this case is a DEM source raster of North Napa Valley. The purpose of this job is process the complete input by using the user processing classes configured for the mapping phase. This class calculates the hillshade of the DEM, and this is set to the output file. No mosaic operation is performed here.

`runimageloader.sh` should be executed as a prerequisite, so that the source raster exists in HDFS. This is 1 band of float 32 datatype DEM rasters..

No parameters are required for OBDA environments, and a single parameter "-s" with the $ALL_ACCESS_FOLDER value is required for non-OBDA environments.

The test script can be found here:

```
/oracle/oracle-spatial-graph/raster/tests/runsingleimageprocessor.sh
```

For ODBA environments, enter:

```
./runsingleimageprocessor.sh
```

For non-ODBA environments, enter:

```
./runsingleimageprocessor.sh -s ALL_ACCESS_FOLDER
```

Upon successful execution, the message `EXPECTED OUTPUT FILE: ALL_ACCESS_FOLDER/processtest/NapaSlope.tif` is displayed, with the path in HDFS where the files are located (this path depends on the definition of ALL_ACCESS_FOLDER) and a list of the created images and thumbnails on HDFS. The output may include:

```
EXPECTED OUTPUT FILE: ALL_ACCESS_FOLDER/processtest/NapaDEM.tif
total 4808
drwxrwxrwx 2 hdfs    hdfs    4096 Sep 10 09:42 .
drwxrwxrwx 9 zherena dba     4096 Sep  9 13:50 ..
-rwxrwxrwx 1 yarn    yarn 4901232 Sep 10 09:42 NapaDEM.tif
IMAGE GENERATED
----------------------------------------------------------------------

YOU MAY VISUALIZE THE OUTPUT IMAGE FOR REVIEW IN THE FOLLOWING PATH:
ALL_ACCESS_FOLDER/processtest/NapaDEM.tif”
```

If the installation and configuration were not successful, then the output is not generated and a message like the following is displayed:

```
MOSAIC WAS NOT SUCCESSFULLY CREATED, CHECK HADOOP LOGS TO REVIEW THE PROBLEM
```

1.6.3.4 Image Processor DEM Test Script

This script executes the processor job by setting a DEM source raster of North Napa Valley and some coordinates that surround it. The job will create a mosaic based on these coordinates and will also calculate the slope on it by setting a processing class in the mosaic configuration XML.

`runimageloader.sh` should be executed as a prerequisite, so that the source rasters exist in HDFS. These are 3 band rasters of byte data type.

No parameters are required for OBDA environments, and a single parameter "-s" with the $ALL_ACCESS_FOLDER value is required for non-OBDA environments.

The test script can be found here:

```
/oracle/oracle-spatial-graph/raster/tests/runimageprocessordem.sh
```

For ODBA environments, enter:

```
./runimageprocessordem.sh
```

For non-ODBA environments, enter:

```
./runimageprocessordem.sh -s ALL_ACCESS_FOLDER
```

Upon successful execution, the message `EXPECTED OUTPUT FILE: ALL_ACCESS_FOLDER/processtest/NapaSlope.tif` is displayed, with the path in HDFS where the files are located (this path depends on the definition of ALL_ACCESS_FOLDER) and a list of the created images and thumbnails on HDFS. The output may include:

```
EXPECTED OUTPUT FILE: ALL_ACCESS_FOLDER/processtest/NapaSlope.tif
total 4808
drwxrwxrwx 2 hdfs    hdfs    4096 Sep 10 09:42 .
drwxrwxrwx 9 zherena dba     4096 Sep  9 13:50 ..
-rwxrwxrwx 1 yarn    yarn 4901232 Sep 10 09:42 NapaSlope.tif
MOSAIC IMAGE GENERATED
----------------------------------------------------------------------

YOU MAY VISUALIZE THE MOSAIC OUTPUT IMAGE FOR REVIEW IN THE FOLLOWING PATH:
ALL_ACCESS_FOLDER/processtest/NapaSlope.tif”
```

If the installation and configuration were not successful, then the output is not generated and a message like the following is displayed:

```
MOSAIC WAS NOT SUCCESSFULLY CREATED, CHECK HADOOP LOGS TO REVIEW THE PROBLEM
```

You may also test the “if” algebra function, where every pixel in this raster with value greater than 2500 will be replaced by the value you set in the command line using the “–c” flag. For example:

For ODBA environments, enter:

```
./runimageprocessordem.sh –c 8000
```

For non-ODBA environments, enter:

```
./runimageprocessordem.sh -s ALL_ACCESS_FOLDER –c 8000
```

You can visualize the output file and notice the difference between simple slope calculation and this altered output, where the areas with pixel values greater than 2500 look more clear.

1.7 Installing and Configuring the Big Data Spatial Image Server

You can access the image processing framework through the Oracle Big Data Spatial Image Server, which provides a web interface for loading and processing images.

Installing and configuring the Spatial Image Server depends upon the distribution being used.

- Installing and Configuring the Image Server for Oracle Big Data Appliance
- Installing and Configuring the Image Server Web for Other Systems (Not Big Data Appliance)

After you perform the installation, verify it (see Post-installation Verification Example for the Image Server Console).

1.7.1 Installing and Configuring the Image Server for Oracle Big Data Appliance

To perform an automatic installation using the provided script, you can perform these steps:

1. Run the following script:

   ```
   sudo /home/osg/configure-jetty/install-jetty-bdsg.sh
   ```

   If the active nodes have changed since the installation, update the configuration file.

2. Start the Jetty server:

   ```
   cd /opt/oracle/oracle-spatial-graph/spatial/jetty
   java -jar start.jar
   ```

If you need more information or need to perform other actions, see the following topics:

- Prerequisites for Installing the Image Server on Oracle Big Data Appliance
- Installing the Image Server Web on an Oracle Big Data Appliance
- Configuring the Environment

1.7.1.1 Prerequisites for Installing the Image Server on Oracle Big Data Appliance

Ensure that you have the prerequisite software installed.

1. Download the latest Jetty core component binary from the Jetty download page `http://www.eclipse.org/jetty/downloads.php` onto the Oracle DBA Resource Manager node.

2. Unzip the `imageserver.war` file into the jetty `webapps` directory or any other directory of choice as follows:

   ```
   unzip /opt/oracle/oracle-spatial-graph/spatial/jlib/
   imageserver.war -d $JETTY_HOME/webapps/imageserver
   ```

   Note:

   The directory or location under which you unzip the file is known as `$JETTY_HOME` in this procedure.

   ---

3. Copy Hadoop dependencies as follows:

   ```
   cp /opt/cloudera/parcels/CDH/lib/hadoop/client/* $JETTY_HOME/
   webapps/imageserver/WEB-INF/lib/
   ```

   Note:

   If the installation of Jetty is done on a non-Oracle DBA cluster, then replace `/opt/cloudera/parcels/CDH/lib/hadoop/` with the actual Hadoop library path, which by default is `/usr/lib/hadoop`.

1.7.1.2 Installing the Image Server Web on an Oracle Big Data Appliance

1. Copy the `gdal.jar` file under `/opt/oracle/oracle-spatial-graph/spatial/jlib/gdal.jar` to `$JETTY_HOME/lib/ext`.
2. Copy the `asm-3.1.jar` file under `/opt/oracle/oracle-spatial-graph/spatial/raster/jlib/asm-3.1.jar` to `$JETTY_HOME/webapps/imageserver/WEB-INF/lib`.

Note:

The `jersey-core*` jars will be duplicated at `$JETTY_HOME/webapps/imageserver/WEB-INF/lib`. Make sure you remove the old ones and leave just `jersey-core-1.17.1.jar` in the folder, as in the next step.

---

3. Enter the following command:

   ```
   ls -lat jersey-core*
   ```

4. Delete the listed libraries, except do **not** delete `jersey-core-1.17.1.jar`.
5. In the same directory ($JETTY_HOME/webapps/imageserver/WEB-INF/lib, delete the xercesImpl jar files:

   ```
    rm xercesImpl*
   ```

6. Start the Jetty server by running: `java -Djetty.deploy.scanInterval=0 -jar start.jar`

   If you need to change the port, specify it. For example: `jetty.http.port=8081`

   Ignore any warnings, such as the following:

   ```
   java.lang.UnsupportedOperationException:  setXIncludeAware is not supported on
   this JAXP implementation or earlier: class
   oracle.xml.jaxp.JXDocumentBuilderFactory
   ```

1.7.1.3 Configuring the Environment

1. Type the `http://thehost:8080/imageserver/console.jsp` address in your browser address bar to open the console.
2. Log in to the console using the credentials you created in "Installing the Image Server Web on an Oracle Big Data Appliance."
3. From the Configuration tab in the **Hadoop Configuration Parameters** section, depending on the cluster configuration change three properties:
   a. `fs.defaultFS`: Type the active `namenode` of your cluster in the format `hdfs://<namenode>:8020` (Check with the administrator for this information).
   b. `yarn.resourcemanager.scheduler.address`: Active Resource manager of your cluster. `<shcedulername>:8030.` This is the Scheduler address.
   c. `yarn.resourcemanager.address`: Active Resource Manager address in the format `<resourcename>:8032`

Note:

Keep the default values for the rest of the configuration. They are pre-loaded for your Oracle Big Data Appliance cluster environment.

---

4. Click **Apply Changes** to save the changes.

   Tip:

   You can review the missing configuration information under the Hadoop Loader tab of the console.

5. From the Configuration tab in the **Global Init Parameters** section, depending on the cluster configuration change these properties:

   a. `shared.gdal.data`: Specify the gdal shared data folder. Follow the instructions specified in Installing the Image Processing Framework for Other Distributions (Not Oracle Big Data Appliance).

   b. `gdal.lib`: Location of the gdal .so libraries.

   c. `start`: Specify a shared folder to start browsing the images. This folder must be shared between the cluster and NFS mountpoint (SHARED_FOLDER).

   d. `saveimages`: Create a child folder named saveimages under `start` (SHARED-FOLDER) with full write access. For example, if start=/home, then `saveimages=/home/saveimages`.

   e. `nfs.mountpoint`: If the cluster requires a mount point to access the SHARED_FOLDER, specify a mount point. For example, `/net/home`. Otherwise, leave it blank.

   f. `yarn.application.classpath`: The classpath for the Hadoop to find the required jars and dependencies. Usually this is under `/usr/lib/hadoop`. For example:

```
/etc/hadoop/conf/,/usr/lib/hadoop/*,/usr/lib/hadoop/lib/*,/usr/lib/hadoop-
hdfs/*,/usr/lib/hadoop-hdfs/lib/*,/usr/lib/hadoop-yarn/*,/usr/lib/hadoop-
yarn/lib/*,/usr/lib/hadoop-mapreduce/*,/usr/lib/hadoop-mapreduce/lib/*
```

Note:

Keep the default values for the rest of the configuration. They are pre-loaded for your Oracle Big Data Appliance cluster environment.

---

6. Click **Apply Changes** to save the changes.

   Tip:

   You can review the missing configuration information under the Hadoop Loader tab of the console.

1.7.2 Installing and Configuring the Image Server Web for Other Systems (Not Big Data Appliance)

Follow the instructions in this topic.

- Prerequisites for Installing the Image Server on Other Systems
- Installing the Image Server Web on Other Systems
- Configuring the Environment

1.7.2.1 Prerequisites for Installing the Image Server on Other Systems

- Follow the instructions specified in "Prerequisites for Installing the Image Processing Framework for Other Distributions."
- Follow the instructions specified in "Installing the Image Processing Framework for Other Distributions."
- Follow the instructions specified in "Configuring the Environment."

1.7.2.2 Installing the Image Server Web on Other Systems

- Follow the instructions specified in "Prerequisites for Installing the Image Server on Oracle Big Data Appliance."
- Follow the instructions specified in "Installing the Image Server Web on an Oracle Big Data Appliance."
- Follow the instructions specified in "Configuring the Environment."

1.7.2.3 Configuring the Environment

1. Type the `http://thehost:8080/imageserver/console.jsp` address in your browser address bar to open the console.
2. Log in to the console using the credentials you created in "Installing the Image Server Web on an Oracle Big Data Appliance."
3. From the Configuration tab in the Hadoop Configuration Parameters section, depending on the cluster configuration change these three properties
    a. Specify a shared folder to start browsing the images. This folder must be shared between the cluster and NFS mountpoint (SHARED_FOLDER).
    b. Create a child folder named **`saveimages`** under Start with full write access. For example, if Start=/home, then `saveimages=/home/saveimages`.
    c. If the cluster requires a mount point to access the SHARED_FOLDER, specify a mount point. For example, `/net/home`. Else, leave it blank and proceed.
    d. Type the folder path that contains the Hadoop native libraries and additional libraries (`HADOOP_LIB_PATH`).
    e. yarn.application.classpath: Type the classpath for the Hadoop to find the required jars and dependencies. Usually this is under `/usr/lib/hadoop`.

    Note:

    Keep the default values for the rest of the configuration. They are pre-loaded for your Oracle Big Data Appliance cluster environment.

4. Click Apply Changes to save the changes.

Tip:

You can review the missing configuration information under the Hadoop Loader tab of the console.

1.7.3 Post-installation Verification Example for the Image Server Console

In this example, you will:

- Load the images from local server to HDFS Hadoop cluster.
- Run a job to create a mosaic image file and a catalog with several images.
- View the mosaic image.

Related subtopics:

- Loading Images from the Local Server to the HDFS Hadoop Cluster
- Creating a Mosaic Image and Catalog
- Creating a Mosaic Directly from the Globe
- Removing Identical Rasters

1.7.3.1 Loading Images from the Local Server to the HDFS Hadoop Cluster

1. Open (`http://<hostname>:8080/imageserver/console.jsp`) the Image Server Console.
2. Log in using the default user/password as **`admin`**/**`admin`**.
3. Go to the **`Hadoop Loader`** tab.
4. Click **`Open`** and browse to the `demo` folder that contains a set of Hawaii images. They can be found at `/opt/shareddir/spatial/demo/imageserver/images`.
5. Select the `images` folder and click **`Load images`**.

   Wait for the message, 'Images loaded successfully'.

Note:

If no errors were shown, then you have successfully installed the Image Loader web interface.

1.7.3.2 Creating a Mosaic Image and Catalog

1. Go to the **`Raster Image processing`** tab.
2. From the Catalog menu select **`Catalog`** > **`New Catalog`** > **`HDFS Catalog`**.

   A new catalog is created.
3. From the Imagery menu select **`Imagery`** > **`Add hdfs image`**.
4. Browse the HDFS host and add images.

   A new file tree gets created with all the images you just loaded from your host.

5. Browse the `newdata` folder and verify the images.

6. Select the images listed in the pre-visualizer and add click **`Add`**.

   The images are added to the bottom sub-panel.

7. Click **`Add images`**.

   The images are added to the main catalog.

8. Save the catalog.

9. From the Imagery menu select **`Imagery`** > **`Mosaic`**.

10. Click **`Load default configuration file`**, browse to `/opt/shareddir/spatial/demo/imageserver` and select `testFS.xml`.

    Note:

    The default configuration file `testFS.xml` is included in the demo.

11. Click **`Create Mosaic`**.

    Wait for the image to be created.

12. Optionally, to download and view the image click **`Download`**.

1.7.3.3 Creating a Mosaic Directly from the Globe

1. Go to the **`Raster Image Loader`** tab.

2. Click **`Refresh Footprint`** and wait until all footprints are displayed on the panel..

3. Click**`Select Footprints`** , then select the desired area, zooming in or out as necessary.

4. Remove or ignore rasters as necessary.

   If identical rasters are in the result, they are shown in yellow

5. Right-click on the map and select **`Generate Mosaic`**.

6. Specify the output folder in which to place the mosaic, or load an existing configuration file.

7. If you want to add an operation on every pixel in the mosaic, click **`Advanced Configuration`**..

8. Click **`Create Mosaic`**, and wait for the result.

9. From the Imagery menu select **`Imagery`** > **`Mosaic`**.

10. If you need to remove the selection, click the red circle in the upper-left corner of the map.

Note:

If you requested the mosaic to be created on HDFS, you must wait until the image is loaded on HDFS.

---

11. Optionally, to download and view the image, click **`Download`**.

1.7.3.4 Removing Identical Rasters

1. Go to the **`Hadoop Loader`** tab.
2. Click **`Refresh Footprint`** and wait until all footprints are displayed on the panel..

   If identical rasters are in the result, they are shown in yellow.
3. For each pair of identical rasters, if you want to select one of them for removal, right-click on its yellow box.

   A new dialog box is displayed.
4. To remove a raster, click the X button for it.
5. To see the thumbnail, click in the image.

1.7.4 Using the Provided Image Server Web Services

The image server has two ready-to-use web services, one for the HDFS loader and the other for the HDFS mosaic processor.

These services can be called from a Java application. They are currently supported only for GET operations. The formats for calling them are:

- **Loader**: `http://host:port/imageserver/rest/hdfsloader?path=string&overlap=string` where:

  `path`: The images to be processed; can be a the path of a single file, or of one or more whole folders. For more than one folder, use commas to separate folder names.

  `overlap` (optional): The overlap between images (default = 10).

- **Mosaic**: `http://host:port/imageserver/rest/mosaic?mosaic=string&config=string` where:

  `mosaic`: The XML mosaic file that contains the images to be processed. If you are using the image server web application, the XML file is generated automatically. Example of a mosaic XML file:

  ```
  <?xml version='1.0'?>
  <catalog type='HDFS'>
      <image>
         <source>Hadoop File System</source>
         <type>HDFS</type>
         <raster> /hawaii.tif.ohif</raster>
          <bands datatype='1' config='1,2,3'>3</bands>
      </image>
      <image>
         <source>Hadoop File System</source>
         <type>HDFS</type>
  ```

```
       <raster>/ /kahoolawe.tif.ohif</raster>
        <bands datatype='1' config='1,2,3'>3</bands>
    </image>
</catalog>
```

`config`: Configuration file; created the first time a mosaic is processed using the image server web application. Example of a configuration file

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<mosaic>
    <output>
        <SRID>26904</SRID>
        <directory type = "FS">/net/system123/scratch/user3/installers</directory>
        <tempFsFolder>/net/system123/scratch/user3/installers</tempFsFolder>
        <filename>test</filename>
        <format>GTIFF</format>
        <width>1800</width>
        <height>1406</height>
        <algorithm order = "0">1</algorithm>
        <bands layers = "3"/>
        <nodata>#000000</nodata>
        <pixelType>1</pixelType>
    </output>
    <crop>

<transform>294444.1905688362,114.06068372059636,0,2517696.9179752027,0,-114.060683
72059636</transform>
    </crop>
    <process/>
    <operations>
        <localnot/>
    </operations>
</mosaic>
```

Java Example: Using the Loader

```
public class RestTest
    public static void main(String args[]) {

        try {
            // Loader http://localhost:7101/imageserver/rest/hdfsloader?
path=string&overlap=string
            // Mosaic http://localhost:7101/imageserver/rest/mosaic?
mosaic=string&config=string
            String path = "/net/system123/scratch/user3/installers/hawaii/
hawaii.tif";

            URL url = new URL(
                    "http://system123.example.com:7101/imageserver/rest/hdfsloader?
path=" +
                            path + "&overlap=2"); // overlap its optional

            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            //conn.setRequestProperty("Accept", "application/json");

            if (conn.getResponseCode() != 200) {
                throw new RuntimeException("Failed : HTTP error code : "
                        + conn.getResponseCode());
            }

            BufferedReader br = new BufferedReader(new InputStreamReader(
```

```
                    (conn.getInputStream()))));

            String output;
            System.out.println("Output from Server .... \n");
            while ((output = br.readLine()) != null) {
                System.out.println(output);
            }

            conn.disconnect();

        } catch (MalformedURLException e) {

            e.printStackTrace();

        } catch (IOException e) {

            e.printStackTrace();

        }
    }
}
```

Java Example: Using the Mosaic Processor

```
public class NetClientPost {
    public static void main(String[] args) {

      try {
          String mosaic = "<?xml version='1.0'?>\n" +
                    "<catalog type='HDFS'>\n" +
                    "    <image>\n" +
                    "       <source>Hadoop File System</source>\n" +
                    "       <type>HDFS</type>\n" +
                    "       <raster>/user/hdfs/newdata/net/system123/scratch/user3/
installers/hawaii/hawaii.tif.ohif</raster>\n" +
                    "       <url>http://system123.example.com:7101/imageserver/temp/
862b5871973372aab7b62094c575884ae13c3a27_thumb.jpg</url>\n" +
                    "       <bands datatype='1' config='1,2,3'>3</bands>\n" +
                    "    </image>\n" +
                    "</catalog>";
             String config = "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=
\"no\"?>\n" +
                     "<mosaic>\n" +
                     "<output>\n" +
                     "<SRID>26904</SRID>\n" +
                     "<directory type=\"FS\">/net/system123/scratch/user3/
installers</directory>\n" +
                     "<tempFsFolder>/net/system123/scratch/user3/installers</
tempFsFolder>\n" +
                     "<filename>test</filename>\n" +
                     "<format>GTIFF</format>\n" +
                     "<width>1800</width>\n" +
                     "<height>1269</height>\n" +
                     "<algorithm order=\"0\">1</algorithm>\n" +
                     "<bands layers=\"3\"/>\n" +
                     "<nodata>#000000</nodata>\n" +
                     "<pixelType>1</pixelType>\n" +
                     "</output>\n" +
                     "<crop>\n" +

"<transform>739481.1311601736,130.5820811245199,0,2254053.5858749463,0,-130.582081124
```

```
5199</transform>\n" +
                     "</crop>\n" +
                     "<process/>\n" +
                     "</mosaic>";
             System.out.println ("asdf");
             URL url2 = new URL("http://192.168.1.67:8080" );
             HttpURLConnection conn2 = (HttpURLConnection) url2.openConnection();
             conn2.setRequestMethod("GET");
             if (conn2.getResponseCode() != 200 ) {
                throw new RuntimeException("Failed : HTTP error code : "
                    + conn2.getResponseCode());
            }
        /*URL url = new URL("http://system123.example.com:7101/imageserver/rest/
mosaic?" +("mosaic=" + URLEncoder.encode(mosaic, "UTF-8") + "&config=" +
                URLEncoder.encode(config, "UTF-8")));
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");

        if (conn.getResponseCode() != 200 ) {
            throw new RuntimeException("Failed : HTTP error code : "
                + conn.getResponseCode());
        }
        BufferedReader br = new BufferedReader(new InputStreamReader(
                (conn.getInputStream())));
        String output;System.out.println("Output from Server .... \n");
        while ((output = br.readLine()) != null)
            System.out.println(output);
        conn.disconnect();*/

      } catch (MalformedURLException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
     }
    }
}
```

1.8 Installing Oracle Big Data Spatial Hadoop Vector Console

To install the Oracle Big Data Spatial Hadoop vector console, follow the instructions in this topic.

- Assumptions and Prerequisite Libraries
- Installing Spatial Hadoop Vector Console on Oracle Big Data Appliance
- Installing Spatial Hadoop Vector Console for Other Systems (Not Big Data Appliance)
- Configuring Spatial Hadoop Vector Console on Oracle Big Data Appliance
- Configuring Spatial Hadoop Vector Console for Other Systems (Not Big Data Appliance)

1.8.1 Assumptions and Prerequisite Libraries

The following assumptions and prerequisites apply for installing and configure the Spatial Hadoop Vector Console.

1.8.1.1 Assumptions

- The API and jobs described here run on a Cloudera CDH5.7, Hortonworks HDP 2.4, or similar Hadoop environment.
- Java 8 or newer versions are present in your environment.

1.8.1.2 Prerequisite Libraries

In addition to the Hadoop environment jars, the libraries listed here are required by the Vector Analysis API.

```
sdohadoop-vector.jar
sdoutil.jar
sdoapi.jar
ojdbc.jar
commons-fileupload-1.3.1.jar
commons-io-2.4.jar
jackson-annotations-2.1.4.jar
jackson-core-2.1.4.jar
jackson-core-asl-1.8.1.jar
jackson-databind-2.1.4.jar
javacsv.jar
lucene-analyzers-common-4.6.0.jar
lucene-core-4.6.0.jar
lucene-queries-4.6.0.jar
lucene-queryparser-4.6.0.jar
mvsuggest_core.jar
```

1.8.2 Installing Spatial Hadoop Vector Console on Oracle Big Data Appliance

You can install the Spatial Hadoop vector console on Big Data Appliance either by using the provided script or by performing a manual configuration.. To use the **provided script**:

1. Run the following script:

   ```
   sudo /home/osg/configure-jetty/install-jetty-bdsg.sh
   ```

   If the active nodes have change after the installation, then update the configuration file as described in Configuring Spatial Hadoop Vector Console on Oracle Big Data Appliance.

2. Start the Jetty server:

   ```
   cd /opt/oracle/oracle-spatial-graph/spatial/jetty
   java -jar start.jar
   ```

To perform a **manual configuration**, follow these steps.

1. Download the latest Jetty core component binary from the Jetty download page `http://www.eclipse.org/jetty/downloads.php` onto the Oracle DBA Resource Manager node.
2. Unzip the `spatialviewer.war` file into the jetty `webapps` directory as follows:

   ```
   unzip /opt/oracle/oracle-spatial-graph/spatial/vector/
   console/spatialviewer.war -d $JETTY_HOME/webapps/
   spatialviewer
   ```

Note:

The directory or location under which you unzip the file is known as `$JETTY_HOME` in this procedure.

---

3. Copy Hadoop dependencies as follows:

   ```
   cp /opt/cloudera/parcels/CDH/lib/hadoop/client/* $JETTY_HOME/
   webapps/spatialviewer/WEB-INF/lib/
   ```

4. Complete the configuration steps mentioned in the "Configuring Spatial Hadoop Vector Console on Oracle Big Data Appliance."

5. Start the jetty server. `$JETTY_HOME/java -Djetty.deploy.scanInterval=0 -jar start.jar`

Optionally, upload sample data (used with examples in other topics) to HDFS:

```
sudo -u hdfs hadoop fs -mkdir /user/oracle/bdsg

sudo -u hdfs hadoop fs -put /opt/oracle/oracle-spatial-graph/spatial/vector/examples/
data/tweets.json /user/oracle/bdsg/
```

1.8.3 Installing Spatial Hadoop Vector Console for Other Systems (Not Big Data Appliance)

Follow the steps for manual configuration described in "Installing Spatial Hadoop Vector Console on Oracle Big Data Appliance." However, in step 3 replace the path `/opt/cloudera/parcels/CDH/lib/` with the actual library path, which by default is `/usr/lib/`.

1.8.4 Configuring Spatial Hadoop Vector Console on Oracle Big Data Appliance

1. Edit the configuration file `$JETTY_HOME/webapps/spatialviewer/conf/console-conf.xml` to specify your own data for sending email and for other configuration parameters.

   Follow these steps with the configuration parameters

   a. Edit the Notification URL: This is the URL where the console server is running. It has to be visible to the Hadoop cluster to notify the end of the jobs. This is an example settings: `<baseurl>http:// hadoop.console.url:8080</baseurl>`

   b. Edit the directory with temporary hierarchical indexes: an HDFS path that will contain temporary data on hierarchical relationships. Example:: `<hierarchydataindexpath>hdfs://hadoop.cluster.url:8020/user/myuser/hierarchyIndexPath</hierarchydataindexpath>`

   c. Edit the HDFS directory that will contain the MVSuggest generated index. Example: `<mvsuggestindex> hdfs://hadoop.cluster.url:8020/user/myuser /mvSuggestIndex</mvsuggestindex>`

   d. If necessary, edit the URL used to get the eLocation background maps. Example: `<elocationmvbaseurl>http://elocation.oracle.com/mapviewer</elocationmvbaseurl>`

e. Edit the HDFS directory that will contain the index metadata. Example: `<indexmetadatapath>hdfs:// hadoop.cluster.url:8020/user/myuser/indexMetadata</indexmetadatapath>`

f. Edit the HDFS directory with temporary data used by the explore data processes. Example: `<exploretempdatapath>hdfs://hadoop.cluster.url:8020/user/myuser/exploreTmp<exploretempdatapath>`

g. Edit the eneral Hadoop jobs configuration: The console uses two Hadoop jobs. The first is used to create a spatial index on existing files in HDFS and the second is used to generate displaying results based on the index. One part of the configuration is common to both jobs and another is specific to each job. The common configuration can be found within the `<hadoopjobs><configuration>` elements. An example configuration is given here:

```
<hadoopjobs>
   <configuration>
           <property>
    <!--hadoop user. The user is a mandatory property.-->
             <name>hadoop.job.ugi</name>
             <value>hdfs</value>
           </property>

           <property>
    <!-- like defined in core-site.xml
    If in core-site.xml the path fs.defaultFS is define as the nameservice
ID
    (High Availability configuration) then set the full address and IPC
port
    of the currently active name node. The service is define in the file
hdfs-site.xml.-->
             <name>fs.defaultFS</name>
             <value>hdfs://hadoop.cluster.url:8020</value>
          </property>

          <property>
    <!-- like defined in mapred-site.xml -->
           <name>mapreduce.framework.name</name>
           <value>yarn</value>
         </property>

         <property>
    <!-- like defined in yarn-site.xml -->
           <name>yarn.resourcemanager.scheduler.address</name>
           <value>hadoop.cluster.url:8030</value>
        </property>

        <property>
    <!-- like defined in yarn-site.xml -->
            <name>yarn.resourcemanager.address</name>
            <value>hadoop.cluster.url:8032</value>
        </property>

        <property>
    <!-- like defined in yarn-site.xml (full path) -->
            <name>yarn.application.classpath</name>
           <value>/etc/hadoop/conf/,/opt/cloudera/parcels/CDH/lib/
hadoop/*,/opt/cloudera/parcels/CDH/lib/hadoop/lib/*,/opt/cloudera/
```

```
parcels/CDH/lib/hadoop-hdfs/*,/opt/cloudera/parcels/CDH/lib/hadoop-
hdfs/lib/*,/opt/cloudera/parcels/CDH/lib/hadoop-yarn/*,/opt/cloudera/
parcels/CDH/lib/hadoop-yarn/lib/*,/opt/cloudera/parcels/CDH/lib/hadoop-
mapreduce/*,/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/lib/*</value>
         </property>
      </configuration>
    <hadoopjobs>
```

2. Create an index job specific configuration. Additional Hadoop parameters can be specified for the job that creates the spatial indexes. An example additional configuration is:

```
<hadoopjobs>
   <configuration>
   ...
   </configuration>
      <indexjobadditionalconfiguration>
         <property>
         <!-- Increase the mapred.max.split.size, so that less mappers are
allocated in slot and thus reduces the mapper initializing overhead. -->
            <name>mapred.max.split.size</name>
            <value>1342177280</value>
           </property>
      </indexjobadditionalconfiguration>
<hadoopjobs>
```

3. Create a specific configuration for the job that generates the categorization results. The following is an example of property settings:

```
<hadoopjobs>
  <configuration>
   ...
  </configuration>

     <indexjobadditionalconfiguration>
      ...
     </indexjobadditionalconfiguration>

     <hierarchicaljobadditionalconfiguration>
        <property>
        <!-- Increase the mapred.max.split.size, so that less mappers are
allocated in slot and thus reduces the mapper initializing overhead. -->
           <name>mapred.max.split.size</name>
           <value>1342177280</value>
         </property>
      </hierarchicaljobadditionalconfiguration>
<hadoopjobs>
```

4. Specify the Notification emails: The email notifications are sent to notify about the job completion status. This is defined within the `<notificationmails>` element. It is mandatory to specify a user (`<user>`), password (`<password>`) and sender email (`<mailfrom>`). In the `<configuration>` element, the configuration properties needed for the Java Mail must be set. This example is a typical configuration to send mails via SMTP server using a SSL connection:

```
<notificationmails>
  <!--Authentication parameters. The Authentication parameters are mandatory.-->
    <user>user@mymail.com</user>
    <password>mypassword</password>
    <mailfrom>user@mymail.com</mailfrom>
```

```
    <!--Parameters that will be set to the system properties. Below the
parameters needed to send mails via SMTP server using a SSL connection.    -->

    <configuration>
       <property>
         <name>mail.smtp.host</name>
         <value>mail.host.com</value>
       </property>

       <property>
         <name>mail.smtp.socketFactory.port</name>
         <value>myport</value>
       </property>

       <property>
         <name>mail.smtp.socketFactory.class</name>
         <value>javax.net.ssl.SSLSocketFactory</value>
       </property>

       <property>
         <name>mail.smtp.auth</name>
         <value>true</value>
       </property>
    </configuration>
</notificationmails>
```

1.8.5 Configuring Spatial Hadoop Vector Console for Other Systems (Not Big Data Appliance)

Follow the steps mentioned in "Configuring Spatial Hadoop Vector Console on Oracle Big Data Appliance." However, in the step (`General Hadoop Job Configuration`), in the Hadoop property `yarn.application.classpath` replace the `/opt/cloudera/parcels/CDH/lib/` with the actual library path, which by default is `/usr/lib/`.

1.9 Installing Property Graph Support on a CDH Cluster or Other Hardware

You can use property graphs on either Oracle Big Data Appliance or commodity hardware.

- Apache HBase Prerequisites
- Property Graph Installation Steps
- About the Property Graph Installation Directory
- Optional Installation Task for In-Memory Analyst Use

See Also:

Configuring Property Graph Support

1.9.1 Apache HBase Prerequisites

The following prerequisites apply to installing property graph support in HBase.

- Linux operating system
- Cloudera's Distribution including Apache Hadoop (CDH)

  For the software download, see: `http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html`
- Apache HBase
- Java Development Kit

Details about supported versions of these products, including any interdependencies, will be provided in a My Oracle Support note.

1.9.2 Property Graph Installation Steps

To install property graph support, follow these steps.

1. Unzip the software package:

   ```
   rpm -i oracle-spatial-graph-<version>.x86_64.rpm
   ```

   By default, the software is installed in the following directory: `/opt/oracle/`

   After the installation completes, the `opt/oracle/oracle-spatial-graph` directory exists and includes a `property_graph` subdirectory.

2. Set the JAVA_HOME environment variable. For example:

   ```
   setenv JAVA_HOME  /usr/local/packages/jdk7
   ```

3. Set the PGX_HOME environment variable. For example:

   ```
   setenv PGX_HOME /opt/oracle/oracle-spatial-graph/pgx
   ```

4. If HBase will be used, set the `HBASE_HOME` environment variable in all HBase region servers in the Apache Hadoop cluster. (`HBASE_HOME` specifies the location of the `hbase` installation directory.) For example:

   ```
   setenv HBASE_HOME /usr/lib/hbase
   ```

   Note that on some installations of Big Data Appliance, Apache HBase is placed in a directory like the following: `/opt/cloudera/parcels/CDH-5.3.3-1.cdh5.3.3.p0.5/lib/hbase/`

5. If HBase will be used, copy the data access layer library into $HBASE_HOME/lib. For example:

   ```
   cp /opt/oracle/oracle-spatial-graph/property_graph/lib/sdopgdal*.jar $HBASE_HOME/
   lib
   ```

6. Tune the HBase or Oracle NoSQL Database configuration, as described in other tuning topics.

7. Log in to Cloudera Manager as the `admin` user, and restart the HBase service. Restarting enables the Region Servers to use the new configuration settings.

1.9.3 About the Property Graph Installation Directory

The installation directory for Oracle Big Data Spatial and Graph property graph features has the following structure:

```
$ tree -dFL 2 /opt/oracle/oracle-spatial-graph/property_graph/
/opt/oracle/oracle-spatial-graph/property_graph/
|-- dal
|   |-- groovy
|   |-- opg-solr-config
|   `-- webapp
|-- data
|-- doc
|   |-- dal
|   `-- pgx
|-- examples
|   |-- dal
|   |-- pgx
|   `-- pyopg
|-- lib
|-- librdf
`-- pgx
    |-- bin
    |-- conf
    |-- groovy
    |-- scripts
    |-- webapp
    `-- yarn
```

1.9.4 Optional Installation Task for In-Memory Analyst Use

Follow this installation task if property graph support is installed on a client without Hadoop, and you want to read graph data stored in the Hadoop Distributed File System (HDFS) into the in-memory analyst and write the results back to the HDFS, and/or use Hadoop NextGen MapReduce (YARN) scheduling to start, monitor and stop the in-memory analyst.

- Installing and Configuring Hadoop
- Running the In-Memory Analyst on Hadoop

1.9.4.1 Installing and Configuring Hadoop

To install and configure Hadoop, follow these steps.

1. Download the tarball for a supported version of the Cloudera CDH.
2. Unpack the tarball into a directory of your choice. For example:

   ```
   tar xvf hadoop-2.5.0-cdh5.2.1.tar.gz -C /opt
   ```

3. Have the `HADOOP_HOME` environment variable point to the installation directory. For example.

   ```
   export HADOOP_HOME=/opt/hadoop-2.5.0-cdh5.2.1
   ```

4. Add `$HADOOP_HOME/bin` to the `PATH` environment variable. For example:

   ```
   export PATH=$HADOOP_HOME/bin:$PATH
   ```

5. Configure `$HADOOP_HOME/etc/hadoop/hdfs-site.xml` to point to the HDFS name node of your Hadoop cluster.
6. Configure `$HADOOP_HOME/etc/hadoop/yarn-site.xml` to point to the resource manager node of your Hadoop cluster.

7. Configure the `fs.defaultFS` field in `$HADOOP_HOME/etc/hadoop/core-site.xml` to point to the HDFS name node of your Hadoop cluster.

1.9.4.2 Running the In-Memory Analyst on Hadoop

When running a Java application using in-memory analytics and HDFS, make sure that `$HADOOP_HOME/etc/hadoop` is on the classpath, so that the configurations get picked up by the Hadoop client libraries. However, you do not need to do this when using the in-memory analyst shell, because it adds `$HADOOP_HOME/etc/hadoop` automatically to the classpath if `HADOOP_HOME` is set.

You do not need to put any extra Cloudera Hadoop libraries (JAR files) on the classpath. The only time you need the YARN libraries is when starting the in-memory analyst as a YARN service. This is done with the `yarn` command, which automatically adds all necessary JAR files from your local installation to the classpath.

You are now ready to load data from HDFS or start the in-memory analyst as a YARN service. For further information about Hadoop, see the CDH 5.x.x documentation.

1.10 Installing and Configuring Multimedia Analytics Support

To use the Multimedia analytics feature, the video analysis framework must be installed and configured.

- Assumptions and Libraries for Multimedia Analytics
- Transcoding Software (Options)

1.10.1 Assumptions and Libraries for Multimedia Analytics

If you have licensed Oracle Big Data Spatial and Graph with Oracle Big Data Appliance, the video analysis framework for Multimedia analytics is already installed and configured. However, you must set `$MMA_HOME` to point to `/opt/oracle/oracle-spatial-graph/multimedia`.

Otherwise, you can install the framework on Cloudera CDH 5 or similar Hadoop environment, as follows:

1. Install the framework by using the following command on each node on the cluster:

   ```
   rpm2cpio oracle-spatial-graph-<version>.x86_64.rpm | cpio -idmv
   ```

2. Set `$MMA_HOME` to point to `/opt/oracle/oracle-spatial-graph/multimedia`.

3. Identify the locations of the following libraries:

   - Hadoop jar files (available in `$HADOOP_HOME/jars`)
   - Video processing libraries (see Transcoding Software (Options)
   - OpenCV libraries (available with the product)

4. If necessary, install the desired video processing software to transcode video data (see Transcoding Software (Options)).

1.10.2 Transcoding Software (Options)

The following options are available for transcoding video data:

- JCodec
- FFmpeg
- Third-party transcoding software

To use Multimedia analytics with JCodec (which is included with the product), when running the Hadoop job to recognize faces, set the `oracle.ord.hadoop.ordframegrabber` property to the following value: `oracle.ord.hadoop.decoder.OrdJCodecFrameGrabber`

To use Multimedia analytics with FFmpeg:

1. Download FFmpeg from: https://www.ffmpeg.org/.
2. Install FFmpeg on the Hadoop cluster.
3. Set the `oracle.ord.hadoop.ordframegrabber` property to the following value: `oracle.ord.hadoop.decoder.OrdFFMPEGFrameGrabber`

To use Multimedia analytics with custom video decoding software, implement the abstract class `oracle.ord.hadoop.decoder.OrdFrameGrabber`. See the Javadoc for more details

2

Using Big Data Spatial and Graph with Spatial Data

This chapter provides conceptual and usage information about loading, storing, accessing, and working with spatial data in a Big Data environment.

- About Big Data Spatial and Graph Support for Spatial Data
- Oracle Big Data Vector and Raster Data Processing
- Oracle Big Data Spatial Hadoop Image Processing Framework for Raster Data Processing
- Loading an Image to Hadoop Using the Image Loader
- Processing an Image Using the Oracle Spatial Hadoop Image Processor
- Loading and Processing an Image Using the Oracle Spatial Hadoop Raster Processing API
- Oracle Big Data Spatial Vector Analysis
- Using the Oracle Big Data Spatial and Graph Vector Console
- Using Oracle Big Data Spatial and Graph Image Server Console

2.1 About Big Data Spatial and Graph Support for Spatial Data

Spatial data represents the location characteristics of real or conceptual objects in relation to the real or conceptual space on a Geographic Information System (GIS) or other location-based application.

Oracle Big Data Spatial and Graph features enable spatial data to be stored, accessed, and analyzed quickly and efficiently for location-based decision making.

These features are used to geotag, enrich, visualize, transform, load, and process the location-specific two and three dimensional geographical images, and manipulate geometrical shapes for GIS functions.

- What is Big Data Spatial and Graph on Apache Hadoop?
- Advantages of Oracle Big Data Spatial and Graph
- Oracle Big Data Spatial Features and Functions
- Oracle Big Data Spatial Files, Formats, and Software Requirements

2.1.1 What is Big Data Spatial and Graph on Apache Hadoop?

Oracle Big Data Spatial and Graph on Apache Hadoop is a framework that uses the MapReduce programs and analytic capabilities in a Hadoop cluster to store, access, and analyze the spatial data. The spatial features provide a schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial data. Big Data Spatial and Graph on Hadoop supports storing and processing spatial images, which could be geometric shapes, raster, or vector images and stored in one of the several hundred supported formats.

Note:

Oracle Spatial and Graph Developer's Guide for an introduction to spatial concepts, data, and operations

2.1.2 Advantages of Oracle Big Data Spatial and Graph

The advantages of using Oracle Big Data Spatial and Graph include the following:

- Unlike some of the GIS-centric spatial processing systems and engines, Oracle Big Data Spatial and Graph is capable of processing both structured and unstructured spatial information.
- Customers are not forced or restricted to store only one particular form of data in their environment. They can have their data stored both as a spatial or nonspatial business data and still can use Oracle Big Data to do their spatial processing.
- This is a framework, and therefore customers can use the available APIs to custom-build their applications or operations.
- Oracle Big Data Spatial can process both vector and raster types of information and images.

2.1.3 Oracle Big Data Spatial Features and Functions

The spatial data is loaded for query and analysis by the Spatial Server and the images are stored and processed by an Image Processing Framework. You can use the Oracle Big Data Spatial and Graph server on Hadoop for:

- Cataloguing the geospatial information, such as geographical map-based footprints, availability of resources in a geography, and so on.
- Topological processing to calculate distance operations, such as nearest neighbor in a map location.
- Categorization to build hierarchical maps of geographies and enrich the map by creating demographic associations within the map elements.

The following functions are built into Oracle Big Data Spatial and Graph:

- Indexing function for faster retrieval of the spatial data.
- Map function to display map-based footprints.
- Zoom function to zoom-in and zoom-out specific geographical regions.

- Mosaic and Group function to group a set of image files for processing to create a mosaic or subset operations.
- Cartesian and geodetic coordinate functions to represent the spatial data in one of these coordinate systems.
- Hierarchical function that builds and relates geometric hierarchy, such as country, state, city, postal code, and so on. This function can process the input data in the form of documents or latitude/longitude coordinates.

2.1.4 Oracle Big Data Spatial Files, Formats, and Software Requirements

The stored spatial data or images can be in one of these supported formats:

- GeoJSON files
- Shapefiles
- Both Geodetic and Cartesian data
- Other GDAL supported formats

You must have the following software, to store and process the spatial data:

- Java runtime
- GCC Compiler - Only when the GDAL-supported formats are used

2.2 Oracle Big Data Vector and Raster Data Processing

Oracle Big Data Spatial and Graph supports the storage and processing of both vector and raster spatial data.

- Oracle Big Data Spatial Raster Data Processing
- Oracle Big Data Spatial Vector Data Processing

2.2.1 Oracle Big Data Spatial Raster Data Processing

For processing the raster data, the GDAL loader loads the raster spatial data or images onto a HDFS environment. The following basic operations can be performed on a raster spatial data:

- Mosaic: Combine multiple raster images to create a single mosaic image.
- Subset: Perform subset operations on individual images.
- Raster algebra operations: Perform algebra operations on every pixel in the rasters (for example, add, divide, multiply, log, pow, sine, sinh, and acos).
- User-specified processing: Raster processing is based on the classes that user sets to be executed in mapping and reducing phases.

This feature supports a MapReduce framework for raster analysis operations. The users have the ability to custom-build their own raster operations, such as performing an algebraic function on a raster data and so on. For example, calculate the slope at each base of a digital elevation model or a 3D representation of a spatial surface, such as a terrain. For details, see Oracle Big Data Spatial Hadoop Image Processing Framework for Raster Data Processing.

2.2.2 Oracle Big Data Spatial Vector Data Processing

This feature supports the processing of spatial vector data:

- Loaded and stored on to a Hadoop HDFS environment
- Stored either as Cartesian or geodetic data

The stored spatial vector data can be used for performing the following query operations and more:

- Point-in-polygon
- Distance calculation
- Anyinteract
- Buffer creation

Sevetal data service operations are supported for the spatial vector data:

- Data enrichment
- Data categorization
- Spatial join

In addition, there is a limited Map Visualization API support for only the HTML5 format. You can access these APIs to create custom operations. For details, see “Oracle Big Data Spatial Vector Analysis.”

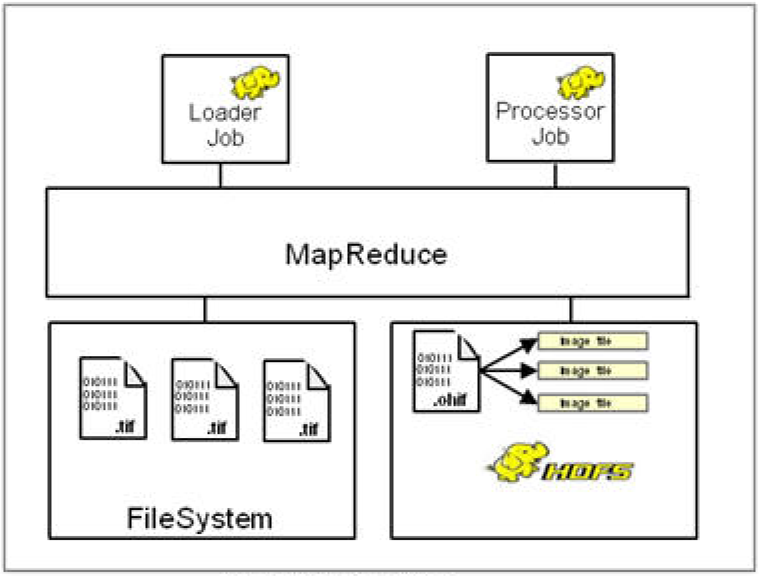
2.3 Oracle Big Data Spatial Hadoop Image Processing Framework for Raster Data Processing

Oracle Spatial Hadoop Image Processing Framework allows the creation of new combined images resulting from a series of processing phases in parallel with the following features:

- HDFS Images storage, where every block size split is stored as a separate tile, ready for future independent processing
- Subset, user-defined, and map algebra operations processed in parallel using the MapReduce framework
- Ability to add custom processing classes to be executed in the mapping or reducing phases in parallel in a transparent way
- Fast processing of georeferenced images
- Support for GDAL formats, multiple bands images, DEMs (digital elevation models), multiple pixel depths, and SRIDs
- Java API providing access to framework operations; useful for web services or standalone Java applications

The Oracle Spatial Hadoop Image Processing Framework consists of two modules, a Loader and Processor, each one represented by a Hadoop job running on different stages in a Hadoop cluster, as represented in the following diagram. Also, you can load and process the images using the Image Server web application, and you can use the Java API to expose the framework's capabilities.

- Image Loader
- Image Processor
- Image Server



Hadoop Cluster

For installation and configuration information, see:

- Installing Oracle Big Data Spatial and Graph on an Oracle Big Data Appliance
- Installing and Configuring the Big Data Spatial Image Processing Framework
- Installing and Configuring the Big Data Spatial Image Server

2.3.1 Image Loader

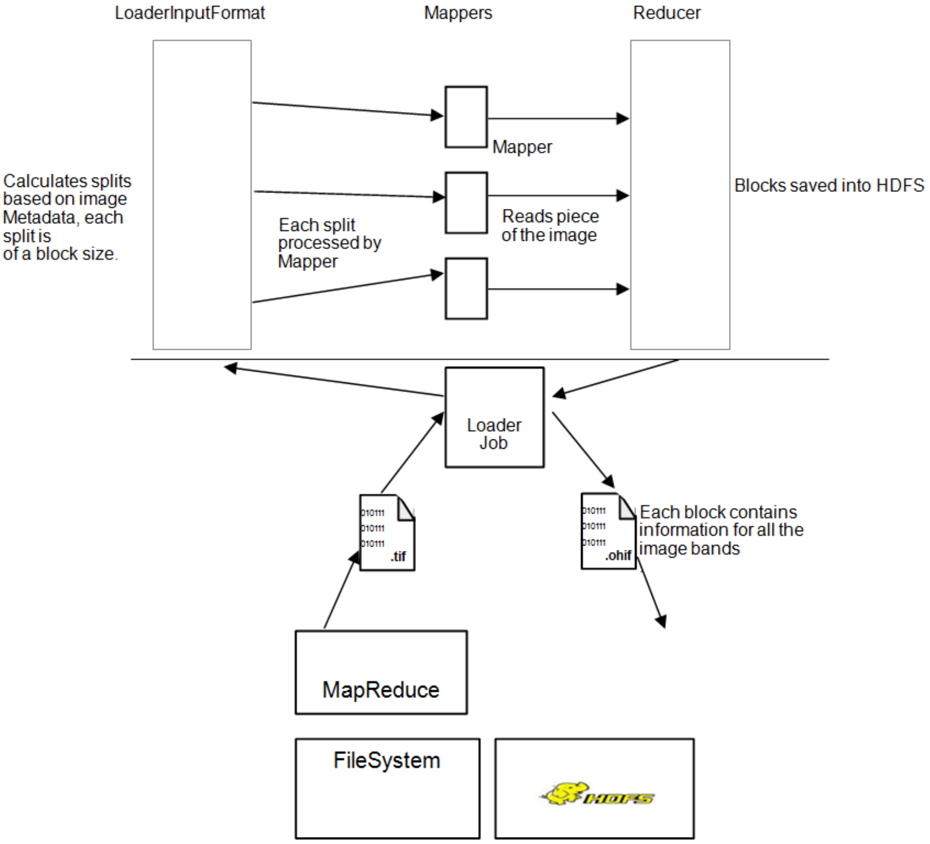
The Image Loader is a Hadoop job that loads a specific image or a group of images into HDFS.

- While importing, the image is tiled and stored as an HDFS block.
- GDAL is used to tile the image.
- Each tile is loaded by a different mapper, so reading is parallel and faster.
- Each tile includes a certain number of overlapping bytes (user input), so that the tiles cover area from the adjacent tiles.
- A MapReduce job uses a mapper to load the information for each tile. There are 'n' number of mappers, depending on the number of tiles, image resolution and block size.
- A single reduce phase per image puts together all the information loaded by the mappers and stores the images into a special `.ohif` format, which contains the

resolution, bands, offsets, and image data. This way the file offset containing each tile and the node location is known.

- Each tile contains information for every band. This is helpful when there is a need to process only a few tiles; then, only the corresponding blocks are loaded.

The following diagram represents an Image Loader process:

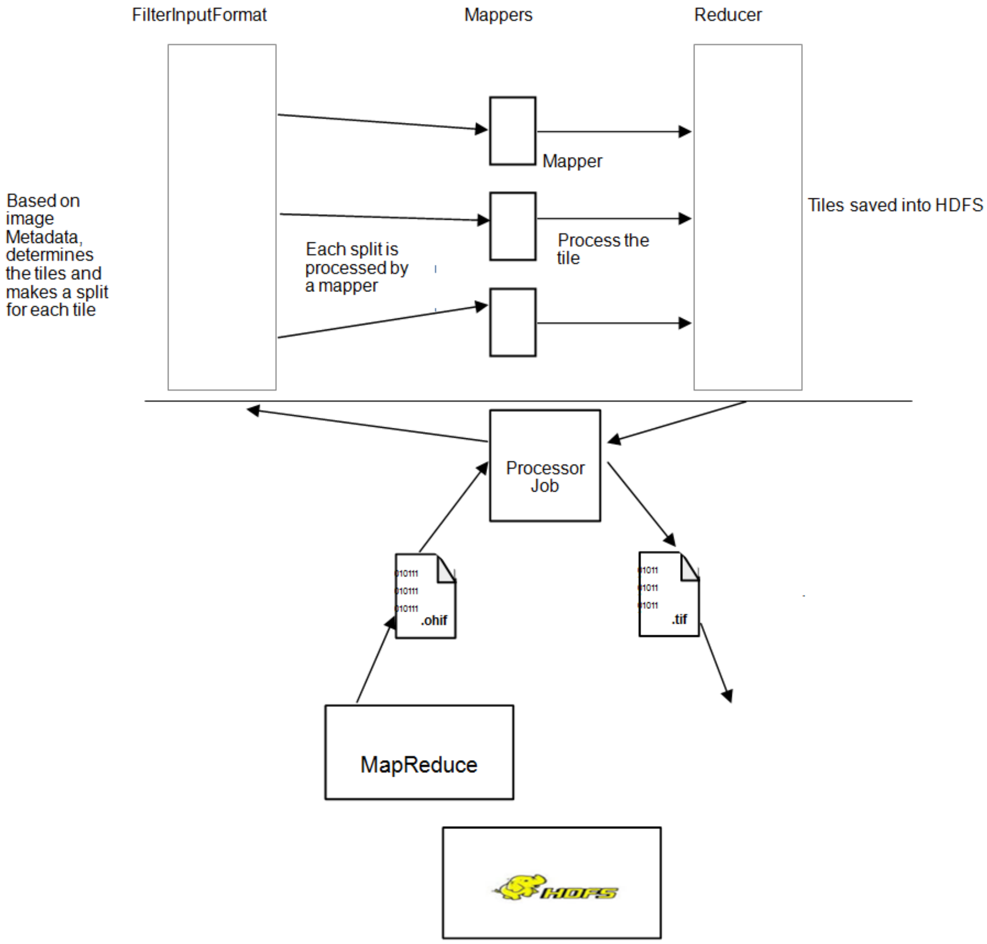


2.3.2 Image Processor

The Image Processor is a Hadoop job that filters tiles to be processed based on the user input and performs processing in parallel to create a new image.

- Processes specific tiles of the image identified by the user. You can identify one, zero, or multiple processing classes. These classes are executed in the mapping or reducing phase, depending on your configuration. For the mapping phase, after the execution of processing classes, a mosaic operation is performed to adapt the pixels to the final output format requested by the user. If no mosaic operation was requested, the input raster is sent to reduce phase as is. For reducer phase, all the tiles are put together into a GDAL data set that is input for user reduce processing class, where final output may be changed or analyzed according to user needs.
- A mapper loads the data corresponding to one tile, conserving data locality.
- Once the data is loaded, the mapper filters the bands requested by the user.
- Filtered information is processed and sent to each mapper in the reduce phase, where bytes are put together and a final processed image is stored into HDFS or regular File System depending on the user request.

The following diagram represents an Image Processor job:



2.3.3 Image Server

The Image Server is a web application the enables you to load and process images from different and variety of sources, especially from the Hadoop File System (HDFS). This Oracle Image Server has several main applications:

- • Visualization of rasters in the entire globe and the ability to create a mosaic from direct selection in the map.
- Raster Image processing to create catalogs from the source images and process into a single unit. You can also view the image thumbnails.
- Hadoop console configuration, used to set up the cluster connection parameters and for the jobs, initial setup.

2.4 Loading an Image to Hadoop Using the Image Loader

The first step to process images using the Oracle Spatial and Graph Hadoop Image Processing Framework is to actually have the images in HDFS, followed by having the images separated into smart tiles. This allows the processing job to work separately on each tile independently. The Image Loader lets you import a single image or a collection of them into HDFS in parallel, which decreases the load time.

The Image Loader imports images from a file system into HDFS, where each block contains data for all the bands of the image, so that if further processing is required on specific positions, the information can be processed on a single node.

- Image Loading Job
- Input Parameters
- Output Parameters

2.4.1 Image Loading Job

The image loading job has its custom input format that splits the image into related image splits. The splits are calculated based on an algorithm that reads square blocks of the image covering a defined area, which is determined by

area = ((blockSize - metadata bytes) / number of bands) / bytes per pixel.

For those pieces that do not use the complete block size, the remaining bytes are refilled with zeros.

Splits are assigned to different mappers where every assigned tile is read using GDAL based on the `ImageSplit` information. As a result an `ImageDataWritable` instance is created and saved in the context.

The metadata set in the `ImageDataWritable` instance is used by the processing classes to set up the tiled image in order to manipulate and process it. Since the source images are read from multiple mappers, the load is performed in parallel and faster.

After the mappers finish reading, the reducer picks up the tiles from the context and puts them together to save the file into HDFS. A special reading process is required to read the image back.

2.4.2 Input Parameters

The following input parameters are supplied to the Hadoop command:

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-
imageloader.jar
  -files <SOURCE_IMGS_PATH>
  -out <HDFS_OUTPUT_FOLDER>
  -gdal <GDAL_LIB_PATH>
  -gdalData <GDAL_DATA_PATH>
  [-overlap <OVERLAPPING_PIXELS>]
  [-thumbnail <THUMBNAIL_PATH>]
  [-expand <false|true>]
```

Where:

`SOURCE_IMGS_PATH` is a path to the source image(s) or folder(s). For multiple inputs use a comma separator. This path must be accessible via NFS to all nodes in the cluster.

`HDFS_OUTPUT_FOLDER` is the HDFS output folder where the loaded images are stored.

`OVERLAPPING_PIXELS` is an optional number of overlapping pixels on the borders of each tile, if this parameter is not specified a default of two overlapping pixels is considered.

`GDAL_LIB_PATH` is the path where GDAL libraries are located.

`GDAL_DATA_PATH` is the path where GDAL data folder is located. This path must be accessible through NFS to all nodes in the cluster.

`THUMBNAIL_PATH` is an optional path to store a thumbnail of the loaded image(s). This path must be accessible through NFS to all nodes in the cluster and must have write access permission for yarn users.
`-expand` controls whether the HDFS path of the loaded raster expands the source path, including all directories. If you set this to `false`, the `.ohif` file is stored directly in the output directory (specified using the `-o` option) without including that directory's path in the raster.

For example, the following command loads all the georeferenced images under the `images` folder and adds an overlapping of 10 pixels on every border possible. The HDFS output folder is `ohiftest` and thumbnail of the loaded image are stored in the `processtest` folder.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-
imageloader.jar   -files /opt/shareddir/spatial/demo/imageserver/images/hawaii.tif -
out ohiftest -overlap 10 -thumbnail /opt/shareddir/spatial/processtest –gdal /opt/
oracle/oracle-spatial-graph/spatial/raster/gdal/lib –gdalData /opt/shareddir/data
```

By default, the Mappers and Reducers are configured to get 2 GB of JVM, but users can override this settings or any other job configuration properties by adding an `imagejob.prop` properties file in the same folder location from where the command is being executed. This properties file may list all the configuration properties that you want to override. For example,

```
mapreduce.map.memory.mb=2560
mapreduce.reduce.memory.mb=2560
mapreduce.reduce.java.opts=-Xmx2684354560
mapreduce.map.java.opts=-Xmx2684354560
```

2.4.3 Output Parameters

The reducer generates two output files per input image. The first one is the `.ohif` file that concentrates all the tiles for the source image, each tile may be processed as a separated instance by a processing mapper. Internally each tile is stored as a HDFS block, blocks are located in several nodes, one node may contain one or more blocks of a specific `.ohif` file. The `.ohif` file is stored in user specified folder with -out flag, under the `/user/<USER_EXECUTING_JOB>/OUT_FOLDER/<PARENT_DIRECTORIES_OF_SOURCE_RASTER>` if the flag `–expand` was not used. Otherwise, the `.ohif` file will be located at `/user/<USER_EXECUTING_JOB>/OUT_FOLDER/`, and the file can be identified as `original_filename.ohif`.

The second output is a related metadata file that lists all the pieces of the image and the coordinates that each one covers. The file is located in HDFS under the metadata location, and its name is hash generated using the name of the `ohif` file. This file is for Oracle internal use only, and lists important metadata of the source raster. Some example lines from a metadata file:

```
srid:26904
datatype:1
resolution:27.90809458890406,-27.90809458890406
file:/user/hdfs/ohiftest/opt/shareddir/spatial/data/rasters/hawaii.tif.ohif
bands:3
mbr:532488.7648166901,4303164.583549625,582723.3350767174,4269619.053853762
0,532488.7648166901,4303164.583549625,582723.3350767174,4269619.053853762
thumbnailpath:/opt/shareddir/spatial/thumb/
```

If the `-thumbnail` flag was specified, a thumbnail of the source image is stored in the related folder. This is a way to visualize a translation of the `.ohif` file. Job execution logs can be accessed using the command `yarn logs -applicationId <applicationId>`.

2.5 Processing an Image Using the Oracle Spatial Hadoop Image Processor

Once the images are loaded into HDFS, they can be processed in parallel using Oracle Spatial Hadoop Image Processing Framework. You specify an output, and the framework filters the tiles to fit into that output, processes them, and puts them all together to store them into a single file. Map algebra operations are also available and, if set, will be the first part of the processing phase. You can specify additional processing classes to be executed before the final output is created by the framework.

The image processor loads specific blocks of data, based on the input (mosaic description or a single raster), and selects only the bands and pixels that fit into the final output. All the specified processing classes are executed and the final output is stored into HDFS or the file system depending on the user request.

- Image Processing Job
- Input Parameters
- Job Execution
- Processing Classes and ImageBandWritable
- Map Algebra Operations
- Output

2.5.1 Image Processing Job

The image processing job has its own custom `FilterInputFormat`, which determines the tiles to be processed, based on the SRID and coordinates. Only images with same data type (pixel depth) as mosaic input data type (pixel depth) are considered. Only the tiles that intersect with coordinates specified by the user for the mosaic output are included. For processing of a single raster, the filter includes all the tiles of the input raster, because the processing will be executed on the complete image. Once the tiles are selected, a custom `ImageProcessSplit` per each one of them is created.

When a mapper receives the `ImageProcessSplit`, it reads the information based on what the `ImageSplit` specifies, performs a filter to select only the bands indicated by the user, and executes the list of map operations and of processing classes defined in the request, if any.

Each mapper process runs in the node, where the data is located. After the map algebra operations and processing classes are executed, a validation occurs to verify if the user is requesting mosaic operation or if analysis includes the complete image; if a mosaic operation is requested, the final process executes the operation. The mosaic operation selects from every tile only the pixels that fit into output and makes the necessary resolution changes to add them in the mosaic output. The resulting bytes are stored in NFS to be recovered by the reducer.

A single reducer picks the tiles and puts them together. If you specified a reducer processing class, the gdal data set with the output raster is sent to this class for analysis and processing. If you selected HDFS output, the ImageLoader is called to store the result into HDFS. Otherwise, by default the image is prepared using GDAL and is stored in the file system (NFS).

2.5.2 Input Parameters

The following input parameters can be supplied to the hadoop command:

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-
imageprocessor.jar
  -config  <MOSAIC_CONFIG_PATH>
  -gdal  <GDAL_LIBRARIES_PATH>
  -gdalData  <GDAL_DATA_PATH>
  [-catalog  <IMAGE_CATALOG_PATH>]
  [-usrlib  <USER_PROCESS_JAR_PATH>]
  [-thumbnail  <THUMBNAIL_PATH>]
  [-nativepath  <USER_NATIVE_LIBRARIES_PATH>]
  [-params  <USER_PARAMETERS>]
  [-file  <SINGLE_RASTER_PATH>]
```

Where:

`MOSAIC_CONFIG_PATH` is the path to the mosaic configuration xml, that defines the features of the output.

`GDAL_LIBRARIES_PATH` is the path where GDAL libraries are located.

`GDAL_DATA_PATH` is the path where the GDAL data folder is located. This path must be accessible via NFS to all nodes in the cluster.

`IMAGE_CATALOG_PATH` is the path to the catalog xml that lists the HDFS image(s) to be processed. This is optional because you can also specify a single raster to process using `–file` flag.

`USER_PROCESS_JAR_PATH` is an optional user-defined jar file or comma-separated list of jar files, each of which contains additional processing classes to be applied to the source images.

`THUMBNAIL_PATH` is an optional flag to activate the thumbnail creation of the loaded image(s). This path must be accessible via NFS to all nodes in the cluster and is valid only for an HDFS output.

`USER_NATIVE_LIBRARIES_PATH` is an optional comma-separated list of additional native libraries to use in the analysis. It can also be a directory containing all the native libraries to load in the application.

`USER_PARAMETERS` is an optional key/value list used to define input data for user processing classes. Use a semicolon to separate parameters. For example: `azimuth=315;altitude=45`

`SINGLE_RASTER_PATH` is an optional path to the `.ohif` file that will be processed by the job. If this is set, you do not need to set a catalog.

For example, the following command will process all the files listed in the catalog file `input.xml` file using the mosaic output definition set in `testFS.xml` file.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-
imageprocessor.jar -catalog /opt/shareddir/spatial/demo/imageserver/images/input.xml
-config /opt/shareddir/spatial/demo/imageserver/images/testFS.xml -thumbnail /opt/
shareddir/spatial/processtest –gdal /opt/oracle/oracle-spatial-graph/spatial/raster/
gdal/lib –gdalData /opt/shareddir/data
```

By default, the Mappers and Reducers are configured to get 2 GB of JVM, but users can override this settings or any other job configuration properties by adding an `imagejob.prop` properties file in the same folder location from where the command is being executed.

2.5.2.1 Catalog XML Structure

The following is an example of input catalog XML used to list every source image considered for mosaic operation generated by the image processing job.

```
-<catalog>
  -<image>
<raster>/user/hdfs/ohiftest/opt/shareddir/spatial/data/rasters/maui.tif.ohif</raster>
<bands datatype='1' config='1,2,3'>3</bands>
   </image>
</catalog>
```

A `<catalog>` element contains the list of <image> elements to process.

Each `<image>` element defines a source image or a source folder within the <raster> element. All the images within the folder are processed.

The `<bands>` element specifies the number of bands of the image, The `datatype` attribute has the raster data type and the `config` attribute specifies which band should appear in the mosaic output band order. For example: 3,1,2 specifies that mosaic output band number 1 will have band number 3 of this raster, mosaic band number 2 will have source band 1, and mosaic band number 3 will have source band 2. This order may change from raster to raster.

2.5.2.2 Mosaic Definition XML Structure

The following is an example of a mosaic configuration XML used to define the features of the output generated by the image processing job.

```
-<mosaic exec="false">
  -<output>
   <SRID>26904</SRID>
   <directory type="FS">/opt/shareddir/spatial/processOutput</directory>
   <!--directory type="HDFS">newData</directory-->
   <tempFSFolder>/opt/shareddir/spatial/tempOutput</tempFSFolder>
   <filename>littlemap</filename>
   <format>GTIFF</format>
  <width>1600</width>
  <height>986</height>
  <algorithm order="0">2</algorithm>
  <bands layers="3" config="3,1,2"/>
  <nodata>#000000</nodata>
  <pixelType>1</pixelType>
  </output>
  -<crop>
   -<transform>
    356958.985610072,280.38843650364862,0,2458324.0825054757,0,-280.38843650364862 </
transform>
  </crop>
<process><classMapper
params="threshold=454,2954">oracle.spatial.hadoop.twc.FarmTransformer</
classMapper><classReducer
params="plot_size=100400">oracle.spatial.hadoop.twc.FarmAlignment</classReducer></
process>
   <operations>
        <localif operator="<" operand="3" newvalue="6"/>
                   <localadd arg="5"/>
                   <localsqrt/>
                   <localround/>
```

```
   </operations>
</mosaic>
```

The `<mosaic>` element defines the specifications of the processing output. The `exec` attribute specifies if the processing will include mosaic operation or not. If set to `“false”`, a mosaic operation is not executed a single raster is processed; if set to `“true”` or not set, a mosaic operation is performed. Some of the following elements are required only for mosaic operations and ignored for single raster processing.

The `<output>` element defines the features such as <SRID> considered for the output. All the images in different SRID are converted to the mosaic SRID in order to decide if any of its tiles fit into the mosaic or not. This element is not required for single raster processing, because the output rster has the same SRID as the input.

The `<directory>` element defines where the output is located. It can be in an HDFS or in regular FileSystem (FS), which is specified in the tag type.

The `<tempFsFolder>` element sets the path to store the mosaic output temporarily.

The `<filename>` and `<format>` elements specify the output filename. `<filename>` is not required for single raster process; and if it is not specified, the name of the input file (determined by the `-file` attribute during the job call) is used for the output file. `<format>` is not required for single raster processing, because the output raster has the same format as the input.

The `<width>` and `<height>` elements set the mosaic output resolution. They are not required for single raster processing, because the output raster has the same resolution as the input.

The `<algorithm>` element sets the order algorithm for the images. A 1 order means, by source last modified date, and a 2 order means, by image size. The order tag represents ascendant or descendant modes. (These properties are for mosaic operations where multiple rasters may overlap.)

The `<bands>` element specifies the number of bands in the output mosaic. Images with fewer bands than this number are discarded. The `config` attribute can be used for single raster processing to set the band configuration for output, because there is no catalog.

The `<nodata>` element specifies the color in the first three bands for all the pixels in the mosaic output that have no value.

The `<pixelType>` element sets the pixel type of the mosaic output. Source images that do not have the same pixel size are discarded for processing. This element is not required for single raster processing: if not specified, the pixel type will be the same as for the input.

The `<crop>` element defines the coordinates included in the mosaic output in the following order: `startcoordinateX`, `pixelXWidth`, `RotationX`, `startcoordinateY`, `RotationY`, and `pixelheightY`. This element is not required for single raster processing: if not specified, the complete image is considered for analysis.

The `<process>` element lists all the classes to execute before the mosaic operation.

The `<classMapper>` element is used for classes that will be executed during mapping phase, and the `<classReducer>` element is used for classes that will be executed during reduce phase. Both elements have the `params` attribute, where you can send input parameters to processing classes according to your needs.

The `<operations>` element lists all the map algebra operations that will be processed for this request.

2.5.3 Job Execution

The first step of the job is to filter the tiles that would fit into the output. As a start, the location files that hold tile metadata are sent to the`InputFormat`.

By extracting the `pixelType`, the filter decides whether the related source image is valid for processing or not. Based on the user definition made in the catalog xml, one of the following happens:

- If the image is valid for processing, then the SRID is evaluated next
- If it is different from the user definition, then the MBR coordinates of every tile are converted into the user SRID and evaluated.

This way, every tile is evaluated for intersection with output definition. Only the intersecting tiles are selected, and a split is created for each one of them.

A mapper processes each split in the node where it is stored. The mapper executes the sequence of map algebra operations and processing classes defined by the user, and then the mosaic process is executed if requested. A single reducer puts together the result of the mappers; and if user-specified reducing processing classes were set, sets the output data set to these classes for analysis or process. Finally, the mapper stores the image into FS or HDFS upon user request. If the user requested is to store the output into HDFS, then the `ImageLoader` job is invoked to store the image as a `.ohif` file.

By default, the mappers and reducers are configured to get 2 GB of JVM, but you can override this settings or any other job configuration properties by adding an `imagejob.prop` properties file in the same folder location from where the command is being executed.

2.5.4 Processing Classes and ImageBandWritable

The processing classes specified in the catalog XML must follow a set of rules to be correctly processed by the job. All the processing classes in the mapping phase must implement the `ImageProcessorInterface` interface. For the reducer phase, they must implement the `ImageProcessorReduceInterface` interface.

When implementing a processing class, you may manipulate the raster using its object representation `ImageBandWritable`. An example of an processing class is provided with the framework to calculate the slope on DEMs. You can create mapping operations, for example, to transforms the pixel values to another value by a function. The `ImageBandWritable` instance defines the content of a tile, such as resolution, size, and pixels. These values must be reflected in the properties that create the definition of the tile. The integrity of the mosaic output depends on the correct manipulation of these properties.

The `ImageBandWritable` instance defines the content of a tile, such as resolution, size, and pixels. These values must be reflected in the properties that create the definition of the tile. The integrity of the output depends on the correct manipulation of these properties.

Table 2-1 ImageBandWritable Properties

Type - Property	Description
IntWritable dstWidthSize	Width size of the tile

Table 2-1 (Cont.) ImageBandWritable Properties

Type - Property	Description
IntWritable dstHeightSize	Height size of the tile
IntWritable bands	Number of bands in the tile
IntWritable dType	Data type of the tile
IntWritable offX	Starting X pixel, in relation to the source image
IntWritable offY	Starting Y pixel, in relation to the source image
IntWritable totalWidth	Width size of the source image
IntWritable totalHeight	Height size of the source image
IntWritable bytesNumber	Number of bytes containing the pixels of the tile and stored into baseArray
BytesWritable[] baseArray	Array containing the bytes representing the tile pixels, each cell represents a band
IntWritable[][] basePaletteArray	Array containing the int values representing the tile palette, each array represents a band. Each integer represents an entry for each color in the color table, there are four entries per color
IntWritable[] baseColorArray	Array containing the int values representing the color interpretation, each cell represents a band
DoubleWritable[] noDataArray	Array containing the NODATA values for the image, each cell contains the value for the related band
ByteWritable isProjection	Specifies if the tile has projection information with Byte.MAX_VALUE
ByteWritable isTransform	Specifies if the tile has the geo transform array information with Byte.MAX_VALUE
ByteWritable isMetadata	Specifies if the tile has metadata information with Byte.MAX_VALUE
IntWritable projectionLength	Specifies the projection information length
BytesWritable projectionRef	Specifies the projection information in bytes
DoubleWritable[] geoTransform	Contains the geo transform array
IntWritable metadataSize	Number of metadata values in the tile
IntWritable[] metadataLength	Array specifying the length of each metadataValue
BytesWritable[] metadata	Array of metadata of the tile

Table 2-1 (Cont.) ImageBandWritable Properties

Type - Property	Description
GeneralInfoWritable mosaicInfo	The user-defined information in the mosaic xml. Do not modify the mosaic output features. Modify the original xml file in a new name and run the process using the new xml
MapWritable extraFields	Map that lists key/value pairs of parameters specific to every tile to be passed to the reducer phase for analysis

Processing Classes and Methods

When modifying the pixels of the tile, first get the band information into an array using the following method:

```
byte [] bandData1 =(byte []) img.getBand(0);
```

The bytes representing the tile pixels of band 1 are now in the bandData1 array. The base index is zero.

The `getBand(int bandId)` method will get the band of the raster in the specified `bandId` position. You can cast the object retrieved to the type of array of the raster; it could be byte, short (unsigned int 16 bits, int 16 bits), int (unsigned int 32 bits, int 32 bits), float (float 32 bits), or double (float 64 bits).

With the array of pixels available, it is possible now to transform them upon a user request.

After processing the pixels, if the same instance of ImageBandWritable must be used, then execute the following method:

```
img.removeBands;
```

This removes the content of previous bands, and you can start adding the new bands. To add a new band use the following method:

```
img.addBand(Object band);
```

Otherwise, you may want to replace a specific band by using trhe following method:

```
img.replaceBand(Object band, int bandId)
```

In the preceding methods, `band` is an array containing the pixel information, and `bandID` is the identifier of the band to be replaced.. Do not forget to update the instance size, data type, bytesNumber and any other property that might be affected as a result of the processing operation. Setters are available for each property.

2.5.4.1 Location of the Classes and Jar Files

All the processing classes must be contained in a single jar file if you are using the Oracle Image Server Console. The processing classes might be placed in different jar files if you are using the command line option.

When new classes are visible in the classpath, they must be added to the mosaic XML in the `<process><classMapper>` or `<process><classReducer>` section. Every `<class>` element added is executed in order of appearance: for mappers, just before the final mosaic operation is performed; and for reducers, just after all the processed tiles are put together in a single data set.

2.5.5 Map Algebra Operations

You can process local map algebra operations on the input rasters, where pixels are altered depending on the operation. The order of operations in the configuration XML determines the order in which the operations are processed. After all the map algebra operations are processed, the processing classes are run, and finally the mosaic operation is performed.

The following map algebra operations can be added in the `<operations>` element in the mosaic configuration XML, with the operation name serving as an element name.

`localnot`: Gets the negation of every pixel, inverts the bit pattern. If the result is a negative value and the data type is unsigned, then the NODATA value is set. If the raster does not have a specified NODATA value, then the original pixel is set.

`locallog`: Returns the natural logarithm (base *e*) of a pixel. If the result is NaN, then original pixel value is set; if the result is Infinite, then the NODATA value is set. If the raster does not have a specified NODATA value, then the original pixel is set.

`locallog10`: Returns the base 10 logarithm of a pixel. If the result is NaN, then the original pixel value is set; if the result is Infinite, then the NODATA value is set. If the raster does not have a specified NODATA value, then the original pixel is set.

`localadd`: Adds the specified value as argument to the pixel .Example: `<localadd arg="5"/>`

`localdivide`: Divides the value of each pixel by the specified value set as argument. Example: `<localdivide arg="5"/>`

`localif`: Modifies the value of each pixel based on the condition and value specified as argument. Valid operators: = , <, >, >=, < !=. Example:: `<localif operator="<" operand="3" newvalue="6"/>`, which modifies all the pixels whose value is less than 3, setting the new value to 6.

`localmultiply`: Multiplies the value of each pixel times the value specified as argument. Example: `<localmultiply arg="5"/>`

`localpow`: Raises the value of each pixel to the power of the value specified as argument. Example: `<localpow arg="5"/>`. If the result is infinite, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localsqrt`: Returns the correctly rounded positive square root of every pixel. If the result is infinite or NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localsubstract`: Subtracts the value specified as argument to every pixel value. Example: `<localsubstract arg="5"/>`

`localacos`: Calculates the arc cosine of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localasin`: Calculates the arc sine of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localatan`: Calculates the arc tangent of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localcos`: Calculates the cosine of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localcosh`: Calculates the hyperbolic cosine of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localsin`: Calculates the sine of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localtan`: Calculates the tangent of a pixel. The pixel is not modified if the cosine of this pixel is 0. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localsinh`: Calculates the arc hyperbolic sine of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localtanh`: Calculates the hyperbolic tangent of a pixel. If the result is NaN, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localdefined`: Maps an integer typed pixel to 1 if the cell value is not NODATA; otherwise, 0.

`localundefined`: Maps an integer typed Raster to 0 if the cell value is not NODATA; otherwise, 1.

`localabs`: Returns the absolute value of signed pixel. If the result is Infinite, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localnegate`: Multiplies by -1 the value of each pixel.

`localceil`: Returns the smallest value that is greater than or equal to the pixel value and is equal to a mathematical integer. If the result is Infinite, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localfloor`: Returns the smallest value that is less than or equal to the pixel value and is equal to a mathematical integer. If the result is Infinite, the NODATA value is set to this pixel. If the raster does not have a specified NODATA value, then the original pixel is set.

`localround`: Returns the closest integer value to every pixel.

2.5.6 Output

When you specify an HDFS directory in the configuration XML, the output generated is an `.ohif` file as in the case of an `ImageLoader` job,

When the user specifies a FS directory in the configuration XML, the output generated is an image with the filename and type specified and is stored into regular FileSystem.

In both the scenarios, the output must comply with the specifications set in the configuration XML. The job execution logs can be accessed using the command `yarn logs -applicationId <applicationId>`.

2.6 Loading and Processing an Image Using the Oracle Spatial Hadoop Raster Processing API

The framework provides a raster processing API that lets you load and process rasters without creating XML but instead using a Java application. The application can be executed inside the cluster or on a remote node.

The API provides access to the framework operations, and is useful for web service or standalone Java applications.

To execute any of the jobs, a `HadoopConfiguration` object must be created. This object is used to set the necessary configuration information (such as the jar file name and the GDAL paths) to create the job, manipulate rasters, and execute the job. The basic logic is as follows:

```
     //Creates Hadoop Configuration
     HadoopConfiguration hadoopConf = new HadoopConfiguration();
     //Assigns GDAL_DATA location based on specified SHAREDDIR, this data folder is
required by gdal to look for data tables that allow SRID conversions
     String gdalData = sharedDir + ProcessConstants.DIRECTORY_SEPARATOR + "data";
     hadoopConf.setGdalDataPath(gdalData);
     //Sets jar name for processor
     hadoopConf.setMapreduceJobJar("hadoop-imageprocessor.jar");
     //Creates the job
     RasterProcessorJob processor = (RasterProcessorJob)
hadoopConf.createRasterProcessorJob();
```

If the API is used on a remote node, you can set properties in the Hadoop Configuration object to connect to the cluster. For example:

```
        //Following config settings are required for standalone execution. (REMOTE
ACCESS)
        hadoopConf.setUser("hdfs");
        hadoopConf.setHdfsPathPrefix("hdfs://den00btb.us.oracle.com:8020");
        hadoopConf.setResourceManagerScheduler("den00btb.us.oracle.com:8030");
        hadoopConf.setResourceManagerAddress("den00btb.us.oracle.com:8032");
        hadoopConf.setYarnApplicationClasspath("/etc/hadoop/conf/,/usr/lib/
hadoop/*,/usr/lib/hadoop/lib/*," +
                          "/usr/lib/hadoop-hdfs/*,/usr/lib/hadoop-
hdfs/lib/*,/usr/lib/hadoop-yarn/*," +
                          "/usr/lib/hadoop-yarn/lib/*,/usr/lib/hadoop-
mapreduce/*,/usr/lib/hadoop-mapreduce/lib/* ");
```

After the job is created, the properties for its execution must be set depending on the job type. There are two job classes: `RasterLoaderJob` to load the rasters into HDFS, and `RasterProcessorJob` to process them.

The following example loads a Hawaii raster into the APICALL_HDFS directory. It creates a thumbnail in a shared folder, and specifies 10 pixels overlapping on each edge of the tiles.

```
    private static void executeLoader(HadoopConfiguration hadoopConf){
        hadoopConf.setMapreduceJobJar("hadoop-imageloader.jar");
        RasterLoaderJob loader = (RasterLoaderJob)
hadoopConf.createRasterLoaderJob();
        loader.setFilesToLoad("/net/den00btb/scratch/zherena/hawaii/hawaii.tif");
        loader.setTileOverlap("10");
        loader.setOutputFolder("APICALL");
        loader.setRasterThumbnailFolder("/net/den00btb/scratch/zherena/
```

```
processOutput");
        try{
        loader.setGdalPath("/net/den00btb/scratch/zherena/gdal/lib");

        boolean loaderSuccess = loader.execute();
            if(loaderSuccess){
                System.out.println("Successfully executed loader job");
            }
            else{
                System.out.println("Failed to execute loader job");
            }
        }catch(Exception e ){
        System.out.println("Problem when trying to execute raster loader " +
e.getMessage());
        }
    }
}
```

The following example processes the loaded raster.

```
private static void executeProcessor(HadoopConfiguration hadoopConf){
    hadoopConf.setMapreduceJobJar("hadoop-imageprocessor.jar");
    RasterProcessorJob processor = (RasterProcessorJob)
hadoopConf.createRasterProcessorJob();

    try{
    processor.setGdalPath("/net/den00btb/scratch/zherena/gdal/lib");
    MosaicConfiguration mosaic = new MosaicConfiguration();
        mosaic.setBands(3);
        mosaic.setDirectory("/net/den00btb/scratch/zherena/processOutput");
        mosaic.setFileName("APIMosaic");
        mosaic.setFileSystem(RasterProcessorJob.FS);
        mosaic.setFormat("GTIFF");
        mosaic.setHeight(3192);
        mosaic.setNoData("#FFFFFF");
        mosaic.setOrderAlgorithm(ProcessConstants.ALGORITMH_FILE_LENGTH);
        mosaic.setOrder("1");
        mosaic.setPixelType("1");
        mosaic.setPixelXWidth(67.457513);
        mosaic.setPixelYWidth(-67.457513);
        mosaic.setSrid("26904");
        mosaic.setUpperLeftX(830763.281336);
        mosaic.setUpperLeftY(2259894.481403);
        mosaic.setWidth(1300);
    processor.setMosaicConfigurationObject(mosaic.getCompactMosaic());
        RasterCatalog catalog = new RasterCatalog();
        Raster raster = new Raster();
        raster.setBands(3);
        raster.setBandsOrder("1,2,3");
        raster.setDataType(1);
        raster.setRasterLocation("/user/hdfs/APICALL/net/den00btb/scratch/zherena/
hawaii/hawaii.tif.ohif");
        catalog.addRasterToCatalog(raster);

        processor.setCatalogObject(catalog.getCompactCatalog());
    boolean processorSuccess = processor.execute();
        if(processorSuccess){
            System.out.println("Successfully executed processor job");
        }
        else{
            System.out.println("Failed to execute processor job");
```

```
        }
    }catch(Exception e ){
    System.out.println("Problem when trying to execute raster processor " +
e.getMessage());
    }
}
```

In the preceding example, the thumbnail is optional if the mosaic results will be stored in HDFS. If a processing jar file is specified (used when the additional user processing classes are specified), the location of the jar file containing these lasses must be specified. The other parameters are required for the mosaic to be generated successfully.

Several examples of using the processing API are provided `/opt/oracle/oracle-spatial-graph/spatial/raster/examples/java/src`. Review the Java classes to understand their purpose. You may execute them using the scripts provided for each example located under `/opt/oracle/oracle-spatial-graph/spatial/raster/examples/java/cmd`.

After you have executed the scripts and validated the results, you can modify the Java source files to experiment on them and compile them using the provided script `/opt/oracle/oracle-spatial-graph/spatial/raster/examples/java/build.xml`. Ensure that you have write access on the `/opt/oracle/oracle-spatial-graph/spatial/raster/jlib directory`.

2.7 Oracle Big Data Spatial Vector Analysis

Oracle Big Data Spatial Vector Analysis is a Spatial Vector Analysis API, which runs as a Hadoop job and provides MapReduce components for spatial processing of data stored in HDFS. These components make use of the Spatial Java API to perform spatial analysis tasks. There is a web console provided along with the API. The supported features include:

- Multiple Hadoop API Support
- Spatial Indexing
- Using MVSuggest
- Spatial Filtering
- Classifying Data Hierarchically
- Generating Buffers
- Spatial Binning
- Spatial Clustering
- Spatial Join
- Spatial Partitioning

In addition, read the following information for understanding the implementation details:

- RecordInfoProvider
- HierarchyInfo

- Using JGeometry in MapReduce Jobs
- Tuning Performance Data of Job Running Times using Vector Analysis API

2.7.1 Multiple Hadoop API Support

Oracle Big Data Spatial Vector Analysis provides classes for both the old and new (context objects) Hadoop APIs. In general, classes in the `mapred` package are used with the old API, while classes in the `mapreduce` package are used with the new API

The examples in this guide use the old Hadoop API; however, all the old Hadoop Vector API classes have equivalent classes in the new API. For example, the old class `oracle.spatial.hadoop.vector.`**`mapred`**`.job.SpatialIndexing` has the equivalent new class named `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.SpatialIndexing`. In general, and unless stated otherwise, only the change from `mapred` to `mapreduce` is needed to use the new Hadoop API Vector classes.

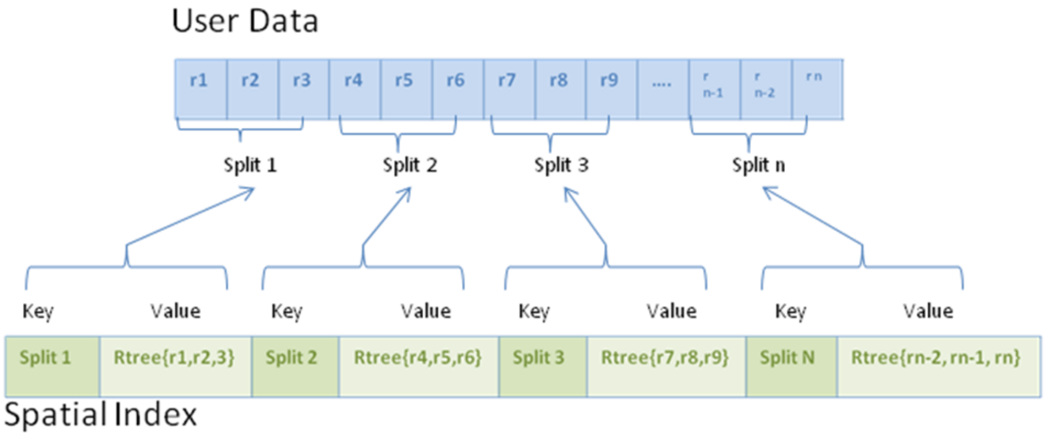
Classes such as `oracle.spatial.hadoop.vector.RecordInfo`, which are not in the mapred or mapreduce package, are compatible with both Hadoop APIs.

2.7.2 Spatial Indexing

A spatial index is in the form of a key/value pair and generated as a Hadoop MapFile. Each MapFile entry contains a spatial index for one split of the original data. The key and value pair contains the following information:

- Key: a split identifier in the form: path + start offset + length.
- Value: a spatial index structure containing the actual indexed records.

The following figure depicts a spatial index in relation to the user data. The records are represented as r1, r2, and so on. The records are grouped into splits (Split 1, Split 2, Split 3, Split n). Each split has a Key-Value pair where the key identifies the split and the value identifies an Rtree index on the records in that split.



Related subtopics:

- Spatial Indexing Class Structure

2.7.2.1 Spatial Indexing Class Structure

Records in a spatial index are represented using the class `oracle.spatial.hadoop.vector.RecordInfo`. A `RecordInfo` typically contains a subset of the original record data and a way to locate the record in the file where it is stored. The specific `RecordInfo` data depends on two things:

- `InputFormat` used to read the data
- `RecordInfoProvider` implementation, which provides the record's data

The fields contained within a `RecordInfo`:

- Id: Text field with the record Id.
- Geometry: `JGeometry` field with the record geometry.
- Extra fields: Additional optional fields of the record can be added as name-value pairs. The values are always represented as text.
- Start offset: The position of the record in a file as a byte offset. This value depends on the `InputFormat` used to read the original data.
- Length: The original record length in bytes.
- Path: The file path can be added optionally. This is optional because the file path can be known using the spatial index entry key. However, to add the path to the `RecordInfo` instances when a spatial index is created, the value of the configuration property `oracle.spatial.recordInfo.includePathField` key is set to `true`.

2.7.2.2 Configuration for Creating a Spatial Index

A spatial index is created using a combination of `FileSplitInputFormat`, `SpatialIndexingMapper`, `InputFormat`, and `RecordInfoProvider`, where the last two are provided by the user. The following code example shows part of the configuration needed to run a job that creates a spatial index for the data located in the HDFS folder `/user/data`.

```
//input

conf.setInputFormat(FileSplitInputFormat.class);
FileSplitInputFormat.setInputPaths(conf, new Path("/user/data"));
FileSplitInputFormat.setInternalInputFormatClass(conf, GeoJsonInputFormat.class);
FileSplitInputFormat.setRecordInfoProviderClass(conf,
GeoJsonRecordInfoProvider.class);

//output

conf.setOutputFormat(MapFileOutputFormat.class);
FileOutputFormat.setOutputPath(conf, new Path("/user/data_spatial_index"));

//mapper

conf.setMapperClass(SpatialIndexingMapper.class);
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(RTreeWritable.class);
```

In this example,

- The `FileSplitInputFormat` is set as the job `InputFormat`. `FileSplitInputFormat` is a subclass of `CompositeInputFormat` (`WrapperInputFormat` in the new Hadoop API version), an abstract class that uses another `InputFormat` implementation (`internalInputFormat`) to read the data. The internal `InputFormat` and the `RecordInfoProvider` implementations are specified by the user and they are set to `GeoJsonInputFormat` and `GeoJsonRecordInfoProvider`, respectively.

- The `MapFileOutputFormat` is set as the `OutputFormat` in order to generate a `MapFile`
- The mapper is set to `SpatialIndexingMappper`. The mapper output key and value types are `Text` (splits identifiers) and `RTreeWritable` (the actual spatial indexes).
- No reducer class is specified so it runs with the default reducer. The reduce phase is needed to sort the output MapFile keys.

Alternatively, this configuration can be set easier by using the `oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing` class. `SpatialIndexing` is a job driver that creates a spatial index. In the following example, a `SpatialIndexing` instance is created, set up, and used to add the settings to the job configuration by calling the `configure()` method. Once the configuration has been set, the job is launched.

```
SpatialIndexing<LongWritable, Text> spatialIndexing = new
SpatialIndexing<LongWritable, Text>();

//path to input data

spatialIndexing.setInput("/user/data");

//path of the spatial index to be generated

spatialIndexing.setOutput("/user/data_spatial_index");

//input format used to read the data

spatialIndexing.setInputFormatClass(TextInputFormat.class);

//record info provider used to extract records information

spatialIndexing.setRecordInfoProviderClass(TwitterLogRecordInfoProvider.class);

//add the spatial indexing configuration to the job configuration

spatialIndexing.configure(jobConf);

//run the job

JobClient.runJob(jobConf);
```

2.7.2.3 Spatial Index Metadata

A metadata file is generated for every spatial index that is created. The spatial index metadata can be used to quickly find information related to a spatial index, such as the number of indexed records, the minimum bounding rectangle (MBR) of the indexed data, and the paths of both the spatial index and the indexed source data. The spatial index metadata can be retrieved using the spatial index name.

A spatial index metadata file contains the following information:

- Spatial index name
- Path to the spatial index

- Number of indexed records
- Number of local indexes
- Extra fields contained in the indexed records
- Geometry layer information such as th SRID, dimensions, tolerance, dimension boundaries, and whether the geometries are geodetic or not
- The following information for each of the local spatial index files: path to the indexed data, path to the local index, and MBR of the indexed data

The following metadata proeprties can be set when creating a spatial index using the `SpatialIndexing` class:

- `indexName`: Name of the spatial index. If not set, the output folder name is used.
- `metadataDir`: Path to the directory where the metadata file will be stored.
  - By default, it will be stored in the following path relative to the user directory: `oracle_spatial/index_metadata`. If the user is `hdfs`, it will be `/user/hdfs/oracle_spatial/index_metadata`.
- `overwriteMetadata`: If set to `true`, then when a spatial index metadata file already exists for a spatial index with the same `indexName` in the current `metadataDir`, the spatial index metadata will be overwritten. If set to `false` and if a spatial index metadata file already exists for a spatial index with the same `indexName` in the current `metadataDir`, then an error is raised.

The following example sets the metadata directory and spatial index name, and specifies to overwrite any existing metadata if the index already exists:

```
spatialIndexing.setMetadataDir("/user/hdfs/myIndexMetadataDir");
spatialIndexing.setIndexName("testIndex");
spatialIndexing.setOverwriteMetadata(true);
```

An existing spatial index can be passed to other jobs by specifying only the `indexName` and optionally the `indexMetadataDir` where the index metadata can be found. When the index name is provided, there is no need to specify the spatial index path and the input format.

The following job drivers accept the `indexName` as a parameter:

- `oracle.spatial.hadoop.vector.mapred.job.Categorization`
- `oracle.spatial.hadoop.vector.mapred.job.SpatialFilter`
- `oracle.spatial.hadoop.vector.mapred.job.Binning`
- Any driver that accepts `oracle.spatial.hadoop.vector.InputDataSet`, such as `SpatialJoin` and `Partitioning`

If the index name is not found in the `indexMetadataDir` path, an error is thrown indicating that the spatial index could not be found.

The following example shows a spatial index being set as the input data set for a binning job:

```
Binning binning = new Binning();
binning.setIndexName("indexExample");
binning.setIndexMetadataDir("indexMetadataDir");
```

2.7.2.4 Input Formats for a Spatial Index

An `InputFormat` must meet the following requisites to be supported:

- It must be a subclass of `FileInputFormat`.
- The `getSplits()`method must return either `FileSplit` or `CombineFileSplit` split types.
- For the old Hadoop API, the `RecordReader`'s `getPos()` method must return the current position to track back a record in the spatial index to its original record in the user file. If the current position is not returned, then the original record cannot be found using the spatial index.

  However, the spatial index still can be created and used in operations that do not require the original record to be read. For example, additional fields can be added as extra fields to avoid having to read the whole original record.

Note:

The spatial indexes are created for each split as returned by the `getSplits()` method. When the spatial index is used for filtering (see Spatial Filtering), it is recommended to use the same `InputFormat` implementation than the one used to create the spatial index to ensure the splits indexes can be found.

The `getPos()` method has been removed from the Hadoop new API; however, `org.apache.hadoop.mapreduce.lib.input.TextInputFormat` and `CombineTextInputFormat` are supported, and it is still possible to get the record start offsets.

Other input formats from the new API are supported, but the record start offsets will not be contained in the spatial index. Therefore, it is not possible to find the original records. The requirements for a new API input format are the same as for the old API. However, they must be translated to the new APIs `FileInputFormat`, `FileSplit`, and `CombineFileSplit`.

2.7.2.5 Support for GeoJSON and Shapefile Formats

The Vector API comes with `InputFormat` and `RecordInfoProvider` implementations for GeoJSON and Shapefile file formats.

The following `InputFormat/RecordInfoProvider` pairs can be used to read and interpret GeoJSON and ShapeFiles, respectively:

```
oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat /
oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider

oracle.spatial.hadoop.vector.shapefile.mapred.ShapeFileInputFormat /
oracle.spatial.hadoop.vector.shapefile.ShapeFileRecordInfoProvider
```

More information about the usage and properties is available in the Javadoc.

2.7.3 Using MVSuggest

`MVSuggest` can be used at the time of spatial indexing to get an approximate location for records that do not have geometry but have some text field. This text field can be used to determine the record location. The geometry returned by `MVSuggest` is used to include the record in the spatial index.

Because it is important to know the field containing the search text for every record, the `RecordInfoProvider` implementation must also implement `LocalizableRecordInfoProvider`. Alternatively, the configuration parameter `oracle.spatial.recordInfo.locationField` can be set with the name of the field containing the search text. For more information, see the Javadoc for `LocalizableRecordInfoProvider`.

A standalone version of `MVSuggest` is shipped with the Vector API and it can be used in some jobs that accept the `MVSConfig` as an input parameter.

The following job drivers can work with `MVSuggest` and all of them have the `setMVSConfig()` method which accepts an instance of `MVSConfig`:

- oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing: has the option of using `MVSuggest` to get approximate spatial location for records which do not contain geometry.
- oracle.spatial.hadoop.vector.mapred.job.Categorization: `MVSuggest` can be used to assign a record to a specific feature in a layer, for example, the feature California in the USA states layer.
- oracle.spatial.hadoop.vector.mapred.job.SuggestService: A simple job that generates a file containing a search text and its match per input record.

The MVSuggest configuration is passed to a job using the MVSConfig or the LocalMVSConfig classes. The basic MVSuggest properties are:

- `serviceLocation`: It is the minimum property required in order to use `MVSuggest`. It contains the path or URL where the `MVSuggest` directory is located or in the case of a URL, where the `MVSuggest` service is deployed.
- `serviceInterfaceType`: the type of `MVSuggest` implementation used. It can be LOCAL(default) for a standalone version and WEB for the web service version.
- `matchLayers`: an array of layer names used to perform the searches.

When using the standalone version of `MVSuggest`, you must specify an `MVSuggest` directory or repository as the `serviceLocation`. An `MVSuggest` directory must have the following structure:

```
mvsuggest_config.json
repository folder
   one or more layer template files in .json format
   optionally, a _config_ directory
   optionally, a _geonames_ directory
```

The `examples` folder comes with many layer template files and a `_config_` directory with the configuration for each template.

It is possible to set the repository folder (the one that contains the templates) as the mvsLocation instead of the whole `MVSuggest` directory. In order to do that, the class `LocalMVSConfig` can be used instead of `MVSConfig` and the `repositoryLocation` property must be set to `true` as shown in the following example:

```
LocalMVSConfig lmvsConf = new LocalMVSConfig();
lmvsConf.setServiceLocation(“file:///home/user/mvs_dir/repository/”);
lmvsConf.setRepositoryLocation(true);
lmvsConf.setPersistentServiceLocation(“/user/hdfs/hdfs_mvs_dir”);
spatialIndexingJob.setMvsConfig(lmvsConf);
```

The preceding example sets a repository folder as the MVS service location. `setRepositoryLocation` is set to `true` to indicate that the service location is a repository instead of the whole `MVSuggest` directory. When the job runs, a whole `MVSuggest` directory will be created using the given repository location; the repository will be indexed and will be placed in a temporary folder while the job finishes. The previously indexed `MVSuggest` directory can be persisted so it can be used later. The preceding example saves the generated `MVSuggest` directory in the HDFS path `/user/hdfs/hdfs_mvs_dir`. Use the MVSDirectory if the `MVSuggest` directory already exists.

2.7.4 Spatial Filtering

Once the spatial index has been generated, it can be used to spatially filter the data. The filtering is performed before the data reaches the mapper and while it is being read. The following sample code example demonstrates how the `SpatialFilterInputFormat` is used to spatially filter the data.

```
//set input path and format

FileInputFormat.setInputPaths(conf, new Path("/user/data/"));
conf.setInputFormat(SpatialFilterInputFormat.class);

//set internal input format

SpatialFilterInputFormat.setInternalInputFormatClass(conf, TextInputFormat.class);
if( spatialIndexPath != null )
{

     //set the path to the spatial index and put it in the distributed cache

     boolean useDistributedCache = true;
     SpatialFilterInputFormat.setSpatialIndexPath(conf, spatialIndexPath,
useDistributedCache);
}
else
{
     //as no spatial index is used a RecordInfoProvider is needed

     SpatialFilterInputFormat.setRecordInfoProviderClass(conf,
TwitterLogRecordInfoProvider.class);
}

//set spatial operation used to filter the records

SpatialOperationConfig spatialOpConf = new SpatialOperationConfig();
spatialOpConf.setOperation(SpatialOperation.IsInside);
spatialOpConf.setJsonQueryWindow("{\"type\":\"Polygon\", \"coordinates\":
[[-106.64595, 25.83997, -106.64595, 36.50061, -93.51001, 36.50061, -93.51001,
25.83997 , -106.64595, 25.83997]]}");
spatialOpConf.setSrid(8307);
spatialOpConf.setTolerance(0.5);
spatialOpConf.setGeodetic(true);
```

`SpatialFilterInputFormat` has to be set as the job's `InputFormat`. The `InputFormat` that actually reads the data must be set as the internal `InputFormat`. In this example, the internal `InputFormat` is `TextInputFormat`.

If a spatial index is specified, it is used for filtering. Otherwise, a `RecordInfoProvider` must be specified in order to get the records geometries, in which case the filtering is performed record by record.

As a final step, the spatial operation and query window to perform the spatial filter are set. It is recommended to use the same internal `InputFormat` implementation used when the spatial index was created or, at least, an implementation that uses the same criteria to generate the splits. For details see "Input Formats for a Spatial Index."

If a simple spatial filtering needs to be performed (that is, only retrieving records that interact with a query window), the built-in job driver `oracle.spatial.hadoop.vector.mapred.job.SpatialFilter` can be used instead. This job driver accepts indexed or non-indexed input and a `SpatialOperationConfig` to perform the filtering.

Related:

- Filtering Records
- Filtering Using the Input Format

2.7.4.1 Filtering Records

The following steps are executed when records are filtered using the `SpatialFilterInputFormat` and a spatial index.

1. `SpatialFilterInputFormat getRecordReader()` method is called when the mapper requests a `RecordReader` for the current split.

2. The spatial index for the current split is retrieved.

3. A spatial query is performed over the records contained in it using the spatial index.

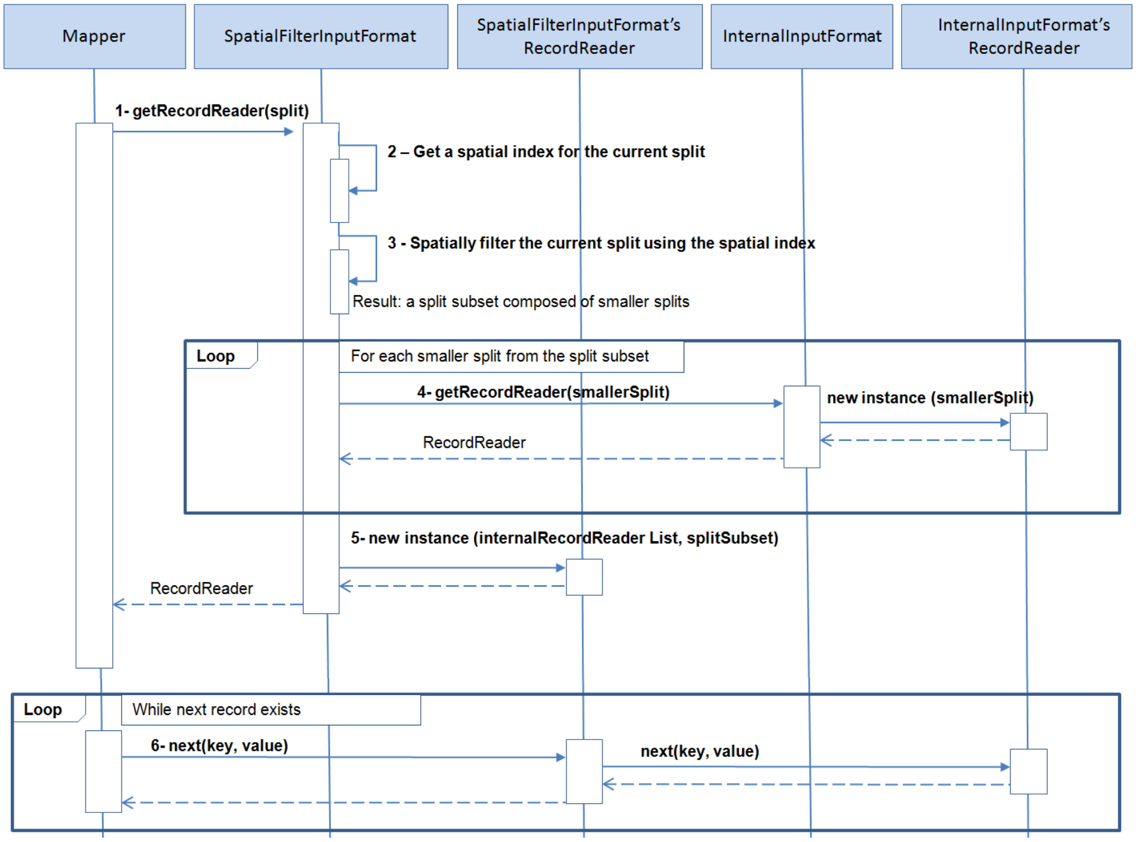
   As a result, the ranges in the split that contains records meeting the spatial filter are known. For example, if a split goes from the file position 1000 to 2000, upon executing the spatial filter it can be determined that records that fulfill the spatial condition are in the ranges 1100-1200, 1500-1600 and 1800-1950. So the result of performing the spatial filtering at this stage is a subset of the original filter containing smaller splits.

4. An Internal`InputFormat` `RecordReader` is requested for every small split from the resulting split subset.

5. A RecordReader is returned to the caller mapper. The returned RecordReader is actually a wrapper RecordReader with one or more RecordReaders returned by the internal `InputFormat`.

6. Every time the mapper calls the RecordReader, the call to next method to read a record is delegated to the internal RecordReader.

These steps are shown in the following spatial filter interaction diagram.



2.7.4.2 Filtering Using the Input Format

A previously generated Spatial Index can be read using the input format implementation `oracle.spatial.hadoop.vector.mapred.input.SpatialIndexInputFormat` (or its new Hadoop API equivalent with the `mapreduce` package instead of `mapred`). `SpatialIndexInputFormat` is used just like any other `FileInputFormat` subclass in that it takes an input path and it is set as the job's input format. The key and values returned are the id `(Text)` and record information `(RecordInfo)` of the records stored in the spatial index.

Aditionally, a spatial filter opertion can be performed by specifying a spatial operation configuration to the input format, so that only the records matching some spatial interaction will be returned to a mapper. The following example shows how to configure a job to read a spatial index to retrieve all the records that are inside a specific area.

```
JobConf conf = new JobConf();
conf.setMapperClass(MyMapper.class);
conf.setInputFormat(SpatialIndexInputFormat.class);
SpatialOperationConfig spatialOpConf = new SpatialOperationConfig();
spatialOpConf.setOperation(SpatialOperation.IsInside);
spatialOpConf.setQueryWindow(JGeometry.createLinearPolygon(new double[]{47.70,
-124.28, 47.70,  -95.12, 35.45, -95.12, 35.45, -124.28, 47.70, -124.28}, 2, 8307));
SpatialIndexInputFormat.setFilterSpatialOperationConfig(spatialOpConf, conf);
```

The mapper in the preceding example can add a nonspatial filter by using the `RecordInfo` extra fields, as shown in the following example.

```
public class MyMapper extends MapReduceBase implements Mapper<Text, RecordInfo,
Text, RecordInfo>{
```

```
    @Override
    public void map(Text key, RecordInfo value, OutputCollector<Text, RecordInfo>
output, Reporter reporter)
            throws IOException {
        if( Integer.valueOf(value.getField("followers_count")) > 0){
            output.collect(key, value);
        }
    }
}
```

2.7.5 Classifying Data Hierarchically

The Vector Analysis API provides a way to classify the data into hierarchical entities. For example, in a given set of catalogs with a defined level of administrative boundaries such as continents, countries and states, it is possible to join a record of the user data to a record of each level of the hierarchy data set. The following example generates a summary count for each hierarchy level, containing the number of user records per continent, country, and state or province:

```
Categorization catJob = new Categorization();
//set a spatial index as the input

catJob.setIndexName("indexExample");

//set the job's output

catJob.setOutput("hierarchy_count");

//set HierarchyInfo implementation which describes the world administrative
boundaries hierarchy

catJob.setHierarchyInfoClass( WorldDynaAdminHierarchyInfo.class );

//specify the paths of the hierarchy data

Path[] hierarchyDataPaths = {
                new Path("file:///home/user/catalogs/world_continents.json"),
                new Path("file:///home/user/catalogs/world_countries.json"),
                new Path("file:///home/user/catalogs/world_states_provinces.json")};
catJob.setHierarchyDataPaths(hierarchyDataPaths);

//set the path where the index for the previous hierarchy data will be generated

catJob.setHierarchyIndexPath(new Path("/user/hierarchy_data_index/"));

//setup the spatial operation which will be used to join records from the two
datasets (spatial index and hierarchy data).
SpatialOperationConfig spatialOpConf = new SpatialOperationConfig();
spatialOpConf.setOperation(SpatialOperation.IsInside);
spatialOpConf.setSrid(8307);
spatialOpConf.setTolerance(0.5);
spatialOpConf.setGeodetic(true);
catJob.setSpatialOperationConfig(spatialOpConf);

//add the previous setup to the job configuration

catJob.configure(conf);

//run the job
RunningJob rj = JobClient.runJob(conf);
```

The preceding example uses the `Categorization` job driver. The configuration can be divided into the following categories:

- Input data: A previously generated spatial index (received as the job input).
- Output data: A folder that contains the summary counts for each hierarchy level.
- Hierarchy data configuration: This contains the following:
  - `HierarchyInfo` class: This is an implementation of `HierarchyInfo` class in charge of describing the current hierarchy data. It provides the number of hierarchy levels, level names, and the data contained at each level.
  - Hierarchy data paths: This is the path to each one of the hierarchy catalogs. These catalogs are read by the `HierarchyInfo` class.
  - Hierarchy index path: This is the path where the hierarchy data index is stored. Hierarchy data needs to be preprocessed to know the parent-child relationships between hierarchy levels. This information is processed once and saved at the hierarchy index, so it can be used later by the current job or even by any other jobs.
- Spatial operation configuration: This is the spatial operation to be performed between records of the user data and the hierarchy data in order to join both datasets. The parameters to set here are the Spatial Operation type (IsInside), SRID (8307), Tolerance (0.5 meters), and whether the geometries are Geodetic (true).

Internally, the `Categorization.configure()` method sets the mapper and reducer to be `SpatialHierarchicalCountMapper` and `SpatialHierarchicalCountReducer`, respectively. `SpatialHierarchicalCountMapper`'s output key is a hierarchy entry identifier in the form `hierarchy_level` + `hierarchy_entry_id`. The mapper output value is a single count for each output key. The reducer sums up all the counts for each key.

Note:

The entire hierarchy data may be read into memory and hence the total size of all the catalogs is expected to be significantly less than the user data. The hierarchy data size should not be larger than a couple of gigabytes.

If you want another type of output instead of counts, for example, a list of user records according to the hierarchy entry. In this case, the `SpatialHierarchicalJoinMapper` can be used. The `SpatialHierarchicalJoinMapper` output value is a `RecordInfo` instance, which can be gathered in a user-defined reducer to produce a different output. The following user-defined reducer generates a MapFile for each hierarchy level using the `MultipleOutputs` class. Each MapFile has the hierarchy entry ids as keys and `ArrayWritable` instances containing the matching records for each hierarchy entry as values. The following is an user-defined reducer that returns a list of records by hierarchy entry:

```
public class HierarchyJoinReducer extends MapReduceBase implements Reducer<Text,
RecordInfo, Text, ArrayWritable> {

       private MultipleOutputs mos = null;
       private Text outKey = new Text();
```

```
       private ArrayWritable outValue = new ArrayWritable( RecordInfo.class );

       @Override
       public void configure(JobConf conf)
       {
         super.configure(conf);

         //use MultipleOutputs to generate different outputs for each hierarchy level

         mos = new MultipleOutputs(conf);
        }
        @Override
        public void reduce(Text key, Iterator<RecordInfo> values,
                          OutputCollector<Text, RecordInfoArrayWritable> output,
Reporter reporter)
                          throws IOException
        {

          //Get the hierarchy level name and the hierarchy entry id from the key

          String[] keyComponents =
HierarchyHelper.getMapRedOutputKeyComponents(key.toString());
          String hierarchyLevelName = keyComponents[0];
          String entryId = keyComponents[1];
          List<Writable> records = new LinkedList<Writable>();

          //load the values to memory to fill output ArrayWritable

          while(values.hasNext())
          {
            RecordInfo recordInfo = new RecordInfo( values.next() );
            records.add( recordInfo );
          }
          if(!records.isEmpty())
          {

            //set the hierarchy entry id as key

            outKey.set(entryId);

            //list of records matching the hierarchy entry id

            outValue.set( records.toArray(new Writable[]{} ) );

            //get the named output for the given hierarchy level

            hierarchyLevelName = FileUtils.toValidMONamedOutput(hierarchyLevelName);
            OutputCollector<Text, ArrayWritable> mout =
mos.getCollector(hierarchyLevelName, reporter);

           //Emit key and value

           mout.collect(outKey, outValue);
          }
}

        @Override
        public void close() throws IOException
        {
          mos.close();
```

```
        }
}
```

The same reducer can be used in a job with the following configuration to generate a list of records according to the hierarchy levels:

```
JobConf conf = new JobConf(getConf());

//input path

FileInputFormat.setInputPaths(conf, new Path("/user/data_spatial_index/") );

//output path

FileOutputFormat.setOutputPath(conf, new Path("/user/records_per_hier_level/") );

//input format used to read the spatial index

conf.setInputFormat( SequenceFileInputFormat.class);

//output format: the real output format will be configured for each multiple output
later

conf.setOutputFormat(NullOutputFormat.class);

//mapper

conf.setMapperClass( SpatialHierarchicalJoinMapper.class );
conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(RecordInfo.class);

//reducer

conf.setReducerClass( HierarchyJoinReducer.class );
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(ArrayWritable.class);

////////////////////////////////////////////////////////////////////

//hierarchy data setup

//set HierarchyInfo class implementation

conf.setClass(ConfigParams.HIERARCHY_INFO_CLASS, WorldAdminHierarchyInfo.class,
HierarchyInfo.class);

//paths to hierarchical catalogs

Path[] hierarchyDataPaths = {
new Path("file:///home/user/catalogs/world_continents.json"),
new Path("file:///home/user/catalogs/world_countries.json"),
new Path("file:///home/user/catalogs/world_states_provinces.json")};

//path to hierarchy index

Path hierarchyDataIndexPath = new Path("/user/hierarchy_data_index/");

//instantiate the HierarchyInfo class to index the data if needed.

HierarchyInfo hierarchyInfo = new WorldAdminHierarchyInfo();
hierarchyInfo.initialize(conf);
```

```
//Create the hierarchy index if needed. If it already exists, it will only load the
hierarchy index to the distributed cache

HierarchyHelper.setupHierarchyDataIndex(hierarchyDataPaths, hierarchyDataIndexPath,
hierarchyInfo, conf);

///////////////////////////////////////////////////////////////////

//setup the multiple named outputs:

int levels = hierarchyInfo.getNumberOfLevels();
for(int i=1; i<=levels; i++)
{
    String levelName = hierarchyInfo.getLevelName(i);

    //the hierarchy level name is used as the named output

    String namedOutput = FileUtils.toValidMONamedOutput(levelName);
    MultipleOutputs.addNamedOutput(conf, namedOutput, MapFileOutputFormat.class,
Text.class, ArrayWritable.class);
}

//finally, setup the spatial operation

SpatialOperationConfig spatialOpConf = new SpatialOperationConfig();
spatialOpConf.setOperation(SpatialOperation.IsInside);
spatialOpConf.setSrid(8307);
spatialOpConf.setTolerance(0.5);
spatialOpConf.setGeodetic(true);
spatialOpConf.store(conf);

//run job

JobClient.runJob(conf);
```

Supposing the output value should be an array of record ids instead of an array of `RecordInfo` instances, it would be enough to perform a couple of changes in the previously defined reducer.

The line where `outValue` is declared, in the previous example, changes to:

```
private ArrayWritable outValue = new ArrayWritable(Text.class);
```

The loop where the input values are retrieved, in the previous example, is changed. Therefore, the record ids are got instead of the whole records:

```
while(values.hasNext())
{
  records.add( new Text(values.next().getId()) );
}
```

While only the record id is needed the mapper emits the whole `RecordInfo` instance. Therefore, a better approach is to change the mappers output value. The mappers output value can be changed by extending `AbstractSpatialJoinMapper`. In the following example, the mapper emits only the record ids instead of the whole `RecorInfo` instance every time a record matches some of the hierarchy entries:

```
public class IdSpatialHierarchicalMapper extends AbstractSpatialHierarchicalMapper<
Text >
{
```

```
        Text outValue = new Text();

        @Override
        protected Text getOutValue(RecordInfo matchingRecordInfo)
        {

          //the out value is the record's id

          outValue.set(matchingRecordInfo.getId());
          return outValue;
        }
}
```

2.7.5.1 Changing the Hierarchy Level Range

By default, all the hierarchy levels defined in the `HierarchyInfo` implementation are loaded when performing the hierarchy search. The range of hierarchy levels loaded is from level 1 (parent level) to the level returned by `HierarchyInfo.getNumberOfLevels()` method. The following example shows how to setup a job to only load the levels 2 and 3.

```
conf.setInt( ConfigParams.HIERARCHY_LOAD_MIN_LEVEL, 2);
conf.setInt( ConfigParams.HIERARCHY_LOAD_MAX_LEVEL, 3);
```

Note:

These parameters are useful when only a subset of the hierarchy levels is required and when you do not want to modify the `HierarchyInfo` implementation.

2.7.5.2 Controlling the Search Hierarchy

The search is always performed only at the bottom hierarchy level (the higher level number). If a user record matches some hierarchy entry at this level, then the match is propagated to the parent entry in upper levels. For example, if a user record matches Los Angeles, then it also matches California, USA, and North America. If there are no matches for a user record at the bottom level, then the search does not continue into the upper levels.

This behavior can be modified by setting the configuration parameter `ConfigParams.HIERARCHY_SEARCH_MULTIPLE_LEVELS` to `true`. Therefore, if a search at the bottom hierarchy level resulted in some unmatched user records, then search continues into the upper levels until the top hierarchy level is reached or there are no more user records to join. This behavior can be used when the geometries of parent levels do not perfectly enclose the geometries of their child entries

2.7.5.3 Using MVSuggest to Classify the Data

`MVSuggest` can be used instead of the spatial index to classify data. For this case, an implementation of `LocalizableRecordInfoProvider` must be known and sent to MVSuggest to perform the search. See the information about `LocalizableRecordInfoProvider`.

In the following example, the program option is changed from spatial to MVS. The input is the path to the user data instead of the spatial index. The `InputFormat` used to read the user record and an implementation of `LocalizableRecordInfoProvider` are specified. The `MVSuggest` service configuration is set. Notice that there is no spatial operation configuration needed in this case.

```
Categorization<LongWritable, Text> hierCount = new Categorization<LongWritable,
Text>();

// the input path is the user's data

hierCount.setInput("/user/data/");

// set the job's output

hierCount.setOutput("/user/mvs_hierarchy_count");

// set HierarchyInfo implementation which describes the world
// administrative boundaries hierarchy

hierCount.setHierarchyInfoClass(WorldDynaAdminHierarchyInfo.class);

// specify the paths of the hierarchy data

Path[] hierarchyDataPaths = { new Path("file:///home/user/catalogs/
world_continents.json"),
        new Path("file:///home/user/catalogs/world_countries.json"),
        new Path("file:///home/user/catalogs/world_states_provinces.json") };
hierCount.setHierarchyDataPaths(hierarchyDataPaths);

// set the path where the index for the previous hierarchy data will be
// generated

hierCount.setHierarchyIndexPath(new Path("/user/hierarchy_data_index/"));

// No spatial operation configuration is needed, Instead, specify the
// InputFormat used to read the user's data and the
// LocalizableRecordInfoProvider class.

hierCount.setInputFormatClass(TextInputFormat.class);
hierCount.setRecordInfoProviderClass(MyLocalizableRecordInfoProvider.class);

// finally, set the MVSuggest configuration

LocalMVSConfig lmvsConf = new LocalMVSConfig();
lmvsConf.setServiceLocation("file:///home/user/mvs_dir/oraclemaps_pub");
lmvsConf.setRepositoryLocation(true);
hierCount.setMvsConfig(lmvsConf);

// add the previous setup to the job configuration
hierCount.configure(conf);

// run the job

JobClient.runJob(conf);
```

Note:

When using `MVSuggest`, the hierarchy data files must be the same as the layer template files used by `MVSuggest`. The hierarchy level names returned by the `HierarchyInfo.getLevelNames()` method are used as the matching layers by `MVSuggest`.

2.7.6 Generating Buffers

The API provides a mapper to generate a buffer around each record's geometry. The following code sample shows how to run a job to generate a buffer for each record geometry by using the `BufferMapper` class.

```
//configure input
conf.setInputFormat(FileSplitInputFormat.class);
FileSplitInputFormat.setInputPaths(conf, "/user/waterlines/");
FileSplitInputFormat.setRecordInfoProviderClass(conf,
GeoJsonRecordInfoProvider.class);

//configure output
conf.setOutputFormat(SequenceFileOutputFormat.class);
SequenceFileOutputFormat.setOutputPath(conf, new Path("/user/data_buffer/"));

//set the BufferMapper as the job mapper
conf.setMapperClass(BufferMapper.class);
conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(RecordInfo.class);
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(RecordInfo.class);

//set the width of the buffers to be generated
conf.setDouble(ConfigParams.BUFFER_WIDTH, 0.2);

//run the job
JobClient.runJob(conf);
```

`BufferMapper` generates a buffer for each input record containing a geometry. The output key and values are the record id and a `RecordInfo` instance containing the generated buffer. The resulting file is a Hadoop `MapFile` containing the mapper output key and values. If necessary, the output format can be modified by implementing a reducer that takes the mapper's output keys and values, and outputs keys and values of a different type.

`BufferMapper` accepts the following parameters:

Parameter	ConfigParam constant	Type	Description
oracle.spatial.buffer.width	BUFFER_WIDTH	double	The buffer width
oracle.spatial.buffer.sma	BUFFER_SMA	double	The semi major axis for the datum used in the coordinate system of the input
oracle.spatial.buffer.iFlat	BUFFER_IFLAT	double	The flattening value
oracle.spatial.buffer.arcT	BUFFER_ARCT	double	The arc tolerance used for geodetic densification

2.7.7 Spatial Binning

The Vector API provides the class `oracle.spatial.hadoop.vector.mapred.job.Binning` to perform spatial binning over a spatial data set. The `Binning` class is a MapReduce job driver that takes an input data set (which can be spatially indexed or not), assigns each record to a bin, and generates a file containing all the bins (which contain one or more records and optionally aggregated values).

A binning job can be configured as follows:

1. Specify the data set to be binned and the way it will be read and interpreted (`InputFormat` and `RecordInfoProvider`), or, specify the name of an existing spatial index.
2. Set the output path.
3. Set the grid MBR, that is, the rectangular area to be binned.
4. Set the shape of the bins: `RECTANGLE` or `HEXAGON`.
5. Specify the bin (cell) size. For rectangles, specify the width and height. For hexagon-shaped cells, specify the hexagon width. Each hexagon is always drawn with only one of its vertices as the base.
6. Optionally, pass a list of numeric field names to be aggregated per bin.

The resulting output is a text file where each record is a bin (cell) in JSON format and contains the following information:

- id: the bin id
- geom: the bin geometry; always a polygon that is a rectangle or a hexagon
- count: the number of points contained in the bin
- aggregated fields: zero or more aggregated fields

The following example configures and runs a binning job:

```
//create job driver
Binning<LongWritable, Text> binJob = new Binning<LongWritable, Text>();
//setup input
binJob.setInput("/user/hdfs/input/part*");
binJob.setInputFormatClass(GeoJsonInputFormat.class);
binJob.setRecordInfoProviderClass(GeoJsonRecordInfoProvider.class);
//set binning output
binJob.setOutput("/user/hdfs/output/binning");
//create a binning configuration to produce rectangular cells
BinningConfig binConf = new BinningConfig();
binConf.setShape(BinShape.RECTANGLE);
//set the bin size
binConf.setCellHeight(0.2);
binConf.setCellWidth(0.2);
//specify the area to be binned
binConf.setGridMbr(new double[]{-50,10,50,40});
binJob.setBinConf(binConf);
//save configuration
binJob.configure(conf);
//run job
JobClient.runJob(conf);
```

2.7.8 Spatial Clustering

The job driver class `oracle.spatial.hadoop.mapred.KMeansClustering` can be used to find spatial clusters in a data set. This class uses a distributed version of the K-means algorithm.

Required parameters:

- Path to the input data set, the `InputFormat` class used to read the input data set and the `RecordInfoProvider` used to extract the spatial information from records.
- Path where the results will be stored.
- Number of clusters to be found.

Optional parameters:

- Maximum number of iterations before the algorithm finishes.
- Criterion function used to determine when the clusters converge. It is given as an implementation of `oracle.spatial.hadoop.vector.cluster.kmeans.CriterionFunction`. The Vector API contains the following criterion function implementations: `SquaredErrorCriterionFunction` and `EuclideanDistanceCriterionFunction`.
- An implementation of `oracle.spatial.hadoop.vector.cluster.kmeans.ClusterShapeGenerator`, which is used to generate a geometry for each cluster. The default implementation is `ConvexHullClusterShapeGenerator` and generates a convex hull for each cluster. If no cluster geometry is needed, the `DummyClusterShapeGenerator` class can be used.
- The initial k cluster points as a sequence of x,y ordinates. For example: x1,y1,x2,y2, …xk,yk

The result is a file named `clusters.json`, which contains an array of clusters called features. Each cluster contains the following information:

- id: Cluster id
- memberCount: Number of elements in the cluster
- geom: Cluster geometry

The following example runs the `KMeansClustering` algorithm to find 5 clusters. By default, the `SquredErrorCriterionFunction` and `ConvexHullClusterShapeGenerator` are used , so you do not need yo set these classes explicitly. Also note that `runIterations()` is called to run the algorithm; internally, it launches one MapReduce per iteration. In this example, the number 20 is passed to `runIterations()` as the maximum number of iterations allowed.

```
//create the cluster job driver
KMeansClustering<LongWritable, Text> clusterJob = new KMeansClustering<LongWritable,
Text>();
//set input properties:
//input dataset path
clusterJob.setInput("/user/hdfs/input/part*");
//InputFormat class
```

```
clusterJob.setInputFormatClass(GeoJsonInputFormat.class);
//RecordInfoProvider implementation
clusterJob.setRecordInfoProviderClass(GeoJsonRecordInfoProvider.class);
//specify where the results will be saved
clusterJob.setOutput("/user/hdfs/output/clusters");
//5 cluster will be found
clusterJob.setK(5);
//run the algorithm
success = clusterJob.runIterations(20, conf);
```

2.7.9 Spatial Join

The spatial join feature allows detecting spatial interactions between records of two different large data sets.

The driver class `oracle.spatial.hadoop.vector.mapred.job.SpatialJoin` can be used to execute or configure a job to perform a spatial join between two data sets. The job driver takes the following inputs:

- Input data sets: Two input data sets are expected. Each input data set is represented using the class `oracle.spatial.hadoop.vector.InputDataSet`, which holds information about where to find and how to read a data set, such as path(s), spatial index, input format, and record info provider used to interpret records from the data set. It also accepts a spatial configuration for the data set.
- Spatial operation configuration: The spatial operation configuration defines the spatial interaction used to determine if two records are related to each other. It also defines the area to cover (MBR), that is, only records within or intersecting the MBR will be considered in the search.
- Partitioning result file path: An optional parameter that points to a previously generated partitioning result for both data sets. Data need to be partitioned in order to distribute the work; if this parameter is not provided, a partitioning process will be executed over the input data sets. (See Spatial Partitioning for more information.)
- Output path: The path where the result file will be written.

The spatial join result is a text file where each line is a pair of records that meet the spatial interaction defined in the spatial operation configuration.

The following table shows the currently supported spatial interactions for the spatial join.

Spatial Operation	Extra Parameters	Type
AnyInteract	None	(NA)
IsInside	None	(N/A)
WithinDistance	oracle.spatial.hadoop.vector.util.SpatialOperationConfig.PARAM_WD_DISTANCE	double

For a `WithinDistance` operation, the distance parameter can be specified in the `SpatialOperationConfig`, as shown in the following example:

```
spatialOpConf.setOperation(SpatialOperation.WithinDistance);
spatialOpConf.addParam(SpatialOperationConfig.PARAM_WD_DISTANCE, 5.0);
```

The following example runs a Spatial Join job for two input data sets. The first data set, postal boundaries, is specified providing the name of its spatial index. For the second data set, tweets, the path to the file, input format, and record info provider are specified. The spatial interaction to detect is `IsInside`, so only tweets (points) that are inside a postal boundary (polygon) will appear in the result along with their containing postal boundary.

```
SpatialJoin spatialJoin = new SpatialJoin();
List<InputDataSet> inputDataSets = new ArrayList<InputDataSet>(2);

// set the spatial index of the 3-digit postal boundaries of the USA as the first
input data set
InputDataSet pbInputDataSet = new InputDataSet();
pbInputDataSet.setIndexName("usa_pcb3_index");

//no input format or record info provider are required here as a spatial index is
provided
inputDataSets.add(pbInputDataSet);

// set the tweets data set in GeoJSON format as the second data set
InputDataSet tweetsDataSet = new InputDataSet();
tweetsDataSet.setPaths(new Path[]{new Path("/user/example/tweets.json")});
tweetsDataSet.setInputFormatClass(GeoJsonInputFormat.class);
tweetsDataSet.setRecordInfoProviderClass(GeoJsonRecordInfoProvider.class);
inputDataSets.add(tweetsDataSet);

//set input data sets
spatialJoin.setInputDataSets(inputDataSets);

//spatial operation configuration
SpatialOperationConfig spatialOpConf = new SpatialOperationConfig();
spatialOpConf.setOperation(SpatialOperation.IsInside);
spatialOpConf.setBoundaries(new double[]{47.70, -124.28, 35.45, -95.12});
spatialOpConf.setSrid(8307);
spatialOpConf.setTolerance(0.5);
spatialOpConf.setGeodetic(true);
spatialJoin.setSpatialOperationConfig(spatialOpConf);

//set output path
spatialJoin.setOutput("/user/example/spatialjoin");

// prepare job
JobConf jobConf = new JobConf(getConf());

//preprocess will partition both data sets as no partitioning result file was
specified
spatialJoin.preprocess(jobConf);
spatialJoin.configure(jobConf);
JobClient.runJob(jobConf);
```

2.7.10 Spatial Partitioning

The partitioning feature is used to spatially partition one or more data sets.

Spatial partitioning consists of dividing the space into multiple rectangles, where each rectangle is intended to contain approximately the same number of points. Eventually these partitions can be used to distribute the work among reducers in other jobs, such as Spatial Join.

The spatial partitioning process is run or configured using the `oracle.spatial.hadoop.mapred.job.Partitioning` driver class, which accepts the following input parameters:

- Input data sets: One or more input data sets can be specified. Each input data set is represented using the class `oracle.spatial.hadoop.vector.InputDataSet`, which holds information about where to find and how to read a data set, such as path(s), spatial index, input format, and record info provider used to interpret records from the data set. It also accepts a spatial configuration for the data set.
- Sampling ratio: Only a fraction of the entire data set or sets is used to perform the partitioning. The sample ratio is the ratio of the sample size to the whole input data set size. If it is not specified, 10 percent (0.1) of the input data set size is used.
- Spatial configuration: Defines the spatial properties of the input data sets, such as the SRID. You must specify at least the dimensional boundaries.
- Output path: The path where the result file will be written.

The generated partitioning result file is in GeoJSON format and contains information for each generated partition, including the partition's geometry and the number of points contained (from the sample).

The following example partitions a tweets data set. Because the sampling ratio is not provided, 0.1 is used by default.

```
Partitioning partitioning = new Partitioning();
List<InputDataSet> inputDataSets = new ArrayList<InputDataSet>(1);

//define the input data set
InputDataSet dataSet = new InputDataSet();
dataSet.setPaths(new Path[]{new Path("/user/example/tweets.json")});
dataSet.setInputFormatClass(GeoJsonInputFormat.class);
dataSet.setRecordInfoProviderClass(GeoJsonRecordInfoProvider.class);
inputDataSets.add(dataSet);
partitioning.setInputDataSets(inputDataSets);

//spatial configuration
SpatialConfig spatialConf = new SpatialConfig();
spatialConf.setSrid(8307);
spatialConf.setBoundaries(new double[]{-180,-90,180,90});
partitioning.setSpatialConfig(spatialConf);

//set output
partitioning.setOutput("/user/example/tweets_partitions.json");

//run the partitioning process
partitioning.runFullPartitioningProcess(new JobConf());
```

2.7.11 RecordInfoProvider

A record read by a MapReduce job from HDFS is represented in memory as a key-value pair using a Java type (typically) Writable subclass, such as LongWritable, Text, ArrayWritable or some user-defined type. For example, records read using TextInputFormat are represented in memory as LongWritable, Text key-value pairs.

`RecordInfoProvider` is the component that interprets these memory record representations and returns the data needed by the Vector Analysis API. Thus, the API is not tied to any specific format and memory representations.

The `RecordInfoProvider` interface has the following methods:

- void setCurrentRecord(K key, V value)
- String getId()
- JGeometry getGeometry()
- boolean getExtraFields(Map<String, String> extraFields)

There is always a `RecordInfoProvider` instance per `InputFormat`. The method `setCurrentRecord()` is called passing the current key-value pair retrieved from the `RecordReader`. The `RecordInfoProvider` is then used to get the current record id, geometry, and extra fields. None of these fields are required fields. Only those records with a geometry participates in the spatial operations. The Id is useful for differentiating records in operations such as categorization. The extra fields can be used to store any record information that can be represented as text and which is desired to be quickly accessed without reading the original record, or for operations where `MVSuggest` is used.

Typically, the information returned by RecordInfoProvider is used to populate `RecordInfo` instances. A RecordInfo can be thought as a light version of a record and contains the information returned by the RecordInfoProvider plus information to locate the original record in a file.

2.7.11.1 Sample RecordInfoProvider Implementation

This sample implementation, called `JsonRecordInfoProvider`, takes text records in JSON format, which are read using `TextInputFormat`. A sample record is shown here:

```
{ "_id":"ABCD1234", "location":" 119.31669, -31.21615", "locationText":"Boston, Ma",
"date":"03-18-2015", "time":"18:05", "device-type":"cellphone", "device-
name":"iPhone"}
```

When a JsonRecordInfoProvider is instantiated, a JSON ObjectMapper is created. The ObjectMapper is used to parse records values later when `setCurrentRecord()` is called. The record key is ignored. The record id, geometry, and one extra field are retrieved from the _id, location and locationText JSON properties. The geometry is represented as latitude-longitude pair and is used to create a point geometry using `JGeometry.createPoint()` method. The extra field (locationText) is added to the extraFields map, which serves as an out parameter and true is returned indicating that an extra field was added.

```
public class JsonRecordInfoProvider implements RecordInfoProvider<LongWritable,
Text> {
private Text value = null;
private ObjectMapper jsonMapper = null;
private JsonNode recordNode = null;

public JsonRecordInfoProvider(){

//json mapper used to parse all the records

jsonMapper = new ObjectMapper();

}

@Override
public void setCurrentRecord(LongWritable key, Text value) throws Exception {
```

```
        try{

           //parse the current value

           recordNode = jsonMapper.readTree(value.toString());
           }catch(Exception ex){
              recordNode = null;
              throw ex;
        }
}

@Override
public String getId() {
        String id = null;
        if(recordNode != null ){
                id = recordNode.get("_id").getTextValue();
        }
        return id;
}
@Override
public JGeometry getGeometry() {
        JGeometry geom = null;
        if(recordNode!= null){
                //location is represented as a lat,lon pair
                String location = recordNode.get("location").getTextValue();
                String[] locTokens = location.split(",");
                double lat = Double.parseDouble(locTokens[0]);
                double lon = Double.parseDouble(locTokens[1]);
                geom =  JGeometry.createPoint( new double[]{lon, lat},  2, 8307);
        }
        return geom;
}

@Override
public boolean getExtraFields(Map<String, String> extraFields) {
        boolean extraFieldsExist = false;
        if(recordNode != null) {
                extraFields.put("locationText",
recordNode.get("locationText").getTextValue() );
                extraFieldsExist = true;
        }
        return extraFieldsExist;
}
}
```

2.7.11.2 LocalizableRecordInfoProvider

This interface extends `RecordInfoProvider` and is used to know the extra fields that can be used as the search text, when `MVSuggest` is used.

The only method added by this interface is `getLocationServiceField()`, which returns the name of the extra field that will be sent to MVSuggest.

In addition, the following is an implementation based on "Sample RecordInfoProvider Implementation." The name returned in this example is `locationText`, which is the name of the extra field included in the parent class.

```
public class LocalizableJsonRecordInfoProvider extends JsonRecordInfoProvider
implements LocalizableRecordInfoProvider<LongWritable, Text> {

@Override
public String getLocationServiceField() {
```

```
    return  "locationText";
}
}
```

An alternative to `LocalizableRecordInfoProvider` is to set the configuration property `oracle.spatial.recordInfo.locationField` with the name of the search field, which value should be sent to `MVSuggest`. Example: `configuration.set(LocatizableRecordInfoProvider.CONF_RECORD_INFO_LOCATION_FIELD, “locationField”)`

2.7.12 HierarchyInfo

The `HierarchyInfo` interface is used to describe a hierarchical dataset. This implementation of HierarchyInfo is expected to provide the number, names, and the entries of the hierarchy levels of the hierarchy it describes.

The root hierarchy level is always the hierarchy level 1. The entries in this level do not have parent entries and this level is referred as the top hierarchy level. Children hierarchy levels will have higher level values. For example: the levels for the hierarchy conformed by continents, countries, and states are 1, 2 and 3 respectively. Entries in the continent layer do not have a parent, but have children entries in the countries layer. Entries at the bottom level, the states layer, do not have children.

A HierarchyInfo implementation is provided out of the box with the Vector Analysis API. The `DynaAdminHierarchyInfo` implementation can be used to read and describe the known hierarchy layers in GeoJSON format. A DynaAdminHierarchyInfo can be instantiated and configured or can be subclassed. The hierarchy layers to be contained are specified by calling the `addLevel()` method, which takes the following parameters:

- The hierarchy level number
- The hierarchy level name, which must match the file name (without extension) of the GeoJSON file that contains the data. For example, the hierarchy level name for the file `world_continents.json` must be `world_continents`, for `world_countries.json` it is `world_countries`, and so on.
- Children join field: This is a JSON property that is used to join entries of the current level with child entries in the lower level. If a null is passed, then the entry id is used.
- Parent join field: This is a JSON property used to join entries of the current level with parent entries in the upper level. This value is not used for the top most level without an upper level to join. If the value is set null for any other level greater than 1, an `IsInside` spatial operation is performed to join parent and child entries. In this scenario, it is supposed that an upper level geometry entry can contain lower level entries.

For example, let us assume a hierarchy containing the following levels from the specified layers: 1- world_continents, 2 - world_countries and 3 - world_states_provinces. A sample entry from each layer would look like the following:

```
world_continents:
 {"type":"Feature","_id":"NA","geometry": {"type":"MultiPolygon", "coordinates":
[ x,y,x,y,x,y] }"properties":{"NAME":"NORTH AMERICA", "CONTINENT_LONG_LABEL":"North
America"},"label_box":[-118.07998,32.21006,-86.58515,44.71352]}

world_countries: {"type":"Feature","_id":"iso_CAN","geometry":
```

```
{"type":"MultiPolygon","coordinates":[x,y,x,y,x,y]},"properties":
{"NAME":"CANADA","CONTINENT":"NA","ALT_REGION":"NA","COUNTRY
CODE":"CAN"},"label_box":[-124.28092,49.90408,-94.44878,66.89287]}

world_states_provinces:
{"type":"Feature","_id":"6093943","geometry": {"type":"Polygon", "coordinates":
[ x,y,x,y,x,y]},"properties":{"COUNTRY":"Canada", "ISO":"CAN",
"STATE_NAME":"Ontario"},"label_box":[-91.84903,49.39557,-82.32462,54.98426]}
```

A DynaAdminHierarchyInfo can be configured to create a hierarchy with the above layers in the following way:

```
DynaAdminHierarchyInfo dahi = new DynaAdminHierarchyInfo();

dahi.addLevel(1, "world_continents", null /*_id is used by default to join with
child entries*/, null /*not needed as there are not upper hierarchy levels*/);

dahi.addLevel(2, "world_countries", "properties.COUNTRY CODE"/*field used to join
with child entries*/, "properties.CONTINENT" /*the value "NA" will be used to find
Canada's parent which is North America and which _id field value is also "NA" */);

dahi.addLevel(3, "world_states_provinces", null /*not needed as not child entries
are expected*/, "properties.ISO"/*field used to join with parent entries. For
Ontario, it is the same value than the field properties.COUNTRY CODE specified for
Canada*/);

//save the previous configuration to the job configuration

dahi.initialize(conf);
```

A similar configuration can be used to create hierarchies from different layers, such as countries, states and counties, or any other layers with a similar JSON format.

Alternatively, to avoid configuring a hierarchy every time a job is executed, the hierarchy configuration can be enclosed in a DynaAdminHierarchyInfo subclass as in the following example:

```
public class WorldDynaAdminHierarchyInfo extends DynaAdminHierarchyInfo \

{
       public WorldDynaAdminHierarchyInfo()

       {
              super();
              addLevel(1, "world_continents", null, null);
              addLevel(2, "world_countries", "properties.COUNTRY CODE",
"properties.CONTINENT");
              addLevel(3, "world_states_provinces", null, "properties.ISO");
       }

}
```

2.7.12.1 Sample HierarchyInfo Implementation

The `HierarchyInfo` interface contains the following methods, which must be implemented to describe a hierarchy. The methods can be divided in to the following three categories:

- Methods to describe the hierarchy
- Methods to load data

- Methods to supply data

Additionally there is an `initialize()` method, which can be used to perform any initialization and to save and read data both to and from the job configuration

```
void initialize(JobConf conf);

//methods to describe the hierarchy

String getLevelName(int level);
int getLevelNumber(String levelName);
int getNumberOfLevels();

//methods to load data

void load(Path[] hierDataPaths, int fromLevel, JobConf conf) throws Exception;
void loadFromIndex(HierarchyDataIndexReader[] readers, int fromLevel, JobConf conf)
throws Exception;

//methods to supply data

Collection<String> getEntriesIds(int level);
JGeometry getEntryGeometry(int level, String entryId);
String getParentId(int childLevel, String childId);
```

The following is a sample HierarchyInfo implementation, which takes the previously mentioned world layers as the hierarchy levels. The first section contains the initialize method and the methods used to describe the hierarchy. In this case, the initialize method does nothing. The methods mentioned in the following example use the `hierarchyLevelNames` array to provide the hierarchy description. The instance variables `entriesGeoms` and `entriesParent` are arrays of `java.util.Map`, which contains the entries geometries and entries parents respectively. The entries ids are used as keys in both cases. Since the arrays indices are zero-based and the hierarchy levels are one-based, the array indices correlate to the hierarchy levels as *array index + 1 = hierarchy level*.

```
public class WorldHierarchyInfo implements HierarchyInfo
{

       private String[] hierarchyLevelNames = {"world_continents",
"world_countries", "world_states_provinces"};
       private Map<String, JGeometry>[] entriesGeoms = new Map[3];
       private Map<String, String>[] entriesParents = new Map[3];

       @Override
       public void initialize(JobConf conf)
      {

         //do nothing for this implementation
}

        @Override
        public int getNumberOfLevels()
        {
          return hierarchyLevelNames.length;
}

        @Override
        public String getLevelName(int level)
        {
           String levelName = null;
```

```
          if(level >=1 && level <= hierarchyLevelNames.length)
          {
            levelName = hierarchyLevelNames[ level - 1];
          }
        return levelName;
        }

       @Override
       public int getLevelNumber(String levelName)
       {
          for(int i=0; i< hierarchyLevelNames.length; i++ )
          {
            if(hierarchyLevelNames.equals( levelName) ) return i+1;
   }
   return -1;
}
```

The following example contains the methods that load the different hierarchy levels data. The `load()` method reads the data from the source files `world_continents.json`, `world_countries.json`, and `world_states_provinces.json`. For the sake of simplicity, the internally called `loadLevel()` method is not specified, but it is supposed to parse and read the JSON files.

The `loadFromIndex()` method only takes the information provided by the `HierarchyIndexReader` instances passed as parameters. The `load()` method is supposed to be executed only once and only if a hierarchy index has not been created, in a job. Once the data is loaded, it is automatically indexed and `loadFromIndex()` method is called every time the hierarchy data is loaded into the memory.

```
      @Override
      public void load(Path[] hierDataPaths, int fromLevel, JobConf conf) throws
Exception {
      int toLevel = fromLevel + hierDataPaths.length - 1;
      int levels = getNumberOfLevels();

      for(int i=0, level=fromLevel; i<hierDataPaths.length && level<=levels; i++,
level++)
      {

         //load current level from the current path

         loadLevel(level, hierDataPaths[i]);
        }
    }

    @Override
    public void loadFromIndex(HierarchyDataIndexReader[] readers, int fromLevel,
JobConf conf)
                throws Exception
    {
     Text parentId = new Text();
     RecordInfoArrayWritable records = new RecordInfoArrayWritable();
     int levels = getNumberOfLevels();

     //iterate through each reader to load each level's entries

     for(int i=0, level=fromLevel; i<readers.length && level<=levels; i++, level++)
     {
       entriesGeoms[ level - 1 ] = new Hashtable<String, JGeometry>();
       entriesParents[ level - 1 ] = new Hashtable<String, String>();
```

```
       //each entry is a parent record id (key) and a list of entries as RecordInfo
(value)

       while(readers[i].nextParentRecords(parentId, records))
       {
          String pId = null;

          //entries with no parent will have the parent id UNDEFINED_PARENT_ID. Such
is the case of the first level entries

           if( ! UNDEFINED_PARENT_ID.equals( parentId.toString() ) )
           {
           pId = parentId.toString();
           }

         //add the current level's entries

           for(Object obj : records.get())
           {
              RecordInfo entry = (RecordInfo) obj;
              entriesGeoms[ level - 1 ].put(entry.getId(), entry.getGeometry());
              if(pId != null)
              {
              entriesParents[ level -1 ].put(entry.getId(), pId);
              }
           }//finishin loading current parent entries
        }//finish reading single hierarchy level index
     }//finish iterating index readers
}
```

Finally, the following code listing contains the methods used to provide information of individual entries in each hierarchy level. The information provided is the ids of all the entries contained in a hierarchy level, the geometry of each entry, and the parent of each entry.

```
@Override
public Collection<String> getEntriesIds(int level)
{
   Collection<String> ids = null;

   if(level >= 1 && level <= getNumberOfLevels() && entriesGeoms[ level - 1 ] !=
null)
   {

     //returns the ids of all the entries from the given level

     ids = entriesGeoms[ level - 1 ].keySet();
   }
   return ids;
}

@Override
public JGeometry getEntryGeometry(int level, String entryId)
{
   JGeometry geom = null;
   if(level >= 1 && level <= getNumberOfLevels() && entriesGeoms[ level - 1 ] !=
null)
   {

      //returns the geometry of the entry with the given id and level
```

```
      geom = entriesGeoms[ level - 1 ].get(entryId);
    }
    return geom;
}

@Override
public String getParentId(int childLevel, String childId)
{
   String parentId = null;
   if(childLevel >= 1 && childLevel <= getNumberOfLevels() &&
entriesGeoms[ childLevel - 1 ] != null)
   {

      //returns the parent id of the entry with the given id and level

      parentId = entriesParents[ childLevel - 1 ].get(childId);
   }
   return parentId;
   }
}//end of class
```

2.7.13 Using JGeometry in MapReduce Jobs

The Spatial Hadoop Vector Analysis only contains a small subset of the functionality provided by the Spatial Java API, which can also be used in the MapReduce jobs. This section provides some simple examples of how JGeometry can be used in Hadoop for spatial processing. The following example contains a simple mapper that performs the `IsInside` test between a dataset and a query geometry using the JGeometry class.

In this example, the query geometry ordinates, srid, geodetic value and tolerance used in the spatial operation are retrieved from the job configuration in the configure method. The query geometry, which is a polygon, is preprocessed to quickly perform the `IsInside` operation.

The map method is where the spatial operation is executed. Each input record value is tested against the query geometry and the id is returned, when the test succeeds.

```
public class IsInsideMapper extends MapReduceBase implements Mapper<LongWritable,
Text, NullWritable, Text>
{
       private JGeometry queryGeom = null;
       private int srid = 0;
       private double tolerance = 0.0;
       private boolean geodetic = false;
       private Text outputValue = new Text();
       private double[] locationPoint = new double[2];


       @Override
       public void configure(JobConf conf)
       {
           super.configure(conf);
           srid = conf.getInt("srid", 8307);
           tolerance = conf.getDouble("tolerance", 0.0);
           geodetic = conf.getBoolean("geodetic", true);

    //The ordinates are represented as a string of comma separated double values

            String[] ordsStr = conf.get("ordinates").split(",");
            double[] ordinates = new double[ordsStr.length];
```

```
            for(int i=0; i<ordsStr.length; i++)
            {
              ordinates[i] = Double.parseDouble(ordsStr[i]);
             }

    //create the query geometry as two-dimensional polygon and the given srid

             queryGeom = JGeometry.createLinearPolygon(ordinates, 2, srid);

    //preprocess the query geometry to make the IsInside operation run faster

              try
              {
                queryGeom.preprocess(tolerance, geodetic,
EnumSet.of(FastOp.ISINSIDE));
               }
               catch (Exception e)
               {
                 e.printStackTrace();
                }

          }

          @Override
          public void map(LongWritable key, Text value,
                      OutputCollector<NullWritable, Text> output, Reporter reporter)
                      throws IOException
          {

     //the input value is a comma separated values text with the following columns:
id, x-ordinate, y-ordinate

           String[] tokens = value.toString().split(",");

     //create a geometry representation of the record's location

            locationPoint[0] = Double.parseDouble(tokens[1]);//x ordinate
            locationPoint[1] = Double.parseDouble(tokens[2]);//y ordinate
            JGeometry location = JGeometry.createPoint(locationPoint, 2, srid);

     //perform spatial test

            try
            {
              if( location.isInside(queryGeom, tolerance, geodetic)){

               //emit the record's id

               outputValue.set( tokens[0] );
               output.collect(NullWritable.get(), outputValue);
             }
     }
             catch (Exception e)
             {
                e.printStackTrace();
              }
}
}
```

A similar approach can be used to perform a spatial operation on the geometry itself. For example, by creating a buffer. The following example uses the same text value format and creates a buffer around each record location. The mapper output key and value are the record id and the generated buffer, which is represented as a `JGeometryWritable`. The `JGeometryWritable` is a Writable implementation contained in the Vector Analysis API that holds a JGeometry instance.

```
public class BufferMapper extends MapReduceBase implements Mapper<LongWritable,
Text, Text, JGeometryWritable>
{
       private int srid = 0;
       private double bufferWidth = 0.0;
       private Text outputKey = new Text();
       private JGeometryWritable outputValue = new JGeometryWritable();
       private double[] locationPoint = new double[2];

       @Override
       public void configure(JobConf conf)
       {
              super.configure(conf);
              srid = conf.getInt("srid", 8307);

              //get the buffer width

              bufferWidth = conf.getDouble("bufferWidth", 0.0);
        }

        @Override
        public void map(LongWritable key, Text value,
               OutputCollector<Text, JGeometryWritable> output, Reporter reporter)
               throws IOException
        {

               //the input value is a comma separated record with the following
columns: id, longitude, latitude

               String[] tokens = value.toString().split(",");

               //create a geometry representation of the record's location

               locationPoint[0] = Double.parseDouble(tokens[1]);
               locationPoint[1] = Double.parseDouble(tokens[2]);
               JGeometry location = JGeometry.createPoint(locationPoint, 2, srid);

               try
               {

                  //create the location's buffer

                  JGeometry buffer = location.buffer(bufferWidth);

                  //emit the record's id and the generated buffer

                  outputKey.set( tokens[0] );
                  outputValue.setGeometry( buffer );
                  output.collect(outputKey, outputValue);
                }

                catch (Exception e)
                {
                   e.printStackTrace();
                 }
```

```
        }
}
```

2.7.14 Tuning Performance Data of Job Running Times using Vector Analysis API

The table lists some running times for jobs built using the Vector Analysis API. The jobs were executed using a 4-node cluster. The times may vary depending on the characteristics of the cluster. The test dataset contains over One billion records and the size is above 1 terabyte.

Table 2-2 Performance time for running jobs using Vector Analysis API

Job Type	Time taken (approximate value)
Spatial Indexing	2 hours
Spatial Filter with Spatial Index	1 hour
Spatial Filter without Spatial Index	3 hours
Hierarchy count with Spatial Index	5 minutes
Hierarchy count without Spatial Index	3 hours

The time taken for the jobs can be decreased by increasing the maximum split size using any of the following configuration parameters.

```
mapred.max.split.size
mapreduce.input.fileinputformat.split.maxsize
```

This results in more splits are being processed by each single mapper and improves the execution time. This is done by using the `SpatialFilterInputFormat` (spatial indexing) or `FileSplitInputFormat` (spatial hierarchical join, buffer). Also, the same results can be achieved by using the implementation of `CombineFileInputFormat` as internal `InputFormat`.

2.8 Oracle Big Data Spatial Vector Hive Analysis

Oracle Big Data Spatial Vector Hive Analysis provides spatial functions to analyze the data using Hive. The spatial data can be in any Hive supported format. You can also use a spatial index created with the Java analysis API (see Spatial Indexing) for fast processing.

The supported features include:

- Using the Hive Spatial API
- Using Spatial Indexes in Hive

See also HiveRecordInfoProvider for details about the implementation of these features.

Hive Spatial Functions provides reference information about the available functions.

Prerequisite Libraries

The following libraries are required by the Spatial Vector Hive Analysis API.

- `sdohadoop-vector-hive.jar`

- `sdohadoop-vector.jar`
- `sdoutil.jar`
- `sdoapi.jar`
- `ojdbc.jar`

2.8.1 HiveRecordInfoProvider

A record in a Hive table may contain a geometry field in any format like JSON, WKT, or a user-specifiedformat. Geometry constructors like ST_Geometry can create a geometry receiving the GeoJSON, WKT, or WKB representation of the geometry. If the geometry is stored in another format, a HiveRecordInfoProvider can be used.

`HiveRecordInfoProvider` is a component that interprets the geometry field representation and returns the geometry in a GeoJSON format.

The returned geometry must contain the geometry SRID, as in the following example format:

```
{"type":<geometry-type", "crs": {"type": "name", "properties": {"name": "EPSG:
4326"}}"coordinates":[c1,c2,....cn]}
```

The `HiveRecordInfoProvider` interface has the following methods:

- `void setCurrentRecord(Object record)`
- `String getGeometry()`

The method `setCurrentRecord()` is called by passing the current geometry field provided when creating a geometry in Hive. The `HiveRecordInfoProvider` is used then to get the geometry or to return null if the record has no spatial information.

The information returned by the `HiveRecordInfoProvider` is used by the Hive Spatial functions to create geometries (see Hive Spatial Functions).

Sample HiveRecordInfoProvider Implementation

This sample implementation, named `SimpleHiveRecordInfoProvider`, takes text records in JSON format. The following is a sample input record:

```
{"longitude":-71.46, "latitude":42.35}
```

When `SimpeHiveRecordInfoProvider` is instantiated, a JSON `ObjectMapper` is created. The `ObjectMapper` is used to parse records values later when `setCurrentRecord()` is called. The geometry is represented as latitude-longitude pair, and is used to create a point geometry using the `JsonUtils.readGeometry()` method. Then the GeoJSON format to be returned is created using `GeoJsonGen.asGeometry()`, and the SRID is added to the GeoJSON using `JsonUtils.addSRIDToGeoJSON().`

```
public class SimpleHiveRecordInfoProvider implements HiveRecordInfoProvider{
  private static final Log LOG =
    LogFactory.getLog(SimpleHiveRecordInfoProvider.class.getName());

  private JsonNode recordNode = null;
  private ObjectMapper jsonMapper = null;

  public SimpleHiveRecordInfoProvider(){
    jsonMapper = new ObjectMapper();
  }
```

```
  @Override
  public void setCurrentRecord(Object record) throws Exception {
    try{
      if(record != null){
        //parse the current value
        recordNode = jsonMapper.readTree(record.toString());
      }
    }catch(Exception ex){
      recordNode = null;
      LOG.warn("Problem reading JSON record
        value:"+record.toString(), ex);
    }
  }

  @Override
  public String getGeometry() {
    if(recordNode == null){
      return null;
    }

    JGeometry geom = null;

    try{
      geom = JsonUtils.readGeometry(recordNode,
          2, //dimensions
          8307 //SRID
          );
    }catch(Exception ex){
      recordNode = null;
      LOG.warn("Problem reading JSON record
        geometry:"+recordNode.toString(), ex);
    }

    if(geom != null){
      StringBuilder res = new StringBuilder();
      //Get a GeoJSON representation of the JGeometry
      GeoJsonGen.asGeometry(geom, res);
      String result = res.toString();
      //add SRID to GeoJSON and return the result
      return JsonUtils.addSRIDToGeoJSON(result, 8307);
    }

     return null;
  }
}
```

2.8.2 Using the Hive Spatial API

The Hive Spatial API consists of Oracle-supplied Hive User Defined Functions that can be used to create geometries and perform operations using one or two geometries.

The functions can be grouped into logical categories: types, single-geometry, and two-geometries. (Hive Spatial Functions lists the functions in each category and provides reference information about each function.)

Example 2-1 Hive Script

The following example script returns information about Twitter users in a data set who are within a specified geographical polygon and who have more than 50 followers. It does the following:

1. Adds the necessary jar files:

   ```
   add jar
     /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/ojdbc8.jar
     /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoutl.jar
     /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoapi.jar
     /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector.jar
     /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector-hive.jar;
   ```

2. Creates the Hive user-defined functions that will be used:

   ```
   create temporary function ST_Point as
   'oracle.spatial.hadoop.vector.hive.ST_Point';
   create temporary function ST_Polygon as
   'oracle.spatial.hadoop.vector.hive.ST_Polygon';
   create temporary function ST_Contains as
   'oracle.spatial.hadoop.vector.hive.function.ST_Contains';
   ```

3. Creates a Hive table based on the files under the HDFS directory /user/oracle/twitter. The `InputFormat` used in this case is `org.apache.hadoop.mapred.TextInputFormat` and the Hive SerDe is a user-provided SerDe `package.TwitterSerDe`. The geometry of the tweets will be saved in the geometry column with the format {"longitude":*n*, "latitude":*n*} :

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS sample_tweets (id STRING, geometry STRING,
   followers_count STRING, friends_count STRING, location
   STRING)
   ROW FORMAT SERDE 'package.TwitterSerDe'
   STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
   OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
   LOCATION '/user/oracle/twitter';
   ```

4. Runs a a spatial query receiving an ST_Polygon query area and the ST_Point tweets geometry, and using 0.5 as the tolerance value for the spatial operation. The HiveRecordInfoProvider implementation that will translate the custom geometry to GeoJSON format is the one described in HiveRecordInfoProvider. The output will be information about Twitter users in the query area who have more than 50 followers.

   ```
   SELECT id, followers_count, friends_count, location FROM sample_tweets
   WHERE ST_Contains(
     ST_Polygon('{"type": "Polygon","coordinates": [[[-106, 25], [-106, 30], [-104,
   30], [-104, 25], [-106, 25]]]}', 8307)
     , ST_Point(geometry, 'package.SimpleHiveRecordInfoProvider')
     , 0.5)
     and followers_count > 50;
   ```

The complete script is as follows:

```
add jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/ojdbc8.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoutl.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoapi.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector-hive.jar;

create temporary function ST_Point as 'oracle.spatial.hadoop.vector.hive.ST_Point';
create temporary function ST_Polygon as
'oracle.spatial.hadoop.vector.hive.ST_Polygon';
create temporary function ST_Contains as
'oracle.spatial.hadoop.vector.hive.function.ST_Contains';
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS sample_tweets (id STRING, geometry STRING,
followers_count STRING, friends_count STRING, location
STRING)
ROW FORMAT SERDE 'package.TwitterSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION '/user/oracle/twitter';


SELECT id, followers_count, friends_count, location FROM sample_tweets
WHERE ST_Contains(
  ST_Polygon('{"type": "Polygon","coordinates": [[[-106, 25], [-106, 30], [-104,
30], [-104, 25], [-106, 25]]]}', 8307)
  , ST_Point(geometry, 'package.SimpleHiveRecordInfoProvider')
  , 0.5)
  and followers_count > 50;
```

2.8.3 Using Spatial Indexes in Hive

Hive spatial queries can use a previously created spatial index, which you can create using the Java API (see Spatial Indexing).

If you do not need to use the index in API functions that will access the original data, you can specify `isMapFileIndex=false` when you call `oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing`, or you can use the function `setMapFileIndex(false)`. In these cases, the index will have the following structure:

```
HDFSIndexDirectory/part-xxxxx
```

And in these cases, when creating a Hive table, just provide the folder where you created the index.

If you need to access the original data and you do not set the parameter `isMapFileIndex=false`, the index structure is as follows:

```
part-xxxxx
  data
  index
```

In such cases, to create a Hive table, the data files of the index are needed. Copy the `data` files into a new HDFS folder, with each data file having a different name, like data1, data2,, and so on. The new folder will be used to create the Hive table.

The index contains the geometry records and extra fields. That data can be used when creating the Hive table.

(Note that Spatial Indexing Class Structure describes the index structure, and RecordInfoProvider provides an example of a `RecordInfoProvider` adding extra fields.)

`InputFormat` `oracle.spatial.hadoop.vector.mapred.input.SpatialIndexTextInputFormat` will be used to read the index. The output of this `InputFormat` is GeoJSON.

Before running any query, you can specify a minimum bounding rectangle (MBR) that will perform a first data filtering using `SpatialIndexTextInputFormat`..

Example 2-2 Hive Script Using a Spatial Index

The following example script returns information about Twitter users in a data set who are within a specified geographical polygon and who have more than 50 followers. It does the following:

1. Adds the necessary jar files:

```
add jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/ojdbc8.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoutl.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoapi.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector-hive.jar;
```

2. Creates the Hive user-defined functions that will be used:

```
create temporary function ST_Point as
'oracle.spatial.hadoop.vector.hive.ST_Point';
create temporary function ST_Polygon as
'oracle.spatial.hadoop.vector.hive.ST_Polygon';
create temporary function ST_Contains as
'oracle.spatial.hadoop.vector.hive.function.ST_Contains';
```

3. Sets the data maximum and minimum boundaries (dim1Min,dim2Min,dim1Max,dim2Max):

```
set oracle.spatial.boundaries=-180,-90,180,90;
```

4. Sets the extra fields contained in the spatial index that will be included in the table creation:

```
set
oracle.spatial.index.includedExtraFields=followers_count,friends_count,location;
```

5. Creates a Hive table based on the files under the HDFS directory /user/oracle/twitter. The `InputFormat` used in this case is `oracle.spatial.hadoop.vector.mapred.input.SpatialIndexTextInputFormat` and the Hive SerDe is a user-provided SerDe `oracle.spatial.hadoop.vector.hive.json.GeoJsonSerDe`. (The code for `oracle.spatial.hadoop.vector.hive.json.GeoJsonSerDe` is included with the Hive examples.) The geometry of the tweets will be saved in the geometry column with the format {"longitude":*n,* "latitude":*n*} :

```
CREATE EXTERNAL TABLE IF NOT EXISTS sample_tweets_index (id STRING, geometry
STRING, followers_count STRING, friends_count STRING, location
STRING)
ROW FORMAT SERDE
'oracle.spatial.hadoop.vector.hive.json.GeoJsonSerDe'
STORED AS INPUTFORMAT
'oracle.spatial.hadoop.vector.mapred.input.SpatialIndexTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION '/user/oracle/twitter/index';
```

6. Defines the minimum bounding rectangle (MBR) to filter in the `SpatialIndexTextInputFormat`. Any spatial query will only have access to the data in this MBR. If no MBR is specified, then the data boundaries will be used. This setting is recommended to improve the performance.

```
set oracle.spatial.spatialQueryWindow={"type": "Polygon","coordinates": [[[-107,
24], [-107, 31], [-103, 31], [-103, 24], [-107, 24]]]};
```

7. Runs a a spatial query receiving an ST_Polygon query area and the ST_Point tweets geometry, and using 0.5 as the tolerance value for the spatial operation. The tweet geometries are in GeoJSON format, and the ST_Point function is usedspecifying the SRID as 8307.. The output will be information about Twitter users in the query area who have more than 50 followers.

   ```
   SELECT id, followers_count, friends_count, location FROM sample_tweets
   WHERE ST_Contains(
     ST_Polygon('{"type": "Polygon","coordinates": [[[-106, 25], [-106, 30], [-104,
   30], [-104, 25], [-106, 25]]]}', 8307)
     , ST_Point(geometry, 8307)
     , 0.5)
     and followers_count > 50;
   ```

The complete script is as follows. (Differences between this script and the one in Using the Hive Spatial API are marked in bold; however, all of the steps are described in the preceding list.)

```
add jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/ojdbc8.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoutl.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdoapi.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector.jar
  /opt/oracle/oracle-spatial-graph/spatial/vector/jlib/sdohadoop-vector-hive.jar;

create temporary function ST_Polygon as
'oracle.spatial.hadoop.vector.hive.ST_Polygon';
create temporary function ST_Point as 'oracle.spatial.hadoop.vector.hive.ST_Point';
create temporary function ST_Contains as
'oracle.spatial.hadoop.vector.hive.function.ST_Contains';

set oracle.spatial.boundaries=-180,-90,180,90;
set oracle.spatial.index.includedExtraFields=followers_count,friends_count,location;

CREATE EXTERNAL TABLE IF NOT EXISTS sample_tweets_index (id STRING, geometry STRING,
followers_count STRING, friends_count STRING, location
STRING)
ROW FORMAT SERDE 'oracle.spatial.hadoop.vector.hive.json.GeoJsonSerDe'
STORED AS INPUTFORMAT
'oracle.spatial.hadoop.vector.mapred.input.SpatialIndexTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION '/user/oracle/twitter/index';

set oracle.spatial.spatialQueryWindow={"type": "Polygon","coordinates": [[[-107,
24], [-107, 31], [-103, 31], [-103, 24], [-107, 24]]]};

SELECT id, followers_count, friends_count, location FROM sample_tweets
WHERE ST_Contains(
  ST_Polygon('{"type": "Polygon","coordinates": [[[-106, 25], [-106, 30], [-104,
30], [-104, 25], [-106, 25]]]}', 8307)
  , ST_Point(geometry, 8307)
  , 0.5)
  and followers_count > 50;
```

2.9 Using the Oracle Big Data Spatial and Graph Vector Console

You can use the Oracle Big Data Spatial and Graph Vector Console to perform tasks related to spatial indexing and creating and showing thematic maps.

- Creating a Spatial Index Using the Console

- Exploring the Indexed Spatial Data
- Running a Categorization Job Using the Console
- Viewing the Categorization Results
- Saving Categorization Results to a File
- Creating and Deleting Templates
- Configuring Templates
- Running a Clustering Job Using the Console
- Viewing the Clustering Results
- Saving Clustering Results to a File
- Running a Binning Job Using the Console
- Viewing the Binning Results
- Saving Binning Results to a File
- Running a Job to Create an Index Using the Command Line
- Running a Job to Create a Categorization Result
- Running a Job to Create a Clustering Result
- Running a Job to Create a Binning Result
- Running a Job to Perform Spatial Filtering
- Running a Job to Get Location Suggestions
- Running a Job to Perform Spatial Filtering

2.9.1 Creating a Spatial Index Using the Console

To create a spatial index using the Oracle Big Data Spatial and Graph Vector Console, follow these steps.

1. Open the console:`http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`
2. Click **Create Index**.
3. Specify all the required details:
   a. Index name.
   b. Path of the file or files to index in HDFS. For example, `hdfs://<server_name_to_store_index>:8020/user/oracle/bdsg/tweets.json`.
   c. New index path: This is the job output path. For example: `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/index`.
   d. SRID of the geometries to be indexed. Example: 8307

   e. Tolerance of the geometries to be indexed. Example: 0.5
   f. Input Format class: The input format class. For example: `oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat`
   g. Record Info Provider class: The class that provides the spatial information. For example: `oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider`.

   Note:

   If the `InputFormat` class or the `RecordInfoProvider` class is not in the API, or in the hadoop API classes, then a jar with the user-defined classes must be provided. To be able to use this jar the user must add it in the `$JETTY_HOME/webapps/spatialviewer/WEB-INF/lib` directory and restart the server.

   ---

   h. Whether the MVSuggest service must be used or not. If the geometry has to be found from a location string, then use the MVSuggest service. In this case the provided `RecordInfoProvider` must implement the interface `oracle.spatial.hadoop.vector.LocalizableRecordInfoProvider`.
   i. MVSuggest service URL(Optional): If the geometry has to be found from a location string then use the MVSuggest service. If the service URL is localhost then each data node must have the MVSuggest application started and running. In this case, the new index contains the point geometry and the layer provided by MVSuggest for each record. If the geometry is a polygon then the geometry is a centroid of the polygon. For example: `http://localhost:8080`
   j. MVSuggest Templates (Optional): When using the MVSuggest service, the user can define the templates used to create the index.
   k. Outcome notification email sent to (Optional): Provide email Ids to receive the notifications when the job finished. Separate multiple email Ids by a semicolon. For example, `mymail@example.com`

4. Click **Create**.

   The submitted job is listed and you should wait to receive a mail notifying that the job completed successfully.

2.9.2 Exploring the Indexed Spatial Data

To explore indexed spatial data, follow these steps.

1. Open the console:`http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`
2. Click **Explore Data**.

For example, you can:

- Select the desired indexed data and click **Refresh Map** to see the data on the map.

- Change the background map style.
- Change the real data zoom level.
- Show data using a heat map.

2.9.3 Running a Categorization Job Using the Console

You can run a categorization job with or without the spatial index. Follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Categorization**, then **Run Job**.
3. Select either With Index or Without Index and provide the following details, as required:
    - With Index
        a. Index name
    - Without Index
        a. Path of the data: Provide the HDFS data path. For example, `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/tweets.json`.
        b. JAR with user classes (Optional): If the `InputFormat` class or the `RecordInfoProvider` class is not in the API, or in the hadoop API classes, then a jar with the user-defined classes must be provided. To be able to use this jar the user must add it in the `$JETTY_HOME/webapps/spatialviewer/WEB-INF/lib` directory and restart the server.
        c. Input Format class: The input format class. For example: `oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat`
        d. Record Info Provider class: The class that will provide the spatial information. For example: `oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider`.
        e. If the `MVSuggest` service has to be used or not. If the geometry must be found from a location string, then use the `MVSuggest` service. In this case the provided `RecordInfoProvider` has to implement the interface `oracle.spatial.hadoop.vector.LocalizableRecordInfoProvider`.
4. Templates: The templates to create the thematic maps.

Note:

If a template refers to point geometries (for example, cities), the result returned is empty for that template, if `MVSuggest` is not used. This is because the spatial operations return results only for polygons.

Tip:

When using the `MVSuggest` service the results will be more accurate if all the templates that could match the results are provided. For example, if the data can refer to any city, state, country, or continent in the world, then the better choice of templates to build results are World Continents, World Countries, World State Provinces, and World Cities. On the other hand, if the data is from the USA states and counties, then the suitable templates are USA States and USA Counties. If an index that was created using the `MVSuggest` service is selected, then select the top hierarchy for an optimal result. For example, if it was created using World Countries, World State Provinces, and World Cities, then use World Countries as the template.

5. Output path: The Hadoop job output path. For example: `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/catoutput`
6. Result name: The result name. If a result exists for a template with the same name, it is overwritten. For example, Tweets test.
7. Outcome notification email sent to (Optional): Provide email Ids to receive the notifications when the job finished. Separate multiple email Ids by a semicolon. For example, `mymail@abccorp.com`.

2.9.4 Viewing the Categorization Results

To view the categorization results, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Categorization**, then **View Results**.
3. Click any one of the Templates. For example, World Continents.

   The World Continents template is displayed.
4. Click any one of the Results displayed.

   Different continents appear with different patches of colors.
5. Click any continent from the map. For example, North America.

   The template changes to World Countries and the focus changes to North America with the results by country.

2.9.5 Saving Categorization Results to a File

You can save categorization results to a file (for example, the result file created with a job executed from the command line) on the local system for possible future uploading and use. The templates are located in the folder `$JETTY_HOME/webapps/spatialviewer/templates`. The templates are GeoJSON files with features and all the features have ids. For example, the first feature in the template *USA States* starts with: `{"type":"Feature","_id":"WYOMING",`...

The results must be JSON files with the following format: `{"id":"JSONFeatureId","result":result}`.

For example, if the template *USA States* is selected, then a valid result is a file containing: `{"id":"WYOMING","result":3232} {"id":"SOUTH DAKOTA","result":74968}`

1. Click **Categorization**, then **View Results**.
2. Select a Template .
3. Specify a Name.
4. Click Choose File to select the File location.
5. Click Save.

   The results can be located in the `$JETTY_HOME/webapps/spatialviewer/results` folder.

2.9.6 Creating and Deleting Templates

To create new templates do the following:

1. Add the template JSON file in the folder `$JETTY_HOME/webapps/spatialviewer/templates/`.
2. Add the template configuration file in the folder `$JETTY_HOME/webapps/spatialviewer/templates/_config_`.

To delete the template, delete the JSON and configuration files added in steps 1 and 2.

2.9.7 Configuring Templates

Each template has a configuration file. The template configuration files are located in the folder `$JETTY_HOME/webapps/spatialviewer/templates/_config_`. The name of the configuration file is the same as the template files suffixed with `config.json` instead of `.json`.For example, the configuration file name of the template file `usa_states.json` is `usa_states.config.json`. The configuration parameters are:

- name: Name of the template to be shown on the console. For example, `name: USA States`.
- display_attribute: When displaying a categorization result, a cursor move on the top of a feature displays this property and result of the feature. For example, `display_attribute: STATE NAME`.
- point_geometry: True, if the template contains point geometries and false, in case of polygons. For example, `point_geometry: false`.
- child_templates (optional): The templates that can have several possible child templates separated by a coma. For example, `child_templates: ["world_states_provinces, usa_states(properties.COUNTRY CODE:properties.PARENT_REGION)"].`

  If the child templates do not specify a linked field, it means that all the features inside the parent features are considered as child features. In this case, the `world_states_provinces` doesn't specify any fields. If the link between parent and child is specified, then the spatial relationship doesn't apply and the feature properties link are checked. In the above example, the relationship with the `usa_states` is found with the property `COUNTRY CODE` in the current template, and the property P`ARENT_REGION` in the template file `usa_states.json`.

- srid: The SRID of the template's geometries. For example, `srid: 8307`.
- back_polygon_template_file_name (optional): A template with polygon geometries to set as background when showing the defined template. For example, `back_polygon_template_file_name: usa_states`.
- vectorLayers: Configuration specific to the `MVSuggest` service. For example:

```
{
"vectorLayers": [
{
        "gnidColumns":["_GNID"],
        "boostValues":[2.0,1.0,1.0,2.0]
     }
     ]
 }
```

  Where:

  – gnidColumns is the name of the column(s) within the Json file that represents the Geoname ID. This value is used to support multiple languages with `MVSuggest`. (See references of that value in the file `templates/_geonames_/alternateNames.json`.) There is nodefault value for this property.

  – boostValues is an array of float numbers that represent how important a column is within the "properties" values for a given row. The higher the number, the more important that field is. A value of zero means the field will be ignored. When boostValues is not present, all fields receive a default value of 1.0, meaning they all are equally important properties. The `MVSuggest` service may return different results depending on those values. For a Json file with the following properties, the boost values might be as follows:

```
"properties":{"Name":"New York City","State":"NY","Country":"United
States","Country Code":"US","Population":8491079,"Time Zone":"UTC-5"}
"boostValues":[3.0,2.0,1.0,1.0,0.0,0.0]
```

2.9.8 Running a Clustering Job Using the Console

To run a clustering job, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Clustering**, then **Run Job**.
3. Provide the following details, as required:
   a. Path of the data: Provide the HDFS data path. For example, `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/tweets.json`.
   b. The SRID of the geometries. For example: 8307
   c. The tolerance of the geometries. For example: 0.5
   d. JAR with user classes (Optional): If the `InputFormat` class or the `RecordInfoProvider` class is not in the API, or in the hadoop API classes,

then a jar with the user-defined classes must be provided. To be able to use this jar the user must add it in the `$JETTY_HOME/webapps/spatialviewer/WEB-INF/lib` directory and restart the server.

e. Input Format class: The input format class. For example: `oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat`

f. Record Info Provider class: The class that will provide the spatial information. For example: `oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider`.

g. Output path: The Hadoop job output path. For example: `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/catoutput`

h. Result name: The result name. If a result exists for a template with the same name, it is overwritten. For example, Tweets test.

i. Outcome notification email sent to (Optional): Provide email Ids to receive the notifications when the job finished. Separate multiple email Ids by a semicolon. For example, `mymail@abccorp.com`.

2.9.9 Viewing the Clustering Results

To view the clustering results, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Clustering**, then **View Results**.
3. Click any one of the Results displayed.

2.9.10 Saving Clustering Results to a File

You can save clustering results to a file on your local system, for later uploading and use. To save the clustering results to a file, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Clustering**, then **View Results**.
3. Click the icon for saving the results.
4. Specify the SRID of the geometries. For example: 8307
5. Click **Choose File** and select the file location.
6. Click **Save**.

2.9.11 Running a Binning Job Using the Console

You can run a binning job with or without the spatial index. Follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.

2. Click **Binning**, then **Run Job**.
3. Select either With Index or Without Index and provide the following details, as required:
   - With Index
     a. Index name
   - Without Index
     a. Path of the data: Provide the HDFS data path. For example, `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/tweets.json`.
     b. The SRID of the geometries. For example: 8307
     c. The tolerance of the geometries. For example: 0.5
     d. JAR with user classes (Optional): If the `InputFormat` class or the `RecordInfoProvider` class is not in the API, or in the hadoop API classes, then a jar with the user-defined classes must be provided. To be able to use this jar the user must add it in the `$JETTY_HOME/webapps/spatialviewer/WEB-INF/lib` directory and restart the server.
     e. Input Format class: The input format class. For example: `oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat`
     f. Record Info Provider class: The class that will provide the spatial information. For example: `oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider`.
4. Binning grid minimum bounding rectangle (MBR). You can click the icon for seeing the MBR on the map.
5. Binning shape: hexagon (specify the hexagon width) or rectangle (specify the width and height).
6. Thematic attribute: If the job uses an index, double-click to see the possible values, which are those returned by the function `getExtraFields` of the `RecordInfoProvider` used when creating the index. If the job does not use an index, then the field can be one of the fields returned by the function `getExtraFields` of the specified `RecordInfoProvider` class. In any case, the `count` attribute is always available and specifies the number of records in the bin.
7. Output path: The Hadoop job output path. For example: `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/oracle/bdsg/binningOutput`
8. Result name: The result name. If a result exists for a template with the same name, it is overwritten. For example, Tweets test.
9. Outcome notification email sent to (Optional): Provide email Ids to receive the notifications when the job finished. Separate multiple email Ids by a semicolon. For example, `mymail@abccorp.com`.

2.9.12 Viewing the Binning Results

To view the binning results, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Binning**, then **View Results**.
3. Click any of the Results displayed.

2.9.13 Saving Binning Results to a File

You can save binning results to a file on your local system, for later uploading and use. To save the binning results to a file, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8045/spatialviewer/`.
2. Click **Binning**, then **View Results**.
3. Click the icon for saving the results.
4. Specify the SRID of the geometries. For example: 8307
5. Specify the thematic attribute, which must be a property of the features in the result. For example, the count attribute can be used to create results depending on the number of results per bin.
6. Click **Choose File** and select the file location.
7. Click **Save**.

2.9.14 Running a Job to Create an Index Using the Command Line

To create a spatial index, use a command in the following format:

```
hadoop jar <HADOOP_LIB_PATH>/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing [generic options]
input=<path|comma_separated_paths|path_pattern> output=<path>
inputFormat=<InputFormat_subclass> recordInfoProvider=<RecordInfoProvider_subclass>
[srid=<integer_value>] [geodetic=<true|false>] [tolerance=<double_value>]
[boundaries=<minX,minY,maxX,maxY>] [indexName=<index_name>]
[indexMetadataDir=<path>] [overwriteIndexMetadata=<true|false>] [  mvsLocation=<path|
URL> [mvsMatchLayers=<comma_separated_layers>][mvsMatchCountry=<country_name>]
[mvsSpatialResponse=<[NONE, FEATURE_GEOMETRY, FEATURE_CENTROID]>]
[mvsInterfaceType=<LOCAL, WEB>][mvsIsRepository=<true|false>][rebuildMVSIndex=<true|
false>][mvsPersistentLocation=<hdfs_path>][mvsOverwritePersistentLocation=<true|
false>] ]
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.SpatialIndexing` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.SpatialIndexing`.

Input/output arguments:

- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression.

- `inputFormat`: the `inputFormat` class implementation used to read the input data.
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class.
- `output`: the path where the spatial index will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.
- `boundaries` (optional, default=unbounded): the minimum and maximum values for each dimension, expressed as comma separated values in the form: minX,minY,maxX,maxY

Spatial index metadata arguments:

- `indexName` (optional, default=output folder name):The name of the index to be generated.
- `indexMetadataDir` (optional, default=hdfs://server:port/user/<current_user>/oracle_spatial/index_metadata/): the directory where the spatial index metadata will be stored.
- `overwriteIndexMetadata` (optional, default=false) boolean argument that indicates whether the index metadata can be overwritten if an index with the same name already exists.

`MVSuggest` arguments:

- `mvsLocation`: The path to the MVSuggest directory or repository for local standalone instances of MVSuggest or the service URL when working with a remote instance. This argument is required when working with MVSuggest.
- `mvsMatchLayers` (optional, default=all): comma separated list of layers. When provided, MVSuggest will only use these layers to perform the search.
- `mvsMatchCountry` (optional, default=none): a country name which MVSuggest will give higher priority when performing matches.
- `mvsSpatialResponse` (optional, default=CENTROID): the type of the spatial results contained in each returned match. It can be one of the following values: NONE, FEATURE_GEOMETRY, FEATURE_CENTROID.
- `mvsInterfaceType` (optional: default=LOCAL): the type of MVSuggest service used, it can be LOCAL or WEB.
- `mvsIsRepository` (optional: default=false) (LOCAL only): boolean value which specifies whether mvsLocation points to a whole MVS directory(false) or only to a repository(true). An MVS repository contains only JSON templates; it may or not contain a _config_ and _geonames_ folder.

- `mvsRebuildIndex` (optional, default=false)(LOCAL only):boolean value specifying whether the repository index should be rebuilt or not.
- `mvsPersistentLocation` (optional, default=none)(LOCAL only): an HDFS path where the MVSuggest directory will be saved.
- `mvsIsOverwritePersistentLocation` (optional, default=false): boolean argument that indicates whether an existing mvsPersistentLocation must be overwritten in case it already exists.

Example: Create a spatial index called `indexExample`. The index metadata will be stored in the HDFS directory `spatialMetadata`.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing input="/user/hdfs/
demo_vector/tweets/part*" output=/user/hdfs/demo_vector/tweets/spatial_index
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider
srid=8307 geodetic=true tolerance=0.5 indexName=indexExample
indexMetadataDir=indexMetadataDir overwriteIndexMetadata=true
```

Example: Create a spatial index using `MVSuggest` to assign a spatial location to records that do not contain geometries.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing input="/user/hdfs/
demo_vector/tweets/part*" output=/user/hdfs/demo_vector/tweets/spatial_index
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=mypackage.Simple LocationRecordInfoProvider srid=8307
geodetic=true tolerance=0.5 indexName=indexExample indexMetadataDir=indexMetadataDir
overwriteIndexMetadata=true mvsLocation=file:///local_folder/mvs_dir/oraclemaps_pub/
mvsRepository=true
```

2.9.15 Running a Job to Create a Categorization Result

To create a categorization result, use a command in one of the following formats.

With a Spatial Index

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Categorization [generic options]
( indexName=<indexName> [indexMetadataDir=<path>] )  | ( input=<path|
comma_separated_paths|path_pattern> isInputIndex=true [srid=<integer_value>]
[geodetic=<true|false>] [tolerance=<double_value>]
[boundaries=<min_x,min_y,max_x,max_y>]  ) output=<path>
hierarchyIndex=<hdfs_hierarchy_index_path> hierarchyInfo=<HierarchyInfo_subclass>
[hierarchyDataPaths=<level1_path,level2_path,,levelN_path>] spatialOperation=<[None,
IsInside, AnyInteract]>
```

Without a Spatial Index

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Categorization [generic options] input=<path|
comma_separated_paths|path_pattern> inputFormat=<InputFormat_subclass>
recordInfoProvider=<RecordInfoProvider_subclass> [srid=<integer_value>]
[geodetic=<true|false>] [tolerance=<double_value>]
[boundaries=<min_x,min_y,max_x,max_y>]  output=<path>
hierarchyIndex=<hdfs_hierarchy_index_path> hierarchyInfo=<HierarchyInfo_subclass>
hierarchyDataPaths=<level1_path,level2_path,,levelN_path>] spatialOperation=<[None,
IsInside, AnyInteract]>
```

Using MVSuggest

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Categorization [generic options]
(indexName=<indexName> [indexMetadataDir=<path>])  |
(
(input=<path|comma_separated_paths|path_pattern> isInputIndex=true)  | (input=<path|
comma_separated_paths|path_pattern> inputFormat=<InputFormat_subclass>
recordInfoProvider=<LocalizableRecordInfoProvider_subclass>)
[srid=<integer_value>] [geodetic=<true|false>] [tolerance=<double_value>]
[boundaries=<min_x,min_y,max_x,max_y>]
) output=<path>
mvsLocation=<path|URL> [mvsMatchLayers=<comma_separated_layers>]
[mvsMatchCountry=<country_name>] [mvsSpatialResponse=<[NONE, FEATURE_GEOMETRY,
FEATURE_CENTROID]>] [mvsInterfaceType=<[UNDEFINED, LOCAL, WEB]>]
[mvsIsRepository=<true|false>] [mvsRebuildIndex=<true|false>]
[mvsPersistentLocation=<hdfs_path>] [mvsOverwritePersistentLocation=<true|false>]
[mvsMaxRequestRecords=<integer_number>]  hierarchyIndex=<hdfs_hierarchy_index_path>
hierarchyInfo=<HierarchyInfo_subclass>
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.Categorization` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.Categorization`.

Input/output arguments:

- `indexName`: the name of an existing spatial index. The index information will be looked at the path given by indexMetadataDir. When used, the argument `input` is ignored.
- `indexMetadataDir` (optional, default=hdfs://server:port/user/<current_user>/oracle_spatial/index_metadata/): the directory where the spatial index metadata is located
- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression. (Ignored if `indexName` is specified.)
- `inputFormat`: the `inputFormat` class implementation used to read the input data. (Ignored if `indexName` is specified.)
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class. (Ignored if `indexName` is specified.)
- `output`: the path where the spatial index will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.
- `boundaries` (optional, default=unbounded): the minimum and maximum values for each dimension, expressed as comma separated values in the form: minX,minY,maxX,maxY

- `spatialOperation`: the spatial operation to perform between the input data set and the hierarchical data set. Allowed values are `IsInside` and `AnyInteract`.

Hierarchical data set arguments:

- `hierarchyIndex`: the HDFS path of an existing hierarchical index or where it can be stored if it needs to be generated.
- `hierarchyInfo`: the fully qualified name of a `HierarchyInfo` subclass which is used to describe the hierarchical data.
- `hierarchyDataPaths` (optional, default=none): a comma separated list of paths of the hierarchy data. The paths should be sorted in ascending way by hierarchy level. If a hierarchy index path does not exist for the given hierarchy data, this argument is required.

`MVSuggest` arguments:

- `mvsLocation`: The path to the MVSuggest directory or repository for local standalone instances of MVSuggest or the service URL when working with a remote instance. This argument is required when working with MVSuggest.
- `mvsMatchLayers` (optional, default=all): comma separated list of layers. When provided, MVSuggest will only use these layers to perform the search.
- `mvsMatchCountry` (optional, default=none): a country name which MVSuggest will give higher priority when performing matches.
- `mvsSpatialResponse` (optional, default=CENTROID): the type of the spatial results contained in each returned match. It can be one of the following values: NONE, FEATURE_GEOMETRY, FEATURE_CENTROID.
- `mvsInterfaceType` (optional: default=LOCAL): the type of MVSuggest service used, it can be LOCAL or WEB.
- `mvsIsRepository` (optional: default=false) (LOCAL only): Boolean value that specifies whether `mvsLocation` points to a whole MVS directory(false) or only to a repository(true). An MVS repository contains only JSON templates; it may or not contain a `_config_` and `_geonames_` folder.
- `mvsRebuildIndex` (optional, default=false)(LOCAL only):boolean value specifying whether the repository index should be rebuilt or not.
- `mvsPersistentLocation` (optional, default=none)(LOCAL only): an HDFS path where the MVSuggest directory will be saved.
- `mvsIsOverwritePersistentLocation` (optional, default=false): boolean argument that indicates whether an existing `mvsPersistentLocation` must be overwritten in case it already exists.

Example: Run a Categorization job to create a summary containing the records counts by continent, country, and state/provinces. The input is an existing spatial index called `indexExample`. The hierarchical data will be indexed and stored in HDFS at the path `hierarchyIndex`.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Categorization indexName= indexExample
output=/user/hdfs/demo_vector/tweets/hier_count_spatial
hierarchyInfo=vectoranalysis.categorization.WorldAdminHierarchyInfo
hierarchyIndex=hierarchyIndex  hierarchyDataPaths=file:///templates/
```

```
world_continents.json,file:///templates/world_countries.json,file:///templates/
world_states_provinces.json spatialOperation=IsInside
```

Example: Run a Categorization job to create a summary of tweet counts per continent, country, states/provinces, and cities using `MVSuggest`.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Categorization input="/user/hdfs/demo_vector/
tweets/part*" inputFormat=<InputFormat_subclass>
recordInfoProvider=<LocalizableRecordInfoProvider_subclass>  output=/user/hdfs/
demo_vector/tweets/hier_count_mvs
hierarchyInfo=vectoranalysis.categorization.WorldAdminHierarchyInfo
hierarchyIndex=hierarchyIndex mvsLocation=file:///mvs_dir
mvsMatchLayers=world_continents,world_countries,world_states_provinces
spatialOperation=IsInside
```

2.9.16 Running a Job to Create a Clustering Result

To create a clustering result, use a command in the following format:

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.KMeansClustering [generic options]
input=<path|comma_separated_paths|path_pattern> inputFormat=<InputFormat_subclass>
recordInfoProvider=<RecordInfoProvider_subclass> output=<path>
[srid=<integer_value>] [geodetic=<true|false>] [tolerance=<double_value>]
[boundaries=<min_x,min_y,max_x,max_y>]  k=<number_of_clusters>
[clustersPoints=<comma_separated_points_ordinates>] [deleteClusterFiles=<true|
false>] [maxIterations=<integer_value>] [critFunClass=<CriterionFunction_subclass>]
[shapeGenClass=<ClusterShapeGenerator_subclass>] [maxMemberDistance=<double_value>]
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.KMeansClustering` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.KMeansClustering`.

Input/output arguments:

- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression.
- `inputFormat`: the `inputFormat` class implementation used to read the input data.
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class.
- `output`: the path where the spatial index will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): Boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.
- `boundaries` (optional, default=unbounded): the minimum and maximum values for each dimension, expressed as comma separated values in the form: minX,minY,maxX,maxY

- `spatialOperation`: the spatial operation to perform between the input data set and the hierarchical data set. Allowed values are `IsInside` and `AnyInteract`.

Clustering arguments:

- `k`: the number of clusters to be found.
- `clusterPoints` (optional, default=none): the initial cluster centers as a comma-separated list of point ordinates in the form: p1_x,p1_y,p2_x,p2_y,…,pk_x,pk_y
- `deleteClusterFiles` (optional, default=true): Boolean arguments that specifies whether the intermediate cluster files generated between iterations should be deleted or not
- `maxIterations` (optional, default=calculated based on the number k): the maximum number of iterations allowed before the job completes.
- `critFunClass` (optional, default=oracle.spatial.hadoop.vector.cluster.kmeans. SquaredErrorCriterionFunction) a fully qualified name of a `CriterionFunction` subclass.
- `shapeGenClass` (optional, default= oracle.spatial.hadoop.vector.cluster.kmeans. ConvexHullClusterShapeGenerator) a fully qualified name of a `ClusterShapeGenerator` subclass used to generate the geometry of the clusters.
- `maxMemberDistance` (optional, default=undefined): a double value that specifies the maximum distance between a cluster center and a cluster member.

Example: Run a Clustering job to generate 5 clusters. The generated clusters geometries will be the convex hull of all .

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.KMeansClustering input="/user/hdfs/
demo_vector/tweets/part*" output=/user/hdfs/demo_vector/tweets/result
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider
srid=8307 geodetic=true tolerance=0.5 k=5
shapeGenClass=oracle.spatial.hadoop.vector.cluster.kmeans.ConvexHullClusterShapeGener
ator
```

2.9.17 Running a Job to Create a Binning Result

To create a binning result, use a command in the following format:

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Binning [generic options]
(indexName=<INDEX_NAME> [indexMetadataDir=<INDEX_METADATA_DIRECTORY>]) |
(input=<DATA_PATH> inputFormat=<INPUT_FORMAT_CLASS>
recordInfoProvider=<RECORD_INFO_PROVIDER_CLASS> [srid=<SRID>] [geodetic=<GEODETIC>]
[tolerance=<TOLERANCE>]) output=<RESULT_PATH> cellSize=<CELL_SIZE>
gridMbr=<GRID_MBR> [cellShape=<CELL_SHAPE>] [aggrFields=<EXTRA_FIELDS>]
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.Binning` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.Binning`.

Input/output arguments:

- `indexName`: the name of an existing spatial index. The index information will be looked at the path given by indexMetadataDir. When used, the argument `input` is ignored.

- `indexMetadataDir` (optional, default=hdfs://server:port/user/<current_user>/oracle_spatial/index_metadata/): the directory where the spatial index metadata is located
- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression.
- `inputFormat`: the `inputFormat` class implementation used to read the input data.
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class.
- `output`: the path where the spatial index will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): Boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.

Binning arguments:

- `cellSize`: the size of the cells in the format: width,height
- `gridMbr` : the minimum and maximum dimension values for the grid in the form: minX,minY,maxX,maxY
- `cellShape` (optional, default=RECTANGLE): the shape of the cells. It can be RECTANGLE or HEXAGON
- `aggrFields` (optional, default=none): a comma-separated list of field names that will be aggregated.

Example: Run a spatial binning job to generate a grid of hexagonal cells and aggregate the value of the field SALES..

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Binning indexName=indexExample
indexMetadataDir=indexMetadataDir output=/user/hdfs/demo_vector/result
cellShape=HEXAGON cellSize=5 gridMbr=-175,-85,175,85 aggrFields=SALES
```

2.9.18 Running a Job to Perform Spatial Filtering

To perform spatial filtering, use a command in the following format:

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialFilter [generic options]
( indexName=<indexName> [indexMetadataDir=<path>] )  |
(
(input=<path|comma_separated_paths|path_pattern> isInputIndex=true)  | (input=<path|
comma_separated_paths|path_pattern> inputFormat=<InputFormat_subclass>
recordInfoProvider=<RecordInfoProvider_subclass>)
[srid=<integer_value>] [geodetic=<true|false>] [tolerance=<double_value>]
[boundaries=<min_x,min_y,max_x,max_y>]
) output=<path>  spatialOperation=<[IsInside, AnyInteract]> queryWindow=<json-
geometry>
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.SpatialFilter` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.SpatialFilter`.

Input/output arguments:

- `indexName`: the name of an existing spatial index. The index information will be looked at the path given by indexMetadataDir. When used, the argument `input` is ignored.
- `indexMetadataDir` (optional, default=hdfs://server:port/user/<current_user>/oracle_spatial/index_metadata/): the directory where the spatial index metadata is located
- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression.
- `inputFormat`: the `inputFormat` class implementation used to read the input data.
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class.
- `output`: the path where the spatial index will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): Boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.

Binning arguments:

- `cellSize`: the size of the cells in the format: width,height
- `gridMbr` : the minimum and maximum dimension values for the grid in the form: minX,minY,maxX,maxY
- `cellShape` (optional, default=RECTANGLE): the shape of the cells. It can be RECTANGLE or HEXAGON
- `aggrFields` (optional, default=none): a comma-separated list of field names that will be aggregated.
- `boundaries` (optional, default=unbounded): the minimum and maximum values for each dimension, expressed as comma separated values in the form: minx,minY,maxX,maxY
- `spatialOperation`: the operation to be applied between the queryWindow and the geometries from the input data set
- `queryWindow`: the geometry used to filter the input dataset.

Example: Perform a spatial filtering operation.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialFilter indexName=indexExample
indexMetadataDir=indexMetadataDir output=/user/hdfs/demo_vector/result
spatialOperation=IsInside queryWindow='{"type":"Polygon", "coordinates":[[-106, 25,
-106, 30, -104, 30, -104, 25, -106, 25]]}'
```

2.9.19 Running a Job to Get Location Suggestions

To create a job to get location suggestions, use a command in the following format.

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SuggestService [generic options] input=<path|
comma_separated_paths|path_pattern> inputFormat=<InputFormat_subclass>
recordInfoProvider=<RecordInfoProvider_subclass> output=<path> mvsLocation=<path|
URL> [mvsMatchLayers=<comma_separated_layers>] [mvsMatchCountry=<country_name>]
[mvsSpatialResponse=<[NONE, FEATURE_GEOMETRY, FEATURE_CENTROID]>]
[mvsInterfaceType=<[UNDEFINED, LOCAL, WEB]>] [mvsIsRepository=<true|false>]
[mvsRebuildIndex=<true|false>] [mvsPersistentLocation=<hdfs_path>]
[mvsOverwritePersistentLocation=<true|false>] [mvsMaxRequestRecords=<integer_number>]
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.SuggestService` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.SuggestService`.

Input/output arguments:

- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression. (Ignored if `indexName` is specified.)
- `inputFormat`: the `inputFormat` class implementation used to read the input data. (Ignored if `indexName` is specified.)
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class. (Ignored if `indexName` is specified.)
- `output`: the path where the spatial index will be stored

`MVSuggest` arguments:

- `mvsLocation`: The path to the MVSuggest directory or repository for local standalone instances of MVSuggest or the service URL when working with a remote instance. This argument is required when working with MVSuggest.
- `mvsMatchLayers` (optional, default=all): comma separated list of layers. When provided, MVSuggest will only use these layers to perform the search.
- `mvsMatchCountry` (optional, default=none): a country name which MVSuggest will give higher priority when performing matches.
- `mvsSpatialResponse` (optional, default=CENTROID): the type of the spatial results contained in each returned match. It can be one of the following values: NONE, FEATURE_GEOMETRY, FEATURE_CENTROID.
- `mvsInterfaceType` (optional: default=LOCAL): the type of MVSuggest service used, it can be LOCAL or WEB.
- `mvsIsRepository` (optional: default=false) (LOCAL only): Boolean value that specifies whether `mvsLocation` points to a whole MVS directory(false) or only to

a repository(true). An MVS repository contains only JSON templates; it may or not contain a `_config_` and `_geonames_` folder.

- `mvsRebuildIndex` (optional, default=false)(LOCAL only):boolean value specifying whether the repository index should be rebuilt or not.
- `mvsPersistentLocation` (optional, default=none)(LOCAL only): an HDFS path where the MVSuggest directory will be saved.
- `mvsIsOverwritePersistentLocation` (optional, default=false): boolean argument that indicates whether an existing `mvsPersistentLocation` must be overwritten in case it already exists.

Example: Get suggestions based on location texts from the input data set..

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SuggestService input="/user/hdfs/demo_vector/
tweets/part*" inputFormat=<InputFormat_subclass>
recordInfoProvider=<LocalizableRecordInfoProvider_subclass> output=/user/hdfs/
demo_vector/tweets/suggest_res mvsLocation=file:///mvs_dir
mvsMatchLayers=world_continents,world_countries,world_states_provinces
```

2.9.20 Running a Job to Perform a Spatial Join

To perform a spatial join operation on two data sets, use a command in the following format.

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job. SpatialJoin [generic options]
inputList={
 {
  ( indexName=<dataset1_spatial_index_name>
indexMetadataDir=<dataset1_spatial_index_metadata_dir_path> )
  |
  ( input=<dataset1_path|comma_separated_paths|path_pattern>
inputFormat=<dataset1_InputFormat_subclass>
recordInfoProvider=<dataset1_RecordInfoProvider_subclass> )
  [boundaries=<min_x,min_y,max_x,max_y>]
 }
 {
  (indexName=<dataset2_spatial_index_name>
indexMetadataDir=<dataset2_spatial_index_metadata_dir_path>
  )
  |
  ( input=<dataset2_path|comma_separated_paths|path_pattern>
inputFormat=<dataset2_InputFormat_subclass>
recordInfoProvider=<dataset2_RecordInfoProvider_subclass>
  )
  [boundaries=<min_x,min_y,max_x,max_y>]
 }
} output=<path>[srid=<integer_value>] [geodetic=<true|false>]
[tolerance=<double_value>] boundaries=<min_x,min_y,max_x,max_y>
spatialOperation=<AnyInteract|IsInside|WithinDistance> [distance=<double_value>]
[samplingRatio=<decimal_value_between_0_and_1> | partitioningResult=<path>]
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.SpatialJoin` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.SpatialJoin`.

`InputList`: A list of two input data sets. The list is enclosed by curly braces ({}). Each list element is an input data set, which is enclosed by curly braces. An input data set can contain the following information, depending on whether the data set is specified as a spatial index.

If specified as a spatial index:

- `indexName`: the name of an existing spatial index.
- `indexMetadataDir` : the directory where the spatial index metadata is located

If not specified as a spatial index:

- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression. (Ignored if `indexName` is specified.)
- `inputFormat`: the `inputFormat` class implementation used to read the input data. (Ignored if `indexName` is specified.)
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class. (Ignored if `indexName` is specified.)

`output`: the path where the results will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.
- `boundaries` (optional, default=unbounded): the minimum and maximum values for each dimension, expressed as comma separated values in the form: minX,minY,maxX,maxY
- `spatialOperation`: the spatial operation to perform between the input data set and the hierarchical data set. Allowed values are `IsInside` and `AnyInteract`.
- `distance`: distance used for `WithinDistance` operations.

Partitioning arguments:

- `samplingRatio` (optional, default=0.1): ratio used to sample the data sets when partitioning needs to be performed
- `partitioningResult` (optional, default=none): Path to a previously generated partitioning result file

Example: Perform a spatial join on two data sets.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialJoin inputList="{{input=/user/hdfs/
demo_vector/world_countries.json
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider}
{input=file="/user/hdfs/demo_vector/tweets/part*”
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider}}"
```

```
output=/user/hdfs/demo_vector/spatial_join srid=8307 spatialOperation=AnyInteract
boundaries=-180,-90,180,90
```

2.9.21 Running a Job to Perform Partitioning

To perform a spatial partitioning, use a command in the following format.

```
hadoop jar <HADOOP_LIB_PATH >/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job. SpatialJoin [generic options]
inputList={
 {
  ( indexName=<dataset1_spatial_index_name>
indexMetadataDir=<dataset1_spatial_index_metadata_dir_path> )
  |
  ( input=<dataset1_path|comma_separated_paths|path_pattern>
inputFormat=<dataset1_InputFormat_subclass>
recordInfoProvider=<dataset1_RecordInfoProvider_subclass> )
  [boundaries=<min_x,min_y,max_x,max_y>]
 }
[
 {
  (indexName=<dataset2_spatial_index_name>
indexMetadataDir=<dataset2_spatial_index_metadata_dir_path>
  )
  |
  ( input=<dataset2_path|comma_separated_paths|path_pattern>
inputFormat=<dataset2_InputFormat_subclass>
recordInfoProvider=<dataset2_RecordInfoProvider_subclass>
  )
  [boundaries=<min_x,min_y,max_x,max_y>]
 }
  ……
 {
  (indexName=<datasetN_spatial_index_name>
indexMetadataDir=<datasetN_spatial_index_metadata_dir_path>
  )
  |
  ( input=<datasetN_path|comma_separated_paths|path_pattern>
inputFormat=<datasetN_InputFormat_subclass>
recordInfoProvider=<datasetN_RecordInfoProvider_subclass>
  )
  [boundaries=<min_x,min_y,max_x,max_y>]
 }

}
] output=<path>[srid=<integer_value>] [geodetic=<true|false>]
[tolerance=<double_value>] boundaries=<min_x,min_y,max_x,max_y>
[samplingRatio=<decimal_value_between_0_and_1>]
```

To use the new Hadoop API format, replace `oracle.spatial.hadoop.vector.`*`mapred`*`.job.Partitioning` with `oracle.spatial.hadoop.vector.`**`mapreduce`**`.job.Partitioning`.

`InputList`: A list of two input data sets. The list is enclosed by curly braces ({}). Each list element is an input data set, which is enclosed by curly braces. An input data set can contain the following information, depending on whether the data set is specified as a spatial index.

If specified as a spatial index:

- `indexName`: the name of an existing spatial index.

- `indexMetadataDir` : the directory where the spatial index metadata is located

If not specified as a spatial index:

- `input` : the location of the input data. It can be expressed as a path, a comma separated list of paths, or a regular expression. (Ignored if `indexName` is specified.)
- `inputFormat`: the `inputFormat` class implementation used to read the input data. (Ignored if `indexName` is specified.)
- `recordInfoProvider`: the `recordInfoProvider` implementation used to extract information from the records read by the `InputFormat` class. (Ignored if `indexName` is specified.)

`output`: the path where the results will be stored

Spatial arguments:

- `srid` (optional, default=0): the spatial reference system (coordinate system) ID of the spatial data.
- `geodetic` (optional, default depends on the srid): boolean value that indicates whether the geometries are geodetic or not.
- `tolerance` (optional, default=0.0): double value that represents the tolerance used when performing spatial operations.
- `boundaries` (optional, default=unbounded): the minimum and maximum values for each dimension, expressed as comma separated values in the form: minX,minY,maxX,maxY

Partitioning arguments:

- `samplingRatio` (optional, default=0.1): ratio used to sample the data sets when partitioning needs to be performed

Example: Partition two data sets.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Partitioning inputList="{{input=/user/hdfs/
demo_vector/world_countries.json
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider}
{input=file="/user/hdfs/demo_vector/tweets/part*”
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider}}"
output=/user/hdfs/demo_vector/partitioning srid=8307 boundaries=-180,-90,180,90
```

2.10 Using Oracle Big Data Spatial and Graph Image Server Console

You can use the Oracle Big Data Spatial and Graph Image Server Console to tasks, such as Loading Images to HDFS Hadoop Cluster to Create a Mosaic.

2.10.1 Loading Images to HDFS Hadoop Cluster to Create a Mosaic

Follow the instructions to create a mosaic:

1. Open `http://<oracle_big_data_image_server_console>:8080/spatialviewer/.`
2. Type the username and password.

3. Click the Configuration tab and review the Hadoop configuration section.

   By default the application is configured to work with the Hadoop cluster and no additional configuration is required.

   Note:

   Only an admin user can make changes to this section.

   ---

4. Click the Hadoop Loader tab and review the displayed instructions or alerts.
5. Follow the instructions and update the runtime configuration, if necessary.
6. Click the Folder icon.

   The File System dialog displays the list of image files and folders.

7. Select the folders or files as required and click Ok.

   The complete path to the image file is displayed.

8. Click Load Images.

   Wait for the images to be loaded successfully. A message is displayed.

9. Proceed to create a mosaic, if there are no errors displayed.

3

Configuring Property Graph Support

This chapter explains how to configure the support for property graphs in a Big Data environment.

It assumes that you have already performed the installation on a Big Data Appliance (see Installing Oracle Big Data Spatial and Graph on an Oracle Big Data Appliance), an Apache Hadoop system (see Installing Property Graph Support on a CDH Cluster or Other Hardware), or an Oracle NoSQL Database.

- Tuning the Software Configuration

3.1 Tuning the Software Configuration

You might be able to improve the performance of property graph support by altering the database and Java configuration settings. The suggestions provided are guidelines, which you should follow only after carefully and thoroughly evaluating your system.

- Tuning Apache HBase for Use with Property Graphs
- Tuning Oracle NoSQL Database for Use with Property Graphs

3.1.1 Tuning Apache HBase for Use with Property Graphs

Modifications to the default Apache HBase and Java Virtual Machine configurations can improve performance.

- Modifying the Apache HBase Configuration
- Modifying the Java Memory Settings

3.1.1.1 Modifying the Apache HBase Configuration

To modify the Apache HBase configuration, follow the steps in this section for your CDH release. (Note that specific steps might change from one CDH release to the next.)

For CDH 5.2.x, CDH 5.3.x, and CDH 5.4.x:

1. Log in to Cloudera Manager as the `admin` user.
2. On the Home page, click **HBase** in the list of services on the left.
3. On the HBase page, click the **Configuration** tab.
4. In the Category panel on the left, expand Service-Wide, and then choose **Advanced**.
5. Edit the value of HBase Service Advanced Configuration Snippet (Safety Valve) for `hbase-site.xml` as follows:

```
<property>
  <name>hbase.regionserver.handler.count</name>
  <value>32</value>
</property>
<property>
  <name>hbase.hregion.max.filesize</name>
  <value>1610612736</value>
</property>
<property>
  <name>hbase.hregion.memstore.block.multiplier</name>
  <value>4</value>
</property>
<property>
   <name>hbase.hregion.memstore.flush.size</name>
   <value>134217728</value>
</property>
<property>
   <name>hbase.hstore.blockingStoreFiles</name>
   <value>200</value></property>
<property>
  <name>hbase.hstore.flusher.count</name>
  <value>1</value>
</property>
```

If the property already exists, then replace the value as required. Otherwise, add the XML property description.

6. Click **Save Changes**.
7. Expand the Actions menu, and then choose **Restart** or **Rolling Restart**, whichever option better suits your situation.

For CDH 5.4.x:

1. Log in to Cloudera Manager as the `admin` user.
2. On the Home page, click **HBase** in the list of services on the left.
3. On the HBase page, click the **Configuration** tab.
4. Expand **SCOPE**.
5. Click **HBase (Service-wide)**, scroll to the bottom of the page, and select **Display All Entries** (*not* Display 25 Entries).
6. On this page, locate **HBase Service Advanced Configuration Snippet (Safety Valve)** for `hbase-site.xml`, and enter the following value for the `<property>` element:

```
<property>
  <name>hbase.regionserver.handler.count</name>
  <value>32</value>
</property>
<property>
  <name>hbase.hregion.max.filesize</name>
  <value>1610612736</value>
</property>
<property>
  <name>hbase.hregion.memstore.block.multiplier</name>
  <value>4</value>
</property>
<property>
```

```
   <name>hbase.hregion.memstore.flush.size</name>
   <value>134217728</value>
</property>
<property>
   <name>hbase.hstore.blockingStoreFiles</name>
   <value>200</value></property>
<property>
  <name>hbase.hstore.flusher.count</name>
  <value>1</value>
</property>
```

If the property already exists, then replace the value as required. Otherwise, add the XML property description.

7. Click **Save Changes**.
8. Expand the Actions menu, and then choose **Restart** or **Rolling Restart**, whichever option better suits your situation.

3.1.1.2 Modifying the Java Memory Settings

To modify the Java memory settings, follow the steps in this section for your CDH release. (Note that specific steps might change from one CDH release to the next.)

For CDH 5.2.x and CDH 5.3.x:

1. Log in to Cloudera Manager as the `admin` user.
2. On the Home page, click **HBase** in the list of services on the left.
3. On the HBase page, click the **Configuration** tab.
4. For **RegionServer Group** (default and others), click **Advanced**, and use the following for **Java Configuration Options** for HBase RegionServer:

   ```
   -Xmn256m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -
   XX:CMSInitiatingOccupancyFraction=70 -XX:+UseCMSInitiatingOccupancyOnly
   ```

5. Click **Resource Management**, and enter an appropriate value (for example, `18G`) for **Java Heap Size** of HBase RegionServer.
6. Click **Save Changes**.
7. Expand the Actions menu, and then choose **Restart** or **Rolling Restart**, whichever option better suits your situation.

For CDH 5.4.x:

1. Log in to Cloudera Manager as the `admin` user.
2. On the Home page, click **HBase** in the list of services on the left.
3. On the HBase page, click the **Configuration** tab.
4. Expand **SCOPE**.
5. Click **RegionServer**, scroll to the bottom of the page, and select **Display All Entries** (*not* Display 25 Entries).
6. On this page, for **Java Configuration Options for HBase RegionServer,** enter the following value:

```
-Xmn256m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -
XX:CMSInitiatingOccupancyFraction=70 -XX:+UseCMSInitiatingOccupancyOnly
```

7. For **Java Heap Size of HBase RegionServer in Bytes**, enter an appropriate value (for example, `18G`).
8. Click **Save Changes**.
9. Expand the Actions menu, and then choose **Restart** or **Rolling Restart**, whichever option better suits your situation.

See Also:

For detailed information about Java garbage collection, see:

http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/

For descriptions of all settings, see the Java *Tools Reference*:

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html

---

3.1.2 Tuning Oracle NoSQL Database for Use with Property Graphs

To obtain the best performance from Oracle NoSQL Database:

- Ensure that the replication groups (shards) are balanced.
- Adjust the user process resource limit setting (`ulimit`). For example:

  ```
  ulimit -u 131072
  ```

- Set the heap size of the Java Virtual Machines (JVMs) on the replication nodes to enable the B-tree indexes to fit in memory.

  To set the heap size, use either the `-memory_mb` option of the `makebookconfig` command or the `memory_mb` parameter for the storage node.

  Oracle NoSQL Database uses 85% of `memory_mb` as the heap size for processes running on the storage node. If the storage node hosts multiple replication nodes, then the heap is divided equally among them. Each replication node uses a cache that is 70% of the heap.

  For example, if you set `memory_mb` to 3000 MB on a storage node that hosts two replication nodes, then each replication node has the following:

  - 1275 MB heap, calculated as (3000 MB * .85)/2
  - 892 MB cache, calculated as 1275 MB * .70

See Also:

Oracle NoSQL Database FAQ at

http://www.oracle.com/technetwork/products/nosqldb/learnmore/nosqldb-faq-518364.html#HowdoesNoSQLDBbudgetmemory

---

4

Using Property Graphs in a Big Data Environment

This chapter provides conceptual and usage information about creating, storing, and working with property graph data in a Big Data environment.

4.1 About Property Graphs

Property graphs allow an easy association of properties (key-value pairs) with graph vertices and edges, and they enable analytical operations based on relationships across a massive set of data.

4.1.1 What Are Property Graphs?

A property graph consists of a set of objects or **vertices**, and a set of arrows or **edges** connecting the objects. Vertices and edges can have multiple properties, which are represented as key-value pairs.

Each vertex has a unique identifier and can have:

- A set of outgoing edges

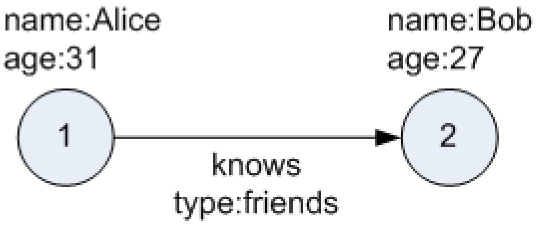
- A set of incoming edges
- A collection of properties

Each edge has a unique identifier and can have:

- An outgoing vertex
- An incoming vertex
- A text label that describes the relationship between the two vertices
- A collection of properties

Figure 4-1 illustrates a very simple property graph with two vertices and one edge. The two vertices have identifiers 1 and 2. Both vertices have properties `name` and `age`. The edge is from the outgoing vertex 1 to the incoming vertex 2. The edge has a text label `knows` and a property `type` identifying the type of relationship between vertices 1 and 2.

Figure 4-1 Simple Property Graph Example



Standards are not available for Big Data Spatial and Graph property graph data model, but it is similar to the W3C standards-based Resource Description Framework (RDF) graph data model. The property graph data model is simpler and much less precise than RDF. These differences make it a good candidate for use cases such as these:

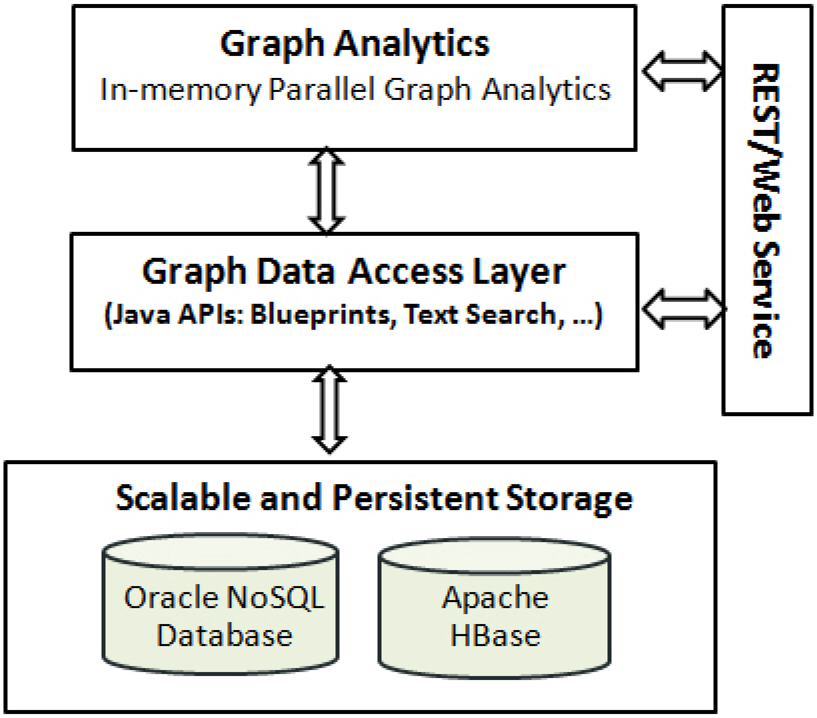
- Identifying influencers in a social network
- Predicting trends and customer behavior
- Discovering relationships based on pattern matching
- Identifying clusters to customize campaigns

4.1.2 What Is Big Data Support for Property Graphs?

Property graphs are supported for Big Data in Hadoop and in Oracle NoSQL Database. This support consists of a data access layer and an analytics layer. A choice of databases in Hadoop provides scalable and persistent storage management.

Figure 4-2 provides an overview of the Oracle property graph architecture.

Figure 4-2 Oracle Property Graph Architecture



4.1.2.1 In-Memory Analyst

The in-memory analyst layer enables you to analyze property graphs using parallel in-memory execution. It provides over 35 analytic functions, including path calculation, ranking, community detection, and recommendations.

4.1.2.2 Data Access Layer

The data access layer provides a set of Java APIs that you can use to create and drop property graphs, add and remove vertices and edges, search for vertices and edges using key-value pairs, create text indexes, and perform other manipulations. The Java APIs include an implementation of TinkerPop Blueprints graph interfaces for the property graph data model. The APIs also integrate with the Apache Lucene and Apache SolrCloud, which are widely-adopted open-source text indexing and search engines.

4.1.2.3 Storage Management

You can store your property graphs in either Oracle NoSQL Database or Apache HBase. Both databases are mature and scalable, and support efficient navigation, querying, and analytics. Both use tables to model the vertices and edges of property graphs.

4.1.2.4 RESTful Web Services

You can also use RESTful web services to access the graph data and perform graph operations. For example, you can use the Linux `curl` command to obtain vertices and edges, and to add and remove graph elements.

4.2 About Property Graph Data Formats

The following graph formats are supported:

- GraphML Data Format
- GraphSON Data Format
- GML Data Format
- Oracle Flat File Format

4.2.1 GraphML Data Format

The GraphML file format uses XML to describe graphs. Example 4-1 shows a GraphML description of the property graph shown in Figure 4-1.

See Also:

"The GraphML File Format" at

`http://graphml.graphdrawing.org/`

Example 4-1 GraphML Description of a Simple Property Graph

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
    <key id="name" for="node" attr.name="name" attr.type="string"/>
    <key id="age" for="node" attr.name="age" attr.type="int"/>
    <key id="type" for="edge" attr.name="type" attr.type="string"/>
    <graph id="PG" edgedefault="directed">
        <node id="1">
            <data key="name">Alice</data>
            <data key="age">31</data>
        </node>
        <node id="2">
            <data key="name">Bob</data>
            <data key="age">27</data>
        </node>
        <edge id="3" source="1" target="2" label="knows">
            <data key="type">friends</data>
        </edge>
    </graph>
</graphml>
```

4.2.2 GraphSON Data Format

The GraphSON file format is based on JavaScript Object Notation (JSON) for describing graphs. Example 4-2 shows a GraphSON description of the property graph shown in Figure 4-1.

See Also:

"GraphSON Reader and Writer Library" at

`https://github.com/tinkerpop/blueprints/wiki/GraphSON-Reader-and-Writer-Library`

Example 4-2 GraphSON Description of a Simple Property Graph

```
{
    "graph": {
        "mode":"NORMAL",
        "vertices": [
            {
                "name": "Alice",
                "age": 31,
                "_id": "1",
                "_type": "vertex"
            },
            {
                "name": "Bob",
                "age": 27,
                "_id": "2",
                "_type": "vertex"
            }
        ],
        "edges": [
            {
                "type": "friends",
                "_id": "3",
                "_type": "edge",
                "_outV": "1",
                "_inV": "2",
                "_label": "knows"
            }
        ]
    }
}
```

4.2.3 GML Data Format

The Graph Modeling Language (GML) file format uses ASCII to describe graphs. Example 4-3 shows a GML description of the property graph shown in Figure 4-1.

See Also:

"GML: A Portable Graph File Format" by Michael Himsolt at

`http://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf`

Example 4-3 GML Description of a Simple Property Graph

```
graph [
   comment "Simple property graph"
   directed 1
   IsPlanar 1
   node [
      id 1
      label "1"
      name "Alice"
      age 31
        ]
   node [
      id 2
      label "2"
      name "Bob"
```

```
      age 27
        ]
   edge [
      source 1
      target 2
      label "knows"
      type "friends"
        ]
      ]
```

4.2.4 Oracle Flat File Format

The Oracle flat file format exclusively describes property graphs. It is more concise and provides better data type support than the other file formats. The Oracle flat file format uses two files for a graph description, one for the vertices and one for edges. Commas separate the fields of the records.

Example 4-4 shows the Oracle flat files that describe the property graph shown in Figure 4-1.

See Also:

"Oracle Flat File Format Definition"

Example 4-4 Oracle Flat File Description of a Simple Property Graph

Vertex file:

```
1,name,1,Alice,,
1,age,2,,31,
2,name,1,Bob,,
2,age,2,,27,
```

Edge file:

```
1,1,2,knows,type,1,friends,,
```

4.3 Getting Started with Property Graphs

To get started with property graphs:

1. The first time you use property graphs, ensure that the software is installed and operational.
2. Create your Java programs, using the classes provided in the Java API.

   See "Using Java APIs for Property Graph Data".

4.4 Using Java APIs for Property Graph Data

Creating a property graph involves using the Java APIs to create the property graph and objects in it.

- Overview of the Java APIs
- Parallel Loading of Graph Data
- Opening and Closing a Property Graph Instance

- Creating the Vertices
- Creating the Edges
- Deleting the Vertices and Edges
- Reading a Graph from a Database into an Embedded In-Memory Analyst
- Building an In-Memory Graph
- Dropping a Property Graph

4.4.1 Overview of the Java APIs

The Java APIs that you can use for property graphs include:

- Oracle Big Data Spatial and Graph Java APIs
- TinkerPop Blueprints Java APIs
- Apache Hadoop Java APIs
- Oracle NoSQL Database Java APIs
- Apache HBase Java APIs

4.4.1.1 Oracle Big Data Spatial and Graph Java APIs

Oracle Big Data Spatial and Graph property graph support provides database-specific APIs for Apache HBase and Oracle NoSQL Database. The data access layer API (`oracle.pg.*`) implements TinkerPop Blueprints APIs, text search, and indexing for property graphs stored in Oracle NoSQL Database and Apache HBase.

To use the Oracle Big Data Spatial and Graph API, import the classes into your Java program:

```
import oracle.pg.nosql.*; // or oracle.pg.hbase.*
import oracle.pgx.config.*;
import oracle.pgx.common.types.*;
```

Also include TinkerPop Blueprints Java APIs.

See Also:

Oracle Big Data Spatial and Graph Java API Reference

4.4.1.2 TinkerPop Blueprints Java APIs

TinkerPop Blueprints supports the property graph data model. The API provides utilities for manipulating graphs, which you use primarily through the Big Data Spatial and Graph data access layer Java APIs.

To use the Blueprints APIs, import the classes into your Java program:

```
import com.tinkerpop.blueprints.Vertex;
import com.tinkerpop.blueprints.Edge;
```

See Also:

"Blueprints: A Property Graph Model Interface API" at

`http://www.tinkerpop.com/docs/javadocs/blueprints/2.3.0/index.html`

4.4.1.3 Apache Hadoop Java APIs

The Apache Hadoop Java APIs enable you to write your Java code as a MapReduce program that runs within the Hadoop distributed framework.

To use the Hadoop Java APIs, import the classes into your Java program. For example:

```
import org.apache.hadoop.conf.Configuration;
```

See Also:

"Apache Hadoop Main 2.5.0-cdh5.3.2 API" at

`http://archive.cloudera.com/cdh5/cdh/5/hadoop/api/`

4.4.1.4 Oracle NoSQL Database Java APIs

The Oracle NoSQL Database APIs enable you to create and populate a key-value (KV) store, and provide interfaces to Hadoop, Hive, and Oracle NoSQL Database.

To use Oracle NoSQL Database as the graph data store, import the classes into your Java program. For example:

```
import oracle.kv.*;
import oracle.kv.table.TableOperation;
```

See Also:

"Oracle NoSQL Database Java API Reference" at

`http://docs.oracle.com/cd/NOSQL/html/javadoc/`

4.4.1.5 Apache HBase Java APIs

The Apache HBase APIs enable you to create and manipulate key-value pairs.

To use HBase as the graph data store, import the classes into your Java program. For example:

```
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.conf.Configuration;
```

See Also:

"HBase 0.98.6-cdh5.3.2 API" at

`http://archive.cloudera.com/cdh5/cdh/5/hbase/apidocs/index.html?overview-summary.html`

4.4.2 Parallel Loading of Graph Data

A Java API is provided for performing parallel loading of graph data.

Given a set of vertex files (or input streams) and a set of edge files (or input streams), they can be split into multiple chunks and loaded into database in parallel. The number of chunks is determined by the degree of parallelism (DOP) specified by the user.

Parallelism is achieved with Splitter threads that split vertex and edge flat files into multiple chunks and Loader threads that load each chunk into the database using separate database connections. Java pipes are used to connect Splitter and Loader threads -- Splitter: `PipedOutputStream` and Loader: `PipedInputStream`.

The simplest usage of data loading API is specifying a property graph instance, one vertex file, one edge file, and a DOP.

The following example of the load process loads graph data stored in a vertices file and an edges file of the optimized Oracle flat file format, and executes the load with 48 degrees of parallelism.

```
opgdl = OraclePropertyGraphDataLoader.getInstance();
vfile = "../../data/connections.opv";
efile = "../../data/connections.ope";
opgdl.loadData(opg, vfile, efile, 48);
```

4.4.2.1 Parallel Data Loading Using Partitions

The data loading API allows loading the data into database using multiple partitions. This API requires the property graph, the vertex file, the edge file, the DOP, the total number of partitions, and the partition offset (from 0 to total number of partitions - 1). For example, to load the data using two partitions, the partition offsets should be 0 and 1. That is, there should be two data loading API calls to fully load the graph, and the only difference between the two API calls is the partition offset (0 and 1).

The following code fragment loads the graph data using 4 partitions. Each call to the data loader can be processed using a separate Java client, on a single system or from multiple systems.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

int totalPartitions = 4;
int dop= 32; // degree of parallelism for each client.

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
SimpleLogBasedDataLoaderListenerImpl dll =
SimpleLogBasedDataLoaderListenerImpl.getInstance(100 /* frequency */,
                                       true /* Continue on error */);

// Run the data loading using 4 partitions (Each call can be run from a
// separate Java Client)
```

```
// Partition 1
OraclePropertyGraphDataLoader opgdlP1 = OraclePropertyGraphDataLoader.getInstance();
opgdlP1.loadData(opg, szOPVFile, szOPEFile, dop,
   4 /* Total number of partitions, default 1 */,
   0 /* Partition to load (from 0 to totalPartitions - 1, default 0 */,
   dll);

// Partition 2
OraclePropertyGraphDataLoader opgdlP2 = OraclePropertyGraphDataLoader.getInstance();
opgdlP2.loadData(opg, szOPVFile, szOPEFile, dop,  4 /* Total number of partitions,
default 1 */,
 1 /* Partition to load (from 0 to totalPartitions - 1, default 0 */, dll);


// Partition 3
OraclePropertyGraphDataLoader opgdlP3 = OraclePropertyGraphDataLoader.getInstance();
opgdlP3.loadData(opg, szOPVFile, szOPEFile, dop,  4 /* Total number of partitions,
default 1 */,
 2 /* Partition to load (from 0 to totalPartitions - 1, default 0 */, dll);

// Partition 4
OraclePropertyGraphDataLoader opgdlP4 = OraclePropertyGraphDataLoader.getInstance();
opgdlP4.loadData(opg, szOPVFile, szOPEFile, dop,  4 /* Total number of partitions,
default 1 */,
 3 /* Partition to load (from 0 to totalPartitions - 1, default 0 */, dll);
```

4.4.2.2 Parallel Data Loading Using Fine-Tuning

Data loading APIs also support fine-tuning those lines in the source vertex and edges files that are to be loaded. You can specify the vertex (or edge) offset line number and vertex (or edge) maximum line number. Data will be loaded from the offset line number until the maximum line number. If the maximum line number is -1, the loading process will scan the data until reaching the end of file.

The following code fragment loads the graph data using fine-tuning.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

int totalPartitions = 4;
int dop= 32; // degree of parallelism for each client.

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
SimpleLogBasedDataLoaderListenerImpl dll =
SimpleLogBasedDataLoaderListenerImpl.getInstance(100 /* frequency */,
                                       true /* Continue on error */);

// Run the data loading using fine tuning
long lVertexOffsetlines = 0;
long lEdgeOffsetLines = 0;
long lVertexMaxlines = 100;
long lEdgeMaxlines = 100;
int totalPartitions = 1;
int idPartition = 0;

OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(m_opg, szOPVFile, szOPEFile,
 lVertexOffsetlines /* offset of lines to start loading
 from partition, default 0*/,
```

```
lEdgeOffsetlines /* offset of lines to start loading
from partition, default 0*/,
lVertexMaxlines /* maximum number of lines to start loading
from partition, default -1 (all lines in partition)*/,
lEdgeMaxlines /* maximun number of lines to start loading
from partition, default -1 (all lines in partition)*/,
dop,
totalPartitions /* Total number of partitions, default 1 */,
idPartition /* Partition to load (from 0 to totalPartitions - 1,
default 0 */,
dll);
```

4.4.2.3 Parallel Data Loading Using Multiple Files

Oracle Big Data Spatial and Graph also support loading multiple vertex files and multiple edges files into database. The given multiple vertex files will be split into DOP chunks and loaded into database in parallel using DOP threads. Similarly, the multiple edge files will also be split and loaded in parallel.

The following code fragment loads multiple vertex fan and edge files using the parallel data loading APIs. In the example, two string arrays szOPVFiles and szOPEFiles are used to hold the input files; Although only one vertex file and one edge file is used in this example, you can supply multiple vertex files and multiple edge files in these two arrays.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

String[] szOPVFiles = new String[] {"../../data/connections.opv"};
String[] szOPEFiles = new String[] {"../../data/connections.ope"};

// Clear existing vertices/edges in the property graph
opg.clearRepository();
opg.setQueueSize(100); // 100 elements

// This object will handle parallel data loading over the property graph
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();

opgdl.loadData(opg, szOPVFiles, szOPEFiles, dop);

System.out.println("Total vertices: " + opg.countVertices());
System.out.println("Total edges: " + opg.countEdges());
```

4.4.2.4 Parallel Retrieval of Graph Data

The parallel property graph query provides a simple Java API to perform parallel scans on vertices (or edges). Parallel retrieval is an optimized solution taking advantage of the distribution of the data among splits with the back-end database, so each split is queried using separate database connections.

Parallel retrieval will produce an array where each element holds all the vertices (or edges) from a specific split. The subset of shards queried will be separated by the given start split ID and the size of the connections array provided. This way, the subset will consider splits in the range of [start, start - 1 + size of connections array]. Note that an integer ID (in the range of [0, N - 1]) is assigned to all the splits in the vertex table with N splits.

The following code loads a property graph using Apache HBase, opens an array of connections, and executes a parallel query to retrieve all vertices and edges using the opened connections. The number of calls to the `getVerticesPartitioned` `(getEdgesPartitioned)` method is controlled by the total number of splits and the number of connections used.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create connections used in parallel query
HConnection hConns= new HConnection[dop];
for (int i = 0; i < dop; i++) {
Configuration conf_new =
HBaseConfiguration.create(opg.getConfiguration());
hConns[i] = HConnectionManager.createConnection(conf_new);
}


long lCountV = 0;
// Iterate over all the vertices' splits to count all the vertices
for (int split = 0; split < opg.getVertexTableSplits();
 split += dop) {
Iterable<Vertex>[] iterables
 = opg.getVerticesPartitioned(hConns /* Connection array */,
 true /* skip store to cache */,
 split /* starting split */);
lCountV += consumeIterables(iterables); /* consume iterables using
 threads */
}

// Count all vertices
System.out.println("Vertices found using parallel query: " + lCountV);

long lCountE = 0;
// Iterate over all the edges' splits to count all the edges
for (int split = 0; split < opg.getEdgeTableSplits();
 split += dop) {
Iterable<Edge>[] iterables
 = opg.getEdgesPartitioned(hConns /* Connection array */,
 true /* skip store to cache */,
 split /* starting split */);
lCountE += consumeIterables(iterables); /* consume iterables using
 threads */
}

// Count all edges
System.out.println("Edges found using parallel query: " + lCountE);

// Close the connections to the database after completed
for (int idx = 0; idx < hConns.length; idx++) {
hConns[idx].close();
}
```

4.4.2.5 Using an Element Filter Callback for Subgraph Extraction

Oracle Big Data Spatial and Graph provides support for an easy subgraph extraction using user-defined element filter callbacks. An element filter callback defines a set of conditions that a vertex (or an edge) must meet in order to keep it in the subgraph. Users can define their own element filtering by implementing the `VertexFilterCallback` and `EdgeFilterCallback` API interfaces.

The following code fragment implements a `VertexFilterCallback` that validates if a vertex does not have a political role and its origin is the United States.

```
/**
* VertexFilterCallback to retrieve a vertex from the United States
* that does not have a political role
*/
private static class NonPoliticianFilterCallback
implements VertexFilterCallback
{
@Override
public boolean keepVertex(OracleVertexBase vertex)
{
String country = vertex.getProperty("country");
String role = vertex.getProperty("role");

if (country != null && country.equals("United States")) {
if (role == null || !role.toLowerCase().contains("political")) {
return true;
}
}

return false;
}

public static NonPoliticianFilterCallback getInstance()
{
return new NonPoliticianFilterCallback();
}
}
```

The following code fragment implements an `EdgeFilterCallback` that uses the `VertexFilterCallback` to keep only edges connected to the given input vertex, and whose connections are not politicians and come from the United States.

```
/**
 * EdgeFilterCallback to retrieve all edges connected to an input
 * vertex with "collaborates" label, and whose vertex is from the
 * United States with a role different than political
*/
private static class CollaboratorsFilterCallback
implements EdgeFilterCallback
{
private VertexFilterCallback m_vfc;
private Vertex m_startV;

public CollaboratorsFilterCallback(VertexFilterCallback vfc,
 Vertex v)
{
m_vfc = vfc;
m_startV = v;
}
```

```
@Override
public boolean keepEdge(OracleEdgeBase edge)
{
if ("collaborates".equals(edge.getLabel())) {
if (edge.getVertex(Direction.IN).equals(m_startV) &&
m_vfc.keepVertex((OracleVertex)
edge.getVertex(Direction.OUT))) {
return true;
}
else if (edge.getVertex(Direction.OUT).equals(m_startV) &&
 m_vfc.keepVertex((OracleVertex)
edge.getVertex(Direction.IN))) {
return true;
}
}

return false;
}

public static CollaboratorsFilterCallback
getInstance(VertexFilterCallback vfc, Vertex v)
{
return new CollaboratorsFilterCallback(vfc, v);
}

}
```

Using the filter callbacks previously defined, the following code fragment loads a property graph, creates an instance of the filter callbacks and later gets all of Barack Obama’s collaborators who are not politicians and come from the United States.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// VertexFilterCallback to retrieve all people from the United States // who are not
politicians
NonPoliticianFilterCallback npvfc = NonPoliticianFilterCallback.getInstance();

// Initial vertex: Barack Obama
Vertex v = opg.getVertices("name", "Barack Obama").iterator().next();

// EdgeFilterCallback to retrieve all collaborators of Barack Obama
// from the United States who are not politicians
CollaboratorsFilterCallback cefc = CollaboratorsFilterCallback.getInstance(npvfc, v);

Iterable<<Edge> obamaCollabs = opg.getEdges((String[])null /* Match any
of the properties */,
cefc /* Match the
EdgeFilterCallback */
);
```

```
Iterator<<Edge> iter = obamaCollabs.iterator();

System.out.println("\n\n--------Collaborators of Barack Obama from " +
 " the US and non-politician\n\n");
long countV = 0;
while (iter.hasNext()) {
Edge edge = iter.next(); // get the edge
// check if obama is the IN vertex
if (edge.getVertex(Direction.IN).equals(v)) {
 System.out.println(edge.getVertex(Direction.OUT) + "(Edge ID: " +
 edge.getId() + ")"); // get out vertex
}
else {
System.out.println(edge.getVertex(Direction.IN)+ "(Edge ID: " +
 edge.getId() + ")"); // get in vertex
}

countV++;
}
```

By default, all reading operations such as get all vertices, get all edges (and parallel approaches) will use the filter callbacks associated with the property graph using the methods `opg.setVertexFilterCallback(vfc)` and `opg.setEdgeFilterCallback(efc)`. If there is no filter callback set, then all the vertices (or edges) and edges will be retrieved.

The following code fragment uses the default edge filter callback set on the property graph to retrieve the edges.

```
// VertexFilterCallback to retrieve all people from the United States // who are not
politicians
NonPoliticianFilterCallback npvfc = NonPoliticianFilterCallback.getInstance();

// Initial vertex: Barack Obama
Vertex v = opg.getVertices("name", "Barack Obama").iterator().next();

// EdgeFilterCallback to retrieve all collaborators of Barack Obama
// from the United States who are not politicians
CollaboratorsFilterCallback cefc = CollaboratorsFilterCallback.getInstance(npvfc, v);

opg.setEdgeFilterCallback(cefc);

Iterable<Edge> obamaCollabs = opg.getEdges();
Iterator<Edge> iter = obamaCollabs.iterator();

System.out.println("\n\n--------Collaborators of Barack Obama from " +
 " the US and non-politician\n\n");
long countV = 0;
while (iter.hasNext()) {
Edge edge = iter.next(); // get the edge
// check if obama is the IN vertex
if (edge.getVertex(Direction.IN).equals(v)) {
 System.out.println(edge.getVertex(Direction.OUT) + "(Edge ID: " +
 edge.getId() + ")"); // get out vertex
}
else {
System.out.println(edge.getVertex(Direction.IN)+ "(Edge ID: " +
 edge.getId() + ")"); // get in vertex
}
```

```
countV++;
}
```

4.4.2.6 Using Optimization Flags on Reads over Property Graph Data

Oracle Big Data Spatial and Graph provides support for optimization flags to improve graph iteration performance. Optimization flags allow processing vertices (or edges) as objects with none or minimal information, such as ID, label, and/or incoming/outgoing vertices. This way, the time required to process each vertex (or edge) during iteration is reduced.

The following table shows the optimization flags available when processing vertices (or edges) in a property graph.

Optimization Flag	Description
DO_NOT_CREATE_OBJECT	Use a predefined constant object when processing vertices or edges.
JUST_EDGE_ID	Construct edge objects with ID only when processing edges.
JUST_LABEL_EDGE_ID	Construct edge objects with ID and label only when processing edges.
JUST_LABEL_VERTEX_EDGE_ID	Construct edge objects with ID, label, and in/out vertex IDs only when processing edges
JUST_VERTEX_EDGE_ID	Construct edge objects with just ID and in/out vertex IDs when processing edges.
JUST_VERTEX_ID	Construct vertex objects with ID only when processing vertices.

The following code fragment uses a set of optimization flags to retrieve only all the IDs from the vertices and edges in the property graph. The objects retrieved by reading all vertices and edges will include only the IDs and no Key/Value properties or additional information.

```
import oracle.pg.common.OraclePropertyGraphBase.OptimizationFlag;
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);


// Optimization flag to retrieve only vertices IDs
OptimizationFlag optFlagVertex = OptimizationFlag.JUST_VERTEX_ID;

// Optimization flag to retrieve only edges IDs
OptimizationFlag optFlagEdge = OptimizationFlag.JUST_EDGE_ID;

// Print all vertices
Iterator<Vertex> vertices =
opg.getVertices((String[])null /* Match any of the
properties */,
```

```
null /* Match the VertexFilterCallback */,
optFlagVertex /* optimization flag */
).iterator();

System.out.println("----- Vertices IDs----");
long vCount = 0;
while (vertices.hasNext()) {
OracleVertex v = vertices.next();
System.out.println((Long) v.getId());
vCount++;
}
System.out.println("Vertices found: " + vCount);


// Print all edges
Iterator<Edge> edges =
opg.getEdges((String[])null /* Match any of the properties */,
null /* Match the EdgeFilterCallback */,
optFlagEdge /* optimization flag */
).iterator();

System.out.println("----- Edges ----");
long eCount = 0;
while (edges.hasNext()) {
Edge e = edges.next();
System.out.println((Long) e.getId());
eCount++;
}
System.out.println("Edges found: " + eCount);
```

By default, all reading operations such as get all vertices, get all edges (and parallel approaches) will use the optimization flag associated with the property graph using the method `opg.setDefaultVertexOptFlag(optFlagVertex)` and `opg.setDefaultEdgeOptFlag(optFlagEdge)`. If the optimization flags for processing vertices and edges are not defined, then all the information about the vertices and edges will be retrieved.

The following code fragment uses the default optimization flags set on the property graph to retrieve only all the IDs from its vertices and edges.

```
import oracle.pg.common.OraclePropertyGraphBase.OptimizationFlag;

// Optimization flag to retrieve only vertices IDs
OptimizationFlag optFlagVertex = OptimizationFlag.JUST_VERTEX_ID;

// Optimization flag to retrieve only edges IDs
OptimizationFlag optFlagEdge = OptimizationFlag.JUST_EDGE_ID;

opg.setDefaultVertexOptFlag(optFlagVertex);
opg.setDefaultEdgeOptFlag(optFlagEdge);

Iterator<Vertex> vertices = opg.getVertices().iterator();
System.out.println("----- Vertices IDs----");
long vCount = 0;
while (vertices.hasNext()) {
OracleVertex v = vertices.next();
System.out.println((Long) v.getId());
vCount++;
}
System.out.println("Vertices found: " + vCount);
```

```
// Print all edges
Iterator<Edge> edges = opg.getEdges().iterator();
System.out.println("----- Edges ----");
long eCount = 0;
while (edges.hasNext()) {
Edge e = edges.next();
System.out.println((Long) e.getId());
eCount++;
}
System.out.println("Edges found: " + eCount);
```

4.4.2.7 Adding and Removing Attributes of a Property Graph Subgraph

Oracle Big Data Spatial and Graph supports updating attributes (key/value pairs) to a subgraph of vertices and/or edges by using a user-customized operation callback. An operation callback defines a set of conditions that a vertex (or an edge) must meet in order to update it (either add or remove the given attribute and value).

You can define your own attribute operations by implementing the `VertexOpCallback` and `EdgeOpCallback` API interfaces. You must override the `needOp` method, which defines the conditions to be satisfied by the vertices (or edges) to be included in the update operation, as well as the `getAttributeKeyName` and `getAttributeKeyValue` methods, which return the key name and value, respectively, to be used when updating the elements.

The following code fragment implements a `VertexOpCallback` that operates over the `obamaCollaborator` attribute associated only with Barack Obama collaborators. The value of this property is specified based on the role of the collaborators.

```
private static class CollaboratorsVertexOpCallback
implements VertexOpCallback
{
private OracleVertexBase m_obama;
private List<Vertex> m_obamaCollaborators;

public CollaboratorsVertexOpCallback(OraclePropertyGraph opg)
{
// Get a list of Barack Obama'sCollaborators
m_obama = (OracleVertexBase) opg.getVertices("name",
 "Barack Obama")
.iterator().next();

Iterable<Vertex> iter = m_obama.getVertices(Direction.BOTH,
"collaborates");
m_obamaCollaborators = OraclePropertyGraphUtils.listify(iter);
}

public static CollaboratorsVertexOpCallback
getInstance(OraclePropertyGraph opg)
{
return new CollaboratorsVertexOpCallback(opg);
}

/**
 * Add attribute if and only if the vertex is a collaborator of Barack
 * Obama
*/
@Override
public boolean needOp(OracleVertexBase v)
{
```

```
return m_obamaCollaborators != null &&
 m_obamaCollaborators.contains(v);
}

@Override
public String getAttributeKeyName(OracleVertexBase v)
{
return "obamaCollaborator";
}

/**
 * Define the property's value based on the vertex role
 */
@Override
public Object getAttributeKeyValue(OracleVertexBase v)
{
String role = v.getProperty("role");
role = role.toLowerCase();
if (role.contains("political")) {
return "political";
}
else if (role.contains("actor") || role.contains("singer") ||
 role.contains("actress") || role.contains("writer") ||
 role.contains("producer") || role.contains("director")) {
return "arts";
}
else if (role.contains("player")) {
return "sports";
}
else if (role.contains("journalist")) {
return "journalism";
}
else if (role.contains("business") || role.contains("economist")) {
return "business";
}
else if (role.contains("philanthropist")) {
return "philanthropy";
}
return " ";
}
}
```

The following code fragment implements an `EdgeOpCallback` that operates over the `obamaFeud` attribute associated only with Barack Obama feuds. The value of this property is specified based on the role of the collaborators.

```
private static class FeudsEdgeOpCallback
implements EdgeOpCallback
{
private OracleVertexBase m_obama;
private List<Edge> m_obamaFeuds;

public FeudsEdgeOpCallback(OraclePropertyGraph opg)
{
// Get a list of Barack Obama's feuds
m_obama = (OracleVertexBase) opg.getVertices("name",
 "Barack Obama")
.iterator().next();

Iterable<Vertex> iter = m_obama.getVertices(Direction.BOTH,
"feuds");
```

```
m_obamaFeuds = OraclePropertyGraphUtils.listify(iter);
}

public static FeudsEdgeOpCallback getInstance(OraclePropertyGraph opg)
{
return new FeudsEdgeOpCallback(opg);
}

/**
 * Add attribute if and only if the edge is in the list of Barack Obama's
 * feuds
*/
@Override
public boolean needOp(OracleEdgeBase e)
{
return m_obamaFeuds != null && m_obamaFeuds.contains(e);
}

@Override
public String getAttributeKeyName(OracleEdgeBase e)
{
return "obamaFeud";
}

/**
 * Define the property's value based on the in/out vertex role
 */
@Override
public Object getAttributeKeyValue(OracleEdgeBase e)
{
OracleVertexBase v = (OracleVertexBase) e.getVertex(Direction.IN);
if (m_obama.equals(v)) {
v = (OracleVertexBase) e.getVertex(Direction.OUT);
}
String role = v.getProperty("role");
role = role.toLowerCase();

if (role.contains("political")) {
return "political";
}
else if (role.contains("actor") || role.contains("singer") ||
 role.contains("actress") || role.contains("writer") ||
 role.contains("producer") || role.contains("director")) {
return "arts";
}
else if (role.contains("journalist")) {
return "journalism";
}
else if (role.contains("player")) {
return "sports";
}
else if (role.contains("business") || role.contains("economist")) {
return "business";
}
else if (role.contains("philanthropist")) {
return "philanthropy";
}
return " ";
}
}
```

Using the operations callbacks defined previously, the following code fragment loads a property graph, creates an instance of the operation callbacks, and later adds the attributes into the pertinent vertices and edges using the `addAttributeToAllVertices` and `addAttributeToAllEdges` methods in `OraclePropertyGraph`.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create the vertex operation callback
CollaboratorsVertexOpCallback cvoc = CollaboratorsVertexOpCallback.getInstance(opg);

// Add attribute to all people collaborating with Obama based on their role
opg.addAttributeToAllVertices(cvoc, true /** Skip store to Cache */, dop);

// Look up for all collaborators of Obama
Iterable<Vertex> collaborators = opg.getVertices("obamaCollaborator", "political");
System.out.println("Political collaborators of Barack Obama " +
getVerticesAsString(collaborators));

collaborators = opg.getVertices("obamaCollaborator", "business");
System.out.println("Business collaborators of Barack Obama " +
getVerticesAsString(collaborators));

// Add an attribute to all people having a feud with Barack Obama to set
// the type of relation they have
FeudsEdgeOpCallback feoc = FeudsEdgeOpCallback.getInstance(opg);
opg.addAttributeToAllEdges(feoc, true /** Skip store to Cache */, dop);

// Look up for all feuds of Obama
Iterable<Edge> feuds = opg.getEdges("obamaFeud", "political");
System.out.println("\n\nPolitical feuds of Barack Obama " + getEdgesAsString(feuds));

feuds = opg.getEdges("obamaFeud", "business");
System.out.println("Business feuds of Barack Obama " +
getEdgesAsString(feuds));
```

The following code fragment defines an implementation of `VertexOpCallback` that can be used to remove vertices having value philanthropy for attribute `obamaCollaborator`, then call the API `removeAttributeFromAllVertices`; It also defines an implementation of `EdgeOpCallback` that can be used to remove edges having value business for attribute `obamaFeud`, then call the API `removeAttributeFromAllEdges`.

```
System.out.println("\n\nRemove 'obamaCollaborator' property from all the" +
 "philanthropy collaborators");
PhilanthropyCollaboratorsVertexOpCallback pvoc =
PhilanthropyCollaboratorsVertexOpCallback.getInstance();

opg.removeAttributeFromAllVertices(pvoc);
```

```
System.out.println("\n\nRemove 'obamaFeud' property from all the" + "business
feuds");
BusinessFeudsEdgeOpCallback beoc = BusinessFeudsEdgeOpCallback.getInstance();

opg.removeAttributeFromAllEdges(beoc);

/**
 * Implementation of a EdgeOpCallback to remove the "obamaCollaborators"
 * property from all people collaborating with Barack Obama that have a
 * philanthropy role
 */
private static class PhilanthropyCollaboratorsVertexOpCallback implements
VertexOpCallback
{
  public static PhilanthropyCollaboratorsVertexOpCallback getInstance()
  {
     return new PhilanthropyCollaboratorsVertexOpCallback();
  }

  /**
   * Remove attribute if and only if the property value for
   * obamaCollaborator is Philanthropy
   */
  @Override
  public boolean needOp(OracleVertexBase v)
  {
    String type = v.getProperty("obamaCollaborator");
    return type != null && type.equals("philanthropy");
  }

  @Override
  public String getAttributeKeyName(OracleVertexBase v)
  {
    return "obamaCollaborator";
  }

  /**
   * Define the property's value. In this case can be empty
   */
  @Override
  public Object getAttributeKeyValue(OracleVertexBase v)
  {
    return " ";
  }
}

/**
 * Implementation of a EdgeOpCallback to remove the "obamaFeud" property
 * from all connections in a feud with Barack Obama that have a business role
 */
private static class BusinessFeudsEdgeOpCallback implements EdgeOpCallback
{
  public static BusinessFeudsEdgeOpCallback getInstance()
  {
    return new BusinessFeudsEdgeOpCallback();
  }

  /**
   * Remove attribute if and only if the property value for obamaFeud is
   * business
   */
```

```
  @Override
  public boolean needOp(OracleEdgeBase e)
  {
    String type = e.getProperty("obamaFeud");
    return type != null && type.equals("business");
  }

 @Override
 public String getAttributeKeyName(OracleEdgeBase e)
 {
   return "obamaFeud";
 }

 /**
  * Define the property's value. In this case can be empty
  */
  @Override
  public Object getAttributeKeyValue(OracleEdgeBase e)
  {
    return " ";
  }
}
```

4.4.2.8 Getting Property Graph Metadata

You can get graph metadata and statistics, such as all graph names in the database; for each graph, getting the minimum/maximum vertex ID, the minimum/maximum edge ID, vertex property names, edge property names, number of splits in graph vertex, and the edge table that supports parallel table scans.

The following code fragment gets the metadata and statistics of the existing property graphs stored in the back-end database (either Oracle NoSQL Database or Apache HBase). The arguments required vary for each database.

```
// Get all graph names in the database
List<String> graphNames = OraclePropertyGraphUtils.getGraphNames(dbArgs);

for (String graphName : graphNames) {
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args,
graphName);

System.err.println("\n Graph name: " + graphName);
System.err.println(" Total vertices: " +
 opg.countVertices(dop));

System.err.println(" Minimum Vertex ID: " +
 opg.getMinVertexID(dop));
System.err.println(" Maximum Vertex ID: " +
 opg.getMaxVertexID(dop));

Set<String> propertyNamesV = new HashSet<String>();
opg.getVertexPropertyNames(dop, 0 /* timeout,0 no timeout */,
 propertyNamesV);

System.err.println(" Vertices property names: " +
getPropertyNamesAsString(propertyNamesV));

System.err.println("\n\n Total edges: " + opg.countEdges(dop));
System.err.println(" Minimum Edge ID: " + opg.getMinEdgeID(dop));
System.err.println(" Maximum Edge ID: " + opg.getMaxEdgeID(dop));
```

```
Set<String> propertyNamesE = new HashSet<String>();
opg.getEdgePropertyNames(dop, 0 /* timeout,0 no timeout */,
 propertyNamesE);

System.err.println(" Edge property names: " +
getPropertyNamesAsString(propertyNamesE));

System.err.println("\n\n Table Information: ");
System.err.println("Vertex table number of splits: " +
 (opg.getVertexTableSplits()));
System.err.println("Edge table number of splits: " +
 (opg.getEdgeTableSplits()));
}
```

4.4.3 Opening and Closing a Property Graph Instance

When describing a property graph, use these Oracle Property Graph classes to open and close the property graph instance properly:

- `OraclePropertyGraph.getInstance`: Opens an instance of an Oracle property graph. This method has two parameters, the connection information and the graph name. The format of the connection information depends on whether you use HBase or Oracle NoSQL Database as the backend database.
- `OraclePropertyGraph.clearRepository`: Removes all vertices and edges from the property graph instance.
- `OraclePropertyGraph.shutdown`: Closes the graph instance.

In addition, you must use the appropriate classes from the Oracle NoSQL Database or HBase APIs.

- Using Oracle NoSQL Database
- Using Apache HBase

4.4.3.1 Using Oracle NoSQL Database

For Oracle NoSQL Database, the `OraclePropertyGraph.getInstance` method uses the KV store name, host computer name, and port number for the connection:

```
String kvHostPort = "cluster02:5000";
String kvStoreName = "kvstore";
String kvGraphName = "my_graph";

// Use NoSQL Java API
KVStoreConfig kvconfig = new KVStoreConfig(kvStoreName, kvHostPort);

OraclePropertyGraph opg = OraclePropertyGraph.getInstance(kvconfig, kvGraphName);
opg.clearRepository();
//     .
//     .  Graph description
//     .
// Close the graph instance
opg.shutdown();
```

If the in-memory analyst functions are required for your application, then it is recommended that you use `GraphConfigBuilder` to create a graph `config` for Oracle NoSQL Database, and instantiates `OraclePropertyGraph` with the `config` as an argument.

As an example, the following code snippet constructs a graph `config`, gets an `OraclePropertyGraph` instance, loads some data into that graph, and gets an in-memory analyst.

```
  import oracle.pgx.config.*;
  import oracle.pgx.api.*;
  import oracle.pgx.common.types.*;

    ...

    String[] hhosts = new String[1];
    hhosts[0]          = "my_host_name:5000"; // need customization
    String szStoreName =  "kvstore";          // need customization
    String szGraphName = "my_graph";
    int dop            =  8;

    PgNosqlGraphConfig cfg = GraphConfigBuilder.forPropertyGraphNosql()
                                                 .setName(szGraphName)
                                                 .setHosts(Arrays.asList(hhosts))
                                                 .setStoreName(szStoreName)
                                                 .addEdgeProperty("lbl",
PropertyType.STRING, "lbl")
                                                 .addEdgeProperty("weight",
PropertyType.DOUBLE, "1000000")
                                                 .build();

    OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);

    String szOPVFile = "../../data/connections.opv";
    String szOPEFile = "../../data/connections.ope";

    // perform a parallel data load
    OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
    opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

    ...
    PgxSession session = Pgx.createSession("session-id-1");
    PgxGraph g = session.readGraphWithProperties(cfg);

    Analyst analyst = session.createAnalyst();
    ...
```

4.4.3.2 Using Apache HBase

For Apache HBase, the `OraclePropertyGraph.getInstance` method uses the Hadoop nodes and the Apache HBase port number for the connection:

```
String hbQuorum = "bda01node01.example.com, bda01node02.example.com,
bda01node03.example.com";
String hbClientPort = "2181"
String hbGraphName = "my_graph";

// Use HBase Java APIs
Configuration conf = HBaseConfiguration.create();
  conf.set("hbase.zookeeper.quorum", hbQuorum);
  conf.set("hbase.zookeper.property.clientPort", hbClientPort);
HConnection conn = HConnectionManager.createConnection(conf);

// Open the property graph
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(conf, conn, hbGraphName);
```

```
opg.clearRepository();
//     .
//     .  Graph description
//     .
// Close the graph instance
opg.shutdown();
// Close the HBase connection
conn.close();
```

If the in-memory analyst functions are required for your application, then it is recommended that you use `GraphConfigBuilder` to create a graph `config`, and instantiates `OraclePropertyGraph` with the `config` as an argument.

As an example, the following code snippet sets the configuration for in memory analytics, constructs a graph config for Apache HBase, instantiates an `OraclePropertyGraph` instance, gets an in-memory analyst, and counts the number of triangles in the graph.

```
    confPgx = new HashMap<PgxConfig.Field, Object>();
    confPgx.put(PgxConfig.Field.ENABLE_GM_COMPILER, false);
    confPgx.put(PgxConfig.Field.NUM_WORKERS_IO, dop + 2);
    confPgx.put(PgxConfig.Field.NUM_WORKERS_ANALYSIS, 8); // <= # of physical cores
    confPgx.put(PgxConfig.Field.NUM_WORKERS_FAST_TRACK_ANALYSIS, 2);
    confPgx.put(PgxConfig.Field.SESSION_TASK_TIMEOUT_SECS, 0);// no timeout set
    confPgx.put(PgxConfig.Field.SESSION_IDLE_TIMEOUT_SECS, 0);  // no timeout set
    ServerInstance instance = Pgx.getInstance();
    instance.startEngine(confPgx);

int iClientPort = Integer.parseInt(hbClientPort);
int splitsPerRegion = 2;

PgHbaseGraphConfig cfg = GraphConfigBuilder.forPropertyGraphHbase()
                           .setName(hbGraphName)
                           .setZkQuorum(hbQuorum)
                           .setZkClientPort(iClientPort)
                           .setZkSessionTimeout(60000)
                           .setMaxNumConnections(dop)
                           .setSplitsPerRegion(splitsPerRegion)
                           .addEdgeProperty("lbl", PropertyType.STRING, "lbl")
                           .addEdgeProperty("weight", PropertyType.DOUBLE, "1000000")
                           .build();

PgxSession session = Pgx.createSession("session-id-1");
PgxGraph g = session.readGraphWithProperties(cfg);
Analyst analyst = session.createAnalyst();

long triangles = analyst.countTriangles(g, false);
```

4.4.4 Creating the Vertices

To create a vertex, use these Oracle Property Graph methods:

- `OraclePropertyGraph.addVertex`: Adds a vertex instance to a graph.
- `OracleVertex.setProperty`: Assigns a key-value property to a vertex.
- `OraclePropertyGraph.commit`: Saves all changes to the property graph instance.

The following code fragment creates two vertices named `V1` and `V2`, with properties for age, name, weight, height, and sex in the `opg` property graph instance. The `v1` properties set the data types explicitly.

```
// Create vertex v1 and assign it properties as key-value pairs
Vertex v1 = opg.addVertex(1l);
  v1.setProperty("age",  Integer.valueOf(31));
  v1.setProperty("name", "Alice");
  v1.setProperty("weight", Float.valueOf(135.0f));
  v1.setProperty("height", Double.valueOf(64.5d));
  v1.setProperty("female", Boolean.TRUE);

Vertex v2 = opg.addVertex(2l);
  v2.setProperty("age",  27);
  v2.setProperty("name", "Bob");
  v2.setProperty("weight", Float.valueOf(156.0f));
  v2.setProperty("height", Double.valueOf(69.5d));
  v2.setProperty("female", Boolean.FALSE);
```

4.4.5 Creating the Edges

To create an edge, use these Oracle Property Graph methods:

- `OraclePropertyGraph.addEdge`: Adds an edge instance to a graph.
- `OracleEdge.setProperty`: Assigns a key-value property to an edge.

The following code fragment creates two vertices (`v1` and `v2`) and one edge (`e1`).

```
// Add vertices v1 and v2
Vertex v1 = opg.addVertex(1l);
v1.setProperty("name", "Alice");
v1.setProperty("age", 31);

Vertex v2 = opg.addVertex(2l);
v2.setProperty("name", "Bob");
v2.setProperty("age", 27);

// Add edge e1
Edge e1 = opg.addEdge(1l, v1, v2, "knows");
e1.setProperty("type", "friends");
```

4.4.6 Deleting the Vertices and Edges

You can remove vertex and edge instances individually, or all of them simultaneously. Use these methods:

- `OraclePropertyGraph.removeEdge`: Removes the specified edge from the graph.
- `OraclePropertyGraph.removeVertex`: Removes the specified vertex from the graph.
- `OraclePropertyGraph.clearRepository`: Removes all vertices and edges from the property graph instance.

The following code fragment removes edge `e1` and vertex `v1` from the graph instance. The adjacent edges will also be deleted from the graph when removing a vertex. This is because every edge must have an beginning and ending vertex. After removing the beginning or ending vertex, the edge is no longer a valid edge.

```
// Remove edge e1
opg.removeEdge(e1);

// Remove vertex v1
opg.removeVertex(v1);
```

The `OraclePropertyGraph.clearRepository` method can be used to remove all contents from an `OraclePropertyGraph` instance. However, use it with care because this action cannot be reversed.

4.4.7 Reading a Graph from a Database into an Embedded In-Memory Analyst

You can read a graph from Apache HBase or Oracle NoSQL Database into an in-memory analyst that is embedded in the same client Java application (a single JVM). For the following Apache HBase example:

- A correct `java.io.tmpdir` setting is required.
- `dop + 2` is a workaround for a performance issue before Release 1.1.2. Effective with Release 1.1.2, you can instead specify a `dop` value directly in the configuration settings.

```
int dop = 8;                    // need customization
Map<PgxConfig.Field, Object> confPgx = new HashMap<PgxConfig.Field, Object>();
confPgx.put(PgxConfig.Field.ENABLE_GM_COMPILER, false);
confPgx.put(PgxConfig.Field.NUM_WORKERS_IO, dop + 2);   // use dop directly with
release 1.1.2 or newer
confPgx.put(PgxConfig.Field.NUM_WORKERS_ANALYSIS, dop); // <= # of physical cores
confPgx.put(PgxConfig.Field.NUM_WORKERS_FAST_TRACK_ANALYSIS, 2);
confPgx.put(PgxConfig.Field.SESSION_TASK_TIMEOUT_SECS, 0);  // no timeout set
confPgx.put(PgxConfig.Field.SESSION_IDLE_TIMEOUT_SECS, 0);  // no timeout set

PgHbaseGraphConfig cfg = GraphConfigBuilder.forPropertyGraphHbase()
                          .setName("mygraph")
                          .setZkQuorum("localhost") // quorum, need customization
                          .setZkClientPort(2181)
                          .addNodeProperty("name", PropertyType.STRING,
"default_name")
                          .build();

OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);
ServerInstance localInstance = Pgx.getInstance();
localInstance.startEngine(confPgx);
PgxSession session = localInstance.createSession("session-id-1"); // Put your
session description here.

Analyst analyst = session.createAnalyst();

// The following call will trigger a read of graph data from the database
PgxGraph pgxGraph = session.readGraphWithProperties(opg.getConfig());

long triangles = analyst.countTriangles(pgxGraph, false);
System.out.println("triangles " + triangles);

// Remove edge e1
opg.removeEdge(e1);

// Remove vertex v1
opg.removeVertex(v1);
```

4.4.8 Building an In-Memory Graph

In addition to Reading Graph Data into Memory, you can create an in-memory graph programmatically. This can simplify development when the size of graph is small or when the content of the graph is highly dynamic. The key Java class is `GraphBuilder`, which can accumulate a set of vertices and edges added with the `addVertex` and `addEdge` APIs. After all changes are made, an in-memory graph instance (`PgxGraph`) can be created by the `GraphBuilder`.

The following Java code snippet illustrates a graph construction flow. Note that there are no explicit calls to `addVertex`, because any vertex that does not already exist will be added dynamically as its adjacent edges are created.

```
import oracle.pgx.api.*;

PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder = session.newGraphBuilder();

builder.addEdge(0, 1, 2);
builder.addEdge(1, 2, 3);
builder.addEdge(2, 2, 4);
builder.addEdge(3, 3, 4);
builder.addEdge(4, 4, 2);

PgxGraph graph = builder.build();
```

To construct a graph with vertex properties, you can use `setProperty` against the vertex objects created.

```
PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder = session.newGraphBuilder();

builder.addVertex(1).setProperty("double-prop", 0.1);
builder.addVertex(2).setProperty("double-prop", 2.0);
builder.addVertex(3).setProperty("double-prop", 0.3);
builder.addVertex(4).setProperty("double-prop", 4.56789);

builder.addEdge(0, 1, 2);
builder.addEdge(1, 2, 3);
builder.addEdge(2, 2, 4);
builder.addEdge(3, 3, 4);
builder.addEdge(4, 4, 2);

PgxGraph graph = builder.build();
```

To use long integers as vertex and edge identifiers, specify `IdType.LONG` when getting a new instance of `GraphBuilder`. For example:

```
import oracle.pgx.common.types.IdType;
GraphBuilder<Long> builder = session.newGraphBuilder(IdType.LONG);
```

During edge construction, you can directly use vertex objects that were previously created in a call to `addEdge`.

```
v1 = builder.addVertex(1l).setProperty("double-prop", 0.5)
v2 = builder.addVertex(2l).setProperty("double-prop", 2.0)

builder.addEdge(0, v1, v2)
```

As with vertices, edges can have properties. The following example sets the edge label by using `setLabel`:

```
builder.addEdge(4, v4, v2).setProperty("edge-prop",
"edge_prop_4_2").setLabel("label")
```

4.4.9 Dropping a Property Graph

To drop a property graph from the database, use the `OraclePropertyGraphUtils.dropPropertyGraph` method. This method has two parameters, the connection information and the graph name.

The format of the connection information depends on whether you use HBase or Oracle NoSQL Database as the backend database. It is the same as the connection information you provide to `OraclePropertyGraph.getInstance`.

- Using Oracle NoSQL Database
- Using Apache HBase

4.4.9.1 Using Oracle NoSQL Database

For Oracle NoSQL Database, the `OraclePropertyGraphUtils.dropPropertyGraph` method uses the KV store name, host computer name, and port number for the connection. This code fragment deletes a graph named `my_graph` from Oracle NoSQL Database.

```
String kvHostPort = "cluster02:5000";
String kvStoreName = "kvstore";
String kvGraphName = "my_graph";

// Use NoSQL Java API
KVStoreConfig kvconfig = new KVStoreConfig(kvStoreName, kvHostPort);

// Drop the graph
OraclePropertyGraphUtils.dropPropertyGraph(kvconfig, kvGraphName);
```

4.4.9.2 Using Apache HBase

For Apache HBase, the `OraclePropertyGraphUtils.dropPropertyGraph` method uses the Hadoop nodes and the Apache HBase port number for the connection. This code fragment deletes a graph named `my_graph` from Apache HBase.

```
String hbQuorum = "bda01node01.example.com, bda01node02.example.com,
bda01node03.example.com";
String hbClientPort = "2181";
String hbGraphName = "my_graph";

// Use HBase Java APIs
Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", hbQuorum);
    conf.set("hbase.zookeper.property.clientPort", hbClientPort);

// Drop the graph
OraclePropertyGraphUtils.dropPropertyGraph(conf, hbGraphName);
```

4.5 Managing Text Indexing for Property Graph Data

Indexes in Oracle Big Data Spatial and Graph allow fast retrieval of elements by a particular key/value or key/text pair. These indexes are created based on an element type (vertices or edges), a set of keys (and values), and an index type.

Two types of indexing structures are supported by Oracle Big Data Spatial and Graph: manual and automatic.

- Automatic text indexes provide automatic indexing of vertices or edges by a set of property keys. Their main purpose is to enhance query performance on vertices and edges based on particular key/value pairs.
- Manual text indexes enable you to define multiple indexes over a designated set of vertices and edges of a property graph. You must specify what graph elements go into the index.

Oracle Big Data Spatial and Graph provides APIs to create manual and automatic text indexes over property graphs for Oracle NoSQL Database and Apache HBase. Indexes are managed using the available search engines, Apache Lucene and SolrCloud. The rest of this section focuses on how to create text indexes using the property graph capabilities of the Data Access Layer.

- Using Automatic Indexes with the Apache Lucene Search Engine
- Using Manual Indexes with the SolrCloud Search Engine
- Handling Data Types
- Uploading a Collection's SolrCloud Configuration to Zookeeper
- Updating Configuration Settings on Text Indexes for Property Graph Data
- Using Parallel Query on Text Indexes for Property Graph Data
- Using Native Query Objects on Text Indexes for Property Graph Data

4.5.1 Using Automatic Indexes with the Apache Lucene Search Engine

The supplied examples ExampleNoSQL6 and ExampleHBase6 create a property graph from an input file, create an automatic text index on vertices, and execute some text search queries using Apache Lucene.

The following code fragment creates an automatic index over an existing property graph's vertices with these property keys: name, role, religion, and country. The automatic text index will be stored under four subdirectories under the `/home/data/text-index` directory. Apache Lucene data types handling is enabled. This example uses a DOP (parallelism) of 4 for re-indexing tasks.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
      args,  szGraphName);

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do a parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
```

```
// Create an automatic index using Apache Lucene engine.
// Specify Index Directory parameters (number of directories,
// number of connections to database, batch size, commit size,
// enable datatypes, location)
OracleIndexParameters indexParams =
     OracleIndexParameters.buildFS(4, 4, 10000, 50000, true,
             "/home/data/text-index ");
opg.setDefaultIndexParameters(indexParams);

// specify indexed keys
String[] indexedKeys = new String[4];
indexedKeys[0] = "name";
indexedKeys[1] = "role";
indexedKeys[2] = "religion";
indexedKeys[3] = "country";

// Create auto indexing on above properties for all vertices
opg.createKeyIndex(indexedKeys, Vertex.class);
```

By default, indexes are configured based on the `OracleIndexParameters` associated with the property graph using the method `opg.setDefaultIndexParameters(indexParams)`.

Indexes can also be created by specifying a different set of parameters. This is shown in the following code snippet.

```
// Create an OracleIndexParameters object to get Index configuration (search engine,
etc).
OracleIndexParameters indexParams = OracleIndexParameters.buildFS(args)

// Create auto indexing on above properties for all vertices
opg.createKeyIndex("name", Vertex.class, indexParams.getParameters());
```

The code fragment in the next example executes a query over all vertices to find all matching vertices with the key/value pair `name:Barack Obama`. This operation will execute a lookup into the text index.

Additionally, wildcard searches are supported by specifying the parameter `useWildCards` in the `getVertices` API call. Wildcard search is only supported when automatic indexes are enabled for the specified property key. For details on text search syntax using Apache Lucene, see `https://lucene.apache.org/core/2_9_4/queryparsersyntax.html`.

```
// Find all vertices with name Barack Obama.
    Iterator<Vertices> vertices = opg.getVertices("name", "Barack Obama").iterator();
    System.out.println("----- Vertices with name Barack Obama -----");
    countV = 0;
    while (vertices.hasNext()) {
      System.out.println(vertices.next());
      countV++;
    }
    System.out.println("Vertices found: " + countV);

   // Find all vertices with name including keyword "Obama"
   // Wildcard searching is supported.
    boolean useWildcard = true;
    Iterator<Vertices> vertices = opg.getVertices("name", "*Obama*").iterator();
    System.out.println("----- Vertices with name *Obama* -----");
    countV = 0;
    while (vertices.hasNext()) {
```

```
      System.out.println(vertices.next());
      countV++;
    }
    System.out.println("Vertices found: " + countV);
```

The preceding code example produces output like the following:

```
----- Vertices with name Barack Obama-----
Vertex ID 1 {name:str:Barack Obama, role:str:political authority, occupation:str:
44th president of United States of America, country:str:United States, political
party:str:Democratic, religion:str:Christianity}
Vertices found: 1

----- Vertices with name *Obama* -----
Vertex ID 1 {name:str:Barack Obama, role:str:political authority, occupation:str:
44th president of United States of America, country:str:United States, political
party:str:Democratic, religion:str:Christianity}
Vertices found: 1
```

See Also:

Exploring the Sample Programs

4.5.2 Using Manual Indexes with the SolrCloud Search Engine

The supplied examples ExampleNoSQL7 and ExampleHBase7 create a property graph from an input file, create a manual text index on edges, put some data into the index, and execute some text search queries using Apache SolrCloud.

When using SolrCloud, you must first load a collection's configuration for the text indexes into Apache Zookeeper, as described in Uploading a Collection's SolrCloud Configuration to Zookeeper.

The following code fragment creates a manual text index over an existing property graph using four shards, one shard per node, and a replication factor of 1. The number of shards corresponds to the number of nodes in the SolrCloud cluster.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args,
                                                          szGraphName);

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do a parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create a manual text index using SolrCloud// Specify Index Directory parameters:
configuration name, Solr Server URL, Solr Node set,
// replication factor, zookeeper timeout (secs),
// maximum number of shards per node,
   // number of connections to database, batch size, commit size,
         // write timeout (in secs)
             String configName = "opgconfig";
             String solrServerUrl = "nodea:2181/solr"
             String solrNodeSet = "nodea:8983_solr,nodeb:8983_solr," +
                                  "nodec:8983_solr,noded:8983_solr";

         int zkTimeout = 15;
```

```
         int numShards = 4;
         int replicationFactor = 1;
         int maxShardsPerNode = 1;

OracleIndexParameters indexParams =
                OracleIndexParameters.buildSolr(configName,
                                      solrServerUrl,
                                      solrNodeSet,
                                      zkTimeout,
                                      numShards,
                                      replicationFactor,
                                      maxShardsPerNode,
                                      4,
                                      10000,
                                      500000,
                                      15);
opg.setDefaultIndexParameters(indexParams);


// Create manual indexing on above properties for all vertices
OracleIndex<Edge> index = ((OracleIndex<Edge>) opg.createIndex("myIdx", Edge.class));

Vertex v1 = opg.getVertices("name", "Barack Obama").iterator().next();

Iterator<Edge> edges
               = v1.getEdges(Direction.OUT, "collaborates").iterator();

          while (edges.hasNext()) {
             Edge edge = edges.next();
             Vertex vIn = edge.getVertex(Direction.IN);
             index.put("collaboratesWith", vIn.getProperty("name"), edge);
          }
```

The next code fragment executes a query over the manual index to get all edges with the key/value pair `collaboratesWith:Beyonce`. Additionally, wildcards search can be supported by specifying the parameter `useWildCards` in the get API call.

```
// Find all edges with collaboratesWith Beyonce.
   // Wildcard searching is supported using true parameter.
    edges = index.get("collaboratesWith", "Beyonce").iterator();
    System.out.println("----- Edges with name Beyonce -----");
    countE = 0;
    while (edges.hasNext()) {
      System.out.println(edges.next());
      countE++;
    }
    System.out.println("Edges found: "+ countE);

// Find all vertices with name including Bey*.
   // Wildcard searching is supported using true parameter.
    edges = index.get("collaboratesWith", "*Bey*", true).iterator();
    System.out.println("----- Edges with collaboratesWith Bey* -----");
    countE = 0;
    while (edges.hasNext()) {
      System.out.println(edges.next());
      countE++;
    }
    System.out.println("Edges found: " + countE);
```

The preceding code example produces output like the following:

```
----- Edges with name Beyonce -----
Edge ID 1000 from Vertex ID 1 {country:str:United States, name:str:Barack Obama,
occupation:str:44th president of United States of America, political
party:str:Democratic, religion:str:Christianity, role:str:political authority}
=[collaborates]=> Vertex ID 2 {country:str:United States, music genre:str:pop soul ,
name:str:Beyonce, role:str:singer actress} edgeKV[{weight:flo:1.0}]
Edges found: 1

----- Edges with collaboratesWith Bey* -----
Edge ID 1000 from Vertex ID 1 {country:str:United States, name:str:Barack Obama,
occupation:str:44th president of United States of America, political
party:str:Democratic, religion:str:Christianity, role:str:political authority}
=[collaborates]=> Vertex ID 2 {country:str:United States, music genre:str:pop soul ,
name:str:Beyonce, role:str:singer actress} edgeKV[{weight:flo:1.0}]
Edges found: 1
```

See Also:

Exploring the Sample Programs

4.5.3 Handling Data Types

Oracle's property graph support indexes and stores an element's Key/Value pairs based on the value data type. The main purpose of handling data types is to provide extensive query support like numeric and date range queries.

By default, searches over a specific key/value pair are matched up to a query expression based on the value's data type. For example, to find vertices with the key/value pair `age:30`, a query is executed over all age fields with a data type integer. If the value is a query expression, you can also specify the data type class of the value to find by calling the API `get(String key, Object value, Class dtClass, Boolean useWildcards)`. If no data type is specified, the query expression will be matched to all possible data types.

When dealing with Boolean operators, each subsequent key/value pair must append the data type's prefix/suffix so the query can find proper matches. The following topics describe how to append this prefix/suffix for Apache Lucene and SolrCloud.

- Appending Data Type Identifiers on Apache Lucene
- Appending Data Type Identifiers on SolrCloud

4.5.3.1 Appending Data Type Identifiers on Apache Lucene

When Lucene's data types handling is enabled, you must append the proper data type identifier as a suffix to the key in the query expression. This can be done by executing a `String.concat()` operation to the key. If Lucene's data types handling is disabled, you must insert the data type identifier as a prefix in the value String. Table 4-1 shows the data type identifiers available for text indexing using Apache Lucene (see also the Javadoc for `LuceneIndex`).

Table 4-1 Apache Lucene Data Type Identifiers

Lucene Data Type Identifier	Description
TYPE_DT_STRING	String
TYPE_DT_BOOL	Boolean

Table 4-1 (Cont.) Apache Lucene Data Type Identifiers

Lucene Data Type Identifier	Description
TYPE_DT_DATE	Date
TYPE_DT_FLOAT	Float
TYPE_DT_DOUBLE	Double
TYPE_DT_INTEGER	Integer
TYPE_DT_SERIALIZABLE	Serializable

The following code fragment creates a manual index on edges using Lucene's data type handling, adds data, and later executes a query over the manual index to get all edges with the key/value pair `collaboratesWith:Beyonce AND country1:United*` using wildcards.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args,
                                                          szGraphName);

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do a parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Specify Index Directory parameters (number of directories,
   // number of connections to database, batch size, commit size,
         // enable datatypes, location)
OracleIndexParameters indexParams =
     OracleIndexParameters.buildFS(4, 4, 10000, 50000, true,
            "/home/data/text-index ");
opg.setDefaultIndexParameters(indexParams);
// Create manual indexing on above properties for all edges
OracleIndex<Edge> index = ((OracleIndex<Edge>) opg.createIndex("myIdx", Edge.class));

Vertex v1 = opg.getVertices("name", "Barack Obama").iterator().next();

Iterator<Edge> edges
                = v1.getEdges(Direction.OUT, "collaborates").iterator();

          while (edges.hasNext()) {
             Edge edge = edges.next();
             Vertex vIn = edge.getVertex(Direction.IN);
             index.put("collaboratesWith", vIn.getProperty("name"), edge);
             index.put("country", vIn.getProperty("country"), edge);
          }

// Wildcard searching is supported using true parameter.
    String key = "country";
    key =
key.concat(String.valueOf(oracle.pg.text.lucene.LuceneIndex.TYPE_DT_STRING));

    String queryExpr = "Beyonce AND " + key + ":United*";
    edges = index.get("collaboratesWith", queryExpr, true /
*UseWildcard*/).iterator();
```

```
    System.out.println("----- Edges with query: " + queryExpr + " -----");
    countE = 0;
    while (edges.hasNext()) {
      System.out.println(edges.next());
      countE++;
    }
    System.out.println("Edges found: "+ countE);
```

The preceding code example might produce output like the following:

```
----- Edges with name Beyonce AND country1:United* -----
Edge ID 1000 from Vertex ID 1 {country:str:United States, name:str:Barack Obama,
occupation:str:44th president of United States of America, political
party:str:Democratic, religion:str:Christianity, role:str:political authority}
=[collaborates]=> Vertex ID 2 {country:str:United States, music genre:str:pop soul ,
name:str:Beyonce, role:str:singer actress} edgeKV[{weight:flo:1.0}]
Edges found: 1
```

The following code fragment creates an automatic index on vertices, disables Lucene's data type handling, adds data, and later executes a query over the manual index from a previous example to get all vertices with the key/value pair `country:United* AND role:1*political*` using wildcards.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args,
                                                          szGraphName);

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do a parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create an automatic index using Apache Lucene engine.
// Specify Index Directory parameters (number of directories,
   // number of connections to database, batch size, commit size,
         // enable datatypes, location)
OracleIndexParameters indexParams =
     OracleIndexParameters.buildFS(4, 4, 10000, 50000, false, "/ home/data/text-
index ");
opg.setDefaultIndexParameters(indexParams);

// specify indexed keys
String[] indexedKeys = new String[4];
indexedKeys[0] = "name";
indexedKeys[1] = "role";
indexedKeys[2] = "religion";
indexedKeys[3] = "country";

// Create auto indexing on above properties for all vertices
opg.createKeyIndex(indexedKeys, Vertex.class);

// Wildcard searching is supported using true parameter.
    String value = "*political*";
    value = String.valueOf(LuceneIndex.TYPE_DT_STRING) + value;
String queryExpr = "United* AND role:" + value;


vertices = opg.getVertices("country", queryExpr,  true /*useWildcard*/).iterator();
    System.out.println("----- Vertices with query: " + queryExpr + " -----");
```

```
countV = 0;
while (vertices.hasNext()) {
  System.out.println(vertices.next());
  countV++;
}
System.out.println("Vertices found: " + countV);
```

The preceding code example might produce output like the following:

```
----- Vertices with query: United* and role:1*political* -----
Vertex ID 30 {name:str:Jerry Brown, role:str:political authority, occupation:str:
34th and 39th governor of California, country:str:United States, political
party:str:Democratic, religion:str:roman catholicism}
Vertex ID 24 {name:str:Edward Snowden, role:str:political authority,
occupation:str:system administrator, country:str:United States,
religion:str:buddhism}
Vertex ID 22 {name:str:John Kerry, role:str:political authority, country:str:United
States, political party:str:Democratic, occupation:str:68th United States Secretary
of State, religion:str:Catholicism}
Vertex ID 21 {name:str:Hillary Clinton, role:str:political authority,
country:str:United States, political party:str:Democratic, occupation:str:67th
United States Secretary of State, religion:str:Methodism}
Vertex ID 19 {name:str:Kirsten Gillibrand, role:str:political authority,
country:str:United States, political party:str:Democratic, occupation:str:junior
United States Senator from New York, religion:str:Methodism}
Vertex ID 13 {name:str:Ertharin Cousin, role:str:political authority,
country:str:United States, political party:str:Democratic}
Vertex ID 11 {name:str:Eric Holder, role:str:political authority, country:str:United
States, political party:str:Democratic, occupation:str:United States Deputy Attorney
General}
Vertex ID 1 {name:str:Barack Obama, role:str:political authority, occupation:str:
44th president of United States of America, country:str:United States, political
party:str:Democratic, religion:str:Christianity}
Vertices found: 8
```

Additionally, Oracle Big Data Spatial and Graph provides a set of utilities to help users write their own Lucene text search queries using the query syntax and data type identifiers required by the automatic and manual text indexes. The method `buildSearchTerm(key, value, dtClass)` in `LuceneIndex` creates a query expression of the form *field:query_expr* by adding the data type identifier to the key (or value) and transforming the value into the required string representation based on the given data type and Apache Lucene's data type handling configuration.

The following code fragment uses the `buildSearchTerm` method to produce a query expression *country1:United** (if Lucene's data type handling is enabled), or *country: 1United** (if Lucene's data type handling is disabled) used in the previous examples:

```
String szQueryStrCountry = index.buildSearchTerm("country",
                                   "United*", String.class);
```

To deal with the key and values as individual objects to construct a different Lucene Query like a WildcardQuery using the required syntax, the methods `appendDatatypesSuffixToKey(key, dtClass)` and `appendDatatypesSuffixToValue(value, dtClass)` in LuceneIndex will append the appropriate data type identifiers and transform the value into the required Lucene string representation based on the given data type.

The following code fragment uses the`appendDatatypesSuffixToKey` method to generate the field name required in a Lucene text query. If Lucene’s data type handling is enabled, the string returned will append the String data type identifier as a suffix of the key (country1). In any other case, the retrieved string will be the original key (country).

```
String key = index.appendDatatypesSuffixToKey("country", String.class);
```

The next code fragment uses the `appendDatatypesSuffixToValue` method to generate the query body expression required in a Lucene text query. If Lucene's data type handling is disabled, the string returned will append the String data type identifier as a prefix of the key (1United*). In all other cases, the string returned will be the string representation of the value (United*).

```
String value = index.appendDatatypesSuffixToValue("United*", String.class);
```

`LuceneIndex` also supports generating a Term object using the method `buildSearchTermObject(key, value, dtClass)`. Term objects are commonly used among different type of Lucene Query objects to constrain the fields and values of the documents to be retrieved. The following code fragment shows how to create a Wildcard Query object using the `buildSearchTermObject` method.

```
Term term = index.buildSearchTermObject("country", "United*", String.class);
Query query = new WildcardQuery(term);
```

4.5.3.2 Appending Data Type Identifiers on SolrCloud

For Boolean operations on SolrCloud text indexes, you must append the proper data type identifier as suffix to the key in the query expression. This can be done by executing a `String.concat()` operation to the key. Table 4-2 shows the data type identifiers available for text indexing using SolrCloud (see the Javadoc for `SolrIndex`).

Table 4-2 SolrCloud Data Type Identifiers

Solr Data Type Identifier	Description
TYPE_DT_STRING	String
TYPE_DT_BOOL	Boolean
TYPE_DT_DATE	Date
TYPE_DT_FLOAT	Float
TYPE_DT_DOUBLE	Double
TYPE_DT_INTEGER	Integer
TYPE_DT_SERIALIZABLE	Serializable

The following code fragment creates a manual index on edges using SolrCloud, adds data, and later executes a query over the manual index to get all edges with the key/value pair `collaboratesWith:Beyonce AND country1:United*` using wildcards.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args,
                                                          szGraphName);

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do a parallel data loading
```

```
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create a manual text index using SolrCloud// Specify Index Directory parameters:
configuration name, Solr Server URL, Solr Node set,
// replication factor, zookeeper timeout (secs),
// maximum number of shards per node,
   // number of connections to database, batch size, commit size,
         // write timeout (in secs)
             String configName = "opgconfig";
             String solrServerUrl = "nodea:2181/solr"
             String solrNodeSet = "nodea:8983_solr,nodeb:8983_solr," +
                                  "nodec:8983_solr,noded:8983_solr";

         int zkTimeout = 15;
         int numShards = 4;
         int replicationFactor = 1;
         int maxShardsPerNode = 1;

OracleIndexParameters indexParams =
                 OracleIndexParameters.buildSolr(configName,
                                       solrServerUrl,
                                       solrNodeSet,
                                       zkTimeout,
                                       numShards,
                                       replicationFactor,
                                       maxShardsPerNode,
                                       4,
                                       10000,
                                       500000,
                                       15);
opg.setDefaultIndexParameters(indexParams);

// Create manual indexing on above properties for all vertices
OracleIndex<Edge> index = ((OracleIndex<Edge>) opg.createIndex("myIdx", Edge.class));

Vertex v1 = opg.getVertices("name", "Barack Obama").iterator().next();

Iterator<Edge> edges
                = v1.getEdges(Direction.OUT, "collaborates").iterator();

          while (edges.hasNext()) {
             Edge edge = edges.next();
             Vertex vIn = edge.getVertex(Direction.IN);
             index.put("collaboratesWith", vIn.getProperty("name"), edge);
             index.put("country", vIn.getProperty("country"), edge);
          }

// Wildcard searching is supported using true parameter.
    String key = "country";
    key = key.concat(oracle.pg.text.solr.SolrIndex.TYPE_DT_STRING);

    String queryExpr = "Beyonce AND " + key + ":United*";
    edges = index.get("collaboratesWith", queryExpr, true /**
UseWildcard*/).iterator();
    System.out.println("----- Edges with query: " + queryExpr + " -----");
    countE = 0;
    while (edges.hasNext()) {
      System.out.println(edges.next());
      countE++;
```

```
}
System.out.println("Edges found: "+ countE);
```

The preceding code example might produce output like the following:

```
----- Edges with name Beyonce AND country_str:United* -----
Edge ID 1000 from Vertex ID 1 {country:str:United States, name:str:Barack Obama,
occupation:str:44th president of United States of America, political
party:str:Democratic, religion:str:Christianity, role:str:political authority}
=[collaborates]=> Vertex ID 2 {country:str:United States, music genre:str:pop soul ,
name:str:Beyonce, role:str:singer actress} edgeKV[{weight:flo:1.0}]
Edges found: 1
```

Additionally, Oracle Big Data Spatial and Graph provides a set of utilities to help users write their own SolrCloud text search queries using the query syntax and data type identifiers required by the automatic and manual text indexes. The method `buildSearchTerm(key, value, dtClass)` in `SolrIndex` creates a query expression of the form *field:query_expr* by adding the data type identifier to the key (or value) and transforming the value into the required string representation using the data type formats required by the index

The following code fragment uses the `buildSearchTerm` method to produce a query expression *country1:United** (if Lucene's data type handling is enabled), or *country: 1United** (if Lucene's data type handling is disabled) used in the previous examples:

```
String szQueryStrCountry = index.buildSearchTerm("country",
                                   "United*", String.class);
```

To deal with the key and values as individual objects to construct a different Lucene Query like a WildcardQuery using the required syntax, the methods `appendDatatypesSuffixToKey(key, dtClass)` and `appendDatatypesSuffixToValue(value, dtClass)` in `SolrIndex` will append the appropriate data type identifiers and transform the value into the required SolrCloud string representation based on the given data type.

The following code fragment uses the`appendDatatypesSuffixToKey` method to generate the field name required in a SolrCloud text query. The retrieved string will append the String data type identifier as a suffix of the key (country_str).

```
String key = index.appendDatatypesSuffixToKey("country", String.class);
```

The next code fragment uses the `appendDatatypesSuffixToValue` method to generate the query body expression required in a SolrCloud text query. The string returned will be the string representation of the value (United*).

```
String value = index.appendDatatypesSuffixToValue("United*", String.class);
```

4.5.4 Uploading a Collection's SolrCloud Configuration to Zookeeper

Before using SolrCloud text indexes on Oracle Big Data Spatial and Graph property graphs, you must upload a collection's configuration to Zookeeper. This can be done using the ZkCli tool from one of the SolrCloud cluster nodes.

A predefined collection configuration directory can be found in `dal/opg-solr-config` under the installation home. The following shows an example on how to upload the PropertyGraph configuration directory.

1. Copy dal/opg-solr-config under the installation home into /tmp directory on one of the Solr cluster nodes. For example:

   ```
   scp –r dal/opg-solr-config user@solr-node:/tmp
   ```

2. Execute the following command line like the following example using the ZkCli tool on the same node:

```
$SOLR_HOME/bin/zkcli.sh -zkhost 127.0.0.1:2181/solr -cmd upconfig –confname
opgconfig -confdir /tmp/opg-solr-config
```

4.5.5 Updating Configuration Settings on Text Indexes for Property Graph Data

Oracle's property graph support manages manual and automatic text indexes through integration with Apache Lucene and SolrCloud. At creation time, you must create an `OracleIndexParameters` object specifying the search engine and other configuration settings to be used by the text index. After a text index for property graph is created, these configuration settings cannot be changed. For automatic indexes, all vertex index keys are managed by a single text index, and all edge index keys are managed by a different text index using the configuration specified when the first vertex or edge key is indexed.

If you need to change the configuration settings, you must first disable the current index and create it again using a new `OracleIndexParameters` object. The following code fragment creates two automatic Apache Lucene-based indexes (on vertices and edges) over an existing property graph, disables them, and recreates them to use SolrCloud.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
      args,  szGraphName);

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create an automatic index using Apache Lucene.
// Specify Index Directory parameters (number of directories,
// number of connections to database, batch size, commit size,
// enable datatypes, location)
OracleIndexParameters luceneIndexParams =
     OracleIndexParameters.buildFS(4, 4, 10000, 50000, true,
             "/home/data/text-index ");

// Specify indexed keys
String[] indexedKeys = new String[4];
indexedKeys[0] = "name";
indexedKeys[1] = "role";
indexedKeys[2] = "religion";
indexedKeys[3] = "country";

// Create auto indexing on above properties for all vertices
opg.createKeyIndex(indexedKeys, Vertex.class, luceneIndexParams.getParameters());

// Create auto indexing on weight for all edges
opg.createKeyIndex("weight", Edge.class, luceneIndexParams.getParameters());

// Disable auto indexes to change parameters
opg.getOracleIndexManager().disableVertexAutoIndexer();
opg.getOracleIndexManager().disableEdgeAutoIndexer();
```

```
// Recreate text indexes using SolrCloud
// Specify Index Directory parameters: configuration name, Solr Server URL, Solr
Node set,
// replication factor, zookeeper timeout (secs),
// maximum number of shards per node,
// number of connections to database, batch size, commit size,
// write timeout (in secs)
String configName = "opgconfig";
String solrServerUrl = "nodea:2181/solr"
String solrNodeSet = "nodea:8983_solr,nodeb:8983_solr," +
   "nodec:8983_solr,noded:8983_solr";

int zkTimeout = 15;
int numShards = 4;
int replicationFactor = 1;
int maxShardsPerNode = 1;

OracleIndexParameters solrIndexParams =
OracleIndexParameters.buildSolr(configName,
                                solrServerUrl,
                                solrNodeSet,
                                zkTimeout,
                                numShards,
                                replicationFactor,
                                maxShardsPerNode,
                                4,
                                10000,
                                500000,
                                15);

// Create auto indexing on above properties for all vertices
opg.createKeyIndex(indexedKeys, Vertex.class, solrIndexParams.getParameters());

// Create auto indexing on weight for all edges
opg.createKeyIndex("weight", Edge.class, solrIndexParams.getParameters());
```

4.5.6 Using Parallel Query on Text Indexes for Property Graph Data

Text indexes in Oracle Big Data Spatial and Graph allow executing text queries over millions of vertices and edges by a particular key/value or key/text pair using parallel query execution.

Parallel text querying is an optimized solution taking advantage of the distribution of the data in the index among shards in SolrCloud (or subdirectories in Apache Lucene), so each one is queried using separate index connection. This involves multiple threads and connections to SolrCloud (or Apache Lucene) search engines to increase performance on read operations and retrieve multiple elements from the index. Note that this approach will not rank the matching results based on their score.

Parallel text query will produce an array where each element holds all the vertices (or edges) with an attribute matching the given K/V pair from a shard. The subset of shards queried will be delimited by the given start sub-directory ID and the size of the connections array provided. This way, the subset will consider shards in the range of [start, start - 1 + size of connections array]. Note that an integer ID (in the range of [0, N - 1]) is assigned to all the shards in index with N shards.

Parallel Text Query Using Apache Lucene

You can use parallel text query using Apache Lucene by calling the method `getPartitioned` in `LuceneIndex`, specifying an array of connections to set of subdirectories (`SearcherManager` objects), the key/value pair to search, and the starting subdirectory ID. Each connection needs to be linked to the appropriate subdirectory, as each subdirectory is independent of the rest of the subdirectories in the index.

The following code fragment generates an automatic text index using the Apache Lucene Search engine, and executes a parallel text query. The number of calls to the `getPartitioned` method in the `LuceneIndex` class is controlled by the total number of subdirectories and the number of connections used.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create an automatic index
OracleIndexParameters indexParams
= OracleIndexParameters.buildFS(dop /* number of directories */,
dop /* number of connections
used when indexing */,
10000 /* batch size before commit*/,
500000 /* commit size before Lucene commit*/,
true /* enable datatypes */,
"./lucene-index" /* index location */);

opg.setDefaultIndexParameters(indexParams);

// Create auto indexing on name property for all vertices
System.out.println("Create automatic index on name for vertices");
opg.createKeyIndex("name", Vertex.class);

// Get the SolrIndex object
LuceneIndex<Vertex> index = (LuceneIndex<Vertex>) opg.getAutoIndex(Vertex.class);


long lCount = 0;
for (int split = 0; split < index.getTotalShards();
 split += conns.length) {
// Gets a connection object from subdirectory split to
//(split + conns.length)
for (int idx = 0; idx < conns.length; idx++) {
conns[idx] = index.getOracleSearcherManager(idx + split);
}

// Gets elements from split to split + conns.length
Iterable<Vertex>[] iterAr }
= index.getPartitioned(conns /* connections */,
 "name"/* key */,
 "*" /* value */,
 true /* wildcards */,
 split /* start split ID */);

lCount = countFromIterables(iterAr); /* Consume iterables in parallel */
```

```
// Close the connections to the sub-directories after completed
for (int idx = 0; idx < conns.length; idx++) {
conns[idx].close();
}
}

// Count all vertices
System.out.println("Vertices found using parallel query: " + lCount);
```

Parallel Text Search Using SolrCloud

You can use parallel text query using SolrCloud by calling the method `getPartitioned` in `SolrIndex`, specifying an array of connections to SolrCloud (`CloudSolrServer` objects), the key/value pair to search, and the starting shard ID.

The following code fragment generates an automatic text index using the SolrCloud Search engine and executes a parallel text query. The number of calls to the `getPartitioned` method in the `SolrIndex` class is controlled by the total number of shards in the index and the number of connections used.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

String configName = "opgconfig";
String solrServerUrl = args[4];//"localhost:2181/solr"
String solrNodeSet = args[5]; //"localhost:8983_solr";

int zkTimeout = 15; // zookeeper timeout in seconds
int numShards = Integer.parseInt(args[6]); // number of shards in the index
int replicationFactor = 1; // replication factor
int maxShardsPerNode = 1; // maximum number of shards per node

// Create an automatic index using SolrCloud
OracleIndexParameters indexParams =
 OracleIndexParameters.buildSolr(configName,
 solrServerUrl,
 solrNodeSet,
 zkTimeout /* zookeeper timeout in seconds */,
 numShards /* total number of shards */,
 replicationFactor /* Replication factor */,
 maxShardsPerNode /* maximum number of shardsper node*/,
 4 /* dop used for scan */,
 10000 /* batch size before commit*/,
 500000 /* commit size before SolrCloud commit*/,
 15 /* write timeout in seconds */);

opg.setDefaultIndexParameters(indexParams);

// Create auto indexing on name property for all vertices
System.out.println("Create automatic index on name for vertices");
opg.createKeyIndex("name", Vertex.class);
```

```
// Get the SolrIndex object
SolrIndex<Vertex> index = (SolrIndex<Vertex>) opg.getAutoIndex(Vertex.class);

// Open an array of connections to handle connections to SolrCloud needed for
parallel text search
CloudSolrServer[] conns = new CloudSolrServer[dop];


for (int idx = 0; idx < conns.length; idx++) {
conns[idx] = index.getCloudSolrServer(15 /* write timeout in
secs*/);
}

// Iterate to cover all the shards in the index
long lCount = 0;
for (int split = 0; split < index.getTotalShards();
 split += conns.length) {
// Gets elements from split to split + conns.length
Iterable<Vertex>[] iterAr = index.getPartitioned(conns /* connections */,
 "name"/* key */,
 "*" /* value */,
 true /* wildcards */,
 split /* start split ID */);

lCount = countFromIterables(iterAr); /* Consume iterables in parallel */
}

// Do not close the connections to the subdirectories after completion,
// because those connections are used by the index itself.

// Count results
System.out.println("Vertices found using parallel query: " + lCount);
```

4.5.7 Using Native Query Objects on Text Indexes for Property Graph Data

Using Query objects directly is for advanced users, enabling them to take full advantage of the underlying query capabilities of the text search engine (Apache Lucene or SolrCloud). For example, you can add constraints to text searches, such as adding a boost to the matching scores and adding sorting clauses.

Using text searches with Query objects will produce an Iterable object holding all the vertices (or edges) with an attribute (or set of attributes) matching the text query while satisfying the constraints. This approach will automatically rank the results based on their matching score.

To build the clauses in the query body, you may need to consider the data type used by the key/value pair to be matched, as well as the configuration of the search engine used. For more information about building a search term, see Handling Data Types.

Using Native Query Objects with Apache Lucene

You can use native query objects using Apache Lucene by calling the method `get(Query)` in `LuceneIndex`. You can also use parallel text query with native query objects by calling the method `getPartitioned(SearcherManager[], Query, int)` in `LuceneIndex` specifying an array of connections to a set of subdirectories (`SearcherManager` objects), the Lucene query object, and the starting subdirectory ID. Each connection must be linked to the appropriate subdirectory, because each subdirectory is independent of the rest of the subdirectories in the index.

The following code fragment generates an automatic text index using Apache Lucene Search engine, creates a Lucene Query, and executes a parallel text query. The number of calls to the `getPartitioned` method in the `LuceneIndex` class is controlled by the total number of subdirectories and the number of connections used.

```
import oracle.pg.text.lucene.LuceneIndex;
import org.apache.lucene.search.*;
import org.apache.lucene.index.*;

...

OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

// Create an automatic index
OracleIndexParameters indexParams = OracleIndexParameters.buildFS(dop /* number of
directories */,
dop /* number of connections
used when indexing */,
10000 /* batch size before commit*/,
500000 /* commit size before Lucene commit*/,
true /* enable datatypes */,
"./lucene-index" /* index location */);

opg.setDefaultIndexParameters(indexParams);

// Create auto indexing on name and country properties for all vertices
System.out.println("Create automatic index on name and country for vertices");
String[] indexedKeys = new String[2];
indexedKeys[0]="name";
indexedKeys[1]="country";
opg.createKeyIndex(indexedKeys, Vertex.class);

// Get the LuceneIndex object
LuceneIndex<Vertex> index = (LuceneIndex<Vertex>) opg.getAutoIndex(Vertex.class);

// Search first for Key name with property value Beyon* using only string
//data types
Term term = index.buildSearchTermObject("name", "Beyo*", String.class);
Query queryBey = new WildcardQuery(term);

// Add another condition to query all the vertices whose country is
//"United States"
String key = index.appendDatatypesSuffixToKey("country", String.class);
String value = index.appendDatatypesSuffixToValue("United States", String.class);

Query queryCountry = new PhraseQuery();
StringTokenizer st = new StringTokenizer(value);
while (st.hasMoreTokens()) {
  queryCountry.add(new Term(key, st.nextToken()));
};

//Concatenate queries
```

```
Boolean bQuery = new BooleanQuery();
bQuery.add(queryBey, BooleanClause.Occur.MUST);
bQuery.add(queryCountry, BooleanClause.Occur.MUST);

long lCount = 0;
conns = new SearcherManager[dop];
for (int split = 0; split < index.getTotalShards(); split += conns.length) {
  // Gets a connection object from subdirectory split to
  //(split + conns.length). Skip the cache so we clone the connection and
  // avoid using the connection used by the index.
  for (int idx = 0; idx < conns.length; idx++) {
    conns[idx] = index.getOracleSearcherManager(idx + split,
                                  true /* skip looking in the
cache*/
);
  }

  // Gets elements from split to split + conns.length
  Iterable<Vertex>[] iterAr = index.getPartitioned(conns /* connections */,
                                                    bQuery,
                                                    split /* start split ID */);

  lCount = countFromIterables(iterAr); /* Consume iterables in parallel */

  // Do not close the connections to the sub-directories after completed,
  // as those connections are used by the index itself

}

// Count all vertices
System.out.println("Vertices found using parallel query: " + lCount);
```

Using Native Query Objects withSolrCloud

You can directly use native query objects against SolrCloud by calling the method `get(SolrQuery)` in `SolrIndex`. You can also use parallel text query with native query objects by calling the method `getPartitioned(CloudSolrServer[],SolrQuery,int)` in `SolrIndex` specifying an array of connections to SolrCloud (`CloudSolrServer` objects), the `SolrQuery` object, and the starting shard ID.

The following code fragment generates an automatic text index using the Apache SolrCloud Search engine, creates a `SolrQuery` object, and executes a parallel text query. The number of calls to the `getPartitioned` method in the `SolrIndex` class is controlled by the total number of subdirectories and the number of connections used.

```
import oracle.pg.text.solr.*;
import org.apache.solr.client.solrj.*;

OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
 args, szGraphName);

// Clear existing vertices/edges in the property graph
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
```

```
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);

String configName = "opgconfig";
String solrServerUrl = args[4];//"localhost:2181/solr"
String solrNodeSet = args[5]; //"localhost:8983_solr";

int zkTimeout = 15; // zookeeper timeout in seconds
int numShards = Integer.parseInt(args[6]); // number of shards in the index
int replicationFactor = 1; // replication factor
int maxShardsPerNode = 1; // maximum number of shards per node

// Create an automatic index using SolrCloud
OracleIndexParameters indexParams =
 OracleIndexParameters.buildSolr(configName,
 solrServerUrl,
 solrNodeSet,
 zkTimeout          /* zookeeper timeout in seconds */,
 numShards          /* total number of shards */,
 replicationFactor  /* Replication factor */,
 maxShardsPerNode   /* maximum number of shardsper node*/,
 4                  /* dop used for scan */,
 10000              /* batch size before commit*/,
 500000             /* commit size before SolrCloud commit*/,
 15                 /* write timeout in seconds */
);

opg.setDefaultIndexParameters(indexParams);

// Create auto indexing on name property for all vertices
System.out.println("Create automatic index on name and country for vertices");
String[] indexedKeys = new String[2];
indexedKeys[0]="name";
indexedKeys[1]="country";
opg.createKeyIndex(indexedKeys, Vertex.class);

// Get the SolrIndex object
SolrIndex<Vertex> index = (SolrIndex<Vertex>) opg.getAutoIndex(Vertex.class);

// Search first for Key name with property value Beyon* using only string
//data types
String szQueryStrBey = index.buildSearchTerm("name", "Beyo*", String.class);
String key = index.appendDatatypesSuffixToKey("country", String.class);
String value = index.appendDatatypesSuffixToValue("United States", String.class);

String szQueryStrCountry = key + ":" + value;
Solrquery query = new SolrQuery(szQueryStrBey + " AND " + szQueryStrCountry);

//Query using get operation
index.get(query);

// Open an array of connections to handle connections to SolrCloud needed
// for parallel text search
CloudSolrServer[] conns = new CloudSolrServer[dop];

for (int idx = 0; idx < conns.length; idx++) {
conns[idx] = index.getCloudSolrServer(15 /* write timeout in
secs*/);
}

// Iterate to cover all the shards in the index
long lCount = 0;
```

```
for (int split = 0; split < index.getTotalShards();
 split += conns.length) {
// Gets elements from split to split + conns.length
Iterable<Vertex>[] iterAr = index.getPartitioned(conns /* connections */,
 query,
 split /* start split ID */);

lCount = countFromIterables(iterAr); /* Consume iterables in parallel */
}

// Close the connections to SolCloud after completion
for (int idx = 0; idx < conns.length; idx++) {
  conns[idx].shutdown();
}

// Count results
System.out.println("Vertices found using parallel query: " + lCount);
```

4.6 Querying Property Graph Data

Oracle Big Data Spatial and Graph supports a rich set of graph pattern matching capabilities. It provides a SQL-like declarative language that allows you to express a graph query pattern that consists of vertices and edges, and constraints on the properties of the vertices and edges.

An example property graph query is as follows. It defines a graph pattern inspired by the famous ancient proverb: *The enemy of my enemy is my friend*. In this example, variables `x`, `y`, `z` are used for vertices, and variables `e1`, `e2` are used for edges. There is a constraint on the edge label, and the query returns (projects) the value of the `name` property of vertices `x` and `y`.

```
SELECT x.name, z.name
WHERE
    x -[e1 WITH label = 'feuds']-> y,
    y -[e2 WITH label = 'feuds']-> z
```

You can run the query either in a Groovy shell environment or from Java. For example, to run the preceding query from the Groovy shell for Apache HBase or Oracle NoSQL Database, you can first read the graph from the database into the in-memory analyst, get an in-memory graph, and invoke the `queryPgql` function.

```
// Read graph data from a backend database into memory
// Note that opg is an instance of OraclePropertyGraph class
opg-hbase> G = session.readGraphWithProperties(opg.getConfig());
opg-hbase>

resultSet = G.queryPgql("SELECT x.name, z.name WHERE x -[e1 WITH label = 'feuds']->
y,  y -[e2 WITH label = 'feuds']-> z")
```

To get the type and variable name of the first projected variable in the result set, you can enter the following:

```
opg-hbase> resultElement = resultElements.get(0)
opg-hbase> type = resultElement.getElementType() // STRING
opg-hbase> varName = resultElement.getVarName() // x.name
```

You can also iterate over the result set. For example:

```
opg-hbase> resultSet.getResults().each { \
       // the variable 'it' is implicitly declared to references each PgqlResult
```

```
instance
     }
```

Finally, you can display (print) results. For example, to display the first 10 rows:

```
opg-hbase> resultSet.print(10) // print the first 10 results
```

4.7 Support for Secure Oracle NoSQL Database

Oracle Big Data Spatial and Graph property graph support works with both secure and non-secure Oracle NoSQL Database installations. This topic provides information about how to use property graph functions with a secure Oracle NoSQL Database setup. It assumes that a secure Oracle NoSQL Database is already installed (a process explained in "Performing a Secure Oracle NoSQL Database Installation" in the *Oracle NoSQL Database Security Guide* at `http://docs.oracle.com/cd/NOSQL/html/SecurityGuide/secure_installation.html`).

You must have the correct credentials to access the secure database. Create a user such as the following:

```
kv-> plan create-user -name myusername -admin -wait
```

Grant this user the `readwrite` and `dbaadmin` roles. For example:

```
kv-> plan grant -user myusername -role readwrite -wait
kv-> plan grant -user myusername -role dbadmin -wait
```

When generating the `login_properties.txt` from the file `client.security`, make sure the user name is correct. For example:

```
oracle.kv.auth.username=myusername
```

On Oracle property graph client side, you must have the security-related files and libraries to interact with the secure Oracle NoSQL Database. First, copy these files (or directories) from `KVROOT/security/` to the client side:

```
client.security
client.trust
login.wallet/
login_properties.txt
```

If Oracle Wallet is used to hold passwords that are needed for accessing the secure database, copy these three libraries to the client side and set the class path correctly:

```
oraclepki.jar
osdt_cert.jar
osdt_core.jar
```

After configuring the database and Oracle property graph client side correctly, you can connect to a graph stored in Secure NoSQL Database using either one of the following two approaches.

- Specify the login properties file, using a Java VM setting with the following format:

  ```
  -Doracle.kv.security=/<your-path>/login_properties.txt
  ```

  You can also set this Java VM property for applications deployed into a J2EE container (including in-memory analytics). For example, before starting WebLogic Server, you can set an environment variable in the following format to refer to the login properties configuration file:

```
setenv JAVA_OPTIONS "-Doracle.kv.security=/<your-path>/login_properties.txt"
```

Then you can call `OraclePropertyGraph.getInstance(kconfig, szGraphName)` as usual to create an `OraclePropertyGraph` instance.

- Call `OraclePropertyGraph.getInstance(kconfig, szGraphName, username, password, truStoreFile)`, where `username` and `password` are the correct credentials to access secure Oracle NoSQL Database, and `truStoreFile` is the path to the client side trust store file `client.trust`.

  The following code fragment creates a property graph in a Secure Oracle NoSQL Database, loads the data, and then counts how many vertices and edges in the graph:

```
// This object will handle operations over the property graph
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(kconfig,
szGraphName,
username,
password,
truStoreFile);

// Clear existing vertices/edges in the property graph
opg.clearRepository();
opg.setQueueSize(100); // 100 elements

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
// This object will handle parallel data loading over the property graph
System.out.println("Load data for graph " + szGraphName);
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
// Count all vertices
long countV = 0;
Iterator<Vertex> vertices = opg.getVertices().iterator();
while (vertices.hasNext()) {
vertices.next();
countV++;
}

System.out.println("Vertices found: " + countV);
// Count all edges
long countE = 0;
Iterator<Edge> edges = opg.getEdges().iterator();
while (edges.hasNext()) {
edges.next();
countE++;
}

System.out.println("Edges found: " + countE);
```

4.8 Implementing Security on Graphs Stored in Apache HBase

Kerberos authentication is recommended for Apache HBase to secure property graphs in Oracle Big Data Spatial and Graph.

Oracle's property graph support works with both secure and non-secure Cloudera Hadoop (CDH) cluster installations. This topic provides information about secure Apache HBase installations.

Kerberos authentication is recommended for Apache HBase to secure property graphs in Oracle Big Data Spatial and Graph.

This topic assumes that a secure Apache HBase is already configured with Kerberos, that the client machine has the Kerberos libraries installed and that you have the correct credentials. For detailed information, see "Configuring Kerberos Authentication for HBase" at: `http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_sg_hbase_authentication.html`. For information about how to set up your Kerberos cluster and clients, see the MIT Kerberos Documentation at `http://web.mit.edu/kerberos/krb5-latest/doc/index.html`.

On the client side, you must have a Kerberos credential to interact with the Kerberos-enabled HDFS daemons. Additionally, you need to modify the Kerberos configuration information (located in `krb5.conf`) to include the realm and mappings of hostnames onto Kerberos realms used in the Secure CDH Cluster.

The following code fragment shows the realm and hostname mapping used in a Secure CDH cluster on BDA.COM:

```
[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = yes

[realms]
 EXAMPLE.COM = {
kdc = hostname1.example.com:88
kdc = hostname2.example.com:88
admin_server = hostname1.example.com:749
default_domain = example.com
 }
BDA.COM = {
kdc = hostname1.bda.com:88
kdc = hostname2.bda.com:88
admin_server = hostname1.bda.com:749
default_domain = bda.com
 }

[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .bda.com = BDA.COM
 bda.com = BDA.COM
```

After modifying `krb5.conf`, you can connect to a graph stored in Apache HBase by using a Java Authentication and Authorization Service (JAAS) configuration file to provide your credentials to the application. This provides the same capabilities of the preceding example without having to modify a single line of your code in case you already have an application that uses an insecure Apache HBase installation.

To use property graph support for for HBase with a JAAS configuration, create a file with content in the following form, replacing the `keytab` and `principal` entries with your own information:

```
Client {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
useTicketCache=true
keyTab="/path/to/your/keytab/user.keytab"
```

```
principal="your-user/your.fully.qualified.domain.name@YOUR.REALM";
};
```

The following code fragment shows an example JAAS file with the realm used in a Secure CDH cluster on BDA.COM:

```
Client {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
useTicketCache=true
keyTab="/path/to/keytab/user.keytab"
principal="hbaseuser/hostname1@BDA.COM";
};
```

In order to run your Secure HBase application you must specify the JAAS configuration file you created by using the java.security.auth.login.config flag. You can run your application using a command in the following format:

```
java -Djava.security.auth.login.config=/path/to/your/jaas.conf/ -classpath ./
classes/:../../lib/'*' YourJavaApplication
```

Then, you can call `OraclePropertyGraph.getInstance(conf, hconn, szGraphName)` as usual to create an Oracle property graph.

Another option to use the Oracle Big Data Spatial and Graph property graph support on a secure Apache HBase installation is to use a secure HBase configuration. The following code fragment shows how to obtain a secure HBase configuration using prepareSecureConfig(). This API requires the security authentication setting used in Apache Hadoop and Apache HBase, as well as Kerberos credentials set to authenticate and obtain an authorized ticket.

The following code fragment creates a property graph in a Secure Apache HBase, loads the data, and then counts how many vertices and edges in the graph.

```
String szQuorum= "hostname1,hostname2,hostname3";
String szCliPort = "2181";
String szGraph = "SecureGraph";

String hbaseSecAuth="kerberos";
String hadoopSecAuth="kerberos";
String hmKerberosPrincipal="hbase/_HOST@BDA.COM";
String rsKerberosPrincipal="hbase/_HOST@BDA.COM";
String userPrincipal = "hbase/hostname1@BDA.COM";
String keytab= "/path/to/your/keytab/hbase.keytab";
int dop= 8;

Configuration conf = HBaseConfiguration.create();
conf.set("hbase.zookeeper.quorum", szQuorum);
conf.set("hbase.zookeeper.property.clientPort", szCliPort);

// Prepare the secure configuration providing the credentials in the keytab
conf = OraclePropertyGraph.prepareSecureConfig(conf,
 hbaseSecAuth,
 hadoopSecAuth,
 hmKerberosPrincipal,
 rsKerberosPrincipal,
 userPrincipal,
 keytab);
HConnection hconn = HConnectionManager.createConnection(conf);

OraclePropertyGraph opg=OraclePropertyGraph.getInstance(conf, hconn, szGraph);
```

```
opg.setInitialNumRegions(24);
opg.clearRepository();

String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";

// Do a parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
opg.commit();
```

4.9 Using the Groovy Shell with Property Graph Data

The Oracle Big Data Spatial and Graph property graph support includes a built-in Groovy shell (based on the original Gremlin Groovy shell script). With this command-line shell interface, you can explore the Java APIs.

To start the Groovy shell, go to the `dal/groovy` directory under the installation home (`/opt/oracle/oracle-spatial-graph/property_graph` by default). For example:

```
cd /opt/oracle/oracle-spatial-graph/property_graph/dal/groovy/
```

Included are the scripts `gremlin-opg-nosql.sh` and `gremlin-opg-hbase.sh`, for connecting to an Oracle NoSQL Database and an Apache HBase, respectively.

Note: To run some gremlin traversal examples, you must first do the following import operation:

```
import com.tinkerpop.pipes.util.structures.*;
```

The following example connects to an Oracle NoSQL Database, gets an instance of `OraclePropertyGraph` with graph name `myGraph`, loads some example graph data, and gets the list of vertices and edges.

```
$ ./gremlin-opg-nosql.sh

opg-nosql>
opg-nosql> hhosts = new String[1];
==>null

opg-nosql> hhosts[0] = "bigdatalite:5000";
==>bigdatalite:5000

opg-nosql> cfg =
GraphConfigBuilder.forPropertyGraphNosql().setName("myGraph").setHosts(Arrays.asList(
hhosts)).setStoreName("mystore").addEdgeProperty("lbl", PropertyType.STRING,
"lbl").addEdgeProperty("weight", PropertyType.DOUBLE, "1000000").build();
==>{"db_engine":"NOSQL","loading":{},"format":"pg","name":"myGraph","error_handling":
{},"hosts":["bigdatalite:5000"],"node_props":[],"store_name":"mystore","edge_props":
[{"type":"string","name":"lbl","default":"lbl"},
{"type":"double","name":"weight","default":"1000000"}]}

opg-nosql> opg = OraclePropertyGraph.getInstance(cfg);
==>oraclepropertygraph with name myGraph

opg-nosql> opgdl = OraclePropertyGraphDataLoader.getInstance();
==>oracle.pg.nosql.OraclePropertyGraphDataLoader@576f1cad
```

```
opg-nosql> opgdl.loadData(opg, new FileInputStream("../../data/connections.opv"),
new FileInputStream("../../data/connections.ope"), 1, 1, 0, null);
==>null

opg-nosql> opg.getVertices();
==>Vertex ID 5 {country:str:Italy, name:str:Pope Francis, occupation:str:pope,
religion:str:Catholicism, role:str:Catholic religion authority}
[... other output lines omitted for brevity ...]

opg-nosql>  opg.getEdges();
==>Edge ID 1139 from Vertex ID 64 {country:str:United States, name:str:Jeff Bezos,
occupation:str:business man} =[leads]=> Vertex ID 37 {country:str:United States,
name:str:Amazon, type:str:online retailing} edgeKV[{weight:flo:1.0}]
[... other output lines omitted for brevity ...]
```

The following example customizes several configuration parameters for in-memory analytics. It connects to an Apache HBase, gets an instance of `OraclePropertyGraph` with graph name `myGraph`, loads some example graph data, gets the list of vertices and edges, gets an in-memory analyst, and execute one of the built-in analytics, triangle counting.

```
$ ./gremlin-opg-hbase.sh
opg-hbase>
opg-hbase> dop=2;   // degree of parallelism
==>2
opg-hbase> confPgx = new HashMap<PgxConfig.Field, Object>();
opg-hbase> confPgx.put(PgxConfig.Field.ENABLE_GM_COMPILER, false);
==>null
opg-hbase> confPgx.put(PgxConfig.Field.NUM_WORKERS_IO, dop + 2);
==>null
opg-hbase> confPgx.put(PgxConfig.Field.NUM_WORKERS_ANALYSIS, 3);
==>null
opg-hbase> confPgx.put(PgxConfig.Field.NUM_WORKERS_FAST_TRACK_ANALYSIS, 2);
==>null
opg-hbase> confPgx.put(PgxConfig.Field.SESSION_TASK_TIMEOUT_SECS, 0);
==>null
opg-hbase> confPgx.put(PgxConfig.Field.SESSION_IDLE_TIMEOUT_SECS, 0);
==>null
opg-hbase> instance = Pgx.getInstance()
==>null
opg-hbase> instance.startEngine(confPgx)
==>null

opg-hbase> cfg =
GraphConfigBuilder.forPropertyGraphHbase() .setName("myGraph") .setZkQuorum("bigdatal
ite") .setZkClientPort(iClientPort) .setZkSessionTimeout(60000) .setMaxNumConnection
s(dop) .setLoadEdgeLabel(true) .setSplitsPerRegion(1) .addEdgeProperty("lbl",
PropertyType.STRING, "lbl") .addEdgeProperty("weight", PropertyType.DOUBLE,
"1000000") .build();
==>{"splits_per_region":1,"max_num_connections":2,"node_props":
[],"format":"pg","load_edge_label":true,"name":"myGraph","zk_client_port":
2181,"zk_quorum":"bigdatalite","edge_props":
[{"type":"string","default":"lbl","name":"lbl"},
{"type":"double","default":"1000000","name":"weight"}],"loading":{},"error_handling":
{},"zk_session_timeout":60000,"db_engine":"HBASE"}

opg-hbase> opg = OraclePropertyGraph.getInstance(cfg);
==>oraclepropertygraph with name myGraph
```

```
opg-hbase> opgdl = OraclePropertyGraphDataLoader.getInstance();
==>oracle.pg.hbase.OraclePropertyGraphDataLoader@3451289b

opg-hbase> opgdl.loadData(opg, "../../data/connections.opv", "../../data/
connections.ope", 1, 1, 0, null);
==>null

opg-hbase> opg.getVertices();
==>Vertex ID 78 {country:str:United States, name:str:Hosain Rahman,
occupation:str:CEO of Jawbone}
...

opg-hbase> opg.getEdges();
==>Edge ID 1139 from Vertex ID 64 {country:str:United States, name:str:Jeff Bezos,
occupation:str:business man} =[leads]=> Vertex ID 37 {country:str:United States,
name:str:Amazon, type:str:online retailing} edgeKV[{weight:flo:1.0}]
[... other output lines omitted for brevity ...]

opg-hbase> session = Pgx.createSession("session-id-1");
opg-hbase> g = session.readGraphWithProperties(cfg);
opg-hbase> analyst = session.createAnalyst();

opg-hbase>  triangles = analyst.countTriangles(false).get();
==>22
```

For detailed information about the Java APIs, see the Javadoc reference information in `doc/dal/` and `doc/pgx/` under the installation home (`/opt/oracle/oracle-spatial-graph/property_graph/` by default).

4.10 Exploring the Sample Programs

The software installation includes a directory of example programs, which you can use to learn about creating and manipulating property graphs.

- About the Sample Programs
- Compiling and Running the Sample Programs
- About the Example Output
- Example: Creating a Property Graph
- Example: Dropping a Property Graph
- Examples: Adding and Dropping Vertices and Edges

4.10.1 About the Sample Programs

The sample programs are distributed in an installation subdirectory named `examples/dal`. The examples are replicated for HBase and Oracle NoSQL Database, so that you can use the set of programs corresponding to your choice of backend database. Table 4-3 describes the some of the programs.

Table 4-3 Property Graph Program Examples (Selected)

Table 4-3 (Cont.) Property Graph Program Examples (Selected)

Program Name	Description
ExampleNoSQL1 ExampleHBase1	Creates a minimal property graph consisting of one vertex, sets properties with various data types on the vertex, and queries the database for the saved graph description.
ExampleNoSQL2 ExampleHBase2	Creates the same minimal property graph as Example1, and then deletes it.
ExampleNoSQL3 ExampleHBase3	Creates a graph with multiple vertices and edges. Deletes some vertices and edges explicitly, and other implicitly by deleting other, required objects. This example queries the database repeatedly to show the current list of objects.

4.10.2 Compiling and Running the Sample Programs

To compile and run the Java source files:

1. Change to the examples directory:

   ```
   cd examples/dal
   ```

2. Use the Java compiler:

   ```
   javac -classpath ../../lib/'*' filename.java
   ```

   For example: `javac -classpath ../../lib/'*' ExampleNoSQL1.java`

3. Execute the compiled code:

   ```
   java -classpath ../../lib/'*':./ filename args
   ```

   The arguments depend on whether you are using Oracle NoSQL Database or Apache HBase to store the graph. The values are passed to `OraclePropertyGraph.getInstance`.

Apache HBase Argument Descriptions

Provide these arguments when using the HBase examples:

1. *quorum*: A comma-delimited list of names identifying the nodes where HBase runs, such as `"node01.example.com, node02.example.com, node03.example.com"`.
2. *client_port*: The HBase client port number, such as `"2181"`.
3. *graph_name*: The name of the graph, such as `"customer_graph"`.

Oracle NoSQL Database Argument Descriptions

Provide these arguments when using the NoSQL examples:

1. *host_name*: The cluster name and port number for Oracle NoSQL Database registration, such as `"cluster02:5000"`.
2. *store_name*: The name of the key-value store, such as `"kvstore"`
3. *graph_name*: The name of the graph, such as `"customer_graph"`.

4.10.3 About the Example Output

The example programs use `System.out.println` to retrieve the property graph descriptions from the database where it is stored, either Oracle NoSQL Database or Apache HBase. The key name, data type, and value are delimited by colons. For example, `weight:flo:30.0` indicates that the key name is `weight`, the data type is `float`, and the value is `30.0`.

Table 4-4 identifies the data type abbreviations used in the output.

Table 4-4 Property Graph Data Type Abbreviations

Abbreviation	Data Type
bol	Boolean
dat	date
dbl	double
flo	float
int	integer
ser	serializable
str	string

4.10.4 Example: Creating a Property Graph

`ExampleNoSQL1` and `ExampleHBase1` create a minimal property graph consisting of one vertex. The code fragment in Example 4-5 creates a vertex named `v1` and sets properties with various data types. It then queries the database for the saved graph description.

Example 4-5 Creating a Property Graph

```
// Create a property graph instance named opg
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args);

// Clear all vertices and edges from opg
    opg.clearRepository();

// Create vertex v1 and assign it properties as key-value pairs
    Vertex v1 = opg.addVertex(1l);
    v1.setProperty("age",  Integer.valueOf(18));
    v1.setProperty("name", "Name");
    v1.setProperty("weight", Float.valueOf(30.0f));
    v1.setProperty("height", Double.valueOf(1.70d));
    v1.setProperty("female", Boolean.TRUE);

// Save the graph in the database
    opg.commit();

// Display the stored vertex description
System.out.println("Fetch 1 vertex: " + opg.getVertices().iterator().next());

// Close the graph instance
    opg.shutdown();
```

The `OraclePropertyGraph.getInstance` arguments (*args*) depend on whether you are using Oracle NoSQL Database or Apache HBase to store the graph. See "Compiling and Running the Sample Programs".

`System.out.println` displays the following output:

```
Fetch 1 vertex: Vertex ID 1 {age:int:18, name:str:Name, weight:flo:30.0, height:dbl:
1.7, female:bol:true}
```

See the property graph support Javadoc (`/opt/oracle/oracle-spatial-graph/property_graph/doc/pgx` by default) for the following:

```
OraclePropertyGraph.addVertex
OraclePropertyGraph.clearRepository
OraclePropertyGraph.getInstance
OraclePropertyGraph.getVertices
OraclePropertyGraph.shutdown
Vertex.setProperty
```

4.10.5 Example: Dropping a Property Graph

ExampleNoSQL2 and ExampleHBase2 create a graph like the one in "Example: Creating a Property Graph", and then drop it from the database.

The code fragment in Example 4-6 drops the graph. See "Compiling and Running the Sample Programs" for descriptions of the `OraclePropertyGraphUtils.dropPropertyGraph` arguments.

Example 4-6 Dropping a Property Graph

```
// Drop the property graph from the database
OraclePropertyGraphUtils.dropPropertyGraph(args);

// Display confirmation that the graph was dropped
System.out.println("Graph " + graph_name + " dropped. ");
```

`System.out.println` displays the following output:

```
Graph graph_name dropped.
```

See the Javadoc for `OraclePropertyGraphUtils.dropPropertyGraph`.

4.10.6 Examples: Adding and Dropping Vertices and Edges

ExampleNoSQL3 and ExampleHBase3 add and drop both vertices and edges.

Example 4-7 Creating the Vertices

The code fragment in Example 4-7 creates three vertices. It is a simple variation of Example 4-5.

```
// Create a property graph instance named opg
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args);

// Clear all vertices and edges from opg
    opg.clearRepository();

// Add vertices a, b, and c
    Vertex a = opg.addVertex(1l);
    a.setProperty("name", "Alice");
    a.setProperty("age", 31);

    Vertex b = opg.addVertex(2l);
```

```
    b.setProperty("name", "Bob");
    b.setProperty("age", 27);

    Vertex c = opg.addVertex(3l);
    c.setProperty("name", "Chris");
    c.setProperty("age", 33);
```

Example 4-8 Creating the Edges

The code fragment in Example 4-8 uses vertices a, b, and c to create the edges.

```
// Add edges e1, e2, and e3
    Edge e1 = opg.addEdge(1l, a, b, "knows");
    e1.setProperty("type", "partners");

    Edge e2 = opg.addEdge(2l, a, c, "knows");
    e2.setProperty("type", "friends");

    Edge e3 = opg.addEdge(3l, b, c, "knows");
    e3.setProperty("type", "colleagues");
```

Example 4-9 Deleting Edges and Vertices

The code fragment in Example 4-9 explicitly deletes edge `e3` and vertex `b`. It implicitly deletes edge `e1`, which was connected to vertex `b`.

```
 // Remove edge e3
    opg.removeEdge(e3);

// Remove vertex b and all related edges
    opg.removeVertex(b);
```

Example 4-10 Querying for Vertices and Edges

This example queries the database to show when objects are added and dropped. The code fragment in Example 4-10 shows the method used.

```
// Print all vertices
    vertices = opg.getVertices().iterator();
    System.out.println("----- Vertices ----");
    vCount = 0;
    while (vertices.hasNext()) {
      System.out.println(vertices.next());
      vCount++;
    }
    System.out.println("Vertices found: " + vCount);

    // Print all edges
    edges = opg.getEdges().iterator();
    System.out.println("----- Edges ----");
    eCount = 0;
    while (edges.hasNext()) {
      System.out.println(edges.next());
      eCount++;
    }
    System.out.println("Edges found: " + eCount);
```

The examples in this topic may produce output like the following:

```
----- Vertices ----
Vertex ID 3 {name:str:Chris, age:int:33}
Vertex ID 1 {name:str:Alice, age:int:31}
Vertex ID 2 {name:str:Bob, age:int:27}
```

```
Vertices found: 3
----- Edges ----
Edge ID 2 from Vertex ID 1 {name:str:Alice, age:int:31} =[knows]=> Vertex ID 3
{name:str:Chris, age:int:33} edgeKV[{type:str:friends}]
Edge ID 3 from Vertex ID 2 {name:str:Bob, age:int:27} =[knows]=> Vertex ID 3
{name:str:Chris, age:int:33} edgeKV[{type:str:colleagues}]
Edge ID 1 from Vertex ID 1 {name:str:Alice, age:int:31} =[knows]=> Vertex ID 2
{name:str:Bob, age:int:27} edgeKV[{type:str:partners}]
Edges found: 3
 Remove edge Edge ID 3 from Vertex ID 2 {name:str:Bob, age:int:27} =[knows]=> Vertex
ID 3 {name:str:Chris, age:int:33} edgeKV[{type:str:colleagues}]
----- Vertices ----
Vertex ID 1 {name:str:Alice, age:int:31}
Vertex ID 2 {name:str:Bob, age:int:27}
Vertex ID 3 {name:str:Chris, age:int:33}
Vertices found: 3
----- Edges ----
Edge ID 2 from Vertex ID 1 {name:str:Alice, age:int:31} =[knows]=> Vertex ID 3
{name:str:Chris, age:int:33} edgeKV[{type:str:friends}]
Edge ID 1 from Vertex ID 1 {name:str:Alice, age:int:31} =[knows]=> Vertex ID 2
{name:str:Bob, age:int:27} edgeKV[{type:str:partners}]
Edges found: 2
Remove vertex Vertex ID 2 {name:str:Bob, age:int:27}
----- Vertices ----
Vertex ID 1 {name:str:Alice, age:int:31}
Vertex ID 3 {name:str:Chris, age:int:33}
Vertices found: 2
----- Edges ----
Edge ID 2 from Vertex ID 1 {name:str:Alice, age:int:31} =[knows]=> Vertex ID 3
{name:str:Chris, age:int:33} edgeKV[{type:str:friends}]
Edges found: 1
```

4.11 Oracle Flat File Format Definition

A property graph can be defined in two flat files, specifically description files for the vertices and edges.

- About the Property Graph Description Files
- Vertex File
- Edge File
- Encoding Special Characters
- Example Property Graph in Oracle Flat File Format

4.11.1 About the Property Graph Description Files

A pair of files describe a property graph:

- **Vertex file**: Describes the vertices of the property graph. This file has an `.opv` file name extension.
- **Edge file**: Describes the edges of the property graph. This file has an `.ope` file name extension.

It is recommended that these two files share the same base name. For example, `simple.opv` and `simple.ope` define a property graph.

4.11.2 Vertex File

Each line in a vertex file is a record that describes a vertex of the property graph. A record can describe one key-value property of a vertex, thus multiple records/lines are used to describe a vertex with multiple properties.

A record contains six fields separated by commas. Each record must contain five commas to delimit all fields, whether or not they have values:

vertex_ID, key_name, value_type, value, value, value

Table 4-5 describes the fields composing a vertex file record.

Table 4-5 Vertex File Record Format

Field Number	Name	Description
1	*vertex_ID*	An integer that uniquely identifies the vertex
2	*key_name*	The name of the key in the key-value pair If the vertex has no properties, then enter a space (`%20`). This example describes vertex 1 with no properties: `1,%20,,,,`
3	*value_type*	An integer that represents the data type of the value in the key-value pair: `1` String `2` Integer `3` Float `4` Double `5` Date `6` Boolean `7` Long integer `8` Short integer `9` Byte `10` Char `101` Serializable Java object
4	*value*	The encoded, nonnull value of *key_name* when it is neither numeric nor date
5	*value*	The encoded, nonnull value of *key_name* when it is numeric

6	*value*	The encoded, nonnull value of *key_name* when it is a date
		Use the Java `SimpleDateFormat` class to identify the format of the date. This example describes the date format of `2015-03-26T00:00:00.000-05:00`:
		`SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSXXX");`
		`encode(sdf.format((java.util.Date) value));`

Required Grouping of Vertices: A vertex can have multiple properties, and the vertex file includes a record (represented by a single line of text in the flat file) for each combination of a vertex ID and a property for that vertex. In the vertex file, all records for each vertex must be grouped together (that is, not have any intervening records for other vertices. You can accomplish this any way you want, but a convenient way is to sort the vertex file records in ascending (or descending) order by vertex ID. (Note, however, a vertex file is not required to have all records sorted by vertex ID; this is merely one way to achieve the grouping requirement.)

4.11.3 Edge File

Each line in an edge file is a record that describes an edge of the property graph. A record can describe one key-value property of an edge, thus multiple records are used to describe an edge with multiple properties.

A record contains nine fields separated by commas. Each record must contain eight commas to delimit all fields, whether or not they have values:

edge_ID, source_vertex_ID, destination_vertex_ID, edge_label, key_name, value_type, value, value, value

Table 4-6 describes the fields composing an edge file record.

Table 4-6 Edge File Record Format

Field Number	Name	Description
1	*edge_ID*	An integer that uniquely identifies the edge
2	*source_vertex_ID*	The *vertex_ID* of the outgoing tail of the edge.
3	*destination_vertex_ID*	The *vertex_ID* of the incoming head of the edge.
4	*edge_label*	The encoded label of the edge, which describes the relationship between the two vertices
5	*key_name*	The encoded name of the key in a key-value pair If the edge has no properties, then enter a space (`%20`). This example describes edge 100 with no properties: `100,1,2,likes,%20,,,,`
6	*value_type*	An integer that represents the data type of the value in the key-value pair: `1` String `2` Integer `3` Float `4` Double `5` Date `6` Boolean `7` Long integer `8` Short integer `9` Byte `10` Char `101` Serializable Java object

Table 4-6 (Cont.) Edge File Record Format

Field Number	Name	Description
7	*value*	The encoded, nonnull value of *key_name* when it is neither numeric nor date
8	*value*	The encoded, nonnull value of *key_name* when it is numeric
9	*value*	The encoded, nonnull value of *key_name* when it is a date Use the Java `SimpleDateFormat` class to identify the format of the date. This example describes the date format of `2015-03-26Th00:00:00.000-05:00`: `SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'Th'HH:mm:ss.SSSXXX"); encode(sdf.format((java.util.Date) value));`

Required Grouping of Edges: An edge can have multiple properties, and the edge file includes a record (represented by a single line of text in the flat file) for each combination of an edge ID and a property for that edge. In the edge file, all records for each edge must be grouped together (that is, not have any intervening records for other edges. You can accomplish this any way you want, but a convenient way is to sort the edge file records in ascending (or descending) order by edge ID. (Note, however, an edge file is not required to have all records sorted by edge ID; this is merely one way to achieve the grouping requirement.)

4.11.4 Encoding Special Characters

The encoding is UTF-8 for the vertex and edge files. Table 4-7 lists the special characters that must be encoded as strings when they appear in a vertex or edge property (key-value pair) or an edge label. No other characters require encoding.

Table 4-7 Special Character Codes in the Oracle Flat File Format

Special Character	String Encoding	Description
`%`	`%25`	Percent
`\t`	`%09`	Tab
	`%20`	Space
`\n`	`%0A`	New line
`\r`	`%0D`	Return
`,`	`%2C`	Comma

4.11.5 Example Property Graph in Oracle Flat File Format

An example property graph in Oracle flat file format is as follows. In this example, there are two vertices (John and Mary), and a single edge denoting that John is a friend of Mary.

```
%cat simple.opv
1,age,2,,10,
1,name,1,John,,
2,name,1,Mary,,
2,hobby,1,soccer,,

%cat simple.ope
100,1,2,friendOf,%20,,,,
```

4.12 Example Python User Interface

The Oracle Big Data Spatial and Graph support for property graphs includes an example Python user interface. It can invoke a set of example Python scripts and modules that perform a variety of property graph operations.

Instructions for installing the example Python user interface are in the `/property_graph/examples/pyopg/README` file under the installation home (`/opt/oracle/oracle-spatial-graph` by default).

The example Python scripts in `/property_graph/examples/pyopg/` can used with Oracle Spatial and Graph Property Graph, and you may want to change and enhance them (or copies of them) to suit your needs.

To invoke the user interface to run the examples, use the script `pyopg.sh`.

The examples include the following:

- Example 1: Connect to an Oracle NoSQL Database and perform a simple check of number of vertices and edges. To run it:

  ```
  cd /opt/oracle/oracle-spatial-graph/property_graph/examples/pyopg
  ./pyopg.sh

  connectONDB("mygraph", "kvstore", "localhost:5000")
  print "vertices", countV()
  print "edges", countE()
  ```

  In the preceding example, `mygraph` is the name of the graph stored in the Oracle NoSQL Database, `kvstore` and `localhost:5000` are the connection information to access the Oracle NoSQL Database. They must be customized for your environment.

- Example 2: Connect to an Apache HBase and perform a simple check of number of vertices and edges. To run it:

  ```
  cd /opt/oracle/oracle-spatial-graph/property_graph/examples/pyopg
  ./pyopg.sh

  connectHBase("mygraph", "localhost", "2181")
  print "vertices", countV()
  print "edges", countE()
  ```

In the preceding example, `mygraph` is the name of the graph stored in the Apache HBase, and `localhost` and `2181` are the connection information to access the Apache HBase. They must be customized for your environment.

- Example 3: Connect to an Oracle NoSQL Database and run a few analytical functions. To run it:

```
cd /opt/oracle/oracle-spatial-graph/property_graph/examples/pyopg
./pyopg.sh

connectONDB("mygraph", "kvstore", "localhost:5000")
print "vertices", countV()
print "edges", countE()

import pprint

analyzer = analyst()
print "# triangles in the graph", analyzer.countTriangles()

graph_communities = [{"commid":i.getName(),"size":i.size()} for i in
analyzer.communities().iterator()]

import pandas as pd
import numpy as np

community_frame = pd.DataFrame(graph_communities)
community_frame[:5]

import matplotlib as mpl
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(16,12));
community_frame["size"].plot(kind="bar", title="Communities and Sizes")
ax.set_xticklabels(community_frame.index);
plt.show()
```

The preceding example connects to an Oracle NoSQL Database, prints basic information about the vertices and edges, get an in memory analyst, computes the number of triangles, performs community detection, and finally plots out in a bar chart communities and their sizes.

- Example 4: Connect to an Apache HBase and run a few analytical functions. To run it:

```
cd /opt/oracle/oracle-spatial-graph/property_graph/examples/pyopg
./pyopg.sh

connectHBase("mygraph", "localhost", "2181")
print "vertices", countV()
print "edges", countE()

import pprint

analyzer = analyst()
print "# triangles in the graph", analyzer.countTriangles()

graph_communities = [{"commid":i.getName(),"size":i.size()} for i in
analyzer.communities().iterator()]
import pandas as pd
import numpy as np
community_frame = pd.DataFrame(graph_communities)
```

```
community_frame[:5]

import matplotlib as mpl
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(16,12));
community_frame["size"].plot(kind="bar", title="Communities and Sizes")
ax.set_xticklabels(community_frame.index);
plt.show()
```

The preceding example connects to an Apache HBase, prints basic information about the vertices and edges, gets an in-memory analyst, computes the number of triangles, performs community detection, and finally plots out in a bar chart communities and their sizes.

For detailed information about this example Python interface, see the following directory under the installation home:

`property_graph/examples/pyopg/doc/`

5

Using the In-Memory Analyst

The in-memory analyst feature of Oracle Spatial and Graph supports a set of analytical functions.

This chapter provides examples using the in-memory analyst (also referred to as Property Graph In-Memory Analytics, and often abbreviated as PGX in the Javadoc, command line, path descriptions, error messages, and examples). It contains the following major topics:

- Reading a Graph into Memory
- Reading Custom Graph Data
- Storing Graph Data on Disk
- Executing Built-in Algorithms
- Creating Subgraphs
- Deploying to Jetty
- Deploying to Apache Tomcat
- Deploying to Oracle WebLogic Server
- Connecting to the In-Memory Analyst Server
- Using the In-Memory Analyst in Distributed Mode
- Reading and Storing Data in HDFS
- Running the In-Memory Analyst as a YARN Application

5.1 Reading a Graph into Memory

This topic provides an example of reading graph interactively into memory using the shell interface. These are the major steps:

1. Connecting to an In-Memory Analyst Server Instance
2. Using the Shell Help (as needed)
3. Providing Graph Metadata in a Configuration File
4. Reading Graph Data into Memory

5.1.1 Connecting to an In-Memory Analyst Server Instance

To start the in-memory analyst shell:

1. Open a terminal session on the system where property graph support is installed.
2. Either start a local (embedded) in-memory analyst instance or connect to a remote in-memory analyst instance

   - Java example of starting a local (embedded) instance:

     ```
     import java.util.Map;
     import java.util.HashMap;
     import oracle.pgx.api.*;
     import oracle.pgx.config.PgxConfig.Field;

     String url = Pgx.EMBEDDED_URL; // local JVM
     ServerInstance instance = Pgx.getInstance(url);
     instance.startEngine(); // will use default configuration
     PgxSession session = instance.createSession("test");
     ```

   - Java example of connecting to a remote instance:

     ```
     import java.util.Map;
     import java.util.HashMap;
     import oracle.pgx.api.*;
     import oracle.pgx.config.PgxConfig.Field;

     String url = "http://my-server.com:8080/pgx" // replace with base URL of your
     setup
     ServerInstance instance = Pgx.getInstance(url);
     PgxSession session = instance.createSession("test");
     ```

3. In the shell, enter the following commands, but select only one of the commands to start or connect to the desired type of instance:

   ```
   cd $PGX_HOME
   ./bin/pgx --help
   ./bin/pgx --version

   # start embedded shell
   ./bin/pgx

   # start remote shell
   ./bin/pgx --base_url http://my-server.com:8080/pgx
   ```

   For the embedded shell, the output should be similar to the following:

   ```
   10:43:46,666 [main] INFO Ctrl$2 - >>> PGX engine running.
   pgx>
   ```

4. Optionally, show the predefined variables:

   ```
   pgx> instance
   ==> PGX Server Instance running on embedded mode
   pgx> session
   ==> PGX session pgxShell registered at PGX Server Instance running on embedded
   mode
   pgx> analyst
   ==> Analyst for PGX session pgxShell registered at PGX Server Instance running on
   embedded mode
   pgx>
   ```

   Examples in some other topics assume that the instance and session variables have been set as shown here.

If the in-memory analyst software is installed correctly, you will see an engine-running log message and the in-memory analyst shell prompt (`pgx>`):

The variables `instance`, `session`, and `analyst` are ready to use.

In the preceding example in this topic, the shell started a local instance because the `pgx` command did not specify a remote URL.

5.1.2 Using the Shell Help

The in-memory analyst shell provides a help system, which you access using the `:help` command.

5.1.3 Providing Graph Metadata in a Configuration File

An example graph is included in the installation directory, under `/opt/oracle/oracle-spatial-graph/property_graph/examples/pgx/graphs/`. It uses a configuration file that describes how the in-memory analyst reads the graph.

```
pgx> cat /opt/oracle/oracle-spatial-graph/property_graph/examples/pgx/graphs/
sample.adj.json
===> {
  "uri": "sample.adj",
  "format": "adj_list",
  "node_props": [{
    "name": "prop",
    "type": "integer"
  }],
  "edge_props": [{
    "name": "cost",
    "type": "double"
  }],
  "separator": " "
}
```

The `uri` field provides the location of the graph data. This path resolves relative to the parent directory of the configuration file. When the in-memory analyst loads the graph, it searches the `examples/graphs` directory for a file named `sample.adj`.

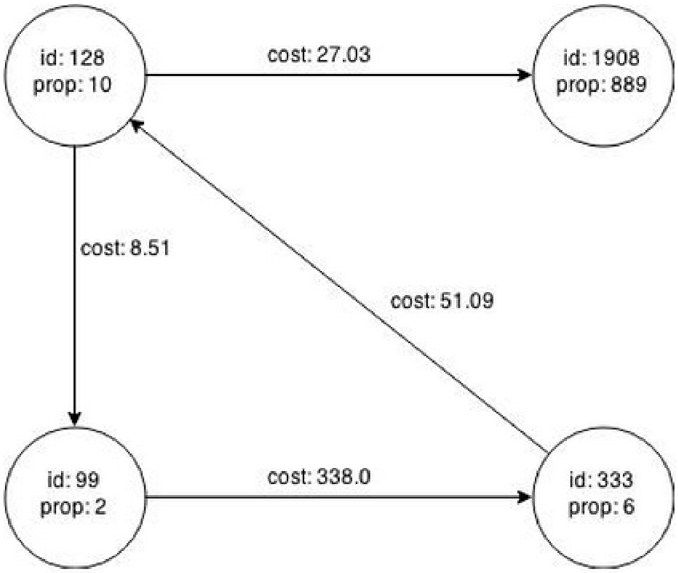
The other fields indicate that the graph data is provided in adjacency list format, and consists of one node property of type `integer` and one edge property of type `double`.

This is the graph data in adjacency list format:

```
pgx> cat /opt/oracle/oracle-spatial-graph/property_graph/examples/pgx/graphs/
sample.adj
===> 128 10 1908 27.03 99 8.51
99 2 333 338.0
1908 889
333 6 128 51.09
```

Figure 5-1 shows a property graph created from the data:

Figure 5-1 Property Graph Rendered by sample.adj Data



5.1.4 Reading Graph Data into Memory

To read a graph into memory, you must pass the following information:

- The path to the graph configuration file that specifies the graph metadata
- A unique alphanumeric name that you can use to reference the graph

  An error results if you previously loaded a different graph with the same name.

Example: Using the Shell to Read a Graph

```
pgx> graph = session.readGraphWithProperties("/opt/oracle/oracle-spatial-graph/
property_graph/examples/pgx/graphs/sample.adj.json", "sample");
==> PGX Graph named sample bound to PGX session pgxShell ...
pgx> graph.getNumVertices()
==> 4
```

Example: Using Java to Read a Graph

```
import oracle.pgx.api.*;

PgxGraph graph = session.readGraphWithProperties("/opt/oracle/oracle-spatial-graph/
property_graph/examples/pgx/graphs/sample.adj.json");
```

The following topics contain additional examples of reading a property graph into memory:

- Read a Graph Stored in Apache HBase into Memory
- Read a Graph Stored in Oracle NoSQL Database into Memory
- Read a Graph Stored in the Local File System into Memory

5.1.4.1 Read a Graph Stored in Apache HBase into Memory

To read a property graph stored in Apache HBase, you can create a JSON based configuration file as follows. Note that the quorum, client port, graph name, and other information must be customized for your own setup.

```
% cat /tmp/my_graph_hbase.json
{
  "format": "pg",
  "db_engine": "hbase",
  "zk_quorum": "scaj31bda07,scaj31bda08,scaj31bda09",
  "zk_client_port": 2181,
  "name": "connections",
  "node_props": [{
    "name": "country",
    "type": "string"
  }],
  "load_edge_label": true,
  "edge_props": [{
    "name": "label",
    "type": "string"
  }, {
    "name": "weight",
    "type": "float"
  }]
}
EOF
```

With the following command, the property graph `connections` will be read into memory:

```
pgx> session.readGraphWithProperties("/tmp/my_graph_hbase.json", "connections")
==> PGX Graph named connections ...
```

Note that when dealing with a large graph, it may become necessary to tune parameters like number of IO workers, number of workers for analysis, task timeout, and others. You can find and change those parameters in the following directory (assume the installation home is `/opt/oracle/oracle-spatial-graph`).

```
/opt/oracle/oracle-spatial-graph/property_graph/pgx/conf
```

5.1.4.2 Read a Graph Stored in Oracle NoSQL Database into Memory

To read a property graph stored in Oracle NoSQL Database, you can create a JSON based configuration file as follows. Note that the hosts, store name, graph name, and other information must be customized for your own setup.

```
% cat /tmp/my_graph_nosql.json
{
  "format": "pg",
  "db_engine": "nosql",
  "hosts": [
    "zathras01:5000"
  ],
  "store_name": "kvstore",
  "name": "connections",
  "node_props": [{
    "name": "country",
    "type": "string"
  }],
  "load_edge_label": true,
  "edge_props": [{
    "name": "label",
    "type": "string"
  }, {
    "name": "weight",
    "type": "float"
```

```
  }]
}
```

Then, read the configuration file into memory. The following example snippet read the file into memory, generates an undirected graph (named `U`) from the original data, and counts the number of triangles.

```
pgx> g = session.readGraphWithProperties("/tmp/my_graph_nosql.json", "connections")
pgx> analyst.countTriangles(g, false)
==> 8
```

5.1.4.3 Read a Graph Stored in the Local File System into Memory

The following command uses the configuration file from “Providing Graph Metadata in a Configuration File” and the name `my-graph`:

```
pgx> g = session.readGraphWithProperties("/opt/oracle/oracle-spatial-graph/
property_graph/examples/pgx/graphs/sample.adj.json", "my-graph")
```

5.2 Reading Custom Graph Data

You can read your own custom graph data. This example creates a graph, alters it, and shows how to read it properly. This graph uses the adjacency list format, but the in-memory analyst supports several graph formats.

The main steps are:

1. Creating a Simple Graph File
2. Adding a Vertex Property
3. Using Strings as Vertex Identifiers
4. Adding an Edge Property

5.2.1 Creating a Simple Graph File

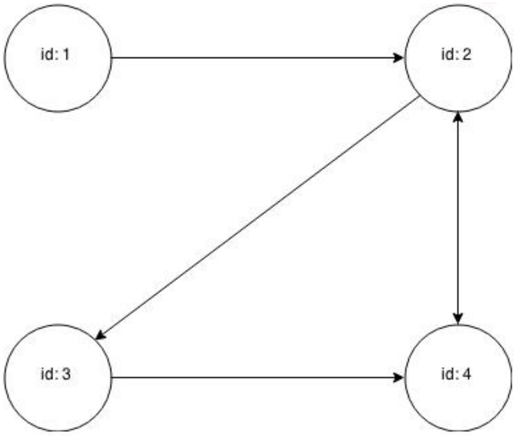
This example creates a small, simple graph in adjacency list format with no vertex or edge properties. Each line contains the vertex (node) ID, followed by the vertex IDs to which iits outgoing edges point:

```
1 2
2 3 4
3 4
4 2
```

In this list, a single space separates the individual tokens. The in-memory analyst supports other separators, which you can specify in the graph configuration file.

Figure 5-2 shows the data rendered as a property graph with 4 vertices and 5 edges. The edge from vertex 2 to vertex 4 points in both directions.

Figure 5-2 Simple Custom Property Graph



Reading a graph into the in-memory analyst requires a graph configuration. You can provide the graph configuration using either of these methods:

- Write the configuration settings in JSON format into a file
- Using a Java `GraphConfigBuilder` object.

This example shows both methods.

JSON Configuration

```
{
    "uri": "graph.adj",
    "format":"adj_list",
    "separator":" "
}
```

Java Configuration

```
import oracle.pgx.config.FileGraphConfig;
import oracle.pgx.config.Format;
import oracle.pgx.config.GraphConfigBuilder;
FileGraphConfig config = GraphConfigBuilder
   .forFileFormat(Format.ADJ_LIST)
   .setUri("graph.adj")
   .setSeparator(" ")
   .build();
```

5.2.2 Adding a Vertex Property

The graph in “Creating a Simple Graph File” consists of vertices and edges, without vertex or edge properties. Vertex properties are positioned directly after the source vertex ID in each line. The graph data looks like this after a `double` vertex (node) property is added to the graph:

```
1 0.1 2
2 2.0 3 4
3 0.3 4
4 4.56789 2
```

Note:

The in-memory analyst supports only homogeneous graphs, in which all vertices have the same number and type of properties.

For the in-memory analyst to read the modified data file, you must add a vertex node) property in the configuration file or the builder code. The following examples provide a descriptive name for the property and set the type to `double`.

JSON Configuration

```
{
    "uri": "graph.adj",
    "format":"adj_list",
    "separator":" ",
    "node_props":[{
        "name":"double-prop",
        "type":"double"
    }]
}
```

Java Configuration

```
import oracle.pgx.common.types.PropertyType;
import oracle.pgx.config.FileGraphConfig;
import oracle.pgx.config.Format;
import oracle.pgx.config.GraphConfigBuilder;

FileGraphConfig config = GraphConfigBuilder.forFileFormat(Format.ADJ_LIST)
    .setUri("graph.adj")
    .setSeparator(" ")
    .addNodeProperty("double-prop", PropertyType.DOUBLE)
    .build();
```

5.2.3 Using Strings as Vertex Identifiers

The previous examples used `integer` vertex (node) IDs. The default in In-Memory Analytics is `integer` vertex IDs, but you can define a graph to use `string` vertex IDs instead.

This data file uses "node 1", "node 2", and so forth instead of just the digit:

```
"node 1" 0.1 "node 2"
"node 2" 2.0 "node 3" "node 4"
"node 3" 0.3 "node 4"
"node 4" 4.56789 "node 2"
```

Again, you must modify the graph configuration to match the data file:

JSON Configuration

```
{
    "uri": "graph.adj",
    "format":"adj_list",
    "separator":" ",
    "node_props":[{
        "name":"double-prop",
        "type":"double"
    }],
    "node_id_type":"string"
}
```

Java Configuration

```
import oracle.pgx.common.types.IdType;
import oracle.pgx.common.types.PropertyType;
import oracle.pgx.config.FileGraphConfig;
import oracle.pgx.config.Format;
import oracle.pgx.config.GraphConfigBuilder;

FileGraphConfig config = GraphConfigBuilder.forFileFormat(Format.ADJ_LIST)
    .setUri("graph.adj")
    .setSeparator(" ")
    .addNodeProperty("double-prop", PropertyType.DOUBLE)
    .setNodeIdType(IdType.STRING)
    .build();
```

Note:

`string` vertex IDs consume much more memory than `integer` vertex IDs.

Any single or double quotes inside the string must be escaped with a backslash (\).

Newlines (\n) inside strings are not supported.

5.2.4 Adding an Edge Property

This example adds an edge property of type `string` to the graph. The edge properties are positioned after the destination vertex (node) ID.

```
"node1" 0.1 "node2" "edge_prop_1_2"
"node2" 2.0 "node3" "edge_prop_2_3" "node4" "edge_prop_2_4"
"node3" 0.3 "node4" "edge_prop_3_4"
"node4" 4.56789 "node2" "edge_prop_4_2"
```

The graph configuration must match the data file:

JSON Configuration

```
{
    "uri": "graph.adj",
    "format":"adj_list",
    "separator":" ",
    "node_props":[{
        "name":"double-prop",
        "type":"double"
    }],
    "node_id_type":"string",
     "edge_props":[{
        "name":"edge-prop",
        "type":"string"
    }]
}
```

Java Configuration

```
import oracle.pgx.common.types.IdType;
import oracle.pgx.common.types.PropertyType;
import oracle.pgx.config.FileGraphConfig;
import oracle.pgx.config.Format;
import oracle.pgx.config.GraphConfigBuilder;
```

```
FileGraphConfig config = GraphConfigBuilder.forFileFormat(Format.ADJ_LIST)
    .setUri("graph.adj")
    .setSeparator(" ")
    .addNodeProperty("double-prop", PropertyType.DOUBLE)
    .setNodeIdType(IdType.STRING)
    .addEdgeProperty("edge-prop", PropertyType.STRING)
    .build();
```

5.3 Storing Graph Data on Disk

After reading a graph into memory using either Java or the Shell, you can store it on disk in different formats. You can then use the stored graph data as input to the in-memory analyst at a later time.

Storing graphs over HTTP/REST is currently not supported.

The options include:

- Storing the Results of Analysis in a Vertex Property
- Storing a Graph in Edge-List Format on Disk

5.3.1 Storing the Results of Analysis in a Vertex Property

This example reads a graph into memory and analyzes it using the Pagerank algorithm. This analysis creates a new vertex property to store the PageRank values.

Using the Shell to Run PageRank

```
pgx> g = session.readGraphWithProperties("/opt/oracle/oracle-spatial-graph/
property_graph/examples/pgx/graphs/sample.adj.json", "my-graph")
==> ...
pgx> rank = analyst.pagerank(g, 0.001, 0.85, 100)
```

Using Java to Run PageRank

```
PgxGraph g = session.readGraphWithProperties("/opt/oracle/oracle-spatial-graph/
property_graph/examples/pgx/graphs/sample.adj.json", "my-graph");
VertexProperty<Integer, Double> rank = session.createAnalyst().pagerank(g, 0.001,
0.85, 100);
```

5.3.2 Storing a Graph in Edge-List Format on Disk

This example stores the graph, the result of the Pagerank analysis, and all original edge properties as a file in edge-list format on disk.

To store a graph, you must specify:

- The graph format
- A path where the file will be stored
- The properties to be stored. Specify VertexProperty.`ALL` or EdgeProperty.`ALL` to store all properties, or VertexProperty.`NONE` or EdgePropery.`NONE` to store no properties. To specify individual properties, pass in the VertexProperty or / EdgeProperty objects you want to store.
- A flag that indicates whether to overwrite an existing file with the same name

The following examples store the graph data in `/tmp/sample_pagerank.elist`, with the `/tmp/sample_pagerank.elist.json` configuration file. The return value is the graph configuration stored in the file. You can use it to read the graph again.

Using the Shell to Store a Graph

```
pgx> config = g.store(Format.EDGE_LIST, "/tmp/sample_pagerank.elist", [rank],
EdgeProperty.ALL, false)
==> {"node_props":[{"name":"session-12kta9mj-vertex-prop-
double-2","type":"double"}],"error_handling":{},"node_id_type":"integer","uri":"/tmp/
g.edge","loading":{},"edge_props":
[{"name":"cost","type":"double"}],"format":"edge_list"}
```

Using Java to Store a Graph

```
import oracle.pgx.api.*;
import oracle.pgx.config.*;

FileGraphConfig config = g.store(Format.EDGE_LIST, "/tmp/sample_pagerank.elist",
Collections.singletonList(rank), EdgeProperty.ALL, false);
```

5.4 Executing Built-in Algorithms

The in-memory analyst contains a set of built-in algorithms that are available as Java APIs. This section describes the use of the in-memory analyst using Triangle Counting and Pagerank analytics as examples.

- About the In-Memory Analyst
- Running the Triangle Counting Algorithm
- Running the Pagerank Algorithm

5.4.1 About the In-Memory Analyst

The in-memory analyst contains a set of built-in algorithms that are available as Java APIs. The details of the APIs are documented in the Javadoc that in included in the product documentation library. Specifically, see the `BuiltinAlgorithms` interface Method Summary for a list of the supported in-memory analyst methods.

For example, this is the Pagerank procedure signature:

```
/**
   * Classic pagerank algorithm. Time complexity: O(E * K) with E = number of edges,
K is a given constant (max
   * iterations)
   *
   * @param graph
   *         graph
   * @param e
   *         maximum error for terminating the iteration
   * @param d
   *         damping factor
   * @param max
   *         maximum number of iterations
   * @return Vertex Property holding the result as a double
   */
  public <ID extends Comparable<ID>> VertexProperty<ID, Double> pagerank(PgxGraph
graph, double e, double d, int max);
```

5.4.2 Running the Triangle Counting Algorithm

For triangle counting, the `sortByDegree` boolean parameter of `countTriangles()` allows you to control whether the graph should first be sorted by degree (`true`) or not (`false`). If `true`, more memory will be used, but the algorithm will run faster; however, if your graph is very large, you might want to turn this optimization off to avoid running out of memory.

Using the Shell to Run Triangle Counting

```
pgx> analyst.countTriangles(graph, true)
==> 1
```

Using Java to Run Triangle Counting

```
import oracle.pgx.api.*;

Analyst analyst = session.createAnalyst();
long triangles = analyst.countTriangles(graph, true);
```

The algorithm finds one triangle in the sample graph.

> **Tip:**
>
> When using the in-memory analyst shell, you can increase the amount of log output during execution by changing the logging level. See information about the `:loglevel` command with `:h :loglevel`.

5.4.3 Running the Pagerank Algorithm

Pagerank computes a rank value between `0` and `1` for each vertex (node) in the graph and stores the values in a `double` property. The algorithm therefore creates a *vertex property* of type `double` for the output.

In the in-memory analyst, there are two types of vertex and edge properties:

- **Persistent Properties**: Properties that are loaded with the graph from a data source are fixed, in-memory copies of the data on disk, and are therefore persistent. Persistent properties are read-only, immutable and shared between sessions.
- **Transient Properties**: Values can only be written to transient properties, which are session private. You can create transient properties by alling `createVertexProperty` and `createEdgeProperty`() on `PgxGraph` objects.

This example obtains the top three vertices with the highest Pagerank values. It uses a transient vertex property of type `double` to hold the computed Pagerank values. The Pagerank algorithm uses a maximum error of 0.001, a damping factor of 0.85, and a maximum number of 100 iterations.

Using the Shell to Run Pagerank

```
pgx> rank = analyst.pagerank(graph, 0.001, 0.85, 100);
==> ...
pgx> rank.getTopKValues(3)
==> 128=0.1402019732468347
==> 333=0.12002296283541904
==> 99=0.09708583862990475
```

Using Java to Run Pagerank

```
import java.util.Map.Entry;
import oracle.pgx.api.*;

Analyst analyst = session.createAnalyst();
VertexProperty<Integer, Double> rank = analyst.pagerank(graph, 0.001, 0.85, 100);
for (Entry<Integer, Double> entry : rank.getTopKValues(3)) {
 System.out.println(entry.getKey() + "=" entry.getValue());
}
```

5.5 Creating Subgraphs

You can create subgraphs based on a graph that has been loaded into memory. You can use filter expressions or create bipartite subgraphs based on a vertex (node) collection that specifies the left set of the bipartite graph.

- About Filter Expressions
- Using a Simple Filter to Create a Subgraph
- Using a Complex Filter to Create a Subgraph
- Using a Vertex Set to Create a Bipartite Subgraph

For information about reading a graph into memory, see Reading Graph Data into Memory.

5.5.1 About Filter Expressions

Filter expressions are expressions that are evaluated for each edge. The expression can define predicates that an edge must fulfil to be contained in the result, in this case a subgraph.

Consider the graph in Figure 5-1, which consists of four vertices (nodes) and four edges. For an edge to match the filter expression `src.prop == 10`, the source vertex `prop` property must equal 10. Two edges match that filter expression, as shown in Figure 5-3.

Figure 5-3 Edges Matching src.prop == 10

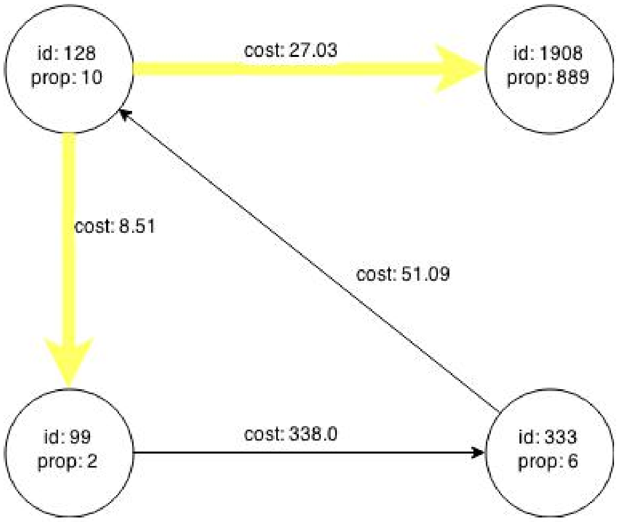
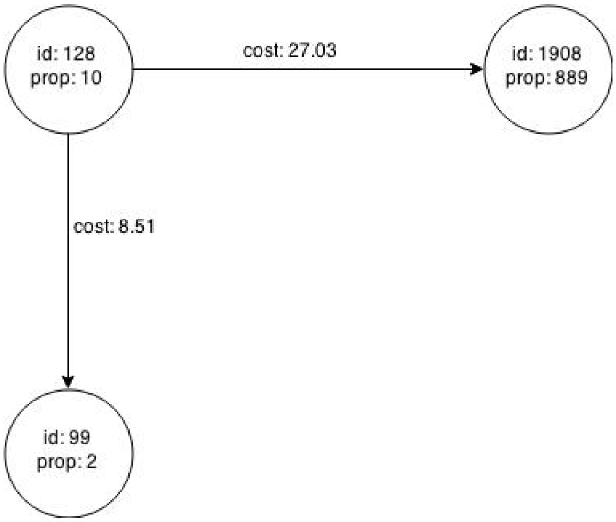


Figure 5-4 shows the graph that results when the filter is applied. The filter excludes the edges associated with vertex 333, and the vertex itself.

Figure 5-4 Graph Created by the Simple Filter



Using filter expressions to select a single vertex or a set of vertices is difficult. For example, selecting only the vertex with the property value `10` is impossible, because the only way to match the vertex is to match an edge where `10` is either the source or destination property value. However, when you match an edge you automatically include the source vertex, destination vertex, and the edge itself in the result.

5.5.2 Using a Simple Filter to Create a Subgraph

The following examples create the subgraph described in "About Filter Expressions".

Using the Shell to Create a Subgraph

```
subgraph = graph.filter(new VertexFilter("vertex.prop == 10"))
```

Using Java to Create a Subgraph

```
import oracle.pgx.api.*;
import oracle.pgx.api.filter.*;

PgxGraph graph = session.readGraphWithProperties(...);
PgxGraph subgraph = graph.filter(new VertexFilter("vertex.prop == 10"));
```

5.5.3 Using a Complex Filter to Create a Subgraph

This example uses a slightly more complex filter. It uses the `outDegree` function, which calculates the number of outgoing edges for an identifier (source `src` or destination `dst`). The following filter expression matches all edges with a `cost` property value greater than 50 and a destination vertex (node) with an `outDegree` greater than 1.

```
dst.outDegree() > 1 && edge.cost > 50
```

One edge in the sample graph matches this filter expression, as shown in Figure 5-5.

Figure 5-5 Edges Matching the outDegree Filter

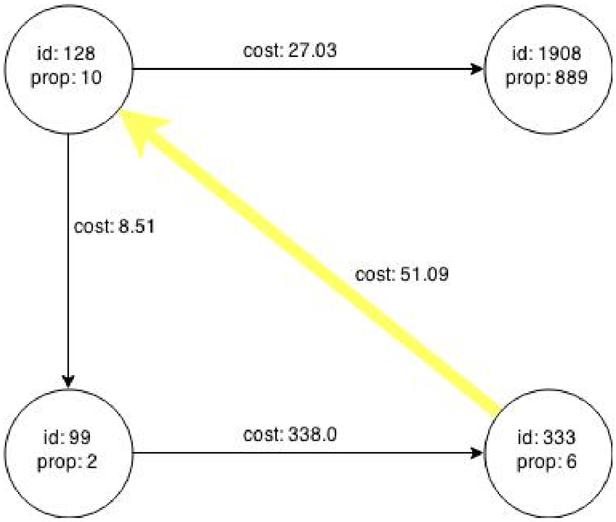
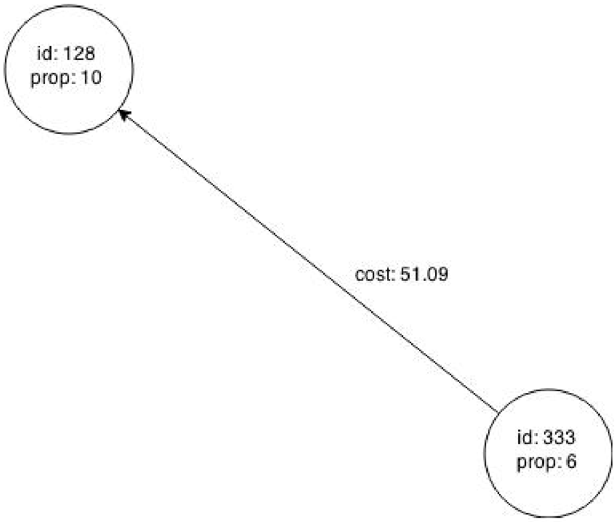


Figure 5-6 shows the graph that results when the filter is applied. The filter excludes the edges associated with vertixes 99 and 1908, and so excludes those vertices also.

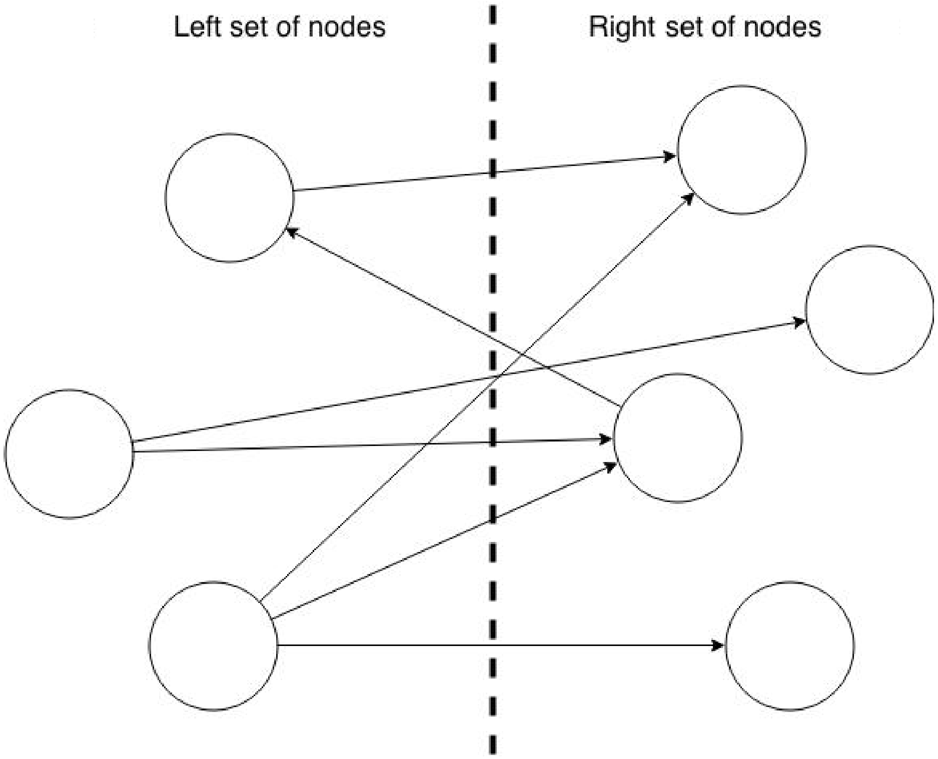
Figure 5-6 Graph Created by the outDegree Filter



5.5.4 Using a Vertex Set to Create a Bipartite Subgraph

You can create a bipartite subgraph by specifying a set of vertices (nodes), which are used as the left side. A bipartite subgraph has edges only between the left set of vertices and the right set of vertices. There are no edges within those sets, such as between two nodes on the left side. In the in-memory analyst, vertices that are isolated because all incoming and outgoing edges were deleted are not part of the bipartite subgraph.

The following figure shows a bipartite subgraph. No properties are shown.



The following examples create a bipartite subgraph from the simple graph created in Figure 5-1. They create a vertex collection and fill it with the vertices for the left side.

Using the Shell to Create a Bipartite Subgraph

```
pgx> s = graph.createVertexSet()
==> ...
pgx> s.addAll([graph.getVertex(333), graph.getVertex(99)])
==> ...
pgx> s.size()
==> 2
pgx> bGraph = graph.bipartiteSubGraphFromLeftSet(s)
==> PGX Bipartite Graph named sample-sub-graph-4
```

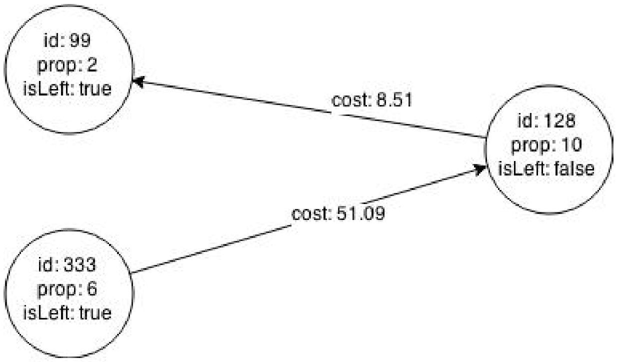
Using Java to Create a Bipartite Subgraph

```
import oracle.pgx.api.*;

VertexSet<Integer> s = graph.createVertexSet();
s.addAll(graph.getVertex(333), graph.getVertex(99));
BipartiteGraph bGraph = graph.bipartiteSubGraphFromLeftSet(s);
```

When you create a subgraph, the in-memory analyst automatically creates a Boolean vertex (node) property that indicates whether the vertex is on the left side. You can specify a unique name for the property.

The resulting bipartite subgraph looks like this:



Vertex 1908 is excluded from the bipartite subgraph. The only edge that connected that vertex extended from 128 to 1908. The edge was removed, because it violated the bipartite properties of the subgraph. Vertex 1908 had no other edges, and so was removed also.

5.6 Deploying to Jetty

You can deploy the in-memory analyst to Eclipse Jetty, Apache Tomcat, or Oracle WebLogic. This example shows how to deploy the in-memory analyst as a web application with Eclipse Jetty.

1. Copy the the in-memory analyst web application archive (WAR) file into the Jetty `webapps` directory:

   ```
   cd $PGX_HOME
   cp $PGX_HOME/webapp/pgx-webapp-1.0.0-for-cdh5.2.1.war $JETTY_HOME/webapps/pgx.war
   ```

2. Set up a security realm within Jetty that specifies where it can find the user names and passwords. To add the most basic security realm, which reads the credentials from a file, add this snippet to `$JETTY_HOME/etc/jetty.xml`:

   ```
   <Call name="addBean">
     <Arg>
       <New class="org.eclipse.jetty.security.HashLoginService">
         <Set name="name">PGX-Realm</Set>
         <Set name="config">
           etc/realm.properties
         </Set>
         <Set name="refreshInterval">0</Set>
       </New>
     </Arg>
   </Call>
   ```

   This snippet instructs Jetty to use the simplest, in-memory login service it supports, the `HashLoginService`. This service uses a configuration file that stores the user names, passwords, and roles.

3. Add the users to `$JETTY_HOME/etc/realm.properties` in the following format:

   ```
   username: password, role
   ```

   For example, this line adds user `SCOTT`, with password `TIGER` and the `USER` role.

   ```
   scott: tiger, USER
   ```

4. Ensure that port 8080 is not already in use, and then start Jetty:

```
cd $JETTY_HOME
java -jar start.jar
```

5. Verify that Jetty is working, using the appropriate credentials for your installation:

```
cd $PGX_HOME
./bin/pgx --base_url http://scott:tiger@localhost:8080/pgx
```

6. (Optional) Modify the in-memory analyst configuration files.

   The configuration file (`pgx.conf`) and the logging parameters (`log4j.xml`) for the in-memory analyst engine are in the WAR file under `WEB-INF/classes`. Restart the server to enable the changes.

   See Also:

   The Jetty documentation for configuration and use at

   `http://eclipse.org/jetty/documentation/`

- About the Authentication Mechanism

5.6.1 About the Authentication Mechanism

The in-memory analyst web deployment uses `BASIC Auth` by default. You should change to a more secure authentication mechanism for a production deployment.

To change the authentication mechanism, modify the `security-constraint` element of the `web.xml` deployment descriptor in the web application archive (WAR) file.

5.7 Deploying to Apache Tomcat

You can deploy the in-memory analyst to Eclipse Jetty, Apache Tomcat, or Oracle WebLogic. This example shows how to deploy In-Memory Analytics as a web application with Apache Tomcat.

The in-memory analyst ships with `BASIC Auth` enabled, which requires a security realm. Tomcat supports many different types of realms. This example configures the simplest one, `MemoryRealm`. See the Tomcat Realm Configuration How-to for information about the other types.

1. Copy the in-memory analyst WAR file into the Tomcat `webapps` directory. For example:

```
cd $PGX_HOME
cp $PGX_HOME/webapp/pgx-webapp-1.0.0-for-cdh5.2.1.war $CATALINA_HOME/webapps/
pgx.war
```

2. Open `$CATALINA_HOME/conf/server.xml` in an editor and add the following realm class declaration under the `<Engine>` element:

```
<Realm className="org.apache.catalina.realm.MemoryRealm" />
```

3. Open `CATALINA_HOME/conf/tomcat-users.xml` in an editor and define a user for the `USER` role. Replace `scott` and `tiger` in this example with an appropriate user name and password:

```
<role rolename="USER" />
<user username="scott" password="tiger" roles="USER" />
```

4. Ensure that port 8080 is not already in use.

5. Start Tomcat:

```
cd $CATALINA_HOME
./bin/startup.sh
```

6. Verify that Tomcat is working:

```
cd $PGX_HOME
./bin/pgx --base_url http://scott:tiger@localhost:8080/pgx
```

Note:

Oracle recommends `BASIC Auth` only for testing. Use stronger authentication mechanisms for all other types of deployments.

See Also:

The Tomcat documentation at

`http://tomcat.apache.org/tomcat-7.0-doc/`

5.8 Deploying to Oracle WebLogic Server

You can deploy the in-memory analysts to Eclipse Jetty, Apache Tomcat, or Oracle WebLogic Server. This example shows how to deploy the in-memory analyst as a web application with Oracle WebLogic Server.

- Installing Oracle WebLogic Server
- Deploying the In-Memory Analyst
- Verifying That the Server Works

5.8.1 Installing Oracle WebLogic Server

To download and install the latest version of Oracle WebLogic Server, see

`http://www.oracle.com/technetwork/middleware/weblogic/documentation/index.html`

5.8.2 Deploying the In-Memory Analyst

To deploy the in-memory analyst to Oracle WebLogic, use commands like the following. Substitute your administrative credentials and WAR file for the values shown in this example:

```
cd $MW_HOME/user_projects/domains/mydomain
. bin/setDomainEnv.sh
java weblogic.Deployer -adminurl http://localhost:7001 -username username -password
password -deploy -upload $PGX_HOME/lib/server/pgx-webapp-0.9.0.war
```

If the script runs successfully, you will see a message like this one:

```
Target state: deploy completed on Server myserver
```

5.8.3 Verifying That the Server Works

Verify that you can connect to the server:

```
$PGX_HOME/bin/pgx --base_url scott:tiger123@localhost:7001/pgx
```

5.9 Connecting to the In-Memory Analyst Server

After the property graph in-memory analyst is installed in a Hadoop cluster -- or on a client system without Hadoop as a web application on Eclipse Jetty, Apache Tomcat, or Oracle WebLogic -- you can connect to the in-memory analyst server.

- Connecting with the In-Memory Analyst Shell
- Connecting with Java
- Connecting with an HTTP Request

5.9.1 Connecting with the In-Memory Analyst Shell

The simplest way to connect to an in-memory analyst instance is to specify the base URL of the server. The following base URL can connect the SCOTT user to the local instance listening on port 8080:

```
http://scott:tiger@localhost:8080/pgx
```

To start the in-memory analyst shell with this base URL, you use the `--base_url` command line argument

```
cd $PGX_HOME
./bin/pgx --base_url http://scott:tiger@localhost:8080/pgx
```

You can connect to a remote instance the same way. However, the in-memory analyst currently does not provide remote support for the Control API.

5.9.1.1 About Logging HTTP Requests

The in-memory analyst shell suppresses all debugging messages by default. To see which HTTP requests are executed, set the log level for `oracle.pgx` to `DEBUG`, as shown in this example:

```
pgx> :loglevel oracle.pgx DEBUG
===> log level of oracle.pgx logger set to DEBUG
pgx> session.readGraphWithProperties("sample_http.adj.json", "sample")
10:24:25,056 [main] DEBUG RemoteUtils - Requesting POST http://scott:tiger@localhost:
8080/pgx/core/session/session-shell-6nqg5dd/graph HTTP/1.1 with payload
{"graphName":"sample","graphConfig":{"uri":"http://path.to.some.server/pgx/
sample.adj","separator":" ","edge_props":
[{"type":"double","name":"cost"}],"node_props":
[{"type":"integer","name":"prop"}],"format":"adj_list"}}
10:24:25,088 [main] DEBUG RemoteUtils - received HTTP status 201
10:24:25,089 [main] DEBUG RemoteUtils - {"futureId":"87d54bed-bdf9-4601-98b7-
ef632ce31463"}
10:24:25,091 [pool-1-thread-3] DEBUG PgxRemoteFuture$1 - Requesting GET http://
scott:tiger@localhost:8080/pgx/future/session/session-shell-6nqg5dd/result/87d54bed-
bdf9-4601-98b7-ef632ce31463 HTTP/1.1
10:24:25,300 [pool-1-thread-3] DEBUG RemoteUtils - received HTTP status 200
10:24:25,301 [pool-1-thread-3] DEBUG RemoteUtils - {"stats":{"loadingTimeMillis":
0,"estimatedMemoryMegabytes":0,"numEdges":4,"numNodes":
```

```
4},"graphName":"sample","nodeProperties":{"prop":"integer"},"edgeProperties":
{"cost":"double"}}
```

This example requires that the graph URI points to a file that the in-memory analyst server can access using HTTP or HDFS.

5.9.2 Connecting with Java

You can specify the base URL when you initialize the in-memory analyst using Java. An example is as follows. A URL to an in-memory analyst server is provided to the `getInMemAnalyst` API call.

```
import oracle.pg.nosql.*;
import oracle.pgx.api.*;

PgNosqlGraphConfig cfg =
GraphConfigBuilder.forNosql().setName("mygraph").setHosts(...).build();
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);
ServerInstance remoteInstance = Pgx.getInstance("http://scott:tiger@hostname:port/
pgx");
PgxSession session = remoteInstance.createSession("my-session");

PgxGraph graph = session.readGraphWithProperties(opg.getConfig());
```

5.9.3 Connecting with an HTTP Request

The in-memory analyst shell uses HTTP requests to communicate with the in-memory analyst server. You can use the same HTTP endpoints directly or use them to write your own client library.

This example uses HTTP to call `create session`:

```
HTTP POST 'http://scott:tiger@localhost:8080/pgx/core/session' with payload
'{"source":"shell"}'
Response: {"sessionId":"session-shell-42v3b9n7"}
```

The call to `create session` returns a session identifier. Most HTTP calls return an in-memory analyst UUID, which identifies the resource that holds the result of the request. Many in-memory analyst requests take a while to complete, but you can obtain a handle to the result immediately. Using that handle, an `HTTP GET` call to a special endpoint provides the result of the request (or block, if the request is not complete).

Most interactions with the in-memory analyst with HTTP look like this example:

```
// any request, with some payload
HTTP POST 'http://scott:tiger@localhost:8080/pgx/core/session/session-shell-42v3b9n7/
graph' with payload '{"graphName":"sample","graphConfig":{"edge_props":
[{"type":"double","name":"cost"}],"format":"adj_list","separator":" ","node_props":
[{"type":"integer","name":"prop"}],"uri":"http://path.to.some.server/pgx/
sample.adj"}}'
Response: {"futureId":"15fc72e9-42e9-4527-9a31-bd20eb0adafb"}

// get the result using the in-memory analyst future UUID.
HTTP GET 'http://scott:tiger@localhost:8080/pgx/future/session/session-
shell-42v3b9n7/result/15fc72e9-42e9-4527-9a31-bd20eb0adafb'
Response: {"stats":{"loadingTimeMillis":0,"estimatedMemoryMegabytes":0,"numNodes":
4,"numEdges":4},"graphName":"sample","nodeProperties":
{"prop":"integer"},"edgeProperties":{"cost":"double"}}
```

5.10 Using the In-Memory Analyst in Distributed Mode

The in-memory analyst can be run in the following modes:

- Shared memory mode

  Multiple threads work in parallel on in-memory graph data stored in a single node (a single, shared memory space). In shared memory mode, the size of the graph is constrained by the physical memory size and by other applications running on the same node.

- Distributed mode

  To overcome the limitations of shared memory mode, you can run the in-memory analyst in distributed mode, in which multiple nodes (computers) form a cluster, partition a large property graph across distributed memory, and work together to provide efficient and scalable graph analytics.

For using the in-memory analyst feature in distributed mode, the following requirements apply to each node in the cluster:

- GNU Compiler Collection (GCC) 4.8.2 or later

  C++ standard libraries built upon 3.4.20 of the GNU C++ API are needed.

- Ability to open a TCP port

  Distributed in-memory analyst requires a designated TCP port to be open for initial handshaking. The default port number is 7777, but you can set it using the run-time parameter `pgx_side_channel_port`.

- Ability to use InfiniBand or UDP on Ethernet

  Data communication among nodes mainly uses InfiniBand (IB) or UDP on Ethernet. When using Ethernet, the machines in the cluster need to accept UDP packets from other computers.

- JDK7 or later

To start the in-memory analyst in distributed mode, do the following. (For this example, assume that four nodes (computers) have been allocated for this purpose, and that they have the host names `hostname0`, `hostname1`, `hostname2`, and `hostname3`.)

On each of the nodes, log in and perform the following operations (modifying the details for your actual environment):

```
export PGX_HOME=/opt/oracle/oracle-spatial-graph/property_graph/pgx
export LD_LIBRARY_PATH=$PGX_HOME/server/distributed/lib:$LD_LIBRARY_PATH

cd $PGX_HOME/server/distributed
./bin/node ./package/ClusterHost.js -server_config=./package/options.json -
pgx_hostnames=hostname0,hostname1,hostname2,hostname3
```

After the operations have successfully completed on all four nodes. you can see a log message similar to the following:

```
17:11:42,709 [hostname0] INFO pgx.dist.cluster_host - PGX.D Server listening on
http://hostname0:8023/pgx
```

The distributed in-memory analyst in now up and running. It provides service through the following endpoint: `http://hostname0:8023/pgx`

This endpoint can be consumed in the same manner as a remotely deployed shared-memory analyst. You can use Java APIs, Groovy shells, and the PGX shell. An example of using the PGX shell is as follows:

```
cd $PGX_HOME
./bin/pgx --base_url=http://hostname0:8023/pgx
```

The following example uses the service from a Groovy shell for Oracle NoSQL Database:

```
opg-nosql> session=Pgx.createSession("http://hostname0:8023/pgx", "session-id-123");
opg-nosql> analyst=session.createAnalyst();
opg-nosql> pgxGraph = session.readGraphWithProperties(opg.getConfig());
```

The following is an example `options.json` file:

```
$ cat ./package/options.json
{
    "pgx_use_infiniband": 1,
    "pgx_command_queue_path": ".",
    "pgx_builtins_path": "./lib",
    "pgx_executable_path": "./bin/pgxd",
    "java_class_path": "./jlib/*",
    "pgx_httpserver_port": 8023,
    "pgx_httpserver_enable_csrf_token": 1,
    "pgx_httpserver_enable_ssl": 0,
    "pgx_httpserver_client_auth": 1,
    "pgx_httpserver_key": "<INSERT_VALUE_HERE>/server_key.pem",
    "pgx_httpserver_cert": "<INSERT_VALUE_HERE>/server_cert.pem",
    "pgx_httpserver_ca": "<INSERT_VALUE_HERE>/server_cert.pem",
    "pgx_httpserver_auth": "<INSERT_VALUE_HERE>/server.auth.json",
    "pgx_log_configure": "./package/log4j.xml",
    "pgx_ranking_query_max_cache_size": 1048576,
    "zookeeper_timeout": 10000,
    "pgx_partitioning_strategy": "out_in",
    "pgx_partitioning_ignore_ghostnodes": false,
    "pgx_ghost_min_neighbors": 5000,
    "pgx_ghost_max_node_counts": 40000,
    "pgx_use_bulk_communication": true,
    "pgx_num_worker_threads": 28
}
```

5.11 Reading and Storing Data in HDFS

The in-memory analyst supports the Hadoop Distributed File System (HDFS). This example shows how to read and access graph data in HDFS using the in-memory analyst APIs.

Graph configuration files are parsed on the client side. The graph data and configuration files must be stored in HDFS. You must install a Hadoop client on the same computer as In-Memory Analytics. See Oracle Big Data Appliance Software User's Guide.

Note:

The in-memory analyst engine runs in memory on one node of the Hadoop cluster only.

- Loading Data from HDFS
- Storing Graph Snapshots in HDFS
- Compiling and Running a Java Application in Hadoop

5.11.1 Loading Data from HDFS

This example copies the `sample.adj` graph data and its configuration file into HDFS, and then loads it into memory.

1. Copy the graph data into HDFS:

```
cd $PGX_HOME
hadoop fs -mkdir -p /user/pgx
hadoop fs -copyFromLocal examples/graphs/sample.adj /user/pgx
```

2. Edit the `uri` field of the graph configuration file to point to an HDFS resource:

```
{
  "uri": "hdfs:/user/pgx/sample.adj",
  "format": "adj_list",
  "node_props": [{
    "name": "prop",
    "type": "integer"
  }],
  "edge_props": [{
    "name": "cost",
    "type": "double"
  }],
  "separator": " "
}
```

3. Copy the configuration file into HDFS:

```
cd $PGX_HOME
hadoop fs -copyFromLocal examples/graphs/sample.adj.json /user/pgx
```

4. Load the sample graph from HDFS into the in-memory analyst, as shown in the following examples.

Using the Shell to Load the Graph from HDFS

```
g = session.readGraphWithProperties("hdfs:/user/pgx/sample.adj.json");
===> {
  "graphName" : "G",
  "nodeProperties" : {
    "prop" : "integer"
  },
  "edgeProperties" : {
    "cost" : "double"
  },
  "stats" : {
    "loadingTimeMillis" : 628,
    "estimatedMemoryMegabytes" : 0,
    "numNodes" : 4,
```

```
    "numEdges" : 4
  }
}
```

Using Java to Load the Graph from HDFS

```
import oracle.pgx.api.*;
PgxGraph g = session.readGraphWithProperties("hdfs:/user/pgx/sample.adj.json");
```

5.11.2 Storing Graph Snapshots in HDFS

The in-memory analyst binary format (`.pgb`) is a proprietary binary graph format for the in-memory analyst. Fundamentally, a `.pgb` file is a binary dump of a graph and its property data, and it is efficient for in-memory analyst operations. You can use this format to quickly serialize a graph snapshot to disk and later read it back into memory.

You should not alter an existing `.pgb` file.

The following examples store the sample graph, currently in memory, in PGB format in HDFS.

Using the Shell to Store a Graph in HDFS

```
g.store(Format.PGB, "hdfs:/user/pgx/sample.pgb", VertexProperty.ALL,
EdgeProperty.ALL, true)
```

Using Java to Store a Graph in HDFS

```
import oracle.pgx.config.GraphConfig;
import oracle.pgx.api.*;

GraphConfig pgbGraphConfig = g.store(Format.PGB, "hdfs:/user/pgx/sample.pgb",
VertexProperty.ALL, EdgeProperty.ALL, true);
```

To verify that the PGB file was created, list the files in the `/user/pgx` HDFS directory:

```
hadoop fs -ls /user/pgx
```

5.11.3 Compiling and Running a Java Application in Hadoop

The following is the `HdfsExample` Java class for the previous examples:

```
import oracle.pgx.api.Pgx;
import oracle.pgx.api.PgxGraph;
import oracle.pgx.api.PgxSession;
import oracle.pgx.api.ServerInstance;
import oracle.pgx.config.Format;
import oracle.pgx.config.GraphConfig;
import oracle.pgx.config.GraphConfigFactory;

public class HdfsDemo {
  public static void main(String[] mainArgs) throws Exception {
    ServerInstance instance = Pgx.getInstance(Pgx.EMBEDDED_URL);
    instance.startEngine();
    PgxSession session = Pgx.createSession("my-session");
    GraphConfig adjConfig = GraphConfigFactory.forAnyFormat().fromHdfs("/user/pgx/
sample.adj.json");
    PgxGraph graph1 = session.readGraphWithProperties(adjConfig);
    GraphConfig pgbConfig = graph1.store(Format.PGB, "hdfs:/user/pgx/sample.pgb");
    PgxGraph graph2 = session.readGraphWithProperties(pgbConfig);
```

```
    System.out.println("graph1 N = " + graph1.getNumVertices() + " E = " +
graph1.getNumEdges());
    System.out.println("graph2 N = " + graph1.getNumVertices() + " E = " +
graph2.getNumEdges());
  }
}
```

These commands compile the HdfsExample class:

```
cd $PGX_HOME
mkdir classes
javac -cp ../lib/* HdfsDemo.java -d classes
```

This command runs the HdfsExample class:

```
java -cp ../lib/*:conf:classes:$HADOOP_CONF_DIR HdfsDemo
```

5.12 Running the In-Memory Analyst as a YARN Application

In this example you will learn how to start, stop and monitor in-memory analyst servers on a Hadoop cluster via Hadoop NextGen MapReduce (YARN) scheduling.

- Starting and Stopping In-Memory Analyst Services
- Connecting to In-Memory Analyst Services
- Monitoring In-Memory Analyst Services

5.12.1 Starting and Stopping In-Memory Analyst Services

Before you can start the in-memory analyst as a YARN application, you must configure the in-memory analyst YARN client.

5.12.1.1 Configuring the In-Memory Analyst YARN Client

The in-memory analyst distribution contains an example YARN client configuration file in `$PGX_HOME/conf/yarn.conf`.

Ensure that all the required fields are set properly. The specified paths must exist in HDFS, and `zookeeper_connect_string` must point to a running ZooKeeper port of the CDH cluster.

5.12.1.2 Starting a New In-Memory Analyst Service

To start a new in-memory analyst service on the Hadoop cluster, use the following command:

```
yarn jar $PGX_HOME/yarn/pgx-yarn-1.0.0-for-cdh5.2.1.jar
```

To use a YARN client configuration file other than `$PGX_HOME/conf/yarn.conf`, provide the file path:

```
yarn jar $PGX_HOME/yarn/pgx-yarn-1.0.0-for-cdh5.2.1.jar /path/to/different/yarn.conf
```

When the service starts, the host name and port of the Hadoop node where the in-memory analyst service launched are displayed.

5.12.1.3 About Long-Running In-Memory Analyst Services

The in-memory analyst YARN applications are configured by default to time out after a specified period. If you disable the time out by setting `pgx_server_timeout_secs` to `0`, the in-memory analyst server keeps running until you or Hadoop explicitly stop it.

5.12.1.4 Stopping In-Memory Analyst Services

To stop a running in-memory analyst service:

```
yarn application -kill appId
```

In this syntax, *appId* is the application ID displayed when the service started.

To inspect the logs of a terminated in-memory analyst service:

```
yarn logs -applicationId appId
```

5.12.2 Connecting to In-Memory Analyst Services

You can connect to in-memory analyst services in YARN the same way you connect to any in-memory analyst server. For example, to connect the Shell interface with the in-memory analyst service, use a command like this one:

```
$PGX_HOME/bin/pgx --base_url username:password@hostname:port
```

In this syntax, *username* and *password* match those specified in the YARN configuration.

5.12.3 Monitoring In-Memory Analyst Services

To monitor in-memory analyst services, click the corresponding YARN application in the Resource Manager Web UI. By default, the Web UI is located at

```
http://resource-manager-hostname:8088/cluster
```

6

Using Multimedia Analytics

You can use the multimedia analytics framework in a Big Data environment to perform facial recognition in videos and images.

6.1 About Multimedia Analytics

The multimedia analytics feature of Oracle Big Data Spatial and Graph provides a framework for processing video and image data in Apache Hadoop. The framework enables distributed processing of video and image data. Features of the framework include:

- APIs to process and analyze video and image data in Apache Hadoop
- Scalable, high speed processing, leveraging the parallelism of Apache Hadoop
- Built-in face recognition using OpenCV
- Ability to install and implement custom video/image processing (for example, license plate recognition) to use the framework to run in Apache Hadoop
- Ability to work with input data in HDFS or HBase

The video analysis framework is installed on Oracle Big Data Appliance if Oracle Spatial and Graph is licensed, and you can install it on other Hadoop clusters.

6.2 Face Recognition Using the Multimedia Analytics Framework

The multimedia analytics feature comes with built-in face recognition. Face recognition uses OpenCV libraries, available with the product. This chapter describes using this face recognition functionality.

Face recognition has two steps:

1. “Training” a model with face images. This step can be run in any Hadoop client or node.
2. Recognizing faces from input video or images using the training model. This step is a MapReduce job that runs in a Hadoop cluster.

The training process creates a **model** stored in a file. This file is used as input for face recognition from videos or images.

Topics:

- Training to Detect Faces
- Selecting Faces to be Used for Training
- Detecting Faces in Videos
- Detecting Faces in Images
- Working with Apache HBase

6.2.1 Training to Detect Faces

Training is done using the Java program `OrdFaceTrainer`, which is part of part of `ordhadoop_multimedia_analytics.jar`. Inputs to this program are a set of images and a label mapping file that maps images to labels. The output is a training model that is written to a file. (You must **not** edit this file.)

To train the multimedia analytics feature to detect (recognize) faces, follow these steps.

1. Create a parent directory and subdirectories to store images that are to be recognized.

   Each subdirectory should contain one or more images of one person. A person can have images in multiple subdirectories, but a subdirectory can have images of only one person. For example, assume that a parent directory named `images` exists where one subdirectory (`d1`) contains images of a person named Andrew, and two subdirectories (`d2` and `d3`) contain images of a person named Betty (such as pictures taken at two different times in two different locations). In this example, the directories and their contents might be as follows:

   - `images/d1` contains five images of Andrew.
   - `images/d2` contains two images of Betty.
   - `images/d3` contains four images of Betty.

2. Create a mapping file that maps image subdirectories to labels.

   A “label” is a numeric ID value to be associated with a person who has images for recognition. For example, Andrew might be assigned the label value 100, and Betty might be assigned the label value 101. Each record (line) in the mapping file must have the following structure:

   ```
   <subdirectory>,<label-id>,<label-text>
   ```

   For example:

   ```
   d1,100,Andrew
   d2,101,Betty
   d3,101,Betty
   ```

3. Set the required configuration properties:

   ```
   oracle.ord.hadoop.ordfacemodel
   oracle.ord.hadoop.ordfacereader
   oracle.ord.hadoop.ordsimplefacereader.dirmap
   oracle.ord.hadoop.ordsimplefacereader.imagedir
   ```

For information about the available properties, see Configuration Properties for Multimedia Analytics.

4. Set the CLASSPATH. Include the following in the Java CLASSPATH definition. Replace each asterisk (*) with the actual version number.

```
$MMA_HOME/lib/ordhadoop-multimedia-analytics.jar
$MMA_HOME/opencv_3.0.0/opencv-300.jar
$HADOOP_HOME/hadoop-common-*.jar
$HADOOP_HOME/hadoop-auth-*.jar
$HADOOP_HOME/commons-lang*.jar
$HADOOP_HOME/commons-logging-*.jar
$HADOOP_HOME/commons-configuration-*.jar
$HADOOP_HOME/commons-collections-*.jar
$HADOOP_HOME/guava-*.jar
$HADOOP_HOME/slf4j-api-*.jar
$HADOOP_HOME/slf4j-log4j12-*.jar
$HADOOP_HOME/log4j-*.jar
$HADOOP_HOME/commons-cli-*.jar
$HADOOP_HOME/protobuf-java-*.jar
$HADOOP_HOME/avro-*.jar
$HADOOP_HOME/hadoop-hdfs-*.jar
$HADOOP_HOME/hadoop-mapreduce-client-core-*.jar
```

5. Create the training model. Enter a command in the following general form:

```
java -classpath <…> oracle.ord.hadoop.recognizer.OrdFaceTrainer
<training_config_file.xml>
```

Note: `$MMA_HOME/example` has a set of sample files. It includes scripts for setting the Java `CLASSPATH`. You can edit the example as needed to create a training model.

6.2.2 Selecting Faces to be Used for Training

Images used to create the training model should contain only the face, with as little extra detail around the face as possible. The following are some examples, showing four images of the same man's face with different facial expressions.

The selection of images for training is important for accurate matching. The following guidelines apply:

- The set of images should contain faces with all possible positions and facial movements, for example, closed eyes, smiles, and so on.
- Try to avoid including images that are very similar.

- If it is necessary to recognize a person with several backgrounds and light conditions, include images with these backgrounds.
- The number of images to include depends on the variety of movements and backgrounds expected in the input data.

6.2.3 Detecting Faces in Videos

To detect (recognize) faces in videos, you have the following options for video processing software to transcode video data:

- Use `OrdOpenCVFaceRecognizerMulti` as the frame processor, along with any of the frontal face cascade classifiers available with OpenCV.

  `Haarcascade_frontalface_alt2.xml` is a good place to start. You can experiment with the different cascade classifiers to identify a good fit for your requirements.
- Use third-party face recognition software.

To perform recognition, follow these steps:

1. Copy the video files (containing video in which you want to recognize faces) to HDFS.
2. Copy these required files to a shared location accessible by all nodes in the cluster:
   - Generated training model
   - Mapping file that maps image subdirectories to labels
   - Cascade classifier XML file
3. Create the configuration file.

   Required configuration parameters:
   - `oracle.ord.hadoop.inputtype`: Type if input data (`video` or `image`).
   - `oracle.ord.hadoop.outputtypes`: Format of generated results (`JSON/text/Image`).
   - `oracle.ord.hadoop.ordframegrabber`: Get a video frame from the video data. You can use the Java classes available with the product or you can provide an implementation for the abstraction.
     - OrdJCodecFrameGrabber is available with the product. This class can be used without any additional steps. See www.jcodec.org for more details on JCodec.
     - OrdFFMPEGFrameGrabber is available with the product. This class requires installation of FFMPEG libraries. See www.ffmpeg.org for more details
   - `oracle.ord.hadoop.ordframeprocessor`: Processor to use on the video frame to recognize faces. You can use the Java classes available with the product or you can provide an implementation for the abstraction.
   - `oracle.ord.hadoop.recognizer.classifier`: Cascade classifier XML file.

- `oracle.ord.hadoop.recognizer.labelnamefile`: Mapping file that maps image subdirectories to labels.

Optional configuration parameters:

- `oracle.ord.hadoop.frameinterval`: Time interval (number of seconds) between frames that are processed. Default: 1.
- `oracle.ord.hadoop.numofsplits`: Number of splits of the video file on the Hadoop cluster, with one split analyzed on each node of the Hadoop cluster. Default: 1.
- `oracle.ord.hadoop.recognizer.cascadeclassifier.scalefactor`: Scale factor to be used for matching images used in training with faces identified in video frames or images. Default: 1.1 (no scaling)
- `oracle.ord.hadoop.recognizer.cascadeclassifier.minneighbor`: Determines size of the sliding window to detect face in video frame or image. Default: 1.
- `oracle.ord.hadoop.recognizer.cascadeclassifier.flags`: Determines type of face detection.
- `oracle.ord.hadoop.recognizer.cascadeclassifier.minsize`: Smallest bounding box used to detect a face.
- `oracle.ord.hadoop.recognizer.cascadeclassifier.maxsize`: Largest bounding box used to detect a face.
- `oracle.ord.hadoop.recognizer.cascadeclassifier.maxconfidence`: Maximum allowable distance between the detected face and a face in the model.
- `oracle.ord.hadoop.ordframeprocessor.k2`: Key class for the implemented class for `OrdFrameProcessor`.
- `oracle.ord.hadoop.ordframeprocessor.v2`: Value class for the implemented class for `OrdFrameProcessor`.

4. Set the HADOOP_CLASSPATH.

   Ensure that HADOOP_CLASSPATH includes the files listed in Training to Detect Faces

5. Run the Hadoop job to recognize faces. Enter a command in the following format:

```
$ hadoop jar $MMA_HOME/lib/orhadoop-multimedia-analytics.jar -conf <conf file>
<hdfs_input_directory_containing_video_data>
<hdfs_output_directory_to_write_results>
```

The accuracy of detecting faces depends on a variety of factors, including lighting, brightness, orientation of the face, distance of the face from the camera, and clarity of the video or image. You should experiment with the configuration properties to determine the best set of values for your use case. Note that it is always possible to have false positives (identifing objects that are not faces as faces) and false recognitions (wrongly labeling a face).

Note: `$MMA_HOME/example` has a set of sample files. It includes scripts for setting the Java CLASSPATH. You can edit as needed to submit a job to detect faces.

6.2.4 Detecting Faces in Images

To detect faces in images, copy the images to HDFS. Specify the following property:

```
<property>
  <name>oracle.ord.hadoop.inputtype</name>
  <value>image</value>
</property>
```

6.2.5 Working with Apache HBase

Apache provides performance improvements when working with small objects such as images. Images can be stored in an HBase table and accessed by the multimedia analytics framework. If input data is video, then the video must be decoded into frames and the frames stored in an HBase table.

The following properties are used when the input or output is an HBase table:

- `oracle.ord.hadoop.datasource` – Storage option for input data. Specify HBase if input data is in an HBase table. Default is HDFS.
- `oracle.ord.hbase.input.table` – Name of the HBase table containing the input data.
- `oracle.ord.hbase.input.columnfamily` – Name of the HBase column family containing the input data.
- `oracle.ord.hbase.input.column` – Name of the HBase column containing the input data.
- `oracle.ord.hadoop.datasink` – Storage option for the output of multimedia analysis. Specify HBase to use an HBase table to store the output. Default is HDFS.
- `oracle.ord.hbase.output.columnfamily` – Name of the HBase column family in the output HBase table.

6.2.6 Examples and Training Materials for Detecting Faces

Several examples and training materials are provided to help you get started detecting faces.

$MMA_HOME contains these directories:

```
video/ (contains a sample video file in mp4 and avi formats)
facetrain/
analytics/
```

`facetrain/` contains an example for training, `facetrain/config/` contains the sample configuration files, and `facetrain/faces/` contains images to create the training model and the mapping file that maps labels to images.

`runFaceTrainExample.sh` is a bash example script to run the training step.

You can create the training model as follows:

```
$ ./runFaceTrainExample.sh
```

The training model will be written to `ordfacemodel_bigdata.dat`.

For detecting faces in videos, `analytics/` contains an example for running a Hadoop job to detect faces in the input video file. This directory contains `conf/` with configuration files for the example.

You can run the job as follows (includes copying the video file to HDFS directory `vinput`)

```
$ ./runFaceDetectionExample.sh
```

The output of the job will be in the HDFS directory `voutput`.

For recognizing faces in videos, `analytics/` contains an example for running a Hadoop job to recognize faces in the input video file. This directory contains `conf/` with configuration files for the example. You can run the job as follows (includes copying the video file to the HDFS directory `vinput`):

```
$ ./runFaceRecognizerExample.sh
```

After the face recognition job, you can display the output images:

```
$ ./runPlayImagesExample.sh
```

6.3 Configuration Properties for Multimedia Analytics

The multimedia analytics framework uses the standard methods for specifying configuration properties in the `hadooop` command. You can use the `–conf` option to identify configuration files, and the `-D` option to specify individual properties. This topic presents reference information about the configuration properties.

Some properties are used for specific tasks. For example, training properties include:

- `oracle.ord.hadoop.ordfacereader`
- `oracle.ord.hadoop.ordsimplefacereader.imagedir`
- `oracle.ord.hadoop.ordsimplefacereader.dirmap`
- `oracle.ord.hadoop.ordfacemodel`
- `oracle.ord.hadoop.ordfacereaderconfig`

The following are the available configuration properties, listed in alphabetical order. For each parameter the parameter name is listed, then information about the parameter.

oracle.ord.hadoop.datasink
String. Storage option for the output of multimedia analysis: `HBase` to use an HBase table to store the output; otherwise, `HDFS`. Default value: `HDFS`. Example:

```
<property>
  <name>oracle.ord.hadoop.datasink</name>
  <value>hbase</value>
</property>
```

oracle.ord.hadoop.datasource
String. Storage option for input data: `HBase` if the input data is in an HBase database; otherwise, `HDFS`. Default value: `HDFS`. Example:

```
<property>
  <name>oracle.ord.hadoop.datasource</name>
  <value>hbase</value>
</property>
```

oracle.ord.hadoop.frameinterval
String.Timestamp interval (in seconds) to extract frames for processing. Allowable values: positive integers and floating point numbers. Default value: 1. Example:

```
<property>
  <name>oracle.ord.hadoop.frameinterval</name>
  <value>1</value>
</property>
```

oracle.ord.hadoop.inputformat
Sring. The `InputFormat` class name in the framework, which represents the input file type in the framework. Default value: `oracle.ord.hadoop.OrdVideoInputFormat`. Example:

```
<property>
  <name>oracle.ord.hadoop.inputformat</name>
  <value>oracle.ord.hadoop.OrdVideoInputFormat</value>
</property>
```

oracle.ord.hadoop.inputtype
String. Type of input data: `video` or `image`. Example:

```
<property>
  <name>oracle.ord.hadoop.inputtype</name>
  <value>video</value>
</property>
```

oracle.ord.hadoop.numofsplits
Positive integer. Number of the splits of the video files on the Hadoop cluster, with one split able to be analyzed in each node of the Hadoop cluster. Recommended value: the number of nodes/processors in the cluster. Default value: 1. Example:

```
<property>
   <name>oracle.ord.hadoop.numofsplits</name>
   <value>1</value>
</property>
```

oracle.ord.hadoop.ordfacemodel
String. Name of the file that stores the model created by the training. Example:

```
<property>
   <name> oracle.ord.hadoop.ordfacemodel </name>
   <value>ordfacemodel_bigdata.dat</value>
</property>
```

oracle.ord.hadoop.ordfacereader
String. Name of the Java class that reads images used for training the face recognition model. Example:

```
<property>
   <name> oracle.ord.hadoop.ordfacereader </name>
   <value> oracle.ord.hadoop.OrdSimpleFaceReader </value>
</property>
```

oracle.ord.hadoop.ordfacereaderconfig
String. File containing additional configuration properties for the specific application. Example:

```
<property>
   <name> oracle.ord.hadoop.ordfacereaderconfig </name>
   <value>config/ordsimplefacereader_bigdata.xml</value>
</property>
```

oracle.ord.hadoop.ordframegrabber
String. Name of the Java class that decodes a video file. This is the implemented class for `OrdFrameGrabber`, and it is used by the mapper to decode the video file. Available installed implementations with the product: `oracle.ord.hadoop.OrdJCodecFrameGrabber` (the default) and `oracle.ord.hadoop.OrdFFMPEGFrameGrabber` (when FFMPEG is installed by the user). You can add custom implementations. Example:

```
<property>
    <name>oracle.ord.hadoop.ordframegrabber</name>
    <value>oracle.ord.hadoop.OrdJCodecFrameGrabber</value>
</property>
```

oracle.ord.hadoop.ordframeprocessor
String. Name of the implemented Java class of interface OrdFrameProcessor, which is used by the mapper to process the frame and recognize the object of interest. Default value: oracle.ord.hadoop.mapreduce.OrdOpenCVFaceRecognizerMulti. Example:

```
<property>
  <name>oracle.ord.hadoop.ordframeprocessor </name>
  <value>oracle.ord.hadoop.mapreduce.OrdOpenCVFaceRecognizerMulti</value>
</property>
```

oracle.ord.hadoop.ordframeprocessor.k2
String. Java class name, output key class of the implemented class of interface `OrdFrameProcessor`. Default value: `org.apache.hadoop.io.Text`. Example:

```
<property>
  <name>oracle.ord.hadoop.ordframeprocessor.k2</name>
  <value>org.apache.hadoop.io.Text</value>
</property>
```

oracle.ord.hadoop.ordframeprocessor.v2
String. Java class name, output value class of the implemented class of interface `OrdFrameProcessor` . Default value: `oracle.ord.hadoop.mapreduce.OrdImageWritable`. Example:

```
<property>
  <name>oracle.ord.hadoop.ordframeprocessor.v2 </name>
  <value>oracle.ord.hadoop.mapreduce.OrdImageWritable</value>
</property>
```

oracle.ord.hadoop.ordoutputprocessor
String. Only only relevant for custom (user-specified) plug-ins: name of the implemented Java class of interface `OrdOutputProcessor` that processes the key-value pair from the map output in the reduce phase. Example:

```
<property>
  <name>oracle.ord.hadoop.ordframeprocessor</name>
```

```
  <value>mypackage.MyOutputProcessorClass</value>
</property>
```

oracle.ord.hadoop.ordsimplefacereader.dirmap
String. Mapping file that maps face labels to directory names and face images. Example:

```
<property>
   <name> oracle.ord.hadoop.ordsimplefacereader.dirmap </name>
   <value>faces/bigdata/dirmap.txt</value>
</property>
```

oracle.ord.hadoop.ordsimplefacereader.imagedir
String. File system directory containing faces used to create a model. This is typically in a local file system. Example:

```
<property>
   <name> oracle.ord.hadoop.ordsimplefacereader.imagedir </name>
   <value>faces/bigdata</value>
</property>
```

oracle.ord.hadoop.outputformat
String. Name of the OutputFormat class, which represents the output file type in the framework. Default value: `org.apache.hadoop.mapreduce.lib.output.TextOutputFormat`. Example:

```
<property>
  <name>oracle.ord.hadoop.outputformat</name>
  <value> org.apache.hadoop.mapreduce.lib.output.TextOutputFormat; </value>
</property>
```

oracle.ord.hadoop.outputtype
String. Format of output that contains face labels of identified faces with the time stamp, location, and confidence of the match: must be `json`, `image`, or `text`. Example:

```
<property>
  <name>oracle.ord.hadoop.outputtype</name>
  <value>json</value>
</property>
```

oracle.ord.hadoop.parameterfile
String. File containing additional configuration properties for the specific job. Example:

```
<property>
  <name>oracle.ord.hadoop.parameterfile </name>
  <value>oracle_multimedia_face_recognition.xml</value>
</property>
```

oracle.ord.hadoop.recognizer.cascadeclassifier.flags
String. Use this property to select the type of object detection. Must be `CASCADE_DO_CANNY_PRUNING`, `CASCADE_SCALE_IMAGE`, `CASCADE_FIND_BIGGEST_OBJECT` (look only for the largest face), or `CASCADE_DO_ROUGH_SEARCH`. . Default: `CASCADE_SCALE_IMAGE | CASCADE_DO_ROUGH_SEARCH`. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.cascadeclassifier.flags</name>
```

```
  <value>CASCADE_SCALE_IMAGE</value>
</property>
```

oracle.ord.hadoop.recognizer.cascadeclassifier.maxconfidence
Floating point value. Specifies how large the distance (difference) between a face in the model and a face in the input data can be. Larger valuse will give more matches but might be less accurate (more false positives). Smaller values will give fewer matches, but be more accurate. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.cascadeclassifier.maxconfidence</name>
  <value>200.0</value>
</property
```

oracle.ord.hadoop.recognizer.cascadeclassifier.maxsize
String, specifically a pair of values. Specifies the maximum size of the bounding box for the object detected. If the object is close by, the bounding box is larger; if the object is far away, like faces on a beach, the bounding box is smaller. Objects with a larger bounding box than the maximum size are ignored. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.cascadeclassifier.maxsize</name>
  <value>(500,500)</value>
</property>
```

oracle.ord.hadoop.recognizer.cascadeclassifier.minneighbor
Integer. Determines the size of the sliding window used to detect the object in the input data. Higher values will detect fewer objects but with higher quality. Default value: 1. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.cascadeclassifier.minneighbor</name>
  <value>1</value>
</property>
```

oracle.ord.hadoop.recognizer.cascadeclassifier.minsize
String, specifically a pair of values. Specifies the minimum size of the bounding box for the object detected. If the object is close by, the bounding box is larger; if the object is far away, like faces on a beach, the bounding box is smaller. Objects with a smaller bounding box than the minimum size are ignored. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.cascadeclassifier.minsize</name>
  <value>(100,100)</value>
</property>
```

oracle.ord.hadoop.recognizer.cascadeclassifier.scalefactor
Floating pointnumber. Scale factor to be used with the mapping file that maps face labels to directory names and face images. A value of 1.1 means to perform no scaling before comparing faces in the run-time input with images stored in subdirectories during the training process. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.cascadeclassifier.scalefactor</name>
  <value>1.1</value>
</property>
```

oracle.ord.hadoop.recognizer.classifier
String. XML file containing classifiers for face. The feature can be used with any of the frontal face pre-trained classifiers available with OpenCV. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.classifier</name>
  <value>haarcascade_frontalface_alt2.xml</value>
</property>
```

oracle.ord.hadoop.recognizer.labelnamefile
String. Mapping file that maps face labels to directory names and face images. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.labelnamefiler</name>
  <value>haarcascade_frontalface_alt2.xml</value>
</property>
```

oracle.ord.hadoop.recognizer.modelfile
String. File containing the model generated in the training step. The file must be in a shared location, accessible by all cluster nodes. Example:

```
<property>
  <name> oracle.ord.hadoop.recognizer.modelfile</name>
  <value>myface_model.dat</value>
</property>
```

oracle.ord.hbase.input.column
String. Name of the HBase column containing the input data. Example:

```
<property>
  <name>oracle.ord.hbase.input.column</name>
  <value>binary_data</value>
</property>
```

oracle.ord.hbase.input.columnfamily
String. Name of the HBase column family containing the input data. Example:

```
<property>
  <name>oracle.ord.hbase.input.columnfamily</name>
  <value>image_data</value>
</property>
```

oracle.ord.hbase.input.table
String. Name of the HBase table containing the input data. Example:

```
<property>
  <name>oracle.ord.hbase.input.table</name>
  <value>images</value>
</property>
```

oracle.ord.hbase.output.columnfamily
String. Name of the HBase column family in the output HBase table. Example:

```
<property>
  <name>oracle.ord.hbase.output.columnfamily</name>
  <value>face_data</value>
</property>
```

oracle.ord.hbase.output.table
String. Name of the HBase table for output data. Example:

```
<property>
  <name>oracle.ord.hbase.output.table</name>
  <value>results</value>
</property>
```

6.4 Using the Multimedia Analytics Framework with Third-Party Software

You can implement and install custom modules for multimedia decoding and processing.

You can use a custom video decoder in the framework by implementing the abstract class `oracle.ord.hadoop.decoder.OrdFrameGrabber`. See the Javadoc for additional details. The product includes two implementations of the video decoder that extend `OrdFrameGrabber` for JCodec and FFMPEG (requires a separate installation of FFMPEG).

You can use custom multimedia analysis in the framework by implementing two abstract classes.

- `oracle.ord.hadoop.mapreduce.OrdFrameProcessor<K1,V1,K2,V2>`. The extended class of `OrdFrameProcessor` is used in the map phase of the MapReduce job that processes the video frames or images. (K1, V1) is the input key-value pair types and (K2, V2) is the output key-value pair type. See the Javadoc for additional details. The product includes an implementation using OpenCV.
- `oracle.ord.hadoop.mapreduce.OrdOutputProcessor<K1,V1,K2,V2>`. The extended class of `OrdFrameProcessor` is used in the reducer phase of the MapReduce job that processes the video frames or images. (K1, V1) is the input key-value pair types and (K2, V2) is the output key-value pair type. See the Javadoc for additional details. Most implementations do not require implementing this class.

An example of framework configuration parameters is available in `$MMA_HOME/example/analytics/conf/oracle_multimedia_analysis_framework.xml`.

6.5 Displaying Images in Output

If the output is displayed as images, oracle.ord.hadoop.OrdPlayImages can be used to display all the images in the output HDFS directory. This will display the image frames marked with labels for identified faces. For example:

```
$ java oracle.ord.hadoop.demo.OrdPlayImages –hadoop_conf_dir $HADOOP_CONF_DIR –
image_file_dir voutput
```

A

Third-Party Licenses for Bundled Software

Oracle Big Data Spatial and Graph installs several third-party products. This appendix lists information that applies to all Apache licensed code, and then it lists license information for the installed third-party products.

- Apache Licensed Code
- ANTLR 3
- AOP Alliance
- Apache Commons CLI
- Apache Commons Codec
- Apache Commons Collections
- Apache Commons Configuration
- Apache Commons IO
- Apache Commons Lang
- Apache Commons Logging
- Apache Commons VFS
- Apache fluent
- Apache Groovy
- Apache htrace
- Apache HTTP Client
- Apache HTTPComponents Core
- Apache Jena
- Apache Log4j
- Apache Lucene
- Apache Tomcat
- Apache Xerces2
- Apache xml-commons
- Argparse4j

- check-types
- Cloudera CDH
- cookie
- Fastutil
- functionaljava
- GeoNames Data
- Geospatial Data Abstraction Library (GDAL)
- Google Guava
- Google Guice
- Google protobuf
- int64-native
- Jackson
- Jansi
- JCodec
- Jettison
- JLine
- Javassist
- json-bignum
- Jung
- Log4js
- MessagePack
- Netty
- Node.js
- node-zookeeper-client
- OpenCV
- rxjava-core
- Slf4j
- Spoofax
- Tinkerpop Blueprints
- Tinkerpop Gremlin
- Tinkerpop Pipes

A.1 Apache Licensed Code

The following is included as a notice in compliance with the terms of the Apache 2.0 License, and applies to all programs licensed under the Apache 2.0 license:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

A copy of the license is also reproduced below.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that you meet the following conditions:

a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

b) You must cause any modified files to carry prominent notices stating that You changed the files; and

c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Do not include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

`http://www.apache.org/licenses/LICENSE-2.0`

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This product includes software developed by The Apache Software Foundation (http://www.apache.org/Opens a new window) (listed below):

A.2 ANTLR 3

This product was build using ANTLR, which was provided to Oracle under the following terms:Copyright (c) 2010 Terence ParrAll rights reserved.Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.3 AOP Alliance

LICENCE: all the source code provided by AOP Alliance is Public Domain.

A.4 Apache Commons CLI

Copyright 2001-2009 The Apache Software FoundationThis product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.5 Apache Commons Codec

Copyright 2002-2009 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

src/test/org/apache/commons/codec/language/DoubleMetaphoneTest.java contains test data from http://aspell.sourceforge.net/test/batch0.tab.

Copyright (C) 2002 Kevin Atkinson (kevina@gnu.org). Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.

A.6 Apache Commons Collections

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

Apache Commons Collections Copyright 2001-2008 The Apache Software Foundation

A.7 Apache Commons Configuration

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

Apache Commons Configuration Copyright 2001-2014 The Apache Software Foundation

A.8 Apache Commons IO

Copyright 2002-2012 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.9 Apache Commons Lang

Copyright 2001-2010 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.10 Apache Commons Logging

Copyright 2003-2007 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.11 Apache Commons VFS

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. A copy of the license is also reproduced in this document.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

A.12 Apache fluent

Copyright © 2011-2014 The Apache Software Foundation. All rights reserved.

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.13 Apache Groovy

Copyright 2009-2015 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.14 Apache htrace

Copyright 2009-2015 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.15 Apache HTTP Client

Copyright 1999-2013 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.16 Apache HTTPComponents Core

Copyright 2005-2013 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

This project contains annotations derived from JCIP-ANNOTATIONS

Copyright (c) 2005 Brian Goetz and Tim Peierls. See http://www.jcip.net

A.17 Apache Jena

Copyright 2011, 2012, 2013, 2014 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

Portions of this software were originally based on the following:

- Copyright 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009 Hewlett-Packard Development Company, LP

- Copyright 2010, 2011 Epimorphics Ltd.

- Copyright 2010, 2011 Talis Systems Ltd.

These have been licensed to the Apache Software Foundation under a software grant.

This product includes software developed by PluggedIn Software under a BSD license.

This product includes software developed by Mort Bay Consulting Pty. Ltd.

Copyright (c) 2004-2009 Mort Bay Consulting Pty. Ltd.

A.18 Apache Log4j

Copyright 2007 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.19 Apache Lucene

Copyright 2011-2012 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.20 Apache Tomcat

Copyright 1999-2014 The Apache Software Foundation

This product includes software developed at The Apache Software Foundation (http://www.apache.org/).

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. A copy of the license is also reproduced in this document.

The Windows Installer is built with the Nullsoft Scriptable Install Sysem (NSIS), which is open source software. The original software and related information is available at http://nsis.sourceforge.net.

Java compilation software for JSP pages is provided by Eclipse, which is open source software. The original software and related information is available at

http://www.eclipse.org.

A.21 Apache Xerces2

Copyright 1999-2012 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

A.22 Apache xml-commons

Apache XML Commons XML APIs

Copyright 1999-2009 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`).

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.

- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.

- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org

A.23 Argparse4j

Copyright (C) 2011, 2014, 2015 Tatsuhiro Tsujikawa

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

A.24 check-types

Copyright © 2012, 2013, 2014, 2015 Phil Booth

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in allcopies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THEAUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.25 Cloudera CDH

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

A.26 cookie

Copyright (c) 2012-2014 Roman Shtylman <shtylman@gmail.com>

Copyright (c) 2015 Douglas Christopher Wilson <doug@somethingdoug.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.27 Fastutil

Fastutil is available under the Apache License, Version 2.0.

A.28 functionaljava

Copyright (c) 2008-2011, Tony Morris, Runar Bjarnason, Tom Adams, Brad Clow, Ricky Clarkson, Jason Zaugg All rights reserved.

Redistribution and use in source and binary forms, with or withoutmodification, are permitted provided that the following conditionsare met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE

FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.29 GeoNames Data

This distribution includes and/or the service uses a modified version of the GeoNames geographical database, for distributions which may be found in a set of files with names in the form world_xxxxx.json: one file for cities, one for counties, one for states, and one for countries. And there is another file with alternate names called db_alternate_names.txt. All of these files are generated from the GeoNames database. The original GeoNames database is available at www.geonames.org under the license set forth below.

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

"Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

"Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

"Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

"Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

"Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

"Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

"You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

"Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

"Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

to Distribute and Publicly Perform the Work including as incorporated in Collections; and, to Distribute and Publicly Perform Adaptations.

For the avoidance of doubt:

Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(b), as requested.

If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

A.30 Geospatial Data Abstraction Library (GDAL)

GDAL/OGR General

In general GDAL/OGR is licensed under an MIT/X style license with the

following terms:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESSOR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

gdal/frmts/gtiff/tif_float.c

Copyright (c) 2002, Industrial Light & Magic, a division of Lucas Digital Ltd. LLC

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Industrial Light & Magic nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

gdal/frmts/hdf4/hdf-eos/*

\-----------------------

Copyright (C) 1996 Hughes and Applied Research Corporation

Permission to use, modify, and distribute this software and its documentation

for any purpose without fee is hereby granted, provided that the above

copyright notice appear in all copies and that both that copyright notice and

this permission notice appear in supporting documentation.

gdal/frmts/pcraster/libcsf

\-------------------------

Copyright (c) 1997-2003, Utrecht University

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Utrecht University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED

AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

gdal/frmts/grib/degrib/*

The degrib and g2clib source code are modified versions of code produced by NOAA NWS and are in the public domain subject to the following restrictions:

http://www.weather.gov/im/softa.htm

DISCLAIMER The United States Government makes no warranty, expressed or implied, as to the usefulness of the software and documentation for any purpose. The U.S. Government, its instrumentalities, officers, employees, and agents assumes no responsibility (1) for the use of the software and documentation listed below, or (2) to provide technical support to users.

http://www.weather.gov/disclaimer.php

The information on government servers are in the public domain, unless specifically annotated otherwise, and may be used freely by the public so long as you do not 1) claim it is your own (e.g. by claiming copyright for NWS information -- see below), 2) use it in a manner that implies an endorsement or affiliation with NOAA/NWS, or 3) modify it in content and then present it as official government material. You also cannot present information of your own in a way that makes it appear to be official government information.

The user assumes the entire risk related to its use of this data. NWS is providing this data "as is," and NWS disclaims any and all warranties, whether express or implied, including (without limitation) any implied warranties of merchantability or fitness for a particular purpose. In no event will NWS be liable to you or to any third party for any direct, indirect, incidental, consequential, special or exemplary damages or lost profit resulting from any use or misuse of this data.

As required by 17 U.S.C. 403, third parties producing copyrighted works consisting predominantly of the material appearing in NWS Web pages must provide notice with such work(s) identifying the NWS material incorporated and stating that such material is not subject to copyright protection.

port/cpl_minizip*

This is version 2005-Feb-10 of the Info-ZIP copyright and license.

The definitive version of this document should be available at

ftp://ftp.info-zip.org/pub/infozip/license.html indefinitely.

Copyright (c) 1990-2005 Info-ZIP. All rights reserved.

For the purposes of this copyright and license, "Info-ZIP" is defined as

the following set of individuals:

Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois,

Jean-loup Gailly, Hunter Goatley, Ed Gordon, Ian Gorman, Chris Herborth,

Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz,

David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs,

Kai Uwe Rommel, Steve Salisbury, Dave Smith, Steven M. Schweda,

Christian Spieler, Cosmin Truta, Antoine Verheijen, Paul von Behren,

Rich Wales, Mike White

This software is provided "as is," without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. Redistributions of source code must retain the above copyright notice, definition, disclaimer, and this list of conditions.

2. Redistributions in binary form (compiled executables) must reproduce the above copyright notice, definition, disclaimer, and this list of conditions in documentation and/or other materials provided with the distribution. The sole exception to this condition is redistribution of a standard UnZipSFX binary (including SFXWiz) as part of a self-extracting archive; that is permitted without inclusion of this license, as long as the normal SFX banner has not been removed from the binary or disabled.

3. Altered versions--including, but not limited to, ports to new operating systems, existing ports with new graphical interfaces, and dynamic, shared, or static library versions--must be plainly marked as such and must not be misrepresented as being the original source. Such altered versions also must not be misrepresented as being Info-ZIP releases--including, but not limited to, labeling of the altered versions with the names "Info-ZIP" (or any variation thereof, including, but not limited to, different capitalizations), "Pocket UnZip," "WiZ" or "MacZip" without the explicit permission of Info-ZIP. Such altered versions are further prohibited from misrepresentative use of the Zip-Bugs or Info-ZIP e-mail addresses or of the Info-ZIP URL(s).

4. Info-ZIP retains the right to use the names "Info-ZIP," "Zip," "UnZip," "UnZipSFX," "WiZ," "Pocket UnZip," "Pocket Zip," and "MacZip" for its own source and binary releases.

gdal/ogr/ogrsf_frmts/dxf/intronurbs.cpp

This code is derived from the code associated with the book "An Introduction to NURBS" by David F. Rogers. More information on the book and the code is available at:

http://www.nar-associates.com/nurbs/

Copyright (c) 2009, David F. Rogers

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the David F. Rogers nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.31 Google Guava

Guava is licensed under the Apache License, Version 2.0

Copyright 2006 - 2011 Google, Inc. All rights reserved.

A.32 Google Guice

Guice is licensed under the Apache License, Version 2.0

Copyright 2006 – 2011 Google, Inc. All rights reserved.

A.33 Google protobuf

Copyright 2008, Google Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN

ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.34 int64-native

Copyright (c) 2014 Robert Kieffer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.35 Jackson

Copyright 2009 FasterXML, LLC

Jackson is available under the Apache License, Version 2.0.

A.36 Jansi

Copyright (C) 2009, Progress Software Corporation and/or its subsidiaries or affiliates.

Jansi is available under the Apache License, Version 2.0.

A.37 JCodec

This software is based in part on the work of the Independent JPEG Group.

All files except two are available under the FreeBSD license:

http://www.jcodec.org/lic.html

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

----

1 file (StringUtils.java) is "borrowed from Apache". This file is from Apache Commons Lang which is licensed under Apache 2.0

http://www.apache.org/licenses/LICENSE-2.0

----

1 file (VP8DCT.java) refers to Independent JPEG Group) which has the following license (note - the configuration scripts and GIF code mentioned by the license are not included):

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.

This software is copyright (C) 1991-2014, Thomas G. Lane, Guido Vollbeding.

All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions:

(1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation.

(2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group".

(3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us.

Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software".

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

The Unix configuration script "configure" was produced with GNU Autoconf.

It is copyright by the Free Software Foundation but is freely distributable.

The same holds for its supporting scripts (config.guess, config.sub, ltmain.sh). Another support script, install-sh, is copyright by X Consortium but is also freely distributable.

The IJG distribution formerly included code to read and write GIF files. To avoid entanglement with the Unisys LZW patent (now expired), GIF reading support has been removed altogether, and the GIF writer has been simplified to produce "uncompressed GIFs". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders.

We are required to state that "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

A.38 Jettison

Copyright 2006 Envoi Solutions LLC.

Jettison is available under the Apache License, Version 2.0.

A.39 JLine

Copyright (c) 2002-2006, Marc Prud'hommeaux <mwp1@cornell.edu>

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of JLine nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.40 Javassist

Copyright 1999-2015 by Shigeru Chiba.

the contents of this software may be used under the terms of the Apache License Version 2.0.

A.41 json-bignum

Copyright (c) 2012-2013 Datalanche, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.42 Jung

THE JUNG LICENSE

Copyright (c) 2003-2004, Regents of the University of California and the JUNG Project All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the University of California nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.43 Log4js

This product includes software developed by the Apache Software Foundation (http://www.apache.org).

Copyright 2015 Gareth Jones (with contributions from many other people)

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SEMVER 4.3.6 license:

The ISC License

Copyright (c) Isaac Z. Schlueter and Contributors

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

----

readable-stream 1.0.33 license:

Copyright Joyent, Inc. and other Node contributors. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

----

core-util-is 1.0.2 license:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

----

inherits 2.0.1 license:

The ISC License

Copyright (c) Isaac Z. Schlueter

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

----

isarray 0.0.1 license:

(MIT)

Copyright (c) 2013 Julian Gruber <julian@juliangruber.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

----

string_decoder 0.10.31 license

Copyright Joyent, Inc. and other Node contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the

following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.44 MessagePack

Copyright (C) 2008-2010 FURUHASHI Sadayuki

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

A.45 Netty

The Netty Project

=================

Please visit the Netty web site for more information:

http://netty.io/

Copyright 2011 The Netty Project

The Netty Project licenses this file to you under the Apache License, version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Also, please refer to each LICENSE.<component>.txt file, which is located in the 'license' directory of the distribution file, for the license terms of the components that this product depends on.

This product contains the extensions to Java Collections Framework which has been derived from the works by JSR-166 EG, Doug Lea, and Jason T. Greene:

* LICENSE:

* license/LICENSE.jsr166y.txt (Public Domain)

* HOMEPAGE:

* http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/

* http://viewvc.jboss.org/cgi-bin/viewvc.cgi/jbosscache/experimental/jsr166/

This product contains a modified version of Robert Harder's Public Domain Base64 Encoder and Decoder, which can be obtained at:

* LICENSE:

* license/LICENSE.base64.txt (Public Domain)

* HOMEPAGE:

* http://iharder.sourceforge.net/current/java/base64/

This product contains a modified version of 'JZlib', a re-implementation of zlib in pure Java, which can be obtained at:

* LICENSE:

* license/LICENSE.jzlib.txt (BSD Style License)

* HOMEPAGE:

* http://www.jcraft.com/jzlib/

Copyright (c) 2000-2011 ymnk, JCraft,Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,

this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in

the documentation and/or other materials provided with the distribution.

3. The names of the authors may not be used to endorse or promote products

derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JCRAFT, INC. OR ANY CONTRIBUTORS TO THIS SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product optionally depends on 'Protocol Buffers', Google's data interchange format, which can be obtained at:

* LICENSE:

* license/LICENSE.protobuf.txt (New BSD License)

* HOMEPAGE:

* http://code.google.com/p/protobuf/

This product optionally depends on 'SLF4J', a simple logging facade for Java, which can be obtained at:

* LICENSE:

* license/LICENSE.slf4j.txt (MIT License)

* HOMEPAGE:

* http://www.slf4j.org/

This product optionally depends on 'Apache Commons Logging', a logging framework, which can be obtained at:

* LICENSE:

* license/LICENSE.commons-logging.txt (Apache License 2.0)

* HOMEPAGE:

* http://commons.apache.org/logging/

This product optionally depends on 'Apache Log4J', a logging framework, which can be obtained at:

* LICENSE:

* license/LICENSE.log4j.txt (Apache License 2.0)

* HOMEPAGE:

* http://logging.apache.org/log4j/

This product optionally depends on 'JBoss Logging', a logging framework, which can be obtained at:

* LICENSE:

* license/LICENSE.jboss-logging.txt (GNU LGPL 2.1)

* HOMEPAGE:

* http://anonsvn.jboss.org/repos/common/common-logging-spi/

This product optionally depends on 'Apache Felix', an open source OSGi framework implementation, which can be obtained at:

* LICENSE:

* license/LICENSE.felix.txt (Apache License 2.0)

* HOMEPAGE:

* http://felix.apache.org/

This product optionally depends on 'Webbit', a Java event based WebSocket and HTTP server:

* LICENSE:

* license/LICENSE.webbit.txt (BSD License)

* HOMEPAGE:

* https://github.com/joewalnes/webbit

A.46 Node.js

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

****

Copyright Joyent, Inc. and other Node contributors. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND

****

Node.js also includes a number of externally maintained third-party dependencies, including the following:

--c-ares v '1.10.1-DEV'

--http-parser v '2.5.2'

--libuv v '1.8.0'

----tree.h

----inet_pton, inet_ntop

----stdint-msvc2008

----pthread-fixes.hs

----android-ifaddrs.h, android-ifaddrs.c

--OpenSSL v '1.0.2g'

--Punnycode.js

--v8 v '4.5.103.35'

----PCRE test suite

----Layout tests

----Strongtalk assembler

----Valgrind client API header

--zlib v '1.2.8'

"""

The following licensees apply to these externally maintained dependencies:

- c-ares is licensed as follows:

"""

- HTTP Parser is licensed as follows:

http_parser.c is based on src/http/ngx_http_parse.c from NGINX copyright Igor Sysoev.

Additional changes are licensed under the same terms as NGINX and copyright Joyent, Inc. and other Node contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

""""

- libuv is licensed as follows:

""""

libuv is part of the Node project: http://nodejs.org/

libuv may be distributed alone under Node's license:

====

Copyright Joyent, Inc. and other Node contributors. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

====

This license applies to all parts of libuv that are not externally maintained libraries.

The externally maintained libraries used by libuv are:

- tree.h (from FreeBSD), copyright Niels Provos. Two clause BSD license.

- inet_pton and inet_ntop implementations, contained in src/inet.c, are copyright the Internet Systems Consortium, Inc., and licensed under the ISC license.

- stdint-msvc2008.h (from msinttypes), copyright Alexander Chemeris. Three clause BSD license.

- pthread-fixes.h, pthread-fixes.c, copyright Google Inc. and Sony Mobile Communications AB. Three clause BSD license.

- android-ifaddrs.h, android-ifaddrs.c, copyright Berkeley Software Design Inc, Kenneth MacKay and Emergya (Cloud4all, FP7/2007-2013, grant agreement n° 289016). Three clause BSD license.

- OpenSSL, located at deps/openssl, is licensed as follows:

```
/*
====================================================================
* Copyright (c) 1998-2016 The OpenSSL Project. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in
* the documentation and/or other materials provided with the
* distribution.
*
* 3. All advertising materials mentioning features or use of this
* software must display the following acknowledgment:
* "This product includes software developed by the OpenSSL Project
* for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
*
* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
* endorse or promote products derived from this software without
* prior written permission. For written permission, please contact
* openssl-core@openssl.org.
*
* 5. Products derived from this software may not be called "OpenSSL"
* nor may "OpenSSL" appear in their names without prior written
```

* permission of the OpenSSL Project.

*

* 6. Redistributions of any form whatsoever must retain the following

* acknowledgment:

* "This product includes software developed by the OpenSSL Project

* for use in the OpenSSL Toolkit (http://www.openssl.org/)"

*

* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY

* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR

* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT

* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,

* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED

* OF THE POSSIBILITY OF SUCH DAMAGE.

* ==

*

* This product includes cryptographic software written by Eric Young

* (eay@cryptsoft.com). This product includes software written by Tim

* Hudson (tjh@cryptsoft.com).

*

*/ """

- Punycode.js, located at lib/punycode.js, is licensed as follows:

Copyright Mathias Bynens <https://mathiasbynens.be/>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

""""

- V8, located at deps/v8, is licensed as follows:

This license applies to all parts of V8 that are not externally maintained libraries. The externally maintained libraries used by V8 are:

- PCRE test suite, located in test/mjsunit/third_party/regexp-pcre/regexp-pcre.js. This is based on the test suite from PCRE-7.3, which is copyrighted by the University of Cambridge and Google, Inc. The copyright notice and license are embedded in regexp-pcre.js.

(/ PCRE LICENCE

// ------------

//

// PCRE is a library of functions to support regular expressions whose syntax

// and semantics are as close as possible to those of the Perl 5 language.

//

// Release 7 of PCRE is distributed under the terms of the "BSD" licence, as

// specified below. The documentation for PCRE, supplied in the "doc"

// directory, is distributed under the same terms as the software itself.

//

// The basic library functions are written in C and are freestanding. Also

// included in the distribution is a set of C++ wrapper functions.)

- Layout tests, located in test/mjsunit/third_party/object-keys. These are based on layout tests from webkit.org which are copyrighted by Apple Computer, Inc. and released under a 3-clause BSD license.

- Strongtalk assembler, the basis of the files assembler-arm-inl.h, assembler-arm.cc, assembler-arm.h, assembler-ia32-inl.h, assembler-ia32.cc, assembler-ia32.h, assembler-x64-inl.h, assembler-x64.cc, assembler-x64.h, assembler-mips-inl.h, assembler-mips.cc, assembler-mips.h, assembler.cc and assembler.h. This code is copyrighted by Sun Microsystems Inc. and released under a 3-clause BSD license.

- Valgrind client API header, located at third_party/valgrind/valgrind.h This is release under the BSD license.

These libraries have their own licenses; we recommend you read them, as their terms may differ from the terms below.

Further license information can be found in LICENSE files located in sub-directories.

Copyright 2014, the V8 project authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- zlib, located at deps/zlib, is licensed as follows:

"""

zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.8, April 28th, 2013

Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly Mark Adler

jloup@gzip.org madler@alumni.caltech.edu

"""

ISC lisence for inet-pton and inet-ntop:

ISC License (ISC)

Copyright (c) 4-digit year, Company or Person's Name <E-mail address>

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

A.47 node-zookeeper-client

note-zookeeper-client, version 0.2.2, is licensed under the following terms:

Copyright (c) 2013 Yahoo! Inc. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

node-zokeeper-client also comes with two related components, async v0.2.10 and underscore v1.4.4.

License for async v0.2.10:

Copyright (c) 2010-2016 Caolan McMahon

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

License for underscore v1.4.4:

Copyright (c) 2009-2016 Jeremy Ashkenas, DocumentCloud and Investigative

Reporters & Editors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.48 OpenCV

IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.

By downloading, copying, installing or using the software you agree to this license. If you do not agree to this license, do not download, install, copy or use the software.

License Agreement

For Open Source Computer Vision Library

Copyright (C) 2000-2008, Intel Corporation, all rights reserved.

Copyright (C) 2008-2011, Willow Garage Inc., all rights reserved.

Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* The name of the copyright holders may not be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

In no event shall the Intel Corporation or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

A.49 rxjava-core

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. A copy of the license is also reproduced in this document.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

A.50 Slf4j

Copyright (c) 2004-2011 QOS.ch

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.51 Spoofax

Copyright 2016 Delft University of Technology

This project includes software developed at the Programming Languages Group at Delft University of Technology (http://www.tudelft.nl).

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

A.52 Tinkerpop Blueprints

Copyright (c) 2009-2012, TinkerPop [http://tinkerpop.com]

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the TinkerPop nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TINKERPOP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.53 Tinkerpop Gremlin

Copyright (c) 2009-2012, TinkerPop [http://tinkerpop.com]

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the TinkerPop nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TINKERPOP BE LIABLE FOR ANY DIRECT,

INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.54 Tinkerpop Pipes

Copyright (c) 2009-2012, TinkerPop [http://tinkerpop.com]

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the TinkerPop nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TINKERPOP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

B

Hive Spatial Functions

This appendix provides reference information about the Hive spatial functions.

To use these functions, you must understand the concepts and techniques described in Oracle Big Data Spatial Vector Hive Analysis, especially Using the Hive Spatial API.

The functions are presented alphabetically. However, they can be grouped into the following logical categories: types, single-geometry functions, and two-geometry functions.

Types:

- ST_Geometry
- ST_LineString
- ST_MultiLineString
- ST_MultiPoint
- ST_MultiPolygon
- ST_Point
- ST_Polygon

Single-geometry functions:

- ST_Area
- ST_AsWKB
- ST_AsWKT
- ST_Buffer
- ST_ConvexHull
- ST_Envelope
- ST_Length
- ST_Simplify
- ST_SimplifyVW
- ST_Volume

Two-geometry functions:

- ST_AnyInteract

- ST_Contains
- ST_Distance
- ST_Inside

B.1 ST_AnyInteract

Format

```
ST_AnyInteract(
  geometry1  ST_Geometry,
  geometry1  ST_Geometry,
  tolerance  NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Determines if `geometry1` has any spatial interaction with `geometry2`, returning `true` or `false`.

Parameters

geometry1
A 2D or 3D geometry object.

geometry2
Another 2D or 3D geometry object.

tolerance
Tolerance at which `geometry2` is valid.

Usage Notes

Both geometries must have the same number of dimensions (2 or 3) and the same spatial reference system (SRID, or coordinate system).

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_AnyInteract(
  ST_Point('{ "type": "Point", "coordinates": [2, 3]}', 8307),
  ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [5, 2], [5, 6], [1, 6],
[1, 2]]]}', 8307))
from hivetable LIMIT 1;
-- return true
```

B.2 ST_Area

Format

```
ST_Area(
  geometry   ST_Geometry
  tolerance  NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Returns the area of a polygon or multipolygon geometry.

Parameters

geometry
An ST_Geometry object.

tolerance
Value reflecting the distance that two points can be apart and still be considered the same.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Area(ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [5, 2], [5,
7], [1, 7], [1, 2]]]}', 0))
  from hivetable LIMIT 1;  -- return 20
```

B.3 ST_AsWKB

Format

```
ST_AsWKB(
  geometry ST_Geometry);
```

Description

Returns the well-known binary (WKB) representation of the geometry.

Parameters

geometry
An ST_Geometry object.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_AsWKB( ST_Point('{ "type": "Point", "coordinates": [0, 5]}', 8307))
   from hivetable LIMIT 1;
```

B.4 ST_AsWKT

Format

```
ST_AsWKT(
  geometry ST_Geometry);
```

Description

Returns the well-known text (WKT) representation of the geometry.

Parameters

geometry
An ST_Geometry object.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_AsWKT(ST_Point('{ "type": "Point", "coordinates": [0, 5]}', 8307))
  from hivetable LIMIT 1;
```

B.5 ST_Buffer

Format

```
ST_Buffer(
  geometry     ST_Geometry,
  bufferWidth  NUMBER,
  arcTol       NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Generates a new ST_Geometry object that is the buffered version of the input geometry.

Parameters

geometry
Any 2D geometry object. If the geometry is geodetic, it is interpreted as longitude/latitude values in the WGS84 spatial reference system, and `bufferWidth` and `tolerance` are interpreted as meters.

bufferWidth
The distance value used for the buffer.

arcTol
Tolerance used for geodetic arc densification. (Ignored for nongeodetic geometries.)

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Buffer(ST_Point('{ "type": "Point", "coordinates": [0, 5]}', 0), 3)
  from hivetable LIMIT 1;
```

```
-- return {"type":"Polygon", "coordinates":[[[-3,5],[0,2],[3,5]]],"crs":
{"type":"name","properties":{"name":"EPSG:0"}}}
```

B.6 ST_Contains

Format

```
ST_Contains(
  geometry1  ST_Geometry,
  geometry1  ST_Geometry,
  tolerance  NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Determines if `geometry1` contains `geometry2`, returning `true` or `false`.

Parameters

geometry1
A polygon or solid geometry object.

geometry2
Another 2D or 3D geometry object.

tolerance
Tolerance at which `geometry2` is valid.

Usage Notes

Both geometries must have the same number of dimensions (2 or 3) and the same spatial reference system (SRID, or coordinate system).

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Contains(
  ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [5, 2], [5, 6], [1, 6],
[1, 2]]]}', 8307),
  ST_Point('{ "type": "Point", "coordinates": [2, 3]}', 8307))
from hivetable LIMIT 1;
-- return true
```

B.7 ST_ConvexHull

Format

```
ST_ConvexHull(
  geometry ST_Geometry);
```

Description

Returns the convex hull of the input geometry as an ST_Geometry object.

Parameters

geometry
A 2D ST_Geometry object.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_ConvexHull(
  ST_MultiPoint(' { "type": "MultiPoint","coordinates": [ [1, 2], [-1, -2], [5,
6] ] }', 0))
from hivetable LIMIT 1;
-- return {"type":"Polygon", "coordinates":[[[5,6],[1,2],[-1,-2],[5,6]]],"crs":
{"type":"name","properties":{"name":"EPSG:0"}}}
```

B.8 ST_Distance

Format

```
ST_Distance(
  geometry1  ST_Geometry,
  geometry1  ST_Geometry,
  tolerance  NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Determines the distance between two 2D geometries.

Parameters

geometry1
A 2D geometry object.

geometry2
A 2D geometry object.

tolerance
Tolerance at which `geometry2` is valid.

Usage Notes

This function returns thedistance between the two given geometries. For projected data, the distance is in the same unit as the unit of projection. For geodetic data, the distance is in meters.

If an error occurs, the function returns -1.

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Distance(
  ST_Point('{ "type": "Point", "coordinates": [0, 0]}', 0),
  ST_Point('{ "type": "Point", "coordinates": [6, 8]}', 0))
from hivetable LIMIT 1;
-- return 10.0
```

B.9 ST_Envelope

Format

```
ST_Envelope(
  geometry ST_Geometry);
```

Description

Returns the envelope (bounding polygon) of the input geometry as an ST_Geometry object.

Parameters

geometry
A 2D ST_Geometry object.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Envelope(
  ST_MultiPoint(' { "type": "MultiPoint","coordinates": [ [1, 2], [-1, -2], [5,
6] ] }', 0))
from hivetable LIMIT 1;
-- return {"type":"Polygon", "coordinates":[[[-1,-2],[5,-2],[5,6],[-1,6],
[-1,-2]]],"crs":{"type":"name","properties":{"name":"EPSG:0"}}}
```

B.10 ST_Geometry

Format

```
ST_GEOMETRY(
  geometry STRING
  srid INT);
```

or

```
ST_GEOMETRY(
  geometry BINARY
  srid INT);
```

or

```
ST_GEOMETRY(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a GeoJSON string representation of the geometry, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a point using GeoJSON
select ST_Geometry (' { "type": "Point", "coordinates": [100.0, 0.0]}', 8307) from
hivetable LIMIT 1;
-- creates a point using WKT
select ST_Geometry ('point(100.0 0.0)', 8307) from hivetable LIMIT 1;
-- creates the geometries using a HiveRecordInfoProvider
select ST_Geometry (geoColumn, ‘hive.samples.SampleHiveRecordInfoProviderImpl’) from
hivetable;
```

B.11 ST_Inside

Format

```
ST_Inside(
  geometry1  ST_Geometry,
  geometry1  ST_Geometry,
  tolerance  NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Determines if `geometry1` is inside `geometry2`, returning `true` or `false`.

Parameters

geometry1
A 2D or 3D geometry object.

geometry2
A polygon or solid geometry object.

tolerance
Tolerance at which `geometry1` is valid.

Usage Notes

Both geometries must have the same number of dimensions (2 or 3) and the same spatial reference system (SRID, or coordinate system).

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Inside(
  ST_Point('{ "type": "Point", "coordinates": [2, 3]}', 8307),
  ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [5, 2], [5, 6], [1, 6],
[1, 2]]]}', 8307))
from hivetable LIMIT 1;
-- return true
```

B.12 ST_Length

Format

```
ST_Length(
  geometry   ST_Geometry
  tolerance  NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Returns the length of a line or polygon geometry.

Parameters

geometry
An ST_Geometry object.

tolerance
Value reflecting the distance that two points can be apart and still be considered the same.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Length(ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [5, 2], [5,
6], [1, 6], [1, 2]]]}', 0))
  from hivetable LIMIT 1;  -- return 16
```

B.13 ST_LineString

Format

```
ST_LineString(
  geometry STRING
  srid INT);
```

or

```
ST_LineString(
  geometry BINARY
  srid INT);
```

or

```
ST_LineString(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a line string geometry in GeoJSON format, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a line using GeoJSON
select ST_LineString (' { "type": "LineString","coordinates": [ [100.0, 0.0],
[101.0, 1.0] ]} ', 8307) from hivetable LIMIT 1;
-- creates a line using WKT
select ST_LineString (' linestring(1 1, 5 5, 10 10, 20 20)', 8307) from hivetable
LIMIT 1;
-- creates the lines using a HiveRecordInfoProvider
select ST_LineString (geoColumn, ‘mypackage.hiveRecordInfoProviderImpl’) from
hivetable;
```

B.14 ST_MultiLineString

Format

```
ST_MultiLineString(
  geometry STRING
  srid INT);
```

or

```
ST_MultiLineString(
  geometry BINARY
  srid INT);
```

or

```
ST_MultiLineString(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a multiline string geometry in GeoJSON format, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a MultiLineString using GeoJSON
select ST_ MultiLineString (' { "type": "MultiLineString","coordinates": [ [ [100.0,
0.0], [101.0, 1.0] ], [ [102.0, 2.0], [103.0, 3.0] ]] }', 8307) from hivetable LIMIT
1;
-- creates a MultiLineString using WKT
select ST_ MultiLineString ('multilinestring ((10 10, 20 20, 10 40),
(40 40, 30 30, 40 20, 30 10))', 8307) from hivetable LIMIT 1;
-- creates MultiLineStrings using a HiveRecordInfoProvider
select ST_ MultiLineString (geoColumn, ‘mypackage.hiveRecordInfoProviderImpl’) from
hivetable;
```

B.15 ST_MultiPoint

Format

```
ST_MultiPoint(
  geometry STRING
  srid INT);
```

or

```
ST_MultiPoint(
  geometry BINARY
  srid INT);
```

or

```
ST_MultiPoint(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a multipoint geometry in GeoJSON format, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a MultiPoint using GeoJSON
select ST_MultiPoint (' { "type": "MultiPoint","coordinates": [ [100.0, 0.0],
[101.0, 1.0] ] }', 8307) from hivetable LIMIT 1;
-- creates a MultiPoint using WKT
select ST_ MultiPoint ('multipoint ((10 40), (40 30), (20 20), (30 10))', 8307) from
hivetable LIMIT 1;
-- creates MultiPoints using a HiveRecordInfoProvider
select ST_ MultiPoint (geoColumn, ‘mypackage.hiveRecordInfoProviderImpl’) from
hivetable;
```

B.16 ST_MultiPolygon

Format

```
ST_MultiPolygon(
  geometry STRING
  srid INT);
```

or

```
ST_MultiPolygon(
  geometry BINARY
  srid INT);
```

or

```
ST_MultiPolygon(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a multipolygon geometry in GeoJSON format, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a MultiPolygon using GeoJSON
select ST_ MultiPolygon (' { "type": "MultiPolygon","coordinates": [[[[102.0, 2.0],
[103.0, 2.0], [103.0, 3.0], [102.0, 3.0], [102.0, 2.0]]], [[[100.0, 0.0], [101.0,
0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]], [[100.2, 0.2], [100.8, 0.2],
[100.8, 0.8], [100.2, 0.8], [100.2, 0.2]]] ] }', 8307) from hivetable LIMIT 1;
-- creates a MultiPolygon using WKT
select ST_ MultiPolygon ('multipolygon(((30 20, 45 40, 10 40, 30 20)),
((15 5, 40 10, 10 20, 5 10, 15 5)))', 8307) from hivetable LIMIT 1;
-- creates MultiPolygons using a HiveRecordInfoProvider
select ST_ MultiPolygon (geoColumn, ‘mypackage.hiveRecordInfoProviderImpl’) from
hivetable;
```

B.17 ST_Point

Format

```
ST_Point(
  geometry STRING
  srid INT);
```

or

```
ST_Point(
  geometry BINARY
  srid INT);
```

or

```
ST_Point(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a point geometry in GeoJSON format, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a point using GeoJSON
select ST_Point (' { "type": "Point", "coordinates": [100.0, 0.0]}', 8307) from
hivetable LIMIT 1;
-- creates a point using WKT
select ST_Point ('point(100.0 0.0)', 8307) from hivetable LIMIT 1;
-- creates the points using a HiveRecordInfoProvider
select ST_Point (geoColumn, ‘hive.samples.SampleHiveRecordInfoProviderImpl’) from
hivetable;
```

B.18 ST_Polygon

Format

```
ST_Polygon(
  geometry STRING
  srid INT);
```

or

```
ST_Polygon(
  geometry BINARY
  srid INT);
```

or

```
ST_Polygon(
  geometry Object
  hiveRecordInfoProvider STRING);
```

Description

Creates a polygon geometry in GeoJSON format, and returns a GeoJSON string representation of the geometry.

Parameters

geometry
To create a geometry from a GeoJSON or WKT string (first format): Geometry definition in GeoJSON or WKT format.

To create a geometry from a WKB object (second format): Geometry definition in WKB format.

To create a geometry using a Hive object (third format): Geometry definition in any Hive supported type.

srid
Spatial reference system (coordinate system) identifier.

HiveRecordProvider
The fully qualified name of an implementation of the interface `oracle.spatial.hadoop.vector.hive.HiveRecordInfoProvider` to extract the geometry in GeoJSON format.

Usage Notes

See Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
-- creates a polygon using GeoJSON
select ST_Polygon (' { "type": "Polygon","coordinates": [ [ [100.0, 0.0], [101.0,
0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0] ]] }', 8307) from hivetable LIMIT 1;
-- creates a polygon using WKT
select ST_ Polygon ('polygon((0 0, 10 0, 10 10, 0 0))', 8307) from hivetable LIMIT 1;
-- creates the polygons using a HiveRecordInfoProvider
select ST_ Polygon (geoColumn, ‘mypackage.hiveRecordInfoProviderImpl’) from
hivetable;
```

B.19 ST_Simplify

Format

```
ST_Simplify(
  geometry   ST_Geometry,
  threshold  NUMBER);
```

Description

Generates a new ST_Geometry object by simplifying the input geometry using the Douglas-Peucker algorithm.

Parameters

geometry
Any 2D geometry object. If the geometry is geodetic, it is interpreted as longitude/latitude values in the WGS84 spatial reference system, and `bufferWidth` and `tolerance` are interpreted as meters.

threshold
Threshold value to be used for the geometry simplification. Should be a positive number. (Zero causes the input geometry to be returned.) If the input geometry is geodetic, the value is the number of meters; if the input geometry is non-geodetic, the value is the number of units associated with the data.

As the threshold value is decreased, the generated geometry is likely to be closer to the input geometry; as the threshold value is increased, fewer vertices are likely to be in the returned geometry.

Usage Notes

Depending on the threshold value, a polygon can simplify into a line or a point, and a line can simplify into a point. Therefore, the output object should be checked for type, because the output geometry type might be different from the input geometry type.

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_Simplify(
  ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [1.01, 2.01], [5, 2], [5,
6], [1, 6], [1, 2]]]}', 8307),
  1)
from hivetable LIMIT 1;
-- return {"type":"Polygon", "coordinates":[[[1,2],[5,2],[5,6],[1,6],[1,2]]],"crs":
{"type":"name","properties":{"name":"EPSG:8307"}}}
```

B.20 ST_SimplifyVW

Format

```
ST_SimplifyVW(
  geometry   ST_Geometry,
  threshold  NUMBER);
```

Description

Generates a new ST_Geometry object by simplifying the input geometry using the Visvalingham-Whyatt algorithm.

Parameters

geometry
Any 2D geometry object. If the geometry is geodetic, it is interpreted as longitude/latitude values in the WGS84 spatial reference system, and `bufferWidth` and `tolerance` are interpreted as meters.

threshold
Threshold value to be used for the geometry simplification. Should be a positive number. (Zero causes the input geometry to be returned.) If the input geometry is geodetic, the value is the number of meters; if the input geometry is non-geodetic, the value is the number of units associated with the data.

As the threshold value is decreased, the generated geometry is likely to be closer to the input geometry; as the threshold value is increased, fewer vertices are likely to be in the returned geometry.

Usage Notes

Depending on the threshold value, a polygon can simplify into a line or a point, and a line can simplify into a point. Therefore, the output object should be checked for type, because the output geometry type might be different from the input geometry type.

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select ST_SimplifyVW(
  ST_Polygon('{"type": "Polygon","coordinates": [[[1, 2], [1.01, 2.01], [5, 2], [5,
6], [1, 6], [1, 2]]]}', 8307),
  50)
from hivetable LIMIT 1;
-- return {"type":"Polygon", "coordinates":[[[1,2],[5,6],[1,6],[1,2]]],"crs":
{"type":"name","properties":{"name":"EPSG:8307"}}}
```

B.21 ST_Volume

Format

```
ST_Volume(
  multipolygon  ST_MultiPolygon,
  tolerance     NUMBER DEFAULT 0 (nongeodetic geometries) or 0.05 (geodetic
geometries));
```

Description

Returns the area of a multipolygon 3D geometry. The multipolygon is handled as a solid.

Parameters

multipolygon
An ST_Multipolygon object.

tolerance
Value reflecting the distance that two points can be apart and still be considered the same.

Usage Notes

For projected data, the volume is in the same unit as the unit of projection. For geodetic data, the volume is in cubic meters.

Returns -1 in case of an error.

See also Oracle Big Data Spatial Vector Hive Analysis for conceptual and usage information.

Examples

```
select select ST_Volume(
  ST_MultiPolygon (' { "type": "MultiPolygon", "coordinates":
    [[[[0, 0, 0], [0, 0, 1], [0, 1, 1], [0, 1, 0], [0, 0, 0]]],
    [[[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0], [0, 0, 0]]],
    [[[0, 0, 0], [1, 0, 0], [1, 0, 1], [0, 0, 1], [0, 0, 0]]],
    [[[1, 1, 0], [1, 1, 1], [1, 0, 1], [1, 0, 0], [1, 1, 0]]],
    [[[0, 1, 0], [0, 1, 1], [1, 1, 1], [1, 1, 0], [0, 1, 0]]],
    [[[0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]]]]}',
    0))
  from hivetable LIMIT 1; -- return 1.0
```

Index