

Oracle® Big Data Spatial and Graph

User's Guide and Reference

Release 1 (1.0)

E62124-01

May 2015

Oracle Big Data Spatial and Graph User's Guide and Reference, Release 1 (1.0)

E62124-01

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

Primary Authors: Donna Carver, Harihara Subramanian, Chuck Murray

Contributors: Bill Beauregard, Hector Briseno, Hassan Chafi, Zazhil Herena, Sungpack Hong, Roberto Infante, Hugo Labra, Gabriela Montoya, 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, then 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

Preface	xi
Audience.....	xi
Documentation Accessibility	xi
Related Documents	xi
Conventions	xi

Part I Oracle Big Data Spatial and Graph Overview

1 Big Data Spatial and Graph Overview

About Big Data Spatial and Graph	1-1
Spatial Features.....	1-1
Property Graph Features	1-2
Property Graph Sizing Recommendations.....	1-3
Installing Oracle Big Data Spatial and Graph on an Oracle Big Data Appliance.....	1-3
Installing and Configuring the Big Data Spatial Image Processing Framework	1-3
Installing Image Processing Framework for Oracle Big Data Appliance Distribution.....	1-3
Installing the Image Processing Framework for Other Distributions (Not Oracle Big Data Appliance) 1-4	
Prerequisites for Installing the Image Processing Framework for Other Distributions...	1-4
Installing the Image Processing Framework for Other Distributions.....	1-4
Configuring the environment	1-5
Post-installation Verification of the Image Processing Framework.....	1-6
Image Loading Test Script.....	1-6
Image Processor Test Script.....	1-6
Installing and Configuring the Big Data Spatial Image Server	1-7
Installing and Configuring the Image Server for Oracle Big Data Appliance	1-7
Prerequisites for installing Image Server on Oracle Big Data Appliance.....	1-7
Installing Image Server Web on an Oracle Big Data Appliance	1-8
Configuring the Environment.....	1-8
Installing and Configuring the Image Server Web for Other Systems (Not Big Data Appliance). 1-8	
Prerequisites for Installing the Image Server on Other Systems.....	1-9
Installing the Image Server Web on Other Systems	1-9
Configuring the Environment.....	1-9
Post-installation Verification Example for the Image Server Console	1-9
Loading images from the local server to HDFS Hadoop cluster	1-10

Creating a mosaic image and catalog.....	1-10
Installing Oracle Big Data Spatial Hadoop Vector Console.....	1-11
Assumptions and Prerequisite Libraries	1-11
Assumptions	1-11
Prerequisite Libraries	1-11
Installing Spatial Hadoop Vector Console on Oracle Big Data Appliance.....	1-11
Installing Spatial Hadoop Vector Console for Other Systems (Not Big Data Appliance)....	1-12
Configuring Spatial Hadoop Vector Console on Oracle Big Data Appliance.....	1-12
Configuring Spatial Hadoop Vector Console for Other Systems (Not Big Data Appliance)	1-15
Installing Property Graph Support on a CDH Cluster or Other Hardware	1-15
Apache HBase Prerequisites.....	1-15
Property Graph Installation Steps	1-15
About the Property Graph Installation Directory	1-16
Optional Installation Task for In-Memory Analytics.....	1-16
Installing and Configuring Hadoop.....	1-17
Running In-Memory Analytics on Hadoop	1-17

Part II Big Data Spatial and Graph on Apache Hadoop

2 Using Big Data Spatial and Graph with Spatial Data

About Big Data Spatial and Graph Support for Spatial Data	2-1
What is Big Data Spatial and Graph on Apache Hadoop?	2-1
Advantages of Oracle Big Data Spatial and Graph.....	2-2
Oracle Big Data Spatial Features and Functions.....	2-2
Oracle Big Data Spatial Files, Formats, and Software Requirements.....	2-3
Oracle Big Data Vector and Raster Data Processing	2-3
Oracle Big Data Spatial Raster Data Processing	2-3
Oracle Big Data Spatial Vector Data Processing.....	2-3
Oracle Big Data Spatial Hadoop Image Processing Framework for Raster Data Processing....	2-4
Image Loader	2-5
Image Processor.....	2-5
Image Server	2-6
Loading an Image to Hadoop Using the Image Loader	2-6
Image Loading Job	2-7
Input Parameters	2-7
Output Parameters.....	2-8
Processing an Image Using the Oracle Spatial Hadoop Image Processor.....	2-8
Image Processing Job	2-9
Input Parameters	2-9
Catalog XML Structure.....	2-10
Mosaic definition XML Structure	2-10
Job Execution	2-12
Processing Classes and ImageBandWritable	2-12
Location of the Classes and Jar Files	2-14
Output.....	2-14
Oracle Big Data Spatial Vector Analysis	2-14

Spatial Indexing.....	2-15
Spatial Indexing Class Structure.....	2-15
Configuration for Creating a Spatial Index.....	2-16
Input Formats for Spatial Index.....	2-17
MVSuggest for Locating Records	2-17
Spatial Filtering	2-18
Filtering Records	2-19
Classifying Data Hierarchically	2-20
Changing the Hierarchy Level Range.....	2-25
Controlling the Search Hierarchy.....	2-25
Using MVSuggest to Classify the Data	2-26
Generating Buffers	2-27
RecordInfoProvider	2-28
Sample RecordInfo Implementation	2-29
LocalizableRecordInfoProvider	2-30
HierarchyInfo.....	2-30
Sample HierarchyInfo Implementation.....	2-32
Using JGeometry in MapReduce Jobs.....	2-35
Tuning Performance Data of Job Running Times using Vector Analysis API.....	2-38
Using Oracle Big Data Spatial and Graph Vector Console.....	2-39
Creating a Spatial Index Using the Console.....	2-39
Running a Hierarchy Job Using the Console	2-40
Viewing the Index Results	2-41
Creating Results Manually	2-42
Creating and Deleting templates	2-42
Configuring Templates.....	2-42
Running a Job to Create an Index Using the Command Line	2-43
Running a Job to Perform a Spatial Filtering	2-44
Running a Job to Create a Hierarchy Result	2-44
Running a Job to Generate Buffer.....	2-45
Using Oracle Big Data Spatial and Graph Image Server Console.....	2-46
Loading Images to HDFS Hadoop Cluster to Create a Mosaic.....	2-46

Part III Property Graphs

3 Configuring Property Graph Support

Tuning the Software Configuration	3-1
Tuning Apache HBase for Use With Property Graphs.....	3-1
Modifying the Apache HBase Configuration	3-1
Modifying the Java Memory Settings	3-3
Tuning Oracle NoSQL Database for Use with Property Graphs	3-4

4 Using Property Graphs in a Big Data Environment

About Property Graphs	4-1
What Are Property Graphs?	4-1
What Is Big Data Support for Property Graphs?	4-2

Analytics Layer.....	4-3
Data Access Layer.....	4-3
Storage Management.....	4-3
RESTful Web Services	4-3
Getting Started With Property Graphs	4-3
About Property Graph Data Formats	4-4
GraphML Data Format.....	4-4
GraphSON Data Format.....	4-4
GML Data Format	4-5
Oracle Flat File Format	4-6
Using Java APIs for Property Graph Data.....	4-6
Overview of the Java APIs.....	4-6
Oracle Big Data Spatial and Graph Java APIs	4-7
TinkerPop Blueprints Java APIs	4-7
Apache Hadoop Java APIs	4-7
Oracle NoSQL Database Java APIs	4-7
Apache HBase Java APIs	4-7
Opening and Closing a Property Graph Instance	4-8
Using Oracle NoSQL Database.....	4-8
Using Apache HBase	4-9
Creating the Vertices.....	4-10
Creating the Edges	4-11
Deleting the Vertices and Edges	4-11
Dropping a Property Graph	4-12
Using Oracle NoSQL Database.....	4-12
Using Apache HBase	4-12
Managing Text Indexing for Property Graph Data	4-12
Using Automatic Indexes with the Apache Lucene Search Engine.....	4-13
Using Manual Indexes with the SolrCloud Search Engine.....	4-15
Handling Data Types.....	4-17
Appending Data Type Identifiers on Apache Lucene.....	4-17
Appending Data Type Identifiers on SolrCloud	4-20
Uploading a Collection's SolrCloud Configuration to Zookeeper.....	4-22
Updating Configuration Settings on Text Indexes for Property Graph Data	4-22
Property Graph Support for Secure Oracle NoSQL Database	4-23
Using the Groovy Shell with Property Graph Data	4-24
Exploring the Sample Programs	4-26
About the Sample Programs.....	4-26
Compiling and Running the Sample Programs.....	4-27
About the Example Output	4-27
Example: Creating a Property Graph.....	4-28
Example: Dropping a Property Graph.....	4-29
Examples: Adding and Dropping Vertices and Edges.....	4-29
Oracle Flat File Format Definition	4-31
About the Property Graph Description Files	4-31
Vertex File.....	4-31
Edge File	4-32

Encoding Special Characters	4-33
Example Property Graph in Oracle Flat File Format	4-33
Example Python User Interface	4-34

5 Using In-Memory Analytics

Reading a Graph into Memory	5-1
Starting the In-Memory Analytics Shell	5-1
Using the Shell Help	5-2
Providing Graph Metadata in a Configuration File	5-2
Reading Graph Data into Memory	5-3
Read a Graph Stored in Apache HBase into Memory	5-4
Read a Graph Stored in Oracle NoSQL Database into Memory	5-5
Read a Graph Stored in the Local File System into Memory	5-6
Reading Custom Graph Data	5-7
Creating a Simple Graph File	5-7
Adding a Vertex Property	5-8
Using Strings as Node Identifiers	5-9
Adding an Edge Property	5-10
Storing Graph Data on Disk	5-11
Storing the Results of Analysis in a Node Property	5-11
Storing a Graph in Edge-List Format on Disk	5-12
Executing Built-in Algorithms	5-12
About In-Memory Analytics	5-12
About the Analyst Interface	5-13
Reading the Graph	5-13
Running the Triangle Counting Algorithm	5-14
Running the Pagerank Algorithm	5-15
Creating Subgraphs	5-16
About Filter Expressions	5-16
Using a Simple Filter to Create a Subgraph	5-17
Using a Complex Filter to Create a Subgraph	5-18
Using a Node List to Create a Bipartite Subgraph	5-19
Deploying to Jetty	5-20
About the Authentication Mechanism	5-21
Deploying to Apache Tomcat	5-22
Deploying to Oracle WebLogic Server	5-22
Installing Oracle WebLogic Server	5-23
Deploying In-Memory Analytics	5-23
Verifying That the Server Works	5-23
Connecting to the In-Memory Analytics Server	5-23
Connecting with the In-Memory Analytics Shell	5-23
About Logging HTTP Requests	5-23
Connecting with Java	5-24
Connecting with an HTTP Request	5-24
Reading and Storing Data in HDFS	5-25
Loading Data from HDFS	5-25
Storing Graph Snapshots in HDFS	5-27

Compiling and Running a Java Application in Hadoop	5-27
Running In-Memory Analytics as a YARN Application	5-28
Starting and Stopping In-Memory Analytics Services	5-28
Configuring the In-Memory Analytics YARN Client.....	5-28
Starting a New In-Memory Analytics Service	5-28
About Long-Running In-Memory Analytics Services.....	5-28
Stopping In-Memory Analytics Services	5-28
Connecting to In-Memory Analytics Services	5-29
Monitoring In-Memory Analytics Services	5-29
Using the Java API Inside the In-Memory Analytics Shell	5-29

A Third-Party Licenses for Bundled Software

Apache Licensed Code	A-2
ANTLR 3	A-5
AOP Alliance.....	A-6
Apache Commons CLI.....	A-6
Apache Commons Codec	A-6
Apache Commons Collections.....	A-6
Apache Commons Configuration	A-6
Apache Commons IO.....	A-7
Apache Commons Lang	A-7
Apache Commons Logging	A-7
Apache fluent	A-7
Apache Groovy	A-7
Apache htrace	A-7
Apache HTTP Client.....	A-7
Apache HTTPComponents Core.....	A-7
Apache Jena	A-8
Apache Log4j.....	A-8
Apache Lucene	A-8
Apache Xerces2	A-8
Apache xml-commons	A-8
Cloudera CDH	A-9
Fastutil	A-9
GeoNames Data	A-9
Geospatial Data Abstraction Library (GDAL).....	A-14
Google Guava.....	A-18
Google Guice.....	A-18
Google protobuf	A-18
Jackson.....	A-19
Jansi	A-19
Jettison	A-19
JLine	A-19
Javassist	A-20
Jung	A-20
MessagePack	A-20
Netty.....	A-21

Slf4j	A-23
Tinkerpop Blueprints	A-24
Tinkerpop Gremlin.....	A-24
Tinkerpop Pipes	A-25

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.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Oracle Big Data Spatial and Graph Overview

Part I contains the following chapters:

- [Chapter 1, "Big Data Spatial and Graph Overview"](#)

Big Data Spatial and Graph Overview

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

- [About Big Data Spatial and Graph](#)
- [Spatial Features](#)
- [Property Graph Features](#)
- [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](#)
- [Installing Oracle Big Data Spatial Hadoop Vector Console](#)
- [Installing Property Graph Support on a CDH Cluster or Other Hardware](#)

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.

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 hundreds of 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

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 Rexter REST APIs, 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](#)

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

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 both the spatial and property graph capabilities. You can enable this option during an initial software installation, or afterward using the `bdaccli` 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.

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](#)).

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 steps to ensure that it works.

- Identify the `HADOOP_LIB_PATH` for Oracle Big Data Appliance is under `/opt/cloudera/parcels/CDH/lib/hadoop/lib/`.
- Make the `imageserver` folder under `opt/sharedir/spatial/demo/` write accessible as follows:

```
chmod 777 /opt/sharedir/spatial/demo/imageserver/
```

- Make the `libproj.so` (Proj.4) Cartographic Projections Library accessible to the users, and copy the `libproj.so` to the `HADOOP_LIB_PATH/native` under Resource Manager Node (and any backup Resource Manager Nodes) as follows:

```
cp libproj.so HADOOP_LIB_PATH/native
```

- Create a folder `native` under `/opt/sharedir/spatial/demo/imageserver/` and copy the `libproj.so` and GDAL libraries to that folder as follows:

```
mkdir /opt/sharedir/spatial/demo/imageserver/native
```

```
cp libproj.so /opt/sharedir/spatial/demo/imageserver/native
```

```
cp -R /opt/oracle/oracle-spatial-graph/spatial/gdal/lib/.  
/opt/sharedir/spatial/demo/imageserver/native
```

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.

Prerequisites for Installing the Image Processing Framework for Other Distributions

- Ensure that `HADOOP_LIB_PATH` is under `/usr/lib/hadoop`.
- 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`. This folder is used for running the test scripts and must be pointed to `/opt/sharedir`. If not, then modify the test script accordingly to point to this folder.
- Download `oracle-spatial-graph-1.0-1.x86_64.rpm` from the Oracle e-delivery web site.
- Execute `oracle-spatial-graph-1.0-1.x86_64.rpm` using the `rpm` command.
- After the `rpm` executes, verify that a directory structure created at `/opt/oracle/oracle-spatial-graph/spatial` contains three folders: `jlib`, `gdal` and `demo`.

Installing the Image Processing Framework for Other Distributions

1. Copy the content under `jlib` (bunch of jar files) to `HADOOP_LIB_PATH` directory in every cluster node.
2. In the Resource Manager Node (and any backup Resource Manager Nodes), copy all the content from `/opt/oracle/oracle-spatial-graph/spatial/gdal/lib` to the `HADOOP_LIB_PATH/native` directory as follows:

```
cp -R --preserve=links  
/opt/oracle/oracle-spatial-graph/spatial/gdal/lib/. HADOOP_LIB_  
PATH/native
```

3. In the Resource Manager Node (and any backup Resource Manager Nodes), 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/gdal/data SHARED_FOLDER
cp -R /opt/oracle/oracle-spatial-graph/spatial/gdal/gdalplugins
SHARED_FOLDER
```

4. Create a folder ALL_ACCESS_FOLDER with write access for all users under SHARED_FOLDER. This folder must be named as spatial in order for test scripts to be executed as they are, otherwise modify test script accordingly, as follows:

Go to the shared folder.

```
cd /opt/sharedir
```

Create a spatial folder.

```
mkdir spatial
```

Provide write access.

```
chmod 777 spatial
```

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

```
cp -R /opt/oracle/oracle-spatial-graph/spatial/demo
/opt/sharedir/spatial
```

6. Provide write access to the imageserver folder under demo as follows:

```
chmod 777 /opt/sharedir/spatial/demo/imageserver/
```

7. Copy the user library libproj.so into the HADOOP_LIB_PATH/native as follows:

```
cp libproj.so HADOOP_LIB_PATH/native
```

8. Create a folder native under /opt/sharedir/spatial/demo/imageserver/ and copy the libproj.so and GDAL libraries to that folder as follows:

```
mkdir /opt/sharedir/spatial/demo/imageserver/native
```

```
cp libproj.so /opt/sharedir/spatial/demo/imageserver/native
```

```
cp -R /opt/oracle/oracle-spatial-graph/spatial/gdal/lib/.
/opt/sharedir/spatial/demo/imageserver/native
```

9. Provide read and execute permissions for the libproj.so library to all users as follows:

```
chmod 755 /opt/sharedir/spatial/demo/imageserver/native/libproj.so
```

Configuring the environment

- Set the following GDAL environment variables in the Resource Manager Node (and any backup Resource Manager Nodes):

```
GDAL_DRIVER_PATH=HADOOP_LIB_PATH/native/gdalplugins
```

```
GDAL_DATA=SHARED_FOLDER/data
```

- Create or update the shared libraries to list the GDAL libraries location:

```
LD_LIBRARY_PATH=HADOOP_LIB_PATH/native
```

Post-installation Verification of the Image Processing Framework

Two test scripts are provided, one to test the Image Loading functionality and another to test the Image Processing functionality. Execute these two scripts as mentioned in this section to verify a successful installation of Image Processing Framework.

Image Loading Test Script

This test script loads three Hawaii images into HDFS, and one block is created for each one of them. The test script can be found in the following path:

`/opt/sharedir/spatial/demo/imageserver/runimageloader.sh.`

The script can be executed using the following example.

Note: All users must have write permission to the parent folder
`/opt/sharedir/spatial/demo/imageserver.`

From the command line type `sudo -u hdfs ./runimageloader.sh.`

Upon a successful execution a list of the created images and thumbnails on HDFS should be listed after the following message: Generated ohifs files are with the list of files displayed and followed by this message: Thumbnails created are with the list of thumbnails.

If the three images and its corresponding thumbnails are listed, then the loading process was successful. If this step was reached, the install and configuration executed successfully.

If the installation and configuration was not successful, then the output is not generated and a message Not all the images were uploaded correctly, check for Hadoop logs.

Image Processor Test Script

This test script creates a mosaic with the three pre-loaded Hawaii images. The mosaic created is 1600 x 1447 pixels. The test script can be found at

`/opt/sharedir/spatial/demo/imageserver/runimageprocessor.sh.`

The script can be executed using the following example.

Note: All users must have write permission to the parent folder
`/opt/sharedir/spatial/demo/imageserver.`

From the command line type `sudo -u hdfs ./runimageprocessor.sh.`

Upon a successful execution a message is displayed: Expected output file:
`/opt/sharedir/spatial/processtest/littlemap.tif.`

If the installation and configuration was successful, then the output is generated with a message Mosaic image generated.

If the installation and configuration was not successful, then the output is not generated and a message Mosaic was not successfully created, check for Hadoop logs.

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](#)).

Installing and Configuring the Image Server for Oracle Big Data Appliance

Follow the instructions in this topic.

- [Prerequisites for installing Image Server on Oracle Big Data Appliance](#)
- [Installing Image Server Web on an Oracle Big Data Appliance](#)
- [Configuring the Environment](#)

Prerequisites for installing Image Server on Oracle Big Data Appliance

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/
```

4. Edit the `$JETTY_HOME/start.ini` file and change the property `jsp-impl=apache` to `jsp-impl=glassfish` optionally. You can download these jars from <http://mvnrepository.com/> or another Apache jar provider:

```
xalan-2.7.1.jar
```

```
xercesImpl-2.11.0.jar
```

```
xml-apis-1.4.01.jar
```

```
serializer-2.7.1.jar
```

5. Copy these jars to `$JETTY_HOME/lib/apache-jsp`.
6. Check the version by running: `$JETTY_HOME/java -jar start.jar -version`

Installing 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 `$JETTY_HOME/lib/ext`.
2. Copy the
`/opt/oracle/oracle-spatial-graph/spatial/conf/jetty-imgserver-realm.properties` file to `$JETTY_HOME/etc` folder
3. Edit the `$JETTY_HOME/etc/jetty-imgserver-realm.properties` file to add a password and role
 - a. Remove the `<password>` and type a new password.
 - b. Remove the `<>` from the `<admin_role>` text and keep the `admin_role`.
4. Start the jetty server by running: `java -jar $JETTY_HOME/start.jar`.

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 Image Server Web on an Oracle Big Data Appliance"](#) on page 1-8.
3. From the Configuration tab in the Hadoop Configuration Parameters section, depending on the cluster configuration change these 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. `<shcedulename>: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.

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](#)

Prerequisites for Installing the Image Server on Other Systems

- Follow the instructions specified in ["Prerequisites for Installing the Image Processing Framework for Other Distributions"](#) on page 1-4.
- Follow the instructions specified in ["Installing the Image Processing Framework for Other Distributions"](#) on page 1-4.
- Follow the instructions specified in ["Configuring the environment"](#) on page 1-5.

Installing the Image Server Web on Other Systems

- Follow the instructions specified in ["Prerequisites for installing Image Server on Oracle Big Data Appliance"](#) on page 1-7.
- Follow the instructions specified in ["Installing Image Server Web on an Oracle Big Data Appliance"](#) on page 1-8.
- Follow the instructions specified in ["Configuring the Environment"](#) on page 1-8.

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 Image Server Web on an Oracle Big Data Appliance"](#) on page 1-8.
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.

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 HDFS Hadoop cluster](#)
- [Creating a mosaic image and catalog](#)

Loading images from the local server to 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/sharedir/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.

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/sharedir/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**.

Installing 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 \(Not Big Data Appliance\)](#)

Assumptions and Prerequisite Libraries

The following assumptions and prerequisites are a must for installing and configure the Spatial Hadoop Vector Console.

Assumptions

- The API and jobs described here runs on a CDH5 or similar Hadoop environment.
- Java 6 or newer versions are present in your environment.
- MVSuggest must be installed separately. Download it from the <http://www.oracle.com/technetwork/index.html> site.

Prerequisite Libraries

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

- Hadoop environment jars
- sdohadoop-vector.jar
- sdoutil.jar
- sdohadoop-vector-demo.jar
- sdoapi.jar
- ojdbc.jar

Installing Spatial Hadoop Vector Console on Oracle Big Data Appliance

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 or any other directory of choice as follows:

```
unzip /opt/oracle/oracle-spatial-graph/spatial/jlib/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](#)" on page 1-12.
5. Start the jetty server. \$JETTY_HOME/java -jar start.jar

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

Follow the steps mentioned in "[Installing Spatial Hadoop Vector Console on Oracle Big Data Appliance](#)" on page 1-11. However, in step 3 replace the path /opt/cloudera/parcels/CDH/lib/ with /usr/lib/.

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 to send mails and change the directory used by the console to build temporary hierarchical indexes.

You can as well change the properties of the Hadoop jobs. The configuration parameters are:

- 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. Directory with temporary hierarchical indexes: An HDFS path that will contain temporary data on hierarchical relationships. This is an example settings:
<hierarchydataindexpath>hdfs://hadoop.cluster.url:8020/user/myuser/hierarchyIndexPath</hierarchydataindexpath>
- c. General 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/cloud
era/parcels/CDH/lib/hadoop/lib/*,/opt/cloudera/parcels/CDH/lib/hadoop-hdfs
/*,/opt/cloudera/parcels/CDH/lib/hadoop-hdfs/lib/*,/opt/cloudera/parcels/C
DH/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 given here:

```

<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 results. The same applies for the second job. The following is an example 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>

```

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" on page 1-12. However, in step 1 C (General Hadoop Job Configuration), in the Hadoop property `yarn.application.classpath` replace the `/opt/cloudera/parcels/CDH/lib/` with the actual library path, by default `/usr/lib/`.

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 Analytics](#)

See Also: [Chapter 3, "Configuring Property Graph Support"](#)

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.

Property Graph Installation Steps

To install property graph support, follow these steps.

1. Unzip the software package:

```
rpm -i oracle-spatial-graph-1.0-1.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.

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
```

Optional Installation Task for In-Memory Analytics

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 in-memory analytics and write the results back to the HDFS, and/or use Hadoop NextGen MapReduce (YARN) scheduling to start, monitor and stop in-memory analytics

- [Installing and Configuring Hadoop](#)

■ Running In-Memory Analytics on Hadoop

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.

Running In-Memory Analytics on Hadoop

When running a Java application using in-memory analytics and HDFS, make sure that `$HADOOP_HOME/etc/hadoop` is on the classpath, so that the configurations get picked up by the Hadoop client libraries. However, you do not need to do this when using the In-Memory Analytics Shell, because it adds `$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 In-Memory Analytics 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 In-Memory Analytics as a YARN service. For further information about Hadoop, refer to the CDH 5.2.x documentation.

Part II

Big Data Spatial and Graph on Apache Hadoop

Part II contains the following chapters:

- [Chapter 2, "Using Big Data Spatial and Graph with Spatial Data"](#)

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](#)
- [Oracle Big Data Spatial Vector Analysis](#)
- [Using Oracle Big Data Spatial and Graph Vector Console](#)
- [Using Oracle Big Data Spatial and Graph Image Server Console](#)

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](#)

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.

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

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.

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.

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

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](#)

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.

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](#)" on page 2-4.

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

Two different data service operations are supported for the spatial vector data:

- Data enrichment

- Data categorization

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](#)" on page 2-14.

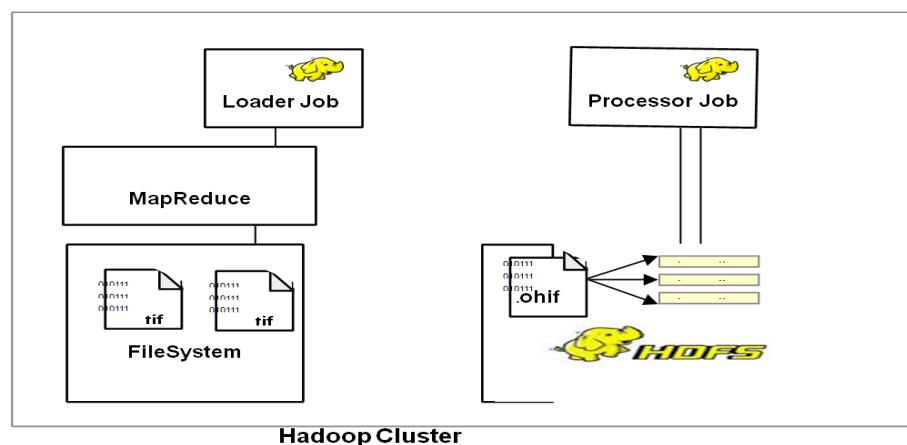
Oracle Big Data Spatial Hadoop Image Processing Framework for Raster Data Processing

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 image
- Subset and user-defined operations processed in parallel using the MapReduce framework
- Ability to add custom processing classes to be executed 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

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 cluster, as represented in the following diagram. Also, you can load and process the images using the Image Server web application.

- [Image Loader](#)
- [Image Processor](#)
- [Image Server](#)



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](#)

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 tile's cover area forms 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:

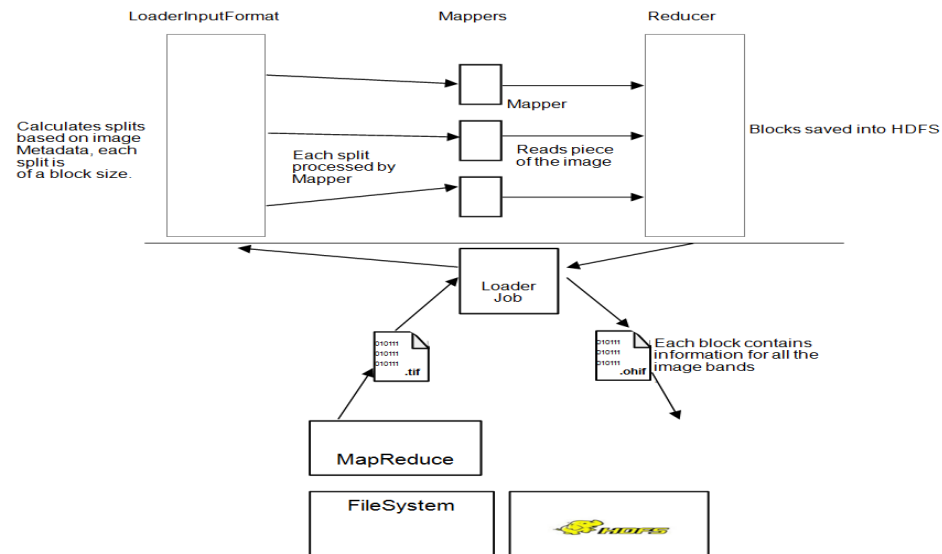


Image Processor

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. After the execution of processing classes, a mosaic operation is performed to adapt the pixels to the final output format requested by the user.
- 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:

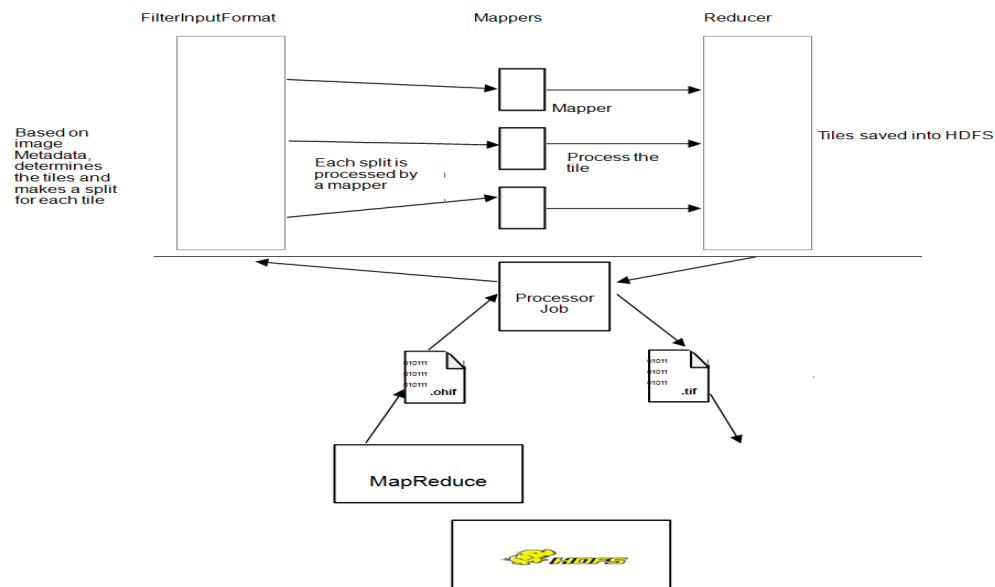


Image Server

The Image Server is a web application that 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 two main applications:

- 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, both server and console. It connects to the Hadoop cluster to load images to HDFS for further processing.

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](#)

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

$$\text{area} = ((\text{blockSize} - \text{metadata bytes}) / \text{number of bands}) / \text{bytes per pixel}.$$

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 completes 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.

Input Parameters

The following input parameters are supplied to the Hadoop command:

```
hadoop jar HADOOP_LIB_PATH/hadoop-imageloader.jar
  -files <SOURCE_IMGS_PATH>
  -out <HDFS_OUTPUT_FOLDER>
  [-overlap <OVERLAPPING_PIXELS>]
  [-thumbnail <THUMBNAIL_PATH>]
  [-gdal <GDAL_LIBRARIES_PATH>]
```

Where:

`HADOOP_LIB_PATH` for Oracle Big Data Appliance distribution is under `/opt/cloudera/parcels/CDH/lib/hadoop/lib/`, and for the standard distributions it is under `/usr/lib/hadoop/lib`.

`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 are considered.

`THUMBNAIL_PATH` is an optional path to store a thumbnail of the loaded image(s). This path must be accessible via NFS to all nodes in the cluster and must have a write access permission for yarn users.

`GDAL_LIBRARIES_PATH` is an optional path for GDAL native libraries, in case, the cluster has them in a path different to the default Oracle Big Data Appliance path `/opt/cloudera/parcels/CDH/lib/hadoop/lib`.

For example, this 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. Since no `-gdal` flag was specified, the gdal libraries are loaded from the default gdal path `/opt/cloudera/parcels/CDH/lib/hadoop/lib/native`.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/hadoop-imageloader.jar
-files /opt/sharedir/spatial/demo/imageserver/images/hawaii.tif -out ohiftest
-overlap 10 -thumbnail /opt/sharedir/spatial/processtest
```

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
```

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>` 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 each one of them cover, the file can be identified as: `SRID_dataType_original_filename.loc` where SRID and data type are extracted from source image. Following is an example of a couple of lines for a metadata file:

```
3,565758.0971152118,4302455.175478269,576788.4332668484,4291424.839326633
```

```
29,576757.962724993,4247455.847429363,582303.6013426667,4240485.710979952
```

The first element is the piece number, the second one is upper left X, followed by upper left Y, lower right X, and lower right Y. The `.loc` file is stored in the metadata folder under user the HDFS folder. 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>`.

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. You can specify additional processing classes to be executed before the final output is created by the framework.

Image processor loads specific blocks of data, based on the input (mosaic description), 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 File System depending on the user request.

- [Image Processing Job](#)

- [Input Parameters](#)
- [Job Execution](#)
- [Processing Classes and ImageBandWritable](#)
- [Output](#)

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. 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 processing classes defined in the request.

Each mapper process runs in the node, where the data is located. Once the processing classes are executed, the final process executes the mosaic 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 set in the context included in the `ImageBandWritable` type.

A single reducer picks the tiles and puts them together. If user selected HDFS output, then 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.

Input Parameters

The following input parameters are supplied to the Hadoop command:

```
hadoop jar HADOOP_LIB_PATH/hadoop-imageprocessor.jar
-catalog <IMAGE_CATALOG_PATH>
-config <MOSAIC_CONFIG_PATH>
[-usrlib <USER_PROCESS_JAR_PATH>]
[-thumbnail <THUMBNAIL_PATH>]
[-gdal <GDAL_LIBRARIES_PATH>]
```

Where:

`HADOOP_LIB_PATH` for Oracle Big Data Appliance distribution is under `/opt/cloudera/parcels/CDH/lib/hadoop/lib/`, and for the standard distributions it is under `/usr/lib/hadoop/lib`.

`IMAGE_CATALOG_PATH` is the path to the catalog xml that lists the HDFS image(s) to be processed.

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

`USER_PROCESS_JAR_PATH` is an optional user defined jar file, which contains additional processing classes to be applied to the source images.

`THUMBNAIL_PATH` is an optional path to store a thumbnail 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.

GDAL_LIBRARIES_PATH is an optional path for GDAL native libraries, in case, the cluster has them in a path different to the default Oracle Big Data Appliance path (/opt/cloudera/parcels/CDH/lib/hadoop/lib).

For example, This 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/cloudera/parcels/CDH/lib/hadoop/lib/hadoop-imageprocessor.jar
-catalog /opt/sharedir/spatial/demo/imageserver/images/input.xml
-config /opt/sharedir/spatial/demo/imageserver/images/testFS.xml
-thumbnail /opt/sharedir/spatial/processTest
```

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.

Catalog XML Structure

This 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>
    <source>HDFS</source>
    <type>File</type>
    -<raster> /user/hdfs/ohif/opt/sharedir/spatial/imageserver/hawaii.tif.ohi
  </raster>
  -<metadata>/user/hdfs/metadata/opt/sharedir/spatial/imageserver/26904_1_
hawaii.tif.loc </metadata>
    <bands>3</bands>
    <defaultRed>1</defaultRed>
    <defaultGreen>2</defaultGreen>
    <defaultBlue>3</defaultBlue>
  </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 <metadata> element specifies a .loc location file related to the source .ohif file. This location file contains the list of tiles for each source image and the coordinates cover per tile.

The <bands> element specifies the number of bands of the image, and the <defaultRed>, <defaultGreen>, and <defaultBlue> specify the band used for the first three channels of the image. In this example, band 1 is used for red channel, band 2 is used for green channel, and band 3 is used for blue channel.

Mosaic definition XML Structure

This is an example of a mosaic configuration xml used to define the features of the mosaic output generated by the image processing job.

```
-<mosaic>
  -<output>
    <SRID>26904</SRID>
    <directory type="FS">/opt/sharedir/spatial/processOutput</directory>
```

```

<!--directory type="HDFS">newData</directory-->
<tempFSFolder>/opt/sharedir/spatial/tempOutput</tempFSFolder>
<filename>littlemap</filename>
<format>GTIFF</format>
<width>1600</width>
<height>986</height>
<algorithm order="0">2</algorithm>
<bands layers="3"/>
<nodata>#000000</nodata>
<pixelType>1</pixelType>
</output>
-<crop>
  -<transform>
    356958.985610072,280.38843650364862,0,2458324.0825054757,0,-280.38843650364862
  </transform>
</crop>
-<process>
  -<class>oracle.spatial.imageprocessor.hadoop.process.ImageSlope </class>
</process>
</mosaic>

```

The `<mosaic>` element defines the specifications of the processing output.

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.

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.

The `<width>` and `<height>` elements set the mosaic output resolution.

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.

The `<bands>` element specifies the number of bands in the output mosaic. Images with fewer bands than this number are discarded.

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.

The `<crop>` element defines the coordinates included in the mosaic output in the following order: startcoordinateX, pixelXWidth, RotationX, startcoordinateY, RotationY, and pixelheightY.

The `<process>` element lists all the classes to execute before the mosaic operation. In this example a slope calculation is applied, and it is valid only for 32 bits images of Digital Elevation Model (DEM) files.

You can specify any other user-created processing classes. When no processing class is defined, only the mosaic operation is performed.

Job Execution

The first step of the job is to filter the tiles that would fit into the mosaic, as a start, the location files listed in the catalog xml are sent to the `InputFormat`, the location file name has the following structure: `SRID_pixelType_fileName.loc`.

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 mosaic definition. Only the intersecting tiles are selected, and a split is created for each one of them.

A mapper process each split in the node where it is stored. The mapper executes the sequence of processing classes defined by the user, and then the mosaic process is executed. A single reducer puts together the result of the mappers and stores the image into FS or HDFS upon user request. If the user requested is to store the output into HDFS, then the `ImageLoaderOverlap` 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.

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 must implement the `ImageProcessorInterface` and the `ImageBandWritable` type 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 mosaic output depends on the correct manipulation of these properties.

Table 2–1 *ImageBandWritable Properties*

Type - Property	Description
<code>IntWritable dstWidthSize</code>	Width size of the tile
<code>IntWritable dstHeightSize</code>	Height size of the tile
<code>IntWritable bands</code>	Number of bands in the tile
<code>IntWritable dType</code>	Data type of the tile
<code>IntWritable offX</code>	Starting X pixel, in relation to the source image
<code>IntWritable offY</code>	Starting Y pixel, in relation to the source image
<code>IntWritable totalWidth</code>	Width size of the source image
<code>IntWritable totalHeight</code>	Height size of the source image
<code>IntWritable bytesNumber</code>	Number of bytes containing the pixels of the tile and stored into <code>baseArray</code>
<code>BytesWritable[] baseArray</code>	Array containing the bytes representing the tile pixels, each cell represents a band

Table 2–1 (Cont.) ImageBandWritable Properties

Type - Property	Description
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
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

Processing Classes and Methods

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

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

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

If the tile has a Float32 data type, then use the following method to get the pixels in a float array:

```
float[] floatBand1 = img.getFloatBand(0);
```

For any other data types, you must convert the byte array into a corresponding type using gdal.

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);
```

Where band is a byte or a float array containing the pixel information. 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.

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.
- The jar files, for Oracle Big Data Appliance, in both cases must be placed under `/opt/cloudera/parcels/CDH/lib/hadoop/lib/`

Once new classes are visible in the classpath, they must be added to the mosaic XML, in the `<process><class>` section. Every `<class>` element added is executed in order of appearance before the final mosaic operation is performed.

Output

When you specify an HDFS directory in the catalog 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 catalog 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 catalog xml. The job execution logs can be accessed using the command `yarn logs -applicationId <applicationId>`.

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:

- [Spatial Indexing](#)
- [Spatial Filtering](#)
- [Classifying Data Hierarchically](#)
- [Generating Buffers](#)

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

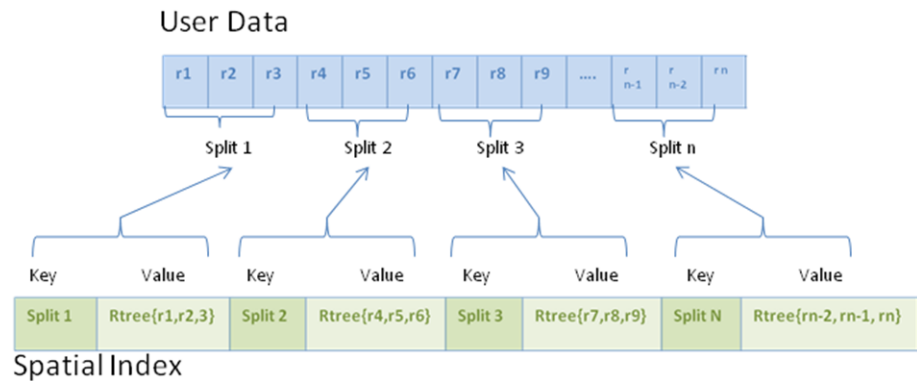
- [RecordInfoProvider](#)
- [HierarchyInfo](#)
- [Using JGeometry in MapReduce Jobs](#)
- [Tuning Performance Data of Job Running Times using Vector Analysis API](#)

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 contain 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.



Related subtopics:

- [Spatial Indexing Class Structure](#)

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`.

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, TextInputFormat.class);
FileSplitInputFormat.setRecordInfoProviderClass(conf,
TwitterLogRecordInfoProvider.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`, an abstract class which uses another `InputFormat` implementation (internal `InputFormat`) to read the data. The internal `InputFormat` and the `RecordInfoProvider` implementations are specified by the user and they are set to `TextInputFormat` and `TwitterLogRecordInfoProvider` respectively.
- The `OutputFormat` used is `MapFileOutputFormat` is the resulting `MapFile`.
- The mapper is set to `SpatialIndexingMapper`. The mapper output key and value types are `Text` (splits identifiers) and `RTreeWritable` (the actual spatial indices).
- 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 `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);

```

Input Formats for 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.
- The `getPos()` method, the `RecordReader` provided by the `InputFormat` 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 indices 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 indices 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 requisites for a new API input format is same as for the old API. However, it must be translated to the new APIs `FileInputFormat`, `FileSplit`, and `CombineFileSplit`. The following example shows an input format from the new Hadoop API that is used as internal input format:

```

AdapterInputFormat.setInternalInputFormatClass(conf,
org.apache.hadoop.mapreduce.lib.input.TextInputFormat);
spatialIndexing.setInputFormatClass(AdapterInputFormat.class);

```

MVSuggest for Locating Records

`MVSuggest` can be used at the time of spatial indexing to get an approximate location for records, which 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.

Since it is important to know the field containing the search text for every record, the `RecordInfoProvider` implementation must also implement `LocalizableRecordInfoProvider`. For more information see the ["LocalizableRecordInfoProvider"](#) on page 2-30.

`MVSuggest` can be used in the ["Input Formats for Spatial Index"](#) on page 17. example by calling the following method in the `SpatialIndexing` class:

```
spatialIndexing.setMVSTurl("http://host:port")
```

If `MVSuggest` service is deployed at every cluster node, then the host can be set to `localhost`. The layers used by `MVSuggest` to perform the search can be specified by passing the name of the layers as an array of strings to the following method:

```
spatialIndexing.setMatchingLayers( new String[]{ "world_continents", "world_countries", "world_cities" } )
```

This settings can also be added directly to the job configuration using the following parameters:

```
conf.set(ConfigParams.MVS_URL, "http://host:port")
conf.set(ConfigParams.MVS_MATCHING_LAYERS, "world_continents, world_countries, world_cities")
```

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

SpatialFilterInputFormat.setSpatialOperation(conf, SpatialOperation.IsInside);
```

```

SpatialFilterInputFormat.setSpatialQueryWindow
(conf, "{\"type\":\"Polygon\", \"coordinates\":[[-106.64595, 25.83997, -106.64595,
36.50061, -93.51001, 36.50061, -93.51001, 25.83997, -106.64595, 25.83997]]}");
SpatialFilterInputFormat.setSRID(conf, 8307);
SpatialFilterInputFormat.setTolerance(conf, 0.5);
SpatialFilterInputFormat.setGeodetic(conf, 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 Spatial Index"](#) on page 17.

Filtering Records

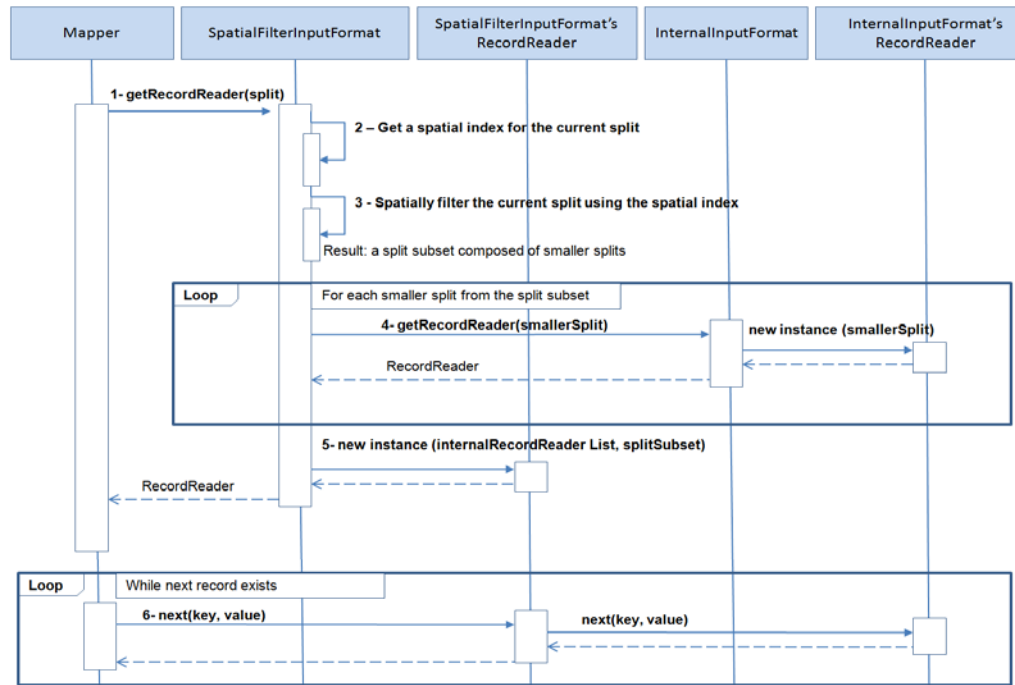
The following steps are executed when records are filtered using the `SpatialFilterInputFormat` and a spatial index.

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 `InternalInputFormat` `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.



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:

```

HierarchicalCount<LongWritable,Text> hierCount = new
HierarchicalCount<LongWritable,Text>()

//when spatial option is set the job will use a spatial index

hierCount.setOption(Option.parseOption(Option.Spatial));

//set the path to the spatial index

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

//set the job's output

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

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

hierCount.setHierarchyInfoClass( WorldAdminHierarchyInfo.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.setHierarchyDataIndexPath("/user/hierarchy_data_index/");

//setup the spatial operation which will be used to join records from the two
datasets (spatial index and hierarchy data).

hierCount.setSpatialOperation(SpatialOperation.IsInside);
hierCount.setSrid(8307);
hierCount.setTolerance(0.5);
hierCount.setGeodetic(true);

//add the previous setup to the job configuration

hierCount.configure(conf);

//run the job

RunningJob rj = JobClient.runJob(conf);

//call jobFinished to properly rename the output files

hierCount.jobFinished(rj.isSuccessful());

```

This example uses the `HierarchicalCount` demo job driver to facilitate the configuration. The configuration can be divided in to following categories:

- Input data: This is a previously generated spatial index.
- Output data: This is a folder which 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`).

The `HierarchyCount.jobFinished()` method is called at the end to properly name the reducer output. Therefore, only one file is left for each hierarchy level instead of multiple files with the `-r-NNNNN` postfix. Internally, the `HierarchyCount.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 loaded 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 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

conf.setInt(ConfigParams.SPATIAL_OPERATION,
SpatialOperation.IsInside.getOrdinal());
conf.setInt(ConfigParams.SRID, 8307);
conf.setFloat(ConfigParams.SPATIAL_TOLERANCE, 0.5f);
conf.setBoolean(ConfigParams.GEODETIC, true);

//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 `RecordInfo` 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;
    }
}
```

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 it is not recommended to modify the `HierarchyInfo` implementation.

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

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 URL is set. Notice that there is no spatial operation configuration needed in this case. The changes are marked bold in the following code listing:

```
HierarchicalCount<LongWritable,Text> hierCount = new
HierarchicalCount<LongWritable,Text>()

//set option to MVS

hierCount.setOption(Option.parseOption(Option.MVS));

//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( WorldAdminHierarchyInfo.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.setHierarchyDataIndexPath("/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 service URL

hierCount.setMVUrl("http://localhost:8080");

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

//run the job

RunningJob rj = JobClient.runJob(conf);

//call jobFinished to properly rename the output files

hierCount.jobFinished(rj.isSuccessful());

```

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

Generating Buffers

The API provides a mapper and a demo job to generate a buffer around each record's geometry. The following code sample shows how to run a job to generate a buffer around each record geometry

```

Buffer<LongWritable, Text> buffer = new Buffer<LongWritable, Text>();

//set path to input data

buffer.setInput("/user/waterlines/");

//set path to the resulting data

buffer.setOutput("/user/data_buffer/");

//set input format used to read the data

buffer.setInputFormatClass( TextInputFormat.class );

//set the record info provider implementation

buffer.setRecordInfoProviderClass(WorldSampleLineRecordInfoProvider.class);

//set the width of the buffers to be generated

buffer.setBufferWidth(Double.parseDouble(args[4]));

//set the previous configuration to the job configuration

buffer.configure(conf);

//run the job

JobClient.runJob(conf);

```

This example job uses `BufferMapper` as the job mapper. `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.

`BufferMapper` accepts the following parameters:

Parameter	ConfigParam constant	Type	Description
oracle.spatial.buffer.width	BUFFER_WIDTH	double	The buffer width
oracle.spatial.buffer.sema	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

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)`

When a record is read from HDFS, the `setCurrentRecord()` method sets the `RecordInfoProvider` instance. The key-value pair passed as arguments are same as the values returned by the `RecordReader` provided by the `InputFormat`. The `RecordInfoProvider` is 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.

Sample RecordInfo Implementation

This is a `JsonRecordInfoProvider` implementation, which takes text records in JSON format and 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. This is used to parse only a record value when `setCurrentRecord()` method 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;
}
}

```

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, and is sent to `MVSuggest`.

In addition, the following is an implementation based on "[Sample RecordInfo Implementation](#)" on page 29. 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";
    }
}

```

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");
        }
    }
}
```

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
```

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();
        }
    }
}

```

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`.

Using 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](#)
- [Running a Hierarchy Job Using the Console](#)
- [Viewing the Index Results](#)
- [Creating Results Manually](#)
- [Creating and Deleting templates](#)
- [Configuring Templates](#)
- [Running a Job to Create an Index Using the Command Line](#)
- [Running a Job to Perform a Spatial Filtering](#)
- [Running a Job to Create a Hierarchy Result](#)
- [Running a Job to Generate Buffer](#)

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 `http://<oracle_big_data_spatial_vector_console>:8080/spatialviewer/`
2. Click Create Index.
3. Specify all the required details:
 - a. Path of data to the index: Path of the index file in HDFS. For example, `hdfs://<server_name_to_store_index>:8020/user/hsaucedo/twitter_logs/part-m-00000`.
 - b. New index path: This is the job output path. For example: `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/rinfante/Twitter/index`.
 - c. Input Format class: The input format class. For example: `org.apache.hadoop.mapred.TextInputFormat`
 - d. Record Info Provider class: The class that provides the spatial information. For example: `oracle.spatial.hadoop.vector.demo.usr.TwitterLogRecordInfoProvider`.

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.

- e. **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`
 - f. **MVSuggest Templates (Optional):** The user can define the templates used to create the index with MVSuggest. For optimal results, it is recommended to select the same templates for the index creation and when running the hierarchy job.
 - g. **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`
4. Click Create.
- The submitted job is listed and you should wait to receive a mail notifying that the job completed successfully.

Running a Hierarchy Job Using the Console

You can run a hierarchy job to with or without the spatial index, follow these steps.

1. Open `http://<oracle_big_data_spatial_vector_console>:8080/spatialviewer/`.
2. Click Run Job.
3. Select either With Index or Without Index and provide the following details, as required:
 - **With Index**
 - a. **Index path:** The index file path in HDFS. For example, `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/Twitter/index/part*/data`.
 - b. **SRID:** The geometries used to build the index. For example, 8307.
 - c. **Tolerance:** The tolerance of the geometry to build the index. For example, 0.5.
 - d. **Geodetic:** Whether the geometries used are geodetic or not. Select Yes or No.
 - **Without Index**
 - a. **Path of the data:** Provide the HDFS data path. For example, `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/*/data`.

- 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:
`org.apache.hadoop.mapred.FileInputFormat`
 - d. Record Info Provider class: The class that will provide the spatial information. For example:
`oracle.spatial.hadoop.vector.demo.usr.TwitterLogRecordInfoProvider`
 - e. MVSuggest service URL: 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`
4. Templates: The templates to create the thematic maps. For optimal results, it is recommended to select the same templates for the index creation and when running the hierarchy job. For example, World Continents, World Countries, and so on.

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 accurate, provided all the templates that could match the results are set. 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, counties, and cities, then the suitable templates are USA States, USA Counties, and USA Cities.

- 5. Output path: The Hadoop job output path. For example, `hdfs://<oracle_big_data_spatial_vector_console>:8020/user/rinfante/Twitter/myoutput`.
- 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`.

Viewing the Index Results

To view the results, follow these steps.

- 1. Open `http://<oracle_big_data_spatial_vector_console>:8080/spatialviewer/`.

2. Click Show Hadoop 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.


Creating Results Manually

It is possible to create and upload result files stored locally. For example, the result file created with a job executed from the command line. 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. From Show Hadoop Results, click the .
2. Select a Template from the drop down.
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.

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_`.
3. If `MVSuggest` is used, then add the new templates to the `MVSuggest` templates.

To delete the template, delete the JSON and configuration files added in steps 1 and 2.

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 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 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 don't 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 `PARENT_REGION` in the template file `usa_states.json`.

- **srid:** The SRID of the template's geometries. For example, `srid: 8307`.
- **bounds:** The bounds of the map when showing the template. For example, `bounds: [-180, -90, 180, 90]`.
- **numberOfZoomLevels:** The map's number of zoom level when showing the template. For example, `numberOfZoomLevels: 19`.
- **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`.

Running a Job to Create an Index Using the Command Line

Run the following command to generate an index:

```
hadoop jar <HADOOP_LIB_PATH>/<jarfile>
oracle.spatial.hadoop.vector.demo.job.SpatialIndexing <DATA_PATH> <SPATIAL_INDEX_PATH> <INPUT_FORMAT_CLASS> <RECORD_INFO_PROVIDER_CLASS>
```

Where,

- `jarfile` is the jar file to be specified by the user for spatial indexing.
- `DATA_PATH` is the location of the data to be indexed.
- `SPATIAL_INDEX_PATH` is the location of the resulting spatial index.
- `INPUT_FORMAT_CLASS` is the `InputFormat` implementation used to read the data.
- `RECORD_INFO_PROVIDER_CLASS` is the implementation used to extract information from the records.

The following example uses the command line to create an index.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.SpatialIndexing "/user/hdfs/demo_vector/tweets/part*" /user/hdfs/demo_vector/tweets/spatial_index
org.apache.hadoop.mapred.TextInputFormat
oracle.spatial.hadoop.vector.demo.usr.TwitterLogRecordInfoProvider
```

Note: The preceding example uses the demo job that comes preloaded.

Running a Job to Perform a Spatial Filtering

Run the following command to perform a spatial filter:

```
hadoop jar HADOOP_LIB_PATH/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.TwitterLogSearch <DATA_PATH> <RESULT_PATH>
<SEARCH_TEXT> <SPATIAL_INTERACTION> <GEOMETRY> <SRID> <TOLERANCE> <GEODETIC>
<SPATIAL_INDEX_PATH>
```

Where,

- DATA_PATH is the data to be filtered.
- RESULT_PATH is the path where the results are generated.
- SEARCH_TEXT is the text to be searched.
- SPATIAL_INTERACTION is the type of spatial interaction. It can be either 1 (is inside) or 2 (any interact).
- GEOMETRY is the geometry used to perform the spatial filter. It should be expressed in JSON format.
- SRID is the SRS id of the query geometry.
- TOLERANCE is the tolerance used for the spatial search.
- GEODETIC mentions whether the geometries are geodetic or not.
- SPATIAL_INDEX_PATH is the path to a previously generated spatial index.

The following example demonstrates the counts all tweets containing some text and interacting with a given geometry.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.TwitterLogSearch "/user/hdfs/demo_
vector/tweets/part*" /user/hdfs/demo_vector/tweets/text_search feel 2
'{"type":"Polygon", "coordinates":[[-106.64595, 25.83997, -106.64595, 36.50061,
-93.51001, 36.50061, -93.51001, 25.83997, -106.64595, 25.83997]]}' 8307 0.0 true
/user/hdfs/demo_vector/tweets/spatial_index
```

Note: The preceding example uses the demo job that comes preloaded.

Running a Job to Create a Hierarchy Result

Run the following command to create a hierarchy result:

```
hadoop jar HADOOP_LIB_PATH/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.HierarchicalCount spatial <SPATIAL_INDEX_
PATH> <RESULT_PATH> <HIERARCHY_INFO_CLASS> <SPATIAL_INTERACTION> <SRID>
<TOLERANCE> <GEODETIC> <HIERARCHY_DATA_INDEX_PATH> <HIERARCHY_DATA_PATHS>
```

Where,

- SPATIAL_INDEX_PATH is the path to a previously generated spatial index. If there are files other than the spatial index path files in this path (for example, the hadoop generated _SUCCESS file), then add the following pattern: SPATIAL_INDEX_PATH/part*/data.

- `RESULT_PATH` is the path where the results are generated. There should be a file named `XXXX_count.json` for each hierarchy level.
- `HIERARCHY_INFO_CLASS` is the location of the resulting spatial index.
- `INPUT_FORMAT_CLASS` is the `HierarchyInfo` implementation. It defines the structure of the current hierarchy data.
- `SPATIAL_INTERACTION`, `SRID`, `TOLERANCE`, `GEODETIC`, all these parameters have the same meaning as defined previously for Text Search Job. They are used to define the spatial operation between the data and the hierarchical information geometries.
- `HIERARCHY_DATA_INDEX_PATH` is the path where the hierarchy data index will be placed. This index is used by the job to avoid finding parent-children relationships each time are required.
- `HIERARCHY_DATA_PATHS` are comma separated list of paths to the hierarchy data. If a hierarchy index was created previously for the same hierarchy data it can be omitted.

The following example uses the command line to create an index.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.HierarchicalCount spatial "/user/hdfs/demo_
vector/tweets/spatial_index/part*/data" /user/hdfs/demo_vector/tweets/hier_count_
spatial oracle.spatial.hadoop.vector.demo.usr.WorldAdminHierarchyInfo 1 8307 0.5
true /user/hdfs/demo_vector/world_hier_index
file:///net/den00btb/scratch/hsaucedo/spatial_bda/demo/vector/catalogs/world_
continents.json,file:///net/den00btb/scratch/hsaucedo/spatial_
bda/demo/vector/catalogs/world_
countries.json,file:///net/den00btb/scratch/hsaucedo/spatial_
bda/demo/vector/catalogs/world_states_provinces.json
```

Note: The preceding example uses the demo job that comes preloaded.

Running a Job to Generate Buffer

Run the following command to generate a buffer:

```
hadoop jar HADOOP_LIB_PATH/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.Buffer <DATA_PATH> <RESULT_PATH> <INPUT_
FORMAT_CLASS> <RECORD_INFO_PROVIDER_CLASS> <BUFFER_WIDTH> <BUFFER_SMA> <BUFFER_
FLAT> <BUFFER_ARCTOL>
```

Where,

- `DATA_PATH` is the data to be filtered.
- `RESULT_PATH` is the path where the results are generated.
- `INPUT_FORMAT_CLASS` is the `InputFormat` implementation used to read the data.
- `RECORD_INFO_PROVIDER_CLASS` is the `RecordInfoProvider` implementation used to extract data from each record.
- `BUFFER_WIDTH` specifies the buffer width.
- `BUFFER_SMA` is the semi major axis.
- `BUFFER_FLAT` is the buffer flattening.
- `BUFFER_ARCTOL` is the arc tolerance for geodetic arc densification.

The following example demonstrates generating a buffer around each record geometry. The resulting file is a MapFile where each entry corresponds to a record from the input data. The entry key is the record id and the value is a `RecordInfo` instance holding the generated buffer and the record location (path, start offset and length).

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop/lib/sdohadoop-vector-demo.jar
oracle.spatial.hadoop.vector.demo.job.Buffer "/user/hdfs/demo_
vector/waterlines/part*" /user/hdfs/demo_vector/waterlines/buffers
org.apache.hadoop.mapred.TextInputFormat
oracle.spatial.hadoop.vector.demo.usr.WorldSampleLineRecordInfoProvider 5.0
```

Note: The preceding example uses the demo job that comes preloaded.

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](#).

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.

Part III

Property Graphs

Part III contains the following chapters:

- Chapter 3, "Configuring Property Graph Support"
- Chapter 4, "Using Property Graphs in a Big Data Environment"
- Chapter 5, "Using In-Memory Analytics"

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, either on a Big Data Appliance (see [Installing Oracle Big Data Spatial and Graph on an Oracle Big Data Appliance](#)) or on an Apache Hadoop system (see [Installing Property Graph Support on a CDH Cluster or Other Hardware](#)).

- [Tuning the Software Configuration](#)

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](#)

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](#)

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 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. 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.

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>

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 $(3000 \text{ MB} * .85) / 2$
- 892 MB cache, calculated as $1275 \text{ MB} * .70$

See Also: Oracle NoSQL Database FAQ at

<http://www.oracle.com/technetwork/products/nosqldb/learnmore/nosqldb-faq-518364.html#HowdoesNoSQLDBbudgetmemory>

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.

- [About Property Graphs](#)
- [Getting Started With Property Graphs](#)
- [About Property Graph Data Formats](#)
- [Using Java APIs for Property Graph Data](#)
- [Managing Text Indexing for Property Graph Data](#)
- [Property Graph Support for Secure Oracle NoSQL Database](#)
- [Using the Groovy Shell with Property Graph Data](#)
- [Exploring the Sample Programs](#)
- [Oracle Flat File Format Definition](#)
- [Example Python User Interface](#)

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.

- [What Are Property Graphs?](#)
- [What Is Big Data Support for Property Graphs?](#)

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:

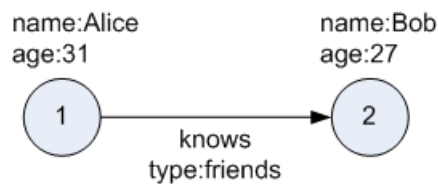
- A set of outgoing edges
- A set of incoming edges
- A collection of properties

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



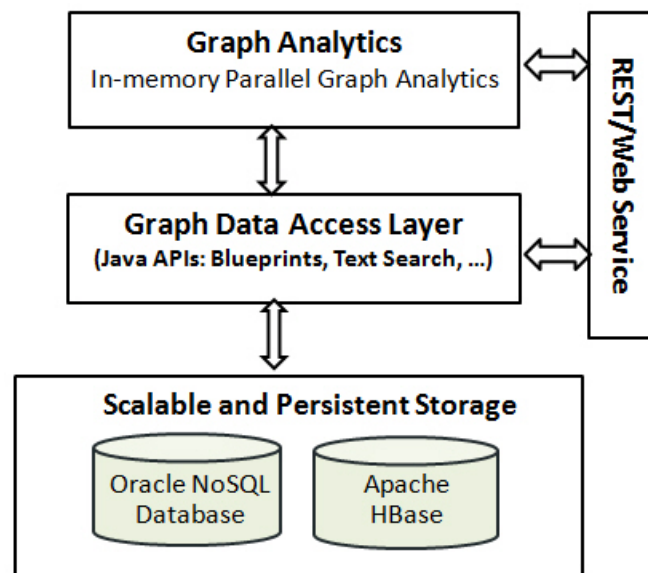
The property graph data model is not based on standards, but is similar to the W3C standards-based Resource Description Framework (RDF) graph data model. The property graph data model is simpler and much less precise than RDF. These differences make it a good candidate for use cases such as these:

- Identifying influencers in a social network
- Predicting trends and customer behavior
- Discovering relationships based on pattern matching
- Identifying clusters to customize campaigns

What Is Big Data Support for Property Graphs?

Property graphs are supported for Big Data in Hadoop. 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

Analytics Layer

The analytics layer enables you to analyze property graphs using MapReduce programs in a Hadoop cluster. It provides over 30 analytic functions, including path calculation, ranking, community detection, and a recommender system.

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.

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.

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.

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 ["Overview of the Java APIs"](#) on page 4-6.

About Property Graph Data Formats

The following graph formats are supported:

- [GraphML Data Format](#)
- [GraphSON Data Format](#)
- [GML Data Format](#)
- [Oracle Flat File Format](#)

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](#).

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>
```

See Also: "The GraphML File Format" at

<http://graphml.graphdrawing.org/>

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](#).

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"
    }
]
}

```

See Also: "GraphSON Reader and Writer Library" at

<https://github.com/tinkerpop/blueprints/wiki/GraphSON-Reader-and-Writer-Library>

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](#).

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"
  ]
]

```

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>

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](#).

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, ,
```

See Also: ["Oracle Flat File Format Definition"](#) on page 4-31

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](#)
- [Opening and Closing a Property Graph Instance](#)
- [Creating the Vertices](#)
- [Creating the Edges](#)
- [Deleting the Vertices and Edges](#)
- [Dropping a Property Graph](#)

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](#)
- [TinkerPop Blueprints Java APIs](#)
- [Oracle NoSQL Database Java APIs](#)
- [Apache HBase Java APIs](#)

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

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>

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/>

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 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/>

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>

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](#)

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 in-memory analytical 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.api.Pgx;
import oracle.pgx.config.*;
import oracle.pgx.api.analyst.*;
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.forNosql()
    .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 opgd1 =
OraclePropertyGraphDataLoader.getInstance();
opgd1.loadData(opg, szOPVFile, szOPEFile, dop);

...
Analyst analyst = opg.getInMemAnalyst();
...
```

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.zookeeper.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 in-memory analytical 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.

```
int dop = 8;  
Map<PgxCfg.Field, Object> confPgx = new HashMap<PgxCfg.Field, Object>();  
confPgx.put(PgxCfg.Field.ENABLE_GM_COMPILER, false);  
confPgx.put(PgxCfg.Field.NUM_WORKERS_IO, dop + 2);  
confPgx.put(PgxCfg.Field.NUM_WORKERS_ANALYSIS, 8); // <= # of physical cores  
confPgx.put(PgxCfg.Field.NUM_WORKERS_FAST_TRACK_ANALYSIS, 2);  
confPgx.put(PgxCfg.Field.SESSION_TASK_TIMEOUT_SECS, 0); // no timeout set  
confPgx.put(PgxCfg.Field.SESSION_IDLE_TIMEOUT_SECS, 0); // no timeout set  
PgxCfg.init(confPgx);  
  
int iClientPort = Integer.parseInt(szClientPort);  
int iSplitsPerRegion = 2;  
  
PgHbaseGraphConfig cfg = GraphConfigBuilder.forHbase()  
    .setName(hbGraphName)  
    .setZkQuorum(hbQuorum)  
    .setZkQuorum(szQuorum)  
    .setZkClientPort(iClientPort)  
    .setZkSessionTimeout(60000)  
    .setMaxNumConnections(dop)  
    .setLoadEdgeLabel(true)  
    .setSplitsPerRegion(splitsPerRegion)  
    .addEdgeProperty("lbl", PropertyType.STRING, "lbl")  
    .addEdgeProperty("weight", PropertyType.DOUBLE,  
        "1000000")  
    .build();  
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);  
Analyst analyst = opg.getInMemAnalyst();  
long triangles = analyst.countTriangles(false).get();
```

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(11);
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(21);
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);

```

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 a = opg.addVertex(11);
v1.setProperty("name", "Alice");
v1.setProperty("age", 31);

Vertex v2 = opg.addVertex(21);
v2.setProperty("name", "Bob");
v2.setProperty("age", 27);

// Add edge e1
Edge e1 = opg.addEdge(11, v1, v2, "knows");
e1.setProperty("type", "friends");

```

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.

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](#)

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);
```

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.zookeeper.property.clientPort", hbClientPort);
HConnection conn = HConnectionManager.createConnection(conf);

// Drop the graph
OraclePropertyGraphUtils.dropPropertyGraph(conf, hbGraphName);
```

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 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 use 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](#)

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 = 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](#)

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](#)

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
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 = 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
```

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 = 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: " + query + " -----");
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

```

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 -cmd upconfig -confname  
opgconfig -confdir -/tmp/opg-solr-config
```

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());

```

Property Graph Support for Secure Oracle NoSQL Database

Oracle's property graph support works with both secure and non-secure Oracle NoSQL Database installations. This topic provides information about secure Oracle NoSQL Database installations. It assumes that a password store is configured for the database (see the *Oracle NoSQL Database Security Guide* at <http://docs.oracle.com/cd/NOSQL/html/SecurityGuide/index.html> for details).

On the client side, you must create a login properties file named `login_properties.txt`, which contains settings like the trust store and SSL transport. For detailed information, see "Using the Authentication APIs" at

<http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuideTables/authentication.html>.

To specify the login properties file, use 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 (including in-memory analytics) deployed into a J2EE container. 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"
```

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.

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.forNosql().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> opgd = OraclePropertyGraphDataLoader.getInstance();
==>oracle.pg.nosql.OraclePropertyGraphDataLoader@576f1cad

opg-nosql> opgd.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> PgxConfig.init(confPgx);
==>null
opg-hbase>

opg-hbase> iClientPort = 2181;
==>2181

opg-hbase> cfg = GraphConfigBuilder.forHbase().setName("myGraph")
.setZkQuorum("bigdatalite").setZkClientPort(iClientPort)
.setZkSessionTimeout(60000).setMaxNumConnections(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> analyst = opg.getInMemAnalyst();
==>oracle.pgx.api.analyst.Analyst@534df5ff

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).

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](#)

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)*

Program Name	Description
ExampleNoSQL1	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.
ExampleHBase1	
ExampleNoSQL2	Creates the same minimal property graph as Example1, and then deletes it.
ExampleHBase2	

Table 4–3 (Cont.) Property Graph Program Examples (Selected)

Program Name	Description
ExampleNoSQL3	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.
ExampleHBase3	

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".

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

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.

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](#)" on page 4-27.

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();
```

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 **** opg javadoc ulink **** for the following:

```
OraclePropertyGraph.addVertex
OraclePropertyGraph.clearRepository
OraclePropertyGraph.getInstance
```

```

OraclePropertyGraph.getVertices
OraclePropertyGraph.shutdown
Vertex.setProperty

```

Example: Dropping a Property Graph

ExampleNoSQL2 and ExampleHBase2 create a graph like the one in ["Example: Creating a Property Graph"](#) on page 4-28, and then drop it from the database.

The code fragment in [Example 4-6](#) drops the graph. See ["Compiling and Running the Sample Programs"](#) on page 4-27 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`.

Examples: Adding and Dropping Vertices and Edges

ExampleNoSQL3 and ExampleHBase3 add and drop both vertices and edges.

The code fragment in [Example 4-7](#) creates three vertices. It is a simple variation of [Example 4-5](#).

Example 4-7 Creating the Vertices

```

// 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(11);
a.setProperty("name", "Alice");
a.setProperty("age", 31);

Vertex b = opg.addVertex(21);
b.setProperty("name", "Bob");
b.setProperty("age", 27);

Vertex c = opg.addVertex(31);
c.setProperty("name", "Chris");
c.setProperty("age", 33);

```

The code fragment in [Example 4-8](#) uses vertices a, b, and c to create the edges.

Example 4-8 Creating the Edges

```

// Add edges e1, e2, and e3
Edge e1 = opg.addEdge(11, a, b, "knows");

```

```
e1.setProperty("type", "partners");

Edge e2 = opg.addEdge(21, a, c, "knows");
e2.setProperty("type", "friends");

Edge e3 = opg.addEdge(31, b, c, "knows");
e3.setProperty("type", "colleagues");
```

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.

Example 4–9 Deleting Edges and Vertices

```
// Remove edge e3
opg.removeEdge(e3);

// Remove vertex b and all related edges
opg.removeVertex(b);
```

This example queries the database to show when objects are added and dropped. The code fragment in [Example 4–10](#) shows the method used.

Example 4–10 Querying for Vertices and Edges

```
// 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

```

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](#)

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.

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 ¹	Description
1	<i>vertex_ID</i>	An integer that uniquely identifies the vertex
2	<i>key_name</i>	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	<i>value_type</i>	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 10 Serializable Java object
4	<i>value</i>	The encoded, nonnull value of <i>key_name</i> when it is neither numeric nor date
5	<i>value</i>	The encoded, nonnull value of <i>key_name</i> when it is numeric
6	<i>value</i>	The encoded, nonnull value of <i>key_name</i> 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: <pre>SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'Th'HH:mm:ss.SSSXXX"); encode(sdf.format((java.util.Date) value));</pre>

¹ Names do not appear in the vertex files, but are provided here to simplify field references.

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 ¹	Description
1	<i>edge_ID</i>	An integer that uniquely identifies the edge
2	<i>source_vertex_ID</i>	The <i>vertex_ID</i> of the outgoing tail of the edge.
3	<i>destination_vertex_ID</i>	The <i>vertex_ID</i> of the incoming head of the edge.
4	<i>edge_label</i>	The encoded label of the edge, which describes the relationship between the two vertices

Table 4–6 (Cont.) Edge File Record Format

Field Number	Name ¹	Description
5	<i>key_name</i>	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	<i>value_type</i>	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 10 Serializable Java object
7	<i>value</i>	The encoded, nonnull value of <i>key_name</i> when it is neither numeric nor date
8	<i>value</i>	The encoded, nonnull value of <i>key_name</i> when it is numeric
9	<i>value</i>	The encoded, nonnull value of <i>key_name</i> when it is a date Use the Java <code>SimpleDateFormat</code> class to identify the format of the date. This example describes the date format of 2015-03-26Th00:00:00.000-05:00: <code>SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'Th'HH:mm:ss.SSSXXX"); encode(sdf.format((java.util.Date) value));</code>

¹ Names do not appear in the edge files, but are provided here to simplify field references.

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

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,,,
```

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 be 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.getId(),"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.getId(),"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/`

Using In-Memory Analytics

This chapter provides examples using In-Memory Analytics (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 Analytics Server](#)
- [Reading and Storing Data in HDFS](#)
- [Running In-Memory Analytics as a YARN Application](#)
- [Using the Java API Inside the In-Memory Analytics Shell](#)

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. [Starting the In-Memory Analytics Shell](#)
2. [Using the Shell Help](#) (as needed)
3. [Providing Graph Metadata in a Configuration File](#)
4. [Reading Graph Data into Memory](#)

Starting the In-Memory Analytics Shell

To start the In-Memory Analytics shell:

1. Open a terminal session on the system where property graph support is installed.
2. Set the GROOVY_HOME environment variable.

The In-Memory Analytics Groovy shell requires the Groovy 2.4.0 be installed and that GROOVY_HOME be defined correctly. For example

```
setenv GROOVY_HOME /<your-directory>/groovy-2.4.0/
```

3. Change to the In-Memory Analytics home directory, and run the `pgx` command:

```
cd $PGX_HOME
./bin/pgx
```

If the In-Memory Analytics software is installed correctly, you will see an engine-running log message and the In-Memory Analytics shell prompt (`pgx:000>`):

```
pgx:000> :load /opt/oracle/oracle-spatial-graph/property_
graph/pgx/conf/pgx.profile
11:48:45,172 [main] INFO Ctrl - >>> PGX engine running.
pgx:000>
```

The shell started a local instance because the `pgx` command did not specify a remote URL.

Using the Shell Help

The In-Memory Analytics shell provides a help system, which you access using the `:help` command.

For example, type `:help :loadGraph` to see how to use the `:loadGraph` command:

```
pgx:000> :help :loadGraph
```

```
usage: :loadGraph <Path to graph config file> <Graph name>
```

Load a graph specified by a graph config file path into memory.

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 In-Memory Analytics reads the graph.

```
pgx:000> :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 In-Memory Analytics loads the graph, it searches the `examples/graphs` directory for a file named `sample.adj`.

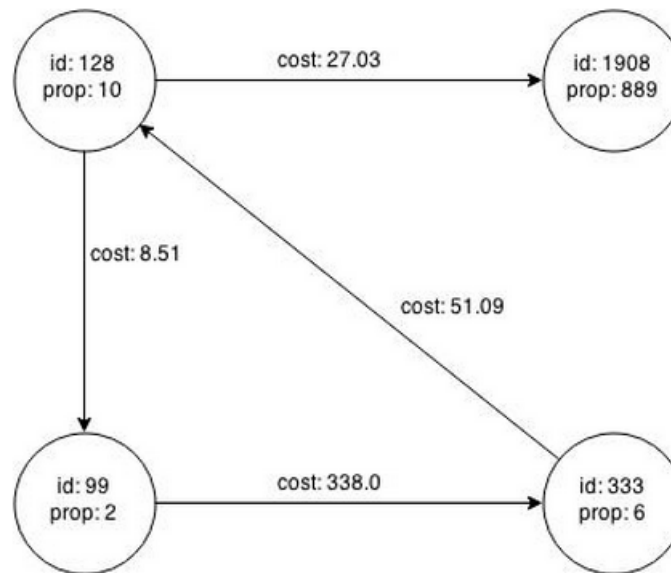
The other fields indicate that the graph data is provided in adjacency list format, and consists of one node property of type integer and one edge property of type double.

This is the graph data in adjacency list format:

```
pgx:000> :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



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

```
:loadGraph /opt/oracle/oracle-spatial-graph/property_
graph/examples/pgx/graphs/sample.adj.json sample
==> {

  "graphName" : "sample",
  [... rest of output omitted for brevity ...]
}
```

Example: Using Java to Read a Graph

```
import oracle.pgx.api.LoadingResult;
import oracle.pgx.config.GraphConfig;
import oracle.pgx.config.GraphConfigFactory;
```

```
GraphConfig graphConfig =
GraphConfigFactory.forAnyFormat().fromPath("/opt/oracle/oracle-spatial-graph/prop
erty_graph/examples/pgx/graphs/sample.adj.json");
LoadingResult lr = core.loadGraphWithProperties(sessionId, graphConfig).get();
String graphName = lr.getGraphName()
```

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](#)

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 a single `:loadGraph` command, the property graph connections will be read into memory:

```
pgx:000> :loadGraph /tmp/my_graph_hbase.json connections
```

The output might look like the following:

```
==> {
"graphName" : "connections",
"metaData" : {
"primitives" : {
"NUM_EDGES" : 164,
"MEMORY_MB" : 0,
"CREATION_TS" : 1429219094588,
"NUM_NODES" : 78,
"CREATION_REQUEST_TS" : 1429219088740,
"DATA_SOURCE_VERSION" : 1448548982
},

```

```

"graphConfig" : {
  "format" : "pg",
  "zk_quorum" : "scaj31bda07,scaj31bda08,scaj31bda09",
  "node_props" : [ {
    "name" : "country",
    "type" : "string"
  } ],
  "zk_client_port" : 2181,
  "db_engine" : "HBASE",
  "name" : "connections",
  "load_edge_label" : true,
  "edge_props" : [ {
    "name" : "label",
    "type" : "string"
  }, {
    "name" : "weight",
    "type" : "float"
  } ]
}
},
"nodeProperties" : {
  "country" : "string"
},
"edgeProperties" : {
  "weight" : "float",
  "label" : "string"
}
}
pgx:000>

```

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
```

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:000> :loadGraph /tmp/my_graph_nosql.json connections
pgx:000> :undirectGraph connections U
pgx:000> :countTriangles U

```

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"](#) on page 5-2 and the name my-graph:

```

pgx:000> :loadGraph /opt/oracle/oracle-spatial-graph/property_
graph/examples/pgx/graphs/sample.adj.json my-graph
==> {
  "graphName" : "my-graph",
  "metaData" : {
    "primitives" : {
      "NUM_EDGES" : 4,
      "DATA_SOURCE_VERSION" : 1422390352000,
      "NUM_NODES" : 4,
      "CREATION_TS" : 1429637170234,
      "MEMORY_MB" : 0,
      "CREATION_REQUEST_TS" : 1429637170130
    },
    "graphConfig" : {
      "format" : "adj_list",
      "separator" : " ",
      "edge_props" : [ {
        "type" : "double",
        "name" : "cost"
      } ],
      "node_props" : [ {
        "type" : "integer",
        "name" : "prop"
      } ],
      "uri" :
"/opt/oracle/oracle-spatial-graph/property_graph/examples/pgx/graphs/sample.adj"
    },
    "edgeProperties" : {
      "cost" : "double"
    },
    "nodeProperties" : {
      "prop" : "integer"
    }
  }
}

```

The `:loadGraph` command returns a `LoadingResult` object, which holds the properties and some basic statistics about the graph, such as the number of loaded nodes. A string representation of the object is returned by `LoadingResult#toString()`, and is JSON-formatted for easy reading.

You can verify the type of returned objects by using the Groovy programming language's `_` variable, which holds the last result object:


```
pgx:000> _.getClass()
==> class oracle.pgx.api.LoadingResult
```

To save objects for later use, you can assign `_` to a variable after executing the command that generates the object. This example saves `my-graph` in a variable named `result`, and then retrieves the graph name:

```
pgx:000> :loadGraph /opt/oracle/oracle-spatial-graph/property_
graph/examples/pgx/graphs/sample.adj.json my-graph
==> ...
// _ holds the loading result
pgx:000> result = _
==> ...
pgx:000> result.getGraphName()
==> my-graph
```

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 In-Memory Analytics supports several graph formats.

The main steps are:

1. [Creating a Simple Graph File](#)
2. [Adding a Vertex Property](#)
3. [Using Strings as Node Identifiers](#)
4. [Adding an Edge Property](#)

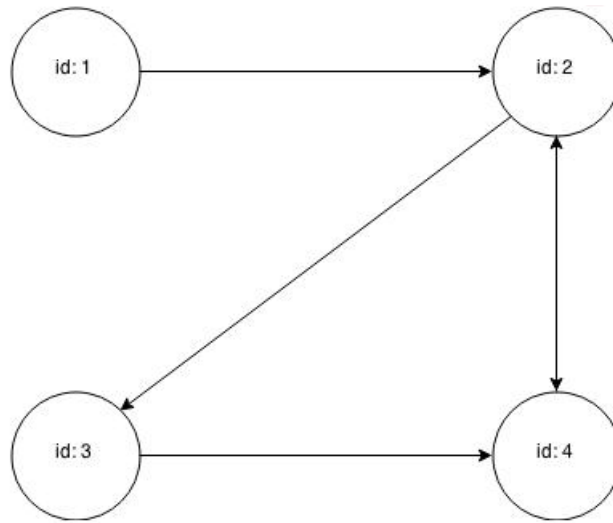
Creating a Simple Graph File

This example creates a small, simple graph in adjacency list format with no node or edge properties. Each line contains the node ID, followed by the node IDs its outgoing edges point to:

```
1 2
2 3 4
3 4
4 2
```

In this list, a single space separates the individual tokens. In-Memory Analytics 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 nodes 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 In-Memory Analytics 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();
```

Adding a Vertex Property

The graph in ["Creating a Simple Graph File"](#) on page 5-7 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 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: In-Memory Analytics supports only homogeneous graphs, in which all vertices have the same number and type of properties.

For In-Memory Analytics to read the modified data file, you must add a 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();
```

Using Strings as Node Identifiers

The previous examples used integer node IDs. The default in In-Memory Analytics is integer node IDs, but you can define a graph to use string node 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 node IDs consume much more memory than integer node IDs.

Any single or double quotes inside the string must be escaped with a backslash (\).

Newlines (\n) inside strings are not supported.

Adding an Edge Property

This example adds an edge property of type string to the graph. The edge properties are positioned after the destination 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();
```

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 In-Memory Analytics at a later time.

Storing graphs over HTTP/REST is currently not supported.

The options include:

- [Storing the Results of Analysis in a Node Property](#)
- [Storing a Graph in Edge-List Format on Disk](#)

Storing the Results of Analysis in a Node Property

This example reads a graph into memory and analyzes it using the Pagerank algorithm. This analysis requires a new node property to store the PageRank values. The following examples add a node property named `prop`.

Using the Shell to Add a Node Property

```
:loadGraph /opt/oracle/oracle-spatial-graph/property_
graph/examples/pgx/graphs/sample.adj.json sample
===> {

    "graphName" : "sample",
    [... rest of output omitted for brevity ...]
}
:createNodeProperty sample DOUBLE rank
===> rank
:pagerank sample 0.001 0.85 100 rank
```

Using Java to Add a Node Property

```
import oracle.pgx.api.LoadingResult;
import oracle.pgx.api.Pgx;
import oracle.pgx.common.types.PropertyType;
import oracle.pgx.config.GraphConfig;
import oracle.pgx.config.GraphConfigFactory;

GraphConfig config =
GraphConfigFactory.forAnyFormat().fromPath("/opt/oracle/oracle-spatial-graph/prop
erty_graph/examples/pgx/graphs/sample.adj.json");
Core core = Pgx.getCore();
LoadingResult lr = core.loadGraphWithProperties(sessionId, config).get();
String graphName = lr.getGraphName();
core.createNodeProperty(sessionId, graphName, PropertyType.DOUBLE, "rank").get();
Pgx.getBuiltinAlgorithms().pagerank(sessionId, graphName, 0.001, 0.85, 100,
"rank").get();
```

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 `ALL` to store all properties or `NONE` to store no properties. To specify individual properties, specify the property names as a list of strings.
- 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

```
:storeGraph sample EDGE_LIST /tmp/sample_pagerank.elist !["rank"] ALL false
==> {"format" : "edge_list", "node_props" : [{"name":"rank","type":"double"}],
"edge_props" : [{"name":"cost","type":"double"}], "uri":"/tmp/sample_
pagerank.elist", "node_id_type":"integer"}
```

Using Java to Store a Graph

```
import java.util.Arrays;
import java.util.List;
import oracle.pgx.api.Properties;
import oracle.pgx.config.Format;

List<String> nodeProperties = Arrays.asList(rankName));
core.storeGraphWithProperties(sessionId, graphName, Format.EDGE_LIST,
"/tmp/sample_pagerank.elist", nodeProperties, Properties.ALL, false).get();
```

Executing Built-in Algorithms

In-Memory Analytics contains a set of built-in algorithms that are available as Java APIs. This section describes the use of the in-memory analytics using Triangle Counting and Pagerank analytics as examples.

- [About In-Memory Analytics](#)
- [About the Analyst Interface](#)
- [Reading the Graph](#)
- [Running the Triangle Counting Algorithm](#)
- [Running the Pagerank Algorithm](#)

About In-Memory Analytics

In-Memory Analytics contains a set of built-in algorithms that are available as Java APIs.

In-Memory Analytics manages all data structures, including graphs, properties, and collections. After you create a data structure, you can refer to it by its string identifier. For example, after reading a graph into memory, you can refer to it by its graph name.

Thus, the Java APIs replace non-primitive types, like graphs and properties in a procedure declaration, with identifier strings.

For example, this is the Pagerank procedure signature:

```
procedure pagerank(G: graph, e,d: double, max: int; rank: nodeProp<double>)
```

In contrast, the following are the APIs for Pagerank:

Pagerank Java API

```
PgxFuture<Void> pagerank(String sessionId, String graphName, double e, double d,
int max, String rankName);
```

Pagerank Shell Interface

```
pgx:000> :h :pagerank
```

```
usage: :pagerank <String graphName> <double e> <double d> <int max> <String
rankName>
```

Classic pagerank algorithm. Time complexity: $O(E * K)$ with E = number of edges, K is a given constant (max iterations)

Arguments:

graphName - graph name

e - maximum error for terminating the iteration

d - damping factor

max - maximum number of iterations

rankName - (output) name of double node property to store result

About the Analyst Interface

In-Memory Analytics also provides a convenience layer on top of the BuiltInAlgorithms API named Analyst, which is both graph- and session-bound. It also wraps the result of the build-in algorithms into a PropertyProxy and returns it. In In-Memory Analytics, *proxies* can be used to conveniently access the values of data structures managed by the instance:

```
PgxFuture<PropertyProxy<Double>> pagerank(double e, double d, int max);
```

The In-Memory Analytics Shell does not provide commands for Analyst methods. However, an `:analyst` command returns an Analyst instance for a graph and the current shell session. See ["Using the Java API Inside the In-Memory Analytics Shell"](#) on page 5-29.

Reading the Graph

To read a graph, you must first open a session, as shown in the following examples. (The In-Memory Analytics shell automatically starts the session.)

Using the Shell to Read a Graph

```
cd $PGX_HOME
./bin/pgx
// starting the shell will create an implicit session

:loadGraph /opt/oracle/oracle-spatial-graph/property_
graph/examples/pgx/graphs/sample.adj.json G
===> {
```

```
"graphName" : "sample",
[... rest of output omitted for brevity ...]
```

Using Java to Read a Graph

```
import java.io.File;
import oracle.pgx.api.*;
import oracle.pgx.api.algorithms.*;
import oracle.pgx.config.*;
import org.apache.commons.io.FileUtils;

...

Core core = Pgx.getCore();
String sessionId = core.createSession("test").get();

GraphConfig graphConfig =
GraphConfigFactory.forAnyFormat().fromPath("/opt/oracle/oracle-spatial-graph/prope
rty_graph/examples/pgx/graphs/sample.adj.json");
LoadingResult lr = core.loadGraphWithProperties(sessionId, graphConfig).get();
String graphName = lr.getGraphName();
```

Running the Triangle Counting Algorithm

Triangle Counting is defined on *undirected* graphs only. In In-Memory Analytics however, graphs are always *directed* by default after loading. Therefore, you must create an undirected copy of a graph before running Triangle Counting.

The `undirectGraph` procedure returns a list of names. The first one is always the name of the new graph, followed by the names of the transformed properties.

Using the Shell to Run Triangle Counting

```
:undirectGraph G U

:countTriangles U
==> 1
```

Using Java to Run Triangle Counting

```
import java.util.List;
import oracle.pgx.api.Pgx;
import oracle.pgx.api.algorithms.BuiltinAlgorithms;

List<String> result = core.createUndirectedGraph(sessionId, graphName).get();
String uGraphName = result.get(0);

BuiltinAlgorithms algorithms = Pgx.getBuiltinAlgorithms();
long triangles = algorithms.countTriangles(sessionId, uGraphName).get();
```

The algorithm finds one triangle in the sample graph.

Tip: When using the In-Memory Analytics 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`.

Running the Pagerank Algorithm

Pagerank computes a rank value between 0 and 1 for each node in the graph and stores the values in a `double` property. The algorithm therefore takes a *node property* of type `double` for the output.

In In-Memory Analytics, there are two types of node 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 creating or copying properties using the In-Memory Analytics Core API.

This example obtains the top three nodes with the highest Pagerank values. It uses a transient node 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.

To access the values of the computed rank property, you must wrap it in a `PropertyProxy` object. **Proxies** are utility classes for read-only access to data structures managed by In-Memory Analytics.

Using the Shell to Run Pagerank

```
:createNodeProperty G DOUBLE rank

:pagerank G 0.001 0.85 100 rank

:propProxy G rank
proxy = _

proxy.getTopKValues(3)
==> [128=0.1402019732468347, 333=0.12002296283541904, 99=0.09708583862990475]
```

This example passes a `DOUBLE` as the second Shell command parameter, although typically this parameter is a `PropertyType` object. However, `PropertyType` is an enum, and the Shell automatically performs a type coercion.

This example uses the Groovy `_` variable to retrieve the last returned object and store it in a variable named `proxy`.

You can skip the `PropertyProxy` indirection by using the two convenience commands `printTopKValues` and `printBottomKValues` to quickly print the extremes in JSON format without first creating a `PropertyProxy` instance:

```
:printTopK G rank 3
==> {
  "128" : 0.1402019732468347,
  "333" : 0.12002296283541904,
  "99" : 0.09708583862990475
}
:printBottomK G rank 2
==> {
  "99" : 0.09708583862990475,
  "1908" : 0.09708583862990475
}
```

Using Java to Run Pagerank

```
import java.util.List;
```

```
import java.util.Map;

import oracle.pgx.api.Pgx;
import oracle.pgx.api.algorithms.BuiltinAlgorithms;
import oracle.pgx.api.analyst.PropertyProxy;
import oracle.pgx.common.types.PropertyType;

String rankName = core.createNodeProperty(sessionId, graphName,
PropertyType.DOUBLE).get();

String rankName = algorithms.pagerank(sessionId, graphName, 0.001 0.85 100,
rankName).get();

PropertyProxy<Double> proxy = core.<Double> getPropertyProxy(sessionId, graphName,
rankName).get();
for (Map.Entry<Object, Double> entry : proxy.getTopKValues(3)) {
    System.out.println(entry.getKey() + "=" + entry.getValue());
}
```

Creating Subgraphs

You can create subgraphs based on a loaded graph. You can use filter expressions or create bipartite subgraphs based on a 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 Node List to Create a Bipartite Subgraph](#)

For information about reading a graph into memory, see [Reading Graph Data into Memory](#).

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 nodes and four edges. For an edge to match the filter expression `src.prop == 10`, the source node `prop` property must equal 10. Two edges match that filter expression, as shown in [Figure 5-3](#).

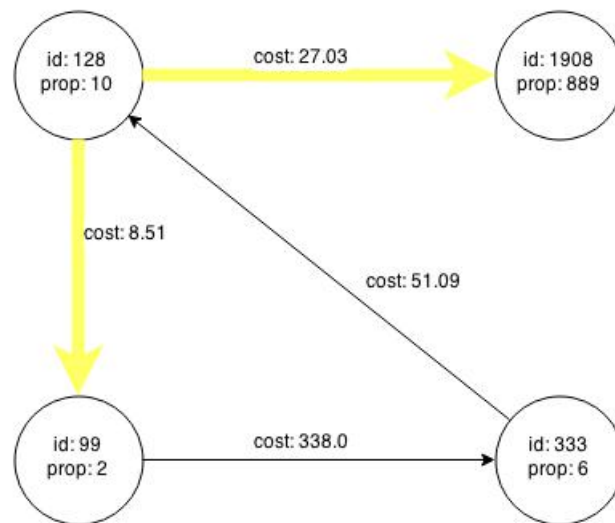
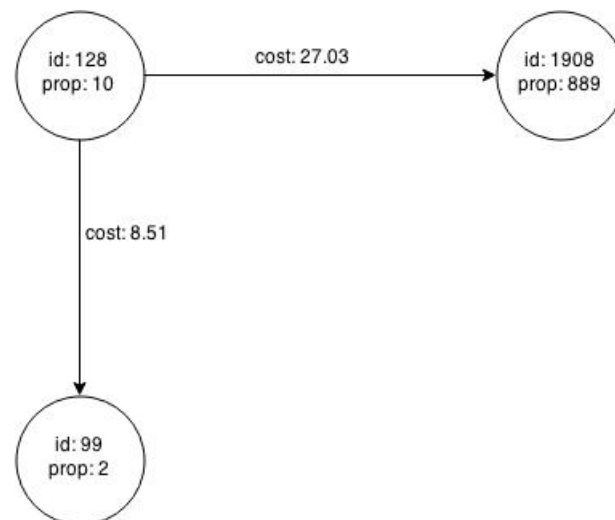
Figure 5-3 Edges Matching `src.prop == 10`

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 node or a set of nodes is difficult. For example, selecting only the node with the property value 10 is impossible, because the only way to match the node 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 node, destination node, and the edge itself in the result.

Using a Simple Filter to Create a Subgraph

The following examples create the subgraph described in ["About Filter Expressions"](#) on page 5-16.

Using the Shell to Create a Subgraph

```
:createSubgraphFromFilter sample "src.prop == 10" sample_filtered
```

```

===> [sample_filtered, prop, cost]
// return value: [<graph name>, <node property 1>, ..., <node property n>, <edge
property 1>, ..., <edge property n>]

```

Using Java to Create a Subgraph

```

import java.util.Arrays;
import java.util.List;

List<String> nodePropNames = Arrays.asList("prop");
List<String> edgePropNames = Arrays.asList("cost");

String filterExpression = "src.prop == 10";

// return value: [<graph name>, <node property 1>, ..., <node property n>, <edge
property 1>, ..., <edge property n>]
List<String> result = core.createSubgraphFromFilter(sessionId, graphName,
nodePropNames, edgePropNames, filterExpression, "sample_filtered").get();
String subgraphName = result.get(0);

```

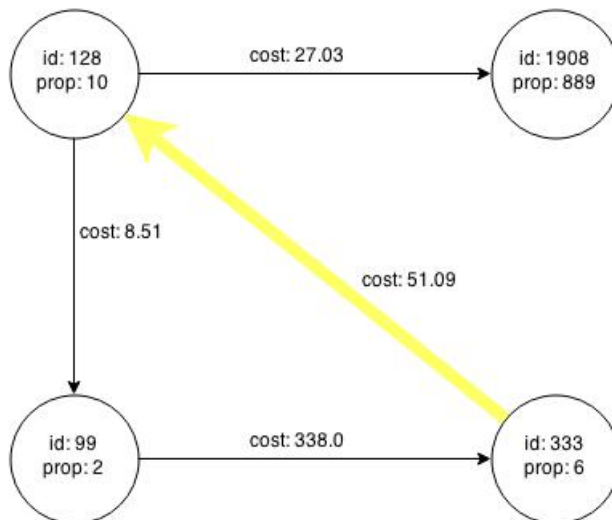
Using a Complex Filter to Create a Subgraph

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 node with an `outDegree` greater than 1.

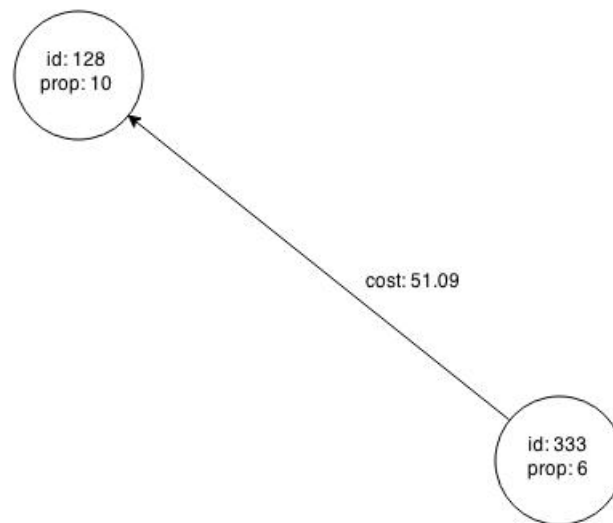
```
dst.outDegree() > 1 && edge.cost > 50
```

One edge in the sample graph matches this filter expression, as shown in [Figure 5–5](#).

Figure 5–5 Edges Matching the `outDegree` Filter



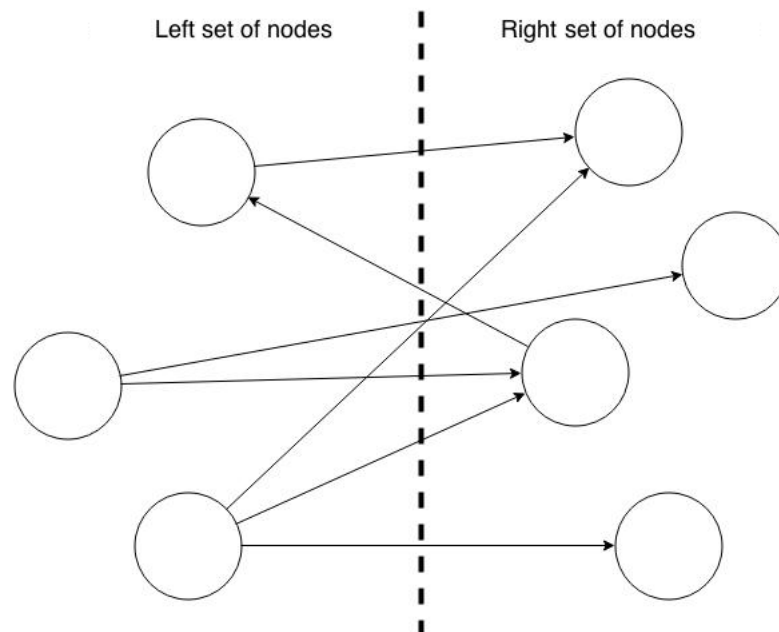
[Figure 5–6](#) shows the graph that results when the filter is applied. The filter excludes the edges associated with nodes 99 and 1908, and so excludes those nodes also.

Figure 5–6 Graph Created by the outDegree Filter

Using a Node List to Create a Bipartite Subgraph

You can create a bipartite subgraph by specifying a set of nodes, which are used as the left side. A bipartite subgraph has edges only between the left set of nodes and the right set of nodes. There are no edges within those sets, such as between two nodes on the left side. In In-Memory Analytics, nodes 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 node collection and fill it with the nodes for the left side.

Using the Shell to Create a Bipartite Subgraph

```
:createCollection sample SET NODE nodeset
```

```

====> nodeset
:addAll nodeset ![333, 99]
====> nodeset

:createBipartiteSubgraphFromLeftSet2 sample !["prop"] NONE nodeset sample_
bipartite isLeft
====> [sample_bipartite, isLeft, prop]

```

Using Java to Create a Bipartite Subgraph

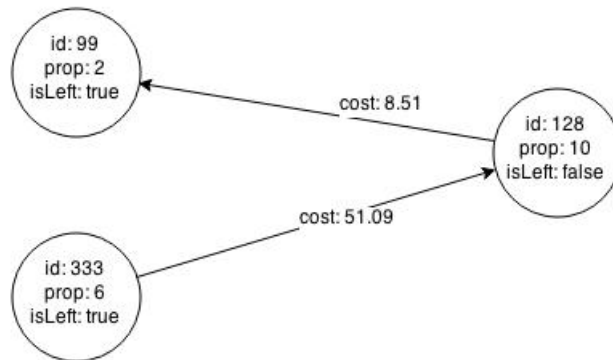
```

import oracle.pgx.common.types.CollectionType;
import oracle.pgx.common.types.EntityType;
import oracle.pgx.api.Properties;
...
List<String> nodePropNames = Arrays.asList("prop");
String nodeSetName = core.createCollection(sessionId, graphName,
CollectionType.SET, EntityType.NODE, "nodeSet").get();
List<Integer> leftSetNodes = Arrays.asList(333, 99);
core.addAllToCollection(sessionId, nodeSetName, leftSetNodes).get();
// Note: Edge properties are not currently supported in bipartite subgraphs
List<String> result = core.createBipartiteSubgraphFromLeftSet(sessionId,
graphName, nodePropNames, Properties.NONE, nodeSetName, "sample_bipartite",
"isLeft").get();
String subgraphName = result[0];

```

When you create a subgraph, In-Memory Analytics automatically creates a Boolean node property that indicates whether the node is on the left side. You can specify a unique name for the property.

The resulting bipartite subgraph looks like this:



Node 1908 is excluded from the bipartite subgraph. The only edge that connected that node extended from 128 to 1908. The edge was removed, because it violated the bipartite properties of the subgraph. Node 1908 had no other edges, and so was removed also.

Deploying to Jetty

You can deploy In-Memory Analytics to Eclipse Jetty, Apache Tomcat, or Oracle WebLogic. This example shows how to deploy In-Memory Analytics as a web application with Eclipse Jetty.

1. Copy the In-Memory Analytics 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 Analytics configuration files.

The configuration file (`pgx.conf`) and the logging parameters (`log4j.xml`) for the In-Memory Analytics 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

About the Authentication Mechanism

The In-Memory Analytics 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.

Deploying to Apache Tomcat

You can deploy In-Memory Analytics to Eclipse Jetty, Apache Tomcat, or Oracle WebLogic. This example shows how to deploy In-Memory Analytics as a web application with Apache Tomcat.

In-Memory Analytics 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 Analytics 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/>

Deploying to Oracle WebLogic Server

You can deploy In-Memory Analytics to Eclipse Jetty, Apache Tomcat, or Oracle WebLogic Server. This example shows how to deploy In-Memory Analytics as a web application with Oracle WebLogic Server.

- [Installing Oracle WebLogic Server](#)
- [Deploying In-Memory Analytics](#)
- [Verifying That the Server Works](#)

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>

Deploying In-Memory Analytics

To deploy In-Memory Analytics 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
-passwd 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
```

Verifying That the Server Works

Verify that you can connect to the server:

```
$PGX_HOME/bin/pgx --base_url scott:tiger123@localhost:7001/pgx
```

Connecting to the In-Memory Analytics Server

After property graph in-memory analytics 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 analytics server.

- [Connecting with the In-Memory Analytics Shell](#)
- [Connecting with Java](#)
- [Connecting with an HTTP Request](#)

Connecting with the In-Memory Analytics Shell

The simplest way to connect to an In-Memory Analytics 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 Analytics 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, In-Memory Analytics currently does not provide remote support for the Control API.

About Logging HTTP Requests

The In-Memory Analytics 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:000> :loglevel oracle.pgx DEBUG
==> log level of oracle.pgx logger set to DEBUG
pgx:000> :loadGraph 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,"numNode
s":4},"graphName":"sample","nodeProperties":{"prop":"integer"},"edgeProperties":{"
cost":"double"}}
==> {
  "nodeProperties" : {
    "prop" : "integer"
  },
  "edgeProperties" : {
    "cost" : "double"
  },
  "stats" : {
    "numNodes" : 4,
    "numEdges" : 4,
    "loadingTimeMillis" : 0,
    "estimatedMemoryMegabytes" : 0
  },
  "graphName" : "sample"
}
```

This example requires that the graph URI points to a file that the *In-Memory Analytics* server can access using HTTP or HDFS.

Connecting with Java

You can specify the base URL when you initialize In-Memory Analytics using Java. An example is as follows. A URL to an In-Memory Analytics server is provided to the `getInMemAnalyst` API call.

```
import oracle.pg.nosql.*;
PgNosqlGraphConfig cfg =
GraphConfigBuilder.forNosql().setName("mygraph").setHosts(...).build();
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);
Analyst analyst = opg.getInMemAnalyst("http://scott:tiger@hostname:port/pgx");
```

Connecting with an HTTP Request

The In-Memory Analytics shell uses HTTP requests to communicate with the In-Memory Analytics 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 Analytics* UUID, which identifies the resource that holds the result of the request. Many In-Memory Analytics 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 In-Memory Analytics 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/sa
mple.adj"}}'
Response: {"futureId":"15fc72e9-42e9-4527-9a31-bd20eb0adafb"}
```



```
// get the result using the In-Memory Analytics future UUID.
HTTP GET
'http://scott:tiger@localhost:8080/pgx/future/session/session-shell-42v3b9n7/resul
t/15fc72e9-42e9-4527-9a31-bd20eb0adafb'
Response:
{"stats":{"loadingTimeMillis":0,"estimatedMemoryMegabytes":0,"numNodes":4,"numEdge
s":4},"graphName":"sample","nodeProperties":{"prop":"integer"},"edgeProperties":{"
cost":"double"}}
```

Reading and Storing Data in HDFS

In-Memory Analytics supports the Hadoop Distributed File System (HDFS). This example shows how to read and access graph data in HDFS using In-Memory Analytics 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 Parallel In-Memory Analytics 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

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 In-Memory Analytics, as shown in the following examples.

Using the Shell to Load the Graph from HDFS

```
:loadGraph hdfs:/user/pgx/sample.adj.json G
```

```
==> {
  "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.*;
import oracle.pgx.config.*;

.
.
.

GraphConfig graphConfig =
GraphConfigFactory.forAnyFormat().fromHdfs("/user/pgx/sample.adj.json");
LoadingResult lr = Pgx.getCore().loadGraphWithProperties(sessionId,
graphConfig).get();
String graphName = lr.getGraphName();
```

Storing Graph Snapshots in HDFS

The In-Memory Analytics binary format (.pgb) is a proprietary binary graph format for In-Memory Analytics. Fundamentally, a .pgb file is a binary dump of a graph and its property data, and it is efficient for In-Memory Analytics 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

```
:storeGraph G PGB hdfs:/user/pgx/sample.pgb ALL ALL true
```

Using Java to Store a Graph in HDFS

```
import oracle.pgx.api.Pgx;
import oracle.pgx.config.Format;
import oracle.pgx.config.GraphConfig;
```

```
GraphConfig pgbGraphConfig = Pgx.getCore().storeGraphWithProperties(sessionId,
graphName,
    Format.PGB, "hdfs:/user/pgx/sample.pgb", true).get();
```

To verify that the PGB file was created, list the files in the /user/pgx HDFS directory:

```
hadoop fs -ls /user/pgx
```

Compiling and Running a Java Application in Hadoop

The following is the HdfsExample Java class for the previous examples:

```
import oracle.pgx.api.*;
import oracle.pgx.config.*;

public class HdfsExample {

    public static void main(String[] mainArgs) throws Exception {
        Core core = Pgx.getCore();

        String sessionId = core.createSession("test").get();
        GraphConfig graphConfig =
GraphConfigFactory.forAnyFormat().fromHdfs("/user/pgx/sample.adj.json");
        LoadingResult lr = Pgx.getCore().loadGraphWithProperties(sessionId,
graphConfig).get();
        String graphName = lr.getGraphName();

        GraphConfig pgbGraphConfig = Pgx.getCore().storeGraphWithProperties(sessionId,
graphName,
            Format.PGB, "hdfs:/user/pgx/sample.pgb", true).get();

        // load PGB graph
        Pgx.getCore().loadGraphWithProperties(sessionId, pgbGraphConfig).get();
    }
}
```

These commands compile the HdfsExample class:

```
cd $PGX_HOME
mkdir classes
javac -cp lib/common/*:lib/embedded/*:third-party/* HdfsExample.java -d classes
```

This command runs the HdfsExample class:

```
java -cp lib/common/*:lib/embedded/*:third-party/*:classes:$HADOOP_HOME/etc/hadoop\
-Dlog4j.configuration=file:conf/log4j.xml HdfsExample
```

Running In-Memory Analytics as a YARN Application

In this example you will learn how to start, stop and monitor In-Memory Analytics servers on a Hadoop cluster via Hadoop NextGen MapReduce (YARN) scheduling.

- [Starting and Stopping In-Memory Analytics Services](#)
- [Connecting to In-Memory Analytics Services](#)
- [Monitoring In-Memory Analytics Services](#)

Starting and Stopping In-Memory Analytics Services

Before you can start In-Memory Analytics as a YARN application, you must configure the In-Memory Analytics YARN client.

Configuring the In-Memory Analytics YARN Client

The In-Memory Analytics 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.

Starting a New In-Memory Analytics Service

To start a new In-Memory Analytics 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 Analytics service launched are displayed.

About Long-Running In-Memory Analytics Services

In-Memory Analytics 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 Analytics server keeps running until you or Hadoop explicitly stop it.

Stopping In-Memory Analytics Services

To stop a running In-Memory Analytics 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 Analytics service:

```
yarn logs -applicationId appId
```

Connecting to In-Memory Analytics Services

You can connect to In-Memory Analytics services in YARN the same way you connect to any In-Memory Analytics server. For example, to connect the Shell interface with the In-Memory Analytics 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.

Monitoring In-Memory Analytics Services

To monitor In-Memory Analytics 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
```

Using the Java API Inside the In-Memory Analytics Shell

The In-Memory Analytics Shell is built on `groovysh` from the Groovy package. The Shell is thus not limited to In-Memory Analytics commands, because most Java syntax is also valid Groovy syntax. You can use all Java APIs provided by In-Memory Analytics to write complex Shell scripts.

The following example uses the In-Memory Analytics Analyst Java API in the shell to run PageRank on a graph and print the results. The shell loads most of the necessary imports by default, so your script does not need to import them explicitly.

```
pgx:000> config =
GraphConfigFactory.forAnyFormat().fromPath("/opt/oracle/oracle-spatial-graph/property_graph/examples/pgx/graphs/sample.adj.json")
==> {"edge_props":[{"name":"cost","type":"double"}], "node_props":[{"name":"prop","type":"integer"}], "uri":"PGX_HOME/examples/graphs/sample.adj", "format":"adj_list", "separator":" "}
pgx:000> analyst = Pgx.createAnalyst(config).get()
==> oracle.pgx.api.analyst.Analyst@7d8f73c9
pgx:000> e = 0.001
==> 0.001
pgx:000> d = 0.85
==> 0.85
pgx:000> max = 100
==> 100
pgx:000> rank = analyst.pagerank(e, d, max).get()
==> oracle.pgx.engine.instance.PropertyProxyImpl@7eb61363
pgx:000> for(x in rank.getValues())
pgx:001>     println x.getKey() + ": " + x.getValue()
128: 0.1402019732468347
1908: 0.09708583862990475
99: 0.09708583862990475
333: 0.12002296283541904
==> null
pgx:000> analyst.close()
==> null
```

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 fluent](#)
- [Apache Groovy](#)
- [Apache htrace](#)
- [Apache HTTP Client](#)
- [Apache HTTPComponents Core](#)
- [Apache Jena](#)
- [Apache Log4j](#)
- [Apache Lucene](#)
- [Apache Xerces2](#)
- [Apache xml-commons](#)
- [Cloudera CDH](#)
- [Fastutil](#)
- [GeoNames Data](#)
- [Geospatial Data Abstraction Library \(GDAL\)](#)
- [Google Guava](#)
- [Google Guice](#)

- [Google protobuf](#)
- [Jackson](#)
- [Jansi](#)
- [Jettison](#)
- [JLine](#)
- [Javassist](#)
- [Jung](#)
- [MessagePack](#)
- [Netty](#)
- [Slf4j](#)
- [Tinkerpop Blueprints](#)
- [Tinkerpop Gremlin](#)
- [Tinkerpop Pipes](#)

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/>) (listed below):

ANTLR 3

This product was build using ANTLR, which was provided to Oracle under the following terms:

Copyright (c) 2010 Terence Parr

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 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.

AOP Alliance

LICENCE: all the source code provided by AOP Alliance is Public Domain.

Apache Commons CLI

Copyright 2001-2009 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

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.

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

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

Apache Commons IO

Copyright 2002-2012 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation
(<http://www.apache.org/>).

Apache Commons Lang

Copyright 2001-2010 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation
(<http://www.apache.org/>).

Apache Commons Logging

Copyright 2003-2007 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation
(<http://www.apache.org/>).

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/>).

Apache Groovy

Copyright 2009-2015 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation
(<http://www.apache.org/>).

Apache htrace

Copyright 2009-2015 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation
(<http://www.apache.org/>).

Apache HTTP Client

Copyright 1999-2013 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation
(<http://www.apache.org/>).

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>

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.

Apache Log4j

Copyright 2007 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Apache Lucene

Copyright 2011-2012 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Apache Xerces2

Copyright 1999-2012 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

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>

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.

Fastutil

Fastutil is available under the Apache License, Version 2.0.

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/>.

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.

Google Guava

Guava is licensed under the Apache License, Version 2.0

Copyright 2006 - 2011 Google, Inc. All rights reserved.

Google Guice

Guice is licensed under the Apache License, Version 2.0

Copyright 2006 – 2011 Google, Inc. All rights reserved.

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.

Jackson

Copyright 2009 FasterXML, LLC

Jackson is available under the Apache License, Version 2.0.

Jansi

Copyright (C) 2009, Progress Software Corporation and/or its subsidiaries or affiliates.

Jansi is available under the Apache License, Version 2.0.

Jettison

Copyright 2006 Envoi Solutions LLC.

Jettison is available under the Apache License, Version 2.0.

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.

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.

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.

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.

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/jboss-cache/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>

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.

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.

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.

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.

