H2O Developer Cookbook

**Preface**

H2O is an in-memory engine for predictive analytics and machine learning.

Discuss the differences between ValueArray and Fluid Vector here.

1 Getting Started from an IDE 7

1.1 Cloning H2O from Github 7

1.2 Downloading Java 7

1.3 Importing the H2O project into Eclipse 7

1.4 Compiling H2O in Eclipse 7

1.5 Starting H2O in Eclipse 7

1.6 Importing the H2O project into IntelliJ IDEA 7

1.7 Compiling H2O in IntelliJ IDEA 7

1.8 Starting H2O in IntelliJ IDEA 7

2 Getting Started from the Command Line 8

2.1 Compiling H2O from the command line 8

2.2 Running H2O from the command line 8

2.3 Running a multi-node H2O cluster from the command line 8

3 Reading Data 9

3.1 Reading a CSV file from local disk 9

3.2 Reading a CSV file from HDFS 9

3.3 Reading a CSV file from S3 9

3.4 Reading a directory of CSV files from local disk 9

3.5 Reading a directory of CSV files from HDFS 9

3.6 Reading a directory of CSV files from S3 9

4 Introduction to Representation of Data 10

4.1 Element 10

4.1.1 Double elements 10

4.1.2 Long elements 10

4.1.3 Enum (aka Categorical) elements 10

4.1.4 Missing (aka NA) values 10

4.1.5 Not-a-Number (aka NaN) values 10

4.1.6 Infinity values 10

4.2 Chunk 10

4.2.1 Reading a double element from a Chunk 10

4.2.2 Reading a long element from a Chunk 10

4.2.3 Updating a double element to a Chunk 10

4.2.4 Updating a long element to a Chunk 10

4.3 Vec 10

4.3.1 Reading an element from a Vec 11

4.3.2 Updating an element to a Vec 11

4.3.3 Testing whether a Vec of integers is an Enum (aka Categorical) or not 11

4.3.4 Printing the list of Domains of an Enum Vec (aka Levels of a Categorical Vec) 11

4.3.5 Updating a Vec element with an Enum value that has never been used before 11

4.3.6 Accessing Vec stats that are computed automatically (like min, max) 11

4.4 Frame 11

4.4.1 Accessing a Vec from a Frame 11

4.4.2 Printing all column (aka Vec) names in a Frame 11

4.4.3 Adding an existing Vec to a Frame 11

4.4.4 Removing a Vec from a Frame 11

4.4.5 Creating a new Frame with a subset of Vecs from an existing Frame 11

4.4.6 Removing all Frame references to a Vec and reclaiming its memory 11

4.4.7 Creating a new double Vec from nothing and adding it to a Frame 11

4.4.8 Creating a new long Vec from nothing and adding it to a Frame 11

4.4.9 Creating a new Enum Vec from nothing and adding it to a Frame 11

5 Sharing Data Across Nodes 12

5.1 DKV (Distributed Key/Value Store) 12

5.1.1 Reading a value from the DKV 12

5.1.2 Writing a new value to the DKV 12

5.1.3 Updating a value in the DKV 12

5.1.4 Removing a value from the DKV 12

5.1.5 Writing multiple values to the DKV 12

5.1.6 Updating multiple values in the DKV 12

5.1.7 Removing multiple values from the DKV 12

5.2 UKV (User-level Key/Value Store) 12

5.2.1 Adding compound objects to the UKV 12

5.2.2 Removing compound objects from the UKV 12

5.2.3 Looking at UKV objects using the Web UI 12

5.3 Vecs and Frames 12

5.3.1 Reading a value updated by a different node 12

5.4 Chunks and Data Parallelism 12

5.4.1 Observing how Chunk arrays are the basic unit of parallelism 12

5.4.2 Observing how Chunk arrays get sprayed across a Cluster for small data 12

5.4.3 Observing how Chunk arrays get sprayed across a Cluster for big data 12

6 Clusters 13

6.1 Knowing when a cluster is ready for use 13

6.2 Checking cluster health 13

7 Working with Data that exceeds the size of Memory 14

7.1 Managing the location of temporary files by setting ICE\_ROOT 14

7.2 Reading a huge CSV file 14

8 Object Serialization 15

8.1 Freezable and Iced 15

8.1.1 Defining a new Iced object 15

8.1.2 Sending an Iced object across the network 15

9 Tasks 16

9.1 DRemoteTask 16

9.1.1 Running a piece of Java code on all H2O nodes 16

9.2 MRTask2 16

9.2.1 Writing a new MRTask2 task to sum the values of a column 16

9.2.2 Returning the value of a MRTask2 task as a member of an Iced object 16

9.2.3 Returning the value of a MRTask2 task by adding new Vecs (aka columns) to a Frame 16

10 Jobs 17

10.1 Creating a new Job 17

10.2 Starting a Job 17

10.3 Monitoring a Job 17

11 Scala and H2O 18

11.1 Getting started with Shalala 18

11.2 Reading data in from Scala 18

11.3 Writing Scala scripts for H2O 18

11.4 Writing a new MRTask2 task in Scala 18

11.5 Calling Java MRTask2 tasks from Scala 18

11.6 Running Scala code in a multi-node H2O cluster 18

12 REST API 19

12.1 Creating a new REST API endpoint 19

13 Logging 20

13.1 Writing new log statements 20

13.2 Finding log files 20

13.3 Configuring a custom logging setup 20

14 Using H2O as a Dependency for a new Project 21

14.1 Compiling your project with H2O as a dependency 21

14.2 Registering a new REST API endpoint with H2O 21

14.3 Running an H2O cluster with a new REST endpoint from IntelliJ IDEA 21

14.4 Running an H2O cluster with a new REST endpoint from the command line 21

15 Embedding H2O 22

15.1 Running H2O inside a Hadoop map task 22

16 Testing 23

16.1 Running JUnit tests from the command line 23

16.2 Running JUnit tests from an IDE 23

16.3 Running one RUnit test from the command line 23

16.4 Running one RUnit test from RStudio 23

16.5 Running all the RUnit tests from the command line 23

16.6 Running one Python test from the command line 23

16.7 Running all the Python tests from the command line 23

16.8 Running H2O CRAN package examples from the command line 23

16.9 Adding a JUnit test 23

16.10 Adding an RUnit test 23

16.11 Adding a Python test 23

17 Common Pitfalls 24

17.1 Frequently hit assertions by new H2O developers 24

17.1.1 Missing chunks 24

17.1.2 Leaking keys 24

17.2 Frequently made programming mistakes 24

17.2.1 Failure to block (aka wait) 24

17.2.2 Calling an MRTask2 from inside another MRTask2 24

17.2.3 Running out of Fork/Join threads 24

18 Things H2O does not Support 25

18.1 Row names 25

18.2 Unique per-row strings for many rows 25

18.3 High availability (HA) 25

19 Glossary 26

# Getting Started from an IDE

## Cloning H2O from Github

## Downloading Java

## Importing the H2O project into Eclipse

## Compiling H2O in Eclipse

## Starting H2O in Eclipse

## Importing the H2O project into IntelliJ IDEA

## Compiling H2O in IntelliJ IDEA

## Starting H2O in IntelliJ IDEA

# Getting Started from the Command Line

## Compiling H2O from the command line

## Running H2O from the command line

## Running a multi-node H2O cluster from the command line

# Reading Data

## Reading a CSV file from local disk

## Reading a CSV file from HDFS

## Reading a CSV file from S3

## Reading a directory of CSV files from local disk

## Reading a directory of CSV files from HDFS

## Reading a directory of CSV files from S3

# Introduction to Representation of Data

Data in H2O is referenced through a Frame. Frames are loosely analogous to Data Frames in R, although in H2O the actual data is stored in vectors (a Vec) rather than in the Frame itself. A Vec may be referenced by more than one Frame. Each Frame is composed of one or more Vecs. Each Vec is composed of one or more Chunks. Each Chunk is composed on one or more Elements.

## Element

Some text here about Elements.

### Double elements

### Long elements

### Enum (aka Categorical) elements

### Missing (aka NA) values

### Not-a-Number (aka NaN) values

### Infinity values

## Chunk

Some text here about Chunks

### Reading a double element from a Chunk

### Reading a long element from a Chunk

### Updating a double element to a Chunk

### Updating a long element to a Chunk

## Vec

Some text here about Vecs

### Reading an element from a Vec

### Updating an element to a Vec

### Testing whether a Vec of integers is an Enum (aka Categorical) or not

### Printing the list of Domains of an Enum Vec (aka Levels of a Categorical Vec)

### Updating a Vec element with an Enum value that has never been used before

### Accessing Vec stats that are computed automatically (like min, max)

## Frame

Some text here about Frames

### Accessing a Vec from a Frame

### Printing all column (aka Vec) names in a Frame

### Adding an existing Vec to a Frame

### Removing a Vec from a Frame

### Creating a new Frame with a subset of Vecs from an existing Frame

### Removing all Frame references to a Vec and reclaiming its memory

### Creating a new double Vec from nothing and adding it to a Frame

### Creating a new long Vec from nothing and adding it to a Frame

### Creating a new Enum Vec from nothing and adding it to a Frame

# Sharing Data Across Nodes

## DKV (Distributed Key/Value Store)

DKV stands for Distributed Key/Value store. The DKV is the high-performance atomic distributed store that provides the clustering support for data in H2O.

### Reading a value from the DKV

### Writing a new value to the DKV

### Updating a value in the DKV

### Removing a value from the DKV

### Writing multiple values to the DKV

### Updating multiple values in the DKV

### Removing multiple values from the DKV

## UKV (User-level Key/Value Store)

The UKV is an abstraction on top of the DKV.

### Adding compound objects to the UKV

### Removing compound objects from the UKV

### Looking at UKV objects using the Web UI

## Vecs and Frames

### Reading a value updated by a different node

## Chunks and Data Parallelism

### Observing how Chunk arrays are the basic unit of parallelism

### Observing how Chunk arrays get sprayed across a Cluster for small data

### Observing how Chunk arrays get sprayed across a Cluster for big data

# Clusters

## Knowing when a cluster is ready for use

## Checking cluster health

# Working with Data that exceeds the size of Memory

## Managing the location of temporary files by setting ICE\_ROOT

## Reading a huge CSV file

# Object Serialization

## Freezable and Iced

### Defining a new Iced object

### Sending an Iced object across the network

# Tasks

## DRemoteTask

### Running a piece of Java code on all H2O nodes

## MRTask2

### Writing a new MRTask2 task to sum the values of a column

### Returning the value of a MRTask2 task as a member of an Iced object

### Returning the value of a MRTask2 task by adding new Vecs (aka columns) to a Frame

# Jobs

A Job is a major piece of work that gets added to the Jobs list and is visible in the Jobs Web UI page. A Job is generally reserved for things that produce an output you would be interested keeping around (a model, for example). Top-level algorithms like Random Forest are implemented as a Job.

## Creating a new Job

## Starting a Job

## Monitoring a Job

# Scala and H2O

## Getting started with Shalala

## Reading data in from Scala

## Writing Scala scripts for H2O

## Writing a new MRTask2 task in Scala

## Calling Java MRTask2 tasks from Scala

## Running Scala code in a multi-node H2O cluster

# REST API

## Creating a new REST API endpoint

# Logging

## Writing new log statements

## Finding log files

## Configuring a custom logging setup

# Using H2O as a Dependency for a new Project

## Compiling your project with H2O as a dependency

## Registering a new REST API endpoint with H2O

## Running an H2O cluster with a new REST endpoint from IntelliJ IDEA

## Running an H2O cluster with a new REST endpoint from the command line

# Embedding H2O

## Running H2O inside a Hadoop map task

# Testing

## Running JUnit tests from the command line

## Running JUnit tests from an IDE

## Running one RUnit test from the command line

## Running one RUnit test from RStudio

## Running all the RUnit tests from the command line

## Running one Python test from the command line

## Running all the Python tests from the command line

## Running H2O CRAN package examples from the command line

## Adding a JUnit test

## Adding an RUnit test

## Adding a Python test

# Common Pitfalls

## Frequently hit assertions by new H2O developers

### Missing chunks

### Leaking keys

## Frequently made programming mistakes

### Failure to block (aka wait)

### Calling an MRTask2 from inside another MRTask2

### Running out of Fork/Join threads

# Things H2O does not Support

It’s worth pointing out the following list of items that are available in some other popular languages and frameworks. Some of these are in the H2O roadmap and some are not a fit for H2O.

## Row names

Unlike R Data Frames, H2O does not support naming individual rows. Allowing this is one factor that inhibits the R runtime from scaling well. Don’t expect H2O to ever support this.

## Unique per-row strings for many rows

H2O currently turns columns with small numbers of unique strings into an Enum, and turns columns with large numbers of unique strings into N/As.

In the future, H2O will be able to read in columns with large numbers of strings and treat a separate “String” datatype as a top-level datatype. These will be unusable for modeling purposes, but flow through to be able to print the value as an output. For some applications, this may serve as an adequate substitute for row names.

## High availability (HA)

H2O is currently vulnerable to single-node failures rendering the entire cluster inoperable. Since H2O has an In-Memory architecture, the response to this is to manually kill and restart the cluster and any jobs that were in progress. Proper HA support is on the roadmap, but a (perhaps sufficient) step on the way there is the ability to checkpoint and restart model building.

# Glossary

Categorical

Cluster (of H2O nodes)

Domain

DKV

DRemoteTask

Enum

Fluid Vector (FV)

Freezable

Future

Fork/Join

H2OCountedCompleter

Iced

In-Core

In-Memory

Job

Key

Level

MRTask2

Node (in a Cluster)

Out-of-Core

REST API endpoint

UKV

ValueArray (VA or sometimes just Array)

Vector Group