Input Data Formats For Hadoop

Overview of Input Data Formats

Splits. Hadoop organizes input data files into splits. Each Hadoop file split is a sequence of bytes from the original file.

The key issue with splits in Hadoop is that they are done *without parsing the input data*. What this means is that when a split is created, Hadoop does not attempt to determine if it is breaking any specific structures.

Because of this issue, parsing certain types of data is non-trivial.

Basic Input Data Formats `org.apache.hadoop.mapreduce.lib.input` contains a number of built-in classes for input formats.

Class	Input Type
`FileInputFormat`	Generic file input
`TextInputFormat`	Generic text file input
`KeyValueTextInputFormat`	User-defined key-value pair records as input
`FixedLengthInputFormat`	Fixed length records as input
`NLineInputFormat`	Allows user control over #lines per mapper
`SequenceFileInputFormat`	Input an instance of `org.apache.hadoop.io.SequenceFile`

Each type of input data format differs from others in the following ways:

- Split organization
- Record organization:
  - `Mapper` input key type
  - `Mapper` input value type
  - Input record determination

The table of types for `Mapper` input keys and values is below. All input types come from `org.apache.hadoop.io` (see below).

Input Data Format	Input key type	Input value type
TextInputFormat	LongWritable	Text
KeyValueTextInputFormat	Text	Text
FixedLengthInputFormat	LongWritable	BytesWritable
NLineInputFormat	LongWritable	Text

Setting Input Formats. To set an input file format, include the following into your code:

```
// to set the input file location
<Class>.addInputPath(job, new Path(<path>, <file>));
 ...
// to override the default input class
job.setInputFormatClass(<Class>.class);
```

replacing `<Class>` with the name of the appropriate input data format class.

Example: KeyValueTextInputFormat. To have input file `./test/data.csv` to be treated as a `KeyValueTextInputFormat` file:

```
// to set the input file location
KeyValueTextInputFormat.addInputPath(job, new Path("./test/", "data.csv"));
 ...
// to override the default input class
job.setInputFormatClass(KeyValueTextInputFormat.class);
```

TextInputFormat

Overview. Class `org.apache.hadoop.mapreduce.lib.input.TextInputFormat` represents the basic text input. This is the **default** File Input Format class.

Records. The records of `TextInputFormat` files are **individual lines** (split by `newline` or `newline/carriage return` characters).

- **Key.** The key for each record is the **byte offset of the first byte** of the record in the file, starting with the offset of 0 for the first line in the file.
- **Value.** The value is the text of the line rendered as a `Text` instance.

Special settings. This is the default class, no special settings are needed. In fact, the statement

```
job.setInputFormatClass(TextInputFormat.class);
```

can be omitted from the code.

KeyValueTextInputFormat

Overview. Use class `org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat` when your data file has one record per line, with the first part of each record being the key you want your `map()` function to use.

Records. Class `KeyValueTextInputFormat` treats each line of input file as a single record, that, *unlike for* `TextInputFormat` records, contains **both, the key, and the value** in it.

- **Key-Value delimiter.** A delimiter separating the key of the key-value pair from the record needs to be specified. (Default separator: `tab`, i.e.,
`t`).
- **Key.** The key is the part of the line before the key-value delimiter represented as a `Text` instance.
- **Value.** The value is the part of the line between the key-value delimiter and the end of line.

Note: If the key-value delimiter is occurs more than once on a single line, *its first occurrence* is used to separate key from value on the line.

Special settings. To set the key-value delimiter, use the following code:

```
Configuration conf = new Configuration();
conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", <DELIMITER>);
Job  job = Job.getInstance(conf);
```

Here, `<DELIMITER>` is a `String` value specifying the key-value delimiter.

FixedLengthInputFormat

Overview. Class `org.apache.hadoop.mapreduce.lib.input.FixedLengthInputFormat` breaks the contents of an input file into records of equal (fixed) size.

Records. The file is broken into records of equal size.

- **Record size.** Record size (length) l is controlled by a parameter that must be set.
- **Key.** The key of each record is its byte offset in the file (starting with 0 for the first record).
- **Value.** The value is the l bytes in the file starting at the offset specified in the key represented as an instance of `BytesWritable` class.

Special Settings. To set the record length, use the `public static void setRecordLength()` method from the `FixedLengthInputFormat` class. The method takes two parameters: the `Configuration` object for the Hadoop MapReduce job, and the length of the record:

```
Configuration conf = new Configuration();
FixedLengthInputFormat.setRecordLength(conf, 5);
Job  job = Job.getInstance(conf);
```

NLineInputFormat

Overview. `org.apache.hadoop.mapreduce.lib.input.NLFileInputFormat` is a version of the default input format file that provides a simple control mechanism for the size of each file split.

Note. File split sizes in Hadoop by default are equal to 128 Mb. This makes processing big data tasks straightforward, but for tasks with small input files, may cause significant inconvenience, because the work won't be distributed among the Hadoop worker nodes (if the input file size is less than 128 Mb, only one split is created, and only one `Mapper` process runs).

`NLFileInputFormat` overcomes this issue by providing a simple default input format class in which the user can control the size of the file split in terms of the **number of lines per split**. This has the side effect of evening the number of records in each split (rather than the number of bytes).

Records. `NLineInputFormat` uses the same approach to breaking files into records as `TextInputFormat` does. Each line of the input file is a record with the byte offset of the first character of the record being the key, and the contents of the line being the record.

Special Settings. The parameter controlling the number of records per split is `mapreduce.input.lineinputformat.linespermap`. The code to set this parameter is

```
Configuration conf = new Configuration();
conf.setInt("mapreduce.input.lineinputformat.linespermap", <Number>);
Job  job = Job.getInstance(conf);
```

(alternatively, if the configuration is already set for the `job` variable above, you can use

```
job.getConfiguration.setInt("mapreduce.input.lineinputformat.linespermap", <Number>);
```

)

Note: `NLineInputFormat` controls the number of splits for the `Mapper` classes only. You can see the input partitioned into multiple splits during your hadoop run. However, this class cannot control by itself the number of reducers, so, at the end, the output will be the same as if you have simply used `TextInputFormat` as input.

Output File Formats

`org.apache.hadoop.mapreduce.lib.output` package contains a number output file formats for Hadoop.

Note. Unlike *input file formats*, most of the time, the default output class `TextOutputFormat` will suffice.

Appendix: Hadoop Data Types for Keys and Values

Hadoop *wraps* the *keys* and *values* passed to `map()` and `reduce()` methods of `Mapper` and `Reducer` classes into `Writable` containers.

The data types for *keys* and *values* are defined in

```
org.apache.hadoop.io
```

Below is a sample of `org.apache.hadoop.io` classes, and the values they represent.

Base Class	`org.apache.hadoop.io` class
`long`	`LongWritable`
`int`	`IntWritable`
`short`	`ShortWritable`
`boolean`	`BooleanWritable`
`float`	`FloatWritable`
`double`	`DoubleWritable`
`byte`	`ByteWritable`
`byte[]`	`BytesWritable`
`EnumSet<E>`	`EnumSetWritable`
`Object`	`ObjectWritable`
`String`	`Text`

To get base values from instances of the `org.apache.hadoop.io` classes we can use the `get()` methods for all classes except `BytesWritable` (for which the method is `getBytes()`, and `Text`, for which the method is `toString()`).