# MIT Java Wordnet Interface (JWI)
# User's Guide

Version 2.4.x

Mark A. Finlayson
(markaf@alum.mit.edu)

October 29, 2015

## 1 Purpose of the Software

This MIT Java Wordnet Interface (JWI) is a Java library for interfacing with the Wordnet electronic dictionary. It features API calls to retrieve index words, synsets, and morphological exceptions from the Wordnet data files. It also has calls that allow following lexical and semantic pointers, and contains classes that can perform simple morphological processing. The library has no GUI elements. It allows users to load the Wordnet dictionary completely into memory for extremely fast access, as well as export and load an in-memory dictionary to and from a stream.

JWI is freely available for all purposes, as long as proper acknowledgement is made. Details can be found in the license, which is included at the end of this document. If you wish to cite JWI in your own work, please use the following:

> Finlayson, M. A. (2014). Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation. In H. Orav, C. Fellbaum, & P. Vossen (Eds.), *Proceedings of the 7th International Global WordNet Conference* (GWC 2014) (pp. 7885). Tartu, Estonia.

## 2 Wordnet Versions Supported

JWI has been tested and works with the following versions of Wordnet.

Princeton Wordnet

- Wordnet 3.1 (no access to standoff files; this may be added in a future release)
- Wordnet 3.0 (both full version and database only distrubtion)
- Wordnet 2.0 and 2.1
- Wordnet 1.7 and 1.71
- Wordnet 1.6 (No access to cousin files or index.gloss file)

Stanford Augmented Wordnet

- Wordnet 2.1 + 10k
- Wordnet 2.1 + 20k
- Wordnet 2.1 + 30k
- Wordnet 2.1 + 40k

JWI is not compatible with Wordnet version 1.5. The Stanford Augmented Wordnet versions 400k(cropped) and 400k(all) have errors in their byte-offsets, alphabetical ordering, and adjective satellite markings, and so JWI does not work reliably with them. No version of Wordnet is included with the JWI distribution, and must be downloaded separately.

# 3 Getting Started

The main interface for accessing dictionary data is the `edu.mit.jwi.IDictionary` interface. The standard implementation of this interface is the `edu.mit.jwi.Dictionary` class. In the simplest case, where you are using Wordnet with the data files on the same filesystem as your Java program, you can instantiate the `Dictionary` class with a single argument, a Java URL object that points to the directory where the Wordnet dictionary data files are located.

An example of this can be found in Listing 1, in the form of a Java method `testDictionary()`. In that method, the first block of three lines (4-6) deals with constructing a URL object that points to the Wordnet data files. In this particular example, the base Wordnet directory is assumed to be stored in a system environment variable called `WNHOME`. Note that the `WNHOME` variable points to the root of the Wordnet installation directory and the dictionary data directory "dict" must be appended to this path. This may be different on your system depending on where your Wordnet files are located. The second block of code, two lines long (9-10), constructs an instance of the default Dictionary object, and opens it by calling the `open()` method. The final block of six lines (13-18) demonstrates searching the dictionary for the first sense of the noun "dog". Listing 2 shows the console output of the method.

```
1  public void testDictionary() throws IOException {
2
3      // construct the URL to the Wordnet dictionary directory
4      String wnhome = System.getenv("WNHOME");
5      String path = wnhome + File.separator + "dict";
6      URL url = new URL("file", null, path);
7
8      // construct the dictionary object and open it
9      IDictionary dict = new Dictionary(url);
10     dict.open();
11
12     // look up first sense of the word "dog"
13     IIndexWord idxWord = dict.getIndexWord("dog", POS.NOUN);
14     IWordID wordID = idxWord.getWordIDs().get(0);
15     IWord word = dict.getWord(wordID);
16     System.out.println("Id = " + wordID);
17     System.out.println("Lemma = " + word.getLemma());
18     System.out.println("Gloss = " + word.getSynset().getGloss());
19 }
```

Listing 1: Basic use of JWI

```
1  Id = WID-2084071-n-?-dog
2  Lemma = dog
3  Gloss = a member of the genus Canis (probably descended from the common
       wolf) that has been domesticated by man since prehistoric times; occurs
        in many breeds; "the dog barked all night"
```

Listing 2: Output of the `testDictionary()` method in Listing 1 (for Wordnet 3.0)

# 4 Loading Wordnet into Memory

JWI 2.2.0 introduced the ability to load a dictionary fully into memory. This begets a substantial performance improvement. To take advantage, use the new `edu.mit.jwi.RAMDictionary` class, as shown in 3.

```
1  public void testRAMDictionary(File wnDir) throws Exception {
2
3      // construct the dictionary object and open it
4      IRAMDictionary dict = new RAMDictionary(wnDir, ILoadPolicy.NO_LOAD);
5      dict.open();
6
7      // do something slowly
8      trek(dict);
9
10     // now load into memory
11     System.out.print("\nLoading Wordnet into memory...");
12     long t = System.currentTimeMillis();
13     dict.load(true);
14     System.out.printf("done (%1d msec)\n", System.currentTimeMillis()-t);
15
16     // do the same thing again, only faster
17     trek(dict);
18 }
19
20 public void trek(IDictionary dict){
21     int tickNext = 0;
22     int tickSize = 20000;
23     int seen = 0;
24     System.out.print("Treking across Wordnet");
25     long t = System.currentTimeMillis();
26     for(POS pos : POS.values())
27         for(Iterator<IIndexWord> i = dict.getIndexWordIterator(pos); i.
               hasNext(); )
28             for(IWordID wid : i.next().getWordIDs()){
29                 seen += dict.getWord(wid).getSynset().getWords().size();
30                 if(seen > tickNext){
31                     System.out.print('.');
32                     tickNext = seen + tickSize;
33                 }
34             }
35     System.out.printf("done (%1d msec)\n", System.currentTimeMillis()-t);
36     System.out.println("In my trek I saw " + seen + " words");
37 }
```

Listing 3: Using JWI completely in-memory

```
1  Treking across Wordnet.........................done (3467 msec)
2  In my trek I saw 522858 words
3
4  Loading Wordnet into memory...done (5728 msec)
5  Treking across Wordnet.........................done (205 msec)
6  In my trek I saw 522858 words
```

Listing 4: Output of the `testRAMDictionary()` method in Listing 3 (for Wordnet 3.0)

# 5 Loading Wordnet from a Stream

JWI 2.4.0 introduced the ability to export an in-memory dictionary image to a stream, and load it back from a stream. This means that you can save a whole in-memory dictionary instance as a single file (pointed to by a file handle or URL), and then restore it This is especially useful for sandboxed environments where access to the local file system is restricted. Code demonstrating how to take advantage of this feature is shown in Listing 5.

```
1  public void demonstrateExportAndLoad(File wnDir) throws IOException {
2
3      // load RAM dictionary data into memory
4      System.out.print("Loading dictionary data...");
5      long t = System.currentTimeMillis();
6      IRAMDictionary ramDict = new RAMDictionary(wnDir);
7      ramDict.setLoadPolicy(ILoadPolicy.IMMEDIATE_LOAD);
8      ramDict.open();
9      t = System.currentTimeMillis()-t;
10     System.out.printf("done (%1d sec)\n", t/1000);
11
12     // we will store our exported Wordnet data here
13     File exFile = File.createTempFile("JWI_Export_", ".wn");
14     exFile.deleteOnExit();
15
16     // export in-memory data
17     System.out.print("Exporting dictionary data...");
18     t = System.currentTimeMillis();
19     ramDict.export(new FileOutputStream(exFile));
20     ramDict.close();
21     t = System.currentTimeMillis()-t;
22     System.out.printf("done (%1d sec)\n", t/1000);
23
24     // take a look at the file
25     System.out.printf("Export is %1d MB\n", exFile.length()/1048576);
26
27     // load RAM dictionary data
28     System.out.print("Loading from exported data...");
29     t = System.currentTimeMillis();
30     ramDict = new RAMDictionary(exFile);
31     ramDict.open();
32     t = System.currentTimeMillis()-t;
33     System.out.printf("done (%1d sec)\n", t/1000);
34
35     trek(ramDict);
36 }
```

Listing 5: Exporting an in-memory JWI Dictionary to a stream

```
1  Loading dictionary data...done (6 sec)
2  Exporting dictionary data...done (27 sec)
3  Export is 26 MB
4  Loading from exported data...done (8 sec)
5  Treking across Wordnet.........................done (184 msec)
6  In my trek I saw 522858 words
```

Listing 6: Output of the `demonstrateExportAndLoad()` method in Listing 5 (for Wordnet 3.0)

# 6 Frequently Asked Questions

## 6.1 Which implementation of `IDictionary` should I use? Do I need to use RAMDictionary?

Most will not need the performance edge that comes from loading all of Wordnet into memory. JWI is already pretty fast. The standard `edu.mit.jwi.Dictionary` class, pointing to a local directory containing the Wordnet files, will do for most. If you're worried about wasted computations, increase the cache size. For most, the inconvenience inherent in waiting five or ten seconds for Wordnet to do a blocking load, or the slowdown of the whole application during a non-blocking load, or having to remember to set the heap size correctly, will be more of a hassle than just using the on-disk implementation.

## 6.2 I'm using `RAMDictionary` and I'm getting an `OutOfMemoryError`. What do I do?

Wordnet is large, and usually won't fit into the standard heap on most 32-bit JVMs. You need to increase your heap size. On the Sun JVM, this involves the command line flag `-Xmx` along with a larger-than-normal heap size, say, 500 MB or 1 GB.

## 6.3 Why doesn't JWI return a word I know is in Wordnet?

The `IDictionary` interface contract states that you must pass in the root form of the word you are looking for. If you pass in a plural form of a noun (such as "dogs" instead of "dog"), or a non-base form of verb ("running" instead of "run"), it won't give you back the right objects. Use the `edu.mit.jwi.morph.WordnetStemmer` class to obtain root forms for words, before passing them to the dictionary for lookup.

## 6.4 How do I retrieve the synonyms of a word?

Each meaning, or *synset*, in Wordnet has multiple lexical forms or collocations associated with it. The meaning of the word dog implied in sentence "My dog Fido barks," has three: *dog*, *domestic_dog*, and *Canis_familiaris*.To obtain this list, we must first get a handle to the `ISynset` object for that meaning. Then, we can get the graphical forms by iterating over all the `IWord` objects associated with that synset. Java code that does this is shown in Listing 7.

```
1  public void getSynonyms(IDictionary dict){
2
3      // look up first sense of the word "dog"
4      IIndexWord idxWord = dict.getIndexWord("dog", POS.NOUN);
5      IWordID wordID = idxWord.getWordIDs().get(0); // 1st meaning
6      IWord word = dict.getWord(wordID);
7      ISynset synset = word.getSynset();
8
9      // iterate over words associated with the synset
10     for(IWord w : synset.getWords())
11         System.out.println(w.getLemma());
12 }
```

Listing 7: Retrieving the synonyms of the first meaning of the word "dog"

```
1 dog
2 domestic_dog
3 Canis_familiaris
```

Listing 8: Output of the `getSynonyms()` method in Listing 7 (for Wordnet 3.0)

## 6.5 How do I retrieve the hypernyms of a word?

Each synset is connected to other synsets by semantic pointers. One of the most prevalent of these is the *Hypernym* pointer, which points from a meaning to other meanings that are "more general" in some fashion; these are called hypernyms. The sense of dog used in the previous example has a two hypernyms: the subset of mammals that are closely related to dogs, such as wolves or jackals (this Synset has synonyms *canine* and *canid*), and the set of domesticated animals (with synonyms *domestic_animal* and *domesticated_animal*). To obtain these synsets, we must first get a handle to the `ISynset` object for the meaning for dog. Then, we can retrieve the hypernym synsets via the `getRelatedSynsets(IPointerType)` method. Java code that does this is shown in Listing 9.

```
1  public void getHypernyms(IDictionary dict){
2
3      // get the synset
4      IIndexWord idxWord = dict.getIndexWord("dog", POS.NOUN);
5      IWordID wordID = idxWord.getWordIDs().get(0); // 1st meaning
6      IWord word = dict.getWord(wordID);
7      ISynset synset = word.getSynset();
8
9      // get the hypernyms
10     List<ISynsetID> hypernyms =
11         synset.getRelatedSynsets(Pointer.HYPERNYM);
12
13     // print out each hypernyms id and synonyms
14     List<IWord> words;
15     for(ISynsetID sid : hypernyms){
16         words = dict.getSynset(sid).getWords();
17         System.out.print(sid + " {");
18         for(Iterator<IWord> i = words.iterator(); i.hasNext();){
19             System.out.print(i.next().getLemma());
20             if(i.hasNext())
21                 System.out.print(", ");
22         }
23         System.out.println("}");
24     }
25 }
```

Listing 9: Retrieving the hypernums of the word "dog"

```
1 SID-2083346-n {canine, canid}
2 SID-1317541-n {domestic_animal, domesticated_animal}
```

Listing 10: Output of the `getHypernyms()` method in Listing 9 (for Wordnet 3.0)

## 6.6 Why doesn't JWI have a pointer from item X to item Y that I know is in Wordnet?

For this, the problem is usually that you are trying to retrieve `ISynset` objects via *lexical pointers*, or `IWord` objects via *semantic* pointers. The key difference between the two is that semantic pointers are between synsets (i.e., meanings, such as the domesticated animal meaning captured in {*dog*, *domestic_dog*, *canis_familiaris*}), and lexical pointers are between word forms (e.g., the dog word form of the above meaning only). To retrieve semantic pointers, one uses the `ISynset.getRelatedSynsets(IPointer)` method, and to retrieve lexical pointers, one should use the `IWord.getRelatedWords(IPointer)` method. If you pass a lexical pointer (say DERIVED) to the `getRelatedSynsets` method, you won't get anything back.

These two types of pointers are not well distinguished in the Wordnet documentation. Figure 1 diagrams the relationship. As can be seen, in JWI lexical pointers are found only between `IWord` objects, and semantic pointers are found only between `ISynset` objects. There are no pointers that connect a word to a synset. Thus, you can only find hypernyms of synsets, and you can only find derived forms of words.

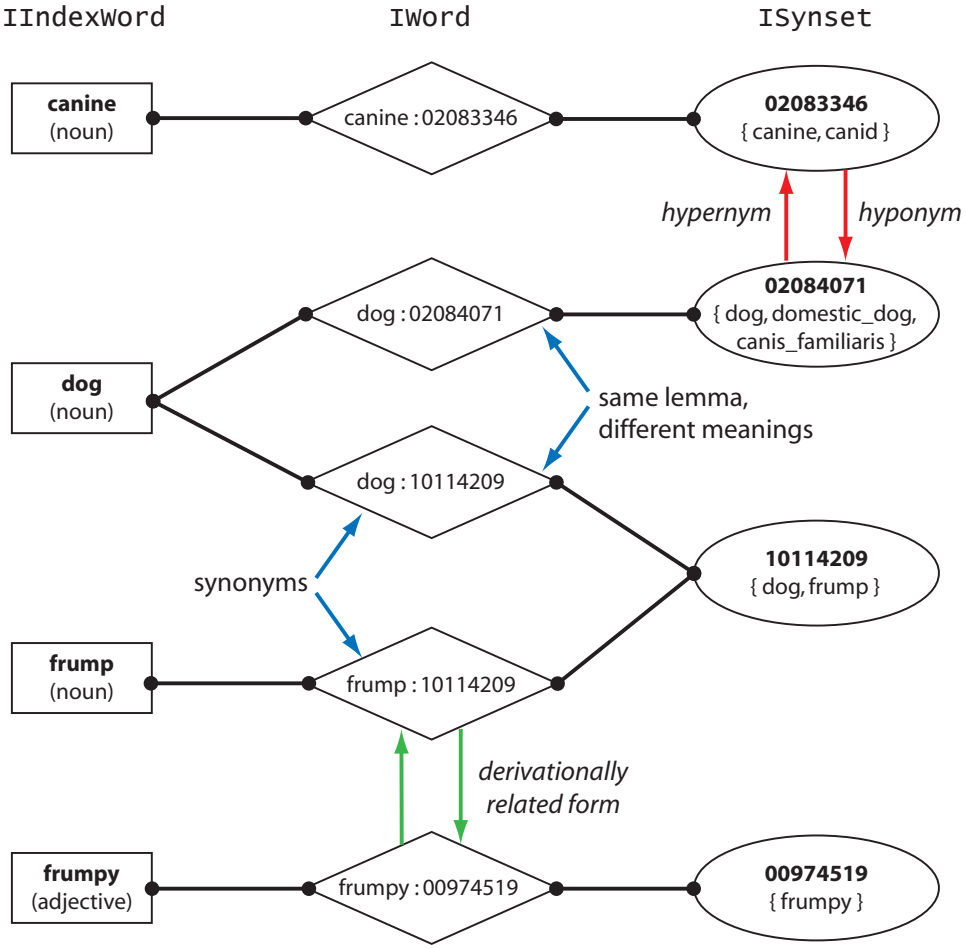


Figure 1: The structure of Wordnet: index words, words, synsets, synonyms, lexical pointers (green arrows) and semantic pointers (red arrows) (for Wordnet 3.0)

## 6.7 How do I know which pointers are lexical and which are semantic?

Unfortunately, the Wordnet documentation doesn't give much hint about which pointers are used as lexical pointers, which are used as semantic pointers, and which as both. Fortunately for those seeking an answer to this question, I wrote some code that sifts through Wordnet and counts the instances of each. Those numbers for Wordnet 3.0 are found in Table 1. The counts for the Wordnet 2.0 and 2.1 are similar, and the separation of lexical and semantic pointers between lexical and semantic is exactly the same, with the exception that Wordnet 2.0 uses REGION and REGION_MEMBER types only as semantic pointers.

| **Pointer** | **Lexical** | **Semantic** | **Total** | **Both?** |
|---|---|---|---|---|
| ALSO_SEE | 580 | 2692 | 3272 | Yes |
| ANTONYM | 7979 | - | 7979 | - |
| ATTRIBUTE | - | 1278 | 1278 | - |
| CAUSE | - | 220 | 220 | - |
| DERIVED | 74714 | - | 74714 | - |
| DERIVED_ADJ | 3222 | - | 3222 | - |
| ENTAILMENT | - | 408 | 408 | - |
| HOLONYM_MEM | - | 12293 | 12293 | - |
| HOLONYM_PRT | - | 9097 | 9097 | - |
| HOLONYM_SUB | - | 797 | 797 | - |
| HYPERNYM | - | 89089 | 89089 | - |
| HYPERNYM_IN | - | 8577 | 8577 | - |
| HYPONYM | - | 89089 | 89089 | - |
| HYPONYM_INS | - | 8577 | 8577 | - |
| MERONYM_MEM | - | 12293 | 12293 | - |
| MERONYM_PRT | - | 9097 | 9097 | - |
| MERONYM_SUB | - | 797 | 797 | - |
| PARTICIPLE | 73 | - | 73 | - |
| PERTAINYM | 4799 | - | 4799 | - |
| REGION | 15 | 1345 | 1360 | Yes |
| REGION_MEM | 15 | 1345 | 1360 | Yes |
| SIMILAR_TO | - | 21386 | 21386 | - |
| TOPIC | 11 | 6643 | 6654 | Yes |
| TOPIC_MEM | 11 | 6643 | 6654 | Yes |
| USAGE | 409 | 967 | 1376 | Yes |
| USAGE_MEM | 409 | 967 | 1376 | Yes |
| VERB_GROUP | 2 | 1748 | 1750 | Yes |
| **Total** | 92239 | 285348 | 377587 | |

Table 1: Counts of Lexical and Semantic Pointers (for Wordnet 3.0)

## 6.8 Can JWI load itself completely into memory?

Yes, JWI has the ability, as of version 2.2.0, to load itself completely into memory. For the cost of several hundred megabytes of memory, you get a significant speedup. See the `edu.mit.jwi.RAMDictionary` class.

## 6.9 Who do the following messages that are following printed to System.err mean?

1229375196772 - Error on direct access in verb data file: check CR/LF endings

The Wordnet .data files (i.e., the four files that either start or end with data, e.g., noun.data, verb.data, etc.) are indexed by byte offset. The line endings in the files are indicated in the standard UNIX way, by a single linefeed (LF) character, often written as "\n". In contrast, the standard linefeed on Windows systems is a carriage-return character followed by a linefeed (CR/LF), often written "\r\n". Some archive extraction programs automatically convert LF linefeeds to CR/LF linefeeds, especially if you're on a Windows system. If this was done, the Wordnet data files will have an extra character at the end of each line, and the byte offsets will no longer index to the right places. As of JWI 2.1.5, there is a check for this, where the above error prints. The error doesn't prevent JWI from running, but it does cause a reduction in performance. I recommend you re-extract your data files from the Wordnet archive, making sure you have the option to automatically convert line endings unchecked. For example, in WinZip 9, there is option under Configuration – Miscellaneous called "TAR smart CR/LF conversion". Make sure it is unchecked when extracting the wordnet files.

## 6.10 Why are all my `ISynset` objects `null`?

See the answer to Question 6.9. In versions of JWI before 2.1.5, there was no check for corrupted line endings in the data files, and the symptom is no errors except null synset objects.

## 6.11 Why does the `Dictionary.getExceptionIterator` method throw a `NullPointerException`?

This happens when using JWI 2.2.3 or earlier on the "database only" distribution of Wordnet that was released for version 3.0. Princeton did not include the exception files in that distrubtion, which causes the `getExceptionIterator` method to throw an exception. The solution is to either upgrade JWI 2.2.4 or later; copy the exception files from another download of Wordnet; or create empty exception files in your wordnet data directory with the names `adj.exc`, `adv.exc`, `noun.exc`, and `verb.exc`.

## 6.12 How do I license JWI for commercial use?

As of version 2.2.2, JWI is free to use for all purposes, as long as proper copyright acknowledgment is made. No special license is needed for commercial use.

## 6.13 How do I report a bug / request a feature / heap praise upon you for making such cool software?

If you find a bug, have a feature request, or wish to send your praises, please contact me via my permanent email forwarding address markaf@alum.mit.edu.

# 7 License

The following is the full text of the MIT Java Wordnet Interface license.

3.a. Attribution.

3.a.1. If You Share the Licensed Material (including in modified form), You must:

3.a.1.A. retain the following if it is supplied by the Licensor with the Licensed Material:

3.a.1.A.i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

3.a.1.A.ii. a copyright notice;

3.a.1.A.iii. a notice that refers to this Public License;

3.a.1.A.iv. a notice that refers to the disclaimer of warranties;

3.a.1.A.v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

3.a.1.B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

3.a.1.C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

3.a.2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3.a.3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

3.a.4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

4.a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

4.b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

4.c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 Disclaimer of Warranties and Limitation of Liability.

5.a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

5.b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

5.c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 Term and Termination.

6.a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

6.b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

6.b.1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

6.b.2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

6.c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

6.d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 Other Terms and Conditions.

7.a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

7.b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 Interpretation.

8.a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

8.b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

8.c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.