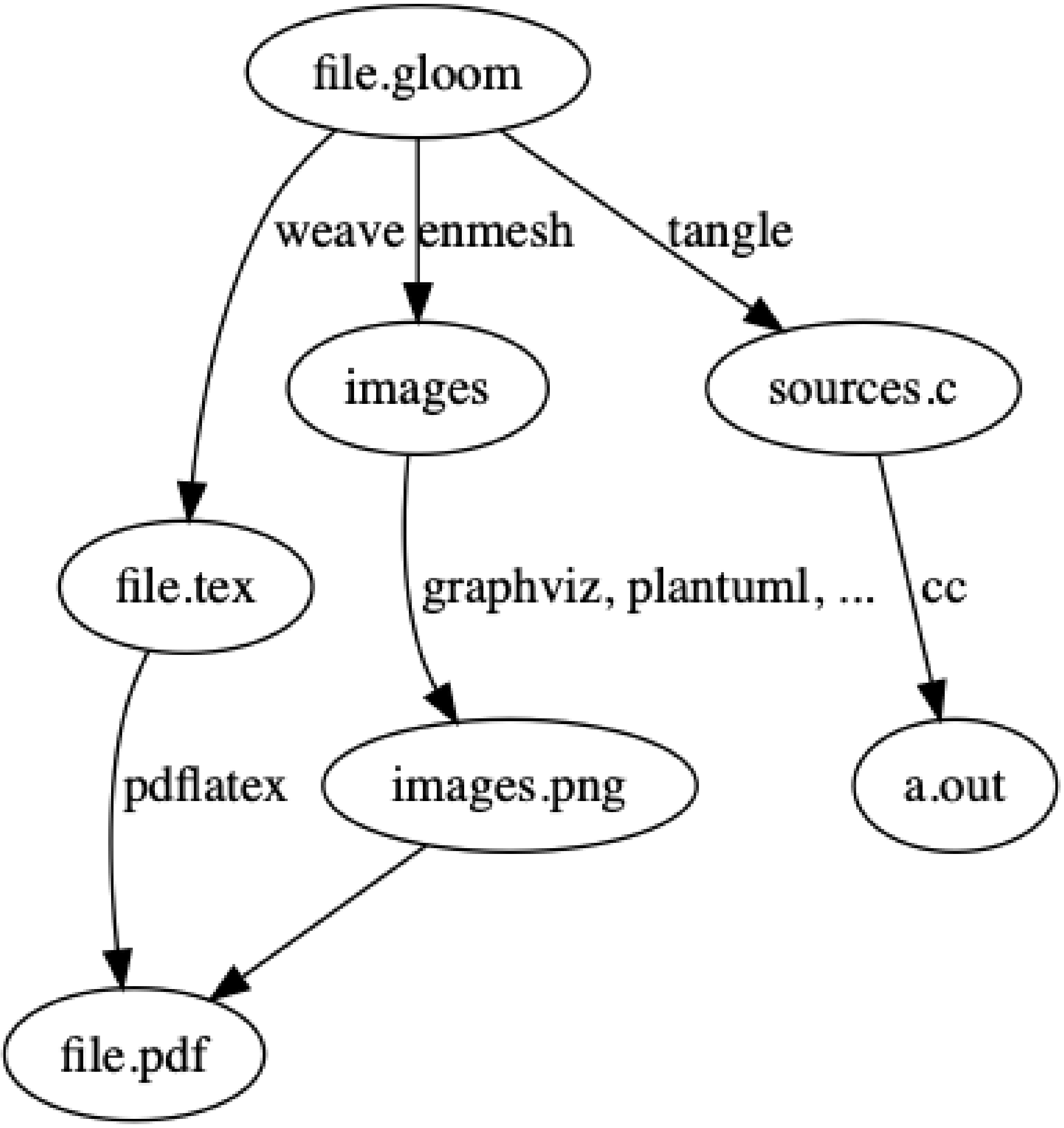

Figure 1: gloom.dot

1

This is `gloom`, the Graphical LOOM. It's a tool for Literate Programming, that lets you describe your programs in text, graphics, and computer source language, all in one place.

`gloom` itself is distributed as a `gloom` file, and you're reading a document now that was generated from that file. A `gloom` document *tells* you how the program works, *shows* you how it works, and indeed *is* the very program being described. Each of these capabilities is supplied by a different transformation on the source document: weaving, tangling, and enmeshing. Figure 1 shows how these come together.

2

You invoke `gloom` with a single argument, the name of the source file. The main function checks that this requirement has been satisfied.

3 < check usage 3 >

```
    if (argc != 2) {
        fprintf(stderr, "Usage: gloom <filename>\n");
        exit(EX_USAGE);
    }
```

This code is used in section 5.

4

`gloom` is actually built of four modules. The weaver, enmesher and tangler all operate on a list of *chunks*, sections of the `gloom` file delimited by particular boundary strings. The remaining module, then, comes before the others, splitting the source file into chunks.

The main function invokes all four of these modules and writes their output to disk.

5 < main.m 5 >

```
«system headers 7»

«module headers 9»


int main(int argc, const char *argv[]) {
  @autoreleasepool {
«check usage 3»

«load gloom file 98»

«find chunks 11»

«weave LaTeX document 53»

«write LaTeX document 55»

«tangle code files 93»

«write code files 94»

«enmesh figure files 95»

«write figure files 96»
  }
  return 0;
}
```

6

Being an Objective-C command line application, `gloom` uses the Foundation library. Additionally it uses standard return codes to signal failure reasons back to the shell.

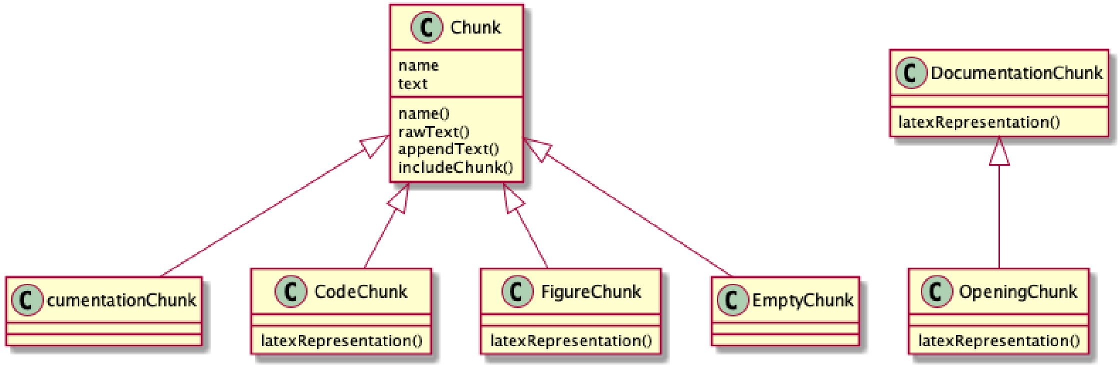

Figure 2: chunks.puml

7 < system headers 7 >

```
#import <Foundation/Foundation.h>
#import <sysexits.h>
```

This code is used in section 5.

8

The main function also needs the definition of each of the module interfaces. It doesn't need to know how chunks work, because it just passes them between the modules. The output of the tangling and enmeshing phases are lists of `CodeFile` objects, and it does need to know how to tell them to write themselves out.

9 < module headers 9 >

```
#import "ChunkFinder.h"
#import "CodeFile.h"
#import "CodeTangler.h"
#import "FigureEnmesher.h"
#import "LatexWeaver.h"
```

This code is used in section 5.

10

We will look at each of the important modules in turn, before coming back to the intricacies of the main function. The first important task it does, once it has worked out that it has the information to do it, is get the list of chunks from the `ChunkFiler`.

11 < find chunks 11 >

```
    id chunks = [[ChunkFinder new] chunksInDocument:file named:basename];
```

This code is used in section 5.

12

Chunks are instances of classes descended from the `Chunk` class. While it knows how to be a chunk, it does not know how to represent itself as LaTeX, or whether it contains code, and so on. Those questions are answered by specific subclasses, shown in Figure 2 (some attributes and methods are elided from the diagram).

13

Of these different types of chunk, only the opening, code, figure, and documentation chunks can be found in a `gloom` file. By definition, the opening chunk is found at the start of a file, so there is no delimiter for it. The delimiters are encapsulated in `ChunkBoundary` classes, which know whether a line of text matches a boundary and what kind of chunk to create if it does.

14 < ChunkBoundary.h 14 >

```
#import <Foundation/Foundation.h>

@interface ChunkBoundary : NSObject

− changedChunkAtLine:text;
− boundaryPattern;
− newChunkFromResult:matchResult inLine:text;

@end
```

15 < ChunkBoundary.m 15 >

```
#import "ChunkBoundary.h"

@implementation ChunkBoundary

− boundaryPattern
{
    return nil;
}

− newChunkFromResult:matchResult inLine:text
{
    return nil;
}
«changed chunk at line 17»


@end
```

16

The questions a subclass of `ChunkBoundary` must answer are: given a line of input to `gloom`, do we need to start a new chunk, and if so what should it be?

17 < changed chunk at line 17 >

```
− changedChunkAtLine:text
{
    id codeChunkRegex = [NSRegularExpression regularExpressionWithPattern:[self boundaryPattern]
                                                                  options:0
                                                                    error:NULL];
    id result = [codeChunkRegex firstMatchInString:text
                                           options:0
                                             range:NSMakeRange(0, [text length])];
    return [self newChunkFromResult:result inLine:text];
}
```

This code is used in section 15.

18

Documentation boundaries are delimited by a line containing the sole character `@`, and as such cannot have a name.

19 < DocumentationBoundary.h 19 >

```
#import "ChunkBoundary.h"

@interface DocumentationBoundary : ChunkBoundary

@end
```

20 < DocumentationBoundary.m 20 >

```
#import "DocumentationBoundary.h"
#import "Chunk.h"

@implementation DocumentationBoundary

− boundaryPattern
{
    return @"^@$";
}

− newChunkFromResult:matchResult inLine:text
{
    return matchResult ? [Chunk documentationChunkWithName:@""] : nil;
}

@end
```

21

Code chunks are preceded by a line containing `«chunk name»=`.

22 < CodeBoundary.h 22 >

```
#import "ChunkBoundary.h"

@interface CodeBoundary : ChunkBoundary

@end
```

23 < CodeBoundary.m 23 >

```
#import "CodeBoundary.h"
#import "Chunk.h"

@implementation CodeBoundary
```

```
- boundaryPattern
{
    return @"^<<"
      @"(.*)>>=$";
}

- newChunkFromResult:matchResult inLine:text
{
    if (matchResult) {
        id name = [text substringWithRange:[matchResult rangeAtIndex:1]];
        return [Chunk codeChunkWithName:name];
    }
    return nil;
}

@end
```

24

And finally, figure boundaries are delineated by `{{figure name}}=`.

25 < FigureBoundary.h 25 >

```
#import "ChunkBoundary.h"

@interface FigureBoundary : ChunkBoundary

@end
```

26 < FigureBoundary.m 26 >

```
#import "FigureBoundary.h"
#import "Chunk.h"

@implementation FigureBoundary

- boundaryPattern
{
    return @"^\\{\\{(.*)\\}\\}=$";
}

- newChunkFromResult:matchResult inLine:text
{
    if (matchResult) {
        id name = [text substringWithRange:[matchResult rangeAtIndex:1]];
        return [Chunk figureChunkWithName:name];
    }
    return nil;
}

@end
```

27

The `ChunkBoundary` class hierarchy therefore represents the "dictionary" of the `gloom` language: if you want a new delimeter for a new kind of chunk, you add it here.

The ChunkFinder uses these boundaries to split the gloom source into chunks.

28 < ChunkFinder.h 28 >

```
#import <Foundation/Foundation.h>

@interface ChunkFinder : NSObject

− chunkBoundaries;
− chunksInDocument:aDocument named:aName;

@end
```

29 < ChunkFinder.m 29 >

```
#import "ChunkFinder.h"
#import "Chunk.h"
#import "CodeBoundary.h"
#import "DocumentationBoundary.h"
#import "FigureBoundary.h"

@implementation ChunkFinder
«chunk boundaries 31»

«chunks in document 33»


@end
```

30

The ChunkFinder's list of chunk boundaries is where it "knows" about each of the different boundaries.

31 < chunk boundaries 31 >

```
− chunkBoundaries
{
    return @[[CodeBoundary new], [DocumentationBoundary new], [FigureBoundary new]];
}
```

This code is used in section 29.

32

It is the chunksInDocument:named: method that actually builds the model: a list of chunks. It starts by creating the "opening chunk", then reads each line in the file. If it matches a chunk boundary, then a new chunk is created. If not, then the line is added to the current chunk.

33 < chunks in document 33 >

```
− chunksInDocument:aDocument named:aName
{
    id boundaries = [self chunkBoundaries];
«create chunk indices 35»
«create opening chunk 37»
    [aDocument enumerateLinesUsingBlock:^(NSString * _Nonnull line, BOOL * _Nonnull stop) {
        __block id changeChunk = nil;
        [boundaries enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
            changeChunk = [obj changedChunkAtLine:line];
            *stop = (changeChunk != nil);
        }];
        if (changeChunk) {
            [chunks addObject:changeChunk];
            chunksByName[[changeChunk name]] = changeChunk;
            currentChunk = changeChunk;
        } else {
            [currentChunk appendText:[NSString stringWithFormat:@"%@\n", line]];
        }
    }];
    [chunks makeObjectsPerformSelector:@selector(findInclusions:) withObject:chunksByName];
    return [chunks copy];
}
```

This code is used in section 29.

34

Two indices are maintained: the list of chunks seen so far, and a dictionary of chunks by name that is used for resolving inclusions (a code chunk can include another code chunk by enclosing its name in guillemets, «like this»).

35 < create chunk indices 35 >

```
    id chunks = [NSMutableArray new];
    id chunksByName = [NSMutableDictionary new];
```

This code is used in section 33.

36

The initial chunk is a special documentation chunk called the *opening chunk*. This comes in helpful when weaving. A cursor is created, and set to the opening chunk.

37 < create opening chunk 37 >

```
    id docChunk = [Chunk openingChunkWithName:aName];
    chunksByName[aName] = docChunk;
    [chunks addObject:docChunk];
    __block id currentChunk = docChunk;
```

This code is used in section 33.

38

The chunk classes are listed below. The sections relevant to constructing the model are shown in-place in the source files, anything relevant to weaving, tangling or enmeshing is left out for now.

39 < Chunk.h 39 >

```
#import <Foundation/Foundation.h>
```

```
@interface Chunk : NSObject

− name;
− rawText;
− includedChunkNames;
− includingChunksEnumerator;
− (BOOL)isDocumentation;
− (BOOL)isCode;
− (BOOL)isFigure;
− (BOOL)wasNeverIncluded;

− (void)findInclusions:chunks;
− (void)appendText:someText;
− (void)replaceTextInRange:(NSRange)range withString:someText;
− (void)includeChunk:otherChunk;
− (void)wasIncludedBy:otherChunk;
− (void)internaliseInclusionsWithChunks:chunks;
− (void)writeOutToFile:file;

− initWithName:aName;

+ codeChunkWithName:aName;
+ documentationChunkWithName:aName;
+ figureChunkWithName:aName;
+ openingChunkWithName:aName;
+ emptyChunkWithName:aName;

«chunk weaving interface 60»

@end
```

40 < Chunk.m 40 >

```
#import "Chunk.h"
#import "CodeChunk.h"
#import "DocumentationChunk.h"
#import "EmptyChunk.h"
#import "FigureChunk.h"
#import "OpeningChunk.h"

@implementation Chunk
{
    id name;
    id text;
    id includedChunkNames;
    id includingChunks;
}

− initWithName:aName
{
    self = [super init];
    if (self) {
        name = [aName copy];
        text = [NSMutableString new];
        includedChunkNames = [NSMutableSet new];
        includingChunks = [NSMutableDictionary new];
    }
    return self;
}

+ codeChunkWithName:aName
{
    return [[CodeChunk alloc] initWithName:aName];
}

+ documentationChunkWithName:aName
{
    return [[DocumentationChunk alloc] initWithName:aName];
}

+ figureChunkWithName:aName
{
    return [[FigureChunk alloc] initWithName:aName];
}

+ openingChunkWithName:aName
{
    return [[OpeningChunk alloc] initWithName:aName];
}

+ emptyChunkWithName:aName
{
    return [[EmptyChunk alloc] initWithName:aName];
}

- name
{
    return name;
}

- rawText
{
    return [text copy];
}

- includedChunkNames
{
    return [includedChunkNames copy];
}

- includingChunksEnumerator
{
    return [includingChunks objectEnumerator];
}

- (BOOL)wasNeverIncluded
{
    return ([includingChunks count] == 0);
}

- (void)findInclusions:(id)chunks
{
    return;
```

```
}

– (void)includeChunk:otherChunk
{
    [includedChunkNames addObject:[otherChunk name]];
    [otherChunk wasIncludedBy:self];
}

– (void)wasIncludedBy:otherChunk
{
    [includingChunks setObject:otherChunk forKey:[otherChunk name]];
}

– (void)internaliseInclusionsWithChunks:chunks
{
    return;
}

– (void)appendText:someText
{
    [text appendString:someText];
}

– (void)replaceTextInRange:(NSRange)range withString:someText
{
    [text replaceCharactersInRange:range withString:someText];
}

– (void)writeOutToFile:aFile
{
    [aFile appendText:[self rawText]];
}

– (BOOL)isDocumentation
{
    return NO;
}

– (BOOL)isCode
{
    return NO;
}

– (BOOL)isFigure
{
    return NO;
}

@end
```

41 < CodeChunk.h 41 >

```
#import "Chunk.h"

@interface CodeChunk : Chunk
```

```
@end
```

42

43 < CodeChunk.m 43 >

```
#import "CodeChunk.h"

@implementation CodeChunk

− (BOOL)isCode
{
    return YES;
}
«code chunk weaving 66»

«code chunk tangling 91»


@end
```

44 < DocumentationChunk.h 44 >

```
#import <Foundation/Foundation.h>
#import "Chunk.h"

@interface DocumentationChunk : Chunk

@end
```

45 < DocumentationChunk.m 45 >

```
#import "DocumentationChunk.h"

@implementation DocumentationChunk

− (BOOL)isDocumentation
{
    return YES;
}
«documentation chunk weaving 63»


@end
```

46 < EmptyChunk.h 46 >

```
#import "Chunk.h"

@interface EmptyChunk : Chunk

@end
```

47 < EmptyChunk.m 47 >

```
#import "EmptyChunk.h"

@implementation EmptyChunk
«empty chunk weaving 72»

@end
```

48 < FigureChunk.h 48 >

```
#import "Chunk.h"

@interface FigureChunk : Chunk

@end
```

49 < FigureChunk.m 49 >

```
#import "FigureChunk.h"

@implementation FigureChunk

- (BOOL)isFigure
{
    return YES;
}
«figure chunk weaving 64»

@end
```

50 < OpeningChunk.h 50 >

```
#import "DocumentationChunk.h"

@interface OpeningChunk : DocumentationChunk

@end
```

51 < OpeningChunk.m 51 >

```
#import "OpeningChunk.h"

@implementation OpeningChunk
«opening chunk weaving 62»

@end
```

52

Once the model has been loaded, we have a list of chunks that represent documentation, code, or figures. The first operation to be undertaken is *weaving*, which turns the chunks into a LaTeX document. The class that does this is called `LatexWeaver`, on the basis that you might want to supply other weavers that understand HTML, Markdown or other formats.

53 < weave LaTeX document 53 >

```
    id latexDocument = [[LatexWeaver new] weave:chunks];
```

This code is used in section 5.

54

Its output is a string that can be written to become the LATEX source file.

55 < write LaTeX document 55 >

```
    BOOL written = [latexDocument writeToFile:[basename stringByAppendingPathExtension:@"tex"]
                                   atomically:NO
                                     encoding:NSUTF8StringEncoding
                                        error:&error];
    if (!written) {
        fprintf(stderr, "%s\n", [[error localizedFailureReason] UTF8String]);
        exit(EX_IOERR);
    }
    fprintf(stdout, "%s woven\n", [filename UTF8String]);
```

This code is used in section 5.

56 < LatexWeaver.h 56 >

```
#import <Foundation/Foundation.h>

@interface LatexWeaver : NSObject

− weave:chunks;

@end
```

57

The `-weave:` method simply asks every chunk in the list for its `-latexRepresentation`. It concatenates these with newlines, and surrounds them with some preamble and postamble to make a valid LATEX document.

58 < LatexWeaver.m 58 >

```
#import "LatexWeaver.h"

@implementation LatexWeaver

− preamble
{
    return @"\\documentclass{article}\n"
    @"\\usepackage{amsmath}\n"
    @"\\usepackage[cm]{fullpage}\n"
    @"\\usepackage{xcolor}\n"
    @"\\usepackage{listings}\n"
    @"\\lstset{\n"
    @" columns=fullflexible,\n"
    @" breaklines=true,\n"
    @" postbreak=\\mbox{\\textcolor{red}{$\\hookrightarrow$}\\space},\n"
    @"}\n"
    @"\\usepackage{graphicx}\n"
    @"\\usepackage[T1]{fontenc}\n"
    @"\\begin{document}\n"
    @"\\section{}\n";
}

− postamble
{
    return @"\\end{document}\n";
}

− weave:chunks
{
    id paragraphs = [[@[[self preamble]] arrayByAddingObjectsFromArray:
                      [chunks valueForKey:@"latexRepresentation"]]
                      arrayByAddingObject:[self postamble]];
    return [paragraphs componentsJoinedByString:@"\n"];
}

@end
```

59

Every specific type of chunk except the empty chunk can appear in the list, so these all know how to represent themselves as LaTeX.[1]

60 < chunk weaving interface 60 >

```
− latexRepresentation;
```

This code is used in section 39.

61

The figure chunk knows to represent itself as a figure that includes a graphics file.

62 < opening chunk weaving 62 >

```
− latexRepresentation
{
    return [self rawText];
}
```

This code is used in section 51.

63 < documentation chunk weaving 63 >

```
− latexRepresentation
{
    return [NSString stringWithFormat:@"\\section{} %@", [self rawText]];
}
```

[1] Throughout the following sections, various literal strings are decomposed into multi-line strings. This is partly for readability, and partly to break `gloom` control strings up so that they are not detected when it parses itself.

This code is used in section 45.

64 < figure chunk weaving 64 >

```
– pngFilename
{
    return [[[self name]
              stringByDeletingPathExtension]
             stringByAppendingPathExtension:@"png"];
}

– latexRepresentation
{
    return [NSString stringWithFormat:
             @"\\begin{figure}\n\\includegraphics[width=0.8\\columnwidth]{%@}\n\\caption{\\label{fig:%@} %@}\
                 ↪ n\\end{figure}\n",
             [self pngFilename],
             [self name],
             [self name]];
}
```

This code is used in section 49.

65

Things are a bit more complicated when it comes to weaving the code chunks. They must refer to both the chunks they include, and the locations where they were included. At its core, the weaving method creates a code listing containing the chunk's code.

66 < code chunk weaving 66 >

```
«code chunk latex label 75»

«code chunk latex reference 74»

– latexRepresentation
{
    id representation = [NSMutableString stringWithFormat:
                          @"\\section{< %@ \\oldstylenums{\\thesection} %@>}\n"
                          @"\\"
                          @"begin{lstlisting}\n%@\n\\"
                          @"end{lstlisting}",
                          [self name],
                          [self latexLabel],
                          [self rawText]];

«list locations including this chunk 68»

«patch up chunks included by this chunk 70»

    return representation;
}
```

This code is used in section 43.

67

The model built a graph of chunks that include other chunks, so this section can find out where it was included and refer to it in the document.

68 < list locations including this chunk 68 >

```
    id includers = [self includingChunksEnumerator];
    id includingChunk = nil;
    while ((includingChunk = [includers nextObject])) {
        [representation appendFormat:@"\nThis code is used in section %@.",
         [includingChunk latexReference]];
    }
```

This code is used in section 66.

69

It also knows the names of chunks that it refers to, so for each chunk that is included «in guillemets» it rewrites the text to incude a section reference.

70 < patch up chunks included by this chunk 70 >

```
    [[self includedChunkNames] enumerateObjectsUsingBlock:^(id _Nonnull name, BOOL * _Nonnull stop) {
        id reference = [NSString stringWithFormat:@"<<"
          @"%@>>", name];
        id replacement = [NSString stringWithFormat:@"\\"
        @"end{lstlisting}\n\\"
        @"texttt{<<%@ \\"
        @"ref{code:%@}>>}\n\\"
        @"begin{lstlisting}", name, name];
        [representation replaceOccurrencesOfString:reference
                                        withString:replacement
                                           options:0
                                             range:NSMakeRange(0, [representation length])];
    }];
```

This code is used in section 66.

71

This is where the empty chunk comes in useful. The code chunk could refer to a section that doesn't exist, because the author hasn't finished writing the document yet or introduced a typo in the reference name. Rather than erroring out, `gloom` patches up the reference to point to an empty chunk. The empty chunk can't appear in the document, but it does know how to describe itself in a document reference.

72 < empty chunk weaving 72 >

```
- latexReference
{
    return @"??";
}
```

This code is used in section 47.

73

The code chunk can also create its own LaTeX reference, as well as a label so that the reference can be matched up to the appropriate section.

74 < code chunk latex reference 74 >

```
- latexReference
{
    return [NSString stringWithFormat:@"\\ref{code:%@}", [self name]];
}
```

This code is used in section 66.

75 < code chunk latex label 75 >

```
- latexLabel
{
    return [NSString stringWithFormat:@"\\label{code:%@}", [self name]];
}
```

This code is used in section 66.

76

Tangling and enmeshing are very similar, they both take the list of chunks and produce a list of code files. A `CodeFile` looks like this.

77 < CodeFile.h 77 >

```
#import <Foundation/Foundation.h>

@interface CodeFile : NSObject

- initWithFilename:aName;
- content;
- filename;

- (void)appendText:text;
- (BOOL)writeOut:(NSError * __autoreleasing *)error;

@end
```

78 < CodeFile.m 78 >

```
#import "CodeFile.h"

@implementation CodeFile
{
    id filename;
    id content;
}

- initWithFilename:aName
{
    self= [super init];
    if (self)
    {
        filename = [aName copy];
        content = [NSMutableString new];
    }
    return self;
}

- filename
{
    return filename;
}

- content
{
    return [content copy];
}

- (void)appendText:text
{
    [content appendString:text];
}

- (BOOL)writeOut:(NSError * __autoreleasing *)error
{
    return [content writeToFile:filename
                     atomically:NO
                       encoding:NSUTF8StringEncoding
                          error:error];
}

@end
```

79

The logic for turning `Chunk`s into `CodeFile`s lives in a `ChunkFiler` class.

80 < ChunkFiler.h 80 >

```
#import <Foundation/Foundation.h>

@interface ChunkFiler : NSObject

- relevantChunkPredicate;

- filesFromChunks:chunks;

@end
```

81

A `FigureEnmesher` is a specific type of a chunk filer.

82 < FigureEnmesher.h 82 >

```
#import "ChunkFiler.h"

@interface FigureEnmesher : ChunkFiler

- enmesh:chunks;
```

```
@end
```

83

It only applies to figure chunks, and tells its parent to make files.

84 < FigureEnmesher.m 84 >

```
#import "FigureEnmesher.h"
#import "Chunk.h"

@implementation FigureEnmesher

- relevantChunkPredicate
{
    return [NSPredicate predicateWithBlock:^BOOL(id _Nullable chunk, NSDictionary<NSString *,id> * _Nullable
        ↪ bindings) {
        return [chunk isFigure];
    }];
}

- enmesh:chunks
{
    return [self filesFromChunks:chunks];
}

@end
```

85

The `CodeTangler` is very similar to the figure enmesher, except that it works on code chunks.

86 < CodeTangler.h 86 >

```
#import <Foundation/Foundation.h>
#import "ChunkFiler.h"

@interface CodeTangler : ChunkFiler

- tangle:chunks;

@end
```

87 < CodeTangler.m 87 >

```
#import "CodeTangler.h"
#import "Chunk.h"

@implementation CodeTangler

- relevantChunkPredicate
{
    return [NSPredicate predicateWithBlock:^BOOL(id _Nullable chunk, NSDictionary<NSString *,id> * _Nullable
        ↪ bindings) {
```

```
        return [chunk isCode];
    }];
}

- tangle:chunks
{
    return [self filesFromChunks:chunks];
}

@end
```

88

The chunk filer asks the chunks to resolve their inclusions, by substituting «other chunk references» for the contents of those chunks. That is done recursively, starting from the top-level chunks that are not included elsewhere. Then it finds all chunks that have the same name, and creates a code file that contains those chunks in order.

89 < ChunkFiler.m 89 >

```
#import "ChunkFiler.h"
#import "Chunk.h"
#import "CodeFile.h"

@implementation ChunkFiler

- relevantChunkPredicate
{
    return nil;
}

- createTangledFileFromChunks:chunks withName:name
{
    id relevantChunks = [chunks filteredArrayUsingPredicate:[NSPredicate predicateWithBlock:^BOOL(id _Nullable
        ↪ chunk, NSDictionary<NSString *,id> * _Nullable bindings) {
        return [[chunk name] isEqualToString:name];
    }]];
    id file = [[CodeFile alloc] initWithFilename:name];
    [relevantChunks makeObjectsPerformSelector:@selector(writeOutToFile:) withObject:file];
    return file;
}

- filesFromChunks:chunks
{
    id chunksToConsider = [chunks filteredArrayUsingPredicate:[self relevantChunkPredicate]];
    id topLevelChunks = [chunksToConsider filteredArrayUsingPredicate:[NSPredicate predicateWithBlock:^BOOL(id
        ↪ _Nullable chunk, NSDictionary<NSString *,id> * _Nullable bindings) {
        return [chunk wasNeverIncluded];
    }]];

    [topLevelChunks makeObjectsPerformSelector:@selector(internaliseInclusionsWithChunks:)
                                    withObject:chunks];

    id allChunkNames = [NSSet setWithArray:[topLevelChunks valueForKey:@"name"]];
    id tangledFiles = [NSMutableArray array];
    [allChunkNames enumerateObjectsUsingBlock:^(id _Nonnull name, NSUInteger idx, BOOL * _Nonnull stop) {
        [tangledFiles addObject:[self createTangledFileFromChunks:chunks withName:name]];
    }];
    return [tangledFiles copy];
}

@end
```

90

It is actually the code chunk that knows how to resolve inclusions, because figure chunks (and non-file chunks) do not need this capability.

91 < code chunk tangling 91 >

```
- (void)findInclusions:(id)chunks
{
    id inclusionFinder = [NSRegularExpression regularExpressionWithPattern:@"<<"
    @"(.*)>>"
                                                                  options:0
                                                                    error:NULL];
    id inclusionMatches = [inclusionFinder matchesInString:[self rawText]
                                                    options:0
                                                      range:NSMakeRange(0, [[self rawText] length])];
    for (id inclusionMatch in inclusionMatches) {
        id matchName = [[self rawText] substringWithRange:[inclusionMatch rangeAtIndex:1]];
        id includedChunk = chunks[matchName];
        if (!includedChunk) {
            includedChunk = [Chunk emptyChunkWithName:matchName];
        }
        [self includeChunk:includedChunk];
    }
}

- (void)internaliseInclusionsWithChunks:chunks
{
    id includedChunkNames = [self includedChunkNames];
    [includedChunkNames enumerateObjectsUsingBlock:^(id _Nonnull name, NSUInteger idx, BOOL * _Nonnull stop
        ↪ ) {
        [[chunks filteredArrayUsingPredicate:[NSPredicate predicateWithBlock:^BOOL(id _Nullable chunk,
            ↪ NSDictionary<NSString *,id> * _Nullable bindings) {
            return [[chunk name] isEqualToString:name];
        }]] makeObjectsPerformSelector:@selector(internaliseInclusionsWithChunks:) withObject:chunks];
    }];

    for (id name in includedChunkNames) {
        id myText = [self rawText];
        id otherChunks = [chunks filteredArrayUsingPredicate:[NSPredicate predicateWithBlock:^BOOL(id _Nullable
            ↪ chunk, NSDictionary<NSString *,id> * _Nullable bindings) {
            return [[chunk name] isEqualToString:name];
        }]];
        id substituteText = [NSMutableString string];
        [otherChunks enumerateObjectsUsingBlock:^(id _Nonnull chunk, NSUInteger idx, BOOL * _Nonnull stop) {
            [substituteText appendString:[chunk rawText]];
        }];
        id inclusionPattern = [NSString stringWithFormat:@"<<"
        @"%@>>", name];
        id finder = [NSRegularExpression regularExpressionWithPattern:inclusionPattern
```

```
                                                                      options:0
                                                                        error:NULL];
        id match = [finder firstMatchInString:myText
                                      options:0
                                        range:NSMakeRange(0, [myText length])];
        [self replaceTextInRange:[match rangeAtIndex:0]
                      withString:substituteText];
    }
}
```

This code is used in section 43.

92

The main function invokes both the tangler and enmesher, telling the resultant files to write themselves to storage.

93 < tangle code files 93 >

```
        id files = [[CodeTangler new] tangle:chunks];
```

This code is used in section 5.

94 < write code files 94 >

```
        __block BOOL writingFailed = NO;
        [files enumerateObjectsUsingBlock:^(id _Nonnull aFile, NSUInteger idx, BOOL ∗ _Nonnull stop) {
            NSError ∗localError = nil;
            if (![aFile writeOut:&localError]) {
                ∗stop = YES;
                writingFailed = YES;
                error = localError;
            }
        }];
        if (writingFailed) {
            fprintf(stderr, "%s\n", [[error localizedFailureReason] UTF8String]);
            exit(EX_IOERR);
        }
        fprintf(stdout, "%s tangled\n", [filename UTF8String]);
```

This code is used in section 5.

95 < enmesh figure files 95 >

```
        id figures = [[FigureEnmesher new] enmesh:chunks];
```

This code is used in section 5.

96 < write figure files 96 >

```
        [figures enumerateObjectsUsingBlock:^(id _Nonnull aFile, NSUInteger idx, BOOL ∗ _Nonnull stop) {
            NSError ∗localError = nil;
            if (![aFile writeOut:&localError]) {
                ∗stop = YES;
                writingFailed = YES;
                error = localError;
            }
        }];
        if (writingFailed) {
            fprintf(stderr, "%s\n", [[error localizedFailureReason] UTF8String]);
            exit(EX_IOERR);
        }
        fprintf(stdout, "%s enmeshed\n", [filename UTF8String]);
```

This code is used in section 5.

97

The one remaining piece of the application to describe is that you have not, so far, seen how to load the `gloom` file, before the chunk finder is invoked.

98 < load gloom file 98 >

```
    id filename = [NSString stringWithUTF8String: argv[1]];
    id basename = [[filename lastPathComponent] stringByDeletingPathExtension];
    __block NSError *error = nil;
    id file = [[NSString alloc] initWithContentsOfFile: filename
                                              encoding: NSUTF8StringEncoding
                                                 error: &error];
    if (!file) {
        fprintf(stderr, "%s\n", [[error localizedFailureReason] UTF8String]);
        exit(EX_IOERR);
    }
```

This code is used in section 5.

99

The following makefile generates all of the artefacts from the woven, tangled and enmeshed files.

100 < Makefile 100 >

```
OBJECT_FILES := Chunk.o CodeChunk.o FigureChunk.o DocumentationChunk.o \
OpeningChunk.o EmptyChunk.o ChunkFinder.o ChunkFiler.o LatexWeaver.o \
FigureEnmesher.o CodeTangler.o ChunkBoundary.o DocumentationBoundary.o \
CodeBoundary.o FigureBoundary.o CodeFile.o main.o

all: gloom gloom.pdf
.PHONY: all

gloom: ${OBJECT_FILES}
	${CC} ${CFLAGS} -o $@ $^ -framework Foundation

gloom.pdf: gloom.tex gloom.png chunks.png
	pdflatex $<
	pdflatex $<

gloom.png: gloom.dot
	dot -Tpng -o $@ $<

chunks.png: chunks.puml
	plantuml $<
```

101

You should also know about the licence: `gloom` is distributed under the terms of the Affero General Public License, version 3 or later.

102 < COPYING 102 >

GNU AFFERO GENERAL PUBLIC LICENSE
Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright−like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non−source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general−purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti−Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non−permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non−Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine−readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer−to−peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non−consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions.

Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non−permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non–permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer–to–peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross−claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non−exclusive, worldwide, royalty−free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non−exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version

3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates