

## Methodological Appendix—CodeRead

Most social research uses data that, at one point or another, are in the form of written text. This includes both traditionally “quantitative” data usually coded from structured or semi-structured interviews as well as “qualitative” data coming from a variety of sources including interviews, focus groups, media analysis, and participant observation. Each of these traditions uses different techniques to distill the information contained in linguistic interactions into analytic categories; generally, survey-based quantitative research codes answers according to pre-determined categories, while qualitative research seeks to identify themes and ideas that recur in the text.

This appendix describes the CodeRead system, a computer analysis package that seeks to facilitate the reliable, specific coding of textual data without requiring a set of *a priori* coding rules. It aims, thereby, to bridge the classic qualitative-quantitative divide by facilitating researchers’ use of unstructured, text-based data in a way that allows for systematic, reproducible analysis. This appendix outlines the purpose of the system as well as the benefits it offers potential users.

### TEXT AS DATA

A variety of qualitative approaches to social research use unstructured text as data. These include studies using media reports (e.g., Franzosi 1995, Tilly 1995), speeches (Batchelor and Szafran 1999), focus groups (Cerulo 1998, Gamson 1992, Sasson 1995), in-depth interviews, and multiple types of text (e.g., Wagner-Pacifici 1994). Most approaches to analyzing such qualitative data involve a coding scheme in which the researcher painstakingly studies the text, using some technology (be it glue and scissors, index cards, highlighter pens, or qualitative data analysis software) to mark portions of the text as representing one or more of the themes salient in the study.

Researchers who use such methods prefer them to structured, survey-style studies because they allow deeper, more extended portrayals of research subjects’ ideas and because they avoid the

problem of pre-determining the categories into which respondents' answers will be divided. However, these methodologies suffer from significant drawbacks as well. Some of these drawbacks are technical: the process of reading and coding text is so time-consuming that studies using qualitative techniques tend to be relatively small, raising questions about their pertinence to broader theoretical questions. Other concerns are more substantive; for example, readers must trust that researchers' coding is correct and consistent, since the studies' conclusions rest entirely on the author's perception of the meaning of the text. It is difficult, if not impossible, to establish measures of the reliability and validity of qualitative codes.

Nevertheless, free-form text is a tremendously promising form of data. Contemporary societies produce huge amounts of such text in the course of everyday life; newspapers, magazines, books, speeches, advertising copy, and legal and political proceedings are just a few of the institutional environments from which text emanates. The growth of the internet makes huge and increasing amounts of this text available for little cost. Furthermore, techniques such as focus groups and interviews yield free-form text that can be analyzed in much the same way. These texts contain far richer, more detailed insights into their subjects than the highly structured quantitative data available.

CodeRead seeks to offer a means for preserving these advantages of free-form textual data while addressing some of the disadvantages. CodeRead uses an open design and a series of algorithms to "learn" patterns of text coded by a researcher, apply these patterns to more text, and produce a variety of summary output for use in analyzing the data. It seeks to approximate the researcher's coding patterns, and its open design allows users to design custom procedures to perform different analyses than those built in.

## COMPUTERS IN TEXTUAL ANALYSIS

CodeRead uses previous work in three traditions: computer content analysis (Stone 1967); computational linguistics (Manning and Schütze 1999); and qualitative data analysis. It is not entirely within any of these traditions; rather, it derives functionality from each.

The computer content analysis tradition goes back to Philip Stone’s 1967 work on the General Inquirer, a computer program that applied “dictionaries” of codes to texts and produced summary output. The General Inquirer – now online at <http://inquirer.wjh.harvard.edu>—relies on a researcher’s predetermined codes to work. That is, researchers provide the program with a dictionary containing a list of words and how they should be coded. The General Inquirer then codes the text(s) accordingly. Also, the General Inquirer codes entirely on the basis of single words; for example, the Harvard dictionary used by the Inquirer’s online version codes “guns” as, among other things, *negative* – an association that is more appropriately tested by the context of the word’s use than assumed *a priori*.

At the time the General Inquirer was originally developed, the computer time necessary was so expensive that it remained impractical to use the system to code and analyze large amounts of text. Also, the requirement that a dictionary be created before the analysis was undertaken meant that coding rules had to be made outside the text itself rather than through an inductive process based on a researcher’s coding practices.

The General Inquirer is able, to a significant extent, to disambiguate word senses; users may therefore find it useful to use the system to attempt to distinguish among the different ways a given word may be used before coding it using CodeRead. For example, the word *over* can have multiple meanings in English. These meanings are established by the context in which the word is used. In the phrase “over the bridge,” *over* has an entirely different meaning than in the phrase

“this meeting is over.” The General Inquirer implements the algorithms published in Kelly and Stone (1975) to distinguish among these “word senses” within texts.

The field of computational linguistics is the most technically developed of the three fields that relate to CodeRead. Computational linguistics seeks to represent complex linguistic structures in systematic ways and, thereby, to enable computers to “speak” in idiomatic ways and to understand human language. Carbonello’s (1981) work, in particular, tried to model the structure of ideological belief systems. However, the emphasis in most computational linguistics is on modeling increasingly complex structures. Most research does not deal with summarizing or coding large amounts of text, or with methods for estimating the meanings of parts of such text.

Finally, there are numerous commercial programs currently available for use in analyzing qualitative data. These include NUD\*IST, WinMax, HyperResearch, Atlas-TI, FolioViews, and AskSam (Dohan and Sanchez-Jankowski 1998). These mostly graphically-oriented programs allow qualitative researchers to mark parts of text with one or another code, then see similarly-coded text arranged by code. With some such software, users can also test hypotheses about the relationships among codes and attach demographic information about the speakers to develop ideas about the texts. Most software in this field, though, concentrates on allowing users to hand-code data and to view the coded data in various ways. There is no attempt to develop systematic ways of coding the texts or of presenting summary information about the data.

The development of microchip-based personal computers and the extraordinary reduction in the price of computer power suggest that computers could be used to standardize and carry out the coding and analysis of very large textual datasets. Computer power that, when *The General Inquirer* was originally written, would have required specialized computer rooms, card punch machines, and large grants is now easily available in inexpensive desktop computers. Similarly, the prices of memory and disk storage have declined so dramatically as to make the storage and manipulation of all but the largest texts a serious possibility.

## FUNCTIONS AND ALGORITHM

CodeRead is a three-stage textual analysis tool available free of charge to interested users.<sup>1</sup>

The three stages a researcher would typically go through in analyzing a dataset using CodeRead are:

- 1.) Feed a portion of the text, coded by hand (see below for details) into the program in order to generate a set of coding rules (called a *pattern file*);
- 2.) Feed the entire text into the program along with a pattern file to generate a systematically, reliably coded textual dataset; and
- 3.) Feed such a coded textual dataset into the program in order to obtain summary information about the data.

All the examples below refer to the following text, the transcript of a radio address given by president Bill Clinton on April 24, 1999 (Clinton 1999). The text has been coded by hand; the codes are shown by angle brackets (<>). Text coded with the ‘parenting’ code has been highlighted for use in the examples. See the technical section below for further details.

```
<:president>
```

```
THE PRESIDENT: Good morning. <religious> Tomorrow in church
services</religious> all across America, we'll be thinking of those
who <death> lost their lives</death> in <Littleton> Littleton,
Colorado. </Littleton> This is a time for all Americans to
<religious>pray </religious> for their families, as well as those
who were <injury> injured </injury> and their loved ones, and all
the people of the <schools> schools </schools> and the community.
It's also a time for all Americans to ask <natl_service> what we
can do—as individuals and as a nation </natl_service> -- to turn
more young people from the path of violence; how we can take
<causality> responsibility, </causality> each and every one of us,
for the future of our <children> children. </children> We've seen
far too many tragedies like the one at Columbine High School.
</natl_service> It's striking that these <violence> violent
assaults on human life </violence> often illuminate the <overcome>
best of the human spirit.
```

<sup>1</sup> CodeRead is not yet complete; this document outlines its specifications. Check <http://demog.berkeley.edu/~aperrin/CodeRead> for updates on the system's progress. CodeRead will be licensed according to the Gnu Public License (GPL), which allows for its free use and distribution. See <http://www.gnu.org/copyleft/gpl.html> for more information.

We marvel at the bravery of the fatally wounded teacher who led 40 students to safety. We look with admiration at the medics and the police officers who rushed to the scene to save lives; <religious> the clergy, </religious> the counselors, the local leaders who immediately began the painful process of helping people to heal; and the parents and students who, <resistance> in the face of hatred, refuse to return it. </resistance> At a moment of such <violence> terrible, terrible violence, </violence> <bravery> these people didn't turn away </bravery> -- <natl\_service> and we can't, either. </natl\_service> Instead, <causality> every one of us must take responsibility to counter the culture of violence. The government must take responsibility. </causality> <legislation> Next week, I'll send to Congress two new bills to <children> keep our children safe. First, we must do more to <gun\_control> keep guns out of the hands of violent juveniles. My bill will crack down on gun shows and illegal gun trafficking, ban violent juveniles from ever being able to buy a gun and close the loophole that lets juveniles own assault rifles. </gun\_control> Second, we must do more to prevent <violence> violence in our schools. </violence> My Safe Schools bill will help schools pay for more counselors and conflict resolution programs, more mentors and more metal detectors. It also includes \$12 million for emergency teams, to help communities respond when tragedy strikes. <causality> And government can help parents take responsibility. It's harder than ever for parents to pass on their values in the face of a <media> media culture that so glorifies violence. </causality> </media>

As Hillary pointed out in her book, the more children see of violence, the more numb they are to the deadly consequences of violence. Now, <video\_games> video games like "Mortal Kombat," "Killer Instinct," and "Doom"—the very game played obsessively by the two young men who ended so many lives in Littleton—make our children more active participants in simulated violence.

A former Lieutenant Colonel and psychologist, Professor David Grossman, has said that these games teach young people to kill with all the precision of a <military\_virtue> military training program, but none of the character training that goes along with it. </military\_virtue> <parenting> **For children who get the right training at home and who have the ability to distinguish between real and unreal consequences, they're still games. But for children who are especially vulnerable to the lure of violence, they can be far more.**</parenting>

<vice\_president> Vice President Gore has led the fight to give <parenting> **parents the tools to limit the exposure of their children to excessive violence, from a <media> television rating system </media> to <internet> new ways of blocking inappropriate material on the Internet </internet> to the V-chip.** </parenting></vice\_president> By this July, fully half of all new televisions will have the V-chip; so will every new television in America by the year 2000. Years ago, Tipper Gore sounded the first alarm about the damaging effects on our children of excessive violence in <media> <video\_games> movies, music and video games. </media> </video\_games> Today, she is still drawing attention to mental illness. This June, she will host the first ever White House Conference on Mental Health, where we'll talk about how to recognize mental illness in young people before it's too late.

```

These are steps the national government is taking <children> to
protect our children. </children> <parenting> But it is not a job
government can or should do alone—parents come first. They should
<media> turn off the television,</media> <internet> pay attention
to what’s on the computer screen, </internet> refuse to buy
products that glorify violence. Make sure your children know you
care about what they’re doing. </parenting>
<media> And to the media and entertainment industries, I say just
this: you know you have enormous power to educate and entertain
our children. Yes, there should be a label on the outside of every
video, but what counts is what’s on the inside and what it will do
to the insides of <children> our young people. </children> I ask
you to make every video game and movie as if your own children were
watching it. </media>
In the days ahead, as we continue the process of healing, we must
pledge ourselves to the task of putting an end to the culture of
violence and building in its place a culture of values we can be
proud to pass on to all our children.
Thanks for listening. </:president>

```

### STAGE 1: GENERATE CODING RULES FROM CODED TEXT

The first step in creating a CodeRead project is to code portions of a text by hand. The amount of text to be coded at this stage is left up to the researcher; it is entirely possible, however, to return to the project multiple times, refining the coding technique using feedback from the system. Such a technique allows users to determine at what point the patterns the computer has ‘learned’ are sufficiently similar to the by-hand coding to perform the rest of the analysis automatically. This original coding is done in exactly the same way the researcher would code traditional qualitative data, and the codes are entered according to the marking scheme discussed below in the technical section of this appendix.

The user then enters this coded text into CodeRead and instructs the program to generate a series of patterns for each code. The patterns CodeRead generates are in the form of *regular expressions* (see Freidl 1997). By convention, regular expressions are bounded by forward slashes (/); I include their schematic representations here for reference. CodeRead will try to generate patterns in each of the following forms:

- containing a particular ordered, contiguous set of one or words (a phrase), e.g., code all instances of ‘parents’ as ‘parenting’, represented as /parents/

- containing a particular ordered, semi-contiguous set of words, e.g., code all text areas containing ‘children’ followed by ‘violence’ in that order, within ten words of one another, as ‘parenting’, represented as `/children\b{1,10}parenting/`; and
- containing a particular non-ordered, semi-contiguous set of words, e.g., code all text areas containing ‘parents,’ ‘violence,’ and ‘children,’ in any order but within ten words of one another, as ‘parenting’, represented as `/parents/&&/violence/&&/children/`.

CodeRead is capable of using *any* properly formed regular expression to code text; however, in its current form, these three types of patterns are the only ones it can ‘learn’ from user-coded text.

The output from this stage is a list of coding rules, each with two scores. The first score (the pattern’s *reliability*) corresponds to the proportion of text areas coded with the relevant code that match the pattern. The second score (the pattern’s *specificity*) is the proportion of text areas *not* coded with the relevant code that do *not* match the pattern. Each of these scores is between 0 and 1. In the case of the example above, the first pattern (all occurrences of ‘parents’) has a *reliability* score of 0.667 (since two of three areas coded as ‘parenting’ match the pattern) and a *specificity* score of only 0.5, since two of the four areas *not* coded as ‘parenting’ also match the pattern.

The second pattern (‘children’ followed by ‘violence’ within 10 words) also scores 0.667 for reliability, since the third area coded ‘parenting’ does not match (since ‘children’ follows ‘violence’), and 0.5 for specificity, since again two of four areas not coded as ‘parenting’ nevertheless match the pattern. Finally, the third pattern (‘parents,’ ‘violence,’ and ‘children’ within ten words of one another) also gets 0.667 for reliability, but gets a 1.0 for specificity, since no text would be incorrectly coded as ‘parenting’ using this pattern.

The file CodeRead would produce, then, would contain (among others) the following lines. The file it produces is called a *pattern file* and is in a format that can be read by Stage 2 of the process.

<code>/parents/</code>	<code>parenting</code>	<code>0.667</code>	<code>0.500</code>
<code>/children\b{1,10}violence/</code>	<code>parenting</code>	<code>0.667</code>	<code>0.500</code>
<code>/parents/~/violence/~/children/</code>	<code>parenting</code>	<code>0.667</code>	<code>1.000</code>

### STAGE 2: APPLY CODING RULES TO TEXT

For the second stage, the user produces a list of patterns in a pattern file – produced by Stage 1, by hand, or both – and uses the list and the entire text to be coded as input. CodeRead applies the patterns to the text, marking sections of text with the appropriate codes. Users may write regular expressions to match patterns they want coded according to a custom pattern; CodeRead will apply any properly-formed regular expression, regardless of its origin. For example, a user may note that the pattern ‘either the word *children* or *products*, followed by *violence*’ would have a reliability of 1.000. Even though CodeRead could not generate this pattern automatically,<sup>2</sup> the user could enter the pattern in a pattern file:

```
/(children|products)\b{1,10}violence/ parenting
```

and CodeRead would mark areas that matched the pattern as ‘parenting.’ Note that it is unnecessary to generate reliability and specificity scores for user-defined patterns; they are not needed to apply patterns to a text. They are only provided by CodeRead to assist users in choosing among patterns to be applied.

When applying a pattern file, CodeRead checks the text in overlapping two-paragraph blocks. That is, it first checks paragraphs 1 and 2 together; then 2 and 3; then 3 and 4; and so on. This approach insures that patterns that could span a paragraph boundary are not missed. It applies the least restrictive (“greediest”) match possible; for example, consider the text:

```
Her children took part in violence, but it was not the violence
that killed them.
```

Applying the pattern above, the match is ambiguous; the pattern matches both “children took part in violence” and “children took part in violence, but it was not the violence”. CodeRead resolves

<sup>2</sup> CodeRead’s modular design means it could be expanded in the future to attempt to generate other types of patterns automatically.

this situation in favor of the more inclusive match; in this case, it codes “children took part in violence, but it was not the violence”:

```
Her <parenting>children took part in violence, but it was not the violence</parenting> that killed them.
```

The output of Stage 2 is a fully-coded text. The text is in a form that CodeRead can then read for the following stage; all codes are marked with angle brackets, and punctuation is preserved.

### STAGE 3: OBTAIN SUMMARY INFORMATION ABOUT THE DATA

Once a fully-coded dataset is available (whether coded by hand or using Stages 1 and/or 2), CodeRead can produce a variety of summary information about it. CodeRead generates an internal representation of the dataset (see below for a complete description) and uses this representation to produce output as requested.

Some forms of output consist solely of sorting text according to code. For example, CodeRead can produce a listing of all areas coded as a particular code, or a “key word in context” (KWIC)-style listing that presents coded text within its context such as the following:

```
but none of the character training that goes along with it.
</military_virtue> <parenting> For children who get the right
training at home and who have the ability to distinguish between
real and unreal consequences, they're still games. But for
children who are especially vulnerable to the lure of violence,
they can be far more.</parenting>
<vice_president> Vice President Gore has led the fight to give
<vice_president> Vice President Gore has led the fight to give
<parenting> parents the tools to limit the exposure of their
children to excessive violence, from a <media> television rating
system </media> to <internet> new ways of blocking inappropriate
material on the Internet </internet> to the V-chip.
</parenting></vice_president> By this July, fully half of all new
These are steps the national government is taking <children> to
protect our children. </children> <parenting> But it is not a job
government can or should do alone—parents come first. They should
<media> turn off the television,</media> <internet> pay attention
to what's on the computer screen, </internet> refuse to buy
products that glorify violence. Make sure your children know you
care about what they're doing. </parenting>
<media> And to the media and entertainment industries, I say just
```

Other output options include a listing of all active codes in the document along with their frequencies and information about how much of the text is coded with one or more codes.

More advanced options are available as well. CodeRead will produce a correspondence matrix suitable for performing a correspondence analysis (see Krinsky 1998; Clausen 1998). It will also perform a *proximity analysis*. This is a score for any two codes corresponding to the mean distance between occurrences of the first code and the nearest occurrence of the second. A kappa score (Kelly and Stone 1975: 41) is also available; this measures the degree of ‘agreement’ between two codes, effectively measuring how much text is coded with both of the two codes.

Because of the modular structure of the algorithm, any analysis that can be done on an entire project can also be done on any portion of the project. Thus users can analyze parts of projects without re-coding each individual part.

In addition to these built-in analysis modes, CodeRead provides a rigorously documented data structure containing the dataset in a flexible, internally-represented form. This allows users to write functions in Perl to access and format the data in whatever way they require. CodeRead is therefore not restricted to performing a few built-in functions; it can be expanded relatively easily as requirements and uses for the system change.

## **CODEREAD**

CodeRead is written entirely in Perl, a free language that runs on most modern computers, including Unix/Linux, Windows, and Macintosh.<sup>3</sup> It is generally available under the GPL and its source code is also available. That means researchers can use it without any additional cost, and they can modify it to suit their needs. Perl is particularly adept at text manipulation using regular expressions, so much of the functionality that powers CodeRead is, in fact, built into Perl itself. Perl is also a fully-functioning programming language, so technically skilled users can build on the CodeRead model to perform a variety of different analyses on textual datasets.

---

<sup>3</sup> CodeRead requires Perl version 5.005 or later because of the addition of the `qr//` operator, which greatly facilitates the use of stored regular expressions. See <http://www.perl.com> for more information on the language or to download a copy.

CodeRead works by applying a ‘brute-force’ algorithm. That is, it does not seek to use artificial intelligence or other predictive techniques to anticipate the likelihood of patterns in text. Rather, it simply counts the occurrences of words and phrases and ‘tries’ them until it encounters one that matches. This is not an especially efficient way of searching text, but it guarantees that all possibilities will be considered. It is particularly inefficient during Stage 1, when the system must try large numbers of patterns gleaned from each coded area. Nevertheless, the amount of computing power available for social science work has grown so quickly that these techniques are quite reasonable, even for large datasets.

In addition to its computational abilities, the CodeRead system has the side benefit of defining a standard format for distributing qualitative datasets. Since CodeRead files are simply standard text files with coding information stored in them, researchers can distribute these files as well as pattern files. This allows for reproducibility and transparency of methods; researchers can share and cross-analyze datasets in much the same way that collaborative work is now done on quantitative datasets.

CodeRead does not offer the user-friendly interface most qualitative data analysis software aims for.<sup>4</sup> It is possible to use CodeRead as an interactive tool, but it is designed mainly to be used in ‘batch’ mode: users prepare a project, a set of instructions, and (if required) a pattern file, then start the system. The output is designed to be interpreted directly, although some output could be used as input to a statistical package, for example to perform a correspondence analysis.

The system is also designed mainly to allow for reductive analyses of texts. Although it is useful for traditional qualitative data analysis tasks such as compiling all text coded in a certain way or displaying coded text in context, the main innovation of CodeRead is its ability to count and graph the occurrences of codes within a text.

---

<sup>4</sup> The Perl/Tk package does allow for producing user-friendly windows-style interfaces to Perl programs, so it is possible to write such interfaces; however, I have no plans to do that in the future.

## TECHNICAL INFORMATION

### REPRESENTING CODED TEXT

For all its operations, CodeRead uses a subset of the Standard Graphics Markup Language (SGML; see ISO 1986) to represent codes inserted into text. Coding tags are enclosed in angle-brackets (<>), which means these cannot appear elsewhere in the text. Coded regions of text begin with a begin-code marker consisting of a code name enclosed in angle-brackets, and end with the same code name, enclosed in angle brackets and preceded by a slash (</>). Code names may be any length, but must contain no spaces, line breaks, backslashes (\), braces ({}), punctuation marks (., ,, !) or hashes (#), since these are interpreted by the compiler. In addition, the equals sign (=) has a special meaning within a coding tag, so it cannot be part of a code name. Codes may overlap or even be coterminous; there is no limit on the number of codes that can be linked to any single part of the text.

Code names that begin with a colon (:) will be ignored by the Stage 1 pattern-recognition engine. This is useful for entering objective codes such as information about a speaker, the circumstances of an interview or focus group, or other information that is not based on the content of the text itself.

In addition to codes, CodeRead uses braces ({}), to mark *processor directives*. Although there are no processor directives implemented in the current version of the system, these will eventually be used to direct the system on how to interpret codes, words, and punctuation marks. Future versions of the software will allow users to specify other parameters as well. For example, users will be able to determine how much text surrounding a matching pattern should be included when applying patterns and how simple or complicated the word combinations for which CodeRead searches should be.

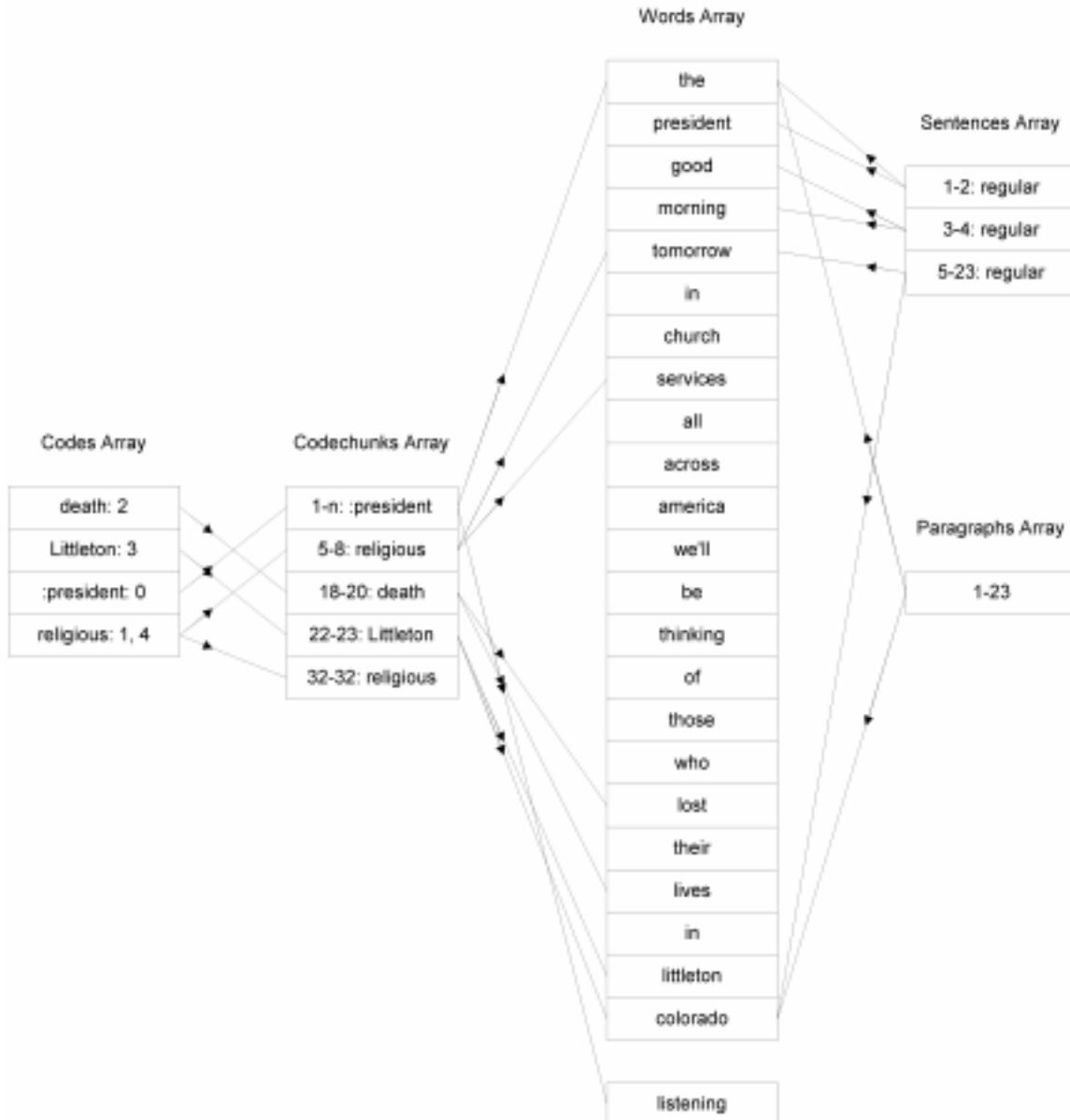


Fig. 1 – The Project data structure encoded in CodeRead::Project

When reading any file – whether coded fully, partially, or not at all – CodeRead generates a large, ordered list (represented internally as an array) of all the words in the text. The words are all represented in lower case to facilitate comparisons, and punctuation and whitespace are removed. This simple data structure, called the *words array*, is the basis for all CodeRead analysis.

In addition to the words array, CodeRead generates a series of pointers to structure the text. Each pointer consists of a *textchunk* structure, which is simply a combination of a pointer to the words array along with numbers corresponding to the area's starting and ending point in the words array (Fig. 2). In addition, some types of textchunks contain codes or other indicators of how

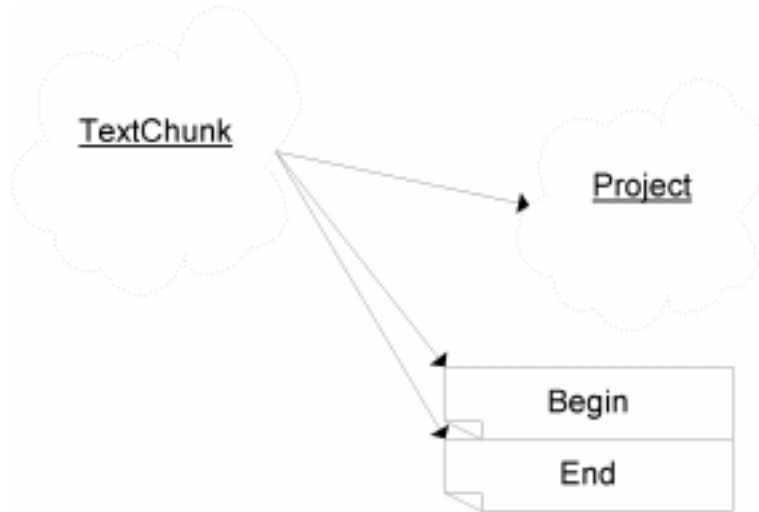


Fig. 2: The TextChunk structure defined in CodeRead::TextChunk.

the chunk of text to which they point is to be interpreted. All CodeRead datasets contain at least two sets of pointers: the *paragraphs* array, which splits the text based on paragraph markers (generally, two newline characters together), and the *sentences* array, which splits it based on punctuation marks. The sentences array contains references to the type of punctuation that closed the sentence: 'regular' for those punctuated with a period, 'question' for those punctuated with a question mark, and 'exclamation' for those punctuated with an exclamation mark. Future versions of the program will allow users to specify processor directives to change this association.

For texts with any codes in them, CodeRead also generates two more indexes. These are the *codechunks* array and the *codes* array.<sup>5</sup> The codechunks array contains an ordered list of parts of

<sup>5</sup> Technically, the codes array is actually an *associative array*, also known as a *hash*, but for ease of reference it is simply referred to as an array. Technical users may consult the .pod documentation inside the program for more detail.

the text that have been coded and the code they have been assigned; the codes array does the reverse, containing a list of coded text areas for each code name.

CodeRead is implemented as a top-level Perl package (CodeRead.pm), which contains most of the documentation but little code, plus two lower-level packages, Project.pm and TextChunk.pm. Each of these lower-level packages defines the object class for which it is named. Most of the data structure used in the system is contained in Project.pm. Project.pm defines the internal representation of CodeRead datasets (Fig. 1).

There are very few methods (procedures) contained in Project.pm; most requests are passed off to the TextChunk.pm module, which contains most of the methods. This insures that any operation that can be performed on a Project can also be performed on any part of a project by declaring it a TextChunk.

In contrast, TextChunk.pm has a very simple data structure; it contains only a reference to the Project of which it is a part, plus begin and end points referring to the parts of the Project's words array to which the TextChunk refers. Indeed, all the references in Figure 1 are in fact TextChunks, which means any of them may be analyzed as a separate project. The complicated work, however, is done by TextChunk; it is here that pattern generation, scoring, and output are created.

Although the three packages – CodeRead.pm, Project.pm, and TextChunk.pm – are distributed as separate files, no one of them is likely to be useful without the other two. Since CodeRead.pm calls the other two, it should never be necessary to use CodeRead::Project or CodeRead::TextChunk explicitly. For further technical documentation, see the CodeRead.pm file, which contains standardized POD (Perl's 'plain old documentation' format). For more information on using Perl, see Wall (1996). For specific information on using object-oriented Perl such as CodeRead, see Srinivasan (1997) or type perldoc perltoot on a computer with a recent installation of Perl.