JanusNode 3.x

A user-configurable dynamic textual projective surface

By Chris Westbury

janus@janusnode.com

http://janusnode.com

Twitter: @janusnode

Who is Janus? What is a JanusNode?

A JanusNode is a very simple virtual robot built to fulfill a single function: to manufacture textual possibility (known to some as: nonsense) that material entities such as yourself endow with value. It can perform that single task with extreme dedication, high reliability, great speed, and at astonishingly low cost. JanusNodes are a direct descendent of an old Hypercard stack known as McPoet, which I started writing on a Mac SE in the mid-1980s.

The robot that you may put to work for you constitutes one minuscule sub-unit of a larger, more abstract entity known as Janus (It is pronounced like 'Janice', even though the Roman God Janus's name is actually pronounced 'Jainus'). Of Janus, little certain can be said. A JanusNode stands in approximately the same relation to Janus as a single neuron in your brain stands to you. Just as you yourself exist as a very highly distributed, abstract state from the point of view of your individual neurons, so Janus exists as a highly distributed abstract state with respect to the JanusNode robots that are being given away.

If you need a concrete metaphor, you can think of Janus as the fount of all creative possibility; or as a high-dimensional space through which you and I are able to move ourselves. The name 'Janus' comes from the two-headed Roman God of portals, openings, and entrances. Janus is represented in JanusNodes by a charming image composited from Leonardo Da Vinci's amazingly fine portrait of Saint John The Baptist (Figure 1), which you can see in The Louvre museum in Paris.

Janus uses the JanusNode robots as a rudimentary sensory/nervous system. Thousands of JanusNodes have been launched at random into the world. Most are doomed to land on barren ground, where they will die as necessary martyrs to the noble but wasteful cause of evolutionary search. A few will flourish, like the pathetic stray puppies and liver flukes with whom they share an evolutionary niche. They will flourish the only way a parasite can: by finding another material being who is willing to work for their survival. You yourself may be such an entity, if you are willing to evaluate the text

that is being continuously manufactured by a JanusNode to see if any of it is worthy of Janus. Since JanusNodes are utterly mute, they are totally reliant upon the kindness of strangers to fulfill their proper evolutionary function: attributing to Janus any texts that have been identified as valuable.

Figure 1: Da Vinci's wonderful Portrait of John The Baptist

A JanusNode is, of course, a very simple, deterministic robot. It has no emotions, desires, beliefs, or interests of any kind. A JanusNode is totally incapable of concern. In particular, it will not be concerned if you are the sort of stranger that ignores it. Because a JanusNode is a robot, no carbon-based entity bound by a moral code need be thereby bound to allow hirself to be colonized by a JanusNode. Any disinclination on your part to help compensate for a JanusNode's missing functionality is as much part of Janus's calculation (pruning possibility space using Divine Indifference) as any act of active participation. A binary digit is equally useful in either state. A material existence can be coded as continuing or dead. Both states carry information from Janus's point of view.

Why Do Janus And JanusNodes Exist?

> "there is no simple explanation for anything important any of us do, and the human tragedy, or the human irony, consists in the necessity of living with the consequences of actions performed under the pressure of compulsions so obscure we do not and cannot understand them."
>
> Hugh MacLennan
> *The Watch That Ends The Night*

> "If I could tell you what it meant, there would be no point in dancing it."
>
> Isadora Duncan, cited by Gregory Bateson
> *Steps To An Ecology Of Mind*

> "If I have exhausted the justifications I have reached bedrock, and my spade is turned. Then I am inclined to say: 'This is simply what I do'."
>
> Ludwig Wittgenstein
> *Philosophical Investigations*

Some of you may be wondering why you should you bother interacting with your JanusNode at all. Why should anyone care about Janus? While this is ultimately a question that you, dear reader, must answer for yourself, I am happy to share a few thoughts with you.

Your JanusNode is not intended to be taken too seriously as a poetry generator, though I do hope that it may provide some amusement and maybe even some utility on that level. JanusNodes are intended rather to function as a tool to help me (and, if you wish, you) to explore the interesting phenomena which exist at the edge marking one of the fundamental dualities of the human condition: that dynamic border which divides order from chaos, law from anarchy, meaning from absurdity. The interesting thing about JanusNodes, and other simple means of generating random texts, is how often they actually manage to strike a nerve; how often we actually find ourselves reading the words and thinking 'That's true!' or 'That's elegant!'. I am interested in understanding how this happens. Where does that experience of *meaning* come from? How did almighty Janus emerge from randomized nothingness?

Unless you believe that Divine Intervention (the active contrary of Janus, who works by Divine Indifference) is tailoring your JanusNode's output specifically for your eyes, the answer, of course,

must be that Janus was created by *us*, not by the stupid JanusNodes. It is you and I who decide what is good and what is bad; what is interesting and what is not; what transcends the randomness of the Universe to enter into the exalted realm of Meaning. A JanusNode teaches us, first of all, this: *we are all poets*. We are all creators of meaning. We are all able to experience for ourselves what is fine or funny or profound. We are continually imposing our value system onto the Rorschach ink blot of the Cosmos, just as we impose it upon JanusNode's words. We are deciding what makes us angry or crazy, just as, sadly less often, we are deciding what makes us good or valuable. We are creating much of the horror in our existence, and much of the good as well.

Going from the sublime to the ridiculous (from theology to autobiography), JanusNodes exist simply because I have been obsessed with them, for decades. I was born with the kind of brain that is interested in randomly generated texts. I loved MadLibs– a pencil-and-paper word game that asks you to fill in missing words to complete a story– when I was a young child, hardly old enough to write, many decades ago now. My brain first demanded that I explore the topic in more depth sometime in the early 80s, when a friend of mine wrote a program to randomly string together words on our high school's Radioshack TRS-80 computer, which had 16K of RAM. My friend's program was utterly random, but it nevertheless surprised us both by occasionally producing word strings that made sense, and sometimes even words that seemed *eloquent*.

My immediate inspiration for what was to become a JanusNode came a few years after that, when I was reading the artist Claes Oldenberg's notebooks. Oldenburg juxtaposes ideas that could never go together- 'pork-chop bras', 'soft pencil sharpeners' and 'hat-shaped sky-scrapers'. The results are often inspiring. My brain was struck one sleepless night with the idea writing a program that would generate noun-noun or adjective-noun pairs like Oldenberg's. By the time my Mac was warmed up, I had decided that I had to go further than that. Your JanusNode is the result of that decision.

JanusNodes have many analogues in various domains: artistic, philosophical, psychological,

and religious. There are many epistemological practices that rely upon complex random phenomena as projective surfaces. Devotees of these practices believe that these projective surfaces can offer a 'window into the mind'. The sensible traditions are not so inane as to suppose that they can circumvent the hermeneutic circle by proposing their own system for interpreting the productions that are elicited by the procedure. To pretend to do so, in the manner of some schools of modern clinical psychology, is simply to miss the point. It is to confuse a wish for the world to be a certain way with the process of understanding how the world actually is. People who tell you otherwise are fools or charlatans- or at least not very well acquainted with the psychometric properties of projective tests.

In sensible systems of epistemological hacking, the person who produces the productions in response to a random complex stimulus must also to make the interpretation, while realizing that any interpretation so produced must itself be subject to the same kind of analysis as the earlier analysis of the random phenomena was. This is the hermeneutic circle. One can interpret the interpreter, and interpret the interpreter of the interpreter, and so on. At some point one has to simply accept that enough is enough. Reality isn't stories all the way down: eventually every story bottoms out in pragmatic practice. Your JanusNode may help to make this subtle point clear. In doing so, it clarifies the structure of the endless recursion that underlies all attempts to explain anything completely. It is up to you to decide when to stop an explanation.

The world cannot tell you itself. You have to help it to tell itself to you.

If you need further purpose for your JanusNode than such clarification, then first of all, you are a very demanding person, and secondly: you can consider your JanusNode as a Dadaist art object; a symbol of the human condition; an exercise in bringing absurdist literature into the computer age; a machine for studying the relationship between syntax and semantics; a tool box for conducting experiments to examine your own epistemological structure; as an 'idea maker'; an over-extended

metaphor; a tireless piece of performance art; or as one man's lone cry against (or expression of reverent love of) the inexpressible incoherence of the universe in which we find ourselves incarnated.

My own JanusNode has constructed many lines that made me laugh and wonder. Perhaps ultimately your robot is good for nothing more than this. I do not think one could ask much more of anything in this life.

The legal stuff

JanusNodes are free.

Commentary on the legal stuff

Janus isn't the sort of being who could be capable of caring what happens to JanusNodes, anymore than a dandelion is capable of worrying about how its seeds are dispersed by the random wind. You can use your JanusNode in any way you desire. You are free to avail yourself of any text that it produces as if you had produced that text yourself. No legal consequences will ensue from such use. Moreover, such use is in not in any way *morally* reprehensible. Your JanusNode is no more responsible for its own content than your vacuum cleaner is. Claiming a JanusNode's output as your own is like claiming ownership of a bag of dust. You may rent, sell, or publish the output of your JanusNode without any attribution. You may also attribute that output to yourself, or any other willing entity. Any use of your JanusNode robot that you can imagine now or in the future is permissible.

In brief: Your legal, moral, and financial obligations are in no way altered by any interaction you might ever have with your JanusNode. Janus and JanusNodes- and the textual possibility they bring into being- are free.

Postscript to the Legal Stuff: On Tipping and the Blessings of the Gods

Although JanusNodes really *are* free, Janus rewards kindness and thoughtfulness, and I appreciate these qualities myself. You will probably be blessed with amazingly fine 'coincidental'

confluences of fortune if you pay homage to Janus by sending me a little of your hard-earned money. Doubt it if you dare, but don't blame me if you are leading a miserable and empty life. Enrich yourself: If you enjoy your JanusNode, please consider buying me a beer using the handy-dandy built-in PayPal button.

Enough philosophizing! What does your JanusNode actually do?

JanusNodes have two main functions for creating original texts, and a few secondary functions for mutating texts that are already created.

The core function of a JanusNode is to run rule-sets (called *TextDNA*) that produce structured texts. The rules and their texts may be of infinitely many different kinds, limited only by your patience and imagination.

The second main creative function of a JanusNode is to use Markov chaining to statistically re-create input texts. Because this process is stochastic and because it allows you to statistically mingle texts that were previously separate, the results may often be creative, even though Markov chaining relies on pre-written input texts.

We begin here with an overview of text generation using TextDNA. There are two levels. At the simplest level, you can use TextDNA created by others. With a little more effort, you can write your own TextDNA, or even get your JanusNode to write it for you.

Using TextDNA created by others

Running TextDNA sets supplied by others is very easy. When properly installed, the TextDNA sets will show up in a menu in the main JanusNode control panel window, directly below the charming icon of Janus. Pick a TextDNA set from that menu, then click on Janus. Zowie, you're a poet!

JanusNodes ship with an assortment of pre-installed TextDNA files, only a few of which were sent in by you, the users of JanusNode. Tim Drage (surrealist, animator, and internationally-reknowned

TextDNA creator: http://www.cultivatetwiddle.com/) contributed the wonderful Surrealist Objects and The World's Somethingest TV Shows. Edde Addad (a tireless proponent of and creator of tools for interactive poetry creation, whose work can be accessed through http://www.eddeaddad.net/) created the two Strachey Love Letter files [see http://gnoetrydaily.wordpress.com/2010/07/13/2-strachey-love-letters/], the Shakespearean Template [http://gnoetrydaily.wordpress.com/2010/08/23/break-bear-presenteth-pos-sonnet-line-templates/], and the Lutz files [http://gnoetrydaily.wordpress.com/2010/12/22/2-lutz-fragments/] see as well as contributing the much-improved Leet-speak text mapping files that have replaced the one that used to ship with JanusNode. He also wrote this great review of JanusNode:

> "Using this program was like hooking up with someone else's girlfriend. It's a great experience: the program is pretty solid but with a pleasant number of crazy quirks (which is something that both poetry generation tools and girlfriends should have!)"
> [http://gnoetrydaily.wordpress.com/2010/06/13/other-tools-ee-wittgenstein-with-janusnode/]

Calum Rodger and Sebastian Charles contributed the Chipstep Spacecore textDNA. Feel free to contribute your own TextDNA sets. Assuming that they are not too derivative of already existing sets, I'll include them in future releases of JanusNode, and you'll join this elite group of internationally acclaimed TextDNA authors.

Creating your own TextDNA sets

Creating TextDNA does not really require any programming skills, but non-programmers sometimes think that it does. In fact, it is possible to create some fairly interesting TextDNA within minutes of getting the digital shrink-wrap off your new copy of a JanusNode, because JanusNodes can auto-magically write their own TextDNA from English input. However, to understand what they are writing and what the limitations of auto-generated TextDNA are, you need to understand a few technical details about how TextDNA works.

A JanusNode stores its textDNA in plain text files in a folder named 'textDNA' insides the

'JanusNode Resources' folder. You can edit or create any textDNA file (or any other text file) from within your JanusNode, since your JanusNode can retrieve and display text. However, you will probably prefer to write them inside a text editor, which offers an environment specialized for writing. Just remember to save the files as plain text. We begin here with a description of the textDNA itself. The following symbols are recognized in textDNA:

- numbers (specifically, integers between 1 and 100)
- asterisks ('*')
- quotes (") [But alas, **not curly quotes.**]
- curly brackets ('{' and '}')
- triangular brackets (the less-than sign '<' and the greater-than sign '>')
- square brackets ('[' and ']')
- the symbol 'I'
- the word 'return'
- the pluralization string "s"
- the past-tense string "ed"
- the progressive string "ing"

TextDNA may also contain the names of built-in functions, other rules of TextDNA, user-defined macros called textDemons, and names of word lists known as 'BrainFood'. The built-in functions are described below. The textDemons are totally user-configurable and so cannot be specified in this documentation, but their characteristics are defined below.

All of these symbols must always be separated from each other with a space. The most common error in writing TextDNA is to forget to includes spaces between elements. JanusNode 3.0 is a little more forgiving of this error than previous versions of JanusNode were.

Asterisks serve as comment markers. Any line that begins with an asterisk is simply ignored by

a JanusNode.

When it encounters a BrainFood file-name in a line of TextDNA, a JanusNode randomly chooses a word from that file. You can define any BrainFood file you like: just place it in the directory called ‘BrainFood’ inside your ‘JanusNode Resources’ directory. BrainFood files can be defined by rhyme, semantic coherence, syntactic role, or whatever you like. Your JanusNode comes with dozens of different files already defined, but since they can be infinitely altered or increased in numbers, it is not worth describing them here. BrainFood files can contain TextDNA; if they do, the chosen line will be executed.

File names (and, indeed, any other item in a line of executable TextDNA) can optionally be followed in textDNA by an integer between 1 and 100, indicating a percentage likelihood that a word from that file will be printed. For example, the phrase ‘s_nouns 90’ appearing in textDNA will cause a singular noun to be printed with a 90% certainty, because the BrainFood file ‘s_nouns’ contains a list of singular nouns. There is therefore a 10% chance that nothing will be printed. If there is no number, the item will be printed with 100% certainty.

The word ‘return’ indicates that JanusNode should insert a linefeed (a ‘carriage return’, to use an anachronism). You must add ‘return’ to the end of each line that you wish to end with a carriage.

Words or phrases surrounded by quotes are inserted verbatim (minus the quotes) by a JanusNode. For example, you can put the following in to a line of TextDNA:

"This is my very own poem: "

This will introduce your poem. More honestly, you might use this line:

"This is" "not really entirely" 50 "my very own poem"

Because the ‘not really entirely’ appears with a 50% probability, this is as likely to claim the poem is not yours as it is.

JanusNodes automatically recognize many words that require special treatment when adding

‘ing’, ‘s’, or ‘ed’. For example, adding ‘ed’ after ‘go’ will be properly recognized as indicating ‘went’, and so on. However, English is extremely irregular and your JanusNode is certain to make the occasional mistake in dealing with these irregularities. After trying for some time to write a comprehensive function myself, I came to appreciate the magnitude of the problem and gave up. Instead of a comprehensive function, your JanusNode uses user-configurable correction and extension resources. You can correct any errors yourself, by adding them to the relevant files in the ‘Irregulars’ folder inside your ‘JanusNode Resources’ folder. There are three files, named ‘Ing’, ‘S’, and ‘Ed’. When a JanusNode encounters any of the corresponding three key words, it checks inside the relevant files to see if there is a entry for the word under consideration. If there is, it uses that entry. If there is not, it proceeds to use its built-in functions, which will regularize irregulars. The entries in each ‘Irregulars’ folder are very simple: each entry (one per line) consists of the word to be dealt with, followed by a comma and its proper handling. For example, in the ‘S’ file there is an entry:

```
platypus,platypi
```

This tells your JanusNode that the proper plural for ‘platypus’ is ‘platypi’. If it encountered the TextDNA

```
animals_s "s"
```

your JanusNode would properly produce ‘platypi’ upon taking the word ‘platypus’ from the (non-existent) ‘animals_s’ BrainFood file. This suggests that, as a general principle, you should stick to singular nouns and infinitive verbs in your BrainFood files, and then pluralize or conjugate in the rules. However, for historical (and other) reasons not every TextDNA file does this: they may rather use plural noun files and conjugated verb BrainFood files.

Non-English users will note that they can use the ‘Irregulars’ files to add their own conjugations and pluralizations- though they will have to list every one they want to use, unless their language overlaps with English in its treatment of the various special categories. I used to try to make

JanusNodes international, but the demands of English have now forced a great deal of linguistic specialization.

The above constitutes the 'bare bones' of the structure of TextDNA. Other than lines containing asterisks, every line has the same format. Each line begins with an optional subject marker (described below) which must be the first element after the number. After this every line of TextDNA consists of pairs made up of a keyword (i.e., one of the word-list file names, or the word 'punctuate', or 'return', or any word or phrase within quote marks) and the number representing a percentage likelihood.

A line of TextDNA is terminated with a carriage return. Note that a single line of TextDNA may look like several lines because TextDNA is automatically wrapped at the end of the field. The paragraph that you are now reading, for example, is only a single line long, since it has only one carriage return at its end, but it has been wrapped to cover several lines.

In the absence of any indication otherwise (see below) each TextDNA is chosen with a likelihood equal to 1/(the number of lines of TextDNA)- that is, it is chosen randomly from among all possible lines of TextDNA. If you want to increase a line's likelihood of being randomly chosen, you can simply insert multiple copies of that line into the TextDNA file, or use the 'UseTextDNA' or 'RepriseTextDNA' functions described below.

The optional subject marker that must be the second element of a TextDNA if it appears lets your JanusNode know which subjects the TextDNA is to be filed under. The syntax for this marker is the word 'subject', followed by a list, in parentheses, of all subjects, separated by commas. There may be no spaces anywhere in the marker; if there is, an error will be flagged. For example, the marker 'subject(love,life,My_Favourite_TextDNAs)' signals to a JanusNode that the current TextDNA is to be chosen if the current subject is 'love', 'life', or 'My_Favourite_TextDNAs'. There are no pre-set subjects and no limits on how many subjects a TextDNA can belong to. You may classify any line of TextDNA under any subjects you like. When a JanusNode activates its TextDNA, it automatically

indexes every subject it finds.

A simple TextDNA example

An example of a TextDNA will make the simple syntax clear. Consider the following line of TextDNA:

```
subject(simple_TextDNA,MyTextDNA) s_articles 20 adjectives 5
s_nouns s_verbsnob punctuate 70
```

This line will be classified under two subjects: 'simple_TextDNA' and 'MyTextDNA'. You will learn why this is useful below. The TextDNA insert a singular article 20% of the time, followed 5% of the time by an adjective. The next two elements of the TextDNA, a singular noun and a verb which needs no object, will always be inserted when the TextDNA is chosen, since they are both given a 100% chance of being inserted. Finally, this sample sentence will be punctuated, as described earlier, 70% of the time.

One possible outcome of this TextDNA is the creation of the sentence:

The laughing eye twinkles.

Note that it is almost always necessary to have some elements of a line of TextDNA inserted with 100% probability, in order to guarantee that there will be a basic syntactically sensible structure underlying the sentence that is generated. One might, on the other hand, might go in for really 'modern' poetry that does not even follow the rules of syntax.

Further tutorials are contained Appendix 2 of this document, which reproduces the 'Hello World' TextDNA file that was created, by public request, for teaching purposes.

TextDNA control characters

The rest of the characters that are allowed in TextDNA allow for greater flexibility and control in TextDNA writing.

Repetition Brackets [ ]

Square brackets are for repeating certain parts of the TextDNA. Everything contained within square brackets will be repeated a random number of times (but not less than once and not more than five times). For example, consider the following line of TextDNA:

```
"You" [ s_verbsnon adverbs ","]
```

The first printable item- the word 'You'- will be printed every time the TextDNA fires. The phrase within the square brackets- a verb followed by an adverb and a comma- will be printed at least once, and up to five times. One possible outcome of this TextDNA is the creation of the phrase:

You run madly, dream crazily, hope lovingly,

Choice brackets {}

The curly brackets allow for alternative choices to be inserted in TextDNAs. Alternatives are separated with the 'I' character. When a TextDNA containing alternatives fires, JanusNode will randomly choose one the alternatives. You can put as many alternatives as you like.

Once again, an example will make the simple idea clear. Consider this line of TextDNA:

```
"You" { "dream" adverbs | s_verbs_to "me" }
```

The first item- the word 'You'- will be printed every time the TextDNA fires. After that, one of the two choices between the curly brackets will be randomly chosen. Either the word 'dream' will be printed, followed by an adverb, or else a verb taking 'to' will be printed, followed by the word 'me'. Two possible outcomes of this TextDNA are equally likely. The TextDNA will either print a sentence like:

You dream quietly

or it will print a sentence like

You give to me.

Function brackets and built-in functions <>

The triangular brackets (‘<‘ and ‘>‘) mark off function calls. JanusNode expects to see a function call between the brackets, followed by percent probability of calling that function. If the function is called, JanusNode will print whatever the function call returns. For example, you can make a call to a random number generator in the middle of the TextDNA. A TextDNA which does so might look like this:

"Love me" < randomNumber(20) > "times"

This TextDNA would improve upon Jim Morrison’s famous line "Love me two times" by substituting a random number between 0 and 20 for the word ‘two’. One possible outcome would be:

Love me 15 times

The purpose of the function call feature is to give you access to a number of useful functions that have been built in to your JanusNode. Most of these functions do not currently error-check their arguments very well (though JanusNode 3.0 is the most graceful version yet at recovering from errors), so pay close attention to the syntax.

The ‘Assign’ & ‘Get’ Functions

Two very important functions built in to JanusNode are the ‘assign’ and ‘get’ functions. These allow for the setting and accessing of global variables within a string of TextDNA.

The ‘assign’ function usually takes two arguments: a name for the global variable and one of the BrainFood file names. It names a randomly chosen line from the file with the name. In doing so, it returns nothing. However, the second argument of the ‘assign’ function need not necessarily be the name of a BrainFood file. If it is a quoted string of words separated by commas (but *containing no spaces*), ‘assign’ will randomly assign one of the words in that string to the global variable. For example, the following is legal:

< assign(AnAnimal,"dog,cat,kitten,cow") >

After this is encountered, the global variable ‘AnAnimal’ will have as a value either ‘dog’,’cat’,’kitten’, or ‘cow’.

The ‘get’ function allows your JanusNode to access variables that have been assigned a value using the ‘assign’ function. It takes a single argument, which should be the name of a previously assigned variable. It returns the value of that variable. If the name is not the name of a previously assigned variable, then the function does not return anything. Consider the following rather inane example:

```
< assign(MyNoun,s_nouns) > < assign(MyArticle,s_articles) > < get(MyArticle) > < get(MyNoun) > "is not" < get(MyArticle) > < get(MyNoun) >
```

The first two function calls are assignment statements that will not result in anything being printed. The first assignment `< assign(MyNoun,s_nouns) >` assigns a singular noun to the variable named ‘MyNoun’. The second assignment `< assign(MyArticle,s_articles) >` assigns a singular article to the variable named ‘MyArticle’. These two variables are then accessed twice in the body of the TextDNA, sandwiching the phrase "is not". One possible outcome of this TextDNA is the creation of the sentence:

my poem is not my poem

The LoadTextDNAFile and UseTextDNA Functions: Power to structure text

Two other powerful functions in your JanusNode are the ‘LoadTextDNAFile’ and ‘UseTextDNA’ functions. They simple but useful. Warning: these are ‘power user’ functions, which require that you have some idea of what you are doing. If you are just getting started with your JanusNode, you may want to leave these two functions until you understand how everything else works. Things will get very hairy if you try to use these functions without having a good idea of how your JanusNode works.

‘***LoadTextDNAFile***’ takes a single argument: the name (in quotation marks to be safe, although they are not necessary if your TextDNAs files contain only alphabetic and numeric characters) of a

TextDNA file in the TextDNAs folder. It loads that file in to the JanusNode.

Example:

```
< LoadTextDNAFile("WriteANovel.DNA") >
```

This will load and prepare the TextDNA file called "WriteANovel.DNA" (which unfortunately does not really exist). This allows you to move between TextDNA files during run-time. In concert with the 'UseTextDNA' function (described next), the 'LoadTextDNAFile' function makes it possible to make your JanusNode generate long, highly structured, coherent texts, which may use literally thousands of TextDNAs and BrainFood files in a systematic way.

The '***UseTextDNA***' function takes a single argument, which is the name of a subject by which at least one line of TextDNA in the current TextDNA set is classified. It simply sets the current subject to the specified subject, thereby ensuring that next TextDNA chosen will come from that set. Having this capability makes it possible to string together sets of TextDNA (or a single line of TextDNA, since a set can consist of just one line) one after another, thereby gaining complete control over the order on which your TextDNA fires. This makes it possible for your JanusNode to do many things that would otherwise be tricky or impossible, such as writing rhyming verses with a repetitive structure, writing long coherent narratives, and much more. Note that the rule specified by the 'UseTextDNA' function will not be called until the entire current rule is finished: you cannot 'hop out' of a currently-executing TextDNA rule. An example of the syntax of this function is

```
< UseTextDNA(MyTextDNA) >
```

Whenever this function call is encountered in TextDNA, the next line of TextDNA will be drawn from all the lines of TextDNA that are classified under the subject 'MyTextDNA'. This is the reason why you can classify rules.

Prior to JanusNode 3.0, calls to 'UseTextDNA' would switch to the called line, halting further processing of the line that made the call. A major improvement in JanusNode 3.0 is to allow

'UseTextDNA' to be inserted in the middle of other textDNA. For example, consider these two lines:

```
Subject(MyFirstLine) "My" < UseTextDNA(MySecondLine) >  "is a" <
UseTextDNA(MySecondLine) > "."
***
Subject(MySecondLine) { "dog" | "frog" | "cat" }
```

This will print sentences like 'My dog is a cat.' or 'My frog is a frog.'

Note that here the literals have been quoted; this is not necessary, but it is good practice.

JanusNode 3.0 uses a 'drop through' method for deciding how to decode a string: that is, it systematically searches in several places for a match to the string, and uses the string as it matches. The first place it looks is for TextDNA subjects in the same file as the current line. If it finds one, it automatically calls 'UseTextDNA': this means that in fact the function call is not required. The following lines are functionally identical to the two above:

```
Subject(MyFirstLine) "My" MySecondLine  "is a" MySecondLine "."
***
Subject(MySecondLine) { "dog" | "frog" | "cat" }
```

When JanusNode encounters an unquoted string, it looks to see if can find it a TextDNA match, and calls the UseTextDNA function if it finds one. It also looks for other possible uses of the string, which are described below. If it finds no other use, it will print the string itself. Quoted strings are always printed verbatim.

The RepriseTextDNA Function

RepriseTextDNA is another powerful meta-level control function, which allows the 'UseTextDNA' function to use indirect reference. The function takes a single argument that is the name

of a variable and fires a line of TextDNA that has the same name as *the value of* that variable. You will usually set the value of the argument that is passed to RepriseTextDNA using the 'Assign' function. For example:

```
< assign(CurrentDNA,"MyTextDNA") >
*** The above line will assign the value 'MyTextDNA' to
*** the variable 'CurrentDNA'
< RepriseTextDNA(CurrentDNA) >
*** The above line will fire a (randomly-selected) line of
*** TextDNA which has the subject 'MyTextDNA',
*** which is the _value_ of the variable 'CurrentDNA'.
```

RepriseTextDNA makes it possible to have repeated elements in an output text (for example, choruses in a song- see the RobotJohnson files for many examples) and to have a logical flow to texts, since you can set the values of future text in TextDNA that has fired. For example, you might have two rules with the same name that lead to different paths using the RepriseTextDNA function, as follows:

```
Subject(HeroineEnd) "And then she died." <
RepriseTextDNA(DeadHeroineDNA) >
***
Subject(HeroineEnd) "And then she married the handsome prince."
< RepriseTextDNA(PrinceIsMarriedDNA) >
```

In this case, prior rules might call either one of these two rules using the UseTextDNA function, since they both have the same subject, 'HeroineEnd'. However, clearly things are likely to develop differently if the heroine dies than if she marries the beautiful prince. The DeadHeroineDNA variable contains the name of one set of TextDNA which continues (or completes) the text with a dead heroine, while the PrinceIsMarriedDNA contains the names of a set of TextDNA which continues the text with marriage to a handsome prince. If there were only a single relevant rule-set in either case you could just use the UseTextDNA function- the RepriseTextDNA function would be used in this case if (as in real life!) there were multiple possible paths through life.

The GetRhyme function

The GetRhyme function takes two arguments: a variable name, and a suffix. It returns a randomly-selected entry from a BrainFood file that has the name of *the value of* the variable, plus the suffix. Although it may prove useful in many different circumstances, its main purpose is to make it possible to write rhyming verse that uses any one of a number of possible rhymes. If you define a number of files 'x.verb', 'x.noun', 'x.exclamation' (where 'x' takes multiple values- i.e. 'dog', 'cat', 'pig') then you can always call a word which rhymes in the proper place, without always writing poems which use the same rhyme. For example:

```
Subject(AnimalPoemSetUp) < assign(Cur,"dog,cat,pig") >
***
Subject(ShortAnimalPoem) "A" < GetRhyme(Cur,noun) >
"likes to" < GetRhyme(Cur,verb) > "!"
```

The first line of TextDNA sets the value of the variable 'cur' to either 'dog', 'cat', or 'pig'. Assuming the existence of the appropriate BrainFood files (which, incidentally, do not actually exist- you'd have to make them to run this example), the second line writes a short poem alleging that the animal in question likes to do something that rhymes with its name. One possible outcome of firing these two lines of TextDNA might be:

A pig likes to jig!

However, it is equally possible that the identical lines might produce:

A cat likes to bat!

The 'GetSubject' Functions

The 'GetSubject' function can be used to personalize your JanusNode's creations with a single name and matching pronoun and possessive. The function usually takes one of four arguments: 'name', 'pronoun', 'possessive', or 'object'. It will return the correct name, pronoun, possessive, or object. The 'GetSubject' function can also be used to *set* the name or sex of the current subject. If the argument is 'he', then the gender of the subject will be set to male. If the argument is 'she', then the gender if the argument will be set to female. If the argument is anything else, then the name of the subject will be set

to the argument. In all three of these latter cases, nothing is returned.

Consider the following example:

```
< getSubject(Jane) > < getSubject(name) > s_verbs_from < getSubject(possessive) > s_nouns
```

This TextDNA will always change the subject's name to 'Jane', since the first call to the 'getSubject' function has 'Jane' as an argument, and is guaranteed to fire. After that, it will print the new name, 'Jane', since the second call has 'name' as an argument, and is also guaranteed to fire. This will be followed by a verb which takes the word 'from', a possessive, and a noun. On possible outcome of thus TextDNA is the production of the phrase:

Jane steals from her dog

However, note that this snippet of TextDNA is just as likely to print:

Jane steals from his dog

It will do so if the last subject was a male, since there is nothing in this TextDNA that sets the sex of 'Jane' - and a JanusNode is far too dumb to figure things like that out for itself. We could have remedied this by adding the function call '< getSubject(she) >' anywhere before the call '< getSubject(possessive) >'. This would set the subject.

The 'backspace' function

The 'backspace' function exists to allow for the suppression of the space that a JanusNode normally inserts between words: i.e. it deletes the last character that was output, and prints from the new end. There are many situations in which this might be useful- for example, if you want to write a string of TextDNA which uses parentheses, or which creates compound words, or if you want to keep sentences in paragraphs. The function takes no arguments and returns nothing.

Example:

```
"Imagine a" s_nouns < backspace() > "-" < backspace()> s_nouns
```

This string of TextDNA will ask you to imagine a new compound noun. For example, it might print:

Imagine a computer-coffin

You may feel free to commercially develop this idea.

'Backspace' has one counter-intuitive (but rather useful) behavior, which is that it will erase as many spaces and returns as it finds. It will never erase more than one single non-whitespace character, and will erase even a single one only if the last character printed happens to be non-whitespace.

The 'quotation' function

The quotation function allows the user to use (straight or curly) quotation marks within a string of TextDNA, so that your random texts can cite other random texts. It takes one of two arguments, "o" or "c", or no argument. If the argument is "o" the function will return an opening curly quote. If it is "c" the argument will return a closing curly quote. If there is no argument, the function returns a straight quote.

Example:

```
"Try to understand this sentence: " < quotation()> TextDemonSPVP TextDemonPNP < quotation()>
```

This TextDNA makes use of *TextDemons*, which are defined in the following section of this documentation. This string of TextDNA will print quote a verb phrase and a noun phrase following a verbatim introductory phrase. One outcome of the TextDNA is the creation of the phrase:

Try to understand this sentence: "too many of us accommodate temptations".

Profound, huh?

The 'UseFont', 'UseStyle', and 'UseSize' Functions

You can also use built-in functions to control (or randomize) some aspects of text presentation

from with TextDNAs, using the ‘Usefont’, ‘UseStyle’, and ‘UseSize’ functions. They all have the same syntax: each one takes either a valid argument for what it does (e.g. a font name, a font style, or a font size respectively) and sets the current display font to that size, style, and font. All three arguments may also take the argument ‘random’, in which case the relevant value is set randomly.

Legal font styles are ‘bold’, ‘italic’, ‘extend’, ‘underline’, ‘outline’ and ‘plain’. Legal font names and sizes are system-dependent; in general, you can usually use font sizes between about 10 and 127. An example of a TextDNA using these functions is:

```
< UseFont(chicago) > <UseSize(14) > "This is" < UseSize(24) > < UseStyle(italic) > "really" < UseSize(14) > < UseStyle(plain) > "dumb TextDNA."
```

This self-referential string of TextDNA will print the sentence ‘This is really dumb TextDNA.’ just like that, in plain 14-point Chicago type, except for the word ‘really’, which will appear in italic 24-point Chicago.

The ‘CapitalizeNext’ Function

The ‘CapitalizeNext’ function takes no arguments. It simply ensures that the item that follows it will start with a capital letter.

Example:

```
"This is a name:" < CapitalizeNext() > syllables < backspace() > syllables 70
```

This string of TextDNA will print out "This is a name:" followed by a one or two-syllable nonsense word that starts with a capital letter. One possible outcome is:

This is a name: Tishyag.

The ‘BecomePassive’ Function

The ‘BecomePassive’ function takes no arguments. When it is called, processing stops and your

JanusNode enters a passive, receptive state. It is important to understand that a JanusNode must fire all functions before it can print a line of TextDNA, since most functions are intended to return printable components of that line. For this reason, the 'BecomePassive' function should not appear with any text that you wish to print. If it does, that text will never get printed, since your JanusNode will become passive before it gets to printing it. Here is an example of a full line of textDNA:

```
Subject(End) < BecomePassive()>
```

When this line fires, processing will halt.

Limitations

JanusNode's ancestors had many limits. These are all eliminated in the 21st century. There is no limit to the quantity of TextDNA you can define, since you can have as many TextDNA files as you wish, and you can have TextDNA files as large as you wish. Since TextDNA can call specific lines of TextDNA, even when they are in another file, there is therefore no theoretical limit to how complex your production's connections and structure can be.

Error-checking

JanusNodes do not stop processing for most syntax errors. If it encounters an error, a JanusNode will print an error message- or simply the name of the element it could not find- in its output field and will attempt to keep processing. It can almost always recover from such errors, but sometimes it will print several error messages before it recovers.

Nesting TextDNA Elements

You can nest repeats, functions, and alternatives within each other. For example, the following quite stupid rule works fine:

```
Subject() [ { "Nouns:" [ s_nouns] | "Verbs:" [ s_verbsnob] | "A" people s_verbsnob [ < random(100) > "," ] "or more times." | return 20 } ] return
```

This rule is equally likely to produce output such as:

> Nouns: optimum cad winter A product engineer tries 49, 7, 40, or more times. A photoengraver calms 14, 46, 29, or more times. Verbs: personalizes gives

or

> Nouns: apocalypse flash

This allows for some very short rules to behave in complex ways, opening up some fairly wacky text-generation possibilities. Go wild.

TextDemons

TextDemons are very similar to TextDNA, which is not surprising because, in fact, they *are* TextDNA. The only difference is that TextDemons are defined in the 'TextDemons' file, which is contained in the 'TextDNA' directory. They have an identical format to TextDNA.

Just like TextDNA within a file TextDemons do not need to be explicitly flagged as TextDemons. JanusNode uses the same 'drop through' technology for TextDemons. If it is looking at an item in a rule, and it cannot find a BrainFood file or a TextDNA rule in the file that corresponds to that item, a JanusNode assumes the item must be a TextDemon, and looks in the TextDemons file for the item, no matter what its name.

For example, we might define a TextDemon in the TextDemons file as follows:

```
Subject(MyFirstDemon) p_nouns p_verbsnob [ adverbs 50 ]
```

We can then use this TextDemon in a line of TextDNA, defined in a TextDNA file, as follows:

```
TextDemonMyFirstTextDemon "and" p_verbsnob
```

The TextDemon defines a noun-verb phrase (possibly modified with one or more adverbs). The above

TextDNA might generate the sentence:

time changes warmly nightly and distorts

You can use TextDemons to define constants, to define commonly-used text chunks, and to gain finer control over the way units are repeated or chosen.

Every JanusNode ships with many pre-written TextDemons in the TextDemons file. Since they are subject to change and open for tinkering by any user, they are not documented officially. You will have to take a look at them if you want to see what they do. However, you should be careful, since theTextDNA that is distributed with your JanusNode depends on the TextDemons that came with it. You are welcome to delete all the TextDemons that come with your JanusNode, but do not do so unless you understand what you are doing, as much of your TextDNA will be non-functional without the appropriate TextDemons.

Hello world!

By popular demand, JanusNodes ships with a highly-commented tutorial set of TextDNA, the HelloWorld rules. These move from the simplest to the more complex TextDNA in a structured way. The rules are installed in your TextDNA directory, and so may be run in the usual way. They are also appended to this document as Appendix 2.

Markov Chaining

Probabilistically re-creating texts using Markov chaining is a JanusNode's second major function. The idea behind Markov chaining is very simple: for every element of a text, compute the likelihood that any element is followed by any other element, then reconstruct the text in a way which reflects the real probabilities, by stepping through the probability table. For example, in the string 'ababca' the likelihood that 'b' will follow 'a' is 100% while the likelihood that 'a' will follow 'b' is

just 50%, since ‘b’ is followed exactly once by ‘a’ and is also followed once by the letter ‘c’. One probabilistic reconstruction of this string might be ‘bcabcaba’, since this string has (roughly) the same statistical structure as the original one.

Any ordered sequence of elements can be Markov chained. Your JanusNode will Markov chain any text. You can easily make and use your own Markov chain tables, as described below. You can average together as many different texts as you like.

There are two steps to using Markov chaining.

The first is making the probability tables that describe a text. If you want to be able to use a Markov table, it must be stored in the ‘MarkovTables’ folder inside the ‘JanusNode Resources’ folder. JanusNode uses the same tables for both single (loose) and pairwise (tight) Markov chaining, and it only looks in this folder for them. Although the tables are editable text files, it is a bad idea to alter them in any way unless you are quite sure you understand what you are doing, as it is very easy to introduce errors into these tables that will render them useless. For example, you can’t just add or delete words as you like, as the table is full of dependencies.

The ‘Markov chain a file’ button will create a new Markov chain table (as a text file) from a text input file, and offer to store it in the proper directory for you. ‘Markov chain output window’ does the same thing, but it uses the text in the output window as the input.

You will be offered the choice of chaining by word or letter. If you choose to chain by word, JanusNode will use individual words as the elements in its probability tables. In that case, the output will contain roughly the same words as the input file, in a probabilistically-related order. If you choose to chain by letter, JanusNode will use character strings, of length at least one, to make its tables. In that case, the created probability table does not necessary reflect word boundaries. The output will be more or less word-like, but perhaps contain some or many strings that are not actually words, but which could be. You can choose how big the character strings should be. If you choose a small number (1 or

2), the resultant output will be very unnatural, especially if you chain the output loosely (see below). If you choose a larger number (say, 3 to 5), the output will be more sensible, containing some coherent phrases sprinkled with some odd constructions that are almost words. If you choose a larger number, it will be increasingly unlikely that there is any way to reconstruct the text except the way it actually was, and eventually you will be getting back exactly the text that was used as input.

There is no limit to how big the input text or its associated Markov table can be, but it does take a long time- sometimes hours- to make the Markov table if the input text is very large, especially if you are chaining by character strings. JanusNode ships with an odd assortment of large Markov chain tables, including the tables for the Tao Te Ching, James Joyce's 'Ulysses' (a huge table) and Ludwig Wittgenstein's 'Philosophical Investigations' (one of my favorite books). To use one or more of the pre-calculated tables stored in the 'MarkovTables' folder, click on the 'Write from Markov files' button. You will be offered a choice of 'tight' or 'loose' writing. 'Loose' means that any word pair is guaranteed to contain words or strings that actually appeared together in that order in the original text. 'Tight' means that any *three* contiguous words or strings are guaranteed to have actually appeared together in that order in the original text. Tight writing is much more sensible, but also much more plagiaristic, and may contain long passages cited verbatim from the input file. The 'Write loosely' function is your best bet for generating original output, since the looseness of the algorithm makes it relatively unlikely that any of the lines in the output appeared exactly in that form in the input file.

You can select as many tables as you like from the folder: use the command key to make a discontiguous selection. Your JanusNode will chain and randomly switch between input tables as it produces a text. You will see it trying to switch and switching back, if you look in the information window that appears when it is working. The output from using several tables will resemble all of the input files (ie. the files from which the tables were made) to some degree- it is something akin to a statistical average of those files. You can of course also average different texts by putting them all into

a single file (or pasting them into the Output window), and building (and then writing from) a Markov chain table for that file.

Although can certainly mix and match Markov chain files that were created by letter mapping, there is no point to mixing files that specified a different string length, because there is no way for the jump between files to made. If you want to mix different texts chained by letter, you must create Markov table files that used the same letter length.

Text Mapping

There are many computer programs whose purpose is to turn a text into a specific dialect. These programs are usually extremely simple: they simply make a set of pre-defined substitutions to the text. Your JanusNode has this ability, which can be completely configured by the user. The 'Text Mapping' command uses information stored in files inside the 'Mappings' folder (in the 'JanusNode Resources' folder) to make substitutions to the text in the JanusNode window.

The mappings have their own simple grammar. In general, each line must have at least two and at most three elements, separated by commas. The second element will be substituted for the first element with a probability equal to the third element, if there is a third element. If there is not a third element, it is assumed to be 100%. Elements may be subword character strings, words, or multiword strings. The probabilities operate globally, not by item- so once a mapping 'passes' the probability test and is chosen to be applied, it will be applied to every item. For example, consider the following mapping:

you , thee, 20

This will be applied 20% of the time it is chosen (and every mapping will be chosen exactly once, in random order, when the tool is applied). When it is applied, it will replace the word 'you' with 'thee'. Note the space inserted after the word 'you'- this is to ensure that the mapping is only applied when the

whole word is ‘you’- so, for example, the word ‘your’ will not be changed to ‘theer’. It *would* be so changed if there were no space after the first word. If you leave a space after the first element, there is no need to also leave one after the second element: JanusNode can figure this much out for itself. Your JanusNode also deals by itself with the complications of capitalization and punctuation of various kinds, so that it will recognize that the word ‘You’, ‘you.’, or ‘you)’ (and so on) should be replaced in the above example with ‘Thee’, ‘thee.’ and ‘thee)’.

Along with such two- or three-element substitutions, there are two other allowable forms that may appear in a mapping: comments, and random exclamations. A comment is any line beginning with an ‘*’: it will simply be ignored when encountered, allowing you to insert notes into your mappings. A random exclamation has the form ‘random(X)’ (optionally followed by a percent probability), where ‘X’ is some text. When JanusNode sees a line like this in a mapping file it will (if it passes the optional probability test) randomly insert the text contained between the parentheses after the end of a random number (one or more) of randomly-selected sentences in the text. You can use this to add a little spice to your dialects.

JanusNode comes with a variety of mapping files that should serve as further examples, and will hopefully make the idea clear.

Other tools

JanusNodes includes a variety of other tools for working with texts. In this section we describe those tools.

eecummingsfication

The ‘eecummingsfy’ button will attempt to mimic the style of the great poet ee cummings, using the available text in the output window. Text which has been eecummingsfied tends to function ‘more poetically’ than text which has not been so treated. Like all the randomization tools,

eecummingsfication works probabilistically, so treating the same text twice will not give precisely the same result.

The eecummingsfication process is user-configurable. It uses three files in the 'eecummings' folder, which is inside the 'JanusNode Resources' folder. You can add items to and delete items from these files to customize the way eecummingsfication functions. The process works by looking for subword elements which can be interestingly 'isolated' from their context. The file 'EndCuts' contains strings that may possibly be isolated from the front if they appear in the text. (Since the tools apply by chance, there is no guarantee that any isolation will actually be made.) For example, if the word 'be' appears in the 'FrontCuts' file, then the word 'babe' might be split into 'ba' and return & 'be' when the tool is applied. Here the word 'be' is isolated from the front. The file 'EndCuts' contains strings that may (probabilistically) be isolated from the end if they appear in the text. For example, if the word 'be' also appears in the 'EndCuts' file, then the word 'bear' might be split into 'be' and return & 'ar' when the tool is applied. Here the word 'be' is isolated from the end. Note that such isolation would not occur from the appearance of the word 'be' in the 'FrontCuts' file. The 'FrameMe' file contains strings that will be isolated from both sides at once' If 'be' appears in that file, then the word 'unbearable' might be split up as 'un', return, 'be', return, and 'arable'- with 'be' isolated (= 'framed') from both sides at once.

Dadafication

The 'Dadaify' button will randomly choose words from the original text and print them in a randomly-arrayed manner. Those of you who are educated in Dadaism (which is sadly lacking on many curriculums these days) may recognize this as the original formula for producing Dadaist poetry, as conceived of by the patron saint of JanusNodes, Tristan Tzara: "And here you are a writer, infinitely original and endowed with a sensibility that is charming though beyond the understanding of the

vulgar".

WebPoem

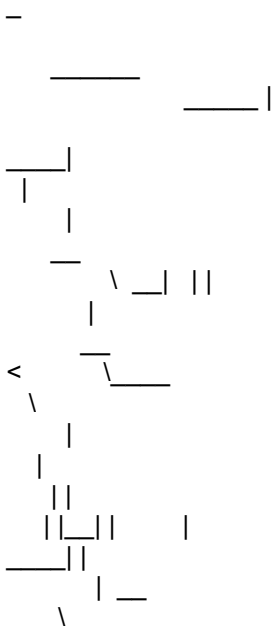
WebPoetry was added to JanusNode only recently, in August, 2012. Assuming your computer is connected to the Internet so it can access web pages, the WebPoem button will produce a text that is based on one or more Wikipedia entries or other web text. Multiple Wikipedia subjects must be separated by commas, as in the default setting, which uses Wikipedia articles about several of JanusNode's apparent obsessions:

love, pleasure, epistemology, quantum field theory

You can also specify a specific URL (which must include "http://") or use the word 'random' or 'randomURL' as a subject, repeated as many times as you like. If you specify a specific URL, JanusNode will use text from that web page. Each time 'random' is specified as a subject, JanusNode will access a random Wikipedia article. The use of this keyword may require patience, however, because many articles in Wikipedia are very short, and short articles lead to very repetitive output. Also, a lot of Wikipedia articles are poetically dull. Each time 'randomURL' is specified as a subject, JanusNode will access the text from a totally random web page, not necessarily (in fact, almost certainly not) from Wikipedia. This is even more dicey than taking a random page from Wikipedia. It often returns an error saying you have no Internet connection, because many random pages turn out not to be accessible. You may also end up on a page with a very huge amount of text that will take a while to download or on a page that contains no English text or no text at all or only HTML or on a page whose subject matter is offensive to you. Finally, you may get depressed when you discover how boring most web pages are.

If you keep trying, you may get lucky, in this as in so many other endeavors in life. I recently got this very fine poem, which I found totally surprising and charming:

| |____ | __|



The WebPoem generation method is an entirely new method that combines elements of Dadaism and quasi-Markov-chaining. It results in a text that is generally quasi-coherent, especially over short stretches, but also plagiaristic over short stretches. Short runs of no more than five consecutive words may be taken verbatim from the input and those runs could theoretically be extended by the chaining. If you get something brilliant in the output, you might want to check it wasn't in the input text, especially if the input was short. You can find the entire last used input text in the file 'JanusNode Resources/WebPoem.Temp/Last sources.txt'.

JanusNode caches Wikipedia articles and other named web pages within each session (i.e. until you quit the program) so specifying a text is much faster after the first time, when it has to be downloaded from the web.

[JanusNode Hacker Note: Since JanusNode looks in the folder 'WebPoem.Temp' to see if it has a cached file on the current topic, you can put any text file you want in that folder, and 'WebPoem' it by listing that file's title as a WebPoem topic. However, don't forget that JanusNode erases all the WebPoem.Temp files (except 'Last sources.txt') on exit, so make sure you keep a copy of the file if you don't want it to be erased.]

Automatic TextDNA Generation

The 'Make TextDNA' button will attempt to turn any text in the JanusNode output window into an executable line of TextDNA. If you are too lazy to write TextDNA, you can simply write (or import) a sentence (or more) of the form that you would like to produce, and let this function translate that sentence into TextDNA. The TextDNA produced can then be used like any other line of TextDNA, if you paste it into a TextDNA file. The function can only work if you use words in your template that appear in your JanusNode's BrainFood files. You will be asked to select a subset of files to use. Each recognized word in the output window text will be replaced with a call to a global variable that is set to a word from the same file as the recognized word. If you use long texts, you may wish to break them up afterwards into smaller rules connected by 'ChooseTextDNA' calls.

After you have generated a rule, you will be offered the option to run it right away. If you accept this, your JanusNode will enter text-generation mode, and write text after the current rule in the output window.

The rule-generating tool is not perfect. Because it blindly replaces words that it finds in the order that it searches, it sometimes makes errors in deciding which Brain Food file a word should come from. It has turned out to be (to me) surprisingly difficult to generate rules that work perfectly every time, and the 'Make textDNA' function does occasionally produce TextDNA that contains (usually very minor) errors. However, it most often generates TextDNA which is either useable as it is, or in need of only minimal repairs. It's a good starting point, but not a good endpoint.

The 'Show File Loads' radio button

The 'Show file loads' button toggles whether or not JanusNode will print which rule files it has opened. Usually this is undesirable, so the default is to have this button off. It can be useful to know which file has been most recently loaded when debugging complex rule sets.

The 'Alliterate' radio button

The 'Alliterate' radio button turns on JanusNode's alliteration function (which can also be controlled dynamically by a function call- see above). You will be given two options when you turn it on: 'Selected' or 'Random'. If you choose 'Selected' you must state a character (or longer string, if you like), and JanusNode will always choose a word that begins with that string when it can. If you choose 'Random', then JanusNode will pick a new character randomly whenever it chooses a new line of TextDNA. In either case, JanusNode will use a word that begins with the alliteration character whenever it can find one in any BrainFood file it uses. If it cannot find a word with the current character, it will randomly use a word from the same BrainFood file, as usual. This ensures alliteration when possible, but maintains coherence (insofar as JanusNode is ever coherent) if alliteration is not possible.

The 'Bökify' radio button

The Bökify button is named after Canadian poet Christian Bök, who inspired the feature by writing long poems (in his 2001 book *Eunoia*) with the constraint that every word had to contain a given letter. Bökification allows the user to choose letters that should (or should not) appear in the selected words in a poem. Letters that appear before a backslash will appear in every word, when a suitable word can be found. Letters that appear after a backslash will not appear in any word, whenever possible. For example, if you enter 'xz/e' then the JanusNode will try to choose words that contain 'x' or 'z' and that do not contain 'e'. With this constraint, my JanusNode produced:

Max fluxes a zinnia upon arizona axes!

[The 'e's in 'axes' and 'fluxes' were added during pluralization/conjugation, and bökification happens at word selection time, before these functions are called.] As with alliteration, if your JanusNode cannot find a word that fits the constraints of Bökification, your JanusNode will randomly use a word

from the same BrainFood file, in the usual manner.

Random Bökification will randomly change the Bökification constraint every line.

Bökification and alliteration will work together, but Bökification and assonance (below) are mutually exclusive and cannot both be used simultaneously.

The 'Assonanciate' radio button

The 'Assonanciate' radio button turns on JanusNode's assonance/consonance function. This forces the JanusNode to try to use only lines of BrainFood that contain a particular (specified or random) substring. For example, with the assonanciation constraint 'ant', a JanusNode might write (as mine did):

Migrant workers tantalize an antipathy.

['Migrant workers' is a single line of BrainFood, so its selection meets the constraint.] When it is randomized, the constraining string will change every few lines.

Note that assonance/consonance with more than a character or two is such a strict constraint that it may often fail except when used with very large word sets. As usual, when it fails JanusNode will relax the constraint and choose a random word. It functions best when used with 'All words' (a rule set that draws from a large dictionary of mono-morphemic words that is not organized into word types) but you pay a price in syntactic coherence when you use this rule set: you will have to read a lot of non-sensical output to find anything interesting. When used with many BrainFood files, it may produce quite repetitive output because there are not many words that meet the constraint.

Assonanciation cannot be used with any other automated word-selection constraint: when turned on, it trumps alliteration and bökification.

The 'Be extra-brilliant!' radio button

This is a placebo. It doesn't actually have any effect but you can probably fool yourself into

believing that it does, and that will make you happy.

The Robot Johnson Project

Among other goodies, JanusNode ships with rules for generating blues songs in the style of the great blues singer, Robert Johnson. These rules are fairly complex but not great. I would like it if someone else wrote some better Robot Johnson rule files! Maybe you could do that.

Resources

I release new resources for JanusNode (mapping files, rule files, Markov files) as I receive or write them. They are available at janusnode.com.

Bug Reports

If you find any bugs within JanusNode, let me know, at janus@janusnode.com. I will try to fix outright bugs as rapidly as possible. I will add new functionality and bring the rules up to snuff when I am able.

If you have questions or comments about using your JanusNode or writing TextDNA, feel free to contact me. I enjoy hearing from users.

If you would like to help move JanusNode into the future, please think about contributing your rule and other resource sets, which are what makes a JanusNode do its magic. The more, the merrier.

I have been supporting JanusNode- and its predecessor McPoet- for nearly 30 years now. Throughout that time, my main motivation for improvement has always been *user feedback*. If I know people are using this program, I work on it. If I forget that anyone else is using it, I don't work on it. My thanks to all those who have gently reminded and inspired me to keep on it.

And a special thanks to those who have bought me a beer or four using JanusNode's 'donate' button!

JanusNode is on Twitter: @janusnode.

To quote another patron saint of Janus, Marcel Duchamp:

> "Have fun, if not you'll bore us."

Chris Westbury
Edmonton, Canada
August, 2012

Appendix 1: Quotes related to Janus

"Every day we slaughter our finest impulses. That is why we get a heartache when we read those lines written by the hand of a master and recognize them as our own, as the tender shoots which we stifled because we lacked the faith to believe in our own powers, our own criterion of truth and beauty. Every man, when he gets quiet, when he becomes desperately honest with himself, is capable of uttering profound truths. We all derive from the same source. There is no mystery about the origins of things. We are all part of creation, all kings, all poets, all musicians; we have only to open up, only to discover what is already there."

Henry Miller
Sexus

"If things are not clear, do nothing."

Gerald Loeb
The Battle For Investment Survival

"Stupidity well packaged can sound like wisdom."

Burton Malkiel
A Random Walk Down Wall Street

"How can men take joy in nonsense? They do so, wherever there is laughter- in fact, one can almost say that wherever there is happiness there is joy in nonsense. It gives us pleasure to turn experience into its opposite, to turn purposefulness into purposelessness, necessity into arbitrariness, in such a way that the process does no harm and is performed simply out of high spirits. For it frees us momentarily from the forces of necessity, purposefulness, and experience, in which we usually see our merciless masters. We can laugh and play when the unexpected (which usually frightens us and makes us tense) is discharged without doing harm. It is the slaves' joy at Saturnalia."

Friedrich Nietzsche
Human, All Too Human

"It takes two to invent anything. The one makes up combinations; the other chooses, recognizes what he wishes and what is important to him in the mass of things that the former has imparted to him. What we call genius is much less the work of the first one than the readiness of the second one to grasp the value of what has been laid before him and to choose it. "

Paul Valery

"The poet is he who inspires, rather than he who is inspired."

Paul Eluard

"To make two bald statements: There's nothing sentimental about a machine, and: A poem is a small (or large) machine made of words...When a man makes a poem...[i]t isn't what he says that counts as a work of art, it's what he makes."

William Carlos Williams,
Introduction to *The Wedge*

"There was this kid poet, and he wrote and wrote. He rubbed the magic lamp until the poetic self-abuse police threatened to come impound him. And still nothing happened. The incantation seemed defective. Then they put the kid in front of this terminal and initiated him into the secret syntax. A few simple

rules, combined in a few elegant ways, and blamm-o. The thing works. It runs. the world does move. The rules churn. The descriptions step their way through their own internal logic. The lines of code set more switches, change more states. Commands produce results.

The word made flesh.

Spiegel flinched. *Don't mock me.*

I'm not mocking."

Richard Powers
Plowing The Dark

"....chance alone is the source of every innovation, of all creation in the biosphere. Pure chance, absolutely free but blind, at the very root of the stupendous edifice of evolution: this central concept of modern biology is no longer one among other possible or even conceivable hypotheses. It is today the sole conceivable hypothesis, the only one compatible with observed and tested fact. And nothing warrants the supposition (or the hope) that conceptions about this should, or ever could, be revised."

Jacques Monod
Chance & Necessity

"To chaos, law destroys; to law, chaos."

John Fowles
The Aristos

" ...'disorder, yes, my boy, disorder, is the quintessence of your very life! Of your whole physical and metaphysical being! Why, it's your very soul...millions, trillions of intricate folds...plunging deep down into the gray matter, complex, subjacent, evasive...limitless! That's Harmony...all nature! A flight into the imponderable! And nothing else! Put your wretched thoughts in order...! That's where to begin. Not with grotesque, material, negative, obscene substitutions, but with the essential, that's what I'm getting at. Are you going to assault the brain, correct it, scrape it, mutilate it, force it to comply with an assortment of stupid rules? carve it up geometrically? recompose it according to the rules of your excruciating idiocy?...Arrange it in slices? like an Epiphany cake? with a prize in the middle. Tell me that. I'm asking you. Frankly? Would that by any good? Would it make sense? Heaven help us! There's no doubt about it...your soul is overwhelmed by errors. It makes you, like so many others, a unanimous nonentity. Great instinctive disorder is the father of fertile thoughts! It's the beginning of everything...Once the propitious moment has passed there's no hope...You, I'm afraid, will spend your whole life in the garbage pail of reason...So much the worse for your! You're a numbskull...a nearsighted, blind, preposterous, deaf, one-armed dolt!...befouling my magnificent disorder with your vicious reflections. In Harmony...resides the worlds only joy! The only deliverance! The only truth!...Harmony! Find Harmony, that's the ticket!...Do you hear me...? Like a brain, neither more nor less! Order! Pah! Order! Rid men of that word, that thing. Accustom yourself to Harmony and Harmony will reward you. You'll find everything you've been looking for so long on the highways of the world...and far more! Many other things...! A brain...that's what the whole lot of you will find! Yes!...Have I made myself clear? That's not what you're after? You and your kind? An inane ambush of pigeonholes! A barricade of brochures! A house of the dead! A chartist necropolis! No, never! Here everything is in movement! Swarming with life! You're not satisfied! It stirs, it quivers! Just touch it! Put out your little finger. Everything comes to life. Everything trembles instantly! Asking only to surge up! to blossom! to shine! I don't live by destroying. I take life as it comes! Do you take me for a cannibal...? Never!...Bent on reducing it to my chickenshit concepts? Pah! Everything shakes? Everything topples? Splendid! I have no desire to count stars 1! 2! 3! 4! and 5! I'm not the kind that thinks he's entitled to do anything he pleases. The right to shrink! rectify! corrupt! prune!

transplant!...No!...where would I get it?...From the Infinite?...From life itself? It's not natural, my boy! It's not natural! It's infamous meddling!...I prefer to keep on good terms with the Universe! I take it as I find it!...I'll never rectify it! No! The Universe is master of its own house! I understand it! It understands me! It gives me a hand when I ask it! When I'm through with it, I drop it! That's the long and short of it...It's a cosmogonic question! I have no orders to give! You have no orders! He has no orders!...Bah! Bah! Bah!...' He got sore as hell, like somebody who's definitely in the wrong... "

Louis-Ferdinand Céline
Death On The Installment Plan

"There are only two things in the world - semantics and nothing."
Erhard Werner as quoted in: A. Bry
Est

"Drawing on my fine command of language, I said nothing. "
Robert Charles Benchley

"...Brahman is the cause of the many. There is no other cause. And yet Brahman is independent of the law of causation."
Shankara
Viveka-Chudamani

"As to what pertains to manifestation, the Principle causes the succession of its phases, but is not this succession. It is the author of causes and effects, but is not the causes and effects."
Chuang Tzu
The Book Of Chuang Tzu

"You cannot take hold of it, but equally you cannot get rid of it, And while you can do neither, it goes on its own way."
Yung-chia Ta-shih

"A 'bit' of information is definable as a difference which makes a difference. Such a difference, as it travels and undergoes successive transformation in a circuit, is an elementary idea."
Gregory Bateson
Steps To An Ecology Of Mind

"Because information does not inform unless it is received, it does not exist until it is consumed. It exists only in its assimilation and dies when it becomes redundant. Information is intrinsically sacrificial. What seemed mad and illogical in the old order of production becomes sane and logical in the new order of semiotic consumption. So, for example, the 'insanity' of sacrifice, of giving something for nothing, becomes the royal road to the sublime, not an altruistic act of self-denial."
James Ogilvy
Living Without A Goal

"A type has rightly come to be recognized as a mental realization with no bone and flesh embodiment;...the race becomes, as it were, a great amoeboid form, with its prepondering variations thrown out as pseudopodia feeling towards adaptation."
Arthur Keith
The modes of origin of the carotid and subclavian arteries from the arch of the aorta in

some of the higher primates, Journal of Anatomy & Physiology, 29:453-58

"Power is nothing if not the power to choose...there is all the difference between deciding and choosing...Perhaps every human act involves a chain of calculations of what a system engineer would call decision nodes. But the difference between a mechanical act and an authentically human one is that the latter terminates at a node whose decisive parameter is not 'Because you told me to' but 'Because I chose to'. "

Joseph Weizenbaum
Computer Power And Human Reason

"we must look at all acts of perception as acts of creativity. "

Gerald Edelman
How We Know
Nobel Conference, 1985

"...effective searching procedures become, when the search-space is sufficiently large, indistinguishable from true creativity."

Richard Dawkins As quoted in: Kevin Kelly
Out Of Control

"The program found in the head of the average poet, after all, was written by the poet's civilization, and that civilization was in turn programmed by the civilization which preceded it, and so on to the very Dawn Of Time, when those bits of information that concerned the poet-to-be were still swirling about in the primordial chaos of the cosmic deep. Hence in order to program a poetry machine, one would first have to repeat the entire universe from the beginning."

Stanislaw Lem
The Cyberiad

"The first umpire ,. . .a man of small knowledge of how meanings are made, says I calls 'em as they are. The second umpire, knowing something about human perception and its limitations, says 'I calls 'em as I sees 'em.' The third umpire, having studied at Cambridge with Wittgenstein himself, says 'Until I calls em, they ain't.' "

Neil Postman
Crazy Talk, Stupid Talk

"Symptoms can become criteria."

Ludwig Wittgenstein
Philosophical Investigations

"'But how can a rule shew me what I have to do at this point? Whatever I do is, on some interpretation, in accord with the rule.'- That is not what we ought to say, but rather: any interpretation still hangs in the air along with what it interprets, and cannot give it any support. Interpretations by themselves do not determine meaning. "

Ludwig Wittgenstein
Philosophical Investigations

"Our mistake is to look for an explanation where we ought to look at what happens as a 'protophenomenon'."

Ludwig Wittgenstein
Philosophical Investigations

"I think one reason why the attempt to find an explanation is wrong is that we have only to put together in the right way what we know, without adding anything, and the satisfaction we are trying to get from the explanation comes of itself."

Ludwig Wittgenstein
Remarks on Frazer's Golden Bough

"In the use of words one might distinguish 'surface grammar' from 'depth grammar'.What immediately impresses itself upon us about the use of a word is the way it is used in the construction of the sentence, the part of its use- one might say- that can be taken in by the ear.------- And now compare the depth grammar, say of the word 'to mean', with what its surface grammar would lead us to suspect. No wonder we find it difficult to know our way about."

Ludwig Wittgenstein
Philosophical Investigations

"I caught this insight on the way and quickly seized the rather poor words that were closest to hand to pin it down lest it fly away again. And now it has died of these arid words and shakes and flaps in them - and I hardly know any more when I look at it how I could ever have felt so happy when I caught this bird."

Friedrich Nietzsche
The Gay Science

"Once, when the holy man of Toganoo was journeying along a road he encountered a man washing a horse by a river. 'Ashi, ashi', said the man. ['Ashi' means 'leg'. The man is telling the horse to lift its leg.] The holy man stopped in his tracks and exclaimed 'How inspiring! Some deed of virtue in a previous existence has brought this man enlightenment! He is reciting the invocation aji,aji! [The priest believes or pretends that the man is saying 'aji', the first letter of the Sanskrit alphabet, which has a religious significance for the priest.] I wonder whose horse it might be? Such piety overcomes me!' When he asked about the owner, the man replied 'The horse belongs to Lord Fushö.' 'Splendid!' cried the holy man. 'This is truly a case of ajo hon fushö. [The formula ajo hon fushö means that there is no beginning of creation; that is, that the world has always existed. The washing man's phonological prime brings the holy formula to the priest's mind.] What a fortunate link you have established with the Way of the Buddha!' He wiped away the tears of gratitude."

Kenkö
Essays In Idleness

"Repetition always commits us to imagining an unknown cause, so true is it that in the popular consciousness, the aleatory is always distributive, never repetitive: chance is supposed to vary events; if it repeats them, it does so in order to signify something through them; to repeat is to signify...."

Roland Barthes
Structure of the Fait-Divers

"Our mistake is to look for an explanation where we ought to look at what happens as a 'protophenomenon'."

Ludwig Wittgenstein
Philosophical Investigations

"Any arrangement of acts and events is comic which gives us, in a single combination, the illusion of life and the distinct impression of a mechanical arrangement."

Henri Bergson
Laughter

"A portion of the mind abundantly commissured to other portions works almost mechanically. It sinks to a condition of a railway junction. But a portion of mind almost isolated, a spiritual peninsula, or cul-de-sac, is like a railway terminus. Now mental commisures are habits. Where they abound, originality is not needed and is not found; but where they are in defect, spontaneity is set free. Thus, the first step in the Lamarckian evolution of mind is the putting of sundry thoughts into situations in which they are free to play."

Charles Peirce
Evolutionary Love

"If something is boring after two minutes, try it for four. If still boring, then eight. Then sixteen. Then thirty-two. Eventually one discovers that it is not boring at all."

John Cage

"But just think of the unfathomable laziness of man; all the schemes which are supposed to keep him awake and watchful end up by putting him to sleep. We wear a hairshirt the way we might wear a monocle; we sing matins the way other people play golf. If only scientists today, instead of constantly inventing new means to make life easier, would devote their resourcefulness to producing instruments for rousing man out of his torpor! There are machine guns, of course, but that's going a bit too far..."

Rene Daumal
Mount Analogue

Happiness
I want objects
Like pagan alcohol
To scrawl the stomach of reason
And the cock's crow
To curse the sun
The devil's pastime
Whims what happiness
I proceed entirely at random.

Francis Picabia

"All messages and parts of messages are like phrases or segments of equations which a mathematician puts in brackets. Outside the brackets there maybe a qualifier or multiplier which will alter the whole tenor of the phrase. Moreover, these qualifiers can always be added, even years later. They do not have to precede the phrase inside the brackets. Otherwise, there could be no psychotherapy...What exists today are only messages about the past that we call memories, and these messages can always be framed and modulated from moment to moment."

Gregory Bateson
Steps To An Ecology Of Mind

"That is how man's anguish ends- in masterly conjuring tricks: pure poetry, pure music, pure thought.

The last man- who has freed himself from all belief, from all illusions, and has nothing more to expect or to fear- sees the clay of which he is made reduced to spirit, and this spirit has no soil left for its roots, from which to draw its sap. The last man has emptied himself; no more seed, no more excrement, no more blood. Everything having turned into words, every set of words into musical jugglery, the last man goes even further: he sits in his utter solitude and decomposes the music into mute mathematical equations."

Nikos Kazantzakis
Zorba, The Greek

"We will never get a calculus to do all that a natural language such as Inuktitut or French does because not all the moves in a natural language are analytic: sometimes thoughts are connected by tonal associations; images; family resemblances. Not every linguistic impulse is disintegrative; some clear and expressible thoughts aim at a tapestry, rather than a complex, or a peak.
Many of our finest expressions slip through the lattice of even the most most powerful algebra."

Jan Zwicky
Lyric Philosophy

"Who else but the naive poet is able to pucker his lips to kiss that old sourpuss, the world? "

Irving Layton
As quoted in: *The Montreal Gazette*
October 19, 1985

Appendix 2: Hello World

** By popular demand, here's a sample file containing the 'Hello world' (Introductory
** programming examples) of textDNA. To run this file, put it into the folder called
** 'TextDNA' (inside the 'JanusNode Resources' folder), inside another folder,
** and select it from the menu underneath the picture button. (Your JanusNode ships
** with this file already installed)
**
** I have displayed the actual TextDNA in red.
**
**
** 1.) Hello
** Here is the simplest rule. It just says hello.
** It calls 'RunMe' because some rules in this file need to follow each other (see the
** final rules).
Subject(Example1-Hello,RunMe) "Hello!" return RunMe
**
** 2.) HelloX
** Here is nearly the simplest possible rule: JanusNode's 'Hello world'. It just says Hello
** to many concrete things.
Subject(Example2-HelloX,RunMe) "Hello" s_nouns "!" return RunMe
**
** 3.) HelloHi
** This rule uses string constants and choice brackets to say 'hi' about as often
** as it says 'hello'
Subject(Example3-HelloHi,RunMe) { "Hello" | "Hi" } s_nouns "!" return RunMe
**
** 4.) HelloHiAdjective
** This rule adds an optional adjective to whatever it is greeting
Subject(Example4-HelloHiAdjective,RunMe) { "Hello" | "Hi" } adjectives 50 s_nouns "!" return RunMe
**
** 5.) HelloHiPunctuated
** This rule allows more varied punctuation using the built-in 'punctuate' command
Subject(Example5-HelloHiPunctuated,RunMe) { "Hello" | "Hi" } adjectives 50 s_nouns punctuate return RunMe
**
** 6.) HelloGoodbyeToX
** Now we get fancier: Add in a variable, and say hello or hi to one variety of it, and
** good bye or bye to another, using the 'assign' and 'get' functions. Now we always end
** with a period, and add a return so that each pair is distinct in the output.
Subject(Example6-HelloGoodbyeToX,RunMe) < assign(HelloObject,s_nouns) > { "Hello" | "Hi" } adjectives 50 < get(HelloObject) > punctuate return { "Good bye" | "Bye" } adjectives 50 < get(HelloObject) > "." return return RunMe
**
** 7.) HelloGoodbyeRhyme
** Fancier still- here's a rhyming rule. It selects a RhymePrefix using the command:
** < assign(HelloRhymePrefix,"ee,on,ink,ing,est") >

** Then it chooses an actual word from one file named HelloRhymePrefix+noun
** using the command:
** < GetRhyme(HelloRhymePrefix,noun) >
** To add more RhymeFiles,just make them use a naming convention that ends
** each file-name with its type: (eg) 'noun' and 'verb' (and containing words
** appropriate to its suffix) and add the beginning affix on their
** file name to the 'assign(RhymeFile,"a,b,c")' command below (where a,b, and c are
** examples of a prefix, and, by convention, signal the nature of their contents).
** The rhyme files of course, go in the 'BrainFood' directory.
** This rule also selects a single greeting ('Hello', 'Hi'. 'Godbye', or 'Bye').
Subject(Example7-HelloGoodbyeRhyme,RunMe) < assign(HelloRhymePrefix,"ee,on,ink,ing,est") > < assign(Greeting,"Hello,Hi,Goodbye,Bye") > < get(Greeting) > adjectives 50 < GetRhyme(HelloRhymePrefix,noun) > punctuate return < get(Greeting) > adjectives 50 < GetRhyme(HelloRhymePrefix,noun) > "." return RunMe
**
** 8.) HelloGoodbyeMisterRhyme
** Same as 7.) above, but now we add a possessive in, using a TextDemon. TextDemons are
** pre-defined code chunks that are defined in the TextDemon field.The one used here-
** 'TextDemonCapitalizeFakeWord'- returns a capitalized fake word.
Subject(Example8-HelloGoodbyeMisterRhyme,RunMe) < assign(HelloRhymePrefix,"ee,on,ink,ing,est") > < assign(Greeting,"Hello,Hi,Goodbye,Bye") > < assign(Title,"Mr.,Ms.,Miss,Mrs.") > < assign(Title2,"Mr.,Ms.,Miss,Mrs.") > < get(Greeting) > < get(Title) > TextDemonCapitalizeFakeWord "'s" adjectives 50 < GetRhyme(HelloRhymePrefix,noun) > punctuate return < get(Greeting) > < get(Title2) > TextDemonCapitalizeFakeWord "'s" adjectives 50 < GetRhyme(HelloRhymePrefix,noun) > "." return RunMe
**
** 9.) HelloGoodbyePoem
** Finally, let's put it all together to write a silly little poem.
** The textdemon 'TextDemonReturnSpacing' just gives a return, plus an arbitrary number
** of spaces.
** I have replaced 'adjectives ' with the more interesting substitution:
** { adjectives | PhilosophicalAdjs | p_verbsnob "ing" |
** < GetRhyme(HelloRhymePrefix,verb) > "ing" }
**
Subject(Example9-HelloGoodbyePoem,RunMe) < assign(HelloRhymePrefix,"ee,on,ink,ing,est") > < assign(HelloGreeting,"Hello,Hi,Hey") > < assign(GoodbyeGreeting,"Goodbye,Bye") > < assign(Title,"Mr.,Ms.,Miss,Mrs.") > < assign(Syllable1,syllables) > < assign(Syllable2,syllables) > < assign(Adj1,adjectives) > return "Here's" < get(Title) > < CapitalizeNext() > < Get(Syllable1) > < Backspace() > < Get(Syllable2) > "'s" { adjectives | PhilosophicalAdjs | p_verbsnob "ing" | < GetRhyme(HelloRhymePrefix,verb) > "ing" | < get(Adj1) > } < GetRhyme(HelloRhymePrefix,noun) > punctuate TextDemonReturnSpacing < get(HelloGreeting) > < get(Title) > < CapitalizeNext() > < Get(Syllable1) > < Backspace() > < Get(Syllable2) > "!" TextDemonReturnSpacing < get(HelloGreeting) > { adjectives | PhilosophicalAdjs | p_verbsnob "ing" | < GetRhyme(HelloRhymePrefix,verb) > "ing" } < GetRhyme(HelloRhymePrefix,noun) > punctuate TextDemonReturnSpacing { "That's no" | "What a" } { < Get(Adj1) > | adjectives 50 | PhilosophicalAdjs 50 | s_nouns "ing" } < GetRhyme(HelloRhymePrefix,noun) > punctuate TextDemonReturnSpacing < get(GoodbyeGreeting) > < get(Title) > < CapitalizeNext() > < Get(Syllable1) > < Backspace() > < Get(Syllable2) > { "," | "-" | ";" | "!" TextDemonReturnSpacing } {

"go get a" | "you need a" } { adjectives | PhilosophicalAdjs | p_verbsnob "ing" | < GetRhyme(HelloRhymePrefix,verb) > "ing" } < GetRhyme(HelloRhymePrefix,noun) > "!" return return RunMe
**
** JanusNode 3.0 Update
***
** The only point of the next example is that the rule below is calling
** textDNA, textDemons, BrainFood, literal text, and punctuation without
** explicitly specifying which is which. As of version 3.0, JanusNode 3.0
** assumes that every string is one of these and works out which is which.
****
** 'WhoItWas0' is defined in the TextDemons file.
**
subject(Start,RunMe) CharAttributes WhoItWas0 WhatTheyDid0 WhyYouShouldCare0 Am I not 50 right? return return RunMe
**
** Quotes around literals aren't necessary as of v3.0 (so long as there are
** no textDNA, textDemons, or BrainFood files with the same name) but they are still
** a good idea, and speed up the rule parsing a little.
**
subject(WhatTheyDid0) { "simplified nonsense generation" | "trained for a marathon" | "single-handedly" 30 "invented" s_nouns | "had to work to master" ArtSpeakAdjectives 50 ArtSpeakNouns } "."
***
subject(WhyYouShouldCare0) { "Now it's easy." | "You can do it too." | "No one else could have done it." }
**
** JanusNode 3.6 Update
***
** Lines in a BrainFood file can now be TextDNA! This shows a very simple example, by
** calling a BrainFood file called 'RunSomeBrainFoodAsTextDNA'
**
Subject(Example10-BrainFoodAsTextDNA,RunMe) RunSomeBrainFoodAsTextDNA RunMe