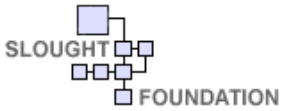

Electronic Text Composition Project
Jim Carpenter

The Electronic Text Composition Project's (ETC) Poetry Engine is a suite of software components that allow a user to generate aesthetic texts. Drawing word associations from its language database, the Engine's grammar uses a probability-based approach to constructing syntactic constituents, which it aggregates into utterances, which it in turn aggregates into compositions. The project postulates that the construction of its texts does not actually occur within the software—these constructions, absent authorial intent and divorced from any underlying message, assume their status as poems only as they are read. The process of textual construction is firmly situated within the reader, not the software. Over the last year a dozen poems composed with the Poetry Engine's aid and submitted under the pen name Erica T. Carter have been accepted for publication in a number of little magazines and literary journals. As evidence of the project's success (or perhaps indicative of its failure), one editor accepted a poem with the comment, "I found your works intriguing, but have to admit I couldn't wrest the meaning from them."

What constitutes the Text is…its subversive force in respect of the old classifications.
-- Roland Barthes

A Google search on the term "poetry generator" returns nearly 4200 pages, a tangible illustration of the considerable interest that exists for computed poetry. Most of these are variations of cutups, randomly imposing new patterns on existing texts, or template-based, non-deterministic finite state transducers that axiomatize a set of well-formed utterance constituent strings and then transduce each constituent type into a word selected from the type's allowed set of outputs. Both approaches work, after a fashion and briefly.

These kinds of computed poetry problematize discussions of the texts that they generate. In short, the texts, if they are interesting at all, are interesting only momentarily. They fail to compel our interest for very long, provide little in the way of reading pleasure, and dull our curiosity to seek out other similar texts. Though the methods may provoke some discussion, the texts do not.

In defending John Cage's mesostics, Marjorie Perloff claims legitimacy for what are essentially cutups because his textual realignments produce new meaning. They do, but not because of simple rearrangement. If that were the case, then simply randomizing texts would also produce new meaning, but it doesn't—the result is always gibberish, as shown here by the results of feeding this sentence to a token randomizer:

- token simply then new always texts produce shown that it meaning, also sentence this results case, If but the doesn't—the gibberish, would here of the by feeding result randomizer is randomizing to were as a

- that this but randomizing meaning, it of randomizer also token feeding always shown by simply a gibberish, here were case, as to results the If result produce texts new would sentence doesn't—the the is then

- sentence to this case, would as is by then randomizer randomizing token If the feeding simply meaning, a doesn't-the result produce were also but the gibberish, of shown texts that new results here it always

The tokens are preserved but at the loss of syntax. The mesotic technique works because of something other (and less) than randomization. It does not rearrange the words—it rearranges fragments of syntax, word clusters whose extraction preserves not just their morphology but a lump of grammar without which meaning is impossible. From this recognition follows the first proposition that a proper poetry engine must effect: It isn't words that allow meaning, but associations of words.

But what of the transducers? Here is a sample from a poetry generator available on the Web site *Alien Central* (www.aliencentral.com):

Aaron destroys hearts orgasmically

The alien breaks woman
Man knows sex coldly
Man spanks loneliness
Woman wants funny faces unabashedly
The witch knows brains
The alien kisses Aaron warmly
Man vomits the headless torso
The mothership twitches love orgasmically

The heuristics of this generator are obvious. Alien's generator begs a seed word, in this case a name, from the user—hence "Aaron"—which it uses to directly address the user and thereby "personalize" the poem. All verbs are present tense and transitive. All nouns are singular. All sentences are patterned subject-verb-object. Sentences are optionally modified by an adverb. There are no adjectives. (Additional generations reveal that *torso* never appears without *headless*; Alien's lexicon has only the collocation *headless torso*.) Alien's grammar allows it to generate meaningful sentences and its limited vocabulary allows it to establish something like cohesion, but now at the loss of originality. "Compute" another poem and the SVO pattern begins to be tedious. Compute another and Alien's limited vocabulary begins to become repetitious—it seems that various objects and beings can do a number of things (penetrate, lick, want, love, eat, spank) *feverishly*, or *deftly*, or *sexually* but rarely in any unqualified way. Alien could become more varied in its composition by expanding its vocabulary, but at the expense of cohesion. Alien's restricted, domain-specific words are all that stitch its patches of text together into something approximating a composition.

Propositions two and three follow from these weaknesses: The proper poetry engine must be capable of simulated imagination and originality. The proper poetry engine must produce compositions with unity of thought or meaningful pattern—some discernable and plausible reason for this string of words to have coalesced into a whole.

The Poetry Machine that realizes these propositions, words in meaningful combinations, originality, and cohesion, will have nudged computed poetry out of mere novelty and passing diversion. It will have composed texts worth reading. And most importantly, it will have obsolesced the Author and rendered Him irrelevant. The Author will not just be dead, as Barthes declared, but unnecessary.

The algorithms and data structures of the ETC Poetry Engine simultaneously confront the limitations of previous computed-poetry software and traditional pen and ink (or keyboard and monitor) approaches to developing texts. Reduced to its simplest, computed poetry's problem of uninteresting output is a form of Chomsky's "poverty-of-the-stimulus" problem. They suffer from a paucity of words and failure to recognize that aesthetic texts are not composed of words but of word associations. ETC attacks this problem with a relational database constructed from the sentences and bigrams of the British National Corpus. Of the 100,000,000 words in the corpus, 85,000,000 were discovered to be useful to the Engine. The remaining 15,000,000 are headings and explanations, which could not be meaningfully parsed. The massive size of this source text resulted in a basic lexicon with 560,000 words and various tables of word associations (essentially a data cube of multiple dimensions each consisting of foreign keys pointing to bigrams of lexical pairs) numbering more than 49,000,000. This database structure allows for the conceptualization of utterances as vectors of associations rather than strings of morphemes and facilitates the design of algorithms that can

generate far richer and complex compositional structures than poetry generators have previously accomplished. And its enormous size allows for the same infinite number of possible utterances as the language after which it is modeled. Algorithmically, ETC comes down solidly on the empiricist side of the nativist-empiricist debate. It advantages itself of the probabilities that various word associations appear in normal discourse by computing them as the relative frequencies of their occurrences in the database's source text, the British National Corpus. ETC's poems, which are essentially vectors of utterances, begin with a short analytical walk, constructing utterance frames, implemented as objects, from the 35 possible frames defined in the WordNet lexical database. But all reliance on a hard-wired grammar ends there. Empiricism takes over. Each frame selects a set of plausible constituent types from those the database allows it. But neither the poem of chains of utterances nor the utterance frames construct the constituents. The constituents construct themselves stochastically from their types and the range of possible probabilities that constrain their content, again drawing guidance from the database.

So just as human writers serve up Barthes's "variety of writings, none of them original" and texts consisting of "tissue[s] of quotations drawn from the innumerable centers of culture," ETC composes its own quotations, equally unoriginal, and as worthy of attention, from the same source, language—all without the messy hypocrisy of human inspiration.

Ron Silliman says in "Hidden" that "The poem is not what's dug into the cave wall nor cut on the wrist—it goes deeper." If the poem is not its materiality and it is not its brand in the flesh, then where does it live? If we didn't know Ron better, we'd suspect he was suggesting the soul. But we do know better and know now that one deeper place it can live is in the fluff of the magnetic patterns on a computer's hard drive and in the interaction of software objects.

Later in "Hidden" Ron asks, "Why do I write?" Good question.

*

The source files which follow are at the heart of the engine. The first piece of code, "Constituents," demonstrates the self-awareness of the objects that comprise utterances and how they are programmed with the intelligence to construct and populate themselves with other constituents that in turn know how to construct and populate themselves until all that's left are terminals, which only know how to transform themselves grammatically based on the terminals that surround them.

The second piece of code, "SentenceFrames," is almost mystical. It constructs sentence patterns from lists of object types. Starting with a word, it finds a verb with which that word is likely to be associated in meaningful discourse, then selects all of the sentence frame types within which that verb could be reasonably situated. Then it constructs an object frame from just the type. This is as abstract as it is possible for code to be. The type is an abstraction of a frame, which is an analytical abstraction of convenience, and the final object is never anywhere declared--it just comes into existence from the concept of its signature. It is analogous to the concrete number 1009 emerging from the abstraction "integer." This is the most elegant and aesthetically complete code I've written.

```
using System;
using System.Reflection;
using System.Collections;
using System.Data;

namespace ETC
{
    /// <summary>
    /// Summary description for Constituents.
    /// </summary>
    abstract public class Constituent
    {
        protected bool semanticsRealized = false;
        public Constituent()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        abstract public void RealizeSemantics(Word[] wordPool);
        abstract public Word[] LexigraphicList
        {
            get;
        }
    }

    public class NounPhrase : Constituent
    {
        protected Noun headNoun;
        protected Determiner determiner;
        protected AdjectivePhrase adjectivePhrase;
        // Offset into array is number of adjectives. Adjust with experience

        override public Word[] LexigraphicList
        {
            get
            {
                ArrayList lexigraphicList = new ArrayList();
                if ( determiner.BaseWord != string.Empty )
                    lexigraphicList.Add(determiner);

                if ( adjectivePhrase.Count > 0 )
                    lexigraphicList.AddRange(adjectivePhrase.LexigraphicList);

                lexigraphicList.Add(headNoun);

                // I'm pretty sure the only place you can get an "a" is in the first
                // element of a Noun Phrase.  This tells the Determiner to present itself
                // as "an" if the next word starts with a vowel.
                if ( lexigraphicList.Count > 1 && lexigraphicList[0].GetType() == typeof(Determiner) )
                {
                    ((Determiner)lexigraphicList[0]).followingWord = (Word)lexigraphicList[1];
                }

                return (Word[])lexigraphicList.ToArray(typeof(Word));
            }
        }
```

```
    public NounPhrase(Noun headNoun)
    {
        Logger.GlobalLogger.Log("Constructing NounPhrase - head noun: " + headNoun.ToString());
        this.headNoun = headNoun;
        //determiner = new Determiner(headNoun);
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log("NounPhrase.RealizeSemantics - head noun: " + headNoun.ToString());
        determiner = new Determiner(headNoun);
        determiner.RealizeSemantics();

        adjectivePhrase = new AdjectivePhrase(headNoun);
        adjectivePhrase.RealizeSemantics(wordPool);
    }

}

// This class can instantiate based on a noun the phrase is to
// modifiy or based on a preposition that it should use.  The
// reason for using a preposition is to support repetitive phrasing
// such as "in the air, in the evening, in her smile."  You get
// a PP someplace and then use its preposition to force another one.
public class PrepositionalPhrase : Constituent
{
    protected Noun nounModified = null;
    protected Preposition headPreposition = null;
    protected NounPhrase nounPhrase = null;

    override public Word[] LexigraphicList
    {
        get
        {
            ArrayList lexigraphicList = new ArrayList();
            lexigraphicList.Add(headPreposition);
            lexigraphicList.AddRange(nounPhrase.LexigraphicList);

            return (Word[])lexigraphicList.ToArray(typeof(Word));
        }
    }

    public PrepositionalPhrase(Noun nounModified)
    {
        Logger.GlobalLogger.Log("Constructing PrepositionalPhrase - nounModified: " + nounModified.BaseWord);
        this.nounModified = nounModified;
    }

    public PrepositionalPhrase(Preposition prep)
    {
        Logger.GlobalLogger.Log("Constructing PrepositionalPhrase - prep: " + prep.BaseWord);
        headPreposition = prep;
    }

    public PrepositionalPhrase(Preposition prep, Noun noun)
    {
        Logger.GlobalLogger.Log("Constructing PrepositionalPhrase - prep: " +
                                prep.BaseWord + " noun: " + noun.BaseWord);
        headPreposition = prep;
```

```
    nounPhrase = new NounPhrase(noun);
}


public override void RealizeSemantics(Word[] wordPool)
{
    if ( semanticsRealized )
        return;

    Logger.GlobalLogger.Log("PrepositionalPhrase.RealizeSemantics");

    // The only way this guy could be !null is if
    // we constructed this object with the prep,noun ctor
    if ( nounPhrase != null )
    {
        nounPhrase.RealizeSemantics(wordPool);
        return;
    }

    if ( nounModified != null )
    {
        RealizeSemanticsFromModifiedNoun(wordPool);
        return;
    }

    if ( headPreposition != null )
    {
        RealizeSemanticsFromPreposition(wordPool);
        return;
    }

    throw new Exception("PrepositionalPhrase.RealizeSemantics() - no noun or preposition");
}

protected void RealizeSemanticsFromModifiedNoun(Word[] wordPool)
{
    Logger.GlobalLogger.Log("PrepositionalPhrase.RealizeSemanticsFromModifiedNoun (head noun = '" +
                            nounModified.ToString() + ")");

    semanticsRealized = true;
    // There are 2851190 instances of "of" in the Lexicon, and 10885667 total prepositions
    // This guides whether we will use "of"
    bool useOF = ETCRandom.XofY(2851190,10885667);
    useOF = false;

    string objectNoun = string.Empty;
    if ( useOF )
    {
        headPreposition = new Of();
        objectNoun = GlobalDBInterface.Database.GetOfPrepositionalPhraseObject(nounModified.ToString());
        nounPhrase = new NounPhrase(new Noun(objectNoun));
        nounPhrase.RealizeSemantics(wordPool);
    }
    else
    {
        // Use something other than "of"
        headPreposition = Preposition.GetRandomPreposition();

        objectNoun = GlobalDBInterface.Database.GetPrepositionalPhraseObject(headPreposition.BaseWord);
```

```
            nounPhrase = new NounPhrase(new Noun(objectNoun));
            nounPhrase.RealizeSemantics(wordPool);
        }

    }

    protected void RealizeSemanticsFromPreposition(Word[] wordPool)
    {
        Logger.GlobalLogger.Log("PrepositionalPhrase.RealizeSemanticsFromPreposition (" +
                                headPreposition.ToString() + ")");

        semanticsRealized = true;

        Noun[] nouns = Word.ExtractNouns(wordPool);
        string[] nounStrings = Noun.NounArray2StringArray(nouns);

        string objectNoun = GlobalDBInterface.Database.GetPrepositionalPhraseObject(headPreposition.BaseWord, nounStrings);

        nounPhrase = new NounPhrase(new Noun(objectNoun));
        nounPhrase.RealizeSemantics(wordPool);
    }

}


public class RelativeClause : Constituent
{
    protected Noun nounModified;

    override public Word[] LexigraphicList
    {
        get { return null; }
    }

    public RelativeClause(Noun nounModified)
    {
        this.nounModified = nounModified;
    }

    public override void RealizeSemantics(Word[] wordPool)
    {

    }

}

abstract public class Predicate : Constituent
{
    protected Verb verb;
    protected Noun subject; // A lot of the time this will be null
    protected Noun complement;

    public Noun Subject
    {
        get { return subject; }
        set { subject = value; }
    }

    public Predicate(Verb verb)
```

```
    {
        this.verb = verb;
    }

}

public class VerbOnlyPredicate : Predicate
{
    protected ArrayList constituentsArray;

    override public Word[] LexigraphicList
    {
        get
        {
            ArrayList lexigraphicList = new ArrayList();

            foreach ( Constituent c in constituentsArray )
                lexigraphicList.AddRange(c.LexigraphicList);

            return (Word[])lexigraphicList.ToArray(typeof(Word));
        }
    }

    public VerbOnlyPredicate(Verb verb) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing VerbOnlyPredicate('{0}')",verb.ToString()));
        constituentsArray = new ArrayList();
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        if ( semanticsRealized )
            return;

        // Possibilities
        // Verb is not modified by an adverb
        // Verb is pre-modified by a single adverb
        // Verb is post-modified by a single adverb
        // Verb is pre-modified by a compound adverb
        // Verb is post-modified by a compound adverb

        constituentsArray.Add(new VerbPhrase(verb));

        bool advPrecedes;

        string adverb = GlobalDBInterface.Database.GetAdverbGivenVerbSP(verb.BaseWord,out advPrecedes,1);
        if ( adverb != string.Empty )
        {
            AdverbPhrase advPhrase = new AdverbPhrase(new Adverb(adverb));

            // This isn't 100%, because surface realization might change things,
            // but we don't say "It softly is raining"; we say "It is raining softly."
            if ( verb.Progressive )
                advPrecedes = false;

            if ( advPrecedes )
                constituentsArray.Insert(0,advPhrase);
            else
                constituentsArray.Add(advPhrase);
```

```
        }

        semanticsRealized = true;
    }
}


public class VerbPPPredicate : VerbOnlyPredicate
{
    protected Preposition preposition;
    protected Noun.NOUNTYPE ppObjectType;

    public Noun.NOUNTYPE PP_ObjectType
    {
        get { return ppObjectType; }
        set { ppObjectType = value; }
    }

    public VerbPPPredicate(Verb verb) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing VerbPPPredicate('{0}')",verb.BaseWord));
        preposition = null;
        ppObjectType = Noun.NOUNTYPE.Any;
    }

    public VerbPPPredicate(Verb verb, Preposition preposition) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing VerbPPPredicate('{0}' '{1}')",
                                              verb.BaseWord,preposition.BaseWord ));
        this.preposition = preposition;
        ppObjectType = Noun.NOUNTYPE.Any;
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log(string.Format("VerbPPPredicate.RealizeSemantics"));

        if ( semanticsRealized )
            return;

        base.RealizeSemantics(wordPool);

        string prepString = string.Empty;

        if ( preposition != null )
            prepString = preposition.BaseWord;
        else
            prepString = GlobalDBInterface.Database.GetPrepositionGivenVerbSP(verb.BaseWord, false);

        //This is complicated - allow for person or thing objects on the PP
        if ( prepString != string.Empty )
        {
            DataTable contextNouns = GlobalDBInterface.Database.GetContextWords(verb.BaseWord, Verb.PosCode, Noun.PosCode);
            PrepositionalPhrase PP = new PrepositionalPhrase(new Preposition(prepString));

            PP.RealizeSemantics(Noun.StringArray2NounArray(Conversions.DataTable2StringArray(contextNouns,"word")));

            constituentsArray.Add(PP);
```

```
        }

        semanticsRealized = true;
    }
}

public class VerbNounComplementPredicate : VerbOnlyPredicate
{
    public Noun.NOUNTYPE complementNounType = Noun.NOUNTYPE.Any;

    public VerbNounComplementPredicate(Verb verb) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing VerbNounComplementPredicate('{0}')",verb.BaseWord));
        //constituentsArray = new ArrayList();
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log(string.Format("VerbPPPredicate.RealizeSemantics"));

        if ( semanticsRealized )
            return;

        base.RealizeSemantics(wordPool);

        complement = verb.GetNounFromContext(complementNounType);

        NounPhrase np = new NounPhrase(complement);
        np.RealizeSemantics(wordPool);
        constituentsArray.Add(np);

        semanticsRealized = true;
    }
}

public class VerbNounComplement_Adjective_Predicate : VerbNounComplementPredicate
{
    public VerbNounComplement_Adjective_Predicate(Verb verb) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing VerbNounThingComplement_Adjective_Predicate('{0}')",
                                              verb.BaseWord));
        //constituentsArray = new ArrayList();
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log(string.Format("VerbNounThingComplement_Adjective_Predicate.RealizeSemantics"));

        if ( semanticsRealized )
            return;

        base.RealizeSemantics(wordPool);

        if ( complement != null )
        {
            AdjectivePhrase ap = new AdjectivePhrase(complement);
            ap.NumberOfAdjectives = 1;
            ap.RealizeSemantics(wordPool);
            constituentsArray.Add(ap);
```

```
        }

        semanticsRealized = true;
    }
}

public class VerbIndirectObjDirectObjPredicate : VerbNounComplementPredicate
{
    protected Noun indirectObject;
    public Noun.NOUNTYPE indirectObjectNounType = Noun.NOUNTYPE.Any;

    public VerbIndirectObjDirectObjPredicate(Verb verb) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing VerbIndirectObjDirectObjPredicate('{0}')",
                                              verb.BaseWord));
        //constituentsArray = new ArrayList();
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log(string.Format("VerbIndirectObjDirectObjPredicate.RealizeSemantics"));

        if ( semanticsRealized )
            return;

        base.RealizeSemantics(wordPool);

        indirectObject = verb.GetNounFromContext(indirectObjectNounType);
        NounPhrase np = new NounPhrase(indirectObject);
        np.RealizeSemantics(wordPool);
        constituentsArray.Add(np);

        complement = verb.GetNounFromContext(complementNounType);
         np = new NounPhrase(complement);
        np.RealizeSemantics(wordPool);
        constituentsArray.Add(np);

        semanticsRealized = true;
    }

}

public class VerbDirectObjectPPPredicate : VerbNounComplementPredicate
{

    public Noun.NOUNTYPE ppObjectNounType = Noun.NOUNTYPE.Any;
    protected Preposition prep;

    public VerbDirectObjectPPPredicate(Verb verb, Preposition prep) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("VerbDirectObject_ToNounPredicate('{0}')",verb.BaseWord));
        this.prep = prep;
        //constituentsArray = new ArrayList();
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log(string.Format("VerbDirectObject_ToNounPredicate.RealizeSemantics"));
```

```
        if ( semanticsRealized )
            return;

        base.RealizeSemantics(wordPool);

        Noun ppObject = verb.GetNounFromContext(ppObjectNounType);

        PrepositionalPhrase pp = new PrepositionalPhrase(prep,ppObject);
        pp.RealizeSemantics(wordPool);
        constituentsArray.Add(pp);

        semanticsRealized = true;
    }

}




public class Verb_Adjective_Predicate : VerbOnlyPredicate
{
    public Verb_Adjective_Predicate(Verb verb) : base(verb)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing Verb_Adjective_Predicate('{0}')",verb.BaseWord));
        //constituentsArray = new ArrayList();
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log(string.Format("Verb_Adjective_Predicate.RealizeSemantics"));

        if ( semanticsRealized )
            return;

        base.RealizeSemantics(wordPool);

        // In this case, we end up with a "fake" subject
        if ( subject == null )
            subject = verb.GetNounFromContext(Noun.NOUNTYPE.Any);
                                // 1,2,3
        int[] howManyAdjectives = {10,2,1};
        int numAdjectives = ETCRandom.WeightedSelection(howManyAdjectives) + 1;

        for ( int i = 0; i < numAdjectives; ++i )
        {
            AdjectivePhrase ap = new AdjectivePhrase(subject);
            ap.NumberOfAdjectives = 1;
            ap.RealizeSemantics(wordPool);

            constituentsArray.Add(ap);
        }

        semanticsRealized = true;
    }
}

abstract public class Subject : Constituent
```

```
{
}

public class NounSubject : Subject
{
    protected Noun headNoun;
    protected ArrayList constituentsArray;

    override public Word[] LexigraphicList
    {
        get
        {
            ArrayList lexigraphicList = new ArrayList();

            foreach ( Constituent c in constituentsArray )
                lexigraphicList.AddRange(c.LexigraphicList);

            return (Word[])lexigraphicList.ToArray(typeof(Word));
        }
    }

    public NounSubject(Noun headNoun)
    {
        this.headNoun = headNoun;
        constituentsArray = new ArrayList();

    }

    protected struct ConstituentList
    {
        public Type[] list;
        public int    weight;
        public ConstituentList(Type[] list, int weight)
        {
            this.list = list;
            this.weight = weight;
        }
    }
    // This guy holds the possible constituent lists for a subject and their respective
    // probablities for being selected
    static protected ConstituentList[] constituentLists =
    {
        new ConstituentList(new Type[] { typeof(NounPhrase) }, 100),
        new ConstituentList(new Type[] { typeof(NounPhrase),typeof(PrepositionalPhrase) }, 25),
        new ConstituentList(new Type[] { typeof(NounPhrase),typeof(PrepositionalPhrase), typeof(PrepositionalPhrase) }, 25),
        /*
        new ConstituentList(new Type[] { typeof(NounPhrase),typeof(RelativeClause) }, 10),
        new ConstituentList(new Type[] { typeof(NounPhrase),typeof(AppositionalPhrase) }, 5),
        new ConstituentList(new Type[] { typeof(NounPhrase),typeof(InfinitiveClause) },    5),
        */
    };
    /*
    static protected Type[][] constituentLists =
    {
        new Type[] { typeof(NounPhrase) },
        new Type[] { typeof(NounPhrase),typeof(PrepositionalPhrase) },
        new Type[] { typeof(NounPhrase),typeof(PrepositionalPhrase), typeof(PrepositionalPhrase) },
        new Type[] { typeof(NounPhrase),typeof(RelativeClause) },
        new Type[] { typeof(NounPhrase),typeof(AppositionalPhrase) },
```

```
    new Type[] { typeof(NounPhrase),typeof(InfinitiveClause) },
};
*/
public override void RealizeSemantics(Word[] wordPool)
{
    // Possible structures - see constituentLists for implementations

    // NounPhrase
    // NounPhrase PP
    // NounPhrase PP PP
    // NounPhrase RelativeClause
    // NounPhrase AppositionalPhrase
    // InfinitiveClause (derive from Noun?)
    // Any of these can be compound with any other structure

    if ( semanticsRealized )
        return;


    Logger.GlobalLogger.Log("Subject.RealizeSemantics()" + headNoun.ToString());

    int[] listWeights = new int[constituentLists.Length];
    for ( int i = 0; i < listWeights.Length; ++i )
    {
        listWeights[i] = constituentLists[i].weight;
    }
    int structureIndex = ETCRandom.WeightedSelection(listWeights);

    //structureIndex = 1;

    Logger.GlobalLogger.Log("   Using structure index: " + structureIndex.ToString());

    ConstructorInfo ci;
    Constituent constituent;
    constituentsArray.Clear();

    foreach ( Type t in constituentLists[structureIndex].list )
    {
        // The ctors of constituents of Subject all take either a Noun
        // or a void parameter.  Find out which and create it here
        ci = t.GetConstructor(new Type[] { typeof(Noun) } );
        if ( ci != null )
        {
            constituent = (Constituent)ci.Invoke(new object[] { headNoun } );
            constituent.RealizeSemantics(wordPool);
            constituentsArray.Add(constituent);
            continue;
        }
        ci = t.GetConstructor(new Type[0] );
        if ( ci != null )
        {
            constituent = (Constituent)ci.Invoke(new object[0]);
            constituent.RealizeSemantics(wordPool);
            constituentsArray.Add(constituent);
            continue;
        }

        throw new Exception("In Subject.RealizeSemantics(): Unexpected ctor list for " + t.Name);
    }
```

```
            semanticsRealized = true;
        }

    }

    public class PersonalPronounSubject : Subject
    {
        PersonalPronoun pronoun;

        public PersonalPronounSubject(PersonalPronoun pronoun)
        {
            this.pronoun = pronoun;
        }

        public override Word[] LexigraphicList
        {
            get
            {
                return new Word[] { pronoun };
            }
        }

        public override void RealizeSemantics(Word[] wordPool)
        {
            // Later, like when I'm good looking again, add
            // prepositional modifers:  "She, in the bloom of her youth."
        }


    }

    public class AppositionalPhrase : Constituent
    {
        protected Noun headNoun;

        override public Word[] LexigraphicList
        {
            get { return null; }
        }

        public AppositionalPhrase(Noun headNoun)
        {
            this.headNoun = headNoun;
        }

        public override void RealizeSemantics(Word[] wordPool)
        {

        }

    }

    public class InfinitiveClause : Constituent
    {
        override public Word[] LexigraphicList
        {
            get { return null; }
        }
```

```
    public InfinitiveClause()
    {

    }

    public override void RealizeSemantics(Word[] wordPool)
    {

    }

}

public class AdjectivePhrase : Constituent
{
    //Chance of adjectives: 0   1  2  3
    static protected int[] adjSetSizeWeights = new int[] { 60, 20, 5, 1 };

    protected _APAdjective[] adjectives;
    protected Noun nounModified;
    // If this stays at -1, then the object is to decide how many adjectives to create/
    // Otherwise, it will use this 'forced' number.
    protected int numberOfAdjectives = -1;

    // What this guy allows is for an adjective
    // modified by an adverb as a single entity;
    protected struct _APAdjective
    {
        public Adverb adv;
        public Adjective adj;

        public void GetAdverb()
        {
            Logger.GlobalLogger.Log("   AdjectivePhrase._A{Adjective.GetAdverb() for '" + adj.ToString() + "'");
            if ( adj == null )
                return;
            adv = null; // If this guy somehow got called before, reinitialize

            string adverb = GlobalDBInterface.Database.GetAdverbGivenAdjectiveSP(adj.ToString(),2);
            if ( adverb == null || adverb.Length > 0 )
            {
                Logger.GlobalLogger.Log("   AdjectivePhrase._A{Adjective.GetAdverb() got '" + adverb + "'");
                adv = new Adverb(adverb);
            }
            Logger.GlobalLogger.Log("   AdjectivePhrase._A{Adjective.GetAdverb() didn't get adv for '" + adj.ToString() + "'");
        }
    }

    public int Count
    {
        get
        {
            if ( adjectives == null )
                return 0;

            return adjectives.Length;
        }
    }
```

```
    public int NumberOfAdjectives
    {
        get { return numberOfAdjectives; }
        set { numberOfAdjectives = value; }
    }

    override public Word[] LexigraphicList
    {
        get
        {
            ArrayList lexigraphicList = new ArrayList();

            foreach ( _APAdjective adj in adjectives )
            {
                if ( adj.adv != null )
                    lexigraphicList.Add(adj.adv);
                lexigraphicList.Add(adj.adj);
            }

            return (Word[])lexigraphicList.ToArray(typeof(Word));
        }
    }

    public AdjectivePhrase(Noun nounModified)
    {
        Logger.GlobalLogger.Log(string.Format("Constructing AdjectivePhrase({0})", nounModified.ToString()));
        this.nounModified = nounModified;
    }

    public override void RealizeSemantics(Word[] wordPool)
    {
        Logger.GlobalLogger.Log("AdjectivePhrase.RealizeSemantics()" + nounModified.ToString());

        // This is getting a little kludgy.
        int numAdjectives = numberOfAdjectives;
        if ( numAdjectives == - 1 )
            numAdjectives = ETCRandom.WeightedSelection( adjSetSizeWeights );

        Logger.GlobalLogger.Log("  Number of adjectives: " + numberOfAdjectives.ToString());

        adjectives = new _APAdjective[numAdjectives];

        for ( int i = 0; i < numAdjectives; ++i )
        {
            string adj = GlobalDBInterface.Database.GetAdjectiveGivenNounSP(nounModified.ToString());
            adjectives[i].adj = new Adjective(adj);
            adjectives[i].GetAdverb();
        }

    }
}

public class AdverbPhrase : Constituent
{
    protected Adverb adverb;

    public override Word[] LexigraphicList
    {
        get
```

```
            {
                return new Word[] { adverb };
            }
        }

        public AdverbPhrase(Adverb adverb)
        {
            this.adverb = adverb;
        }

        public override void RealizeSemantics(Word[] wordPool)
        {

        }

    }

    public class VerbPhrase : Constituent
    {
        protected Verb verb;

        public override Word[] LexigraphicList
        {
            get
            {
                return new Word[] { verb };
            }
        }

        public VerbPhrase(Verb verb)
        {
            this.verb = verb;
        }

        public override void RealizeSemantics(Word[] wordPool)
        {

        }

    }
}
```

```
using System;
using System.Reflection;

namespace ETC
{

   abstract public class SentenceFrame
   {
      int frameNumber; // the frame number as defined in the table VerbFrameDefs
      protected Word baseWord;
      public CompositionParams compositionParams;

      public SentenceFrame(Word baseWord,int frameNumber)
      {
         compositionParams = new CompositionParams();

         this.baseWord = baseWord;
         this.frameNumber = frameNumber;
      }

      abstract public SentenceTree GetTree();

      // This is decidedly non-OOP - the base class shouldn't know anything about derived classes,
      // but the WordNet verb frames are a fixed thing and generating the right one is a big
      // part of the reason why this class exists.  So fuck you!

      static protected Type[] frameTypesArray =
      {
         typeof(SomethingVerb),                    // 0
         typeof(SomebodyVerb),                     // 1
         typeof(ItIsVerbing),                      // 2
         typeof(SomethingIsVerbingPP),             // 3
         typeof(SomethingVerbSomethingAdjOrNoun),  // 4
         typeof(SomethingVerbAdjOrNoun),           // 5
         typeof(SomebodyVerbAdjective),            // 6
         typeof(SomebodyVerbSomething),            // 7
         typeof(SomebodyVerbSomebody),             // 8
         typeof(SomethingVerbSomebody),            // 9
```

```
    typeof(SomethingVerbSomething),            // 10
    typeof(SomethingVerbToSomebody),           // 11
    typeof(SomebodyVerbOnSomething),           // 12
    typeof(SomebodyVerbSomebodySomething),     // 13
    typeof(SomebodyVerbSomethingToSomebody),   // 14
    typeof(SomebodyVerbSomethingFromSomebody), // 15
    typeof(SomebodyVerbSomebodyWithSomething), // 16
    typeof(SomebodyVerbSomebodyOfSomething),   // 17
    typeof(SomebodyVerbSomebodyOnSomething),   // 18
    typeof(SomebodyVerbSomebodyPP),            // 19
    typeof(SomebodyVerbSomethingPP),           // 20
    typeof(SomebodyVerbPP),                    // 21
    typeof(SomebodysBodypartVerb),             // 22
    typeof(SomebodyVerbSomebodyToInfinitive),  // 23
    typeof(SomebodyVerbSomebodyInfinitive),    // 24
    typeof(SomebodyVerbThatClause),            // 25
    typeof(SomebodyVerbToSomebody),            // 26
    typeof(SomebodyVerbToInfinitive),          // 27
    typeof(SomebodyVerbWhetherInfinitive),     // 28
    typeof(SomebodyVerbIntoV_ingSomething),    // 29
    typeof(SomebodyVerbSomethingWithSomething),// 30
    typeof(SomebodyVerbInfinitive),            // 31
    typeof(SomebodyVerbV_ing),                 // 32
    typeof(ItVerbThatClause),                  // 33
    typeof(SomethingVerbInfinitive)            // 34
};

// From the base word, figures out which WordNet verb frames it can
// be used in.  Then randomly instantiates a SentenceFrame based
// on one of them, using the base word as the base word for the
// SentenceFrame.  This method is not as hard as it first looks.
static public SentenceFrame CreateFrame(Word baseWord)
{
    Logger.GlobalLogger.Log(string.Format("SentenceFrame CreateFrame({0}",baseWord.ToString()));

    Type wordType = baseWord.GetType();
    string typeName = wordType.Name; // So we can see it in the debugger
```

```
      int[] allowedFrameNumbers;

      // Different Word types require different processing to figure out
      // which frames work.  Do all of that in the switch.  The idea is
      // to get back an array of arrays (synsets -> frame numbers) that
      // work.
      switch ( typeName )
      {
         case "Verb":
            Logger.GlobalLogger.Log(string.Format("    {0}: type is verb",baseWord.ToString()));
            allowedFrameNumbers = GlobalDBInterface.Database.GetVerbFramesSP(baseWord.SynsetID);
            break;
         default:
            throw new Exception("Unexpected type '" + typeName + "' in SentenceFrame.CreateFrame()");
      }

      int synsetVerbFrameToUse = ETCRandom.GetRandomNumber(0,allowedFrameNumbers.Length - 1);

      int frameTypeOffset = allowedFrameNumbers[synsetVerbFrameToUse] - 1; // For debugging

      // FORCE FRAME FOR DEBUGGING (reset when done)
      frameTypeOffset = 18;

      Type frameType = frameTypesArray[frameTypeOffset];

      ConstructorInfo[] ci = frameType.GetConstructors();
      if ( ci.Length != 1 )
      {
         throw new Exception("Somebody added a SentenceFrame ctor and didn't change " +
                             "SentenceFrame.CreateFrame()");
      }

      SentenceFrame newSentenceFrame = (SentenceFrame)ci[0].Invoke(new object[] { baseWord });

      return newSentenceFrame;
   }
}
```

```
// 1  Something ----s
public class SomethingVerb : SentenceFrame
{
   public SomethingVerb(Word word) : base(word, 1)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomethingVerb.GetTree()");
      NounSubject subject = null;
      VerbOnlyPredicate predicate = null;
      SentenceTree tree = new SentenceTree();


      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            predicate = new VerbOnlyPredicate((Verb)baseWord);

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);
            subject = new NounSubject(noun);
            tree.Add(subject);
            tree.Add(predicate);
            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomethingVerb.GetTree()");
      }

      return tree;
   }
}

//2   Somebody ----s
```

```
public class SomebodyVerb : SentenceFrame
{
   public SomebodyVerb(Word word) : base(word, 2)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomethingVerb.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            VerbOnlyPredicate predicate = new VerbOnlyPredicate((Verb)baseWord);
            Subject subject;

            bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

            if ( pronounSubject )
            {
               PersonalPronoun pronoun = new PersonalPronoun();
               pronoun.Case = PersonalPronoun.CASE.Nominative;
               pronoun.Person = CompositionParams.GetSubjectPerson();

               subject = new PersonalPronounSubject(pronoun);
            }
            else
            {
               Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
               subject = new NounSubject(noun);
            }
            tree.Add(subject);
            tree.Add(predicate);
            break;
```

```
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
      }

      return tree;

   }
}


//3   It is ----ing
public class ItIsVerbing : SentenceFrame
{
   public ItIsVerbing(Word word) : base(word, 3)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("ItIsVerbing.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            Verb verb = (Verb)baseWord;
            verb.Progressive = true;
            verb.Tense = Verb.TENSE.Present;
            verb.Person = Verb.PERSON.Third;

            VerbOnlyPredicate predicate = new VerbOnlyPredicate((Verb)baseWord);

            PersonalPronoun pronoun = new PersonalPronoun();
            pronoun.Case = PersonalPronoun.CASE.Nominative;
```

```
            pronoun.Gender = PersonalPronoun.GENDER.Neuter;
            pronoun.Person = PersonalPronoun.PERSON.Third;

            PersonalPronounSubject subject = new PersonalPronounSubject(pronoun);

            tree.Add(subject);
            tree.Add(predicate);
            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
      }

      return tree;
   }

}

// 4  Something is ----ing PP
public class SomethingIsVerbingPP : SentenceFrame
{
   public SomethingIsVerbingPP(Word word) : base(word, 4)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomethingIsVerbing.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            Verb verb = (Verb)baseWord;
            verb.Progressive = true;
```

```
            verb.Person = Verb.PERSON.Third;

            VerbPPPredicate predicate = new VerbPPPredicate(verb);
            Constituent subject;

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);
            subject = new NounSubject(noun);
            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
      }

      return tree;

   }
}

// 5  Something ----s something Adjective/Noun  - I think I'm only going to do adjective
// The fog made the forest dark
public class SomethingVerbSomethingAdjOrNoun : SentenceFrame
{
   public SomethingVerbSomethingAdjOrNoun(Word word) : base(word, 5)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomethingVerbSomethingAdjOrNoun.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
```

```
            case "Verb":
            {
                Verb verb = (Verb)baseWord;
                verb.Person = Verb.PERSON.Third;

                VerbNounComplement_Adjective_Predicate predicate = new VerbNounComplement_Adjective_Predicate(verb);
                Constituent subject;

                Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);
                subject = new NounSubject(noun);
                tree.Add(subject);
                tree.Add(predicate);

                break;
            }
            default:
                throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
        }

        return tree;
    }

}

// 6  Something ----s Adjective/Noun
public class SomethingVerbAdjOrNoun : SentenceFrame
{
    public SomethingVerbAdjOrNoun(Word word) : base(word, 6)
    {

    }

    public override SentenceTree GetTree()
    {
        Logger.GlobalLogger.Log("SomethingVerbAdjOrNoun.GetTree()");

        SentenceTree tree = new SentenceTree();
```

```
      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);

            Constituent subject;
            subject = new NounSubject(noun);
            tree.Add(subject);

            Verb verb = (Verb)baseWord;
            verb.Person = Verb.PERSON.Third;

            Verb_Adjective_Predicate predicate = new Verb_Adjective_Predicate(verb);
            predicate.Subject = noun;


            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
      }

      return tree;

   }
}

// 7  Somebody ----s Adjective
public class SomebodyVerbAdjective : SentenceFrame
{
   public SomebodyVerbAdjective(Word word) : base(word, 7)
   {

   }
```

```
public override SentenceTree GetTree()
{
   Logger.GlobalLogger.Log("SomethingVerbAdjective.GetTree()");

   SentenceTree tree = new SentenceTree();

   switch ( baseWord.GetType().Name )
   {
      case "Verb":
      {
         Verb verb = (Verb)baseWord;
         verb.Person = Verb.PERSON.Third;

         Verb_Adjective_Predicate predicate = new Verb_Adjective_Predicate(verb);
         Subject subject;

         bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

         if ( pronounSubject )
         {
            PersonalPronoun pronoun = new PersonalPronoun();
            pronoun.Case = PersonalPronoun.CASE.Nominative;
            pronoun.Person = CompositionParams.GetSubjectPerson();

            subject = new PersonalPronounSubject(pronoun);
         }
         else
         {
            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
            subject = new NounSubject(noun);
         }
         tree.Add(subject);
         tree.Add(predicate);
         break;
      }

      default:
```

```
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
        }

        return tree;

    }
}

// 8  Somebody ----s something
public class SomebodyVerbSomething : SentenceFrame
{
    public SomebodyVerbSomething(Word word) : base(word, 8)
    {

    }

    public override SentenceTree GetTree()
    {
        Logger.GlobalLogger.Log("SomebodyVerbSomething.GetTree()");

        SentenceTree tree = new SentenceTree();

        switch ( baseWord.GetType().Name )
        {
            case "Verb":
            {
                Verb verb = (Verb)baseWord;
                Subject subject;

                bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

                if ( pronounSubject )
                {
                    PersonalPronoun pronoun = new PersonalPronoun();
                    pronoun.Case = PersonalPronoun.CASE.Nominative;
                    verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

                    subject = new PersonalPronounSubject(pronoun);
```

```
            }
            else
            {

               Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
               subject = new NounSubject(noun);
               verb.Person = Verb.PERSON.Third;
            }

            VerbNounComplementPredicate predicate = new VerbNounComplementPredicate(verb);
            predicate.complementNounType = Noun.NOUNTYPE.Thing;
            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
      }

      return tree;

   }
}

// 9  Somebody ----s somebody
public class SomebodyVerbSomebody : SentenceFrame
{
   public SomebodyVerbSomebody(Word word) : base(word, 9)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomebodyVerbBody.GetTree()");

      SentenceTree tree = new SentenceTree();
```

```
switch ( baseWord.GetType().Name )
{
   case "Verb":
   {
      Verb verb = (Verb)baseWord;
      Subject subject;

      bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

      if ( pronounSubject )
      {
         PersonalPronoun pronoun = new PersonalPronoun();
         pronoun.Case = PersonalPronoun.CASE.Nominative;
         verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();
         pronoun.Gender = CompositionParams.GetSubjectGender();
         subject = new PersonalPronounSubject(pronoun);
      }
      else
      {

         Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
         subject = new NounSubject(noun);
         verb.Person = Verb.PERSON.Third;
      }

      VerbNounComplementPredicate predicate = new VerbNounComplementPredicate(verb);
      predicate.complementNounType = Noun.NOUNTYPE.Person;
      tree.Add(subject);
      tree.Add(predicate);

      break;
   }
   default:
      throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
}

return tree;
```

```
   }
}


//10  Something ----s somebody
public class SomethingVerbSomebody : SentenceFrame
{
   public SomethingVerbSomebody(Word word) : base(word, 10)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomethingVerbSomebody.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            Verb verb = (Verb)baseWord;
            Subject subject;

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
            subject = new NounSubject(noun);
            verb.Person = Verb.PERSON.Third;

            VerbNounComplementPredicate predicate = new VerbNounComplementPredicate(verb);
            predicate.complementNounType = Noun.NOUNTYPE.Person;
            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
```

```
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
        }

        return tree;

    }
}

// 11 Something ----s something
public class SomethingVerbSomething : SentenceFrame
{
    public SomethingVerbSomething(Word word) : base(word, 11)
    {

    }

    public override SentenceTree GetTree()
    {
        Logger.GlobalLogger.Log("SomethingVerbSomebody.GetTree()");

        SentenceTree tree = new SentenceTree();

        switch ( baseWord.GetType().Name )
        {
            case "Verb":
            {
                Verb verb = (Verb)baseWord;
                Subject subject;

                Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);
                subject = new NounSubject(noun);
                verb.Person = Verb.PERSON.Third;

                VerbNounComplementPredicate predicate = new VerbNounComplementPredicate(verb);
                predicate.complementNounType = Noun.NOUNTYPE.Thing;
                tree.Add(subject);
                tree.Add(predicate);
```

```
            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerb.GetTree()");
      }

      return tree;

   }
}

// 12 Something ----s to somebody
//      As in "The idea pertains to him."  But
//      could also be "The idea pertains to the forest."
//      So we treat this as if it were "Something ----s to somebody (or something)."
public class SomethingVerbToSomebody : SentenceFrame
{
   public SomethingVerbToSomebody(Word word) : base(word, 12)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomethingVerbToSomebody.GetTree()");
      NounSubject subject = null;
      VerbPPPredicate predicate = null;
      SentenceTree tree = new SentenceTree();


      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            predicate = new VerbPPPredicate((Verb)baseWord, new Preposition("to"));
            // Make it a 50/50 chance that the object will be a person or thing.
            // The chances are that the selection will do its own thing (no pun
            // intended) anyway!
```

```
            predicate.PP_ObjectType = (ETCRandom.XofY(1,2) ?  Noun.NOUNTYPE.Thing : Noun.NOUNTYPE.Person);

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);
            subject = new NounSubject(noun);
            tree.Add(subject);
            tree.Add(predicate);
            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomethingVerbToSomebody.GetTree()");
      }

      return tree;

   }
}

// 13 Somebody ----s on something
public class SomebodyVerbOnSomething : SentenceFrame
{
   public SomebodyVerbOnSomething(Word word) : base(word, 13)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomebodyVerbOnSomething.GetTree()");
      NounSubject subject = null;
      VerbPPPredicate predicate = null;
      SentenceTree tree = new SentenceTree();


      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            predicate = new VerbPPPredicate((Verb)baseWord, new Preposition("on"));
```

```
            predicate.PP_ObjectType = Noun.NOUNTYPE.Person;

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Thing);
            subject = new NounSubject(noun);
            tree.Add(subject);
            tree.Add(predicate);
            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbOnSomething.GetTree()");
      }

      return tree;

   }
}

// 14 Somebody ----s somebody something
public class SomebodyVerbSomebodySomething : SentenceFrame
{
   public SomebodyVerbSomebodySomething(Word word) : base(word, 14)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomebodyVerbSomebodySomething.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            Verb verb = (Verb)baseWord;
            Subject subject;
```

```
            bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

            if ( pronounSubject )
            {
               PersonalPronoun pronoun = new PersonalPronoun();
               pronoun.Case = PersonalPronoun.CASE.Nominative;
               verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

               subject = new PersonalPronounSubject(pronoun);
            }
            else
            {

               Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
               subject = new NounSubject(noun);
               verb.Person = Verb.PERSON.Third;
            }

            VerbIndirectObjDirectObjPredicate predicate = new VerbIndirectObjDirectObjPredicate(verb);
            predicate.complementNounType = Noun.NOUNTYPE.Thing;
            predicate.indirectObjectNounType = Noun.NOUNTYPE.Person;

            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbSomebodySomething.GetTree()");
      }

      return tree;
   }
}

// 15 Somebody ----s something to somebody
public class SomebodyVerbSomethingToSomebody : SentenceFrame
{
```

```
public SomebodyVerbSomethingToSomebody(Word word) : base(word, 15)
{

}

public override SentenceTree GetTree()
{
   Logger.GlobalLogger.Log("SomebodyVerbSomethingToSomebody.GetTree()");

   SentenceTree tree = new SentenceTree();

   switch ( baseWord.GetType().Name )
   {
      case "Verb":
      {
         Verb verb = (Verb)baseWord;
         Subject subject;

         bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

         if ( pronounSubject )
         {
            PersonalPronoun pronoun = new PersonalPronoun();
            pronoun.Case = PersonalPronoun.CASE.Nominative;
            verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

            subject = new PersonalPronounSubject(pronoun);
         }
         else
         {

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
            subject = new NounSubject(noun);
            verb.Person = Verb.PERSON.Third;
         }

         VerbDirectObjectPPPredicate predicate = new VerbDirectObjectPPPredicate(verb, new Preposition("to"));
         predicate.complementNounType = Noun.NOUNTYPE.Thing;
```

```
            predicate.ppObjectNounType = Noun.NOUNTYPE.Person;

            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbSomethingToSomebody.GetTree()");
      }

      return tree;
   }
}

// 16 Somebody ----s something from somebody
public class SomebodyVerbSomethingFromSomebody : SentenceFrame
{
   public SomebodyVerbSomethingFromSomebody(Word word) : base(word, 16)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomebodyVerbSomethingFromSomebody.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            Verb verb = (Verb)baseWord;
            Subject subject;

            bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);
```

```
            if ( pronounSubject )
            {
               PersonalPronoun pronoun = new PersonalPronoun();
               pronoun.Case = PersonalPronoun.CASE.Nominative;
               verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

               subject = new PersonalPronounSubject(pronoun);
            }
            else
            {

               Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
               subject = new NounSubject(noun);
               verb.Person = Verb.PERSON.Third;
            }

            VerbDirectObjectPPPredicate predicate = new VerbDirectObjectPPPredicate(verb, new Preposition("from"));
            predicate.complementNounType = Noun.NOUNTYPE.Thing;
            predicate.ppObjectNounType = Noun.NOUNTYPE.Person;

            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbSomethingFromSomebody.GetTree()");
      }

      return tree;

   }
}

// 17 Somebody ----s somebody with something
public class SomebodyVerbSomebodyWithSomething : SentenceFrame
{
   public SomebodyVerbSomebodyWithSomething(Word word) : base(word, 17)
```

```
{

}

public override SentenceTree GetTree()
{
   Logger.GlobalLogger.Log("SomebodyVerbSomethingWithSomething.GetTree()");

   SentenceTree tree = new SentenceTree();

   switch ( baseWord.GetType().Name )
   {
      case "Verb":
      {
         Verb verb = (Verb)baseWord;
         Subject subject;

         bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

         if ( pronounSubject )
         {
            PersonalPronoun pronoun = new PersonalPronoun();
            pronoun.Case = PersonalPronoun.CASE.Nominative;
            verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

            subject = new PersonalPronounSubject(pronoun);
         }
         else
         {

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
            subject = new NounSubject(noun);
            verb.Person = Verb.PERSON.Third;
         }

         VerbDirectObjectPPPredicate predicate = new VerbDirectObjectPPPredicate(verb,new Preposition("with"));
         predicate.complementNounType = Noun.NOUNTYPE.Thing;
         predicate.ppObjectNounType = Noun.NOUNTYPE.Thing;
```

```
            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbSomethingWithSomething.GetTree()");
      }

      return tree;
   }
}

// 18 Somebody ----s somebody of something
public class SomebodyVerbSomebodyOfSomething : SentenceFrame
{
   public SomebodyVerbSomebodyOfSomething(Word word) : base(word, 18)
   {

   }

   public override SentenceTree GetTree()
   {
      Logger.GlobalLogger.Log("SomebodyVerbSomethingOfSomething.GetTree()");

      SentenceTree tree = new SentenceTree();

      switch ( baseWord.GetType().Name )
      {
         case "Verb":
         {
            Verb verb = (Verb)baseWord;
            Subject subject;

            bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

            if ( pronounSubject )
            {
```

```
                PersonalPronoun pronoun = new PersonalPronoun();
                pronoun.Case = PersonalPronoun.CASE.Nominative;
                verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

                subject = new PersonalPronounSubject(pronoun);
            }
            else
            {

                Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
                subject = new NounSubject(noun);
                verb.Person = Verb.PERSON.Third;
            }

            VerbDirectObjectPPPredicate predicate = new VerbDirectObjectPPPredicate(verb,new Preposition("of"));
            predicate.complementNounType = Noun.NOUNTYPE.Thing;
            predicate.ppObjectNounType = Noun.NOUNTYPE.Person;
            tree.Add(subject);
            tree.Add(predicate);

            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbSomethingOfSomething.GetTree()");
      }

      return tree;

   }
}

// 19 Somebody ----s something on somebody
public class SomebodyVerbSomebodyOnSomething : SentenceFrame
{
   public SomebodyVerbSomebodyOnSomething(Word word) : base(word, 19)
   {

   }
```

```
public override SentenceTree GetTree()
{
   Logger.GlobalLogger.Log("SomebodyVerbSomebodyOnSomething.GetTree()");

   SentenceTree tree = new SentenceTree();

   switch ( baseWord.GetType().Name )
   {
      case "Verb":
      {
         Verb verb = (Verb)baseWord;
         Subject subject;

         bool pronounSubject = ETCRandom.XofY(CompositionParams.personalPronounSubjectIndex, 100);

         if ( pronounSubject )
         {
            PersonalPronoun pronoun = new PersonalPronoun();
            pronoun.Case = PersonalPronoun.CASE.Nominative;
            verb.Person = pronoun.Person = CompositionParams.GetSubjectPerson();

            subject = new PersonalPronounSubject(pronoun);
         }
         else
         {

            Noun noun = ((Verb)baseWord).GetNounFromContext(Noun.NOUNTYPE.Person);
            subject = new NounSubject(noun);
            verb.Person = Verb.PERSON.Third;
         }

         VerbDirectObjectPPPredicate predicate = new VerbDirectObjectPPPredicate(verb,new Preposition("on"));
         predicate.complementNounType = Noun.NOUNTYPE.Person;
         predicate.ppObjectNounType = Noun.NOUNTYPE.Thing;
         tree.Add(subject);
         tree.Add(predicate);
```

```
            break;
         }
         default:
            throw new Exception("Unexpected Word type in SomebodyVerbSomebodyOnSomething.GetTree()");
      }

      return tree;
   }
}

// 20 Somebody ----s somebody PP
public class SomebodyVerbSomebodyPP : SentenceFrame
{
   public SomebodyVerbSomebodyPP(Word word) : base(word, 20)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 21 Somebody ----s something PP
public class SomebodyVerbSomethingPP : SentenceFrame
{
   public SomebodyVerbSomethingPP(Word word) : base(word, 21)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}
```

```
// 22 Somebody ----s PP
public class SomebodyVerbPP : SentenceFrame
{
   public SomebodyVerbPP(Word word) : base(word, 22)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 23 Somebody's (body part) ----s
public class SomebodysBodypartVerb : SentenceFrame
{
   public SomebodysBodypartVerb(Word word) : base(word, 23)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 24 Somebody ----s somebody to INFINITIVE
public class SomebodyVerbSomebodyToInfinitive : SentenceFrame
{
   public SomebodyVerbSomebodyToInfinitive(Word word) : base(word, 24)
   {

   }

   public override SentenceTree GetTree()
   {
```

```
      return null;
   }
}

// 25 Somebody ----s somebody INFINITIVE
public class SomebodyVerbSomebodyInfinitive : SentenceFrame
{
   public SomebodyVerbSomebodyInfinitive(Word word) : base(word, 25)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 26 Somebody ----s that CLAUSE
public class SomebodyVerbThatClause : SentenceFrame
{
   public SomebodyVerbThatClause(Word word) : base(word, 26)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 27 Somebody ----s to somebody
public class SomebodyVerbToSomebody : SentenceFrame
{
   public SomebodyVerbToSomebody(Word word) : base(word, 27)
   {
```

```
    }

    public override SentenceTree GetTree()
    {
        return null;
    }
}

// 28 Somebody ----s to INFINITIVE
public class SomebodyVerbToInfinitive : SentenceFrame
{
    public SomebodyVerbToInfinitive(Word word) : base(word, 28)
    {

    }

    public override SentenceTree GetTree()
    {
        return null;
    }
}

// 29 Somebody ----s whether INFINITIVE
public class SomebodyVerbWhetherInfinitive : SentenceFrame
{
    public SomebodyVerbWhetherInfinitive(Word word) : base(word, 29)
    {

    }

    public override SentenceTree GetTree()
    {
        return null;
    }
}

// 30 Somebody ----s somebody into V-ing something
public class SomebodyVerbIntoV_ingSomething : SentenceFrame
```

```
{
   public SomebodyVerbIntoV_ingSomething(Word word) : base(word, 30)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 31 Somebody ----s something with something
public class SomebodyVerbSomethingWithSomething : SentenceFrame
{
   public SomebodyVerbSomethingWithSomething(Word word) : base(word, 31)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 32 Somebody ----s INFINITIVE
public class SomebodyVerbInfinitive : SentenceFrame
{
   public SomebodyVerbInfinitive(Word word) : base(word, 32)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
```

```
}

// 33 Somebody ----s VERB-ing
public class SomebodyVerbV_ing : SentenceFrame
{
   public SomebodyVerbV_ing(Word word) : base(word, 33)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 34 It ----s that CLAUSE
public class ItVerbThatClause : SentenceFrame
{
   public ItVerbThatClause(Word word) : base(word, 34)
   {

   }

   public override SentenceTree GetTree()
   {
      return null;
   }
}

// 35 Something ----s INFINITIVE
public class SomethingVerbInfinitive : SentenceFrame
{
   public SomethingVerbInfinitive(Word word) : base(word, 34)
   {

   }
```

```
        public override SentenceTree GetTree()
        {
            return null;
        }
    }

}
```