# Document history

* 17/03/2011 – first version, for discussion
* 24/03/2011 – API proposals
* 31/03/2011 – updates to API to handle inflectional variants properly

# Introduction

This document makes some proposals concerning the structure of the Lexicon component in simpleNLG. These are motivated by the following issues:

* Currently, we do not handle variants properly. Specifically, we want to provide users with full control over:
  + spelling variants (*heart attack* vs *heart-attack*)
  + inflectional variants (*fish* – which can be pluralised as *fish* or *fishes*)
  + (occasionally, a spelling variant may itself be an inflectional variant)

A prerequisite for user control over which variant of a word to use is that the information that these are indeed variants – ie that they are related forms of the same lexeme – is explicitly represented.

* We have two “word” types in simpleNLG: WordElement and InflectedWordElement and their use needs to be rationalised. For example, the Lexicon’s lookup facility will return a WordElement, even when the string searched for is inflected (thus, calling Lexicon.lookupWord(“dogs”) returns the WordElement for *dog*).
* Longer term, I would like to have a framework that is broad enough to accommodate new information, such as derivational relations (e.g. the relationship between *do* and *doable*). If we have a flexible way of representing such relationships as those between variants, then this could also be expanded later to incorporate these new relationships.

# The sorts of things we’re concerned with

The “word”-like objects manipulated by simpleNLG include the following:

1. **Base words:** Simple lexemes, such as *heart*. Fundamentally, these are strings that have:
   1. Their basic form
   2. Possibly a unique ID
   3. A category (noun, verb etc)
   4. Zero or more inflectional forms (*hearts* etc)
2. **Base words + particles:** essentially a variation of (1), where in addition to properties (1a-c) we also have a particle (*e.g. get up*)
3. **Inflectional forms:** these are forms of base words with specific morpho-syntactic information, such as *plural* or *tense*. In a sense, inflectional forms are really just modified forms of the base words themselves (e.g. *slept* is really just *sleep* with a past tense feature).
4. **(Longer term) Derivational forms:** We may in future want to include derivation in the lexicon (some of this is already available in the NIH Specialist Lexicon). Like inflections, derivations are related to base words, but have a different category. Hence, unlike inflectional forms, they are not strictly modified forms of a base word. They’re more like independent words (e.g. *doable* is not just a form of *do* with a feature; it’s a different word, an adjective, with a different meaning). In the long term, I propose that these be handled as independent words, with the Lexicon holding information about the derivational relations.
5. **(Longer term) Phrasal lexemes:** we may in future want to include a more general treatment of “phrases” such as idioms and multi-word expressions. These will probably need to be included in the lexicon. (Multi-word expressions were explicitly included in the lexicon in the BabyTalk application, for example). There are already several cases of such expressions (including compounds) in the NIH Lexicon. Should these be treated as atomic strings, on a par with base words? This may be a good first approximation.

## The problem of variants

In addition, we also have the problem of variants:

* A single base word may exhibit different spellings
* A single word/multiword expression may have an acronym
* A base word may also have different inflectional forms (e.g. count or uncount).

The question is: should all of these be encapsulated within a single object, or treated as separate forms? Note that this is akin to the problem of inflections – should inflections be treated as separate objects, or as part of a word?

# A proposal

I think the basic word element we should think about is the **lemma**, by which I mean a single object which contains:

* (Possibly) an ID
* A baseform
* A category
* Zero or more inflectional forms. These might just be stored as feature information, to be rendered by the morphology module. But it’s the word itself which holds things like what type of plural it takes (reg, glreg etc).
* Zero or more spelling variants
* Information about possible inflectional variants. For example, the WordElement for *fish* would incorporatethe information that this is both regular (*fishes*) and uncount (plural *fish*), leaving it up to the user to choose the default.
* Information about acronyms. Since acronyms may have different morphological properties, compared to the full forms, they should probably be treated as separate WordElements. However, the connection between the full form and the acronym/abbreviation should be preserved.

In short, the idea is to **represent within a single lemma (a WordElement) all the information related to a single word.**

## An example

Let’s take a specific example from the NIH Lexicon:

|  |  |
| --- | --- |
| **NIH Entry** | **Current simpleNLG 4 representation** |
| {base=ATPase  spelling\_variant=ATP-ase  spelling\_variant=AT-Pase  entry=E0000126  cat=noun  variants=uncount  variants=reg  acronym\_of=adenosinetriphosphatase|E0066297} | WordElement  *id* = E0000126  *base*=ATPase  *category*=noun    *features:*  acronym\_of=... |

* spelling variants are currently not incorporated into the lexical entry; they need to be looked up separately and are returned as distinct WordElements via Lexicon.getWordsFromVariant(String,LexicalCategory). This means that the information about the relationship between the variants and the entry is lost.
* Since one of the inflectional variants of this word is “reg”, its other variant (uncount) is ignored. (*reg* is not actually marked as a feature; it’s assumed by default in the absence of other inflectional features).
  + There is a problem of transparency here: inflectional properties within WordElement are separate LexicalFeatures. This means the word is assigned LexicalFeature.REG=true. To maintain all information, it would also need to be assigned LexicalFeature.NON\_COUNT=true. In this case, every time the word is inflected, it would have to be checked for both features (in the worst case, a word would need to be checked for many inflectional possibilities individually). An alternative would be to have a single String-valued feature, INFL, whose value can be one or more strings. The WordElement would also need to be equipped with a default function, where the user could directly specify what the default inflectional form to use is for their application.
* Acronyms are only marked in one direction: an acronym points to its full form via the acronym\_of feature. Ideally, the relationship should be marked both ways. The ACRONYM\_OF LexicalFeature is currently specified as expecting a string value. It should be a list and have a corresponding ACRONYMS feature (pointing to the acronyms of a word) whcih is also a list. They must be lists because the acronym-word relation is in principle many-to-many.

An alternative representation would be as shown below (changes marked in boldface):

WordElement

*id* = E0000126

*base*=ATPase

*category*=noun

***default\_spell\_variant =* ATPase**

***default\_infl\_variant =* reg**

*features:*

**spelling\_variants = {ATP-ase, AT-Pase}** [type: List<String>]

acronym\_of={...} [type: List<WordElement>]

**acronyms={...}** [type: List<WordElement>]

This would also require two additions to the API:

* void setDefaultInflectionalVariant(Inflection variant) //specify which one of the variants to use – this is a value of the Inflection feature (formerly called Pattern)
* void addInflectionalVariant(Inflection variant) //specify that this is another possible variant
* Collection<Inflection> getInflectionalVariants() //get all the inflectional variants – a collection of Inflection values
* List<String> getSpellingVariants() //get all the spelling variants
* void setDefaultSpellingVariant(String variant) //specify which spelling to use

This would mark specific spelling or inflectional variants to use as the default in the lifetime of an application. On lookup and load, these can be set automatically (e.g. always use “reg” as the default inflectional variant if available, use the baseform as the spelling variant). But they can be overridden. If overridden, then they are the ones used in the creation of InflectedWordElements.

In addition, since non-regular inflectional variants need to be stored, WordElements need to have an internal representation of the various forms associated with a given variant. This requires the following addition to the API:

* addInflectionalVariant(Inflection, String feature, String form) //associate the wordform *form* with feature *feature* for the variant given

This is used to store forms of a word. If the default inflectional variant is switched, the stored forms replace the ones originally set.

To take an example, currently, the verb *lie* is loaded into the NIHDBLexicon keeping only its regular form. In the new implementation, *lie* is still loaded as regular, but internally stores the possible variant Inflection.IRREGULAR, with the associated forms (lay, laid etc). If setDefaultInflectionalVariant is called with Inflection.IRREGULAR, it is these forms which are used.

**NB:** we may want to have methods in the Lexicon class to set defaults in this way, for example:

* Lexicon.setDefaultInflectionByBase(String base, String variant)
* Lexicon.setDefaultInflectionById(String id, String variant)
* Lexicon.setDefaultSpellingByBase(String base, String variant)
* Lexicon.setDefaultSpellingById(String id, String variant)

Internally, the lexicon could do this by looking up the relevant WordElement(s) given the base, and delegating to them directly by calling WordElement.setDefaultInflectionalVariant(variant) and so on. However, these don’t seem to be very necessary.

**NB2:** Note that the representation above assumes that we’re **not** marking inflectional features separately (e.g. LexicalFeature.NON\_COUNT = true) as before, but just a single INFL feature. The MorphologyProcessor and related classes will have to be altered accordingly. In particular, the MorphologyRules class checks whether LexicalFeature.NON\_COUNT is set for nouns. This needs t be modified to check whether the DEFAULT\_INFL feature is “uncount”.

As for acronyms, my proposal is to have the following:

* Full word forms (“coronary bypass graft”) and their acronyms (“CBG”) are represented as separate WordElements.
* The WordElement for the full form points to the WordElement for the acronym(s). The API therefore includes:
  + boolean WordElement.hasAcronym()
  + List<WordElement> getAcronyms() //return all the word elements that are acronyms of this word – there may be more than one
  + boolean isAcronym() //return true if the WordElement is an acronym
  + List<WordElement> getFullForms() //return the word elements of which this acronym is a possible full form.

**NB3:** The current (2011) implementation of the NIH lexicon API which is used by the NIHDBLexicon provides a GetAcronyms() method in the LexicalRecord object. In reality, this method retrieves the **full forms** corresponding to an acronym. So, given the record for *UK*, it will return *United Kingdom* etc. This method is used by the NIHLexiconDB class to obtain the full forms of acronyms (see NIHDBLExicon.addAcronymInfo()). However, the API does not provide the reverse method, i.e. to get the actual acronyms of full forms. So, given *United Kingdom,* there is no way to actually obtain *UK*. This is partly due to the way the DB is set up: UK points to United Kingdom but not vice versa. Therefore, I propose to allow WordElement to have the ACRONYM feature and related methods specified, but this won’t be set up automatically by the lexicon; it will be up to the user to specify.

# WordElement vs InflectedWordElement

* A WordElement holds all the information that can *potentially create a word form* from a base.
* An InflectedWordElement is an actual word form that is marked with specific morpho-syntactic information for use in a specific grammatical context. InflectedWordElements contain a reference to the WordElement of which they are a form. InflectedWordElements are used by the MorphologyProcessor to

There is currently some inconsistency in the treatment of these two in the NLGFactory.createNLGElement and Lexicon.lookupWord.

For words, a user may specify the word as a string and this can be either an *inflected* string (“dogs”) or a baseform. In either case, a WordElement is returned. For example, if the user wants “dogs”, whether or not she directly calls Lexicon.lookupWord or NLGFactory.createNLGElement (which will delegate to Lexicon.lookupWord), the WordElement for *dog* is returned, losing the info that the user explicitly specified the plural.

The following changes should ideally be implemented:

* If the user queries the lexicon for an inflected form, then an InflectedWordElement shuold be returned; this has a reference to the base WordElement in any case.
* Otherwise, a WordElement should be returned.

The above would require that the Lexicon.lookupWord and methods be altered. They currently return a WordElement. The problem is that InflectedWordElement is not a subclass of WordElement (for good reason: since InflectedWordElements point to their base WordElement). This can be resolved by creating a common superclass or interface for the two (say, LexicalElement) and generalising the relevant methods.

# Summary of changes to API

## simplenlg.features.LexicalFeature

|  |  |  |
| --- | --- | --- |
| **Feature** | **Expected Type** | **Action** |
| ACRONYM\_OF | List<WordElement> | Changed from String to a list of WordElements. |
| ACRONYMS | List<WordElement> | New feature |
| DEFAULT\_INFL | String | New feature |
| SPELL\_VARS | List<String> | New feature |
| DEFAULT\_SPELL | String | New feature |
| NON\_COUNT | boolean | Removed. Superseded by INFLECTIONS and DEFAULT\_INFL |

In addition, there is a new enum, Inflection, which contains the possible inflectional variants, and which replaces the old Pattern enum.

## simplenlg.framework.WordElement

* public void setDefaultInflectionalVariant(Inflection variant)
* public Inflection getDefaultInflectionalVariant()
* public Collection<Inflection> getInflectionalVariants()
* public boolean hasInflectionalVariant(Inflection infl)
* public List<String> getSpellingVariants()
* public void setDefaultSpellingVariant(String variant)
* public String getDefaultSpellingVariant()
* public List<String> getFeatureAsStringList() //required for inflectional and spelling variants