API Functions available under technology owned by ACI

A. The following is a tentative list of parts of speech we will use to match an existing parser:

adjective
adverb
interjection
noun
verb
auxiliary verb
conjunction
determiner (article, possibly quantifier, possibly demonstrative adj. and possessive adj.)
measure word (possibly)
adposition (preposition, postposition, and circumposition)
pronoun
cardinal number (possibly)

The following is a tentative list of parts of speech we will add for various languages:

coverb
particle
preverb

Contractions will be supported in their expanded form. Clitics will probably be regarded as combined with, and part of the relative word.

B. Data types are as follows:

Visible to user:

1. Simple types:

spelling	string	' most common dictionary spelling
part of speech	integer	' enumerated type based on above lists
descriptor	string	' indicator of dictionary definition
pronunciation	string	' dictionary pronunciation (s)
primary	boolean	' if primary definition, true, else false
word index	integer	' unique arbitrarily assigned value
delta index	integer	' index of a result of word operations

2. Complex types

```
type word value                                  '
        spelling          string        '  most common dictionary spelling
        part of speech    integer       '  enumerated type based on above lists
        descriptor        string        '  indicator of dictionary definition
        pronunciation     string        '  dictionary pronunciation(s)
        primary           boolean       '  if primary definition, true, else false
end type                                         '
```

Invisible to user:

Complex types

```
type word definition
        definition        proprietary   '  ACI definition value
end type

type differential                                '
        value             proprietary   '  ACI result value
        parent a          integer       '  word index of first source operand
        parent b          integer       '  word index of second source operand
        process           integer       '  identifier of creating process
end type                                         '
```

Notes:

The terms "superset" and "subset" are used for hypernym and hyponym respectively. Hypernym and hyponym are less well known and do not appear in most dictionaries. Furthermore hypernym and hyponym are nearly homophones and can be confused easily in conversation. Consistently with this, the term "part" is used for meronym, and "whole" for holonym.

Secondary spellings of words are assumed to be evaluated by the parser (for example "stops" is evaluated to "stop"). Only string values that can be returned by the parser as primary values are presently maintained in the dictionary.

The word index may be negative in a special case. When a calling process has no editing rights, software may nevertheless create a word definition that results from operations. If there is a match to a dictionary word, the index of that word is returned. But if there is no match for this word in the dictionary, then it will put it into an temporary array, and the value of the index is returned as a negated value to indicate the fact that it is special. At the end of the session in which the word was created, the temporary words created for that session are destroyed and the index values associated with them are freed for later use. Take, for example, an old joke, "What is black and dangerous and lives in a tree?" The joke answer is, "A crow with a machine gun." Dictionary software can create an definition meaning "crow with machine gun," even though no such thing exists in reality. If we find the difference between "soldier" and "machine gunner," and integrate it with the definition of "crow," we get "crow with machine gun." This made-up definition is created and a negative index value returned. At the end of the session, the definition is destroyed.

Some dictionary definitions contain multi-word terms. These are often referred to as terms, but may sometimes be referred to as words also. The fact that we refer to something as a word does not automatically mean it is a single word.

The delta index is an index of a differential in an array of values unique to the creating process. The difference between a soldier and a machine gunner is an example. The array is destroyed at the end of the session and its index values freed.

In the examples, the language is Visual Basic, and it assumes that the functions of the given names have been written to get the service functions over the internet. Some return values are arrays, these would have to be declared public and accessed directly in the functions called.

Error returns may result from operands that are out of range. Such is the case if we ask for a differential between "reluctant" and "orange." The operands cannot be compared, whether by ACI's technology or by human beings.

C. The following are the service functions that we are considering providing to web users (note that parts of speech are an enumerated data type, with 0 meaning "any"):

#1. Spell word, given its word index.

A word index is passed to the function. The spelling is returned if the word index is positive. If it is negative (indicating a made-up word, not in the dictionary), then a closest fitting superset value is found and the spelling for the word index is filled in for the spelling of that index, with that value; then a string consisting of "type of ", concatenated with the spelling, is returned.

Programming notes:

Return spelling for positive index value.
Return "type of " plus closest superset value for negative index value.
Return 0 for unrecognized index value, and set error value to invalid index.

#2. Get all visible word data for a single word index.

A word index is passed to the function, and all visible data associated with the word is returned. Return value is a structure containing one of each visible data type: spelling, part of speech, word index, descriptor, pronunciation, primary. An error, probably indicating an invalid word index, produces a return value of 0.

#3. Describe a word, given its word index.

Value passed is a word index. The return values is a string representing a partial definition, together with the word's descriptor (but note that not all words have descriptors filled in). The return value is a null-terminate string. If multiple returns are possible, then if one is marked as primary, that is returned, otherwise an error is returned and the error value set to ambiguity. Error return is 0, which may also indicate that the word is not in the dictionary.

Example:

```
dim x as string
x = describe(definition("car", noun, "sedan"))
```

The example line gets a partial definition of the primary definition of the noun, "car." The return value is "land vehicle, passenger, motor; automobile".

#4. Get index of definition of word, given spelling, part of speech, and spelling word of similar meaning.

A spelling of a word is passed, along with its part of speech, and a spelling of a word with a similar meaning. The similar word may have any relationship to the word in question except contextual. The error return value is 0, which may indicate that the first spelling/part of speech or second spelling are not in the dictionary (no such spelling is set for get-error function), that there is no combination of the second spelling that is in range of the first spelling/part of speech (out of range is set), or that there are multiple possible answers (ambiguity is set). Clearly, the word with the similar meaning must be chosen with care.

Examples:

```
dim a, b, integer
a = definition("car", noun, "vehicle")          ' automobile, rail car, cart?
a = definition("car", noun, "automobile")       ' this line succeeds
b = definition("iris", noun, "eye")             ' this line succeeds
```

#5. Get indices of all definitions based on spelling and part of speech.

A spelling of a word is passed to the function, along with a part of speech. The string is a single word or term expected to be defined in the dictionary. The return value is an array of integers of all word index values for that spelling of the same part of speech as the part of speech passed. If the part of speech is 0, then indices of words of all parts of speech are returned. After the last index in the array that is filled in, there is a value of 0. An error is returned for words not in the dictionary, and is indicated by the first index being set to 0.

#6. Get index of primary definition based on spelling and part of speech.

A spelling is passed, along with a part of speech. The function finds the primary definition for that spelling, if one exists, of the same part of speech as the part of speech passed. If the part of speech is 0, all parts of speech are regarded as matches. If there is only one definition for the spelling, that definition is returned, regardless of whether it is marked primary. Error return is 0. This may indicate that the spelling is not found in the dictionary (no such spelling is set for get error function), or that there is more than one definition, but none is marked as primary for that spelling (no primary is set). For an example, see contextual match (item #8).

#7. Get indices of all synonyms based on spelling and part of speech.

A spelling is passed, along with a part of speech. The software returns an array of indices of synonyms of words with the spelling, of the same part of speech as the part of speech

passed. To be a synonym, a word must have the same definition, but does not necessarily have the same context. The part of speech must be specified as a non-0 value, and such a value will cause the return value to be set to 0. If there are no synonyms, the function returns a value of 0. Note that the indices returned are for words of different definitions. One index is returned for each definition, and the service ensures that no two definitions are represented by synonyms of the same spelling.

#8. Get contextual matches given a spelling and a word index.

Values passed are a spelling, a part of speech, and a word index. The words represented by the spelling are checked do find contextual connections with the word index. The indices of those having such a connection are returned in a 0 terminated array.

Example: The example line searches many definitions of "air".

```
dim a, as integer                          ' word index
dim x(30) as integer                       '  return value
a = primary("breathe", verb)               '
contextual_match("air", noun, a, x)        ' fills in x array
```

This example returns x as an array of type word index consisting of a single entry, a noun whose spelling is "air", and whose descriptor says "atmosphere".

#9. Get contextual array given a pair of spellings.

Values passed are two spellings. The words represented by the first spelling are each checked against those of the second spelling. The pairs that match are returned in an array of type word index. This requires a two-dimensional array in which each row consists of a pair of matching indices.

#10. Get relationship between two words, given their word indices.

Values passed are word index type for each of two words. The relationship is returned in a bit field as:

0	none, out of range,
1	the first is a subset (or synonym) of the second,
2	the first is a superset (or synonym) of the second,
4	the words are antonyms
8	the first is a part/member of the second
16	the second is a part/member of the first
32	one of the words is a part of the other
64	the words are antonyms as defined in the dictionary,
128	the words have an identifiable common subset

If the words are synonyms then the value 3, subset AND superset, is returned. Synonyms are based on definition only, and are without regard to differences in contexts of words.

The part/member relationship needs explanation. A nostril is part of a nose, and a nose is part of a face, so the nostril is part of the face. A comparison of nostril and nose would set both the 16 and the 32 bits, as would a comparison of a nostril and a face. A stinger is a part of a bee, and a bee is a member of a hive, but the stinger is neither a part nor a member of a hive. A comparison of a bee and a hive would set the 16 bit, but not the 32 bit. A comparison of the stinger and a hive would set neither.

Antonyms are not necessarily those a person would expect. For example, "insufficient" and "sufficient" are antonyms, and "insufficient" and "excess" are also antonyms, even though "sufficient" and "excess" are compared with the same definition of "insufficient" and are not synonyms. (Ordinary dictionaries have problems with this, too. Antonyms often need to have usage explained.)

The existence of an identifiable common subset does not imply that the subset is in the dictionary, but only that it could exist, or be created, if need be.

There is no specific result for common superset, as any relationship (possibly excepting membership) implies this.

#11. Get index of most complete common superset, given the indices of two words.

Values passed are word index type for each of two words. The return value is the most complete common superset, if such a value can be computed. Return value of 0 means such a value cannot be computed, which indicates the words are not closely enough related to have such a value. The most complete common superset is a word that contains all information common to the two definitions. It may be a term made up for the occasion.

For example:

```
s = spell(superset(primary("skateboard", noun), primary ("sloop", noun)))
```

will set s to "vehicle".

#12. Get index of most complete common subset, given the indices of two words.

Values passed are word index type for each of two words. The return value is the most complete common subset, if such a value can be returned. It is necessary in this case that the words be very similar as any conflict between definitions will produce an error. The error return is 0, probably indicating the operands conflict or are out of range, or an index is invalid.

For example:

s1 = spell(subset(definition("coupe", noun, "car"), definition("convertible", noun, "car")))

will set s1 to "roadster".

#13. Get index of nearest known superset, given index of word.

Value passed is a word index. The word index of the closest known superset is returned. Error return probably indicates an invalid source word index (i.e., the value passed).

#14. Get supersets of a word, given its word index.

Value passed is a word index. The return value is a two dimensional array in which the first column is of type word index of supersets and the second is a count of the differences between the superset and the set provided. The array is not necessarily complete, but gets one or more supersets that generalize the definition. For example, the supersets of "roadster" include "convertible", "sports car", and "automobile," but do not include supersets of automobile. By repeating the request with "automobile", one gets "motor vehicle", "passenger vehicle", and "land vehicle". Reiteration with the last of these produces only "vehicle", and reiteration with that produces "mechanism". Indices are for words in the dictionary and temporary definitions; temporary definitions are not created for this function, though extant ones are returned.

#15. Get subsets of a word, given its word index.

Value passed is a word index. The return value is an array of type word index of subsets. The array is not necessarily complete. Subsets of subsets are not returned. Indices are not limited to words in the dictionary; temporary definitions may be created for this function.

#16. Get parts or members, given the name of a whole object.

Value passed is a word index. The return value is an array of type word index of parts. The array is not necessarily complete as the function is not recursive. Parts of parts are not returned. Thus, if the word index of "body" is passed, the return value will have parts of a body, such as the eye, but it will not include parts of the eye, such as iris.

#17. Get the whole/assembly this is a part/member of.

Value passed is a word index. The return value is a word index of the whole. The function is not recursive, so calling the service again with the return value could itself be

a part of something else. Thus if the part is a biological iris, the return value is "eye", even though that is itself a body part.

#18. Get differential between two words, given their word indices.

A pair of word indices is passed. This functions asks what the difference between two word definitions is. The return value is a delta index. The error return value is 0, which probably indicates that the words are out of range. For an example, see integrate (item #17).

#19. Get index of word representing integral, given the delta index and the word index.

A word index and delta index are passed. The function integrates the delta index into a copy of the definition associated with word index and returns a word index. In the event that the value created is not in the dictionary, it creates a negative temporary word index and returns that (with the spelling not filled in). In the event of an error, which probably indicates either operands that are out of range or an invalid index, the return value is 0.

Example: The following to lines ask, in effect, "As a car is to a motorboat so is a rickshaw to X. What is X?"

```
dim a, b, c, X as integer          ' word indices
dim delta as integer               ' delta index
dim s as string                    '

a = primary("car", noun)           '
b = primary("motorboat", noun)     '
c = primary("rickshaw", noun)      '
delta = differential(a, b)         '
X = integrate(c, delta)            '
s = spell(X)                       '
```

In the last line, the spelling of the return value is assigned to s, which takes the value, "type of sampan". I.e., "As a car is to a motorboat, so is a rickshaw to a sampan."

#20. Set language. This function is passed the name of a language as a string. It sets the dictionary for use for all other functions. For now, the languages are English (11,000 words and growing) and German (7000 words and dormant at present). We are beginning work on a Japanese dictionary. We are anticipating working on Arabic and French dictionaries.

#21. Get error. This function returns an enumerated data type telling what the last error was. Return values include:

(1) invalid index,
(2) no such spelling,
(3) no primary definition,
(4) out of range,
(5) multiple possible answers.