**Keywords Notes – August/September 2011**

Installation 2

Un-installation 2

Installation 2

Troubleshooting 3

Testing Notes / Changes 4

Sectionizer 4

Custom Configuration (bug fix) 4

Selected Sections (bug fix) 4

Concept Finder 4

Metamap 4

Negation 5

UI Changes 5

UIMA Integration 5

Notes / Todo / Clarifications 5

Regular Expression Documentation 6

# Installation

## Un-installation

If a previous version was installed, uninstall it:

* Stop Tomcat
* Delete tomcat/work/\*
* Delete tomcat/webapps/v3nlp-server.war
* Delete tomcat/webapps/v3nlp-server
* Remove old expressionlib.sqlite database.
* Remove v3nlp-server.properties

## Installation

* If UIMA AS is not installed, download and install it.
  + Download from <http://www.apache.org/dist/uima/uima-as-2.3.1-bin.zip>
  + Unzip it somewhere, for instance, c:\tools\
  + Set UIMA\_HOME as an environmental variable (c:\tools\apache-uima-as-2.3.1/)
  + Start the UIMA AS Broker: UIMA\_HOME\bin\startBroker.bat

**Note: The broker must be running for UIMA modules in v3nlp-server to work. Always make sure it is running before starting tomcat. If errors occur running pipelines, also check this console for errors.**

* Install the v3nlp-server.war file into TOMCAT\_HOME\webapps
* If this is the first time running in an environment, you will need to copy expressionlib.sqlite to a spot tomcat has access to. TOMCAT\_HOME\data\ for instance.
* Check TOMCAT\_HOME\conf\Catalina\localhost\v3nlp-keywords.xml to insure the path to the database is correct. (from previous step, use forward slashes in paths, even on Windows Platform.)
* The properties file (v3nlp-server.properties) needs to be in the tomcat classpath. The easiest place to put this is TOMCAT\_HOME\lib.
* Check the v3nlp-server.properties file to insure it meets your environment:
  + metamapServerUrl=http://inlp.bmi.utah.edu:8080/mm-service-2011.06.1-SNAPSHOT/httpinvoker/map
  + **directoryToStoreResults** the directory to store results files (c:/temp/data for example)
  + **templateDirectory** the directory where template files are stored (c:/v3nlp-templates/ for example). **Note, files should have a .v3nlp extension. If they are in a subdirectory, they are grouped by the sub-directory name on the UI.**
  + **flapPropertiesFile** the properties file flap needs for UIMA AS.
  + **corpusSuperReaderDescriptorPath** the path to the corpusSuperReaderDescriptor.
* UIMA AS Descriptors need copied to a directory tomcat can access. For this example, they can be copied to **c:/tools/tomcat/nlp-cp** . This location is used in the step below, so if you use a different location, change the step below.
* Tomcat classpath needs to include the UIMA AS Descriptor directory. This is done by adding:  
    
  **CLASSPATH=c:/tools/tomcat/nlp-cp**  
    
  To setenv.bat in the TOMCAT\_HOME\bin directory.
* Start-up tomcat.
* Install the v3nlp-client.exe. This should launch automatically. Once in, change the configuration url, and re-start.
* Installation complete!

## Troubleshooting

* The first step is to determine if tomcat is up and running properly. Browse to <http://localhost:8080/v3nlp-server>. There should be a welcome page. If not, an error occurred during tomcat startup. Check the tomcat console and log files for errors.
* Make sure in the v3nlp-client that the configuration is pointed to the correct url.

# Testing Notes / Changes

## Sectionizer

### Custom Configuration (bug fix)

1. Run Sectionizer with default config on test data with keep annotations in results checked. Note, file **Exercise8\_01.txt** has one header flagged, SUMMARY.
2. Go back to the pipeline, click on advanced in the sectionizer, and enable custom configuration. At the start of the configuration, **below the opening headers tag**, add a new header:  
     
    **<header categories="MYCUSTOM\_HEADER" captGroupNum="0" >**

**<![CDATA[(?i)COMMENTS:]]>**

**</header>**

1. Re-run the pipeline. File Exercise8\_01.txt now has 2 headers, including the COMMENTS: one entered above.

### Selected Sections (bug fix)

1. Note, this release changes the way sections are annotated. Only selected sections will be annotated/excluded. If you select sectionizer, and do not select any sections, nothing will be sectionized.
2. Note: Previous releases tagged the area before the first section header as an “UNCLASSIFIED” section. This release removes that, only true, selected sections are annotated.

## Concept Finder

Aside from tokenizer, it only looks for regular expressions in the annotations the module directly before it created. So, for instance, if you run sectionizer for section *other*, regular expressions will only be looked for in the *other* section. If you put OParser, it would only search each individual phrase for the regular expression. The scope is dramatically narrowed depending on the module you put in front of it.

## Metamap

Like Concept Finder, Metamap only looks at previous module annotations. So, if you just run Metamap with no other services on the test documents, it will take substantial time (several minutes). If you add sentence splitter before Metamap, time should be reduced because Metamap is dealing with smaller chunks. (Same for sectionizer, however OParsing lengthens the time, presumably because we then make the remote call so many times.)

## Negation

Negation requires sentence splitter, and either a **concept (RegEx) OR UMLSConcept (Metamap)**. Negation only occurs on concept/UMLSConcept, and only on sentences that contain the concept/UMLSConcept.

## UI Changes

1. In the UI, modules have the category name (Tokenizer) now, instead of implementation name (gov.va…TokenizerImpl).
2. Templates data grid: **Load** column label changed to **Edit**.
3. On RegEx, Regular Expression library moved from the details panel to a button on the RegEx Panel. Selecting the button requires highlighting a regular expression.
4. If you close all pipelines, and try to add fetch, a service, or review results, you are now presented with an error message. (Instead of a Flex internal error.)
5. The way we saved and loaded results has been re-written. Old pipelines will no longer work. This functionality needs to be well tested.

## UIMA Integration

1. UIMA/Framework integration has begun. For Tokenizer, Sentence Splitter, and Sectionizer, when clicking the button, you can now select UIMA modules. These should run in the same manner as the existing Gate modules.
2. You CANNOT currently mix Gate and UIMA modules. (This is likely forthcoming in the next release.)
3. UIMA Sectionizer just marks sections, it does not allow for section inclusion/exclusion.

# Notes / Todo / Clarifications

1. Should phrase parser only forward noun phrases?
2. All test pipelines should go into template library.
3. PMASS Design Documents

# Regular Expression Documentation

**Note: The following documentation is from Oracle on regular expressions in Java. (http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html)**

#### Summary of regular-expression constructs

|  |  |
| --- | --- |
| **Construct** | **Matches** |
|  |  |
| **Characters** | |
| *x* | The character *x* |
| \\ | The backslash character |
| \0*n* | The character with octal value 0*n* (0 <= *n* <= 7) |
| \0*nn* | The character with octal value 0*nn* (0 <= *n* <= 7) |
| \0*mnn* | The character with octal value 0*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7) |
| \x*hh* | The character with hexadecimal value 0x*hh* |
| \u*hhhh* | The character with hexadecimal value 0x*hhhh* |
| \t | The tab character ('\u0009') |
| \n | The newline (line feed) character ('\u000A') |
| \r | The carriage-return character ('\u000D') |
| \f | The form-feed character ('\u000C') |
| \a | The alert (bell) character ('\u0007') |
| \e | The escape character ('\u001B') |
| \c*x* | The control character corresponding to *x* |
|  |  |
| **Character classes** | |
| [abc] | a, b, or c (simple class) |
| [^abc] | Any character except a, b, or c (negation) |
| [a-zA-Z] | a through z or A through Z, inclusive (range) |
| [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| [a-z&&[def]] | d, e, or f (intersection) |
| [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| [a-z&&[^m-p]] | a through z, and not m through p: [a-lq-z](subtraction) |
|  |  |
| **Predefined character classes** | |
| . | Any character (may or may not match [line terminators](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#lt)) |
| \d | A digit: [0-9] |
| \D | A non-digit: [^0-9] |
| \s | A whitespace character: [ \t\n\x0B\f\r] |
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [a-zA-Z\_0-9] |
| \W | A non-word character: [^\w] |
|  |  |
| **POSIX character classes (US-ASCII only)** | |
| \p{Lower} | A lower-case alphabetic character: [a-z] |
| \p{Upper} | An upper-case alphabetic character:[A-Z] |
| \p{ASCII} | All ASCII:[\x00-\x7F] |
| \p{Alpha} | An alphabetic character:[\p{Lower}\p{Upper}] |
| \p{Digit} | A decimal digit: [0-9] |
| \p{Alnum} | An alphanumeric character:[\p{Alpha}\p{Digit}] |
| \p{Punct} | Punctuation: One of !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~ |
| \p{Graph} | A visible character: [\p{Alnum}\p{Punct}] |
| \p{Print} | A printable character: [\p{Graph}] |
| \p{Blank} | A space or a tab: [ \t] |
| \p{Cntrl} | A control character: [\x00-\x1F\x7F] |
| \p{XDigit} | A hexadecimal digit: [0-9a-fA-F] |
| \p{Space} | A whitespace character: [ \t\n\x0B\f\r] |
|  |  |
| **Classes for Unicode blocks and categories** | |
| \p{InGreek} | A character in the Greek block (simple [block](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#ubc)) |
| \p{Lu} | An uppercase letter (simple [category](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#ubc)) |
| \p{Sc} | A currency symbol |
| \P{InGreek} | Any character except one in the Greek block (negation) |
| [\p{L}&&[^\p{Lu}]] | Any letter except an uppercase letter (subtraction) |
|  |  |
| **Boundary matchers** | |
| ^ | The beginning of a line |
| $ | The end of a line |
| \b | A word boundary |
| \B | A non-word boundary |
| \A | The beginning of the input |
| \G | The end of the previous match |
| \Z | The end of the input but for the final [terminator](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#lt), if any |
| \z | The end of the input |
|  |  |
| **Greedy quantifiers** | |
| *X*? | *X*, once or not at all |
| *X*\* | *X*, zero or more times |
| *X*+ | *X*, one or more times |
| *X*{*n*} | *X*, exactly *n* times |
| *X*{*n*,} | *X*, at least *n* times |
| *X*{*n*,*m*} | *X*, at least *n* but not more than *m* times |
|  |  |
| **Reluctant quantifiers** | |
| *X*?? | *X*, once or not at all |
| *X*\*? | *X*, zero or more times |
| *X*+? | *X*, one or more times |
| *X*{*n*}? | *X*, exactly *n* times |
| *X*{*n*,}? | *X*, at least *n* times |
| *X*{*n*,*m*}? | *X*, at least *n* but not more than *m* times |
|  |  |
| **Possessive quantifiers** | |
| *X*?+ | *X*, once or not at all |
| *X*\*+ | *X*, zero or more times |
| *X*++ | *X*, one or more times |
| *X*{*n*}+ | *X*, exactly *n* times |
| *X*{*n*,}+ | *X*, at least *n* times |
| *X*{*n*,*m*}+ | *X*, at least *n* but not more than *m* times |
|  |  |
| **Logical operators** | |
| *XY* | *X* followed by *Y* |
| *X*|*Y* | Either *X* or *Y* |
| (*X*) | X, as a [capturing group](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#cg) |
|  |  |
| **Back references** | |
| \*n* | Whatever the *n*th [capturing group](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#cg) matched |
|  |  |
| **Quotation** | |
| \ | Nothing, but quotes the following character |
| \Q | Nothing, but quotes all characters until \E |
| \E | Nothing, but ends quoting started by \Q |
|  |  |
| **Special constructs (non-capturing)** | |
| (?:*X*) | *X*, as a non-capturing group |
| (?idmsux-idmsux) | Nothing, but turns match flags on - off |
| (?idmsux-idmsux:*X*) | *X*, as a [non-capturing group](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#cg) with the given flags on - off |
| (?=*X*) | *X*, via zero-width positive lookahead |
| (?!*X*) | *X*, via zero-width negative lookahead |
| (?<=*X*) | *X*, via zero-width positive lookbehind |
| (?<!*X*) | *X*, via zero-width negative lookbehind |
| (?>*X*) | *X*, as an independent, non-capturing group |

#### Backslashes, escapes, and quoting

The backslash character ('\') serves to introduce escaped constructs, as defined in the table above, as well as to quote characters that otherwise would be interpreted as unescaped constructs. Thus the expression \\ matches a single backslash and \{ matches a left brace.

It is an error to use a backslash prior to any alphabetic character that does not denote an escaped construct; these are reserved for future extensions to the regular-expression language. A backslash may be used prior to a non-alphabetic character regardless of whether that character is part of an unescaped construct.

Backslashes within string literals in Java source code are interpreted as required by the [Java Language Specification](http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html) as either [Unicode escapes](http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html#100850) or other [character escapes](http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html#101089). It is therefore necessary to double backslashes in string literals that represent regular expressions to protect them from interpretation by the Java bytecode compiler. The string literal "\b", for example, matches a single backspace character when interpreted as a regular expression, while "\\b" matches a word boundary. The string literal "\(hello\)" is illegal and leads to a compile-time error; in order to match the string (hello) the string literal "\\(hello\\)" must be used.

#### Character Classes

Character classes may appear within other character classes, and may be composed by the union operator (implicit) and the intersection operator (&&). The union operator denotes a class that contains every character that is in at least one of its operand classes. The intersection operator denotes a class that contains every character that is in both of its operand classes.

The precedence of character-class operators is as follows, from highest to lowest:

|  |  |  |
| --- | --- | --- |
| **1** | Literal escape | \x |
| **2** | Grouping | [...] |
| **3** | Range | a-z |
| **4** | Union | [a-e][i-u] |
| **5** | Intersection | [a-z&&[aeiou]] |

Note that a different set of metacharacters are in effect inside a character class than outside a character class. For instance, the regular expression . loses its special meaning inside a character class, while the expression - becomes a range forming metacharacter.

#### Line terminators

A *line terminator* is a one- or two-character sequence that marks the end of a line of the input character sequence. The following are recognized as line terminators:

* A newline (line feed) character ('\n'),
* A carriage-return character followed immediately by a newline character ("\r\n"),
* A standalone carriage-return character ('\r'),
* A next-line character ('\u0085'),
* A line-separator character ('\u2028'), or
* A paragraph-separator character ('\u2029).

If [UNIX\_LINES](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html" \l "UNIX_LINES) mode is activated, then the only line terminators recognized are newline characters.

The regular expression . matches any character except a line terminator unless the [DOTALL](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#DOTALL) flag is specified.

By default, the regular expressions ^ and $ ignore line terminators and only match at the beginning and the end, respectively, of the entire input sequence. If [MULTILINE](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#MULTILINE) mode is activated then ^ matches at the beginning of input and after any line terminator except at the end of input. When in [MULTILINE](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#MULTILINE) mode $ matches just before a line terminator or the end of the input sequence.

#### Groups and capturing

Capturing groups are numbered by counting their opening parentheses from left to right. In the expression ((A)(B(C))), for example, there are four such groups:

|  |  |
| --- | --- |
| **1** | ((A)(B(C))) |
| **2** | (A) |
| **3** | (B(C)) |
| **4** | (C) |

Group zero always stands for the entire expression.

Capturing groups are so named because, during a match, each subsequence of the input sequence that matches such a group is saved. The captured subsequence may be used later in the expression, via a back reference, and may also be retrieved from the matcher once the match operation is complete.

The captured input associated with a group is always the subsequence that the group most recently matched. If a group is evaluated a second time because of quantification then its previously-captured value, if any, will be retained if the second evaluation fails. Matching the string "aba" against the expression (a(b)?)+, for example, leaves group two set to "b". All captured input is discarded at the beginning of each match.

Groups beginning with (? are pure, *non-capturing* groups that do not capture text and do not count towards the group total.

#### Unicode support

This class follows *[Unicode Technical Report #18: Unicode Regular Expression Guidelines](http://www.unicode.org/unicode/reports/tr18/)*, implementing its second level of support though with a slightly different concrete syntax.

Unicode escape sequences such as \u2014 in Java source code are processed as described in [?3.3](http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html#100850) of the Java Language Specification. Such escape sequences are also implemented directly by the regular-expression parser so that Unicode escapes can be used in expressions that are read from files or from the keyboard. Thus the strings "\u2014" and "\\u2014", while not equal, compile into the same pattern, which matches the character with hexadecimal value 0x2014.

Unicode blocks and categories are written with the \p and \P constructs as in Perl. \p{*prop*} matches if the input has the property *prop*, while \P{*prop*} does not match if the input has that property. Blocks are specified with the prefix In, as in InMongolian. Categories may be specified with the optional prefix Is: Both \p{L} and \p{IsL} denote the category of Unicode letters. Blocks and categories can be used both inside and outside of a character class.

The supported blocks and categories are those of *[The Unicode Standard, Version 3.0](http://www.unicode.org/unicode/standard/standard.html)*. The block names are those defined in Chapter 14 and in the file [Blocks-3.txt](http://www.unicode.org/Public/3.0-Update/Blocks-3.txt) of the [Unicode Character Database](http://www.unicode.org/Public/3.0-Update/UnicodeCharacterDatabase-3.0.0.html) except that the spaces are removed; "Basic Latin", for example, becomes "BasicLatin". The category names are those defined in table 4-5 of the Standard (p. 88), both normative and informative.

#### Comparison to Perl 5

Perl constructs not supported by this class:

* The conditional constructs (?{*X*}) and (?(*condition*)*X*|*Y*),
* The embedded code constructs (?{*code*}) and (??{*code*}),
* The embedded comment syntax (?#comment), and
* The preprocessing operations \l \u, \L, and \U.

Constructs supported by this class but not by Perl:

* Possessive quantifiers, which greedily match as much as they can and do not back off, even when doing so would allow the overall match to succeed.
* Character-class union and intersection as described [above](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html#cc).

Notable differences from Perl:

* In Perl, \1 through \9 are always interpreted as back references; a backslash-escaped number greater than 9 is treated as a back reference if at least that many subexpressions exist, otherwise it is interpreted, if possible, as an octal escape. In this class octal escapes must always begin with a zero. In this class, \1 through \9 are always interpreted as back references, and a larger number is accepted as a back reference if at least that many subexpressions exist at that point in the regular expression, otherwise the parser will drop digits until the number is smaller or equal to the existing number of groups or it is one digit.
* Perl uses the g flag to request a match that resumes where the last match left off. This functionality is provided implicitly by the [Matcher](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Matcher.html) class: Repeated invocations of the [find](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Matcher.html#find%28%29) method will resume where the last match left off, unless the matcher is reset.
* In Perl, embedded flags at the top level of an expression affect the whole expression. In this class, embedded flags always take effect at the point at which they appear, whether they are at the top level or within a group; in the latter case, flags are restored at the end of the group just as in Perl.
* Perl is forgiving about malformed matching constructs, as in the expression \*a, as well as dangling brackets, as in the expression abc], and treats them as literals. This class also accepts dangling brackets but is strict about dangling metacharacters like +, ? and \*, and will throw a [PatternSyntaxException](http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/PatternSyntaxException.html) if it encounters them.

For a more precise description of the behavior of regular expression constructs, please see [*Mastering Regular Expressions, 2nd Edition*, Jeffrey E. F. Friedl, O'Reilly and Associates, 2002.](http://www.oreilly.com/catalog/regex2/)