Abstract
We have implemented BioC natural language preprocessing pipelines in two popular programming languages: C++ and Java. They are largely based on the well-known natural language processing tool sets, MedPost and Stanford. Tools integrated in the pipelines include sentence segmentation, tokenization, part-of-speech (POS) tagging, lemmatization and sentence parsing. These pipelines can be easily integrated along with other BioC programs into new BioC compliant text mining systems. We converted the NCBI disease corpus to BioC format and all the tools described here were run on this corpus to demonstrate their functionality. Code and data can be downloaded from: http://bioc.sourceforge.net.

Introduction
BioC (1) is a new format and associated source code libraries for sharing text and annotations. This allows for the simple and convenient processing of text corpora. With the provided libraries, it is straightforward to incorporate BioC code into existing programs to read in data from BioC formatted input files and write out results to BioC formatted output files.

Text preprocessing is integral to virtually all natural language processing (NLP) systems. It reformats the original text into meaningful units that contain important linguistic features before performing subsequent text mining strategies. Generally, several preprocessing steps need to be performed, such as sentence segmentation, part-of-speech (POS) tagging, and sentence parsing. Poor text preprocessing performance will have a detrimental effect on downstream processing. Compared to general English texts, a particularly challenging aspect of preprocessing biomedical text is the wide variety of domain-specific technical terminology encountered.

Our contribution to the interoperability track of the BioCreative IV challenge is BioC text-preprocessing pipelines in C++ and Java. These tools integrate a selection of state-of-the-art text preprocessing tools and produce corresponding text analyses in the BioC XML format. The integrated tools are considered representative in the target domains and have been reported to
yield competitive results on biomedical texts. Instead of being all-inclusive, the intension of our work is to provide essential text preprocessing functionalities to BioC users. The processing is implemented in a flexible way so that users can incorporate other tools according to their needs. The implementation is freely available to the NLP and text mining research communities, and is released as open source software that can be downloaded.

While many researchers have their own favorite natural language preprocessing tools, it is useful to have examples of commonly used tools available in the BioC format. We use the NCBI disease corpus as a model corpus. The outputs of these programs provide examples of how the BioC format links the results of different tools in an interoperable and integrated fashion. This demonstrates how different programs can use and produce data in a consistent format, regardless of their implementation language.

C++
C++ is a high performance, compiled language with very good execution time and memory usage performance. Figure 1 shows the overall flow of our pipeline. The different tools, sentence segmentation, tokenization, part-of-speech tagging, and dependency parsing, are implemented as separate stand-alone programs. They are represented by the inner boxes in the Figure 1. This is convenient if the results of only one, or a few, of the tools are desired.

The pieces of the C++ pipeline were drawn from the MedPost (2) collection of natural language processing tools. One challenge was that MedPost normalizes some of the results. For example, multiple spaces between tokens in a sentence would be normalized to a single space. This had to be taken into account when determining the offset to the original text as encouraged by the BioC format.

Sentence Segmenting and Tokenizing
MedPost uses a rule based approach to sentence segmenting. A recent paper using a token lattice design pattern and the adapted Viterbi algorithm achieved a tokenization accuracy of 92.4% compared to MedPost’s 92.9% (3).

Parts-of-Speech
The principle MedPost tool is a high accuracy part-of-speech tagger trained on a MEDLINE corpus (2). Using its own tag set, it achieves 97.43% accuracy on a 1000 sentence test set. It achieves 96.9% accuracy using the Penn Treebank tag set. In addition to part-of-speech, MedPost can tokenize text and segment sentences. It has been widely used and is included in the popular LingPipe natural language toolkit1. While better results have been achieved on a specialized medical sublanguage, MedPost still provided a valuable baseline (4).

1 http://alias-i.com/lingpipe/
Lemmatization
The C++ pipeline does not include its own lemmatizer. For the BioNLP 2013 shared task, the C++ pipeline was run to prepare supporting material (6). The Java Biolemmatizer (5) was also run, to complete our supporting material package. This demonstrates a benefit of a language neutral data format. Results can be easily combined regardless of the implementation language.

Dependency Parse
MedPost also includes a wrapping of the C&C dependency parser (7). In addition to the expected head and dependent tokens, some relations include a type token. This is described in the cnc.key file.

Java
Figure 1 also presents the detailed annotation flow of the Java implementation of our BioC text-preprocessing pipeline, which includes text tokenization, sentence segmenting, POS tagging, lemmatization and sentence parsing. The pipeline takes as input a BioC collection. Preprocessing is then invoked for each BioC passage on which the integrated tools are performed sequentially to produce corresponding text analyses. In the end, the generated annotations, along with the BioC collection information, are inserted into a BioC data structure to produce an annotated BioC XML file.

Sentence Segments
An efficient sentence segmenter, DocumentPreprocessor, is used to produce a list of sentences from a plain text. It is a creation of the Stanford NLP group using a heuristic finite-state machine.
that assumes the sentence ending is always signaled by a fixed set of characters. Tokenization is performed by the default, rule-based tokenizer of the sentence segmenter, PTBTokenizer, prior to the segmenting process in order to divide text into a sequence of tokens. The “invertible” option of the tokenizer is invoked to ensure that multiple whitespaces are reflected in token offsets so that the resulting tokens can be faithfully converted back to the original text. Sentence segmentation is then a deterministic consequence of tokenization.

**POS tagging**
The MaxentTagger based on a maximum entropy model is used for part-of-speech tagging. The MaxentTagger is also the default POS tagger used by the Stanford parser before parsing the text.

**Lemmatization**
BioLemmatizer (5) is used to perform the morphological analysis of biomedical literature. It has been demonstrated that the BioLemmatizer achieves the best lemmatization performance on biomedical texts and contributes to biomedical information retrieval/extraction tasks. The word form and the part-of-speech of a token are required as input to the BioLemmatizer to retrieve the corresponding lemma.

**Sentence parsing**
The POS-tagged sentences are then submitted to the Stanford unlexicalized natural language parser (8) to analyze the syntactic and semantic structure of the sentences. The Stanford parser has been reported to be one of the state-of-the-art parsers in terms of speed and accuracy (9,10). When applied to the biomedical domain, it has successfully helped to extract various types of biological relations and events from the literature (11,12) and identify medical treatment terms from randomized clinical trial (RCT) reports (13). The Stanford parser is parameterized to return both Penn Treebank parse tree and dependency representations for each sentence. While the flat version of the Penn Treebank parse tree is directly encoded into the XML, the dependency representations are recorded directly in BioC as grammatical relations between participating tokens referred to by their token IDs. Because of the Unicode compatibility of the integrated tools, the pipeline should work well over texts encoded in both ASCII and Unicode. The pipeline currently performs an end-to-end annotation from text tokenization to sentence parsing. However, even though sentence parsing is useful for tasks such as question answering or relation extraction, it is not often considered by tasks like named entity detection or concept recognition. In addition, due to the constituent-based parsing nature of the Stanford parser, sentence parsing accounts for most of the execution time of the pipeline. Therefore, we plan to provide more flexibility to the pipeline users in the next release to allow them to choose the annotation steps according to their needs.

**Comparison**
We compared the output of the C++ and Java BioC tools by running them on a small set of 10 PubMed references. We performed a detailed comparison of the output of both pipelines. The
BioC-formatted output could be easily handled by the same BioC-compatible program. The segmented sentences were identical, as expected, since the set of abstracts was known not to have any challenging cases. The tokens and parts-of-speech tags were very similar, with only expected differences. For example, MedPost and Stanford parsers make different decisions on splitting tokens containing hyphens (-) or slash (/) characters.

The dependency graphs were verified by processing them with the same program to produce visual graphs based on the graph description language DOT (14). Again, using the same program to produce graphs is a benefit of taking advantage of the BioC format for the output of both pipelines.

**Application of BioC NLP tools on the NCBI disease corpus**

We used the NCBI disease corpus to provide a more rigorous evaluation of the BioC NLP tools. The NCBI disease corpus (15,16) is a manually annotated resource for disease name recognition and normalization in biomedical text, which comprises a collection of 793 PubMed abstracts and a total of 6,892 disease mentions, which further correspond to 790 disease concepts mapped to MeSH descriptors or OMIM identifiers. It was completed by a team of fourteen annotators in three annotation rounds and provides a high-quality, reliable and consistently annotated resource. This resource was used to develop a highly effective disease normalization method (17). In order to make the NCBI disease corpus more accessible, and to promote its usage for other related biomedical information extraction tasks, the collection was converted to BioC-XML format, and is used here as a model test case to run the BioC NLP pre-processing pipeline tools.

Figure 2 illustrates the disease mention and concept annotation in the NCBI disease corpus expressed in BioC XML format. Each annotation contains the textual mention with the appropriate location information, given with the precise document offset and length. Since the annotation of the same textual string is two-facetted, infons are used to express the semantics of the annotation: the infon key="EntityType” is used to distinguish the four disease categories as specified by the annotators of the corpus, the infon key="Nomenclature” is used to distinguish the correct terminology resource selected for the annotation, and the infon key="ConceptID” specifies the unique concept identifier for the textual mention. The infon key-value pairs in the annotation elements correspond to the original corpus format, the PubTator format (18). The tool to convert PubTator annotation data to BioC is also available for download.

Next, we ran both C++ and Java pipelines on the BioC-formatted NCBI disease corpus, thereby enriching this resource with machine-assisted annotations and basically pre-processing the data ready for use by any BioC compliant application. The machine-assisted annotations consist of: sentence segmentation, tokenization and POS tagging processed using both MedPost and Stanford parsers, lemmatization using BioLemmatizer, dependency parsing using both C&C and
Stanford parses, as well as abbreviation definition detection using Ab3P, Schwartz & Hearst and NatLAb algorithms(19).

Figure 2. Illustration of annotations in the enriched NCBI disease corpus, manual annotations of disease mentions and concepts, and BioC-tools produced annotations for text pre-processing.
Summary
We have implemented BioC natural language preprocessing pipelines in two popular programming languages: C++ and Java. They are largely based on well-known natural language processing tool sets, MedPost and Stanford. A benefit of BioC is the interoperability between tools written in different programming languages. Using these tools, it is straightforward to use the Stanford tools in a C++ pipeline, or MedPost in a Java pipeline.

The BioC text preprocessing pipelines serve as starting points for automatically annotating BioC collections. The pipelines are implemented in a flexible manner enabling researchers to integrate other tools as needed. Alleviating the burden of interoperability challenges will encourage the development of novel approaches allowing improved natural language processing performance. We also present the NCBI disease corpus in the BioC XML format. As a test case application for the pipelines, the corpus is used as input to all BioC NLP tools in both C++ and Java. The result is a rich collection of annotations that combines manual annotations (for disease entity mentions and concept normalization) with tool-provided machine annotations. This collection, along with the BioC tools, is available to the community from the BioC website: http://bioc.sourceforge.net.

Acknowledgements
This research was supported by the Intramural Research Program of the NIH, National Library of Medicine.

References


