Adept API
Readme

Adept API Readme, for Adept API 2.0

Contents

1 Introduction

Under the DARPA DEFT program, BBN developed a scalable, multi-layered, plug-and-play framework named Adept with capabilities that enable rapid integration of information extraction and other natural language processing (NLP) algorithms. The Adept framework incorporates standardized definitions for NLP data structures such as Token, Chunk, Entity, Relation, Mention, etc. and provides a uniform catalog of algorithm interfaces that facilitate semantic interoperability, algorithm fusion, and Big Data processing in a scalable, parallel computing environment. Additionally, the Adept framework provides interfaces for all DEFT algorithms, designed comprehensively to support sentence-level, document-level, and corpus-level data processing, such that algorithms in any of these categories can be easily plugged into the framework.

1.1 This Readme

This document contains a description of the Adept API and its data structures and algorithm interfaces. It explains how to implement, invoke, and test your algorithm and provides example code. It addresses special topics about reading different input file formats, and integrating code from languages other than Java.

1.2 Contact

For questions or comments, please contact deft-software-requests@bbn.com.

1.3 Adept API Contents

The Adept API contains the following top-level directories. Start with the example, javadocs and adept-api folders to get an overview of the project. The other source-code folders provide specialized capabilities which are described later in this document.

- adept-api
- adept-thrift-api
- adept-thrift-java
- adept-api-mappers
- adept-mapreduce
- adept-pipeline
- adept-thrift-cpp
- adept-thrift-perl
- example
- javadocs
- license

1.4 Licensing

1.5 Getting Started

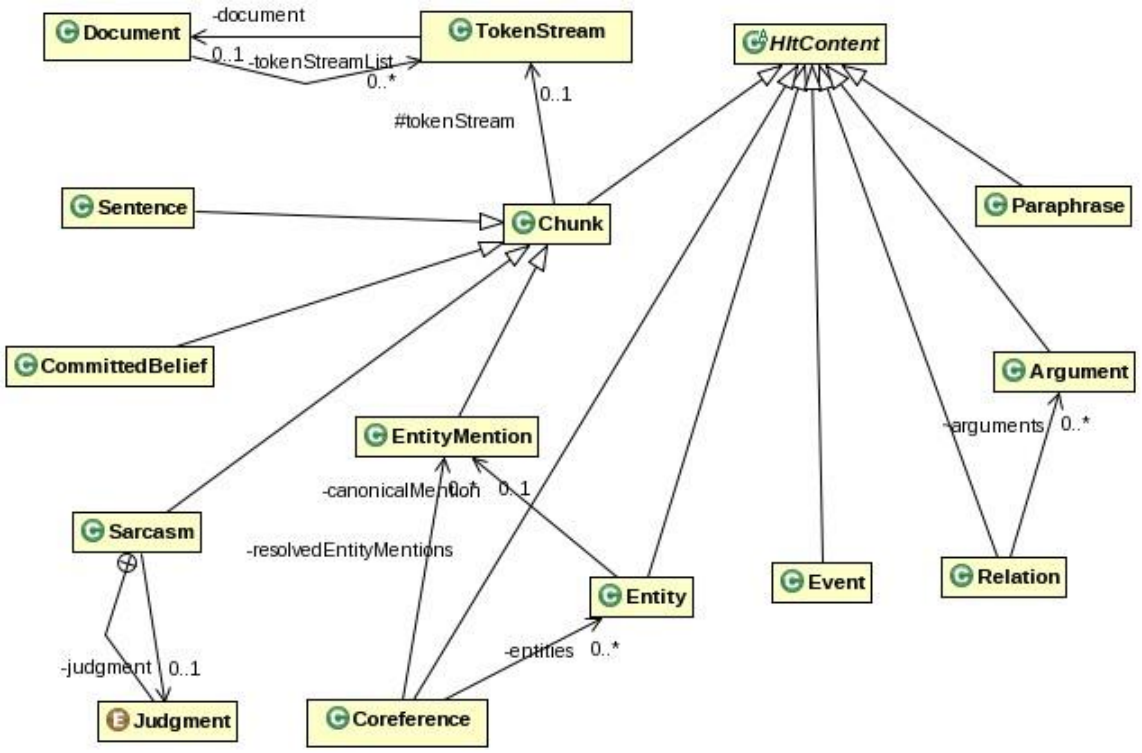
System requirements:

- Git version 1.7.4.1 or later.
- Maven version 3.0.5 or later.
- Java(TM) SE Runtime Environment (build 1.6 or later).

Building from source:

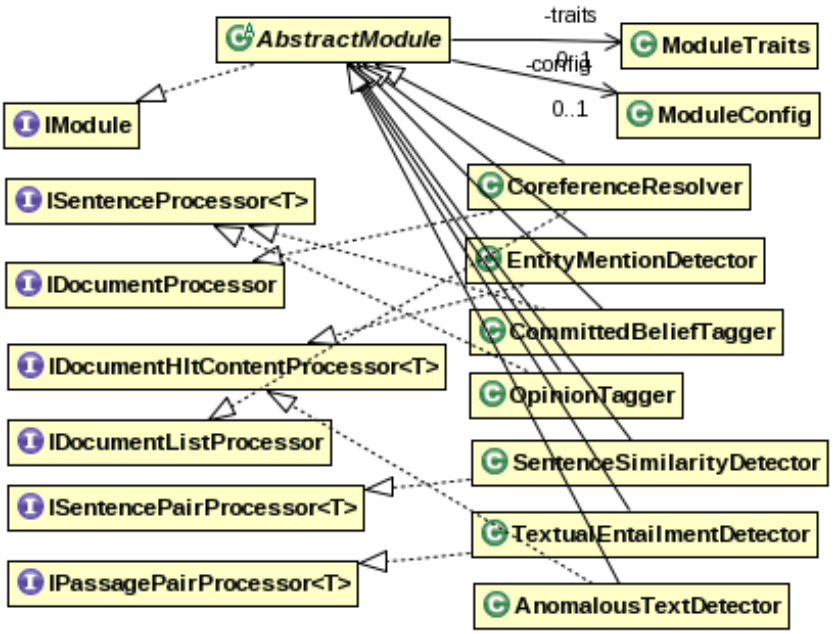
- Create a directory for Adept and cd into it.
- git clone https://github.com/BBN-E/Adept.git
- cd Adept
- mvn clean test-compile

2 Adept Data Structures



- **Key features**
  - Provides standardized NLP data structures to represent text documents and information extraction output
  - Specifies global definition of HLT terms, such as, entity mentions, relations, events across multiple algorithms

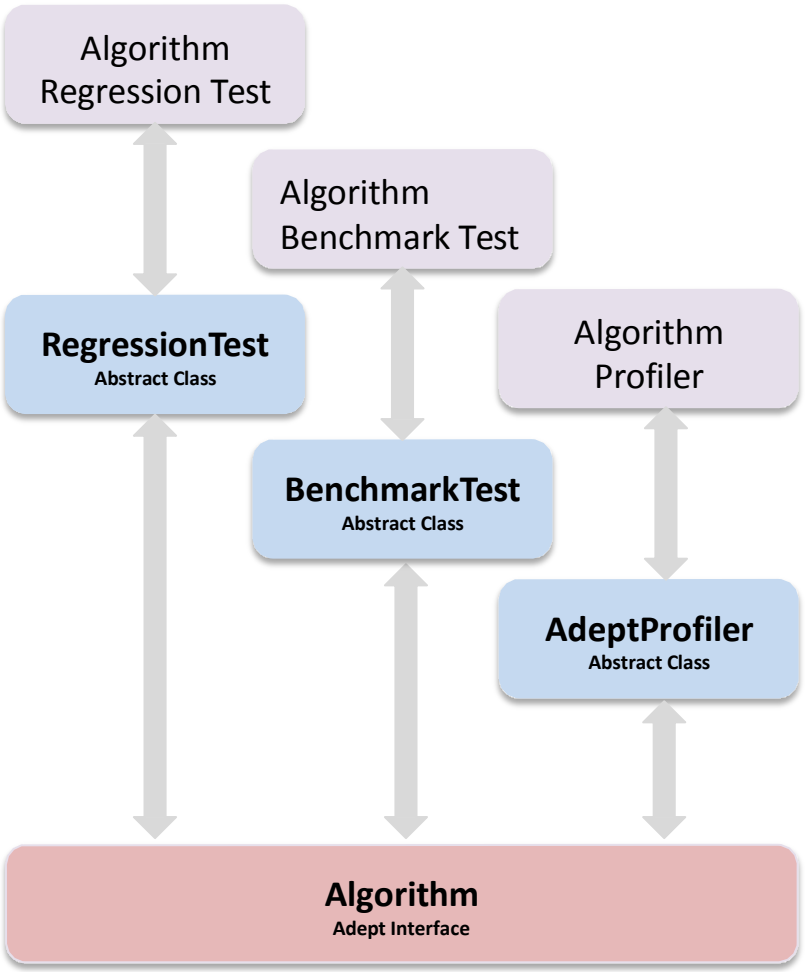
3 Adept Algorithm Interfaces



- **Key features**
  - Imposes uniform contract for all algorithms to implement; e.g., Activate, Deactivate, Process
  - Separates contract on Process method for each algorithm; e.g., sentence vs. document vs. corpus

4 Testing and Profiling

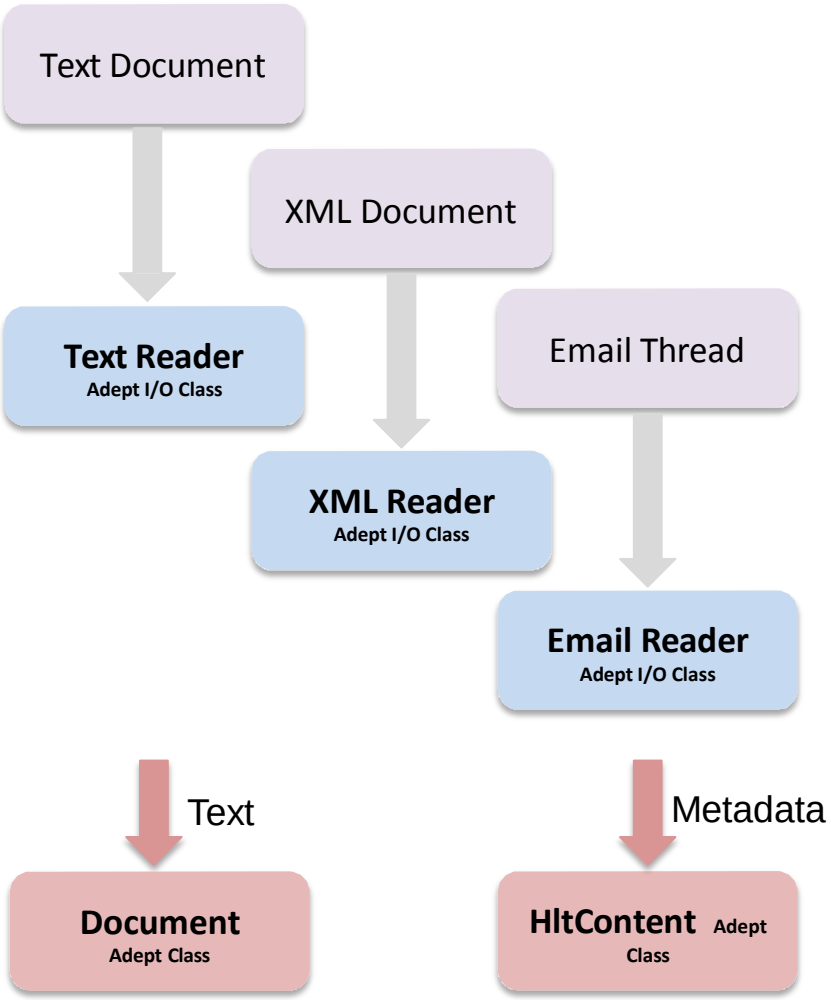
Adept provides tools and procedures for users to rapidly develop and deliver tests.



- **Key features**
  - File reading/writing abstraction
  - Built-in determinism testing for regression
  - Input/output serialization utilities
  - Scoring framework for performance benchmarking

5 Data Readers

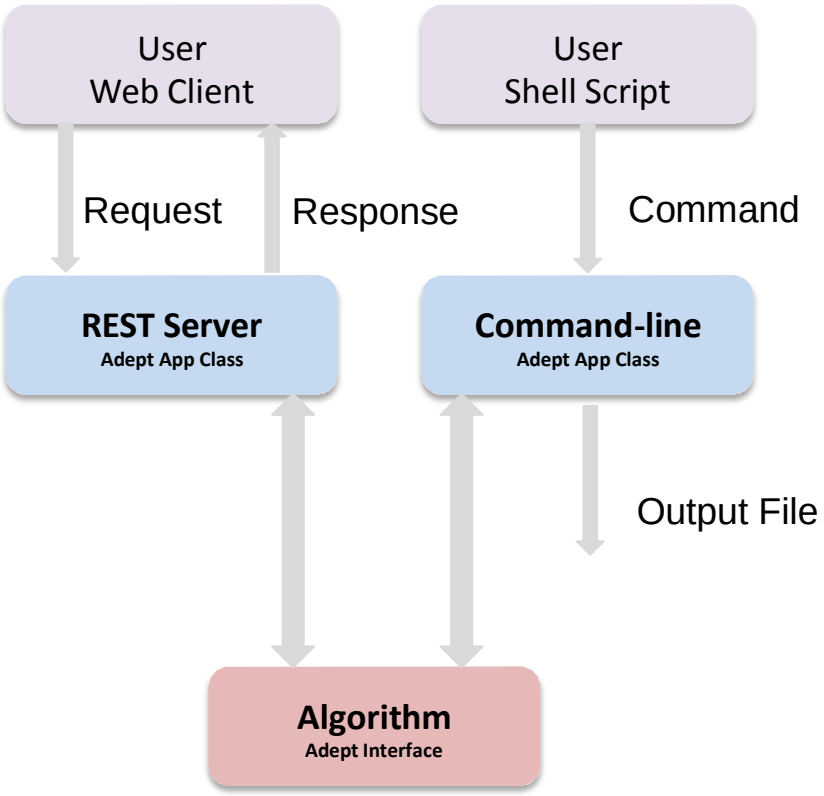
Adept provides tools for ingesting certain LDC-released corpora as well as forum and email formatted data. See the "Adept API Specifications v1.8" for specifics.



- **Key features**
  - Parses Plain Text and XML formatted data
  - Loads annotations into Adept HLT classes (e.g. EntityMention)
  - Handles parsing of in-band metadata (e.g. author name in email thread)
- **Notes**
  - In-band metadata may lead to token offset issues when loading LDC annotations
  - Mapping of token offsets required to conform LDC-annotated data to Adept data definition

6 Algorithm Invocation

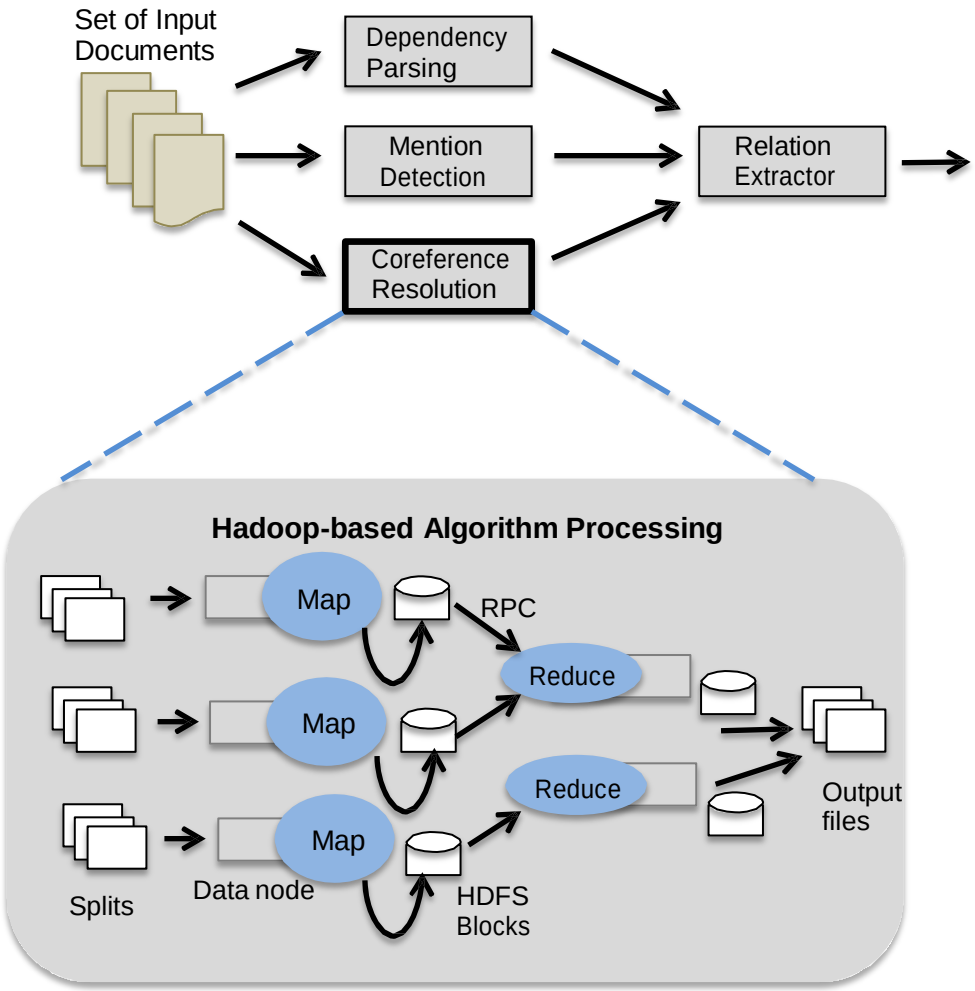
Adept provides flexibility for the user to run algorithms as command-line apps or as RESTful web services.



- **Key features**
  - REST API abstracts functions (POST, GET) from the underlying algorithm
  - Allows client to POST/GET data using cURL utility
  - Enables rapid integration into existing web services
  - Simple Unix-like command-line tool with options to run on a single document or a collection
- **Notes**
  - Remote invocation capability via web services allows easy, language and platform-independent integration

7 Parallelization

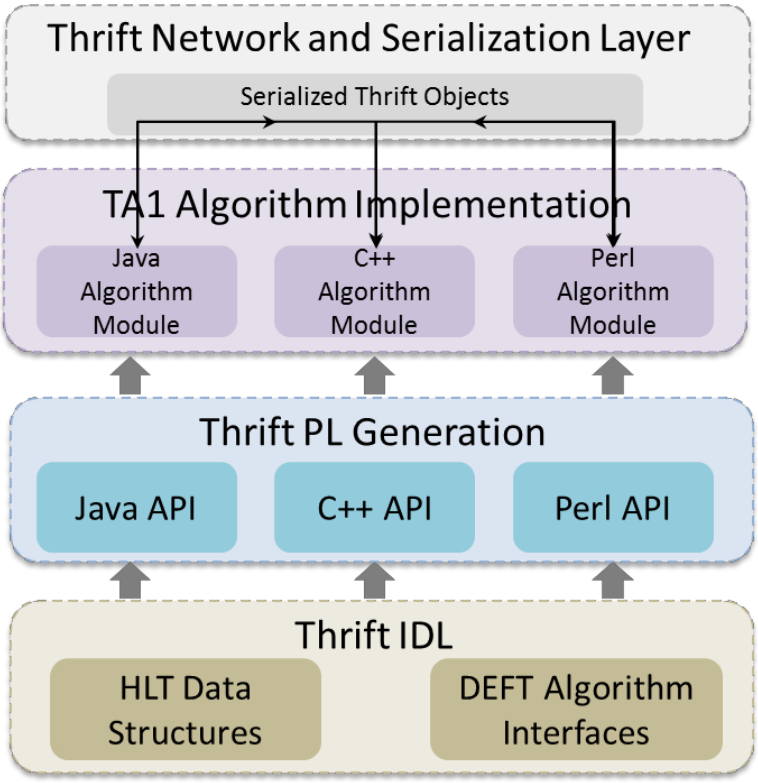
Adept provides tools to run the algorithms in the parallel processing framework Hadoop. This example diagram shows four algorithms combined in a pipeline and producing Relation objects as output.



- **Key features**
  - REST API for loading batch of documents on HDFS
  - Abstracted mappers and reducers from algorithm implementation
  - Configuration options for creating a pipeline from interdependent algorithms
- **Notes**
  - Most algorithms are single-threaded and require process-level parallelization

8 Programming Language Interoperability

Adept provides interfaces using Apache Thrift for supporting development in Java, C++, Python, and Perl. From Wikipedia: "Thrift is an interface definition language and binary communication protocol that is used to define and create services for numerous languages. It is used as a remote procedure call (RPC) framework and was developed at Facebook for 'scalable cross-language services development'. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly" among multiple programming languages.



- **Key features**
  - Java algorithms can communicate with Perl/C++ modules
  - Exchange low footprint, serialized Thrift objects
  - Run as RESTful services via Thrift networking layer

9 Example Algorithms

Two examples are provided to show simple usage of the Adept API.

9.1 ExampleNamedEntityTagger

This simple algorithm implements a simplistic English-language Named Entity Recognizer. It is only to be used as an example of how to build NLP modules in terms of what classes to extend, methods to define, and so on (and not as an example of an accurate approach to named entity tagging). The algorithm extends the class adept.module.NamedEntityTagger.

The algorithm naively assumes that any word inside a sentence which starts with an uppercase letter is a Named Entity.

It is accompanied by a simple test class called ExampleNamedEntityTaggerTest.java.

9.2 ExamplePassageTokenizer

This example demonstrates use of a built-in tokenizer when creating an adept.common.Document object.

It is implemented by a simple test class called ExamplePassageTokenizerTest.java.

10 How to Add Your Own Algorithm

Implementing an algorithm requires creating the algorithm itself, testing it, and then deploying a way for clients to run it. The Adept API offers support for two different kinds of testing, regression and benchmark. It also supports two different kinds of deployment, as a command-line application or as a REST API.

The Adept API provides superclasses for use in implementing an algorithm and its tests and tools:

1. Algorithm – a processor which turns Document objects into HltContentContainer objects.
2. Regression Test – a test that the algorithm returns the same output after a change to its code or data.
3. Benchmark Test – a test which applies a scoring metric to the output of the algorithm.
4. Command-line Application – a runnable console application for the algorithm.
5. REST API – a runnable HTTP interface for the algorithm.

Listed here are the steps to implement an algorithm using the Adept API, as shown in the table below.

a) Import Package
   At the top of your Java file, import the adept package shown in this column.

b) Extends Class
   When defining your class, add "extends" and the name of the class shown in this column.

c) Implement Methods
   Add an implementation for each method shown in this column. The purpose of each method is described here.
   - activate() – put your algorithm initialization code here.
   - process() – put your algorithm processing logic here.
   - deactivate() – put your algorithm cleanup code here.
   - doActivate() – put your application initialization code here.
   - doProcess()– put your application processing logic code here.
   - doScore() – put your benchmark scoring metric here.
   - doDeactivate() – put your application cleanup code here.
   - main() – put your REST client top-level code here.

	Import Package	Extends Class	Implement Methods
Algorithm	adept.module	AbstractModule	activate() process() deactivate()
Regression Test	adept.utilities	RegressionTest	doActivate() doProcess() doDeactivate()
Benchmark Test	adept.utilities	BenchmarkTest	doActivate() doProcess() doScore() doDeactivate()
Command-line Application	adept.utilities	CommandLineApp	doActivate() doProcess() doDeactivate()
REST API Client	adept.restapi	RestClient	main()
REST API Servlet	adept.restapi	RestServlet	doActivate() doProcess() doDeactivate()

11 How to Read Input Files

The Adept API provides multiple ways to read input files, and can be extended by users to handle additional file formats.

11.1 With the DeftReaderApp

The DeftReaderApp (package adept.utilities) is able to take text files as input, and produce serialized HltContentContainer objects as output.

It requires an input argument, either a single file or directory, and an output argument, either a single file or directory. Other arguments are optional.

The app supports four input formats:

- SGM (as exemplified by .sgm files in LDC's LDC2009T13 with one <DOC> element per file),
- text,
- ERE (osc or proxy) (as exemplified by the files in data/source/osc/proxy in LDC's LDC2013E64),
- CoNLL 2011.

The full list of arguments is as follows:

Argument	Description
-h, --help	Displays list of available arguments.
-i, --input_file <filename>	Single input file to read and convert. Use in conjunction with -o, --output_file <filename>
--input_directory <directory>	Directory containing files to read and convert. Use in conjunction with --output_directory <directory>. All files in directory are assumed to be of the same type.
--input_format	Specifies input file format. Valid options are not case sensitive: 'text,' 'sgm,' 'ere,' 'conll.' 'Ere' works for both osc and proxy formats. If omitted, 'text' is assumed by default.
--input_language <language>	Defaults to "English." Only supports English at this time
-a, --annotation_file <filename>	Specifies single annotation files corresponding to input. Use in conjunction with -i and -o.
--annotation_directory <directory>	Specifies directory containing annotation files corresponding to inputs. Use in conjuction with --input_directory <directory> and --output_directory <directory>.
-o, --output_file <filename>	Name of single output file to write serialization to. Use in conjunction with -i, --input_file <filename>.
--output_directory <directory>	Location of directory to write all output files to. Use with --input_directory <directory>. Names are automatically generated based on input names.
--output_format	Specifies the output file format. Valid options are not case sensitive: 'json,' 'xml,' 'binary.' If omitted, 'xml' is assumed by default.

Argument	Description
--table_output	Write a tabular output for each HltContentContainer along with the serializations.
-v, --verbose	Prints additional information.
--version	Prints product version and exists.

Usage example with MAVEN:
mvn exec:java -Dexec.mainClass="adept.utilities.DeftReaderApp" -Dexec.args="-i someOSCFile.txt -a someOSCAnnotation.ere.xml -o serialized.xml --input_format ere --output_format json"

11.2 With Reader API

The Reader API (Reader.java, package adept.io) provides methods for reading text and annotations from ERE and CoNLL 2011 files and producing HltContentContainers.

- **HltContentContainer EREtoHltContentContainer(String EREPath, String AnnPath)**
  - EREPath must be a path to a valid ERE file. AnnPath is the path to the corresponding annotation file.
  - Produces container for ERE files, either in OSC or Proxy format.
  - If the AnnPath does not correspond to a file, a container without annotation information is returned.
- **HltContentContainer CoNLLtoHltContentContainer(String filePath)**
  - filePath must be the path to a valid CoNLL file
  - Returns a container object derived from the CoNLL file

12 Appendix: The Thrift Adept Module

BBN's Thrift module allows developers to interface with Adept while working in the programming language of their preference. It consists of a set of data structures which emulate the Adept API classes in multiple languages, as well as interfaces developers can use to process their data with cross-language algorithms.

Any distributed application includes individual components, often written in different languages and hosted in multiple locations, which must communicate quickly and efficiently. Apache Thrift is a communication framework that enables cross-language remote procedure calls and serialization. Apache Thrift supports embedded, mobile, web, and server environments and a host of languages ranging from JavaScript to C++.

As an introduction to the Thrift module, this guide will explain how to get started as a Perl developer trying to interface with BBN's Adept module.

12.1 Thrift and Adept

The Thrift module attempts in many ways to emulate the Adept API created for Java developers, and the Adept API should be used as a resource to help you understand the Thrift module.

A key difference between the Adept API and the Thrift module is that whereas the Adept classes have been defined with fields and methods, Thrift objects consist only of fields.

A Thrift struct is conceptually similar to a C struct -- a convenient way of grouping together (and encapsulating) related items. Structs translate to classes in object-oriented languages.

12.1.1 Importing Modules

In Perl, every thrift.adept.common struct is defined in the same module, thrift::adept::common::Types. Likewise, every thrift.adept.module struct is in the thrift::adept::module::Types module. Every interface has its own module. In order to use an Interface like ISentenceProcessor with some thrift.adept.common structs, the following must be at the top of the perl file:

```
use lib '/<path to thrift module>/src/main/perl/';
use thrift::adept::common::Types;
use thrift::adept::module::ISentenceProcessor;
```

12.1.2 Key API Differences

For the most part, the Thrift module objects mirror the classes in the Adept API Specifications. Here are some key differences:

-The objects CharOffset and TokenOffset have fields names ‘beginIndex' and ‘endIndex'. This is because ‘begin' and ‘end' as they are specified in the Adept API are special keywords in Thrift.

-Anywhere a field in the Adept API is defined as a BoundedList<Pair<Object1, Object2>>, the Thrift module instead uses a Map<Object1, Object2>.

-TokenStreams, rather than extending the class ArrayList<Token> have a field named ‘tokenList' which is a list of tokens.

12.2 Constructing Objects

Take for example the Adept class Passage - according to the Adept API Specifications, it has 6 fields (contentType, sequenceId, tokenOffset, tokenStream, id, value) and 15 methods including its constructor. In Perl, the thrift::adept::common::Passage object has all of the same fields, but no methods. To construct a Perl version of a Passage, rather than use a constructor like the one defined in the Adept API, simply instantiate a new passage and then set the fields. The following example constructs a Passage assuming a TokenOffset and TokenStream have been previously defined:

```
my $passage = new thrift::adept::common::Passage();
$passage->sequenceId(24);
$passage->tokenOffset($tokenOffset);
$passage->tokenStream($tokenStream);
```

Getting the values of fields follows a similar syntax, where the name of the field desired is put in single quotes:

```
my $tokenOffset = $passage->{‘tokenOffset'};
my $channelName = $passage->{‘tokenStream'}->{‘channelName'};
```

Note in the above example that ‘channelName' is a field in a TokenStream struct, and ‘tokenStream' is a field in a Passage.

With a few exceptions, the names of fields in the Thrift module are identical to those found in the Adept API specifications.

To control the kind of objects passed through the Thrift interfaces, some fields are required to be present in a Thrift object in order for the interface to accept the object. Every field specified in the Adept API for an equivalent class's constructor method is a required field and must be set in order to pass the object through a Thrift interface. E.g., the Adept API specification lists:

```
Passage(long sequenceId, TokenOffset tokenOffset, TokenStream tokenStream)
```

as a Passage's constructor method. Thus a Thrift Passage needs to have its sequenceId, tokenOffset, and tokenStream fields defined in order to be passed through an interface.

12.2.1 Constructing Enums

To set a Perl enum, use the normal setting technique with the full name of the enum value, or using the appropriate enum number. E.g.,

```
$tokenStream->contentType(thrift::adept::common::ContentType::TEXT);
$tokenStream->speechUnit(1);
```


When you get an enum value, you simply receive an integer corresponding to the value as defined in the enum.

```
my $contentType = $tokenStream->{'contentType'};
print "Content Type: $contentType\n";
```

This will print 'Content Type: 0' using the previously defined $tokenStream and its contentType field. The numbers corresponding to each enum value can be found at the top of the Types Perl module file in which the enum is defined. The ordering of enum fields as listed in the Adept API specification is not the same as the field ordering found in the actual implementation, so be sure to check the Types file to confirm the field numbers.

12.2.2 Lists

List fields can be created like any other Perl list, but when you set the field, rather than try to set the list itself as a field value, use a reference to the list. For example:

```
my $hltContentContainer = new thrift::adept::common::HltContentContainer();
my @passages;
for (my $i=0; $i<5; $i++ ) {
  push(@passages, $passage);
}
$hltContentContainer->passages(\@passages);
```

12.2.3 Unions

Some of the interfaces defined in the Adept API use templates in their arguments, which Thrift doesn't allow. As an example, according to the Adept API, the ISentenceProcessor interface has one method called 'process' that takes in a Sentence and sends back a list of objects that extend HltContent. Because Perl objects can't extend the way Java classes can, the Thrift module version of 'process' returns a list of HltContentUnion objects. A union is a type of struct where only one of its fields can be set. The HltContentUnion has a field for every class that extends HltContent in the Adept API. The fields can be set and read like any other struct, where the name of the field is the name of the object type being used. Example:

- $hltContentUnion->sarcasm($sarcasm);
- my $sarcasm = $hltContentUnion->{'sarcasm'};

In general, if the Adept API lists a method argument as "T extends <Adept Class>" the Thrift module replaces <Adept Class> with <Adept Class>Union.

12.3 Client/Server Interfaces

Thrift provides libraries that allow you to set up and run a client or server in any of the supported languages. These libraries must be visible to client or server code, and BBN has provided them when you install the Thrift module via Maven. They can be located in:

<path to local Maven repo>/apache/thrift/0.9.0/thrift-0.9.0/lib/perl/lib/

Service definitions are semantically equivalent to defining an interface (or a pure virtual abstract class) in object-oriented programming. The Thrift compiler generates fully functional client and server stubs that implement the interface.

12.3.1 Clients

Creating a client and connecting it to a server requires access to the Thrift libraries, the appropriate interface, and the socket the server is connected to. Take this example, which connects to socket 9090 on the local machine, to make calls through an ISentenceProcessor:

```
use Thrift;
use Thrift::BinaryProtocol;
use Thrift::Socket;
use Thrift::BufferedTransport;
use thrift::adept::module::ISentenceProcessor;
use thrift::adept::common::Types;

my $socket  = new Thrift::Socket('localhost',9090);
my $transport = new Thrift::BufferedTransport($socket,1024,1024);
my $protocol = new Thrift::BinaryProtocol($transport);
my $client  = new thrift::adept::module::ISentenceProcessorClient($protocol);
$transport->open();
```

The above code instantiates a client communicating with a server on socket 9090 of the local machine. Note that the nomenclature to instantiate a client for the ISentenceProcessor is ISentenceProcessorClient.

In this example the client calls ISentenceProcessor's ‘process' method by sending it a Sentence, and it receives a reference to a list of HltContentUnions.

```
my $hltContentUnionList = $client->process($sentence);
```

Recall that the $hltContentUnionList must be dereferenced (@$hltContentUnionList) in order to be used like a normal Perl list.

Once the application is done making calls to the server incorporate the following line to close the connection and allow other clients to connect to the server:

```
$transport->close();
```

At that point, the client is ready to access a compatible server.

12.3.2 Servers

Creating a Server to process calls requires access to the Thrift libraries, and a "handler" which is simply an implementation of whatever interface your server uses.

A Server carries out the following actions:

1. Create a transport
2. Create input/output protocols for the transport
3. Create a processor based on the input/output protocols

4. Wait for incoming connections and hand them off to the processor. Take this example, which connects to socket 9090 on the local machine, to receive calls for an ISentenceProcessor:

```
use Thrift::Socket;
use Thrift::Server;
use thrift::adept::module::ISentenceProcessor;
use thrift::adept::common::Types;

my $handler = new example::ExampleHandler();
my $processor = new thrift::adept::module::ISentenceProcessorProcessor($handler);
my $serversocket = new Thrift::ServerSocket(9090);
my $forkingserver = new Thrift::ForkingServer($processor, $serversocket);
print "Starting the server...\n";
$forkingserver->serve();
```

Note the processor nomenclature appends "Processor" to the end of the interface name. A Processor encapsulates the ability to read data from input streams and write to output streams. "ISentenceProcessorProcessor" may look odd, but this name is automatically generated based on the name of the interface.

The ExampleHandler should be a Perl script that implements all of the methods defined by the ISentenceProcessor interface. It is possible for the handler to be defined in the same file as the server itself.

12.4 Adept-Concrete Mapping

Concrete is an attempt to map out various NLP data types in a protocol buffer schema for use in projects across Johns Hopkins University (JHU). This standardized schema allows researchers to use a common, underlying data model for all NLP tasks, and thus, facilitates integration between projects. The Concrete GitHub page is located here: https://github.com/hltcoe/concrete

Concrete was created as part of one of the SCALE Workshops hosted at JHU's Human Language Technology Center of Excellence (HLTCOE). More information about the SCALE workshops can be found here: http://hltcoe.jhu.edu/research/scale-workshops/

BBN has introduced a sub-module named adept-api-mappers which uses Apache Thrift to map between Adept API classes and Concrete classes. This module includes a ConcreteAdeptMapper class designed to

facilitate integration with the Adept API for developers who have previously worked with Concrete classes. It supports Concrete version 1.1.8.

The adept-api-mappers module uses Thrift to provide bi-directional mapping between Adept and Concrete classes for much of the Concrete specification. This section is intended to serve as an introduction to the adept-api-mappers module for those developers who may find it useful.

1. Add adept-api-mappers as a dependency in your module's pom.xml

```
<dependency>
 <groupId>adept</groupId>
 <artifactId>adept-api-mappers</artifactId>
 <version>${project.parent.version}</version>
</dependency>
```

2. Import adept.mappers.concrete.ConcreteAdeptMapper into the code that requires Adept-Concrete conversion. The ConcreteAdeptMapper can be initialized with an empty constructor, e.g.

```
package […]
import adept.mappers.concrete.ConcreteAdeptMapper;
    […]
    // Initialize Concrete Adept mappings.
ConcreteAdeptMapper mapper = new ConcreteAdeptMapper();
```

3. To map between Adept and Concrete classes, use the ConcreteAdeptMapper's "map(Object)" method. The ConcreteAdeptMapper supports bi-directional class mapping between the modules adept-api and concrete-protobufs, and as a general rule submitting an Adept object to the mapper returns a Concrete object and vice-versa. Consider for example the Adept classes AudioOffset and CharOffset, which map to the Concrete classes AudioSpan and TextSpan:

```
adept.common.CharOffset charOffset = new CharOffset(4,8);
system.out.printf("CharOffset starts at: %d.\n", charOffset.getBegin());        // prints 4
edu.jhu.hlt.concrete.Concrete.TextSpan textSpan = mapper.map(charOffset);
system.out.printf("TextSpan starts at: %d.\n", textSpan.getStart());     // prints 4
```

4. Some associations between Adept and Concrete classes are between classes of the same name – for example, an Adept Token maps bi-directionally to a Concrete Token.

In some cases the association between classes is not as apparent, so refer to the following table to see the currently supported mappings:

Concrete Class	Adept Class
AudioSpan	AudioOffset
Communication	HltContentContainer
EmailAddress	EmailAddress
EmailCommunicationInfo	Message
LanguageIdentification	LanguageIdentification
SentenceSegmentation	Session
Sound	Audio
TextSpan	CharOffset
Token	Token
Tokenization	TokenStream
Tokenization.TokenLattice	TokenLattice
Tokenization.TokenLattice.Arc	Arc
Tokenization.TokenLattice.LatticePath	LatticePath
UUID	ID

5. Because of the nature of EntityMentions in Adept and Concrete, when mapping Entity Mentions, the mapper requires more information than just a source class. The special methods for mapping EntityMentions are as follows:

To map a Concrete EntityMention to an Adept EntityMention, there are two options – either use

map(Concrete.Communication, Concrete.EntityMention)

or use

map(ConcreteEntityMention)

Note that ConcreteEntityMention is the class edu.jhu.hlt.concrete.ConcreteEntityMention, which is distinct from the class edu.jhu.hlt.concrete.Concrete.EntityMention. Mapping from an Adept EntityMention will return a edu.jhu.hlt.concrete.Concrete.EntityMention object.

An Adept EntityMention can also be mapped to a Concrete SituationMention via the method

mapSituationMention(adept.common.EntityMention)

As in the case of Concrete EntityMentions, a Concrete SituationMention can be mapped to an Adept EntityMention via

map(Concrete.Communication, Concrete.SituationMention)

