Raytheon
BBN Technologies

Adept API README

Export Notice:

This document does not contain technology or Technical Data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations.

Adept API README, for Adept API 2.7.3

Contents

Figures

Tables

Acknowledgement

The Adept system was developed under the sponsorship of DARPA/AFRL contract FA8750-13-C-0008. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

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, and Mention. It also 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 also addresses special topics about reading different input file formats.

1.2 Contact

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

1.3 Adept API Contents

The Adept API repository contains the following top-level directories:

- adept-api
- adept-kb
- example
- javadocs

1.4 Licensing

1.5 Getting Started

System requirements:

- Git version 2.7.4 or later.
- Maven version 3.0.5 or later.
- Java(TM) SDK Environment (build 1.8 or later).

Building from source:

- Ensure that the JAVA_HOME environment variable is set to the root directory of the Java SDK.

- If desired, create a top-directory for Adept development and make it your working directory:
```
$> mkdir work; cd work
```
- Clone the Adept repository to a new repository on your system:
```
$> git clone https://github.com/BBN-E/Adept.git
```
- Change your working directory to the newly-created repository:
```
$> cd Adept
```
- Build the Adept system and populate your local repository:
```
$> mvn clean install
```
- Run the unit tests:
```
$> mvn test –DskipUnitTests=false
```

2 Adept Data Structures

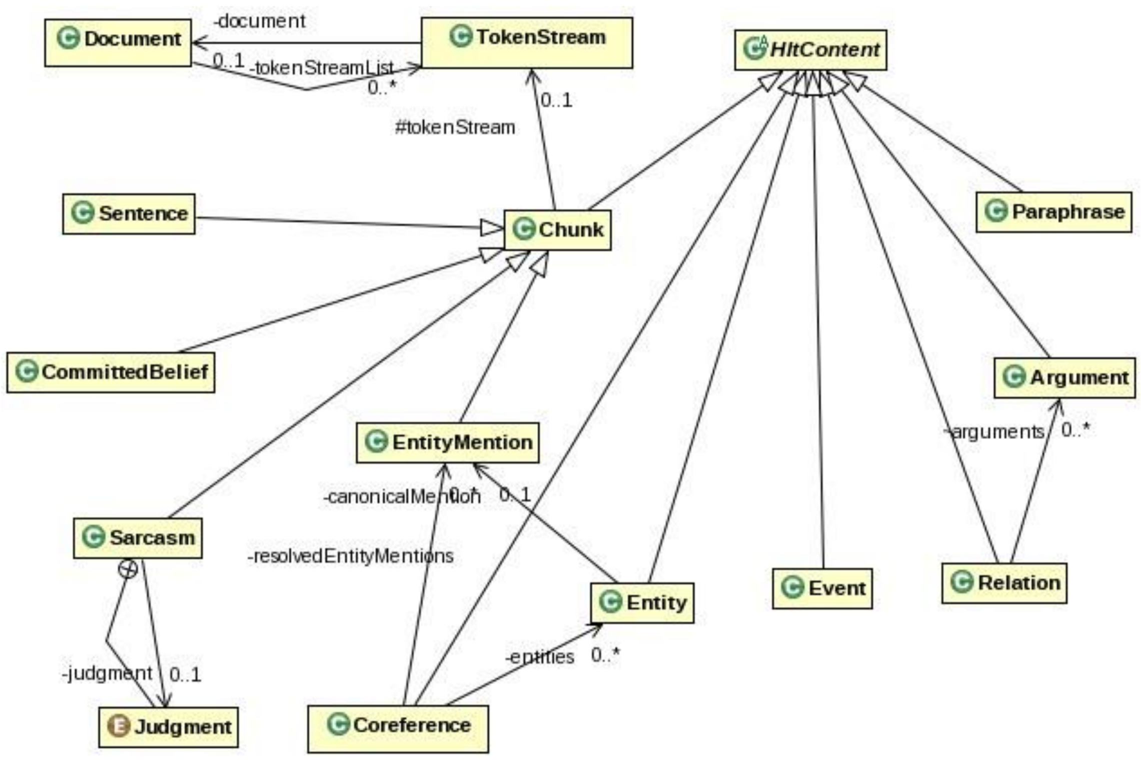

Figure 1: Adept Data Structures

- **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, and events across multiple algorithms

3 Adept Algorithm Interfaces

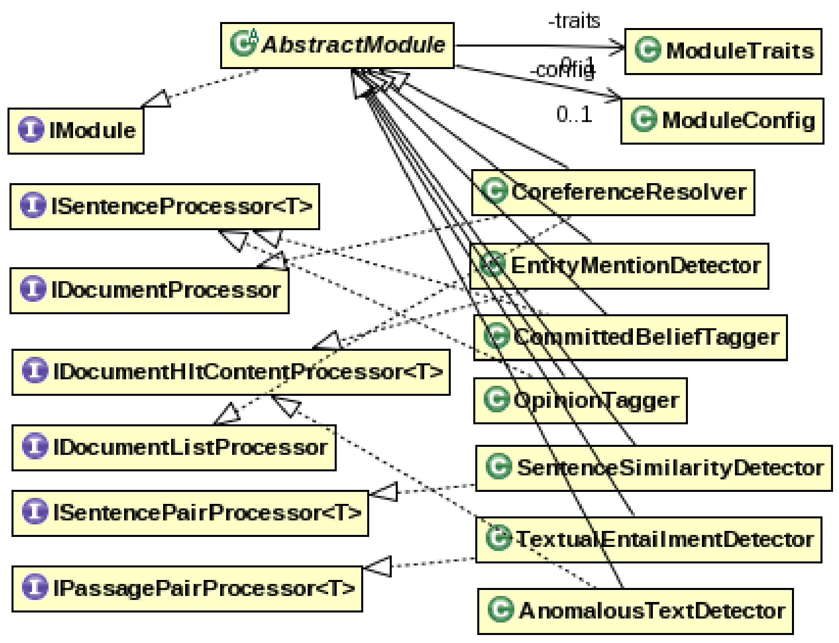

Figure 2: Adept Algorithm Interfaces

- **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.

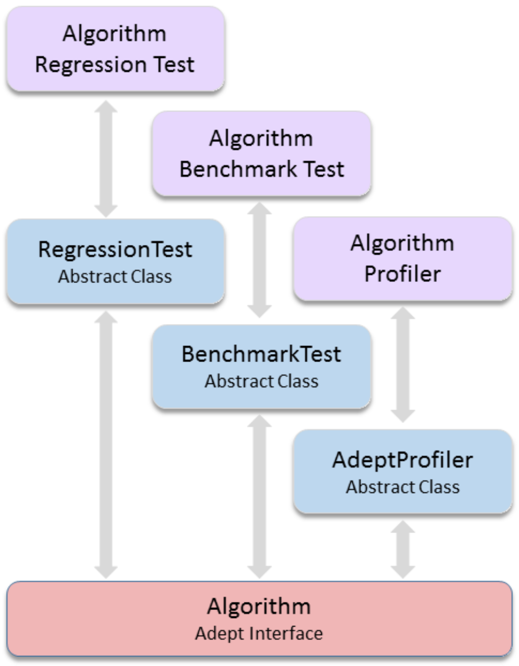
Algorithm Regression Test
Algorithm Benchmark Test
RegressionTest Abstract Class
Algorithm Profiler
BenchmarkTest Abstract Class
AdeptProfiler Abstract Class
Algorithm Adept Interface

Figure 3: Test Support Classes

- **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 document "Adept API Specifications v3.10" for specifics.

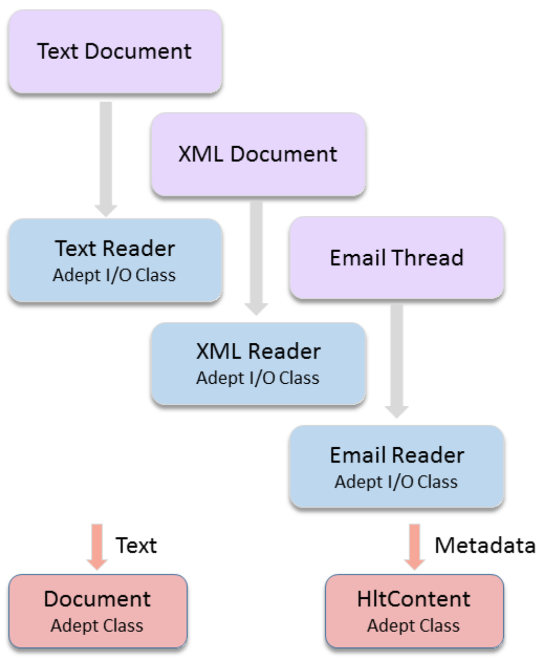

Figure 4: Data Reading Classes

- **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.

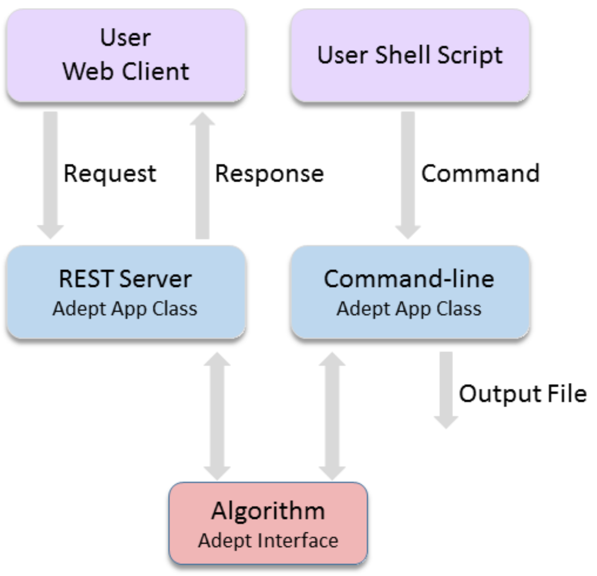

Figure 5: Alternate Means of Invoking Algorithms

- **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 Hadoop parallel processing framework. This example diagram shows four algorithms combined in a pipeline and producing Relation objects as output.

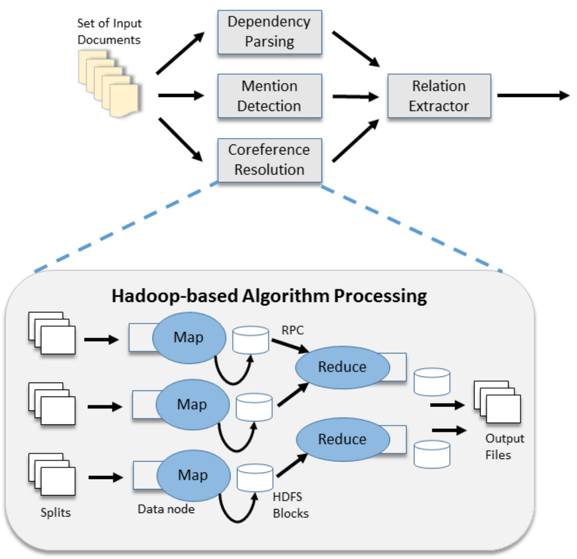

Figure 6: Adept Processing using Hadoop

- **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 Example Algorithms

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

8.1 ExampleNamedEntityTagger

This simple algorithm implements a simplistic English-language Named Entity Recognizer. It is only intended to be used as an example of how to build NLP modules, showing what classes to extend and what methods to define.

The algorithm naively assumes that any word inside a sentence which starts with an uppercase letter is a Named Entity. As such, it is not meant to be an example of an accurate approach to named entity tagging, and is only intended as an example of how to use the Adept API.

The tagger is accompanied by a simple test class called ExampleNamedEntityTaggerTest.java.

8.2 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.

9 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 models.
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) Extend 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()

Table 1: Methods to Implement for a New Algorithm

10 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.

10.1 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 options and 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>.

Argument	Description
--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: 'json,' 'xml,' 'binary.' If omitted, 'xml' is assumed by default.
--table_output	Write a tabular output for each HltContentContainer along with the serializations.
-v, --verbose	Prints additional information.
--version	Prints product version and exists.

Table 2: DeftReaderApp Command Line Options and Arguments

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"
```

10.2 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