This document provides a guide to the basics of using the Cloud Natural Language API. This conceptual guide covers the types of requests you can make to the Natural Language API, how to construct those requests, and how to handle their responses. We recommend that all users of the Natural Language API read this guide and one of the associated tutorials before diving into the API itself.

The Natural Language API has several methods for performing analysis and annotation on your text. Each level of analysis provides valuable information for language understanding. These methods are listed below:

- **Sentiment analysis** inspects the given text and identifies the prevailing emotional opinion within the text, especially to determine a writer's attitude as positive, negative, or neutral. Sentiment analysis is performed through the `analyzeSentiment` method.
- **Entity analysis** inspects the given text for known entities (Proper nouns such as public figures, landmarks, and so on. Common nouns such as restaurant, stadium, and so on.) and returns information about those entities. Entity analysis is performed with the `analyzeEntities` method.
- **Entity sentiment analysis** inspects the given text for known entities (proper nouns and common nouns), returns information about those entities, and identifies the prevailing emotional opinion of the entity within the text, especially to determine a writer's attitude toward the entity as positive, negative, or neutral. Entity analysis is performed with the `analyzeEntitySentiment` method.
- **Syntactic analysis** extracts linguistic information, breaking up the given text into a series of sentences and tokens (generally, word boundaries), providing further analysis on those tokens. Syntactic Analysis is performed with the `analyzeSyntax` method.
- **Content classification** analyzes text content and returns a content category for the content. Content classification is performed by using the `classifyText` method.

Each API call also detects and returns the language, if a language is not specified by the caller in the initial request.

Additionally, if you wish to perform several natural language operations on given text using only one API call, the `annotateText` request can also be used to perform sentiment analysis and entity

analysis.

The Natural Language API is a REST API, and consists of JSON requests and response. A simple Natural Language JSON Entity Analysis request appears below:

These fields are explained below:

- `document` (/natural-language/docs/reference/rest/v1/documents#Document) contains the data for this request, which consists of the following sub-fields:
  - `type` - document type (`HTML` or `PLAIN_TEXT`)
  - `language` - (optional) the language of the text within the request. If not specified, language will be automatically detected. For information on which languages are supported by the Natural Language API, see Language Support (/natural-language/docs/languages). Unsupported languages will return an error in the JSON response.
  - Either `content` or `gcsContentUri` which contain the text to evaluate. If passing `content`, this text is included directly in the JSON request (as shown above). If passing `gcsContentUri`, the field must contain a URI pointing to text content within Google Cloud Storage.
- encodingType (/natural-language/docs/reference/rest/v1/EncodingType) - (required) the encoding scheme in which returned character offsets into the text should be calculated, which must match the encoding of the passed text. If this parameter is not set, the request will not error, but all such offsets will be set to `-1`.

When passing a Natural Language API request, you specify the text to process in one of two ways:

- Passing the text directly within a `content` field.
- Passing a Google Cloud Storage URI within a `gcsContentUri` field.

In either case, you should make sure not to pass more than the Content Limits (/natural-language/limits#content) allow. Note that these content limits are by *byte*, not by *character*; character length therefore depends on your text's encoding.

The request below refers to a Google Cloud Storage file containing the Gettysburg Address:

Sentiment analysis attempts to determine the overall attitude (positive or negative) expressed within the text. Sentiment is represented by numerical `score` and `magnitude` values.

A sample `analyzeSentiment` response to the Gettysburg Address (#sentiment-request) is shown below:

These field values are described below:

- `documentSentiment` contains the overall sentiment of the document, which consists of the following fields:
  - `score` of the sentiment ranges between `-1.0` (negative) and `1.0` (positive) and corresponds to the overall emotional leaning of the text.
  - `magnitude` indicates the overall strength of emotion (both positive and negative) within the given text, between `0.0` and `+inf`. Unlike `score`, `magnitude` is not normalized; each expression of emotion within the text (both positive and negative) contributes to the text's `magnitude` (so longer text blocks may have greater magnitudes).
- `language` contains the language of the document, either passed in the initial request, or automatically detected if absent.
- `sentences` contains a list of the sentences extracted from the original document, which contains:
  - `sentiment` contains the *sentence level sentiment* values attached to each sentence, which contain `score` and `magnitude` values as described above.

A response value to the Gettysburg Address of `0.2` score indicates a document which is slightly positive in emotion, while the value of `3.6` indicates a relatively emotional document, given its small size (of about a paragraph). Note that the first sentence of the Gettysburg address contains a very high positive `score` of `0.8`.

The *score* of a document's sentiment indicates the overall emotion of a document. The *magnitude* of a document's sentiment indicates how much emotional content is present within the document, and

this value is often proportional to the length of the document.

It is important to note that the Natural Language API indicates differences between positive and negative emotion in a document, but does not identify specific positive and negative emotions. For example, "angry" and "sad" are both considered negative emotions. However, when the Natural Language API analyzes text that is considered "angry", or text that is considered "sad", the response only indicates that the sentiment in the text is negative, not "sad" or "angry".

A document with a neutral score (around `0.0`) may indicate a low-emotion document, or may indicate mixed emotions, with both high positive and negative values which cancel each out. Generally, you can use `magnitude` values to disambiguate these cases, as truly neutral documents will have a low `magnitude` value, while mixed documents will have higher magnitude values.

When comparing documents to each other (especially documents of different length), make sure to use the `magnitude` values to calibrate your scores, as they can help you gauge the relevant amount of emotional content.

The chart below shows some sample values and how to interpret them:

Sentiment	Sample Values
Clearly Positive*	"`score`": 0.8, "`magnitude`": 3.0
Clearly Negative*	"`score`": -0.6, "`magnitude`": 4.0
Neutral	"`score`": 0.1, "`magnitude`": 0.0
Mixed	"`score`": 0.0, "`magnitude`": 4.0

* “Clearly positive” and “clearly negative” sentiment varies for different use cases and customers. You might find differing results for your specific scenario. We recommend that you define a threshold that works for you, and then adjust the threshold after testing and verifying the results. For example, you may define a threshold of any score over 0.25 as clearly positive, and then modify the score threshold to 0.15 after reviewing your data and results and finding that scores from 0.15-0.25 should be considered positive as well.

Entity Analysis provides information about entities in the text, which generally refer to named "things" such as famous individuals, landmarks, common objects, etc.

Entities broadly fall into two categories: proper nouns (https://en.wikipedia.org/wiki/Proper_noun) that map to unique entities (specific people, places, etc.) or common nouns (also called "nominals" in natural language processing). A good general practice to follow is that if something is a noun, it qualifies as an "entity." Entities are returned as indexed offsets into the original text.

An Entity Analysis request should pass an `encodingType` argument, so that the returned offsets can be properly interpreted.

Entity analysis returns a set of detected entities, and parameters associated with those entities, such as the entity's type, relevance of the entity to the overall text, and locations in the text that refer to the same entity. Entities are returned in the order (highest to lowest) of their `salience` scores, which reflect their relevance to the overall text.

An `analyzeEntities` response to the entity request (#entity-request) is shown below:

Note that the Natural Language API returns entities for "Lawrence of Arabia" (the film) and "T.E. Lawrence" (the person). Entity analysis is useful for disambiguating similar entities such as "Lawrence" in this case.

The fields used to store the entity's parameters are listed below:

- `type` indicates the type of this entity (for example if the entity is a person, location, consumer good, etc.) This information helps distinguish and/or disambiguate entities, and can be used for writing patterns or extracting information. For example, a `type` value can help distinguish similarly named entities such as “Lawrence of Arabia”, tagged as a `WORK_OF_ART` (film), from “T.E. Lawrence”, tagged as a `PERSON`, for example. (See Entity Types (/natural-language/docs/reference/rest/v1/Entity#Type) for more information.)
- `metadata` contains source information about the entity's knowledge repository Additional repositories may be exposed in the future. This field may contain the following subfields:
  - `wikipedia_url`, if present, contains the Wikipedia (https://www.wikipedia.org/) URL pertaining to this entity.
  - `mid`, if present, contains a machine-generated identifier (MID) corresponding to the entity's Google Knowledge Graph (https://www.google.com/intl/bn/insidesearch/features/search/knowledge.html) entry. Note that `mid` values remain unique across different languages, so you can use such values to tie entities together from different languages. For inspecting these MID values, please consult the Google Knowledge Graph Search API (https://developers.google.com/knowledge-graph/) documentation.

    ★ Not all returned `mid` values are available for inspection with the Google Knowledge Graph API. You can use the `mid` as a unique identifier for an entity, however you should not rely on using it to look up additional information about the entity.

- `salience` indicates the importance or relevance of this entity to the entire document text. This score can assist information retrieval and summarization by prioritizing salient entities. Scores closer to `0.0` are less important, while scores closer to `1.0` are highly important.
- `mentions` indicate offset positions within the text where an entity is mentioned. This information can be useful if you want to find all mentions of the person “Lawrence” in the text but not the film title. You can also use mentions to collect the list of entity aliases, such as “Lawrence,” that refer to the same entity “T.E. Lawrence”. An entity mention may be one of two types: `PROPER` or `COMMON`. A proper noun Entity for "Lawrence of Arabia," for example, could be mentioned directly as the film title, or as a common noun ("film biography" of T.E. Lawrence).

Entity sentiment analysis combines both entity analysis and sentiment analysis and attempts to determine the sentiment (positive or negative) expressed about entities within the text. Entity

sentiment is represented by numerical score and magnitude values and is determined for each mention of an entity. Those scores are then aggregated into an overall sentiment score and magnitude for an entity.

Entity Sentiment Analysis requests are sent to the Natural Language API through use of the `analyzeEntitySentiment` (/natural-language/docs/reference/rest/v1/documents/analyzeEntitySentiment) method in the following form:

You can specify an optional `language` parameter with your request that identifies the language code for the text in the `content` parameter. If you do not specify a `language` parameter, then the Natural Language API auto-detects the language for your request content. For information on which languages are supported by the Natural Language API, see Language Support (/natural-language/docs/languages).

The Natural Language API processes the given text to extract the entities and determine sentiment. An Entity Sentiment Analysis request returns a response containing the `entities` (/natural-language/docs/basics#entity-analysis-response) that were found in the document content, a `mentions` entry for each time the entity is mentioned, and the numerical `score` and `magnitude` values for each mention, as described in Interpreting sentiment analysis values (/natural-language/docs/basics#sentiment-analysis-values). The overall `score` and `magnitude` values for an entity are an aggregate of the specific `score` and `magnitude` values for each mention of the entity. The `score` and `magnitude` values for an entity can be `0`, if there was low sentiment in the text, resulting in a `magnitude` of 0, or the sentiment is mixed, resulting in a `score` of 0.

For an example, see Analyzing Entity Sentiment (/natural-language/docs/analyzing-entity-sentiment).

The Natural Language API provides a powerful set of tools for analyzing and parsing text through syntactic analysis. To perform syntactic analysis, use the `analyzeSyntax` method.

Syntactic Analysis consists of the following operations:

- Sentence extraction (#sentence-extraction) breaks up the stream of text into a series of sentences.
- *Tokenization* (#tokenization) breaks the stream of text up into a series of tokens, with each token usually corresponding to a single word.
- The Natural Language API then processes the tokens and, using their locations within sentences, adds syntactic information to the tokens.

Full documentation on the set of syntactic tokens is within the Morphology & Dependency Trees (/natural-language/docs/morphology) guide.

Syntactic Analysis requests are sent to the Natural Language API through use of the `analyzeSyntax` method in the following form:

The Natural Language API processes the given text to extract sentences and tokens. A Syntactic Analysis request returns a response containing these `sentences` and `tokens` in the following form:

When performing syntactic analysis, the Natural Language API returns an array of sentences extracted from the provided text, with each sentence containing the following fields within a `text` parent:

- `beginOffset` indicating the (zero-based) character offset within the given text where the sentence begins. Note that this offset is calculated using the passed `encodingType`.
- `content` containing the full text of the extracted sentence.

For example, the following `sentences` element is received for a Syntactic Analysis request of the Gettysburg Address (#sentiment-analysis):

A syntactic analysis request to the Natural Language API will also include a set of tokens. You can use the information associated with each token to perform further analysis on the sentences returned. More information on these tokens can be found in the Morphology & Dependency Trees (/natural-language/docs/morphology) guide.

The `analyzeSyntax` method also transforms text into a series of tokens, which correspond to the different textual elements (word boundaries) of the passed content. The process by which the Natural Language API develops this set of tokens is known as *tokenization*.

Once these tokens are extracted, the Natural Language API processes them to determine their associated part of speech (including morphological information) and lemma. Additionally, tokens are evaluated and placed within a *dependency tree* (/natural-language/docs/morphology#dependency-tree), which allows you to determine the syntactic meaning of the tokens, and illustrate the relationship of tokens to each other, and their containing sentences. The syntactic and morphological information associated with these tokens are useful for understanding the syntactic structure of sentences within the Natural Language API.

The set of token fields returned in a syntactic analysis JSON response appears below:

- `text` contains the text data associated with this token, with the following child fields:
  - `beginOffset` contains the (zero-based) character offset within the provided text. Note that although dependencies (described below) exist only within sentences, token offsets are positioned within the text as a whole. Note that this offset is calculated using the passed `encodingType`.
  - `content` contains the actual textual content from the original text.
- `partOfSpeech` provides grammatical information, including morphological information (https://en.wikipedia.org/wiki/Morphology_(linguistics)), about the token, such as the token's tense, person, number, gender, etc. (For more complete information on these fields, consult the Morphology & Dependency Trees (/natural-language/docs/morphology) guide.)
- `lemma` contains the "root" word upon which this word is based, which allows you to canonicalize word usage within your text. For example, the words "write", "writing", "wrote" and "written" all are based on the same lemma ("write"). As well, plural and singular forms are based on lemmas: "house" and "houses" both refer to the same form. (See Lemma (morphology) (https://en.wikipedia.org/wiki/Lemma_(morphology)).)
- `dependencyEdge` fields identify the relationship between words in a token's containing sentence via edges in a directed tree. This information can be valuable for translation, information extraction, and summarization. (The Morphology & Dependency Trees (/natural-language/docs/morphology) guide contains more detailed information about dependency parsing.) Each `dependencyEdge` field contains the following child fields:
  - `headTokenIndex` provides the (zero-based) index value of this token's "parent token" within the token's encapsulating sentence. A token with no parent indexes itself.
  - `label` provides the type of dependency of this token on its head token.

The following quote from Franklin D. Roosevelt's Inaugural speech (https://en.wikipedia.org/wiki/First_inauguration_of_Franklin_D._Roosevelt) will produce the following tokens:

NOTE: all `partOfSpeech` tags containing `*_UNKNOWN` values have been removed for clarity.

You can have the Natural Language API analyze a document and return a list of content categories that apply to the text found in the document. To classify the content in a document, call the `classifyText` (/natural-language/reference/rest/v1/documents/classifyText) method.

A complete list of content categories returned for the `classifyText` (/natural-language/reference/rest/v1/documents/classifyText) method are found here (/natural-language/docs/categories).

The Natural Language API filters the categories returned by the `classifyText` method to include only the most relevant categories for a request. For instance, if `/Science` and `/Science/Astronomy` both apply to a document, then only the `/Science/Astronomy` category is returned, as it is the more specific result.

:ant: You must supply a text block (document) with at least twenty tokens (words) to the `classifyText` ıral-language/reference/rest/v1/documents/classifyText) method.

For an example of content classification with the Natural Language API, see Classifying Content (/natural-language/docs/classifying-text).

If you wish to perform a set of Natural Language API operations within a single method call, you can use `annotateText`as a general purpose Natural Language API request. A Text Annotation JSON request is similar to a standard Entity Analysis request (#entity-request) but additionally requires a set of passed features (/natural-language/docs/reference/rest/v1/documents/annotateText#Features) to indicate the operations to perform on the text. These features are listed below:

- `extractDocumentSentiment` performs sentiment analysis, as described in the Sentiment Analysis (#sentiment_analysis) section.

- `extractEntities` performs entity analysis, as described in the Entity Analysis (#entity_analysis) section.
- `extractSyntax` indicates that the given text should be processed to perform syntactic analysis, as described in the Syntactic Analysis (#syntactic_analysis) section.

The following request calls the API to annotate `features` in a short sentence.