

# ***Introducing Syntactica***

An English language processing API  
September 2006

By James Mathewson  
Editor in Chief, IBM.com  
Former Editor in Chief, ComputerUser.com

## Table of contents

Abstract .....	3
Introduction .....	3
How <a href="#">Syntactica works</a> .....	<a href="#">4</a>
<a href="#">Why Syntactica succeeds</a> .....	<a href="#">5</a>
<a href="#">A few applications</a> .....	<a href="#">7</a>
<a href="#">Conclusion</a> .....	<a href="#">8</a>
<a href="#">About the author</a> .....	<a href="#">8</a>
<a href="#">Appendix I- History</a> .....	<a href="#">9</a>
<a href="#">Appendix II--Sample Output-Index</a> .....	<a href="#">11</a>
<a href="#">Appendix III--Sample Output-Abstract</a> .....	<a href="#">13</a>
<a href="#">Appendix IV--Sample Output-Rotated Index</a> .....	<a href="#">14</a>

---

## Abstract

This paper describes an application that can take an entire article and give you its essential meaning in a short prioritized list of sentences. The application programmer interface (API) at the core of the application can be embedded in a any number of programs that process natural language English, including search output, search indexing, and book indexing.

This white paper describes an API that can be embedded in those applications and more. It is called Syntactica™ because, given any grammatical English text input, it can provide an output that provides the essential meaning, or sense, of the text.

---

## Introduction

For decades, one of the highest goals for computer scientists is the program that can process natural language. The trail to this Holy Grail, however, is littered with failed attempts, the most notable of which is Artificial Intelligence (AI).

With the advent of the Web, the bar has been raised considerably. Instead of trying to make sense of large databases of structured data, as AI researchers attempted, programmers now must find a way to make sense of massive databases of unstructured data on the Web.

Search engines are the highest profile examples of failure in writing programs that can make sense of Web text. For more than a decade, the best and the brightest minds in computer science have attempted to provide users with relevant search results for their queries. Most advances have nipped at the edges of the problem by enabling closer approximations of the meaning of users' inputs, including Boolean search strings, exact phrase matches, and natural language search strings. In every case, they have enabled users to more accurately type what they want in the search field. But they have failed to appreciably improve the way their programs match search outputs to user's meanings.

Recent advancements in XML metadata, such as Web ontologies, have attempted to add semantic structure to sites. In theory, the Web Ontology Language (OWL) and other advances of the Semantic Web allow browsers, search engines, or other agents to understand the meaning of Web content and give users desired outputs—executive summaries, abstracts, or whatever form of encapsulated meaning the user defines.

In practice, the Semantic Web is a long way from the Holy Grail. Because the onus is on individual sites to define their terms in ways that can be understood by browsers and other agents, adoption will be incomplete, and will suffer from a lack of consistent standards. Just as it took years for HTML, XML, and related technologies to become sufficiently robust to allow easy-to-use user interface designs, so it will take years for OWL and related technologies to become sufficiently robust to allow browsers to decode the meaning of Web content. And some doubt that it will ever reach the Holy Grail.

But there is a program-- Syntactica —that can do today what OWL might only do after much rapid refinement and adoption. And unlike the Semantic Web, Syntactica requires no work by the webmasters of existing Web sites. It does all the work to gather the data,

analyze it, and produce the output. It currently works on any document, but could be easily adapted to plug into Web browsers, word processing programs, or PDF readers.

Syntactica is not a program *per se*, but a C++ language application programmer interface (API) that can be embedded in any number of applications to return relevant outputs given a wide variety of natural language inputs. In addition to plugging into Web browsers or search engines, it could plug into word processing programs to automatically provide abstracts, executive summaries, back-of-the book indexes, and writing or translation support.

It also could be plugged into tools used by editors and indexers of search sites to better track and evaluate content on the Web. It could also be used by corporate webmasters to give users more personalized information based on their stated desires. Indeed the range of applications that could take advantage of an API like Syntactica is quite broad

How did developers of Syntactica get to this solution when there has been so much unsuccessful effort expended to solve the problem? That's a long story that will be told in the course of this white paper. Along the way, we will tell the story of where Syntactica came from and we will explain why and how Syntactica succeeds where others have not succeeded.

---

## How ePrécis works

Unlike conventional search-and-retrieval programs, Syntactica does not simply match strings of letters to other strings of letters in an index. It analyzes words in the context of the sentence structures in which the words reside. To take a simple example, a word will have a different weight if it is a noun than if it is a modifier. Syntactica takes into account far more complex aspects of English grammar than the part of speech in which the word is used to determine if a word or phrase is relevant or not.

The program first determines the semantic weight of the word from either the small or the large dictionaries. The small dictionary filters all the short words such as articles out and gives them their meanings. The rest of the words are analyzed by large dictionary. The large dictionary assigns a series of codes to the words based on what their dictionary definition is together with how much information is contained in them. After the semantic weight is determined, the program determines the word's place in the text's syntactic structure to determine its overall relevance. Once a term's relevance has been determined, the program follows more rules that compare the weight of all the terms within a text, and generates sentences based on the relevance of all its terms to produce the desired output.

The program is really four programs put together, called Syn20, Syn40, Syn60, and Syn80. Each part is described as follows:

- **Syn20** scans the document for all small common words, such as articles and morphemes, looks them up in the small dictionary and decodes them.
- **Syn40** is the main dictionary program. It takes all the words that are not filtered into the small dictionary and assigns semantic codes to them. The codes are

based on a practice called paradigmming, in which they are placed in a matrix relating how much information is in the word. The quantity of information is based on the word's complexity. The word *table* is relatively simple because there are few types of tables. The word *automobile* is relatively complex because there are hundreds of types of automobiles. Each of the 55,000 words in the large dictionary was at one time hand coded according to the paradigmming matrix. In several cases, the codes were hand adjusted as additional information related to the words was discovered, such as additional connotations, relationships with other words, or contextual variations in usage. Syn40 also assigns suffix codes and syntactic codes, which relate to the word's part of speech and other grammatical features of the word.

- **Syn60** contains the meat of the syntactic analyzing. It is the most involved and the most often refined aspect of the program because of the aforementioned complexities of the English language. Given the list of words with their semantic codes, suffix codes and parts of speech codes, it analyses their overall relevance. Several subroutines revolve around the main program, like electrons revolving around the nucleus of an atom. As anomalies in English syntax are discovered, these subroutines are added or modified to more accurately reflect specialized forms of English usage, such as legalese.
- **Syn80** takes the results of the Syn60 analysis and creates an output currently in the form of a set of numbered sentences. The numbers relate to relative importance of the sentences, where 1 is the most important sentence summarizing the point of the text. Syn80 can be modified to create other outputs as needed for the application. For example, once you have a list of numbered sentences, it's a fairly simple matter to add routines that turn this output into an executive summary.

---

## Why ePrécis succeeds

Computer scientists have tried to codify the linguistic structure of language over the last several decades with little success. But Syntactica succeeded where others failed because it does not oversimplify natural language. A lot of the refinements to the program's rules and its dictionary must be coded by brute force, rather than through fixed rules. For example, most verbs behave normally such that even if you have never used a verb before, you know how to form the past tense or gerund form from the present perfect. Still, there are more than 200 irregular verbs in English that require special rules in order to analyze sentences and create grammatical outputs containing them. Syntactica linguists created a separate routine for each and every irregular verb in the English language. Each one of these routines is like the electron orbiting around the nucleus.

Most computer scientists and practitioners of AI focus on the rules of a language when they try to codify it. In this regard, Syntactica is just a more sophisticated AI program. Syntactica's sophistication comes from Syntactica's linguists understanding of Linguistics: as scholars in the field, they knew early on that natural language cannot be

adequately circumscribed by a few dozen finite and rigid rules. The program needed subroutines that reflect the flexibility of its use in ordinary discourse.

But the sophistication of Syntactica's rules-based algorithms only makes it more accurate than other natural language programs based on AI. Its sophistication does not, by itself, allow Syntactica to succeed. The real key is its dictionary. The linguists, have meticulously gone through the entire dictionary of the English language, considering each word in depth. The work starts with the dictionary definitions of the word. Often times, there are primary and secondary meanings, and these need to be coded in. Sometimes these secondary meanings are in different parts of speech: A noun can have an alternative meaning as an adjective, for example.

Starting with the dictionary and moving into other aspects of a word's use, such as technical usage in certain professional contexts, for example, a set of linguistic values or codes was assigned to each word. These codes help disambiguate the meaning of the word for the program, in each of its particular contexts or uses. For example, the word "cold" has a multiplicity of meanings, both as a noun and an adjective. This coding system takes into account each context or usage, and determines the word's linguistic meaning in each particular context. The codes are of the linguists' own devising, but they have been tested, retested, and refined over the decades.

With each change or addition in the dictionary, related words often had to adjust related. For example, if codes for the word "excuse" need to be modified, the words "plea," "justification," and a host of adverbs such as "clumsily," "accidentally," and "mistakenly" might also need adjustment. It is a painstaking process that can only be performed by a master of linguistics and a man of letters such as the lead linguist, Arnie Schultz. With each modification, Syntactica gets closer to what Noam Chomsky called the semantic base—the true meaning of what he called utterances, which includes words, phrases, sentences, and other components of whole texts of natural language.

The simple fact is it took years of determination and trial and error to refine a program so that it approximates the complexities of the English language. The program was born with the solid linguistic principles of Chomsky and the best efforts of Schulz. But it needed to mature under his constant vigilance. Just as a child learns language largely by being corrected by her parents and teachers, so, over the years, Syntactica learned the English language by being corrected by its creator. Whenever the program returned an undesirable result, Schulz considered it a bug, found the source of the problem, and added a routine to fix the problem. Often these problems had nothing to do with the coding itself but with anomalies of English. If so, the dictionary typically needed to be modified as well to change the relative importance of one or more terms, and to produce better output as a result.

Artificial intelligence has yet to succeed because it attempts to create machines that understand language through a finite system of rules. Chomsky often claimed that these rules are innately part of the human brain. But even with these innate rules intact, humans must learn natural language through a long process of trial and error. Syntactica succeeds where AI does not by adding this long process of trial and error to an ingenious system of rules. Like a human being, Syntactica has learned the English language, which allows it to understand grammatical English sentences, where all AI programs can do is parse strings of alphabetic letters.

---

## A few applications

As mentioned above, the applications of a natural language processing API are quite broad. Consider the following applications:

- Suppose you run a search engine and want to improve the presentation of the pages users get when they make search queries. Currently, you reproduce only the first few lines of content within the Web site that match user's queries. This is not efficient for users because they often can't evaluate whether or not a given site closely matches what they want from the first few lines of content.

What is needed, is an API that will scan the content of the site and produce a quick abstract. Then, instead of presenting the first few lines of content under the link for a search result, the abstract is presented. In this way, users don't have to click multiple sites and repeatedly hit the **Back** button when they are disappointed by the sites' contents.

By plugging Syntactica into the front end of the search engine, you can develop a competitive advantage: Users will prefer the site because they will more quickly and easily find the information that closely matches their search queries.

- You could also plug Syntactica into the back end of the search engine, the engine itself could more accurately return results that match user's queries. Because the abstracts will do a better job of determining the relevance of a site to a search string, search results will be more relevant to users, which is the ultimate competitive advantage for search engines.
- Syntactica could also be a browser plug-in that allows users to see a site's abstract simply by mousing over its search result.
- Syntactica could enable the content creation process by plugging into a suite of Web tools in which developers create and code Web content. Running Syntactica enables them to create front-end pages that drive traffic to the content, to check the content they have created for translatability and relevance, among other things.
- Syntactica could be embedded into a PDF reader, enabling researchers to read abstracts or executive summaries of PDF documents, rather than needing to scan the whole document.

These are just a few of the useful applications that could derive from using Syntactica API.

---

## Conclusion

Computer scientists have never succeeded in developing programs that understand English because programming English is more of an art than a science. It took an artist's

eye to understand the complexities of the English language. It took an inventor's perseverance to see that those details were coded into Syntactica API. And it took a scientist's mind to ensure that the code works to perfection

Excellent products do not ensure their own commercial success. They require the vision of investors and executives to see their market application, to wisely tailor the products for those markets, and to ensure that the product receives enough attention from those interested in the products within the market. The great value of Syntactica is its flexibility. As a C++ API, it can seamlessly fit within any program that holds, searches, analyses, or presents information in the English language. As such, it is a chameleon technology, changing itself to fit the market. As simple intellectual property, how it is tailored to the market can also be flexible. It could be licensed, sold, turned into products and sold, or some hybrid combination.

---

## **About the author**

James Mathewson has 12 years of experience in information technology journalism and analysis. He is a former editor and current editor at large of ComputerUser magazine. He also works as a consultant for a variety of technology firms, including IBM. He holds an advanced degree in the philosophy of language and linguistics from the University of Minnesota.

## **Appendix I**

---

## History

In the 1960s, Arnie Schulz was a graduate student at the University of Minnesota studying English language and literature. While working for the University of Minnesota Medical School, he received a grant from the National Institutes of Health (NIH) to develop the first computer text search-and-retrieval program. The assembler language program was an initial success for the NIH because it was able to search and retrieve terms from medical journals. This enabled NIH to conduct studies on uses of keywords related to important diseases and their diagnoses and treatments. The basic principle of the program was to use the syntactic structures of the English language to pick out the words most likely to contribute to the meaning of sentences, and then to form meaningful output from the results.

Over 40 years, Schulz continued to refine the program based on its early foundations. The commercial history of the program follows a long and winding trail. In the 1970s, Marathon Oil owned the rights to the program under the brand name Masquerade, which was derived from the fact that the program uses syntactic structures to mask out words that are irrelevant when searching for meaningful words.

In the 1980s, after Marathon Oil relinquished its rights, Schulz retained the rights to the program and once again began refining the various aspects of it, including its vast dictionary, to more accurately pick out the essential meaning of texts. For a time, Schulz worked for West Law, where he converted it into a program that could create back-of-the-book indexes for law texts. Though the program would have increased indexers' productivity by orders of magnitude, West Law was not interested in it. West Law made more money if indexers were able to bill for more hours, so indexer's productivity was not important to the company.

Later in the 1980s, after leaving West Law, Schulz gained venture capital investors and, together with several partners, formed Syntactic Analyzer Inc. as the owning corporation of its rights. After converting it to C++, refining it, and developing subroutines that produced abstracts in addition to indexes, company developers rebranded it as Syntactica.

In the 1990s, rights to Syntactica were sold to Innovex, Inc. of Minneapolis. Innovex formed a subsidiary called Iconovex specifically to develop and market products related to Syntactica. The two primary products were AnchorPage, a plug-in to Netscape Navigator that enabled Web browsers to search the Internet prior to search engines, and Indexicon, an add-on for WordPerfect that created back-of-the-book indexes. The programs worked very well and received a lot of praise from the computer press. In fact, in 1995 and 1996, AnchorPage and Indexicon won Best of COMDEX awards. However, Innovex was more interested in other aspects of its business, and despite a \$10 million investment in Iconovex, the program languished. In 2002, Innovex relinquished Syntactica's rights to the shareholders of Syntactic Analyzer Inc.

Since 2002, the program has once again been refined and improved in the very capable hands of Arnie Schultz. While Iconovex developers left its central routines and subroutines intact, they did not understand the importance of the dictionary, which gives words strategic weight that help the program better understand their importance relative

to an overall text. After two years of refinement, the 55,000 word dictionary is complete, and the API is ready for commercial applications.

While Syntactica's abstract program remains, the API on which it's based was recently rebranded Syntactica because of its unique ability to elevate the relevance of certain keywords within a text in order to capture the essential meaning, or **sense**, of the text.

(The name also alludes to the influential philosopher of language and linguist J.L. Austin and his groundbreaking lectures—*Sense and Sencibilia*<sup>1</sup>. In his career, Austin showed that the linguistic rules of language alone cannot adequately define the meaning of natural language sentences or words. Determining the sense of a sentence or statement also requires an understanding of how words are used in ordinary speech, which is uniquely determined in Syntactica by its dictionary. The method used to create Syntactica's dictionary is similar to the method Austin refers to towards the end of his paper *A Plea for Excuses*.<sup>2</sup>)

---

<sup>1</sup> Oxford, 1959.

<sup>2</sup> Proceedings of the Aristotelean Society, 1956-7.

## Appendix II—Sample Output--Index

This Index was created by the Syntactica API.

Within the algorithm, the “depth” of the Index can be set on a scale of 1 to 7. The depth of this particular Index was set at a level of “3”.

- Ability
  - unique, 5
- Abstract
  - quick, 8
  - site's, 8
- Abstract 3 Introduction 3 History 4, 2
- Abstracts, 5
- Account, 7
- Adjustment, 7
- Advancements, 3
- Adverbs
  - justification and host of, 7
- Analysis
  - results of Syn60, 6
- Application Error, 2
- Artist's eye, 9
- Basic principle of program, 4
- Capital, 4
- Code Web content, 8
- Codes, 6
  - series of, 5
- Coding system
  - Schulz's, 7
- Commercial history of program, 4
- Complexities, 7, 9
- Computer press, 5
- Computer science, 3
- Computer text search-and-retrieval program
  - first, 4
- Content, 4
  - Web, 3
- Content of site, 8
- Corporation of rights
  - owning, 4
- Dictionary, 5, 7, 8
  - large, 5, 6
  - ePrécis's, 5
  - small, 6
- Editors, 4
- Electron, 7
- Engines, 3
- Examples of failure
  - highest profile, 3
- Flexibility of use, 7
- Force
  - brute, 7
- Graduate, 4
- Grammatical features of word, 6
- Grant, 4
- History, 4
- Importance
  - relative, 8
- Indexes, 5
- Information, 8
  - additional, 6
  - much, 6
- Interface
  - programmer, 4
- Interface designs
  - user, 3
- Investors, 9
- Job
  - better, 8
- Language, 7, 8
  - English, 4, 7, 9
  - linguistic rules of, 5
  - natural, 3, 4
- Language and literature
  - English, 4
- Language application
  - C, 4
- Language sentences or words
  - natural, 5
- Linguistic, 7
- Linguistic values or codes, 7
- Linguistics and man of letters
  - master of, 7
- Machines, 8
- Market application, 9
- Markets, 9
- Matrix, 6
  - paradigming, 6
- Metadata
  - XML, 3
- Nucleus, 7
- Nucleus of atom, 6
- Owl, 3
- Paper A Plea
  - end of, 5

- Parents, 7
- Plea
  - words, 7
- Process, 3
  - content creation, 8
  - painstaking, 7
- Product, 9
- Products, 9
- Refinements program's rules and dictionary, 7
- Search, 3
- Search engine and want, 8
- Search output, 3
- Search queries, 8
- Search result, 8
- Search results
  - relevant, 3
- Search sites
  - indexers of, 4
- Search string, 8
- Search-and-retrieval programs
  - conventional, 5
- Semantic codes, 6
- Semantic structure, 3
- Sentences, 6
  - numbered, 6
- Speech
  - ordinary, 5
  - word's part of, 6
- Student, 4
- Subroutines, 4, 7
  - several, 6
- Suffix codes, 6
- Syntactic codes, 6
- Syntactic structures, 4
- System of rules
  - finite, 8
- Teachers, 7
- Technologies, 3
- Text, 5, 6
  - sense of, 5
  - sense of Web, 3
- Tools, 4
  - suite of Web, 8
- To-use
  - easy-, 3
- Track, 4
- Trail, 4
- Trial and error
  - took years of determination and, 7
- Venture, 4
- Vision, 9
- Weight terms, 6
- Well
  - very, 5

## Appendix III—Sample Output--Abstract

This Abstract was created by the Syntactica API.

Within the algorithm, the “depth” of the Abstract can be set on a scale of 1 to 7. The depth of this particular Abstract was set at a level of “3”.

Process natural language English including search output . . . Best and brightest minds in computer science attempting to provide users with relevant search results for queries . . . Just as took years for HTML, XML and relating technologies to become sufficiently robust to allow easy-to-use user interface designs . . . Commercial history of program following long and winding trail . . . Austin showed that linguistic rules of language alone not adequately defining meaning of natural language sentences or words . . . Several subroutines revolving around main program like electrons revolving around nucleus of atom . . . Took years of determination and trial and error to refine program approximating complexities of English language . . . Running search engine and want to improve presentation of pages users getting when making search queries . . . Quickly finding information closely matching search queries . . . Trademarks or registering trademarks mentioned being property of respective holders . . . Rights Reserved trademarks or registering trademarks mentioned being property of respective holders

## Appendix IV—Sample Output--Rotated Index

This sample "Rotated Index" was created by the Syntactica API, focusing on the term "LANGUAGE".

Within the algorithm, the "depth" of the Rotated Index can be set on a scale of 1 to 7. The depth of this particular Rotated index was set at a level of "3".

LANGUAGE English including search output; Process natural  
LANGUAGE; Holy Grail for computer scientists being program processing  
natural  
LANGUAGE application programmer interface to be embedded in number of  
applications to return relevant outputs giving wide variety of natural  
language inputs; C  
LANGUAGE inputs; C language application programmer interface to be  
embedded in number of applications to return relevant outputs giving  
wide variety of natural  
LANGUAGE and literature; History in 1960s, Arnie Schulz being graduate  
student at University of Minnesota studying English  
LANGUAGE program being initial success for NIH because able to search  
and retrieve terms from medical journals; Assembler  
LANGUAGE to pick out words; Basic principle of program to use  
syntactic structures of English  
LANGUAGE alone not adequately defining meaning of natural language  
sentences or words; Austin showed that linguistic rules of  
LANGUAGE sentences or words; Austin showed that linguistic rules of  
language alone not adequately defining meaning of natural  
LANGUAGE not to be adequately circumscribed by dozen finite and rigid  
rules; Knowing early on natural  
LANGUAGE considering each word in depth; Meticulously going through  
entire dictionary of English  
LANGUAGE; Took years of determination and trial and error to refine  
program approximating complexities of English  
LANGUAGE largely by corrected by parents and teachers over years; Just  
as child learning  
LANGUAGE through finite system of rules; Attempts to create machines  
understanding  
LANGUAGE; Taking artist's eye to understand complexities of English  
LANGUAGE; Seamlessly fitting within programing that holds, searching,  
analysing or presents information in English