Creating Your Controlled Language

Developer and Analyst Guide

Rochelle Fisher

Creating Your Controlled Language: Developer and Analyst Guide

Rochelle Fisher

Publication date 20 August 2025

Abstract

This version of the documentation is for technical writers or developers who have Python installed, can run the scripts given in this document, and can continue with the linguistic analysis to create a proprietary controlled language.

Sample Sources:

- Rubicon Communications LLC. (Apr 25, 2025) netgate© Security Gateway Manual: Amazon AWS. © Copyright 2025 Rubicon Communications LLC. https://docs.netgate.com/manuals/pfsense/en/latest/aws-vpn-appliance-security-gateway-manual.pdf
- OpenCTI. (2025) OpenCTI User Guide: Manual Creations. © 2025 Filigran. All rights reserved. https://docs.opencti.io/latest/usage/manual-creation/
- MISP. (2024) MISP User Guide: A Threat Sharing Platform. GPL and CC-BY-SA 4.0 international.

Table of Contents

List of Tables

Introduction to Controlled Languages

A Controlled Language (CL) is set of rules for grammar, sentence length, and vocabulary. The rules define words and grammar as allowed or not allowed. This enforces consistency for all your documentation and conforms to or builds an industry standard for your products.

Intended Audience

This document is for a developer or technical writer with coding experience. Python installation is a requirement. If you are a linguist or responsible for the content of your organization, but do not have Python, you can share this document with a developer or IT. They can run the scripts, and you can continue with the CL itself when they are done.

Scope of Project

This is a personal project of Rochelle Fisher, August 2025. The goal is to make a sample of technical writing and a sample of a proprietary CL for cybersecurity.

If you use part, parts, or all of this project for personal use, for profit, or for an organization (non-profit or for-profit), please include a reference to this project or its documentation.

Reference Example:

```
Rochelle Fisher, Sample Cyber Security CL, version 1.0, August 2025.
```

Workflow

These are the procedures in this project:

1. Collect samples.

   You will create a CL from your documentation. This project creates a CL for cybersecurity from online, GPL, or unlicensed content.

2. Aggregate the samples to one text file.

3. Run the Python script on the text file, to get a CSV file of each word and its number of instances used.

   Optional Recommended: Run Python scripts to get default definitions and word context.

4. From the Python output, create the Dictionary file.

   This is the last step for a developer who will not work on the Dictionary. Up to this point, the steps are run only one time. If you repeat a step, all work done so far on the Dictionary by technical writers or linguistic analysts will be lost.

5. For each word, analyze usage for the one definition, one part of speech, allowed or not allowed, audience, and examples.

6. Create grammar and content rules.

   This project will start with ASD-STE 100 rules and then customize them.

7. Out of scope: Select an AI tool, or create a Python script, to test content for CL conformance (we like writer.com).

Collecting Samples

This is the cloudiest part for a sample project because we do not want to infringe on copyright laws. We will not use the content as-is, available for consumer use. We will use it only to get the vocabulary for our CL. When you create your CL, use your organization's documentation.

Sources:

- Rubicon Communications LLC. (Apr 25, 2025) netgate© Security Gateway Manual: Amazon AWS. © Copyright 2025 Rubicon Communications LLC. https://docs.netgate.com/manuals/pfsense/en/latest/aws-vpn-appliance-security-gateway-manual.pdf
- OpenCTI. (2025) OpenCTI User Guide: Manual Creations. © 2025 Filigran. All rights reserved. https://docs.opencti.io/latest/usage/manual-creation/
- MISP. (2024) MISP User Guide: A Threat Sharing Platform. GPL and CC-BY-SA 4.0 international.

You can collect your samples or complete content as PDF, DOC, DOCX, HTML, or text. Aggregate all the samples to one file. Convert the file to plain text. Keep this file. You will use it as input the Python script and as the main source for word analysis.

Creating the Word List

Create and Run the Word Counter Python Script

To create the dictionary, we start with the unique words used and how many times each word is used in our sample. This Python script will give us a file formatted for easy import to a spreadsheet.

Procedure 1. To create and run the python script word_counter.py:

1. Save your aggregated sample as `sampleAgg.txt`.
2. Copy this script in a text editor.
3. In the same folder as `sampleAgg.txt`, save the file as `word_counter.py`.
4. Run the python script:

   python word_counter.py > dictionary.csv

```
import re
import sys
from collections import Counter

def analyze_document_frequency(filepath: str = "sampleAgg.txt"):
    """
    Analyzes a text document to count word frequency.

    This function reads a text file, converts its content to lowercase,
    extracts words between 4 and 15 characters long,
    and counts the occurrences of each extracted word. It prints
    the words sorted by frequency in descending order, then
    alphabetically for ties.

    Args:
        filepath (str): The path to the input text file.
        Defaults to "sampleAgg.txt".

    Returns:
        None: Prints the word frequencies to standard output.
        Errors are printed to stderr.
    """
    try:
        # Open and read the document, converting content to lowercase.
        # 'utf-8' encoding is specified for broad compatibility.
        with open(filepath, 'r', encoding='utf-8') as document_file:
            text_string = document_file.read().lower()
    except FileNotFoundError:
        # Handle the case where the specified input file does not exist.
        print(f"Error: '{filepath}' not found.", file=sys.stderr)
        return
    except Exception as e:
        # Catch other general exceptions during file reading.
        print(f"Error while reading: {e}", file=sys.stderr)
        return
```

```
    # Use regular expression to find all words.
    # '\b' ensures whole words. '[a-z]{4,15}' matches lowercase
    # letters between 4 and 15 characters.
    match_pattern = re.findall(r'\b[a-z]{4,15}\b', text_string)

    # Count word occurrences using collections.
    # Counter for efficiency.
    word_counts = Counter(match_pattern)

    # Sort words: primary key is count (descending),
    # secondary key is word (ascending).
    # Lambda function creates a tuple for sorting:
    # (-count) for descending, then word for ascending.
    sorted_words = sorted(word_counts.items(), \
    key=lambda item: (-item[1], item[0]))

    print("Word Frequencies:")
    print("-----------------")
    # Iterate through the sorted words and
    # print each word and its count, separated by a pipe.
    for word, count in sorted_words:
        print(f"{word} | {count}")

if __name__ == "__main__":
    # Make sure the function runs only when
    # the script is executed directly,
    # not when imported as a module.
    analyze_document_frequency()
```

Note on Code Source: This code started with the python script in *Alchemy of Tomes* [Fisher (2020)], which does not work in the latest Python versions. We used Gemini (Google AI) to update this script.

Create the Dictionary File

With three small samples of cybersecurity documentation from different organizations, the Python script gave us 4755 unique words. Your sample will be much larger. Your goal is to create a cybersecurity dictionary of approximately 1,000 words. This does not include product names or company-specific words.

Prerequisites: Run the Python script.

Effort: Use the output file from the Python script to create the dictionary file. This will require approximately ten minutes.

Procedure 2. To create the dictionary file:

1. Import the output file to a spreadsheet.

   We will use Google Sheets. You can use Microsoft Excel or similar.

   Our Python script uses a pipe (|) as a delimiter between the word and its frequency of use. When you import the file, set the pipe as the delimiter.

2. Set the column headers.

   If you know that you will use a specific AI-driven tool, such as writer.com or jasper.ai, change the dictionary headers and values to work with the acquired tool.

If you do not have a checker tool yet, change row 1 to be these headers: `Word, Count, Part of Speech, Definition, Good Example, Bad Example, Allowed?, Audience, Alt1, Alt2`.

3. If there are rows between the header row and the first word, delete them.

4. In a separate tab, enter your audience personas in one column. Make sure `all` is in this list.

5. In the main tab, Audience column, set data validation to select the persona from the list.

The script sorted the output by highest frequency to lowest. Start with the words most used. These will be the easiest.

Add Default Definitions

The main principle of a controlled language is that every word is used with one definition and one part of speech. The definition of the word in the Dictionary is important and must exist. The technical writer could create the definition for each word as they get to it. That requires some time in dictionary.com or similar sites, to add definitions for basic words. It is true that the technical writer will have to work out the best definition of words specific to cybersecurity, but if you can add default definitions when you create the dictionary file, it can save a lot of time.

This code gets definitions of the words in the dictionary from the Free Dictionary API. Save this code as a python file and run it. Open the results in a Google Sheet. Replace the empty Definition column with the results. Format the cells to show the writer that these definitions must be reviewed. For example, make the background of the cell light red. The writer will change the color when they update the definition for cybersecurity.

Note: if the word is not in the free dictionary, the definition is empty. This is acceptable for the first run through. The technical writer will add the missing definition.

```
import requests
import json
import time
import os

def get_definition_from_free_dictionary_api(word):
    """
    Fetches the definition of a word from the Free Dictionary API.
    Returns a cleaned, single-line definition or an error/not-found message.
    """
    url = f"https://api.dictionaryapi.dev/api/v2/entries/en/{word}"

    try:
        response = requests.get(url)
        response.raise_for_status()
        # Raise an exception for HTTP errors (404 for not found)

        data = response.json()

        if isinstance(data, list) and data:
            # Prioritize a concise definition
            definitions_list = []
            if 'meanings' in data[0]:
                for meaning in data[0]['meanings']:
                    if 'definitions' in meaning:
                        for definition_obj in meaning['definitions']:
                            definition_text = \
                            definition_obj.get('definition')
                            if definition_text:
                                definitions_list.append \
                                (definition_text.replace \
                                ('\n', ' ').strip())
                                # Remove newlines in definition

                if definitions_list:
                    # Join the first few distinct definitions if available
                    # Take unique definitions to avoid repetition
                    unique_defs = []
                    for d in definitions_list:
```

```
                        if d not in unique_defs:
                            unique_defs.append(d)
                        if len(unique_defs) >= 2:
                        # Get up to 2 concise definitions
                            break
                    return "; ".join(unique_defs)
                return "No concise definition found."

            elif isinstance(data, dict) and data.get('title') \
            == 'No Definitions Found':
                return "No definition found for this word."
            else:
                return f"API returned unexpected data for \
                '{word}'."

        except requests.exceptions.RequestException as e:
            return f"Error: {e}"
        except json.JSONDecodeError:
            return f"Error: Invalid JSON response."
        except Exception as e:
            return f"An unexpected error occurred: {e}"

    def process_word_list_file(filepath):
        """
        Reads words from the file, fetches definitions, and writes them back
        to the same file with a comma delimiter.
        """
        temp_lines = []
        processed_count = 0

        try:
            # Read the existing content first
            with open(filepath, 'r', encoding='utf-8') as f:
                for line in f:
                    original_line = line.strip()
                    word = original_line.split(',')[0].strip()
                    # Get the word before the first comma

                    if not word: # Skip empty lines
                        temp_lines.append(original_line)
                        # Keep empty lines as they are
                        continue
                    """
                    Check if the line already contains a definition
                    (has a comma)
                    """
                    if ',' in original_line:
                     """
                     Assuming comma exists: it is already processed,
                     or we want to re-process it.
                     If you want to skip already defined words,
                     add a check here.
                     """
                        pass

                    print(f"Processing '{word}'...")
                    definition = get_definition_from_free_dictionary_api(word)
```

```
                    # Combine word and definition with a comma
                    new_line = f"{word},{definition}"
                    temp_lines.append(new_line)
                    processed_count += 1
                    time.sleep(0.5) # Be polite to the API

            # Write all processed lines back to the same file
            with open(filepath, 'w', encoding='utf-8') as f:
                for line in temp_lines:
                    f.write(line + '\n') # Add newline back for each line

            print(f"\nSuccessfully processed {processed_count} \
             words and updated '{filepath}'.")

        except FileNotFoundError:
            print(f"Error: The file '{filepath}' was not found.")
        except Exception as e:
            print(f"An error occurred during file processing: {e}")

    if __name__ == "__main__":
        # Define the name of your word list file
        word_list_filename = "UWords.txt"

        # Construct the full path to the words file
        script_dir = os.path.dirname(__file__)
        word_file_path = os.path.join(script_dir, word_list_filename)

        # File exists? If not, create it with example words
        if not os.path.exists(word_file_path):
            print(f"'{word_list_filename}' not found. \
             Creating it with example words.")
            with open(word_file_path, 'w', encoding='utf-8') as f:
                f.write("apple\n")
                f.write("run\n")
                f.write("beautiful\n")
                f.write("galaxy\n")
                f.write("nonexistentword123\n")
                f.write("serendipity\n")
                f.write("ubiquitous\n")
            print("Example words added. Run again to fetch definitions.")
        else:
            # If the file exists, process it
            process_word_list_file(word_file_path)
```

Script created by Gemini, Google AI.

Provide Context

The main principle of a controlled language is that every word is used with one definition and one part of speech (PoS). The PoS of the word in the Dictionary is important and must exist. The technical writer will review all the uses of each word in the aggregated sample, to analyze which PoS is most used, which best fits the style guide, and which is least ambiguous. The technical writer will use a search-all feature in Adobe PDF or in a feature-rich text editor, to get all instances of each word, with context, to analyze usage and make a decision. You can save the time of each lookup with a script that gets the context results for the technical writer.

This code gets the context of each word: 5 words before the word and 5 after. You can change the number of words when you run the script.

1. Save the words, the [0] item in each line of the CSV or the first column of the Dictionary file, as a file.

2. Save this code as a Python file and run it:

   python wordContext.py > words.csv aggregate.txt Context.txt 5

3. Send the results to the technical writer.

```
import re

def extract_word_context_to_file(words_file, content_file, output_file, \
 context_word_count=5):
    """
    Finds words from a list in a content file and extracts a context of
    'context_word_count' words before and after the found word.
    Writes the results to a new output file.

    Args:
        words_file (str): Path to file with words, one in each line.
        content_file (str): Path to the aggregate file.
        output_file (str): Pathname of file with results.
        context_word_count (int): Number of words before and after.
    """
    try:
        with open(words_file, 'r', encoding='utf-8') as wf:
            target_words = [word.strip() for word in wf if word.strip()]

        with open(content_file, 'r', encoding='utf-8') as cf:
            content = cf.read()
        """
        Split content into words, keep punctuation for simpler splitting.
        A more robust solution might separate punctuation.
        """
        all_words = re.findall(r'\b\w+\b|[^\w\s]', content.lower()) \
        # Lowercase for matching

        found_contexts = []

        for target_word in target_words:
            # Iterate through the content to find matches
            for i, word_in_content in enumerate(all_words):
                if word_in_content == target_word.lower(): \
                # Case-insensitive matching
```

```
                        # Determine start and end index for context
                        start_index = max(0, i - context_word_count)
                        end_index = min(len(all_words), \
                         i + context_word_count + 1)

                        # Extract the context words
                        context = all_words[start_index:end_index]

                        # Reconstruct the context string
                        context_string = ' '.join(context)

                        found_contexts.append(f"Target Word: {target_word}\n \
                         Context: {context_string.strip()}\n")

            with open(output_file, 'w', encoding='utf-8') as of:
                if found_contexts:
                    for item in found_contexts:
                        of.write(item + "-----\n") \
                        # Added a delimiter for readability between entries
                    print(f"Contexts successfully written to '{output_file}'.")
                else:
                    of.write(f"No matching words found for words in \
                     '{words_file}' within '{content_file}'.")
                    print(f"No matching words found. '{output_file}' \
                    created with a note.")

        except FileNotFoundError:
            print(f"Error: Required file not found. \
            Make sure '{words_file}' and '{content_file}' exist.")
        except Exception as e:
            print(f"An error occurred: {e}")

if __name__ == "__main__":
    UWORDS_FILE = "UWords.txt"
    SAMPLE_AGG_FILE = "SampleAgg.txt"
    OUTPUT_CONTEXT_FILE = "FoundWordContexts.txt" \
    # New output file for contexts
    CONTEXT_WORDS = 5 \
    # Number of words before and after

    extract_word_context_to_file \
     (UWORDS_FILE, SAMPLE_AGG_FILE, OUTPUT_CONTEXT_FILE, CONTEXT_WORDS)
```

Script created by Gemini, Google AI.

Getting Started with the Dictionary

Work the First Words

The first words that you set up in your dictionary will be the easiest and will have the most impact.

Prerequisites: Have the dictionary file in a spreadsheet. Make sure it is sorted for COUNT.

Effort: This will require less than an hour.

Procedure 3. To work through the most used words in your dictionary:

1. In the word with the highest count, set the Part of Speech (PoS).

   A typical result for the most used words are the names of your organization or product. ASD-STE calls these *technical names*. Set the PoS of the technical names in the top results as `Name`.

   This lets you filter for proper nouns which change more often than regular nouns.

   For example, the company name will change if your organization delivers a white label product.

2. Enter the PoS for the other most common words, such as *the that this from*. When you get to a word that is may be used in multiple parts of speech and is not a common word for all English content, skip it.

3. In the Allowed column, enter `T` (for true) or `F` (for false).

   Most of these first words will be allowed.

4. In the Audience column, enter `all` or select a persona from the list, if you are sure this word will be allowed only for this persona.

Examples of the First Words

In our example, one of the product or company names is the word with the highest frequency. This should be your most used word too.

Table 1. The Most Used Word

Word	PoS	Definition	Allowed?	Audience
company or product name	`Name`	*Add your marketing definition.*	`T`	`all`

In the top words, we have many that are clearly to be allowed: *this that will with from your*. These are pronouns, prepositions, and modular verbs. In a technical writing dictionary for native speakers, it does not give a lot of information to define the PoS. Is *that* a pronoun, adverb, conjunction, or determiner? Enter any reasonable value for the PoS.

In our style guide, we set a rule that we do not use minimalist rules. If *that* helps make a sentence easier to understand, we use it.

Table 2. Most Used Words

Word	Count	PoS	Definition	Allowed?	Audience
`this`	`730`	`pronoun`	*blank*	`T`	`all`
`that`	`706`	`pronoun`	*blank*	`T`	`all`

Word	Count	PoS	Definition	Allowed?	Audience
`will`	`667`	`verb`	*sets future tense of main verb (we defined this word to differentiate from its other definitions)*	`T`	`all`
`with`	`612`	`preposition`	*blank*	`T`	`all`
`from`	`501`	`preposition`	*blank*	`T`	`all`
`your`	`376`	`pronoun`	*blank*	`T`	`all`

We have words that are industry standard for software technical writing: *introduction user data attributes attribute example organisation*.

- We set the PoS and Definition for our words. We set Allowed to `T` and Audience to `all`.
- Notice that *attribute* and *attributes* are two items.

  We restrict the values of the Word and ALT columns to one word or phrase, to prepare for a checker tool that might be a simple Python script. This is different from the way ASD-STE is written, but if you begin with this restriction, it will be easier later.

  If you know that you will use a specific AI-driven tool, such as writer.com or jasper.ai, change the dictionary headers and values to work with the acquired tool.

- See *organisation*. We know we need this word, but this spelling is British. We know our rules will tell us to use American spelling. In this case, we will make a decision for our dictionary without analysis, or to put it more accurately, despite analysis. The spelling of *organisation* is the most commonly used form, but we will not use it. We will use the American spelling.

  1. Sort the complete range of dictionary alphabetically, by *Word*.
  2. Set *organization* and *organizations* as allowed (`T` in Allowed).
  3. Set the British *organisation* and *organisations* as not allowed (`F` in Allowed).
  4. In the allowed words, set the definition: `nonspecific body of people with a purpose`.

     We will use *organization* for a company, nonprofit org, military base, and all similar bodies.
  5. In ALT1 for *organisation* and *organisations*, enter *organization* and *organizations*.

  When we sorted by Word to see all the issues of *organi**, we saw *organizational*, with a COUNT of `1`. Why not add *organizational* on the fly? Answer: It is used only one time. We will analyze the text and find a more common rewrite in that one sentence. Or maybe that one instance is for *Organizational Unit (OU)* in Active Directory. If so, we will make that phrase a "word".

NEXT: When you are done with the most used words that you want to complete now (best practice: stop after an hour), sort the range by Count. We will analyze the words that are most likely to be NOT allowed: words used only one time. After that, go through the words in the order that best works for you.

Analyzing Words

Work the Next Words

You will analyze how your dictionary words are used in your sample.

Procedure 4. To analyze words:

1. If you have a file created by the Context script, open it. It shows your words with 5 words before and after it, to see context.

   If you do not have that file and do not want to create it:

   a. Open your aggregated sample document that you used as input for the Python script.

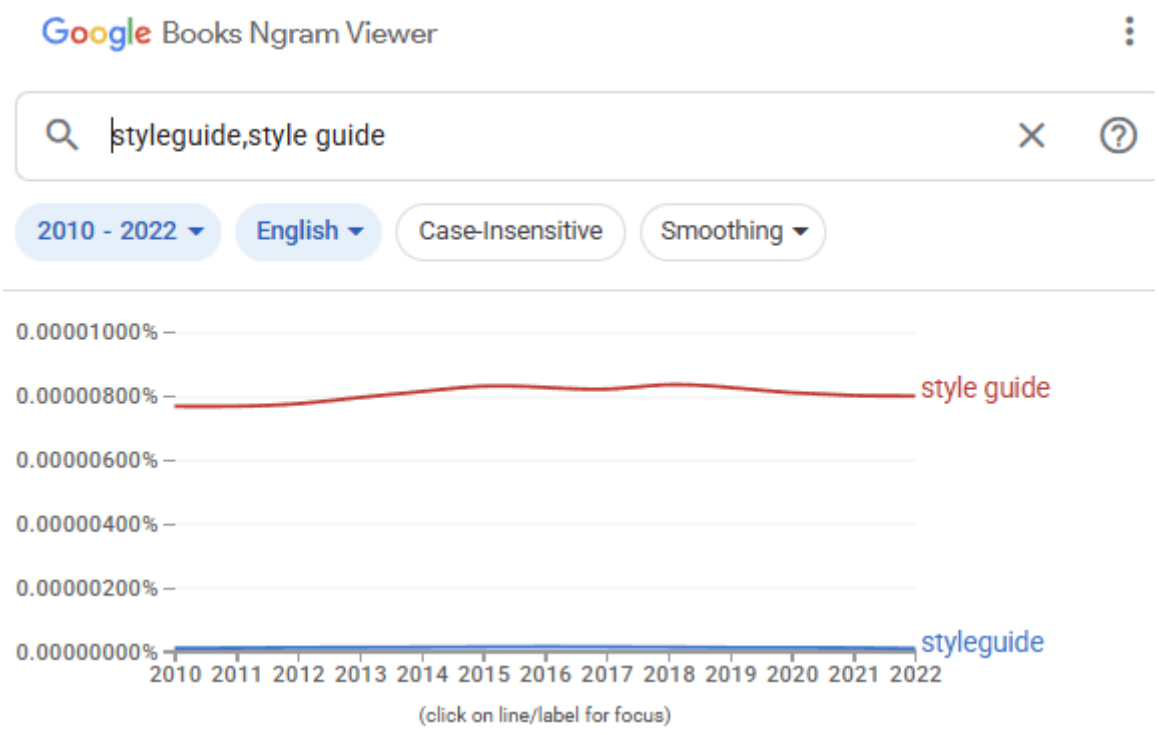
   b. Search for the word to analyze. We like to use Notepad++, to get all the results in a list, each result with context.

2. From the context, see if the word is used in more than one part of speech. If it is, and it is not obvious which is the most common, count the instances of each PoS.

   I usually paste or import the results in a spreadsheet and then use COUNTIF, filter, or pivot table.

3. See if the word, in the allowed PoS, is used with more than one definition. If it is, note the definitions.

   Use counts, industry standards, and internet searches to get the best definition. Get the ASD-STE 100 document and see what their dictionary says about the word. NGRAM is also a good tool.



4. See if the draft definition works in a large random sample, or in all uses. If you decide to make the word not allowed, see if the chosen alternate word or syntax works.

Best Practice: When you write a definition, do not use the word in it. This will help you find words used as synonyms, which is not what we want. In a CL, not only does each word have one definition, but each definition is represented by one word.

For example, your documentation uses the past participles (past tense verb that functions as an adjective) *protected* and *secured* to mean the same thing. When you define them, the definition is `endpoint on which Product services run or network on which Product services run on all endpoints`. You decide that works for *protected*, but when you get to *secured*, you change the definition to be: `endpoint on which remediation ran to solve security vulnerability or exploit`. With these specific definitions, you can use both words, each with more meaning for the user.

You may find that two or more words are used for the same definition and PoS, and you can restrict your dictionary to only one. This will make it easier for users to read and to follow the instructions. For example, if you use *configure* to mean *enter*, you could confuse your users when the docs say, `configure the IP address`. What is there to configure? Is it not enough to simply enter the IP address? Can you remove *configure* and use *create*, *enter*, and other more specific words for the action?

5. Update the rules, definition, and PoS. Enter good and bad examples.

   Best practice: You do not need a bad example for allowed words. It is best to use it for words that are not allowed. See ASD-STE 100 for examples.

6. If not allowed, make sure the words you set as alternatives are allowed and configured in the dictionary for PoS and definition, with examples.

Examples of Word Analysis

Let's start with *organizational*.

Procedure 5. To set *organizational* as a word in our CL:

1. Open the file with all your textual content. We named this file `sampleAgg.txt`.

2. Search for *organizational*.

   We found this sentence (owned by one of the organizations from which we used samples):

   > Produce intelligence that will be embedded into organizational workflows and would serve decision-makers.

3. Analyzing the use of *organizational*, we see ambiguity. Does the author mean that the workflows are organized? That they are for the organization? That there are different workflows for different groups in the hierarchy?

   If we remove the word, it does not change the meaning, as far as we can see. We decide that this word is not allowed. If we had access to the SME, we would discuss their meaning and find alternatives.

   Suggestion: You have access to your SMEs. Set your CL words as best as you can. Then, discuss multiple words with similar issues. Edit your CL for alternative words and other decisions.

The next task is to work through the top words that are not obviously allowed or not allowed. Sort the dictionary by Count.

Our next word to work through is *event*.

Procedure 6. Analyzing *event*:

1. Search for *event* in the sample content or context file.

   Our sample has 1417 results, for *event* and *events*.

2. Read the results. Make the definition draft. For example:

   ```
   a security incident detected on the secured network
   ```

3. Fix the definition as you read more result lines.

   For example: The fifth result is for a procedure to create an event in the security application. We learn that an event always includes threat intelligence data and usage. We update the definition to:

   > object that contains a cybersecurity incident, report, or finding, with attributes, identifiers, and other data for analysis, prevention, and mitigation

   We see that *event* is used with other definitions.

   - A system or user action

     This is a technical name for a specific product. For our general cybersecurity dictionary, it does not fit. We can communicate actions in errors and logs with their names, without the use of *event*.

   - A phrase: "in the event this happens"

     This is a verbose phrase to mean *if*.

   - CLI or API commands and pathnames. Our checker tool must ignore code, filenames, and pathnames.

     This is an advantage of an XML technical writing tool. We can use elements that set content by type: <code>, <codeblock>, <filename>, <pathname>, and similar. We can then set the checker tool to ignore text in these elements. We add a rule to our style guide to use these elements.

4. Sort the dictionary by Word, to configure all similar words (plural with singular, commands, and so on).

5. In *event*, we enter the definition, good example, and bad example. In *events*, enter: `plural of event`.

6. A non-AI checker tool (and even some AI tools) cannot see the difference of meanings in the use of *event* with our specific definition or the use of the non-allowed definition. In Rule, enter: `Do not use to mean action of a user or server`.

   Our checker tool will show rules to help writers be consistent.

7. For the words that start with *event* and are obviously commands or pathnames, set the PoS to `command` and Allowed to `T`.

   Add a row for the NOT allowed phrase `in the event` and make sure *if* is allowed.

In this table, we don't show Audience (`all` for each row) or Allowed? (`T` for each row)

Table 1. Dictionary Rows for *event*

Word	Definition	PoS	Rule / ALT1
event	object that contains a cybersecurity inci-	noun	Do not use to mean action of a user or server

Word	Definition	PoS	Rule / ALT1
	dent, report, or finding, with attributes, identifiers, and other data for analysis, prevention, and mitigation		
events	plural of EVENT	noun	Do not use to mean action of a user or server
eventblocklists		command	
eventgraph		command	
eventid		command	
eventinfo		command	
eventreports		command	
eventtag		command	

Table 2. Dictionary Rows to Replace *in the event*

Word	Count	Definition	PoS	Allowed?	Rule / ALT1
in the event	172		phrase	F	IF
if	added	introduces condition	conjunction	T	Separate the condition from the action or result with a comma

Procedure 7. Analyzing *type*:

This word is an excellent example. It is used in different parts of speech with different definitions in writing and native speaking. To control our written language, we must restrict this word to one PoS and one definition. Or we can decide to set it to not allowed, to be replaced with specific words.

1. Search for *type* in the sample.

2. Skim the hits with context. If it is not clear which PoS is mostly used, enter the PoS of each row. We found that it was most often used as a noun or technical name, but there were sentences with it used as a verb.

type yes when	verb
Set the IPv4 Configuration Type to Static IPv4	name
corresponds to the desired type of instance	noun

3. Add a row for *type* as a verb and set it to not allowed.

Table 3. Dictionary Rows for *type* as a verb

Word	Count	Definition	PoS	Good Example	Bad Example	Allowed?	Audience	Rule	ALT1
type	added	1) to categorize; 2) to enter in-	verb	enter yes	type yes	F	all	Do not use as a verb	ENTER

Word	Count	Definition	PoS	Good Example	Bad Example	Allowed?	Audience	Rule	ALT1
		put with a keyboard							

4. Make sure *enter* is allowed.

Table 4. Dictionary Rows for *enter*

Word	Count	Definition	PoS	Good Example	Bad Example	Allowed?	Audience
enter	added	to input values	verb	enter yes		T	all

5. We see that there many uses of *type* in the GUI and CLI. We could try to make it a technical name for user interface (UX) creators. The word *type* would be allowed in micro-copy and coding but not in technical writing. We would add a row for the UX persona in audience that would be Allowed for *type* as a `Name`. We would add another row for the other audiences that makes use of *type* as a noun not allowed.

   But we see in the results that *type* is used in text that cannot easily be rewritten. We must allow it for everyone, but only with the required definition, as an object in the product.

Table 5. Dictionary Rows for *type* as a noun

Word	Count	Definition	PoS	Good Example	Bad Example	Allowed?	Audience	Rule
type	added	product group of objects with common characteristics	Name	some attribute types require associated events	some types of attributes are more complex	T	all	Do not use as general "kind"; use only as a category specific to the product

6. Make sure the style guide rule that all text on the interface (GUI, API, or CLI) must be wrapped in an element, such as <code>, <codeblock>, <guilabel>. We can then set the checker tool to ignore text in these elements. If your checker tool shows the rules, it will not show this rule for interface labels, where it would cause user fatigue and be ignored when it is necessary.

7. We look through the uses of *type* to mean a general group of people or things having common characteristics. We can rewrite those.

   - Given the text:

     > supports various relationship types, and their usage depends on the entity types being linked

     This one sentence uses *type* with two definitions. The first can be removed. The second fits the allowed definition.

     Controlled version: `supports various relationships, and their usage requires linked entity types`

- Given the text:

  > there are two types of admins: Org Admins and Site Admins

  The use of *type* is not required. If the SME does not like *level*, we can change it to a different word (*set*, *permissions*). Also note that we remove *two*. It is always best to not enumerate features, to make sure you do not create a conflict in the text when a new feature is added.

  Controlled version: `there are different admin levels: Org Admins and Site Admins`

- Given the text:

  > the type of storage used by Product can have an impact

  The full text discussed SSD devices and feed caching technology. We guess that "type" meant hardware and configuration.

  Controlled version: `The storage you use can have an impact.` OR `your storage hardware and algorithm can impact Product` OR `storage hardware can impact Product`

What Now?

You have a dictionary in progress. You must go through all the words, update the definitions and rules as you go.

When you are done with the top twenty or thirty words, sort your dictionary to see at the words with only one count. These words will be easy to set as not allowed, commands, or misspellings.

- If the word is not allowed, set its values. Make sure the alternative words are allowed.
- If the word is a command, in PoS, enter `command`. Set it to Allowed = `T`. Make sure your style guide says to wrap commands in relevant elements.
- If the word is a mistake, in PoS, enter `misspelling`. When your content is fixed for this mistake, you can remove it from the dictionary.
- If the word is correct and allowed, investigate. If it is a word for a specific audience, set it to Allowed for that audience. If a product owner or sponsor wants it for everyone, discuss why.

  For example, *absent* is used one time, in the phrase *absent a route*. This is a specific network configuration action. It is correct. We allow it for the user, sysadmin, and internal audiences. But we do not want it in the UX micro-copy or C-level marketing.

You will need a checker tool. You can acquire the tool before you complete the dictionary. You can ask a Python developer to make a script that returns an email or a webpage with results (unallowed words used, rules as reminders on allowed words, and unknown words). Or you can acquire an AI tool that integrates with your source CCMS. The important thing is that your dictionary is used. It is a dynamic tool for all content creators in your organization.