# Creating a Data Dictionary from SAS Scripts using AST

## 1. Overview

This document provides a detailed guide to create a Data Dictionary by parsing SAS scripts into an Abstract Syntax Tree (AST) and extracting metadata. The process involves parsing the SAS code, traversing the AST, extracting dataset and column details, and exporting them into a structured format such as CSV, Excel, or Word.

## 2. High-Level Architecture

The architecture to achieve this can be broken into these major components:  
• Input Layer: SAS code files (.sas) that contain data step definitions and PROC SQL statements.  
• Parser: An AST parser (e.g., ANTLR with SAS grammar) to convert SAS code into a structured JSON or tree format.  
• Metadata Extractor: A program (Python, etc.) that walks through the AST to collect datasets, columns, data types, labels, and derivations.  
• Output Layer: Generates a Data Dictionary in editable formats like CSV, XLSX, or DOCX.

## 3. Step-by-Step Procedure

### Step 1: Parse SAS Code into AST

• Use a SAS grammar (for example, ANTLR4 SAS grammar) or another SAS parser to convert SAS scripts into an AST.  
• The AST is typically represented as a JSON-like hierarchical structure describing each DATA step, PROC SQL block, variables, and expressions.  
• Example: {"data\_steps":[{"dataset":"work.sales","columns":[{"name":"id","type":"numeric"}, ...]}]}

### Step 2: Traverse the AST

• Write a traversal function (in Python or another language) to recursively walk through the AST nodes.  
• Identify nodes such as DATA steps, PROC SQL create table statements, length/attrib statements, assignments, and select lists.  
• Extract dataset names, column names, types, lengths, labels, formats, and derivations.

### Step 3: Build the Data Dictionary Structure

• Store extracted metadata in a tabular structure.  
• Recommended columns: dataset, column, data\_type, length, label, format, informat, origin (DATA step or PROC SQL), derivation, notes.  
• Deduplicate rows for the same dataset/column and merge derivation or note fields.

### Step 4: Export the Data Dictionary

• Use libraries such as pandas to create CSV or Excel files.  
• Use python-docx to create an editable Word document containing the data dictionary.  
• Ensure each row represents a variable with all relevant metadata.

## 4. Sample Python Implementation

The following pseudo-code outlines the extraction process:  
```  
import json, pandas as pd  
with open('sas\_ast.json') as f:  
 ast = json.load(f)  
def extract(ast):  
 rows = []  
 for step in ast.get('data\_steps', []):  
 dataset = step['dataset']  
 for col in step.get('columns', []):  
 rows.append({  
 'dataset': dataset,  
 'column': col['name'],  
 'data\_type': col.get('type','unknown')  
 })  
 return pd.DataFrame(rows)  
df = extract(ast)  
df.to\_excel('data\_dictionary.xlsx', index=False)  
```  
This code assumes you already have the AST as a JSON file.

## 5. Best Practices and Considerations

• Ensure the SAS parser handles all SAS language features used in your code.  
• Handle macros and dynamic code carefully, as they may require preprocessing.  
• Validate the final data dictionary by comparing with actual SAS dataset metadata (PROC CONTENTS) if available.  
• Consider incremental updates: regenerate the dictionary when SAS code changes.

## 6. Summary

By following this architecture—parsing SAS code to an AST, extracting dataset and column metadata, and exporting results—you can maintain an up-to-date, automated Data Dictionary that improves documentation and governance of your SAS environment.