# Manual do Usuário do COBOL Analyzer

## 1. Introdução

O COBOL Analyzer é uma ferramenta avançada para análise de código COBOL, projetada para auxiliar no entendimento, documentação e modernização de sistemas legados. Ele utiliza modelos de linguagem (LLMs) para gerar análises detalhadas de programas COBOL, copybooks e JCL, fornecendo insights sobre regras de negócio, estrutura do código, variáveis e muito mais.

## 2. Instalação

Para instalar o COBOL Analyzer, siga os passos abaixo:

1. **Pré-requisitos:** Certifique-se de ter o Python 3.8 ou superior instalado em seu sistema.
2. **Instalação do pacote:**

pip install cobol-to-docs

1. Se você estiver instalando a partir do arquivo .whl local (fornecido após a compilação):

pip install /caminho/para/cobol\_to\_docs-3.1.0-py3-none-any.whl

## 3. Uso Básico

O COBOL Analyzer é executado via linha de comando. A sintaxe básica é:

cobol-to-docs [OPÇÕES]

## 4. Parâmetros Disponíveis

Abaixo estão todos os parâmetros de linha de comando disponíveis para o COBOL Analyzer:

* --fontes <PATH>:
  + **Descrição:** Caminho para um arquivo contendo programas COBOL. Dependendo do uso de --content-files, este arquivo pode conter o código COBOL diretamente ou uma lista de caminhos para arquivos COBOL.
  + **Tipo:** Path
  + **Obrigatório:** Não
* --books <PATH>:
  + **Descrição:** Caminho para um arquivo contendo copybooks COBOL. Semelhante a --fontes, este arquivo pode conter o código do copybook diretamente ou uma lista de caminhos para arquivos de copybook.
  + **Tipo:** Path
  + **Obrigatório:** Não
* --content-files:
  + **Descrição:** Uma flag que, quando presente, indica que os arquivos especificados por --fontes e --books contêm o código COBOL (ou copybook) diretamente, e não uma lista de caminhos para esses arquivos. Se omitido, os arquivos são tratados como listas de caminhos.
  + **Tipo:** boolean (flag)
  + **Obrigatório:** Não
* --output-dir <PATH>:
  + **Descrição:** Diretório onde os resultados da análise serão salvos. Se o diretório não existir, ele será criado.
  + **Tipo:** Path
  + **Padrão:** ./analysis\_results
  + **Obrigatório:** Não
* --log-level <LEVEL>:
  + **Descrição:** Define o nível de detalhe dos logs gerados pela aplicação.
  + **Tipo:** string
  + **Valores possíveis:** DEBUG, INFO, WARNING, ERROR, CRITICAL
  + **Padrão:** INFO
  + **Obrigatório:** Não
* --deep-analysis:
  + **Descrição:** Uma flag que, quando presente, habilita uma análise mais aprofundada dos programas COBOL. Esta opção pode consumir mais recursos e tempo.
  + **Tipo:** boolean (flag)
  + **Obrigatório:** Não
* --prompt-set <NAME>:
  + **Descrição:** Nome do conjunto de prompts a ser utilizado para a análise. Isso permite customizar o tipo de análise gerada pelo LLM.
  + **Tipo:** string
  + **Padrão:** default
  + **Obrigatório:** Não
* --llm-model <MODEL\_NAME>:
  + **Descrição:** Nome do modelo de LLM (Large Language Model) a ser utilizado para realizar a análise. Modelos específicos podem ter diferentes capacidades e custos.
  + **Tipo:** string
  + **Padrão:** gemini-1.5-pro-latest
  + **Obrigatório:** Não
* --provider <PROVIDER\_NAME>:
  + **Descrição:** Nome do provedor de IA a ser utilizado. Permite alternar entre diferentes implementações de provedores (ex: mock para testes, ou provedores reais).
  + **Tipo:** string
  + **Padrão:** enhanced\_mock
  + **Obrigatório:** Não

## 5. Formas de Uso e Exemplos

O COBOL Analyzer suporta diversas formas de entrada, desde arquivos únicos até listas de programas e copybooks. Abaixo, detalhamos os cenários de uso mais comuns.

### 5.1. Análise de um Único Programa COBOL

Para analisar um único programa COBOL, você pode passar o código do programa diretamente através de um arquivo e usar a flag --content-files.

**Exemplo:**

Crie um arquivo meu\_programa.cbl com o conteúdo do seu programa COBOL:

IDENTIFICATION DIVISION.

PROGRAM-ID. MEUPROG.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-VAR PIC X(10) VALUE 'HELLO'.

PROCEDURE DIVISION.

DISPLAY 'WS-VAR: ' WS-VAR.

STOP RUN.

Então, execute o analyzer:

cobol-to-docs --fontes meu\_programa.cbl --content-files --output-dir ./resultados\_prog\_unico

### 5.2. Análise de Múltiplos Programas COBOL (Lista de Caminhos)

Para analisar vários programas COBOL, você pode criar um arquivo que lista os caminhos para cada programa.

**Exemplo:**

Crie os arquivos PROG1.CBL e PROG2.CBL com o conteúdo dos seus programas COBOL.

PROG1.CBL:

IDENTIFICATION DIVISION.

PROGRAM-ID. PROG1.

PROCEDURE DIVISION.

DISPLAY 'Programa 1 executado'.

STOP RUN.

PROG2.CBL:

IDENTIFICATION DIVISION.

PROGRAM-ID. PROG2.

PROCEDURE DIVISION.

DISPLAY 'Programa 2 executado'.

STOP RUN.

Crie um arquivo lista\_programas.txt com os caminhos para esses programas (um por linha):

/caminho/para/PROG1.CBL

/caminho/para/PROG2.CBL

Então, execute o analyzer (sem a flag --content-files):

cobol-to-docs --fontes lista\_programas.txt --output-dir ./resultados\_multi\_prog

### 5.3. Análise com Copybooks

Copybooks são essenciais em muitos programas COBOL. O analyzer pode processá-los junto com os programas.

**Exemplo:**

Crie um arquivo MEUPROG.CBL que usa um copybook:

IDENTIFICATION DIVISION.

PROGRAM-ID. MEUPROG.

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY 'MEUCOPY.CPY'.

PROCEDURE DIVISION.

DISPLAY 'WS-CAMPO: ' WS-CAMPO.

STOP RUN.

Crie um arquivo MEUCOPY.CPY com o conteúdo do copybook:

01 MEU-COPYBOOK.

05 WS-CAMPO PIC X(15) VALUE 'DADOS DO COPYBOOK'.

Crie um arquivo lista\_copybooks.txt com o caminho para o copybook:

/caminho/para/MEUCOPY.CPY

Crie um arquivo lista\_programas.txt com o caminho para o programa:

/caminho/para/MEUPROG.CBL

Então, execute o analyzer:

cobol-to-docs --fontes lista\_programas.txt --books lista\_copybooks.txt --output-dir ./resultados\_com\_copybooks

### 5.4. Análise de JCL (Job Control Language)

Embora o main.py atual não tenha um parâmetro explícito para JCL, a arquitetura subjacente permite a extensão para análise de JCL. Para este manual, vamos considerar um cenário onde o JCL seria passado de forma similar a um programa, se a funcionalidade for implementada no futuro.

**Nota:** A funcionalidade de análise de JCL não está diretamente exposta como um parâmetro --jcl no main.py atual. No entanto, a estrutura permite que o InputHandler seja estendido para carregar e processar arquivos JCL, e o ProgramAnalyzer para analisá-los. Se desejar esta funcionalidade, ela precisaria ser adicionada ao main.py e aos componentes de análise.

Se a funcionalidade de JCL fosse adicionada, um exemplo de uso seria:

Crie um arquivo MEUJOB.JCL:

//MEUJOB JOB (ACCOUNT),'NOME',CLASS=A,MSGCLASS=X

//STEP1 EXEC PGM=IEBGENER

//SYSUT1 DD DSN=INPUT.FILE,DISP=SHR

//SYSUT2 DD DSN=OUTPUT.FILE,DISP=(NEW,CATLG),UNIT=SYSDA,SPACE=(TRK,(1,1))

//SYSPRINT DD SYSOUT=\*

//SYSIN DD DUMMY

E um arquivo lista\_jcl.txt:

/caminho/para/MEUJOB.JCL

Então, a execução (com um parâmetro --jcl hipotético) seria:

cobol-to-docs --jcl lista\_jcl.txt --output-dir ./resultados\_jcl

## 6. Saída da Análise

Os resultados da análise serão salvos no diretório especificado por --output-dir. Cada programa analisado terá um arquivo Markdown correspondente com os detalhes da análise, incluindo:

* Visão Geral do Programa
* Regras de Negócio Identificadas
* Variáveis e Estruturas de Dados
* Fluxo de Controle (se deep-analysis ativado)
* Resumo de Estatísticas

## 7. Desenvolvimento e Extensão

O COBOL Analyzer é modular, permitindo fácil extensão para:

* Adicionar novos provedores de IA.
* Customizar conjuntos de prompts para diferentes tipos de análise.
* Implementar novas funcionalidades de análise (ex: JCL, análise de desempenho).

Consulte a documentação interna do projeto para detalhes sobre a arquitetura e como estender suas capacidades.