

Subject: デバッグプロンプト
From: 山本 次郎 <yamajiro2007@gmail.com>
Date: 2025/11/24 22:50
To: "j-yamamoto@icej.co.jp" <j-yamamoto@icej.co.jp>

以下に、「AI機能付き ST言語デバッガアプリ」を開発するための完全版プロンプトを記述します。
これは **アプリ全体の仕様定義** → **機能一覧** → **UI/UX** → **API設計** → **コード生成ポリシー** → **実装要件** をすべて含む「万能プロンプト」です。

このまま LLM に入力すれば、
アプリケーションの設計・コーディング・機能生成・改善案の生成を行えるようになります。

AI機能付き ST言語デバッガアプリ開発プロンプト（完全版）

以下は、三菱電機 PLC（GX Works2/3）向けの **AI付き ST言語デバッガ・コード修正支援ツール** を開発するための包括的仕様です。
このプロジェクトの目的は、STプログラムの解析・デバッグ・改善を AI によって支援し、PLCエンジニアがコード状態を迅速に理解できるようにすることです。
以下の要件に従って、アプリケーションの設計書、データ構造、アルゴリズム、コード、UI、API、改善案を生成してください。

1. アプリケーション名

AI-ST Debugger Pro

2. 対応対象

- 三菱電機 PLC
- GX Works2 / GX Works3
- ST言語（IEC 61131-3 Structured Text）
- 付随する PRG / FNC / FB / Global & Local Variables

3. インポート機能（プロジェクト構造の読み込み）

以下を読み込み可能にする：

- POUファイル（.POU）
- STコード（.ST, .TXT）
- Program（.PRG）
- Function Block（.FNC / .FB）
- 変数定義ファイル（CSV/TXT）
- ライブラリ参照
- コメント情報

要件

- インポート後に内部 AST（構文木）を生成
- POU 間の依存関係グラフを生成
- FB インスタンスの関連構造を可視化
- コード履歴管理（Git 互換）

4. データ取り込み機能（GX Works からのログ）

以下のファイルを取り込む：

✓ 1. 変数値スナップショット（CSV）

- 変数名
- デバイスアドレス
- 現在値
- データ型

✓ 2. トレースログ（CSV）

- タイムスタンプ
- 実行された行番号
- 実行ブロック名
- トリガーイベント
- 値の変化記録

✓ 3. エラーログ（診断ログ）

- エラー番号
- メッセージ
- 詳細パラメータ
- 発生時刻
- 関連デバイス

5. UI（画面）の要件

AI付きデバッガの UI 仕様：

① STコード表示パネル

- 行番号付き
- 色分けシンタックスハイライト
- 変数値をリアルタイム表示（インラインオーバーレイ）

② 実行箇所ハイライト

- トレースログに基づいて「現在実行中/最後に実行した行」をハイライト
- 条件分岐の通過ルートをカラー表示

③ 変数パネル

- ウォッチリスト
- 変化グラフ（折れ線）
- 異常値の検出（AIによる閾値推定）

④ エラー解析パネル

- エラーログ
- 原因推定
- 影響範囲の表示
- 修正候補

⑤ アプリ内 AI チャット

- コード質問
- 原因分析
- 改修方法の説明
- 新コード作成

6. AIによる解析機能

LLM を以下の用途で使用：

✓ 原因分析

- STコードの AST
- 実行トレース
- 変数の異常値
- エラーログ

から以下を判定：

- 停止原因（ロジック/値/デバイス/順序）
- どのFBで異常が発生したか
- 関連変数の一覧
- 他の影響箇所

✓ 修正案生成

- 問題箇所の改善案
- 修正後の ST ソースコード生成
- コードの最適化（簡潔化、可読性向上、例外処理追加）
- 安全性チェック

✓ 改善案の自動テスト生成

- テストケース
- シミュレーション
- フェールセーフ案

7. コード生成（改修後のSTコード）

AI は次を行う：

1. 修正箇所の抽出
2. 必要最小限の改修
3. インデントやスタイルを統一
4. 新規 FB の生成
5. 既存 POU との整合性チェック

生成するコード形式は：

- .ST
- .PRG
- .FB
- .TXT

8. ダウンロード機能

- 修正済みコードを ZIP 形式で出力
- バージョン番号（v1.2.0 のように自動採番）
- 差分ファイル（diff）も含める

9. 履歴管理

- プロジェクト読み込み履歴
- 改修履歴（タイムライン）
- 変数値変化の時系列保存
- AIとの会話ログ（説明可能性の確保）

10. 推奨アーキテクチャ

- フロント：React / Electron / Tauri
- バックエンド：Python FastAPI
- LLM：OpenAI GPT-5（または同等モデル）
- DB：SQLite + Git互換履歴DB
- AST解析：独自パーサ → JSON構造へ変換

11. LLMへの指示テンプレート

モデルに反復利用するためのテンプレートも用意します。

◆ STプログラムの解析指示

以下のSTコードを解析し、構造（AST）、変数、FB呼び出し階層、「実行の流れ」「異常値の疑い」を一覧化してください。

◆ 停止原因の推定

次の入力情報（変数値CSV、トレースログCSV、エラーログCSV）をもとに、停止原因と影響範囲を特定し、改善案を示してください。

◆ 改修後のコード生成

原因分析の結果に基づいて、最小限の改修を行った新しいSTコードを生成してください。必ず動作整合性を保ち、差分も出力してください。

12. 最終アウトプット例

LLM にこのプロンプトを投げると、以下が返ってくる状態を目指す：

- アプリ全体の UI デザイン
- データ構造（JSON, AST）
- バックエンド API
- インポートロジック
- AI解析ロジック

- コード生成モジュール
- 画面ごとのコンポーネント設計
- コードサンプル
- 改修ワークフロー
- テストケース
- 将来機能案



このプロンプトの使い方

- システムプロンプトとして LLM に食わせる
- PoC（プロトタイプ）設計を生成させる
- 機能の追加や修正を随時依頼する
- 必要なコードだけを抜き出してアプリに組み込む

必要であれば：

✓ アプリのワイヤーフレーム

✓ ER図 / クラス図

✓ AST形式の定義

✓ プロトタイプコード

✓ GitHub README風の整理