# VB Decompiler

## ユーザーマニュアル

日本語

Visual Basic 5.0/6.0 · .NET

DotFix Software

# 目次

**1 概要** . . . . . **4**

**2 クイックスタート** . . . . . **6**

**3 メインウィンドウ** . . . . . **7**

**4 コードの表示** . . . . . **9**

**5 AI によるコード改善** . . . . . **12**

**6 開く・保存する** . . . . . **16**

6.1 ファイルを開く . . . . . 16

6.2 解析データベースの操作 . . . . . 18

6.3 プロシージャー覧の保存 . . . . . 19

**7 プログラムの機能** . . . . . **20**

7.1 アドレスへの移動 . . . . . 20

7.2 文字列検索 . . . . . 21

7.3 文字列参照 . . . . . 22

7.4 指定アドレスからの逆コンパイル . . . . . 24

7.5 データの変更 . . . . . 25

7.6 難読化 . . . . . 27

**8 プログラムオプション** . . . . . **28**

8.1 プログラムオプション . . . . . 28

8.2 フォーム . . . . . 29

8.3 逆コンパイル . . . . . 30

8.4 逆アセンブラ . . . . . 32

8.5 配色 . . . . . 33

8.6 高度な機能 . . . . . 35

8.7 言語の選択 . . . . . 39

**9 プラグイン . . . . . 40**

**10 フォームの視覚的な復元 . . . . . 45**

**11 トレース . . . . . 49**

**12 解析機能 . . . . . 51**

**13 難読化の制限 . . . . . 53**

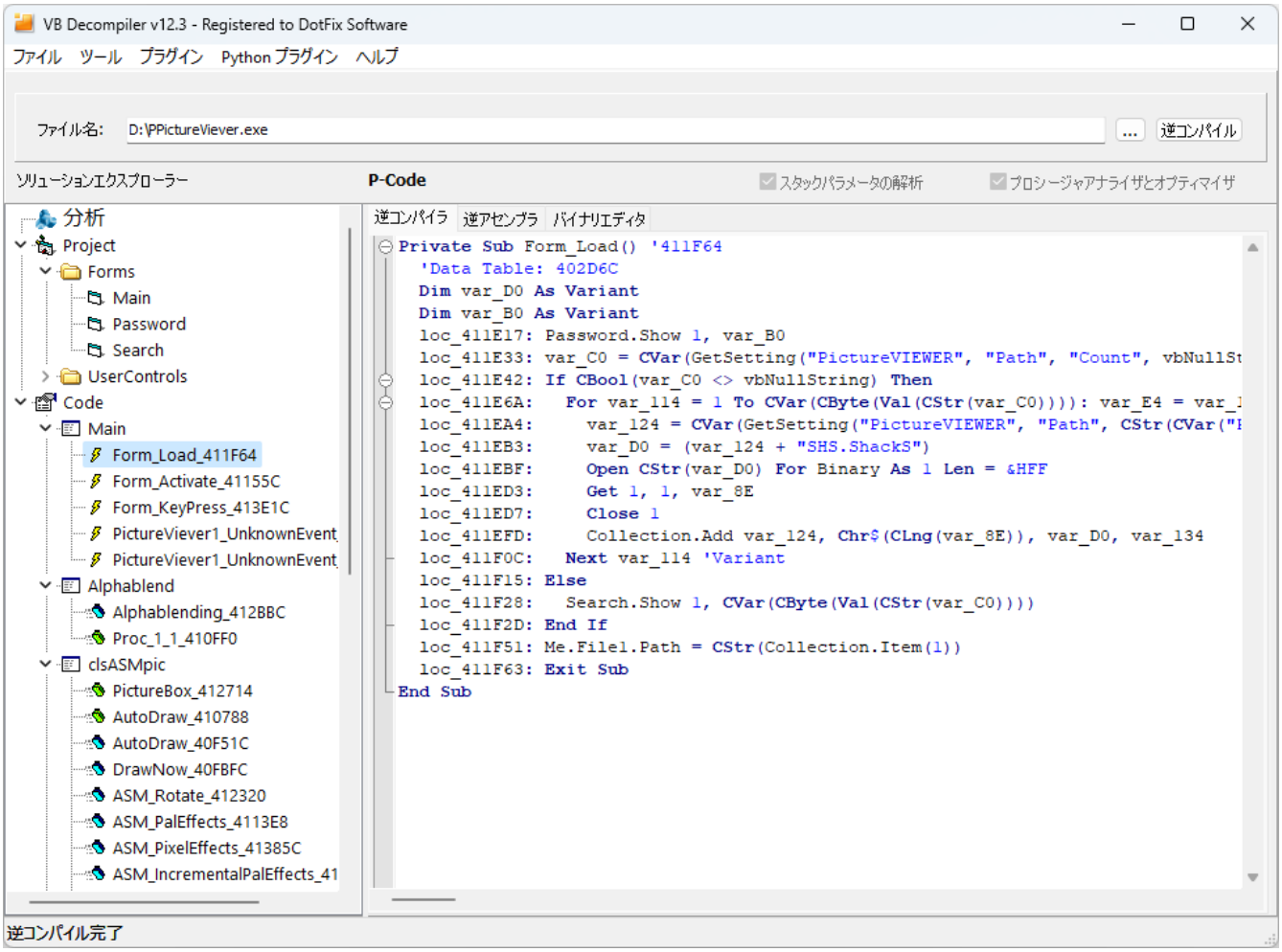
**14 テクニカルサポート . . . . . 55**

**15 登録 . . . . . 55**

**16 開発者について . . . . . 56**

# 1 概要

**VB Decompiler** は、Visual Basic 5.0/6.0 プログラムおよび .NET アセンブリ向けのリバースエンジニアリングおよびソースコード復元ツールです。高水準の逆コンパイル、低レベルの逆アセンブル、メタデータ検査、内蔵の HEX エディタ、およびビジュアルインターフェースの再構築を統合しています。主な用途には、レガシーソフトウェアの保守、インシデント対応、マルウェア分析、および未文書化されたアプリケーション動作の調査が含まれます。




## サポートされているコード表現

- **VB5/6 P-Code:** は、コンパイラの P-Code 命令とランタイム操作から Visual Basic ステートメントを再構築します。
- **VB5/6 Native Code:** は x86 命令を逆アセンブルし、内部エミュレータ、ランタイムシグネチャ、およびヒューリスティクスを使用して Visual Basic 風のコードを再構築します。
- **.NET:** は共通中間言語 (CIL、MS IL とも呼ばれます) を逆アセンブルし、サポートされている IL パターンを C# 風のコードに逆コンパイルします。元の .NET ソース言語に関係なく処理されます。.NET アセンブリ内のネイティブ x86 および x64 コードも逆アセンブル可能です。

.NET 分析エンジンでは、コード分析またはフォームの視覚的な復元に .NET ランタイムや Reflection を必要としません。

## 分析ツール

プロシージャおよびテキスト検索、相互参照と変数名の再命名、および文字列参照を使用して動作を調査します。ビジュアルビューアは、コンパイルされた BAML リソースに格納されているクラシック VB フォーム、.NET Windows Forms、および WPF ウィンドウ、ページ、ユーザーコントロールを再構築します。

トレース機能を備えたライセンスは、VB5/6 ネイティブコードと P-Code のステップバイステップエミュレーションを提供します。分析機能を備えたライセンスは、VB5/6 および .NET アプリケーションのカテゴリ化されたレポートを提供します。AI ヘルパーは、Ollama または OpenAI API 統合を使用して復元された VB5/6 または C# コードを処理できます。

## 復元されたコードの解釈

コンパイルにより、コメント、多くのローカル変数名、および一部の型およびプロシージャメタデータなどの情報が削除されます。最適化およびオブフスケーションによりコード構造がさらに変更される可能性があります。再構築されたリストはコード分析と復元をサポートします。重要な結論は、逆アセンブル結果、メタデータ、文字列参照、および関連するプロシージャのコードと照合してください。

# 2 クイックスタート

1 **ファイル** > **プログラムを開く** (**File** > **Open program**) を選択し、解析対象の EXE、DLL、または OCX ファイルを開きます。処理が自動的に開始されます。プロジェクトツリーが読み込まれ、処理が完了するまでお待ちください。

2 **ソリューションエクスプローラー** (**Solution explorer**) でプロシージャを開き、**逆コンパイラ** (**Decompiler**) と **逆アセンブラ** (**Disassembler**) のビューを比較します。**バイナリエディタ** (**HEX Editor**) を使用して対応するバイト列を確認します。

3 **Ctrl+T** を押下して、プロシージャ名またはその一部で検索します。表示されたテキストを検索するには **Ctrl+F** を使用し、文字列とそれらを使用するプロシージャを調査するには **ツール** > **文字列の参照** (**Tools** > **String references**) を使用します。

4 復元されたフォームを開き、**デザイナー (読み取り専用)** (**Designer (read only)**) で視覚的なプレビューを選択します。クラシック VB、Windows Forms、および WPF については、復元されたイベントを使用して対応するハンドラコードに移動します。

5 後で再開する必要がある場合や他の分析者と結果を共有する場合、**ファイル** > **データベース** (**File** > **Data Base**) を通じて解析結果を保存します。

## 逆コンパイルモードの選択

高速逆コンパイル は、プロシージャを開いた際に処理を実行します。**プロトタイプの分析** (**Analyze prototypes**) を有効にして、Private として宣言されたプロシージャのシグネチャに関する追加の予備解析を行います。すべての逆コンパイルされたテキストを検索するか、プロジェクト全体をエクスポートする場合は、高速逆コンパイルを無効にし、ファイルを再度処理します。

## 復元されたソースコードの保存

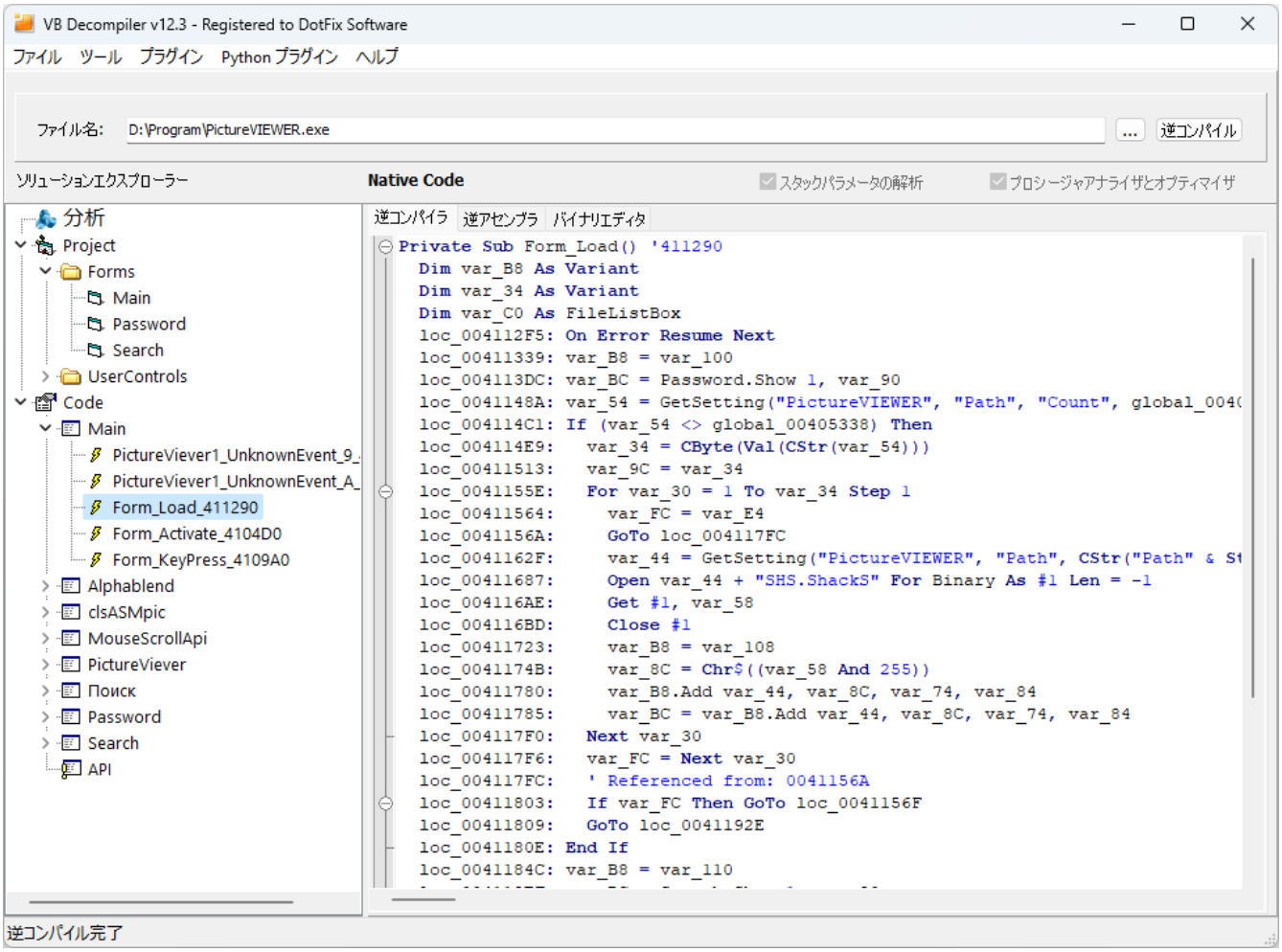
VB5/6 の逆コンパイル結果をエクスポートするには、**ファイル** > **逆コンパイルされたプロジェクトを保存** (**File** > **Save decompiled project**) を選択し、出力フォルダを選択します。エクスポートには復元されたフォームとモジュールが含まれ、FRX ファイルは関連する画像およびその他のバイナリフォームデータを保持します。

エクスポートしたコードを IDE で開く前に、復元されたソースコードに含まれる逆コンパイルの不正確な箇所と非互換性をすべて手動で修正してください。

詳細なワークフローについては、フォームの視覚的な復元、コードの表示、および AI によるコード改善 を参照してください。

# 3 メインウィンドウ

**ソリューションエクスプローラー (Solution explorer)** は、復元されたプロジェクトをフォーム、モジュール、クラス、およびプロシージャに整理します。隣接するビューは選択されたコード、フォーム、またはメタデータを表示します。**Visual Studioのようなウィンドウスタイル (Window style as in Visual Studio)** オプションはツリーを右側に、コードを左側に配置します。




## コードおよびデータビュー

- **逆コンパイラ (Decompiler):** 高水準で再構築されたコード。
- **逆アセンブラ (Disassembler):** 対応する x86/x64 命令、VB P-Code、または .NET IL、およびアドレス。
- **バイナリエディタ (HEX Editor):** 基盤となるファイルバイト、仮想アドレスまたはファイルオフセットによるナビゲーション。
- **デザイナー (読み取り専用) (Designer (read only)):** 再構築されたフォームとその復元されたプロパティ（サポートされている場合）。
- **AI 支援 (AI Helper):** 現在逆コンパイルされているプロシージャのオプション AI 処理。

利用可能なタブは選択されたオブジェクトによって異なります。コードタイプインジケーターは **P-Code**、**Native Code**、および **.NET Assembly** を区別します。

## スタックパラメータの解析 (Parse stack parameters)

P-Code の場合、このコントロールはスタック操作とコールを引数を持つ式に結合します。これをオフにすると中間操作が表示され、未知のプロトタイプが予期しないスタック値を消費する際に役立つ場合があります。Native Code の場合、これを有効にするとエミュレータを使用して高水準の式を再構築し、無効にするとアセンブリレベルの出力が表示されます。.NET の場合、スタック処理は IL を C# 風の式として再構築することに寄与します。

## プロシージャアナライザとオプティマイザ (Procedure analyzer and optimizer)

このコントロールは、式と認識された制御フローパターンを結合し、認識されたコンパイラの補助コードを除去することで、再構築されたリストを簡素化します。VB5/6 および .NET コードに適用されます。.NET 最適化器は、サポートされている `while`、`do...while`、および `foreach` パターン、ネストされた条件、演算子式、および補間文字列ハンドラーのパターンを再構築できます。これらの変換は逆コンパイラ自体によって実行され、AI 支援 を必要としません。

式または分岐が不明瞭な場合、最適化を無効にした出力と比較し、逆アセンブルを検討してください。**高速逆コンパイル (Fast decompilation)** モードでキャッシュなしの場合、これらのコントロールはプロシージャを検査する間に変更できます。完全な逆コンパイルまたはキャッシュされた出力の場合、設定を変更した後にファイルを再度処理します。

## ナビゲーションとキーボードアクセス

ツリー上の矢印ボタンは以前表示されたプロシージャをナビゲートするために使用されます。**Esc** は前のプロシージャに戻ります。マウス Back および Forward ボタンは、プロシージャ内の記憶された位置を含む同じ履歴を使用します。

**Alt+1** を押して ソリューションエクスプローラー とアクティブなビューの間でフォーカスを切り替えます。ツリーで **Enter** を押して選択されたオブジェクトを開きます。HEX エディタでは、**Tab** は十六進数フィールドとテキストフィールドの間を移動し、次に次のコントロールに移動します。インターフェースラベルはスクリーンリーダーナビゲーションのためにその入力フィールドに関連付けられています。

コードの表示 を参照して文脈依存のショートカット、相互参照、および名前変更を、分析機能 を参照して分析 (Analytics) ビューを確認してください。

# 4 コードの表示

コードビューは、シンタックスハイライト、折りたたみ可能なブロック、アドレスベースのナビゲーション、および相互参照を統合しています。プロシージャを理解するには高水準のリストを使用し、その結果がどのように得られたかを確認するには逆アセンブルとファイルバイトを使用します。

## 識別子のハイライトとナビゲーション

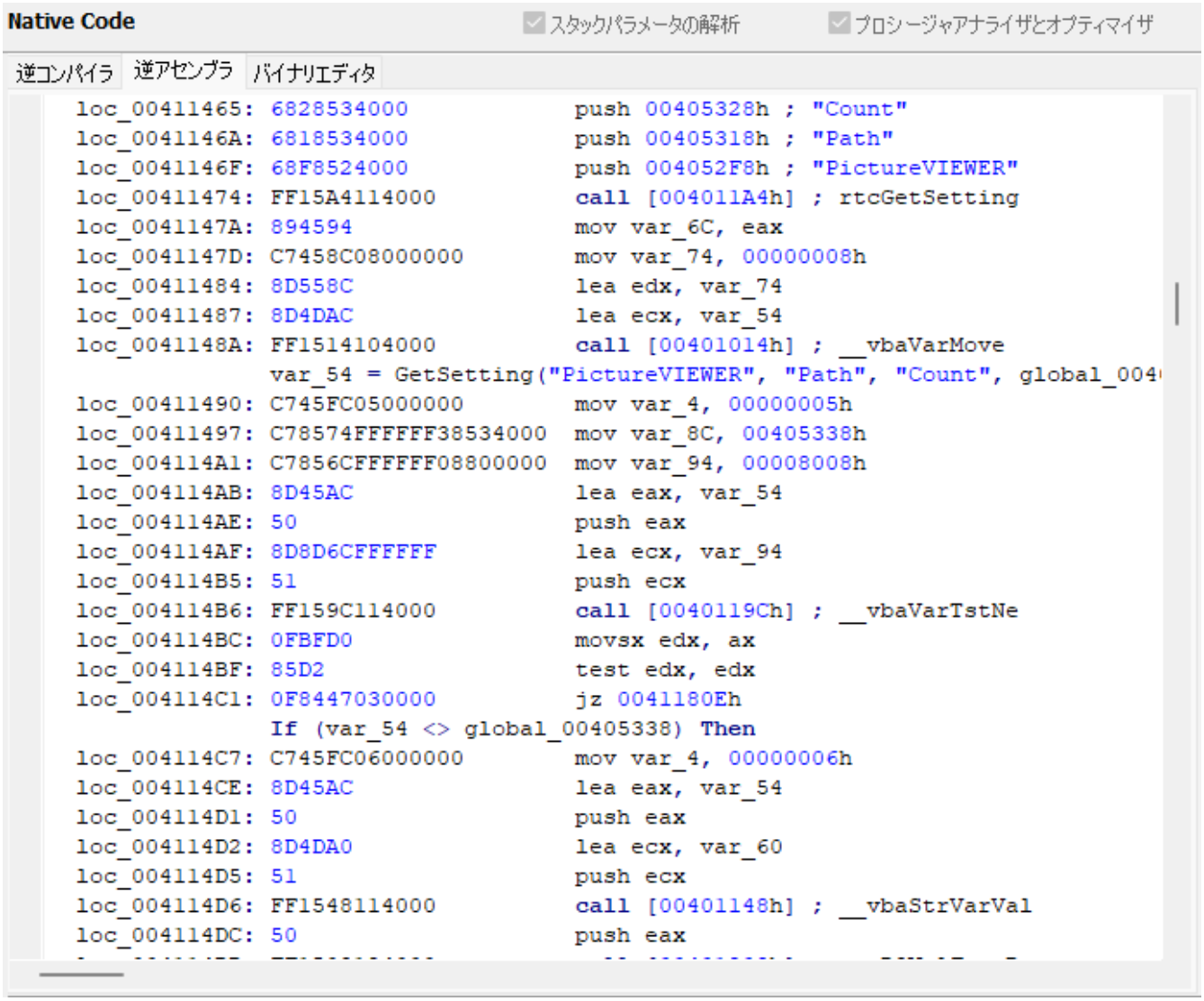
レジスタ、変数、または関数名にカーソルを置き、現在のリスト内の一致する出現箇所をハイライトします。ローカル変数である `var_XX` および `str_XX` は、メンバー表現の基底として使用される場合（例えば `str_1.Value`）もハイライトされます。`global_XX` 識別子を選択すると、その出現箇所が現在のプロシージャ内でハイライトされ、これはプロジェクト全体の参照検索とは別です。

認識されたプロシージャ呼び出しをダブルクリックして呼び出されたプロシージャを開きます。代わりにアドレスのような定数は、HEX エディタ内の対応するバイトに導くことができます。**Esc**、ナビゲーション矢印、またはマウスの Back/Forward ボタンを使用して表示履歴を移動します。

## 逆コンパイルされたコードと逆アセンブルされたコードの統合

逆コンパイルされたリストを右クリックし、**逆アセンブラにコピー (Copy to disassembler)** を選択するか、コードビューで **D** を押します。これにより、再構築されたステートメントと現在のプロシージャの低レベルリストが統合されます。自分の位置を保持するには、コマンドを呼び出す前にカーソルを関連する `loc_` アドレスに置きます。

この操作では **アドレスを追加 (Add Addresses)** を有効のままにしてください（逆コンパイル設定）。逆アセンブラ (Disassembler) は常にアドレスを表示します。アドレスを追加 は 逆コンパイラ (Decompiler) ビューのアドレス表示を制御します。




## 変数の名前変更

VB5/6 P-Code および Native Code については、変数を右クリックし、**名前変更 (Rename variable)** を選択するか、変数にカーソルを置き **N** を押します。意味のある解析名を入力します。逆コンパイラは名前衝突と予約済みキーワードをチェックします。名前変更を元に戻すには、**名前の変更を元に戻す (Undo renaming)** または **Ctrl+Shift+Z** を使用します。このコマンドは通常のテキスト編集の Undo とは独立しています。

名前変更により復元されたリストの解析が容易になりますが、コンパイルによって削除された名前は復元しません。アナリストが割り当てた名前は、発見事項を伝える際に注釈として扱ってください。

## 呼び出し元とグローバル変数の参照の検索

呼び出し元を見つけるには、ソリューションエクスプローラー (Solution explorer) 内のプロシージャを右クリックし、**参照の検索 (Find references)** を選択するか、ツリーでそのプロシージャが選択されている状態で **X** を押します。結果には、選択したプロシージャへの呼び出しが認識されたプロシージャが一覧表示されます。結果をダブルクリックして呼び出し元を検査します。このワークフローは VB5/6 および .NET をサポートしており、.NET サポートはバージョン 12.4 で追加されました。

グローバル変数を調査するには、コードビュー内のその識別子にカーソルを置き **X** を押します。したがって同じキーは、ツリー内のプロシージャ参照とコード内のグローバル変数参照を検索します。認識された参照のリストは、追加の間接的、動的解決、または難読化された使用がないことを確立するものではありません。

## 文脈依存のショートカット

| フォーカス | ショートカット | アクション |
|---|---|---|
| コード | N | カーソル下の変数の名前変更（VB5/6）。 |
| コード | X | カーソル下のグローバル変数の参照を検索。 |
| コード | D | 現在の逆コンパイルされたコードを逆アセンブラにコピー。 |
| コード | F4 | 現在の行のアドレスにトレース；トレース機能が必要です。 |
| コード | Ctrl+Shift+Z | カーソル下の変数の名前変更を元に戻す。 |
| ツリー | X | 選択されたプロシージャの参照を検索。 |
| ツリー | G | セクションへのジャンプを開く。 |
| ツリー | Ctrl+C | 選択されたオブジェクトの完全なツリーパスをコピー。 |
| ツリー | Space or Insert | 選択された項目をマークまたはアンマーク。 |

単一文字のショートカットはフォーカスされたビューに依存し、入力フィールドでの通常のタイピングを置き換えません。フォーカスを切り替えるには **Alt+1** を使用します。Ctrl+T および Ctrl+F については 文字列検索、ツリーパスとアドレスについては アドレスへのジャンプ、エミュレータの制限については トレース を参照してください。

# 5 AI によるコード改善

**AI 支援** (**AI Helper**) は、VB Decompiler が既に再構築したコードを処理します。.NET アセンブリからの逆コンパイルされた C# および P-Code または Native Code からの Visual Basic 5.0/6.0 コードをサポートします。Ollama を通じてローカルモデルを使用するか、OpenAI API インテグレーションを通じてサービスに接続できます。

AI は、より明確な制御フロー、変数名、宣言、および式を提案できます。しかし、バイナリから欠落している情報を確実に復元したり、逆コンパイラが誤って解釈した断片を修復したりすることはできません。復元されたソースコードや調査報告書で使用する前に、元の逆コンパイルされたリスト、逆アセンブル、および周辺のプロシージャと比較してください。

## プロシージャの処理

1. ソリューションエクスプローラー (Solution explorer) で必要なプロシージャを開き、その **逆コンパイラ** (**Decompiler**) リストを確認します。
2. **AI 支援** に切り替え、以下に説明されているようにプロバイダー、ホスト、およびモデルを設定します。
3. モデルセレクターの横にあるアクションボタンで処理を開始します。
4. AI 支援 で返されたコードを確認し、逆コンパイラの出力と比較します。

AI 処理は別のスレッドで実行されます。モデルの応答に時間がかかりすぎる場合は **キャンセル** (**Cancel**) を使用してください。追加の接続設定を折りたたむことで、コード表示用のスペースを増やすことができます。

## Ollama を通じてローカルモデルを使用する

1. 公式ダウンロードページ から Ollama をインストールして起動します。
2. ターミナルで `ollama pull qwen3:8b` を実行して推奨モデルをダウンロードします。
3. AI 支援 でローカル LLM/Ollama プロバイダーを選択し、**Ollama host** にサーバーアドレスを設定します。デフォルトのローカルアドレスは `http://localhost:11434` です。
4. **モデル** (**Model**) でインストールしたモデルを選択して処理を開始します。

`qwen3:8b` を推奨します。AI 支援 の組み込みプロンプトは、この軽量モデルに合わせて調整されています。より新しいモデルでは、処理結果が改善されないままコードの処理時間が長くなる場合があります。

モデルの選択に制限はありません。選択した Ollama インスタンスにインストールされている任意のモデルを使用できます。未知の応答形式では、応答パーサーがモデル固有のタグやその他の補助テキストを除去できず、表示結果に残す場合があります。

### Ollama をローカルで実行するためのハードウェア

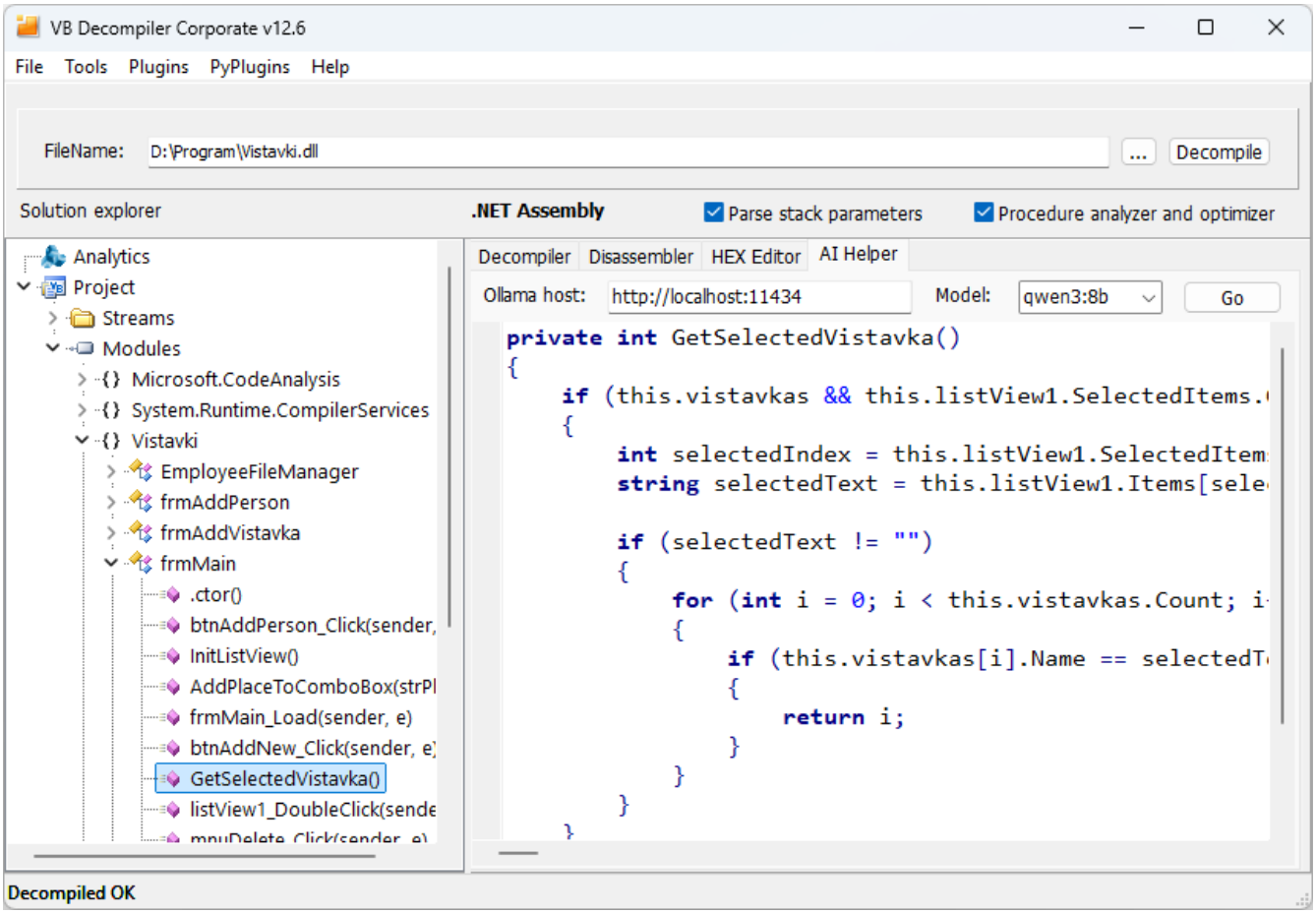
ローカル AI 処理は Ollama を実行しているコンピューター上で行われ、逆コンパイルよりも多くのリソースを必要とします。以下は、`Q4_K_M` 量子化の重みと標準の `f16` コンテキストキャッシュを使用する `qwen3:8b` で、同時に 1 件のリクエストを処理する場合の実用的な構成の目安です。コンテキストはトークン単位で、4K は 4096 トークン、128K は 131072 トークンです。この容量には指示、ソースコード、生成される回答が含まれます。

| 項目 | 実用上の最小構成：4K | 推奨構成：128K |
|---|---|---|
| OS | 64 ビット版 Windows 10（22H2）または Windows 11 | 64 ビット版 Windows 10（22H2）または Windows 11 |
| CPU | AVX2 対応の現行 x64 CPU、物理コア 4 個以上 | AVX2 対応の現行 x64 CPU、物理コア 8 個以上 |
| システムメモリ（RAM） | 16 GB | 64 GB |
| グラフィックスカード | Ollama 対応 GPU、専用 VRAM 8 GB 以上 | Ollama 対応 GPU、専用 VRAM 32 GB 以上 |
| ストレージ | Ollama とこのモデル用に 20 GB 以上の空き容量がある SSD | Ollama とこのモデル用に 20 GB 以上の空き容量がある SSD |

Ollama 設定のスライダーでコンテキスト長を選択します。4K に適した 8 GB のカードでは、上記の 128K 構成には不足します。モデルのダウンロードサイズは約 5.2 GB ですが、必要なメモリの総量はこれより大きくなります。128K では `f16` コンテキストキャッシュだけで約 18 GiB を使用し、さらにモデルの重みと作業用バッファーが必要です。Windows や他のアプリケーション用のメモリも確保してください。

十分な RAM があれば Ollama は CPU でも処理できますが、コードの処理が非常に遅くなる場合があります。Pentium M 搭載機や同程度の古いコンピューターは、`qwen3:8b` のローカル実行には適しません。旧世代の Intel HD Graphics の共有メモリは、上記の専用 VRAM の代わりにはなりません。これらの要件は Ollama を実行するコンピューターに適用されます。リモートサーバーや外部 API サービスでは、そのサービス側のハードウェアを使用します。

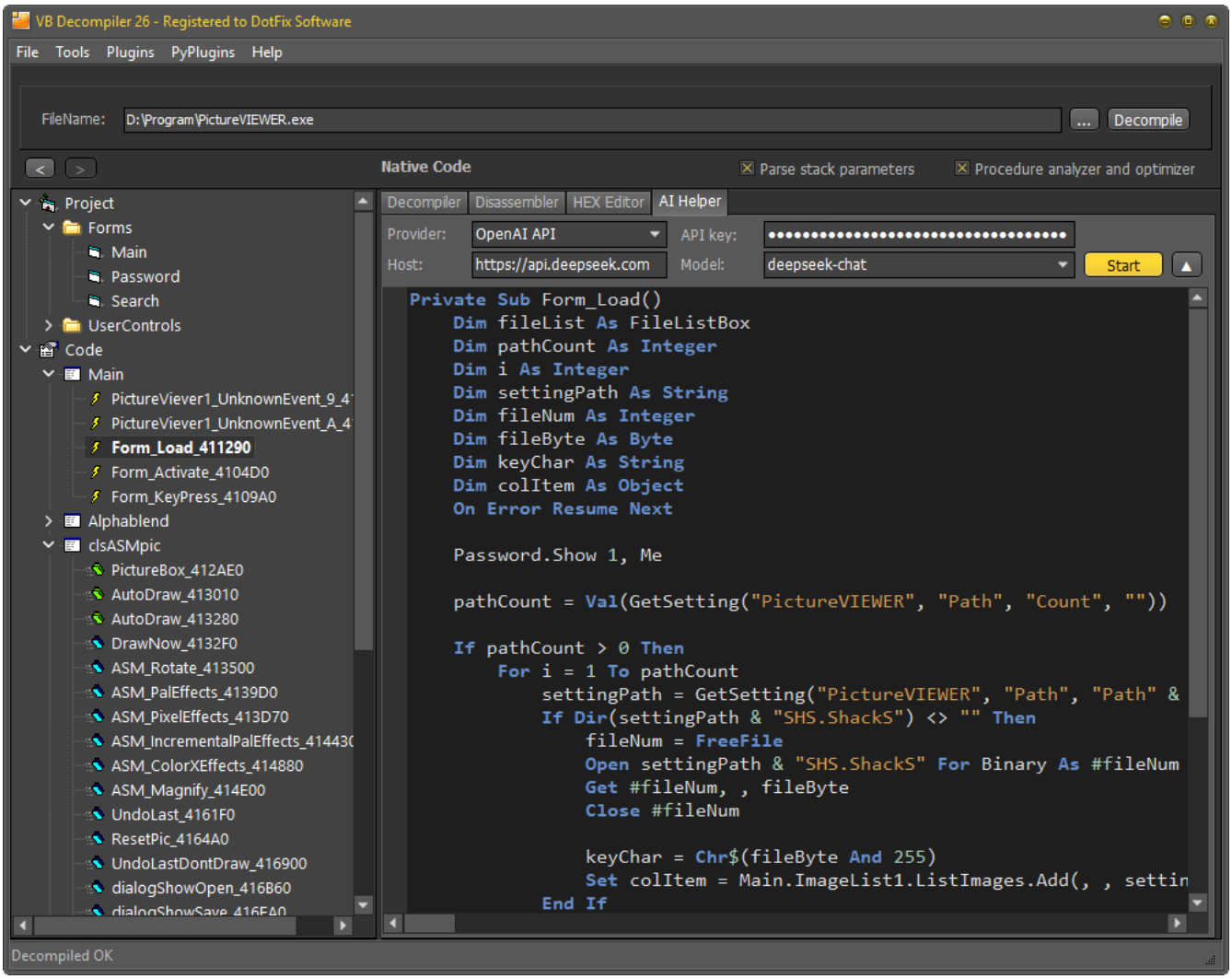
短いプロシージャでは 4K から始めてください。コンテキストを大きくすると、メモリ使用量と長い入力の処理時間が増加します。処理が遅すぎる場合は中止し、コンテキストを小さくして、`ollama ps` で GPU の使用状況を確認してください。実際のメモリ使用量は Ollama のバージョン、バックエンド、他の実行中の処理によって異なります。




## OpenAI API インテグレーションを使用する

**提供元** (**Provider**) を **OpenAI API** に設定します。**ホスト** (**Host**)、**APIキー** (**API key**)、および **モデル** を意図したサービスに合わせて設定します。有効な API キーと、そのキーで利用可能なモデル識別子を入力します。API キーをスクリーンショットや共有された分析資料から除外してください。

AI 支援 は **OpenAI API** プロバイダーを通じて DeepSeek もサポートしています。対応するフィールドに DeepSeek のホスト、API キー、およびモデル識別子を記入します。以下のスクリーンショットがこの設定を示しています。




## コードが処理される場所

AI 支援 は、現在の逆コンパイルされたコードを構成されたサービスに送信します。ローカル専用ワークフローの場合、ローカルで実行されている Ollama サーバーとローカルで実行されたモデルを使用します。別のマシン上の Ollama ホスト、Ollama クラウドモデル、または外部 API サービスは、コードを分析ワークステーションの外へ転送できます。

機密ソースの復元やセキュリティ調査の場合、組織のデータ処理要件に応じてエンドポイントを選択してください。Ollama や localhost URL の存在だけでオフライン動作を推測しないでください。

## トラブルシューティング

- ローカルモデルが欠落している場合は、ホストフィールドで指定された Ollama インスタンスにインストールされていることを確認します。
- 応答がない場合は、選択したサービスが実行中であり到達可能であることを確認し、エンドポイント、認証情報、およびモデル識別子が同じプロバイダーに対応していることを確認します。
- 処理が遅すぎる場合はキャンセルします。別のローカルモデルを使用する場合、推奨される `qwen3:8b` モデルを試してください。
- 未知のモデルが応答にタグやその他の補助テキストを残す場合は、`qwen3:8b` を試すか、結果を確認しながらそのテキストを手動で削除してください。

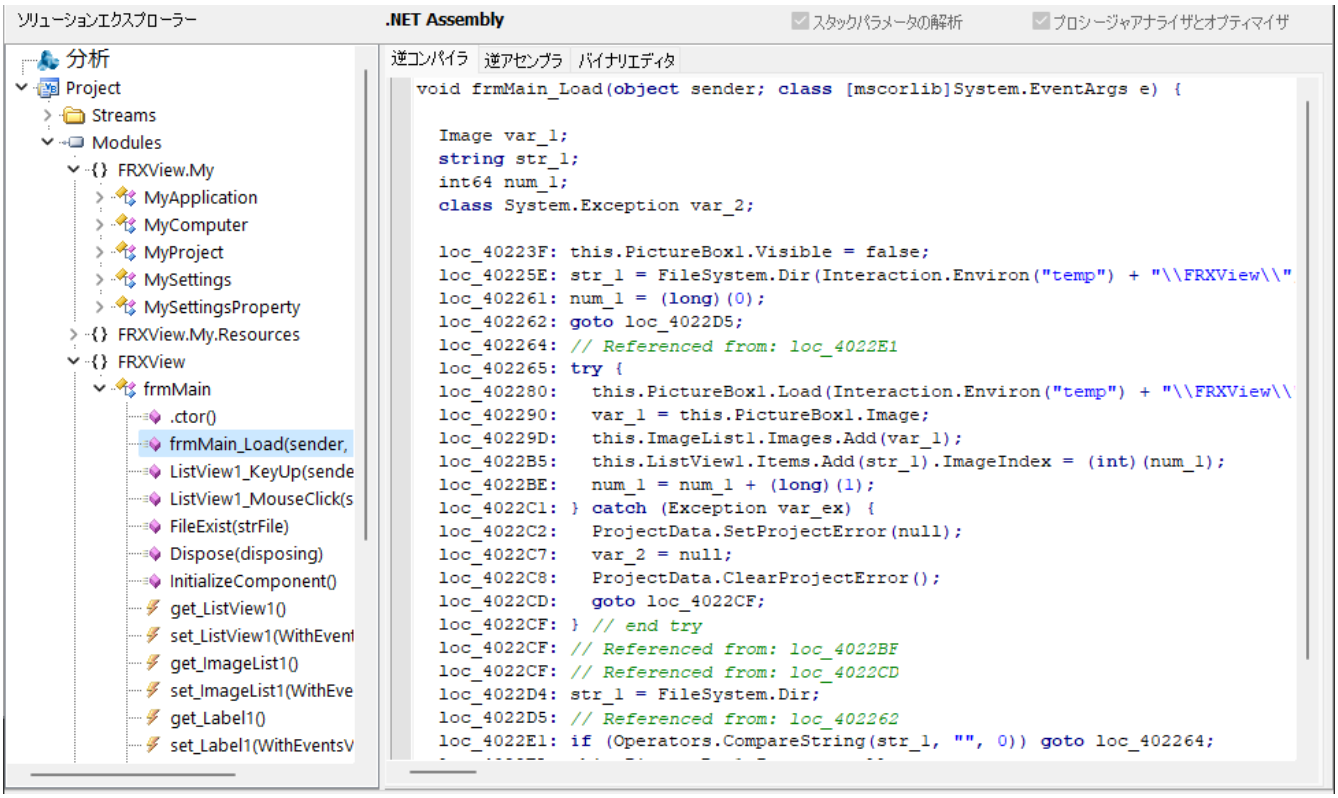
- 結果が条件、呼び出し、定数、または型を予期せず変更した場合は、低レベルリストに対して確認してください。読みやすさだけでは、元のコードと同等の動作であることは確認できません。

Ollama および外部モデルプロバイダーはサードパーティ製製品です。その可用性、モデルの動作、およびサービスポリシーは VB Decompiler に依存しません。

# 6 開く・保存する

## 6.1 ファイルを開く

逆コンパイル処理を開始する前に、解析対象となるファイルを選択する必要があります。この目的のために、**ファイル** (File) メニュー内の **プログラムを開く** (Open program) セクションがあります。**VB Decompiler** は、**Visual Basic** バージョン 5.0 または 6.0 で作成された EXE/OCX/DLL ファイルおよび **Visual Studio .NET** でコンパイルされたアプリケーションの読み込みをサポートします。後者の場合、ソース言語は重要ではなく、Visual Studio .NET は元の言語に関係なく、すべてのアプリケーションを単一の内部形式 - **MS IL** にコンパイルするためです。




ファイルを開いた後、逆コンパイラはプログラムが指定された開発環境に属するかどうかを確認します。開かれたプログラムがサポートされていない環境（例：**Delphi** または **Visual C++**）で開発された場合、そのような環境はサポートされておらず、プログラムを逆コンパイルできない旨のメッセージが表示されます。このメッセージが表示された場合は、当社にご連絡ください。私たちは **Delphi** および **FreePascal** アプリケーションの逆コンパイル分野で内部開発を行っており、そのようなファイルに対する逆コンパイルサービスを提供できます。

開かれたプログラムが **EXE パッカー**、オブフスケーターまたは保護によってパックされている場合、逆コンパイラはこのファイルをアンパックしようとするかどうかを提案します。この機能は 32 ビットの Windows バージョンでのみ利用可能であり、いかなる保証も伴いません。アンパックが可能であれば、ファイルは起動され、メモリ内でアンパックされ、別の名前でディスクに保存され、逆コンパイルのために開かれます。**注意！** 組み込みのアンパッカーを使用して悪意のあるプログラムやシステムを損傷する可能性がある他のコー

ドをアンパックしようとしてはいけません。アンパック時に開かれているプログラムが実行されるため、その実行による PC への影響は極めて深刻な結果をもたらす可能性があります。

**重要な注意事項：** 必要な権限のないソフトウェアを逆コンパイルまたはアンパックしようとしてはいけません。また、著作権者がライセンスによってこのファイルの圧縮解除および解析を禁止している場合も同様です。これはあなたの国の法律に違反する可能性があります。

*Visual Studio .NET、Visual C++ および Visual Basic は Microsoft Corporation の商標です。Delphi は Embarcadero の商標です。

## その他のファイルの開き方

Explorer からファイルを VB Decompiler にドラッグ＆ドロップできます。不明な拡張子または拡張子のないファイルもこの方法で提出できます。認識は引き続きファイルの実際のコンテンツに依存します。サポートされている Windows **.lnk** ショートカットは、参照されるファイルに解決されます。保存された VDD データベース もドラッグ＆ドロップまたはコマンドラインから開くことができます。

## 6.2 解析データベースの操作

この機能は、**データベース** (**Data Base**) セクションを通じて**ファイル** (**File**) メニューで利用可能であり、逆コンパイル結果を別の圧縮ファイルに保存することを可能にする。これにより、コードを再逆コンパイルする必要なく、将来の逆コンパイル結果の読み込みを高速化できる。

プログラム分析に戻りたい場合や、同僚とこれらの結果を共有する必要がある場合は、再度逆コンパイルを行う必要はない。完全な逆コンパイル（**高速逆コンパイル** (**Fast decompilation**) オプションが無効の場合）では、このプロセスは時間がかかり、より多くのリソースを必要とする可能性がある。**Database** に逆コンパイルされたコードを事前に保存しておく方がはるかに便利である。

**VB Decompiler** は内部圧縮データベース形式を使用しており、そのようなデータベースに含まれる情報の保存に必要な容量の削減を可能にする。さらに、VB Decompiler の以前のバージョンで保存されたデータベースとの互換性があり、製品の新バージョンでそのようなデータベースを開くことができないというリスクなしに情報を保存できる。

### VDD データベースを開く・移動する

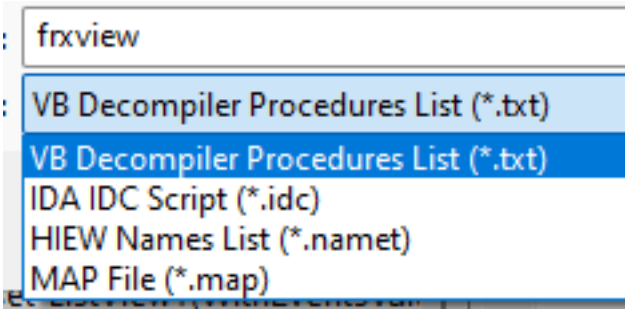
保存された分析データベースは**.vdd** 拡張子を使用する。VDD データベースを開くには、通常のファイルを開く操作を通じて、逆コンパイラウィンドウにドラッグするか、コマンドラインでそのパスを渡すことができる。

元の分析されたバイナリを検出する際、VB Decompiler は保存されたパスとデータベースを含むディレクトリの両方をチェックする。コラボレーションのため、元のパスが利用できない場合は、対応するバイナリを VDD ファイルの隣に配置しておく。アドレスとファイルバイトの意味を保つために、正確な分析済みビルドを保持する。

## 6.3 プロシージャー覧の保存

この操作は、**プロシージャリストを保存** (**Save procedures list**) オプションを **ファイル** (**File**) メニューで呼び出すことで実行され、外部ユーティリティが **VB Decompiler** を支援することを可能にするため、プログラムの解析に強力なツールです。

現在、処理されたプログラムで使用されるアドレス、関数名、およびプロシージャをいくつかの形式で保存できます。標準的な保存ダイアログでファイルタイプを指定することで、これらのオプションのいずれかを選択できます。




**TXT 形式での保存** を使用すると、プロシージャのリストを単純なテキストファイルに保存できます。これは、プログラムの異なるバージョン間のプロシージャを比較したり、アドレスの変化を確認したりする際に役立ちます。

**IDC スクリプト形式での保存** を使用すると、関数およびプロシージャのアドレスを **IDA Pro** 逆アセンブラにロードできます。分析されたプログラムが **Native Code** にコンパイルされている場合、この機能は非常に役立ちます。

**NAMET 形式での保存** を使用すると、**HIEW** ユーティリティを使用した解析を簡素化できます。取得したファイルを分析されたプログラムがあるフォルダにコピーします。次に、プログラムを HIEW で開き、F12 キーを押します。空のプロシージャのリストを持つウィンドウが表示されます。作成された NAMET ファイルのアドレスが一致している場合はもう一度 F12 キーを押して Enter キーを押します。プロシージャのアドレスが HIEW にロードされ、F12 キーを押してリストを使用してメソッドをナビゲートできるようになります。

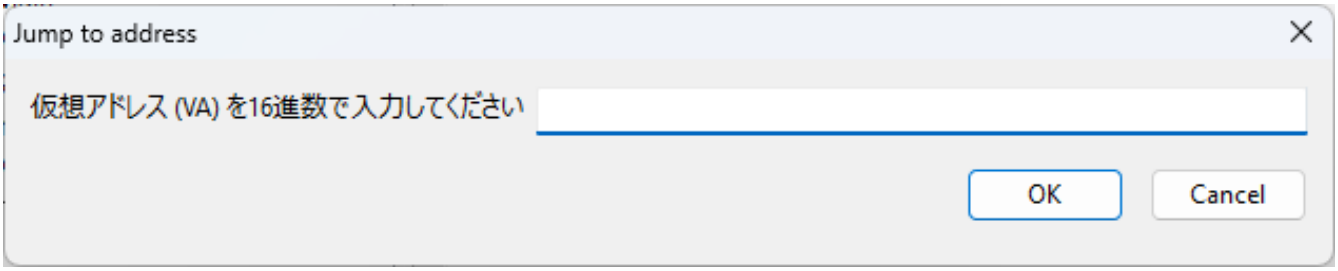
**MAP 形式での保存** は、通常コンパイラによって作成される標準的なマップファイルを生成します。このファイルは、多くのデバッガーおよび逆アセンブラで使用してプログラム関数の開始点をマークするために使用できます。Visual Basic で書かれたプログラムは、サードパーティのソフトウェアによって誤って逆アセンブルされることが多いため、**VB Decompiler** で生成された MAP ファイルはプロセスを大幅に簡素化します。この形式は、**OllyDbg**、**IDA Pro** などのさまざまな製品でプラグインを通じてサポートされています。

# 7 プログラムの機能

## 7.1 アドレスへの移動

### 仮想アドレスへのジャンプ

**ツール** > **仮想アドレスにジャンプ** (**Tools** > **Jump to virtual address**) を選択するか、**Ctrl+G** を押します。検査するコードの仮想アドレスを入力します。リストに存在するアドレスを選択し、Ctrl+G を呼び出すこともできます。逆コンパイラは、そのアドレスに関連付けられたコードを開きます。




仮想アドレス（VA）は、実行可能イメージのアドレス空間内の位置を特定し、そのイメージベースを含みます。ファイルオフセットは、ディスク上のファイル内のバイト位置を特定します。一方を他方と置き換えないでください：HEX エディタ は、両方の形式のナビゲーションをサポートしています。

### ナビゲーション履歴を通じて戻る

ツリーの上にある矢印ボタンを使用して、以前表示されたプロシージャを前後に移動します。**Esc** は前のプロシージャに戻ります。マウス Back および Forward ボタンもこの履歴を使用し、プロシージャ内の位置は保持されます。

### ツリーパスによって場所を共有する

Solution explorerで、必要なオブジェクトを右クリックして **セクション名をコピー** (**Copy the name of section**) を選択するか、ツリーがフォーカスされている状態で **Ctrl+C** を押します。これにより、選択されたノードの完全なパスがコピーされます。

その場所を再訪問するには、ツリーのコンテキストメニューで **セクションへジャンプ** (**Jump to the section**) を選択するか、ツリーで **G** を押して、コピーしたパスをダイアログに貼り付けます。調査を別の分析者に引き継ぐ場合や大規模なプロジェクトに戻るときに便利です。保存したパスは、解析時と同じファイルで使用してください。別のビルドからのパスは、同じコードを特定しなくなる可能性があります。

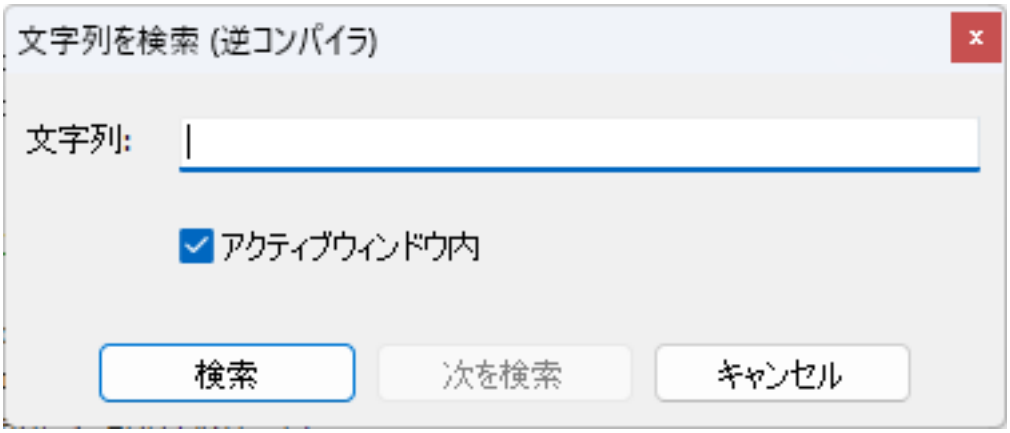
## 7.2 文字列検索

### 名前によるプロシージャの検索

**ツール** > **検索** (**Tools** > **Find procedure**) を選択するか、**Ctrl**+**T** を押します。完全なプロシージャ名またはその特徴的な部分を入力します。検索はプロジェクトツリー全体を対象とするため、どのモジュールまたはクラスにプロシージャが含まれているかを知る必要はありません。複数の名前が一致する場合、結果リストにはすべての一致するエントリが含まれます。必要なエントリをダブルクリックして開きます。

### 表示されたテキストまたはプロジェクト全体の検索

**ツール** > **Find string** (**Tools** > **Find string**) を選択するか、**Ctrl**+**F** を押して文字列検索ダイアログを開きます。**アクティブウィンドウ内** (**In active window**) が選択されている場合、検索は現在表示されているコードまたはフォームデータに適用されます。このチェックボックスをクリアすると、逆コンパイルされたプロジェクト全体を検索します。




**文字列** (**String**) にテキストを入力し、**検索** (**Find**) を選択します。**次を検索** (**Find next**) を使用して同じテキストの検索を続けます。**キャンセル** (**Cancel**) でダイアログを閉じます。

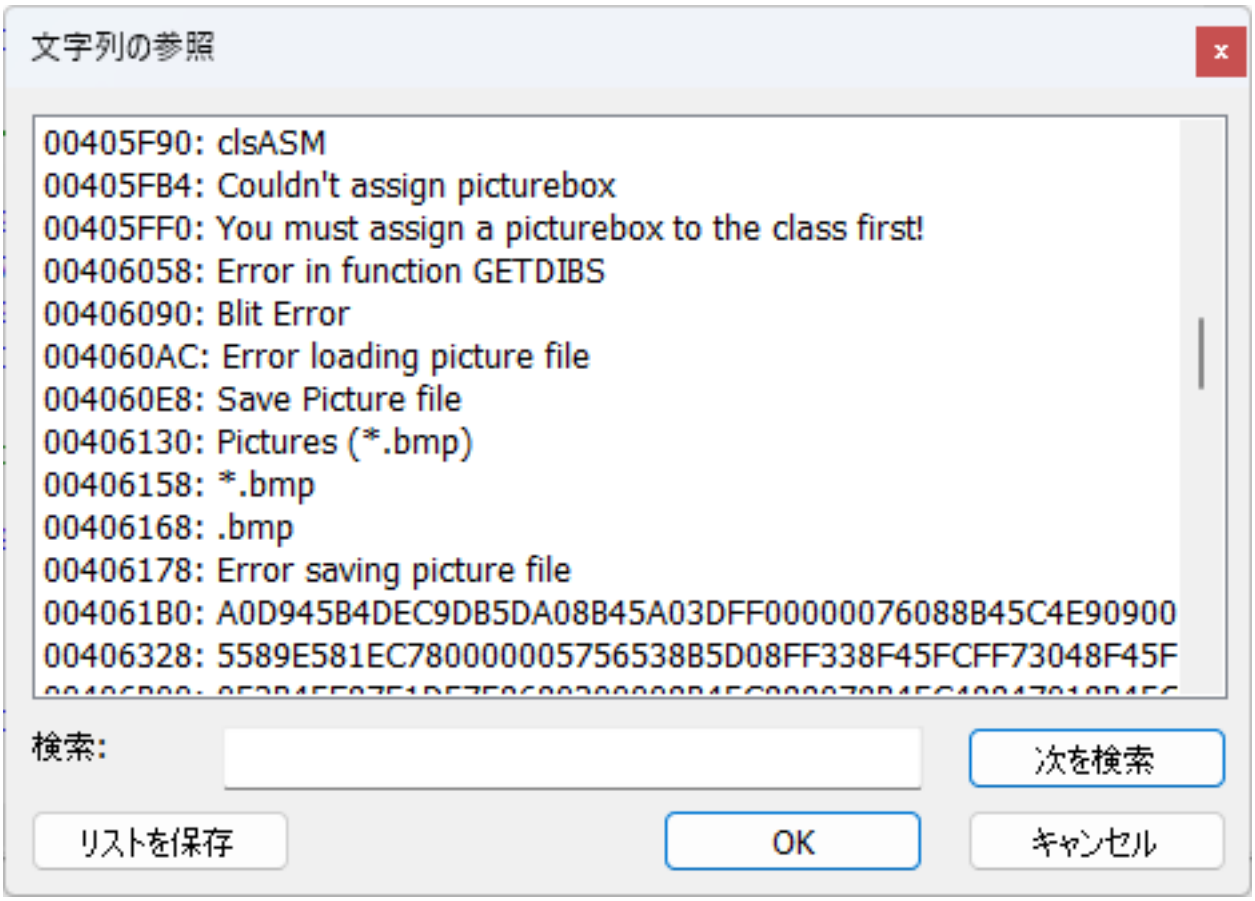
すべての逆コンパイル済みテキストを検索するには完全な逆コンパイルが必要です。**高速逆コンパイル** (**Fast decompilation**) を無効にし（高度な機能）、先にファイルを再処理してください。プロシージャ名の検索は、プロジェクト構造に対して行う別の操作です。

### 文字列参照の検索

リテラル文字列とそれらを使用するコードが必要な場合は、文字列参照 を開きます。**検索** (**Search**) フィールドと **次を検索** ボタンで文字列リストを検索します。文字列参照は、高速逆コンパイルモードでの事前の逆アセンブルによって収集でき、すべての完全に逆コンパイルされたプロシージャを検索する必要はありません。

## 7.3 文字列参照

プログラムのコード内では、頻繁に文字列データ（レジストリキー、ファイル名、エラーメッセージなど）に出会うことがあります。逆コンパイルされたアプリケーションで使用されるプロシージャや関数の文字列リストを表示することは非常に有用です。それがまさに私たちの**文字列の参照** (**String references**) 機能の目的であり、**ツール** (**Tools**) メニューを通じてアクセスできます。これを使用するには、**ツール** -> **文字列の参照** に移動するだけで十分です。プログラムに文字列データが含まれている場合、それらは以下のように表示されます：

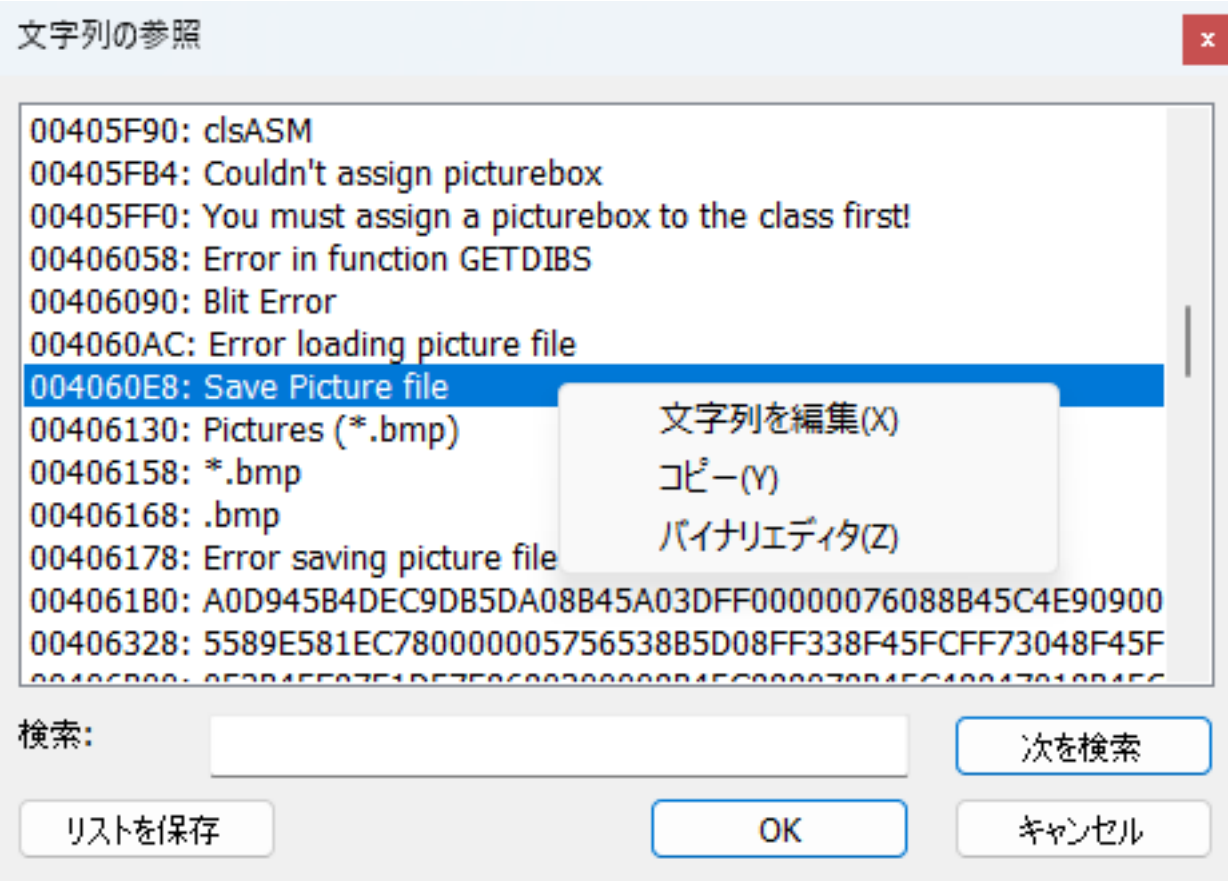



リスト内の目的の行を見つけ、ダブルクリックします。その行を含むコードセグメントが特定され、対応するプロシージャがコードウィンドウで開かれます。行を再度ダブルクリックすると、プログラムコード内でのこの行の検索が継続されます。「OK」をクリックすると、ハイライト表示された行がコード内で見つかり、行選択ダイアログが閉じます。キャンセルは単にダイアログボックスを閉じます。

**文字列の参照** を表示する機能は、ソフトウェアのライセンス版でのみ利用可能であり、**Lite** 版ではアクセスできません。

重要な注意点：バージョン 1.x でのアクティブモジュールのみの表示と異なり、現在はプログラム全体の文字列を表示します。この変更は、多数のフォームとモジュールを持つプログラムの分析を容易にするために導入されました。

**VB Decompiler** は文字列の編集もサポートしています。変更したい文字列を右クリックし、出現するコンテキストメニューから**文字列を編集** (**Edit string**) を選択して文字列編集ダイアログを呼び出します。新しい文字列を入力するためのダイアログボックスが表示されます。文字列を変更した後、**Ok** をクリックすると、文字列は編集されたファイルに直接修正されます。この操作ではバックアップが作成されないため、かつこの操作を元に戻せないため、編集前に解析対象ファイルのバックアップを作成することをお勧めします。




重要な注意点：文字列を編集する際、新しい文字列の長さが古い文字列の長さを超えないようにしてください。これは、より長い文字列を挿入するためのスペースが予約されていないバイナリアプリケーションファイルの形式に関連しています。

HEX エディタでも文字列を表示し、部分的に編集することもできます。これを行うには、文字列を右クリックし、出現するメニューからバイナリエディタ (HEX Editor)を選択します。HEX エディタには、変更の元に戻すとやり直す機能があり、アプリケーションの文字列データを修正する際の実験を簡素化します。

文字列リストを持つウィンドウには、**検索 (Search)** と呼ばれるテキストフィールドと**次を検索 (Find next)** ボタンもあります。この機能は、特に手動での視覚的な検索が非常に困難な大規模なプロジェクトでリスト内の文字列を検索する際に有用です。

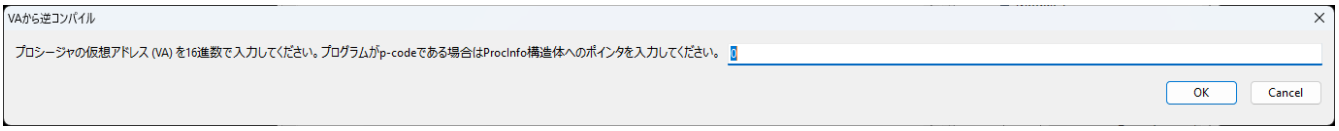
## 高速逆コンパイルと大規模な文字列リスト

文字列参照は、事前の逆アセンブルを通じて収集でき、**高速逆コンパイル (Fast decompilation)** モードを含むことができます。これは、完全に逆コンパイルされたテキストを検索することとは異なります。バージョン 26.2 は、32,000 参照を超えてダイアログを拡張し、大規模なリストのメモリ処理を改善します。

.NET アセンブリの場合、#US ヒープからのユーザー文字列は、そのファイル場所とともに検査できます。リストは静的解析に利用可能な文字列を表します；暗号化または動的に生成された文字列は別途調査を必要とする場合があります。

## 7.4 指定アドレスからの逆コンパイル

**ツール** (**Tools**) メニューには、非常に有用な機能である **VAから逆コンパイル** (**Decompile from VA**)（指定された仮想アドレスから任意のコードを逆コンパイル）があります。この機能の主な特徴は、逆コンパイルされるファイルが正しい構造を持っており、標準的な方法で処理されている必要がないことです。もちろん、この場合、コードにはいくつかの制限がかかります。例えば、インポートされた関数や OLE オブジェクトが事前に処理されていない場合、その部分のコードから得られる情報が限られます。




ファイル内の特定のコード部分を逆コンパイルするには、そのコードの仮想アドレスを指定するだけで済みます（**Native Code** または **.NET アセンブリ** の場合）。または、**ProcInfo** 構造体参照を提供します（クラシックな **Visual Basic P-Code** の場合）。このような **P-Code** で作業する独自性は、**ProcInfo** 構造体がコードで使用されるデータ配列を含むことです。この配列がないと、コードがどの関数と連携しているか、参照している文字列は何か、などを決定することは不可能です。**.NET アセンブリ** で作業する場合、アドレスを入力した後、逆コンパイラは **MS IL** 中間コードとしてコードを逆コンパイルするか、このコードを逆アセンブルするかの選択を提示します。このオプションは、.NET アセンブリには MS IL 中間コード（**マネージドコード**）とマシンコード（いわゆる **アンマネージドコード**）の両方が含まれるためです。

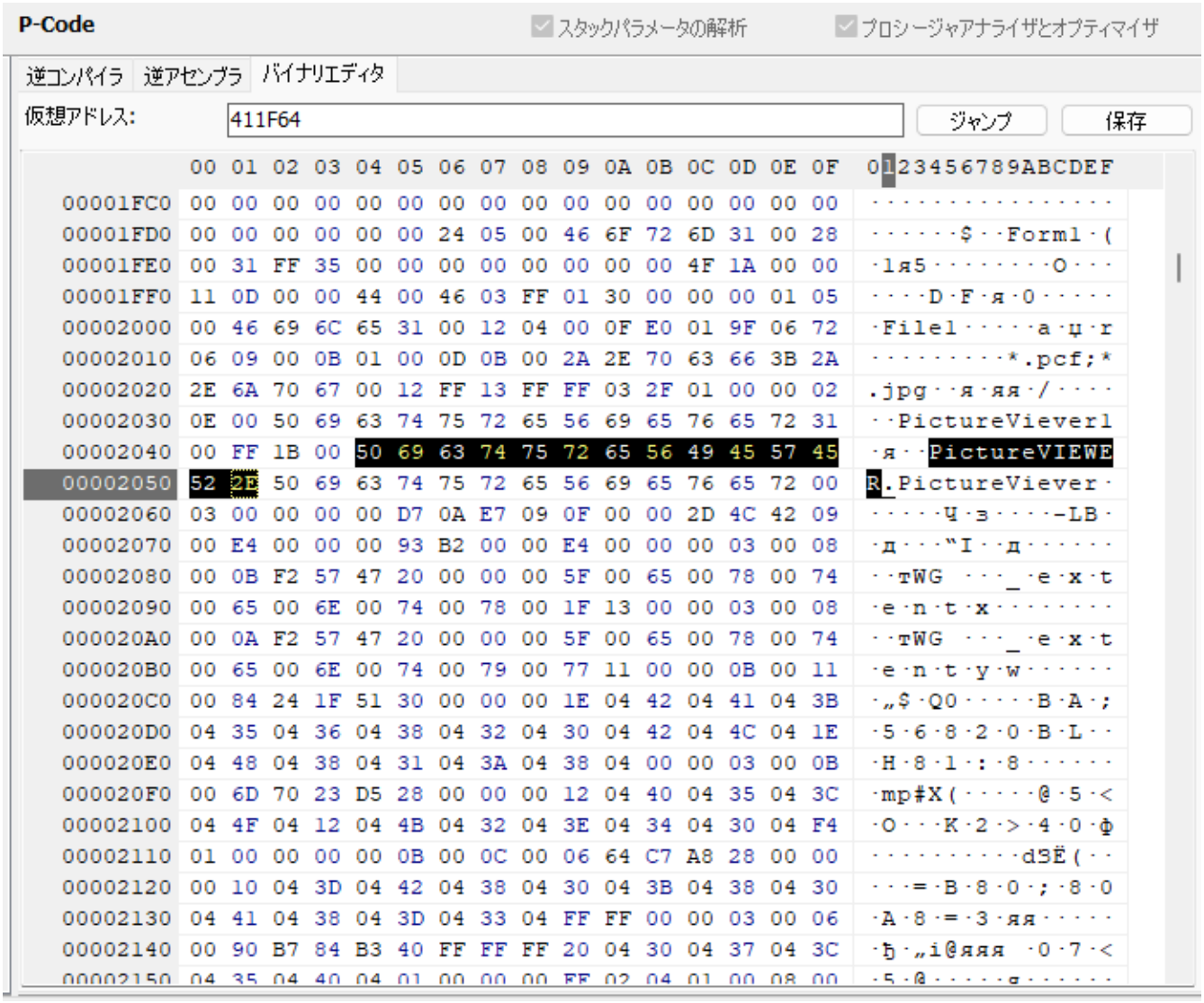
ファイルがすでに逆コンパイラで開かれている場合、**VAから逆コンパイル** 機能を呼び出して処理用のアドレスを指定するだけで済みます。ファイルが開かれていない場合は、自動的に提案され、簡易モードでファイルが開かれます。

**VAから逆コンパイル** 関数は、必要なコードが自動的に逆コンパイルされない場合や、ファイルの一部が破損して通常の方法では解析できない場合に特に有用です。もちろん、この場合の逆コンパイルの結果は、標準的な逆コンパイルよりも劣っていることを理解する必要があります。

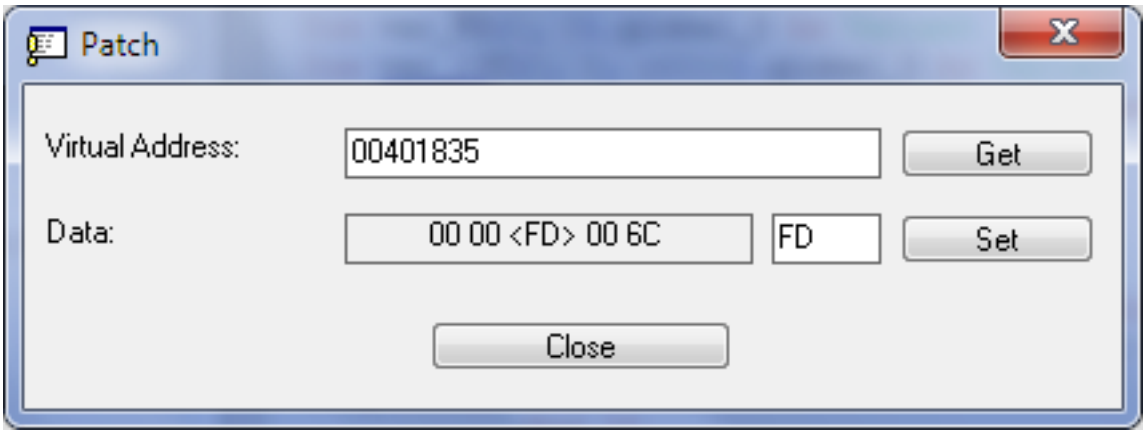
## 7.5 データの変更

バージョン 9.4 以降、**VB Decompiler** は、ファイル内のバイナリデータを編集するための現代的で便利なツール - 完全な**バイナリエディタ (HEX Editor)** を搭載しています。これは、各プロシージャまたはフォームの**バイナリエディタ** タブで逆コンパイルされたコードと並んでメインプログラムウィンドウからアクセス可能です。

**バイナリエディタ** を使用すると、ファイルのバイト単位およびブロック単位でデータを編集できます。検査中のファイル内の目的のアドレスに移動するには、仮想アドレス（**Image Base** を含む）またはオフセット（ファイルの先頭に対する必要なデータのオフセットを示す通常のアドレス）を使用できます。




逆コンパイラの以前のバージョン（バージョン 9.4 以前）では、ファイル編集機能はより原始的なツールである Data Patcher を使用して適用されました。その動作は以下の通りです。**ツール (Tools)** メニュー->**パッチデータ (Patch data)** には、プログラムコードを分析した後、一部の部分を修正する必要がある場合に使用できるパッチャーというツールがありました。外部ツールを使わずに必要なバイトを変更できるように、**VB Decompiler** は組み込み機能を持っていました。パッチャーのウィンドウは以下の通りです：



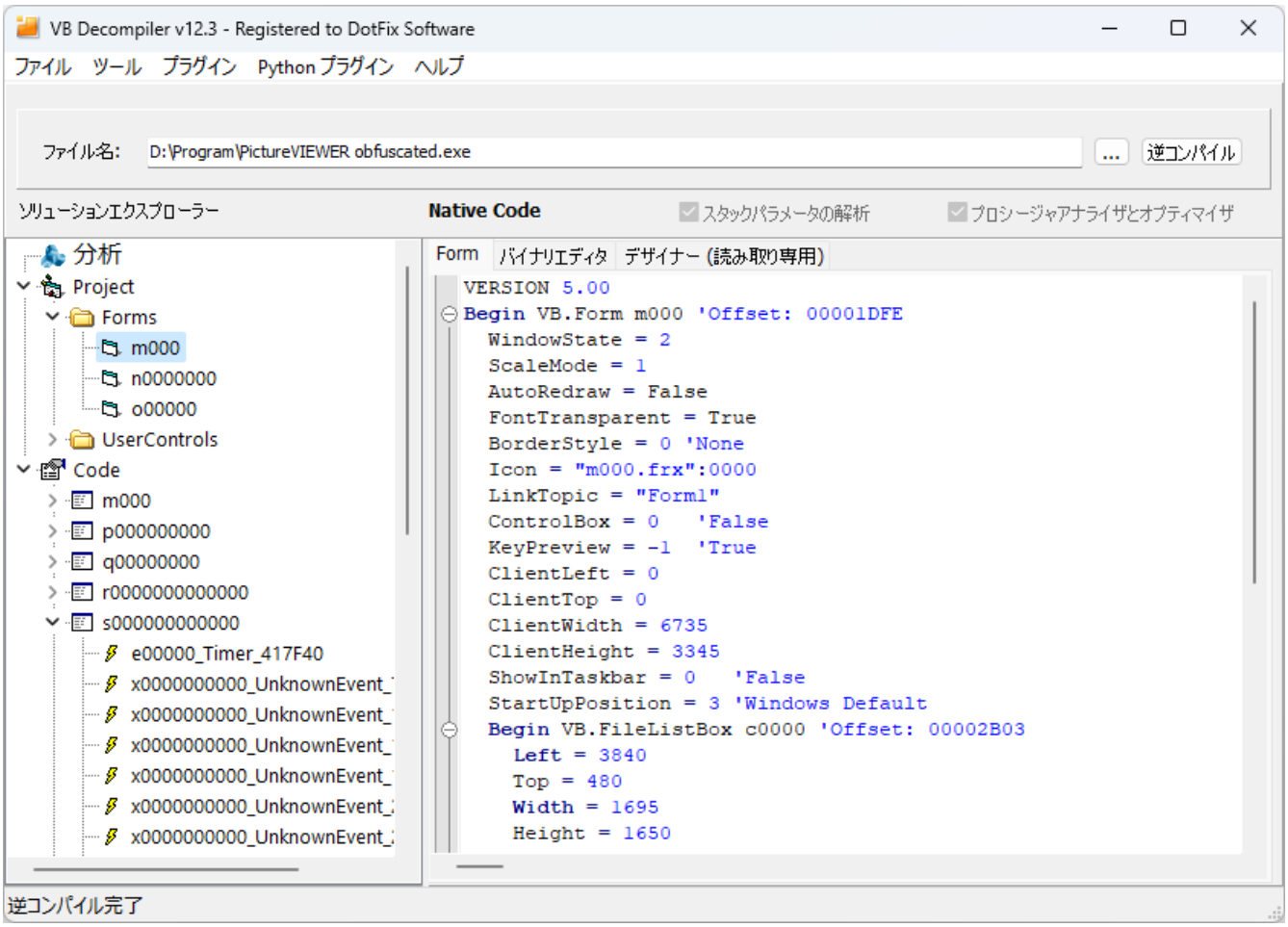

パッチャーとの作業は非常に簡単でした。**仮想アドレス** (**Virtual Address**) フィールドにバイトを編集するアドレスを入力し、その後**Enter** を押すか**Get** をクリックしました。選択されたアドレスの近傍にあるデータバイトが Data フィールドに表示され、選択されたバイトを変更した後、**Set** をクリックして入力したバイトを変更する必要がありました。

プログラムオプション で**高速逆コンパイル** (**Fast decompilation**) のオプションが有効で、**逆コンパイルされたコードをキャッシュする** (**Cache decompiled code**) のオプションが無効の場合、すべての変更はバイトが修正されたその関数に反映されました。これを行うには、ツリー内のその関数の名前でダブルクリックして同じ関数を再オープンするだけで十分でした。

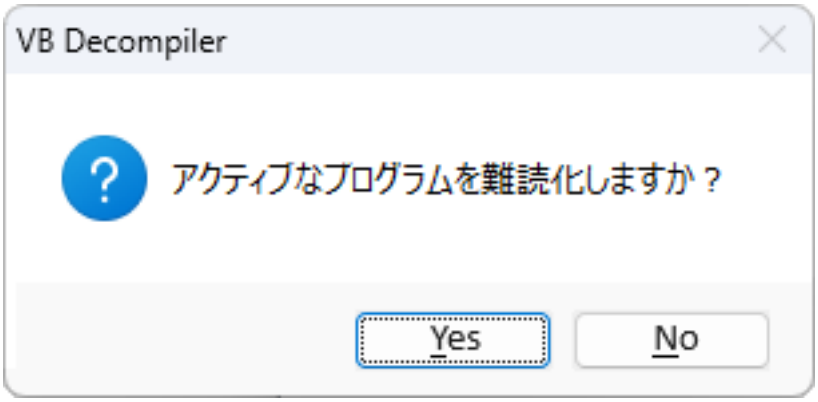
重要な注意点： この機能は、プログラムファイルの編集または適切な権利を持つファイルの編集、およびウイルス対策研究所での悪意のあるコードの研究のために設計されています。他人のコードを変更する前に、そのような権利の存在に関するライセンス契約を確認してください。

## 7.6 難読化

**Visual Basic 5.0/6.0** でプログラムを開発している場合、逆コンパイルに対する保護を強化できます。これを実現するために、**Obfuscator** が逆コンパイラに埋め込まれており、プログラムのバイナリファイルからすべての関数、フォーム、モジュール、およびオブジェクト名を削除し、さまざまなシグネチャも取り除くことができます。これにより、提供されている逆コンパイラまたは他のソリューションを使用してプログラムを分析する際に困難になります。さらに、ほとんどのサードパーティ製製品はこのファイルを逆コンパイルすることを拒否します。




この保護を適用するには、逆コンパイラでプロジェクトを開き、**プロジェクトを難読化 (Obfuscate project)** を **ツール (Tools)** メニューから選択します。このプロセスにはそれほど時間がかからないため、プロジェクトを公開する前に難読化することをお勧めします。




この機能は、ソフトウェア保護のための包括的なソリューション **DotFix NiceProtect** にも統合されており、www.niceprotect.com からダウンロードできます。

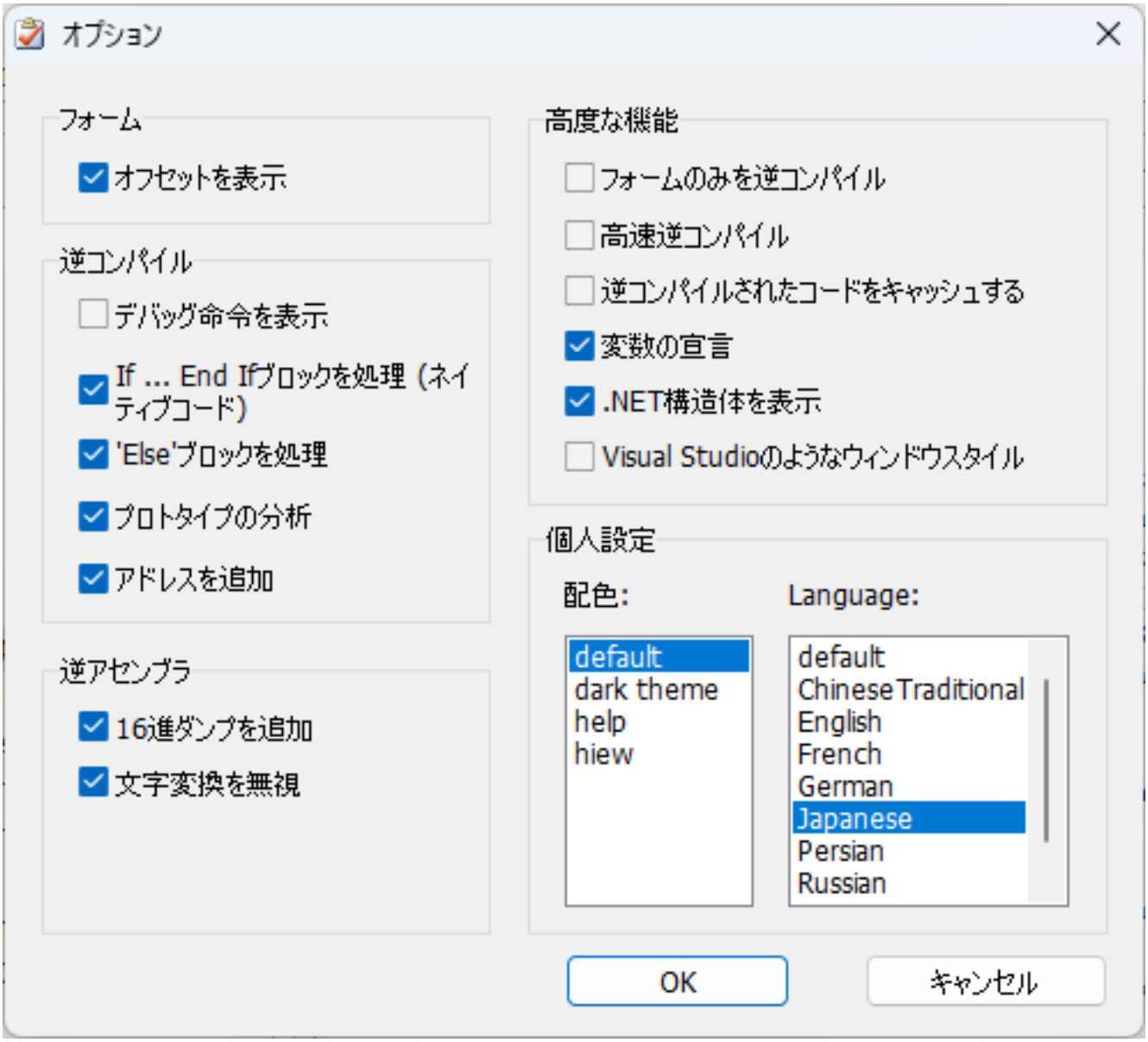
難読化は、ビジネスライセンスを保有する **VB Decompiler** ユーザーのみ利用できます。

難読化制限

# 8 プログラムオプション

## 8.1 プログラムオプション

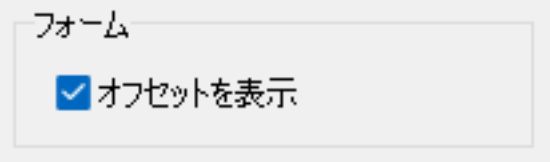
プログラム設定を変更するには、**ツール** (**Tools**) メニュー > **オプション** (**Options**) に移動します。表示されるダイアログには、以下のオプションが表示されます：




フォーム
逆コンパイル
逆アセンブラ
配色 高度な機能 言語の選択

## 8.2 フォーム

以下の逆コンパイル設定をフォームに対して変更できます：




この設定セクションでは、フォームオブジェクトのオフセットを表示する機能を有効にできます。これは、ソースコードへのアクセスがない状態で別の言語でプログラムをローカライズする場合などに必要になる場合があります。**オフセットを表示 (Show offsets)** を使用すると、これらのオブジェクトがプログラム内のどこにあるか、その名前とプロパティを見つけることができます。

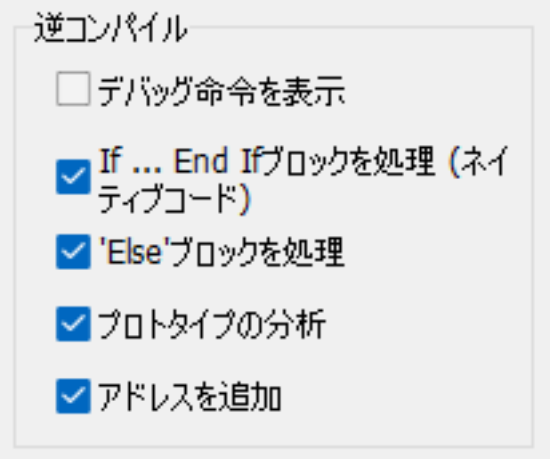
その後、HEX エディタ を使用してこれらの値を変更できます。ただし、バイナリファイルの内部動作に不慣れな場合や、基本的なユーザーレベルの知識しかない場合は、このオプションを有効にするのは推奨されません。ファイルへの誤った変更は機能を損ない、不要なアドレスは単に邪魔になるだけです。

### ビジュアルプレビューとイベントナビゲーション

フォームオプションは **デザイナー (読み取り専用) (Designer (read only))** ビューアとは別です。フォームの視覚的な復元 を参照してください。クラシック VB フォーム、Windows Forms、WPF/BAML サポート、イベントハンドラナビゲーション、および各インターフェース技術の制限について説明されています。

## 8.3 逆コンパイル

**ツール** > **オプション** (**Tools** > **Options**) を開いて逆コンパイル設定を構成します。対話的に設定を比較する場合は、キャッシュなし高速逆コンパイルを使用してください。オプションを変更した後は、ファイルを再度処理してください。




### デバッグ命令を表示 (Show debug commands)

クラシック VB P-Code の場合、高水準リストから通常省略されるコンパイラおよびデバッガーの操作を含めます。再構築されたプログラムロジックだけでなく、基礎となる命令シーケンスを検証する際に有効にしてください。

### If ... End Ifブロックを処理 (ネイティブコード) (Process If ... End If blocks (Native Code))

認識された Native Code 条件分岐を `If...End If` ブロックとして再構築します。元の分岐パターンを検証する必要がある場合、または構造が不正確に見えるプロシージャを調査する場合に無効にしてください。

### 'Else'ブロックを処理 (Process 'Else' blocks)

追加のヒューリスティクスを使用して、Else 分岐とネストされた条件および GoTo ジャンプを区別します。Native Code の場合、If ブロックの処理が必要です。通常の読み取りには有効のままにしてください。再構築された分岐が曖昧な場合は、無効にした状態と比較してください。

### プロトタイプの分析 (Analyze prototypes)

プロシージャの引数および戻り値の予備ヒューリスティック解析を実行します。これは特に高速逆コンパイルで有用です：標準モジュール内のプライベート関数およびプロシージャには、明示的なシグネチャメタデータがない場合があります。予備解析は追加時間を要しますが、現在既知のスタック内容のみに基づいて呼び出しを扱うことを避けます。ヒューリスティックなシグネチャも推論であり、特にメタデータ、外部型、またはコードが不完全な場合です。

### アドレスを追加 (Add Addresses)

再構築されたステートメントの隣に仮想アドレスを表示します。コードを逆アセンブル、ファイルバイト、および他のツールの発見事項と相関させるために有効のままにしてください。また、**逆アセンブラにコピー** (**Copy to disassembler**) で高水準ステートメントとその低水準対応物をマージする必要があるためです。逆アセンブラは常にこの設定に関係なくアドレスを独立して表示します。

## Native Code 再構築の理解

Native Code エンジンは、サポートされている `Select Case` パターン、多次元配列アクセス、COM 呼び出し、および VB ランタイム操作を認識します。エラー処理は `On Error Resume Next`、`On Error GoTo`、および `On Error GoTo 0` として再構築できます。コンパイラ最適化および欠落する型情報は依然として結果を制限します；高水準出力が不完全な場合は逆アセンブルを検証してください。

バージョン 26.2 は、Variant の型判定、多次元 SafeArray 読み取り/書き込み、および配列要素の `VarPtr` 再構築を拡張しました。出力は、分析エンジンが認識したランタイム構造を反映して、配列参照を `SafeArray` と特定する場合があります。ブランチ状態の分離および一時変数の解析も、複雑な If/Else コードの再構築を改善します。バージョン 26.5 はさらに Select Case および VB5/VB6 ランタイム関数処理を改善します。

## COM および ActiveX 型情報

VB Decompiler はまず既知のインターフェースのための組み込み知識ベースを使用します。未知の外部 COM オブジェクトの場合、システムレジストリを通じてライブラリ、クラス、およびインターフェース識別子を解決し、対応する DLL/OCX 型ライブラリを解析できます。これは、事前バインディングおよびサポートされている実行時バインディング呼び出しのためのメソッドおよびプロパティシグネチャ（返されるインターフェース型を含む）を提供します。

必要な型ライブラリが欠落している場合、またはその登録を解決できない場合、一部の呼び出しは特定されない場合があります；例えば `UnkVCall` として。したがって、解析環境ごとの出力を比較する際には、依存する型ライブラリの有無を考慮してください。拡張された IID 知識ベースには、Office オートメーション、ADO/DAO、XML および HTTP、スクリプティング、シェル、メッセージング、およびディレクトリサービスインターフェースが含まれます；すべての可能な COM 呼び出しが解決できることを意味するものではありません。

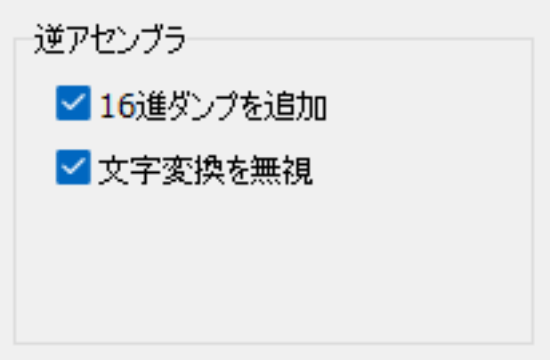
一部の VB バイナリでは、UDT メタデータは限られたものしか残存しません。逆コンパイラは、構造のサイズや文字列フィールド数などの情報を復元できますが、完全な元のフィールドレイアウトを復元しない場合があります。部分的な宣言を完全なソースレベル定義として解釈しないでください。

## .NET 再構築と最適化

サポートされている IL は C# 風の式および制御フローとして再構築されます。**プロシージャアナライザとオプティマイザ (Procedure analyzer and optimizer)** はメインウィンドウのコントロールで、サポートされているループ、条件、演算子、および一時変数のパターンを簡素化できます。サポートされていないまたは難読化されたフラグメントが残っている場合は IL およびメタデータを検証してください。この処理は AI ヘルパー とは独立しています。

## 8.4 逆アセンブラ

このオプションブロックは、マシンコードの逆アセンブルを担当します：




**16進ダンプを追加** (**Add HEX dump**) は、各逆アセンブルされたコマンドの前にマシンコードを表示します。これは、分析後に HEX エディタ でバイナリファイルを修正する必要がある場合にのみ必要になる場合があります。

**文字変換を無視** (**Ignore chars convertions**) は、非英語の文字列データを数値的な Unicode に変換することを無効化するために使用されます。このオプションを有効にすると、表示される文字列が読みやすくなります。数値的な Unicode 表現が必要な場合はこれを無効にしてください。

**ユーザークラスを表示** (**Show user classes**) は、**Delphi** / **Free Pascal** アプリケーション内のユーザークラスを解析を有効にします。このオプションは **Visual Basic** および **C#** アプリケーションには使用されません。

**VCLクラスを表示** (**Show VCL classes**) は、**VCL** ライブラリ内のすべての内部クラスを逆コンパイルを有効にします。このオプションは無効にするのが推奨されており、逆アセンブラが通常、プログラムを分析する際に不要な数千のサービスクラスを分析しないようにするためです。このオプションは **Delphi** アプリケーションの内部目的での逆コンパイルにのみ使用され、他の製品との統一のために作成されましたが、**VB Decompiler** では実装されていません。

**Delphi** および **Free Pascal** アプリケーションの部分的な逆コンパイルは、現在、ソフトウェアではなく別々のサービスとしてのみ提供されています。

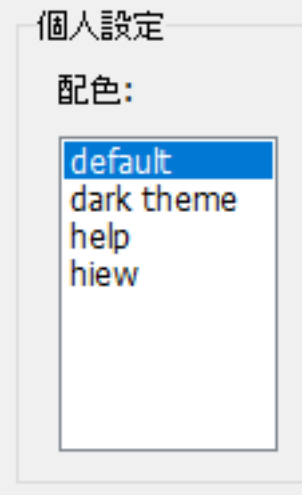
## VB P-Code と .NET IL

**16進ダンプを追加** は低レベルの P-Code リストにも対応しています。復元されたステートメントと P-Code またはネイティブ命令を同じリストで比較するには、逆コンパイラ (Decompiler) ビューから **逆アセンブラにコピー** (**Copy to disassembler**) を使用してください。

.NET アセンブリの場合、逆アセンブラは IL またはネイティブ x86/x64 メソッドを表示できます。復元されたローカル変数のシグネチャには、配列、ジェネリック、値型、および TypeSpec 参照が含まれる場合があります。コードと併せてメタデータテーブルやヒープを調べる場合は、**.NET構造体を表示** (**Show .NET structures**) を有効にしてください。後のリリースでは、フィールドポインタテーブルを含む難読化されたメタデータのサポートが拡張されますが、これは損傷または難読化されたアセンブリのすべてを復元することを保証するものではありません。

## 8.5 配色

このセクションでは、逆コンパイラでコードを表示する際のカラースキームと構文ハイライトを定義します：




プリセットは、VB Decompiler がインストールされているディレクトリ内の .ini ファイルにある "colors" フォルダに格納されます。ファイル名がプログラム設定セクションでのプリセット名を決定します。これらのファイルは単純な形式であり、ご自身で簡単に作成したり、既存のものを編集したりできます。

設定形式は以下の通りです：

```
[Colors]
Background = decimal color code (for example: 1973790)
Foreground = decimal color code (for example: 14671840)
KeyWords = decimal color code (for example: 13471849)
LocationsAndArguments = decimal color code (for example: 11849128)
Procedures = decimal color code (for example: 5676246)
; next properties is only available in version 12 or later
UnderCursorBackground = decimal color code (for example: 6986069)
UnderCursorForeground = decimal color code (for example: 1973790)
UseDarkMode = 0 or 1 (0 for Light Theme, 1 for Dark Theme)
Colors2 = 0 or 1 (using extended colors at Colors2 section)
```

```
[Font]
Name = font name (for example: Consolas)
Size = font size (for example: 11)
KeyWordsBold = 0 or 1 (font bold)
LocationsAndArgumentsBold = 0 or 1 (font bold)
ProceduresBold = 0 or 1 (font bold)
```

```
; this section is only available in version 12 or later
[Colors2]
TVFontName = Solution Explorer treeview font name or list of font names, separated by comma (for example:
    Segoe UI, Tahoma, Arial)
TVFontSize = size of the font (for example: 10)
FontName = Decompiler and Disassembler font name or list of font names, separated by comma (for example:
    Courier New, Consolas, Liberation Mono, monospace)
FontSize = size of the font (for example: 11)
Background = BackColor in hexadecimal for Decompiler and Disassembler tab (for example: 1E1E1E)
Foreground = ForeColor in hexadecimal for Decompiler and Disassembler tab (for example: DFDFE0)
; code highllighting
Key = color in hexadecimal (for example: CD9069)
Comment = color in hexadecimal (for example: CD9069)
Char = color in hexadecimal (for example: B4CDA8)
Float = color in hexadecimal (for example: B4CDA8)
Hex = color in hexadecimal (for example: B4CDA8)
Number = color in hexadecimal (for example: B4CDA8)
String = color in hexadecimal (for example: 569CD6)
Symbol = color in hexadecimal (for example: DFDFE0)
; HEX Editor font and colors
HexFontName = font name or list of font names, separated by comma (for example: Courier New, Consolas,
    Liberation Mono, monospace)
HexFontSize = size of the font (for example: 11)
HexBackground = color in hexadecimal (for example: 1E1E1E)
HexForeground = color in hexadecimal (for example: DFDFE0)
HexGrid = color in hexadecimal (for example: 1E1E1E)
HexActiveFieldBackground = color in hexadecimal (for example: 1E1E1E)
HexOffsetBackground = color in hexadecimal (for example: 1E1E1E)
HexCurrentOffset = color in hexadecimal (for example: B4CDA8)
HexOddColumn = color in hexadecimal (for example: B4CDA8)
HexEvenColumn = color in hexadecimal (for example: B4CDA8)
HexOffset = color in hexadecimal (for example: 569CD6)
```

カラーコードは 32 ビットの真色形式で指定されます。Colors セクションでは 10 進数（VB Decompiler の旧バージョンとの互換性のため）、Colors2 では 16 進数です。

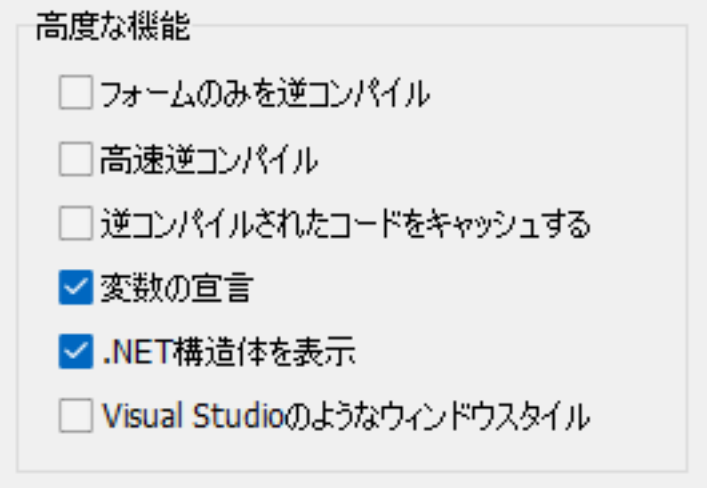
## コード解析中のテーマ変更

テーマシステムはダークモードをサポートしており、VB Decompiler を再起動せずに選択したテーマを適用できます。INI ファイルの編集後にも同様です。拡張設定内のフォント設定には、コンマ区切りのフォールバック名を含めることができ、Windows と互換環境間で同等のフォントが異なる場合に役立ちます。

カーソル下の色は識別子のハイライトを制御します。ローカル変数、メンバー変数、グローバル変数のハイライトについては コードの表示 を参照してください。

## 8.6 高度な機能

これらの設定は、プロジェクトの処理方法と復元された情報の表示方法を制御します。




### フォームのみを逆コンパイル (Decompile only forms)

復元されたフォームと UserControl の定義のみが必要な場合に、コードの逆コンパイルを無効にします。ビジュアルビューアとそのテクノロジ固有の機能については、フォームの視覚的な復元を参照してください。

### 高速逆コンパイル (Fast decompilation)

有効にすると、VB Decompiler はまずプロジェクト構造を処理し、個別のプロシージャを開くときにそれらを逆コンパイルします。これにより、選択されたプロシージャを調査する際の初期待機時間が短縮されます。**逆コンパイルされたコードをキャッシュする** (**Cache decompiled code**) を無効にすると、分析設定を変更してプロシージャを再オープンし、新しいリストを取得できます。

**プロトタイプの分析** (**Analyze prototypes**) を逆コンパイル設定で有効にすると、プライベートプロシージャのシグネチャを事前にヒューリスティック解析します。初期処理時間は増えますが、完全な逆コンパイルを行わずに引数の復元を改善できます。

高速逆コンパイルでは、プロジェクトの完全な検索可能な高水準コードリストは提供されません。それを無効にして、ファイルの再処理を行い、プロジェクト全体の テキスト検索 や完全な逆コンパイルプロジェクトのエクスポートを使用してください。文字列参照 は異なります：それらは予備的な逆アセンブルを通じて収集でき、高速逆コンパイルモードでも利用可能です。

### 完全逆コンパイル

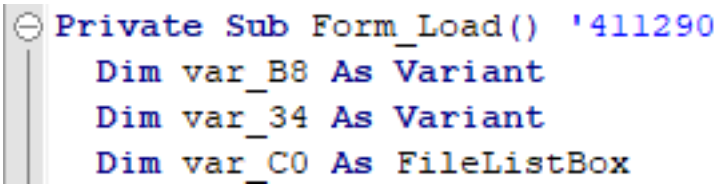
高速逆コンパイルを無効にすると、逆コンパイラはプロジェクトのプロシージャを事前に処理します。これにより時間がかかりますが、生成された完全なリストがテキスト検索とエクスポートのために利用可能になります。逆コンパイル設定の変更には、ファイルの再処理が必要です。

### 逆コンパイルされたコードをキャッシュする

高速逆コンパイルモードで既に生成されたリストを保持します。これにより、開かれたプロシージャを繰り返し逆コンパイルする必要がなくなります。ただし、キャッシュされた結果は後の分析設定の変更を反映しません。**スタックパラメータの解析** (**Parse stack parameters**) または **プロシージャアナライザとオプティマイザ** (**Procedure analyzer and optimizer**) の設定を対話的に比較する場合は、キャッシュを無効にしてください。

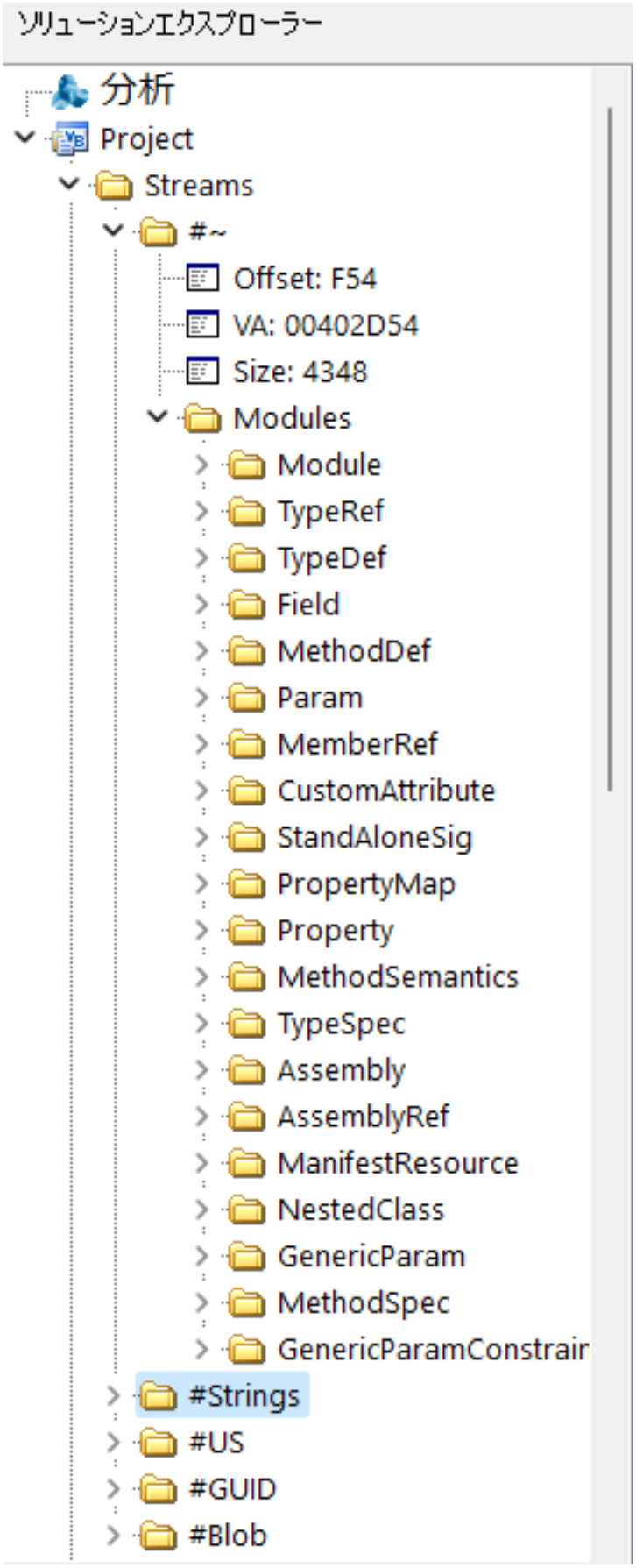
## 変数の宣言 (Declare variables)

復元された変数に対して、各逆コンパイルされたプロシージャの開始部分に宣言ブロックを追加します。これはソースコードの復元と型検査に役立ちます。プロシージャの処理内容を簡潔に確認したい場合は、宣言を非表示にできます。復元された型は、逆コンパイラが利用可能なメタデータと分析の対象となります。




## .NET構造体を表示 (Show .NET structures)

ソリューションエクスプローラー (Solution explorer) に .NET メタデータテーブルとヒープを表示します。これを使用して、コードと一緒に構造とシグネチャを検査します。テーブルを選択するとその内容が表示されます。#US ヒープにはユーザー文字列が含まれています。メタデータナビゲーションが必要ない場合はオプションを無効にしてください。


ソリューションエクスプローラー
分析
Project
Streams
#~
Offset: F54
VA: 00402D54
Size: 4348
Modules
Module
TypeRef
TypeDef
Field
MethodDef
Param
MemberRef
CustomAttribute
StandAloneSig
PropertyMap
Property
MethodSemantics
TypeSpec
Assembly
AssemblyRef
ManifestResource
NestedClass
GenericParam
MethodSpec
GenericParamConstrair
#Strings
#US
#GUID
#Blob



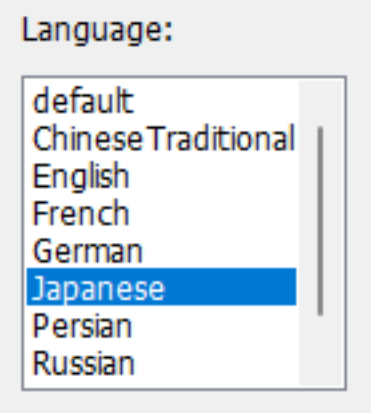

## Visual Studioのようなウィンドウスタイル (Window style as in Visual Studio)

コードを左側、ソリューションエクスプローラー を右側に配置します。ワークスペースの配置を変更するだけで、復元されたコードは変わりません。

## 8.7 言語の選択

**VB Decompiler** は、世界中で使用されている 20 を超える言語に対するインターフェースのローカライズを提供します。ドキュメントも複数の言語で利用可能です。

プログラムで使用する好ましい言語インターフェースを選択するには、**ツール** (**Tools**) -> **オプション** (**Options**) メニュー内の**個人設定** (**Personalize**)設定で選択します。言語を選択した後、言語設定をプログラム全体に正しく適用するには**VB Decompiler** を再起動する必要があります。プログラムの再起動が望ましくない場合は**VB Decompiler** を引き続き使用できますが、一部のテキストは以前の言語で表示される場合があります。




ヘルプの言語は、お使いのコンピュータのシステム言語に基づいて自動的に選択されます。別の言語でマニュアルを読むには、**VB Decompiler** がインストールされているフォルダから対応するヘルプファイルを開きます。ローカライズされたマニュアルの主要な形式は PDF です。CHM ヘルプファイルも一部の言語で利用可能です。

英語のマニュアルがすべての翻訳の元になります。ローカライズされたマニュアルにエラーや不整合を発見した場合は、テクニカルサポートへご報告ください。

### マニュアルへのアクセス

ローカルヘルプファイルまたは必要な PDF ビューアが利用できない場合、VB Decompiler はオンラインドキュメントを開くことができます。オフライン調査環境では、必要なローカルヘルプファイルを事前に用意しておいてください。

# 9 プラグイン

プラグインはネイティブ DLL を通じて VB Decompiler の機能を拡張します。インストールされたプラグインは **プラグイン** (**Plugins**) メニューから利用可能です。コールバック API により、復元されたプロジェクト情報やコードへのアクセスが可能となり、プラグインがサポートされている内部構造を検査または修正できるようになります。

## VB/DE Decompiler MCP Server 2.0.2

VB DecompilerまたはDE Decompilerで開いているプロジェクトをMCPクライアントから調べるためのプラグインです。ホストのプラグインSDKを通じて、モジュール、フォーム、プロシージャ、復元されたコードを参照できます。解析対象のプログラムは実行されません。

MCP Server 2.0.2（x86ホスト用）をダウンロード

`vb_get_form`は、フォームのインデックス`form_index`、名前`form_name`、プロパティのテキスト`form_text`を返します。名前を取得できない場合、`form_name`は空文字列になります。 MCPツール一覧と使用例

### 動作要件

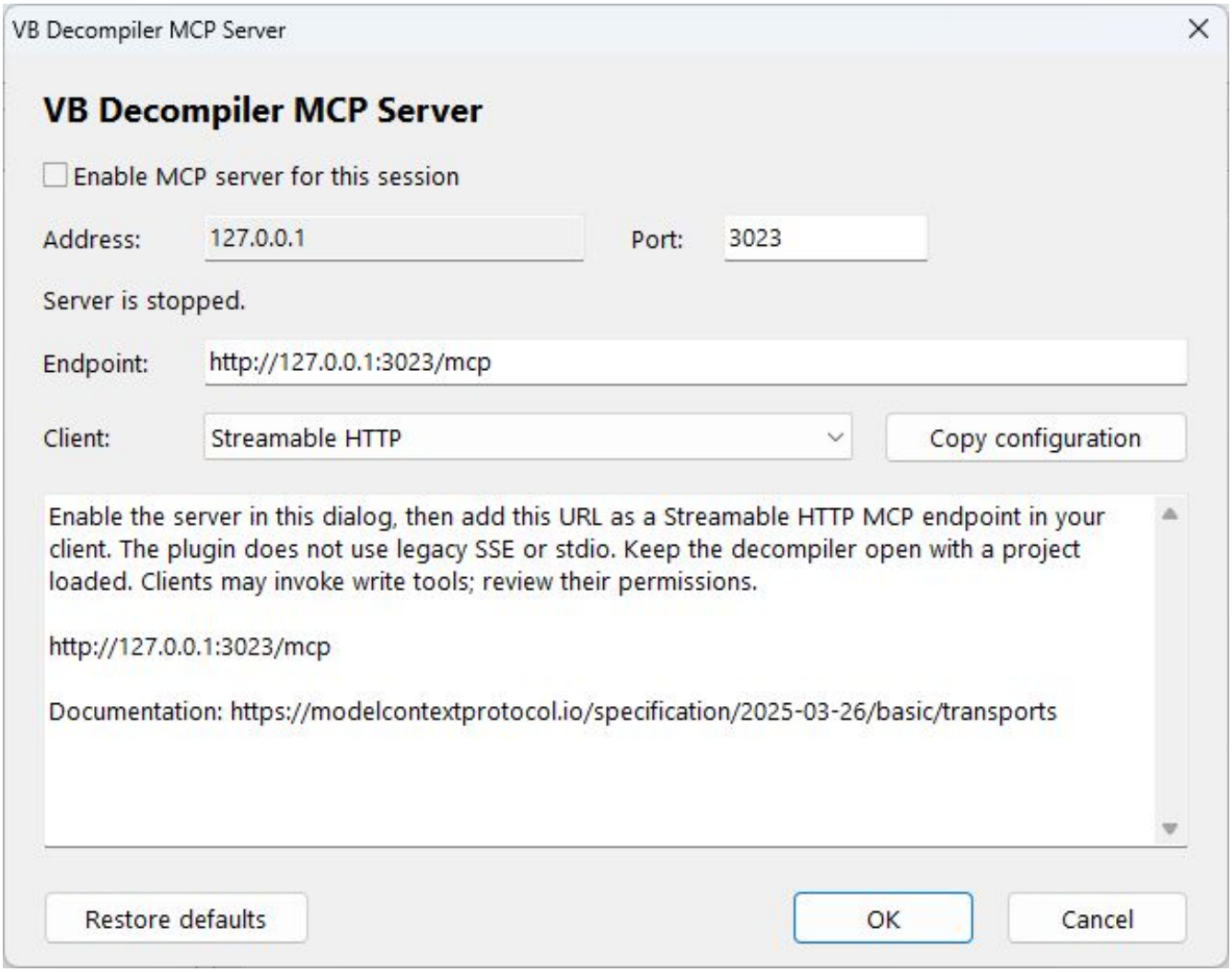
ネイティブDLLプラグインの読み込みに対応したWindows版のVB DecompilerまたはDE Decompilerを使用してください。

配布するDLLはWindowsの32ビット版ホスト用です。DLLのアーキテクチャは、解析対象のファイルではなく、プラグインを読み込む逆コンパイラに合わせてください。64ビットファイルの解析可否は、ホストの対応範囲によります。

利用できるツールは、逆コンパイラのバージョン、エディション、開いているファイルによって異なります。旧バージョンのSDKでは、取得済みの逆コンパイル結果を参照するツールや、利用できない作業領域の操作があります。この移植版のツール名は`vb_`で始まります。別の実装から移行した場合は、クライアント側でツール一覧を更新してください。

### インストールと起動

1. 逆コンパイラを終了し、アーカイブを展開します。`vbmcp.dll`をインストール先の`plugins`サブフォルダーにコピーします。C版をインストール済みの場合は、既存のDLLを置き換えてください。
2. 逆コンパイラを起動し、解析権限のあるファイルを開きます。**Plugins → VB/DE Decompiler MCP Server**を選択します。
3. **Enable MCP server for this session**にチェックを入れ、**OK**をクリックします。ダイアログに表示された接続先URLをコピーします。




VB DecompilerのMCP Server設定画面。 プラグインは読み込み元の逆コンパイラを識別し、その製品名をウィンドウのタイトルに表示します。

## MCPクライアントの接続

Streamable HTTPを使用します。既定の接続先は`http://127.0.0.1:3023/mcp`です。両方の逆コンパイラでMCPサーバーを同時に起動する場合は、それぞれに異なるポートを設定し、接続したい側のダイアログに表示されるURLを使用してください。

### Codex

```
codex mcp add vbmcp --url http://127.0.0.1:3023/mcp
```

### Claude Code

```
claude mcp add --transport http vbmcp http://127.0.0.1:3023/mcp
```

### VS Code

VS Codeでは、`.vscode/mcp.json`に以下の設定を追加するか、**MCP: Add Server**でHTTPを選択します。

```
{
  "servers": {
    "vbmcp": {
      "type": "http",
      "url": "http://127.0.0.1:3023/mcp"
    }
  }
}
```

クライアント設定のリファレンス · Codex · Claude Code · VS Code

#### 接続の確認

フォームを含むサンプルを開きます。読み取り専用の`vb_list_forms`で`limit`を`5`に指定し、フォームを最大5件一覧表示するようクライアントに指示します。接続できない場合は、現在のセッションでサーバーが有効か、接続先がダイアログの表示と一致しているかを確認してください。

#### ローカルアクセスとプロジェクトの変更

サーバーは`127.0.0.1`でのみ待ち受け、ユーザー認証は行いません。接続できるローカルアプリケーションは、プロジェクトの読み取りに加え、逆コンパイラの作業領域でリストや関数のマークの変更を要求できます。クライアントの書き込み操作の承認は有効にしておいてください。サーバーを停止するには、**Enable MCP server for this session**のチェックを外して**OK**をクリックします。逆コンパイラを起動するたび、サーバーは停止した状態から始まります。

プラグインの通信先はローカルです。取得したコードやメタデータをモデルの提供元に送るかどうかは、MCPクライアントの設定によります。

DotFix SoftwareによるDelphi移植版です。Cで開発されたオリジナルのプラグインを基にしています。MITライセンスで配布します。

## FRX Image Viewer

FRX Image Viewer 1.2.0 は、VB Decompiler または DE 逆コンパイラ (Decompiler) が登録した画像を表示・抽出します。VB5/6 および .NET のフォーム画像には VB Decompiler、Delphi のバイナリ DFM 画像には DE 逆コンパイラ を使用します。BMP、PNG、JPEG、GIF、ICO、CUR に対応しています。

#### インストールと起動

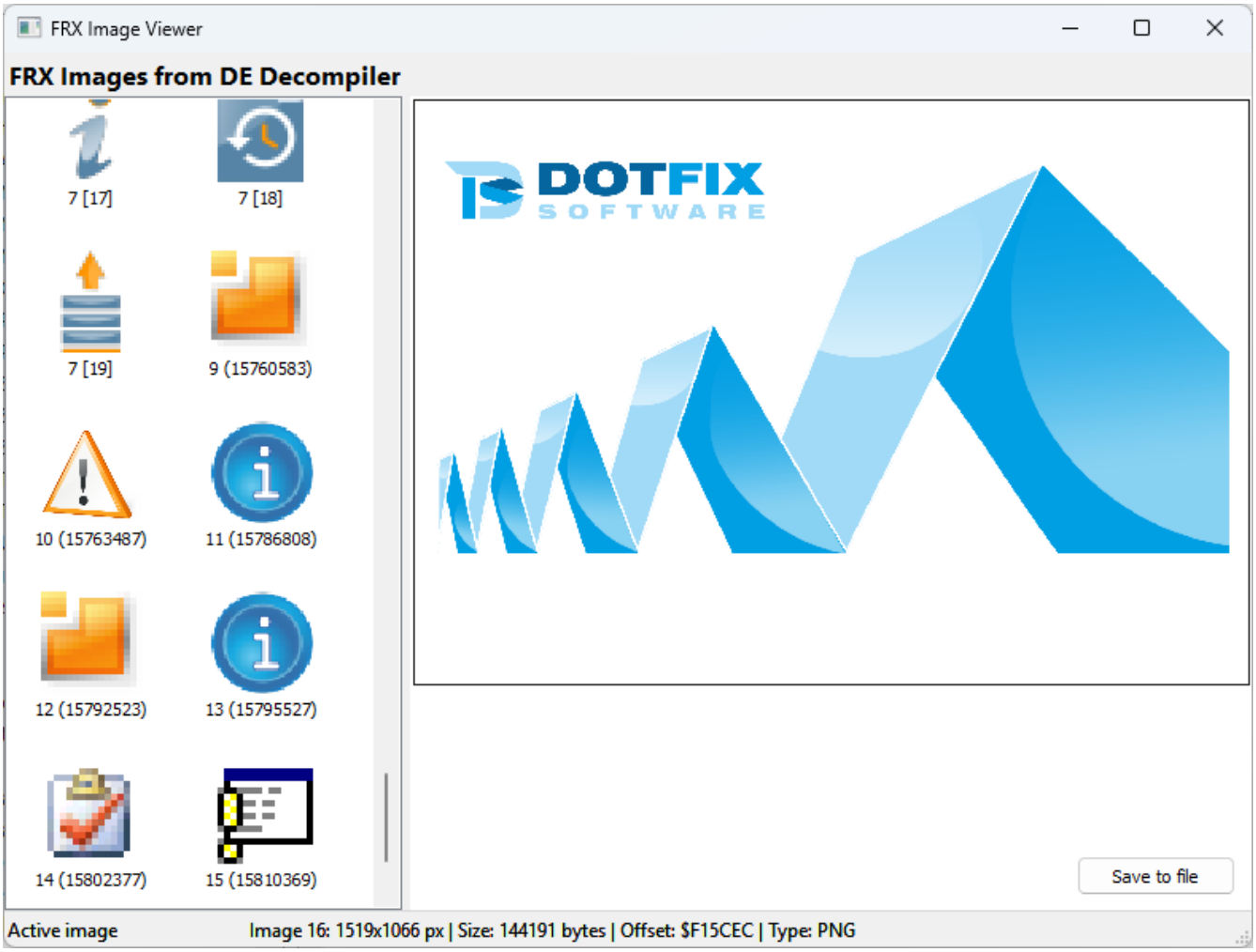
1. `FRXViewer.dll` をホストのインストール先にある `plugins` サブフォルダーへコピーし、逆コンパイラを再起動します。DLL のアーキテクチャは、解析対象ファイルではなくホストアプリケーションに合わせてください。
2. EXE または DLL を開き、フォームを解析してホストに画像を登録させます。
3. **プラグイン** > **FRX Image Viewer** を選択し、サムネイルをクリックします。

1.2.0 ダウンロードには x86 ホスト用 DLL が含まれます。x64 ホストには対応するビルドが必要です。プラグインには Windows 10/11 が必要です。ビルド手順はソースコードリポジトリにあります。

### プレビューとエクスポート

ギャラリーとプレビューには、複数の画像を含む ICO/CUR ファイルなどの利用可能な画像が表示されます。ステータス欄には項目、寸法、データサイズまたは元ビットマップのオフセット、形式が表示されます。ホストのテーマに追従し、ダークモードにも対応します。

**Save to file** を選択して項目をエクスポートします。独立した画像は元のバイト列と形式を保持します。ImageList の各セルはマスクとアルファ透過を保持した ICO として保存されます。この ICO は元のビットマップから生成されたものです。




DE 逆コンパイラ の FRX Image Viewer。Delphi DFM 画像と ImageList の個別アイコンを表示しています。

### ネイティブ Delphi ImageList と互換性

対応ホストでは、認識されたネイティブ Delphi ImageList が個別のアイコンとして表示されます。コレクション項目内で独立して符号化された画像は通常の個別画像のままです。未対応のストリームでは、元の画像エントリが利用できる場合、画像全体の処理を維持します。

VB Decompiler 26.6 と DE 逆コンパイラ 26.5 が対応する共通 SDK リビジョン 26.6 では、省略可能な `GetFrxImageListInfo` (63) と `GetProductName` (64) が追加されています。古いホストでは従来の画像 API と画像全体の処理を維持します。機能検出とオフセットの規則はSDK リファレンスを参照してください。

表示と抽出では、解析対象ファイルを実行せず、そのコンポーネントも読み込みません。アニメーション GIF は最初のフレームを表示します。利用可能なリソースは、ホストが正常に解析・登録した画像に依存します。

## プラグインの開発

SDK には Visual C++、Rust、Delphi、Free Pascal 向けの例が含まれています。エクスポートされたエントリーポイント、コールバックシグネチャ、コマンド識別子、データ表現、および例プロジェクトへのリンクについては VB Decompiler プラグイン API リファレンス を参照してください。

DLL プラグインは名前と著者情報を提供し、`VBDecompilerPluginLoad` 経由でホストコールバックを受信します。このコールバックを使用してサポートされているプロジェクトデータを要求または更新します。このインターフェースを実装する際は、SDK の呼び出し規約、ポインタ、文字コード化、バッファ要件に従ってください。マネージド言語を使用する場合は、ネイティブインターフェースと互換性のある相互運用層を用意してください。

VB Decompiler は起動時に利用可能な DLL プラグインを初期化します。意図したホストアーキテクチャと Windows 環境向けにプラグインをビルドし、その環境で生成された DLL を検証してください。SDK はアクティブビューコードとプロジェクトレベルのコードアクセスの両方を公開しており、すべての逆コンパイル済みプロシージャを必要とする操作では、完全な逆コンパイルが必要です。必要に応じて生成したコード表示だけでは不十分です。

# 10 フォームの視覚的な復元

**デザイナー (読み取り専用) (Designer (read only))** タブは、解析されたバイナリからサポートされているインターフェースのビジュアル再構築を表示します。復元されたすべてのプロパティや初期化ステートメントを手動で解釈することなく、インターフェース要素をアプリケーション動作と関連付けるのに役立ちます。

## 復元されたフォームを開く

1 アプリケーションを開き、VB Decompiler がプロジェクト構造を復元させるのを待ちます。

2 ソリューションエクスプローラー (Solution explorer) で復元されたフォームを選択し、**デザイナー (読み取り専用)** を開きます。

3 検査するコントロールを選択し、利用可能なプロパティと復元されたイベント情報を確認します。イベントを使用して対応するハンドラコードにナビゲートします。

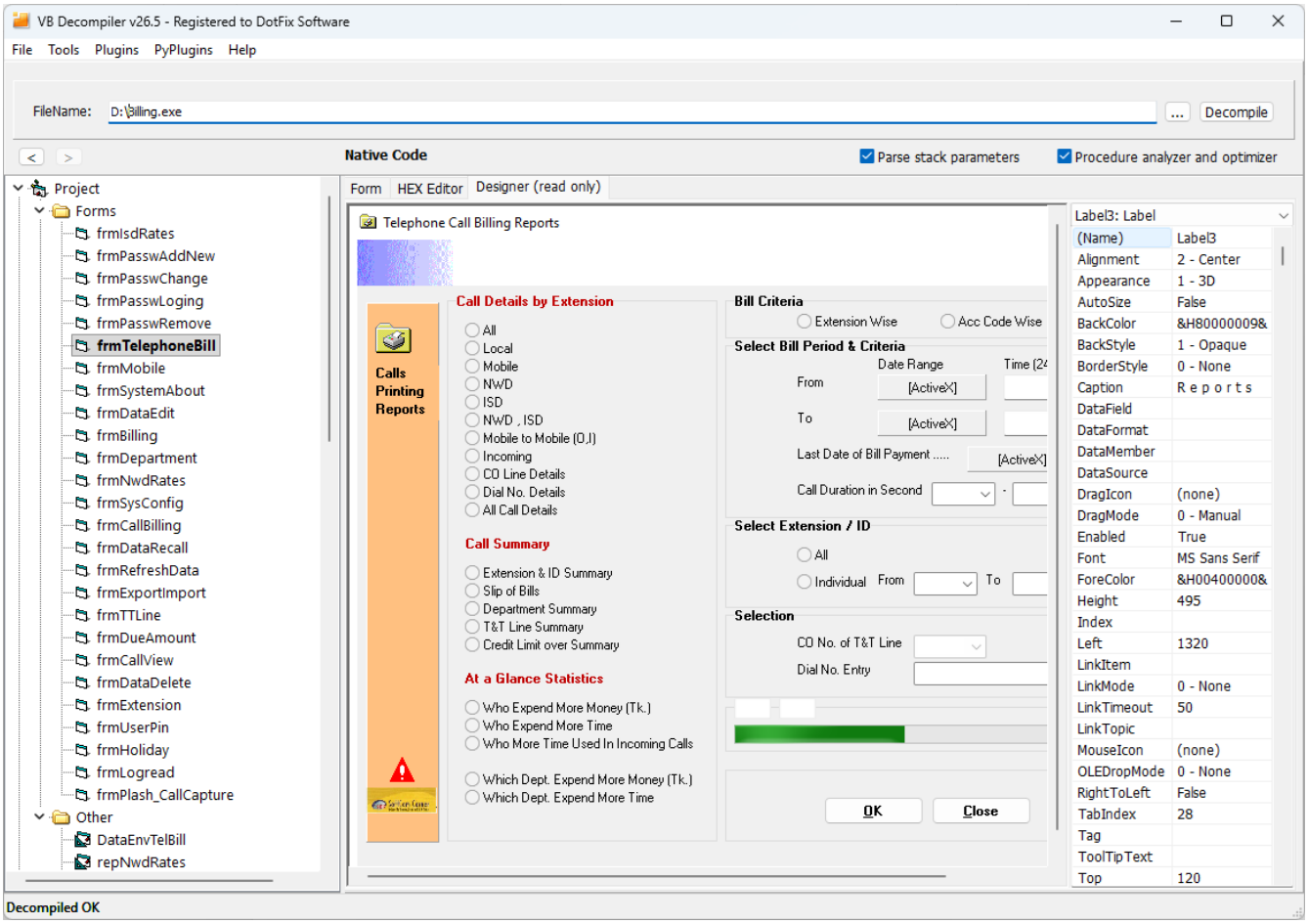
プレビューは保存された情報から再構築されます。元のインターフェースの実行インスタンスではありません。サポートされていないコントロール、ランタイムで生成されたコンテンツ、不足しているリソース、およびカスタムレイアウトロジックは、不完全または近似の結果を生成する可能性があります。

## Visual Basic 5.0/6.0

ビューアはフォームと UserControl を再構築し、サポートされているコントロールプロパティ、フォント、色、および画像を含みます。リスト、コンボボックス、および他のコントロールで使用されるバイナリフォームデータを読み取ります。画像にはアイコン、カーソル、ビットマップ、JPEG、PNG、および GIF データが含まれることができます。

ビジュアルメニューの再構築は P-Code および Native Code アプリケーションの両方をサポートします。サポートされているコントロールには `SSTab`、`MSComctlLib.StatusBar`、`MSComctlLib.ListView`、`MSComctlLib.ProgressBar`、および `MSComctlLib.TreeView` があります。

クラシック VB コントロールのハンドラを検査するには、コントロールを選択し、プロパティリストの **_Event** フィールドをダブルクリックします。結果として得られるリストから必要なイベントを選択してハンドラを開きます。

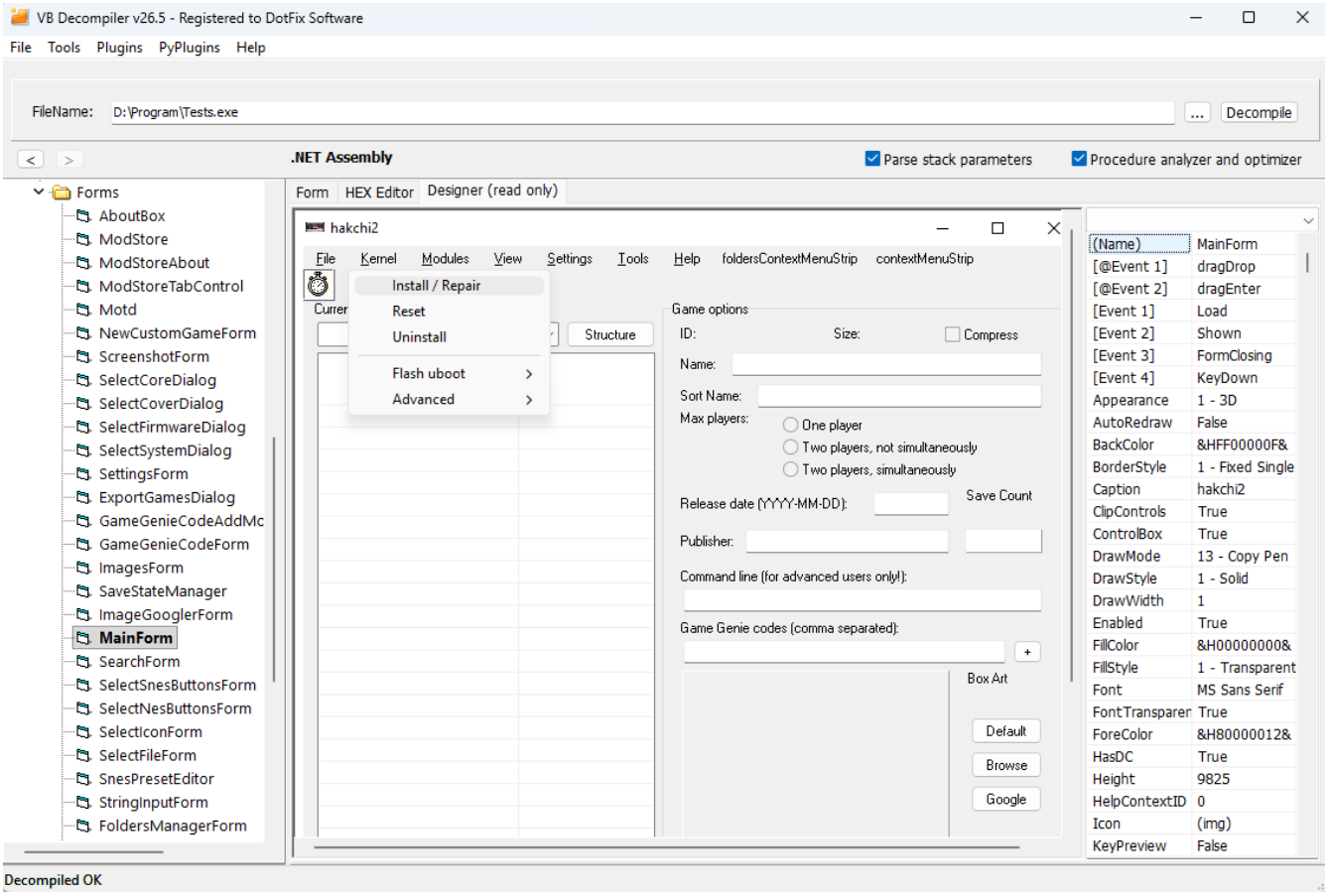



## .NET Windows Forms

VB Decompiler は `InitializeComponent` を静的に分析し、サポートされている Windows Forms コントロールとそのレイアウトを再構築します。パーサーとレンダラーは .NET ランタイムまたはターゲットアプリケーションの実行を必要としません。復元された画像もコントロールおよびフォーム背景に表示できます。

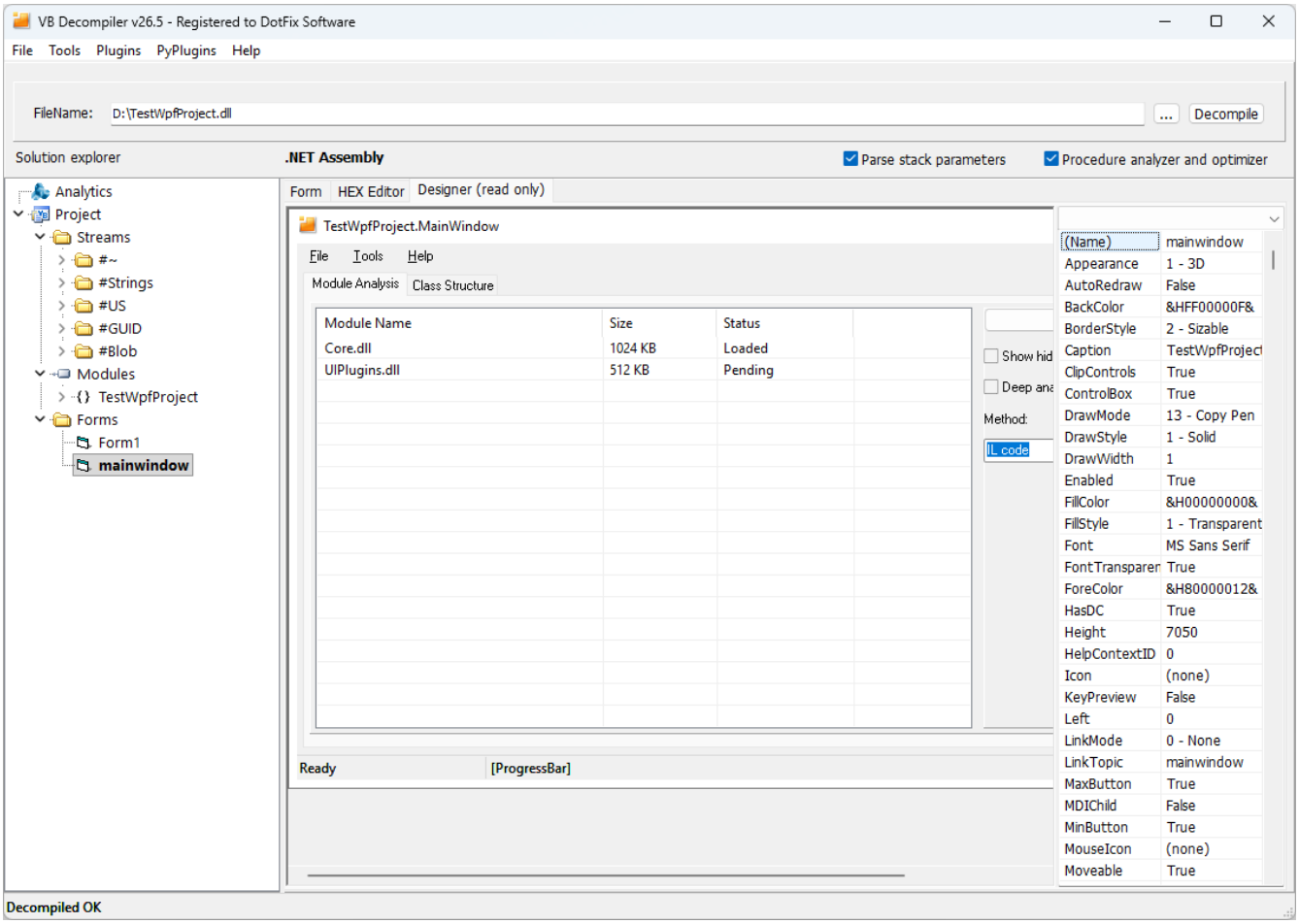
コントロールを選択して復元されたイベントを検査し、次にイベントをダブルクリックして対応するハンドラを開きます。ビューアはメニューおよびコンテキストメニュー階層を再構築します；復元されたメニュー項目をクリックすると対応する Click イベントが開きます。カスタム Windows Forms イベントは @ プレフィックスで表示されます。

ビューアはコントロール Z 順序、継承されたフォアグラウンドおよびバックグラウンド色、`Color.FromArgb` 値、数値 `FontStyle` フラグ、Boolean リソースプロパティ、および `MinimumSize` とともに `ClientSize` を処理します。




## WPF ウィンドウ、ページ、およびユーザーコントロール

ビューアは **BAML** として保存されたコンパイルされた XAML リソースからサポートされている WPF ウィンドウ、ページ、およびユーザーコントロールを再構築します。




サポートされているレイアウトコンテナには、固定、Auto、および比例 (*) 行および列サイズを持つ **Grid**、**StackPanel**、**DockPanel**、および **Canvas** が含まれ、ネストされたレイアウトを含みます。再構築はサポートされているサイズ、マージン、パディング、アライメント、方向、ドッキング、および Grid スパンを考慮します。

共通入力コントロールを超えて、ビューアはタブ、グループボックス、エクスパンダー、メニュー階層およびセパレータ、ステータスバー、ListView 列およびサポートされている静的データ、静的 TreeView ノード、および静的コンボボックス項目を再構築できます。リボンインターフェース、ナビゲーションペイン、およびドッキングされたパネルは簡略化された表現を持つことができます。

リソースデータが利用可能な場合、ビューアは名前、表示テキスト、ツールチップ、フォント、背景色、有効状態と表示状態、埋め込まれた画像（サポートされている PNG 透過性と背景不透明度を含む）を復元します。サポートされていないカスタムコントロールは元の WPF タイプを識別するプレースホルダーとして表示されます。

WPF ビューアーは、復元されたコントロールのイベントを対応するハンドラーに関連付けます。ビューアーのイベント情報から、対応する逆コンパイル済みコードを開くことができます。

**復元の制限：** ビジュアルプレビューは近似です。テンプレート、スタイル、バインディング、カスタムコントロール、および動的に生成されたコンテンツは部分的に表現される可能性があります。

## フォームデータの検査と抽出

オフセットを表示 (Show offsets) 設定およびリンクされたバイナリ検査については フォーム を参照してください。FRX Image Viewer プラグイン を使用してサポートされている復元された画像アセットを検査し、エクスポートします。

# 11 トレース

**Tracing features** は、Visual Basic 5.0/6.0 の Native Code と P-Code をステップごとにエミュレートする機能です。利用にはトレース機能付きのライセンスが必要です。トレーサーはエミュレーター内のプロシージャの内部表現を処理し、ワークステーションの CPU で対象プロシージャを実行することはありません。

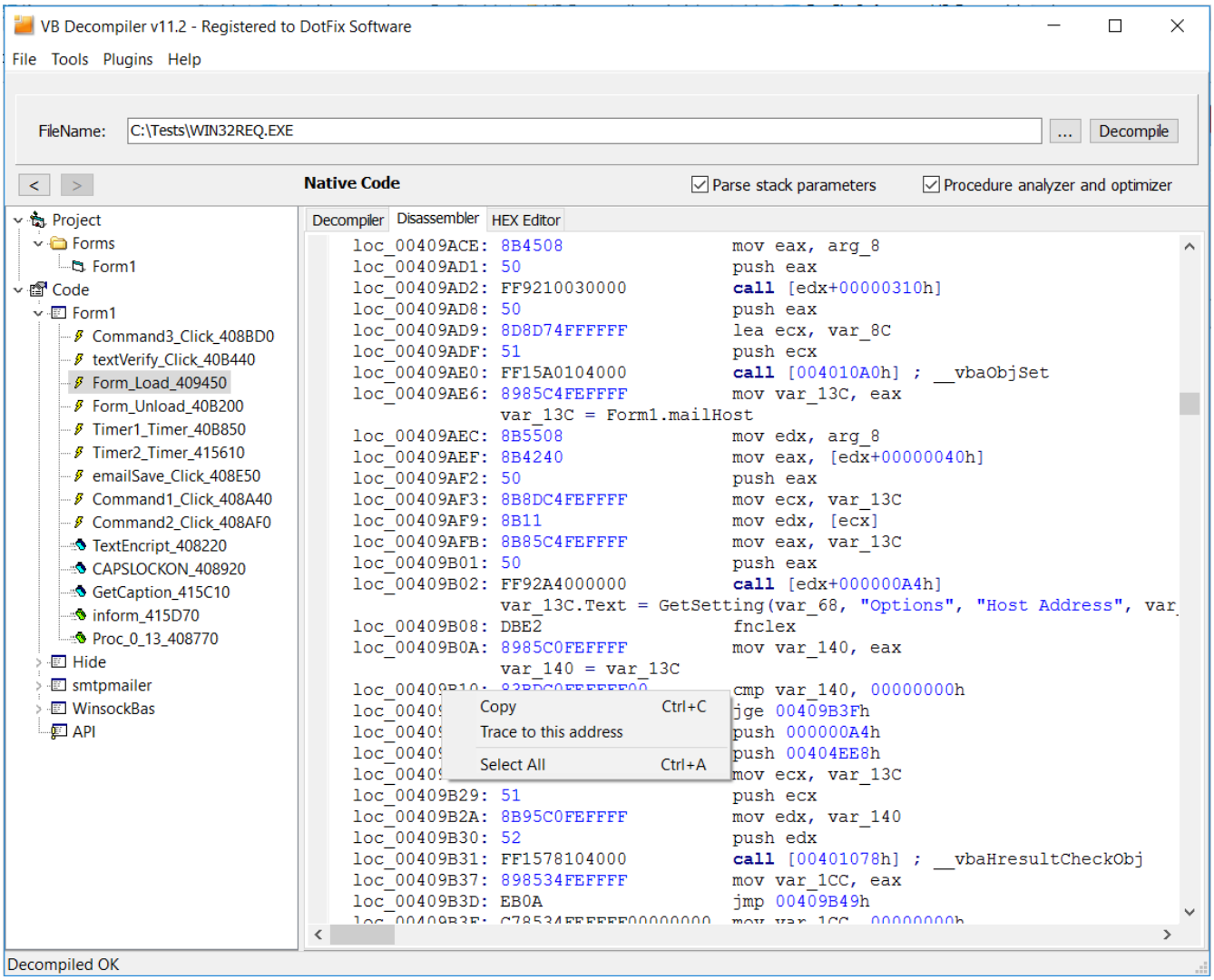
## トレースの開始

1 VB5/6 プロシージャを開き、リストにアドレスを表示したままにします。

2 調査する行で、コードコンテキストメニューから**このアドレスまでトレース** (**Trace to this address**)を呼び出したり、**F4**を押したりします。

3 トレーサーのコードおよび状態表示を検査し、**次** (**Next**)を使用して進めます。完了したらトレースをキャンセルします。

バージョン 26.4 以降では、コマンドは行の先頭の空白をトリミングした後、loc_アドレスを解決できます。カーソルはそのアドレスラベルに正確には配置しなくても、その行の他の場所に配置できます。



## ネイティブコードの状態

エミュレーションが進むにつれて、トレーサーは CPU および FPU レジスタ、スタック、変数、および再構築されたコードを表示します。これらのビューを使用して、値または式がどのように導出されたかを調べます。変数リストの検索と現在のオブジェクトの復元された型の表示は、大きなプロシージャに役立ちます。

## P-Code のトレース

バージョン 11 で導入された P-Code のトレースでは、変数の情報および逆コンパイルされたステートメントが処理中にどのように発展するかを示します。これは、中間の P-Code 操作と最適化された高水準リスト間の違いを説明するのに役立ちます。

## スコープと制限

- トレースの対象は個々のプロシージャです。実行中のプロセスの状態を使用するのではなく、エミュレートするレジスター、スタック、変数が空の状態から開始します。
- サポートされていないネイティブコード命令はスキップされます。外部関数の呼び出しは実行されません。サポートされている呼び出し規約の処理は、外部関数の動作を再現せずにモデル化されたスタックを調整できます。
- トレーサーは完全なプロセッサフラグ表示を提供しません。その状態を完全なハードウェア実行トレースとして解釈しないでください。
- トレースは前方向へのステップ単位のエミュレーションであり、逆方向のトレースやループの繰り返し実行は行いません。
- このワークフローは VB5/6 のトレースを説明します。.NET の逆アセンブル機能があることは、.NET ランタイムデバッガーがあることを意味しません。

逆コンパイラの再構築および中間値を検査するためにトレースを使用します。エミュレートされた値のみでは、初期化されたグローバル、外部依存関係、および入力データを持つ実際のプロセスが生成する値を保証するものではありません。

# 12 解析機能

**Analytic features** を持つライセンスは、解析されたプログラムで認識された操作のカテゴリ化されたレポートを生成します。これらのレポートは、セキュリティアナリストがコードベース全体を読む前に調査に値するプロシージャを特定するのを助けます。これらは Visual Basic 5.0/6.0 アプリケーションをサポートし、11.7 バージョン以降では .NET アセンブリもサポートします。

## レポートの生成

1 **ツール** > **オプション** (**Tools** > **Options**) で **プロトタイプの分析** (**Analyze prototypes**) を有効にします。

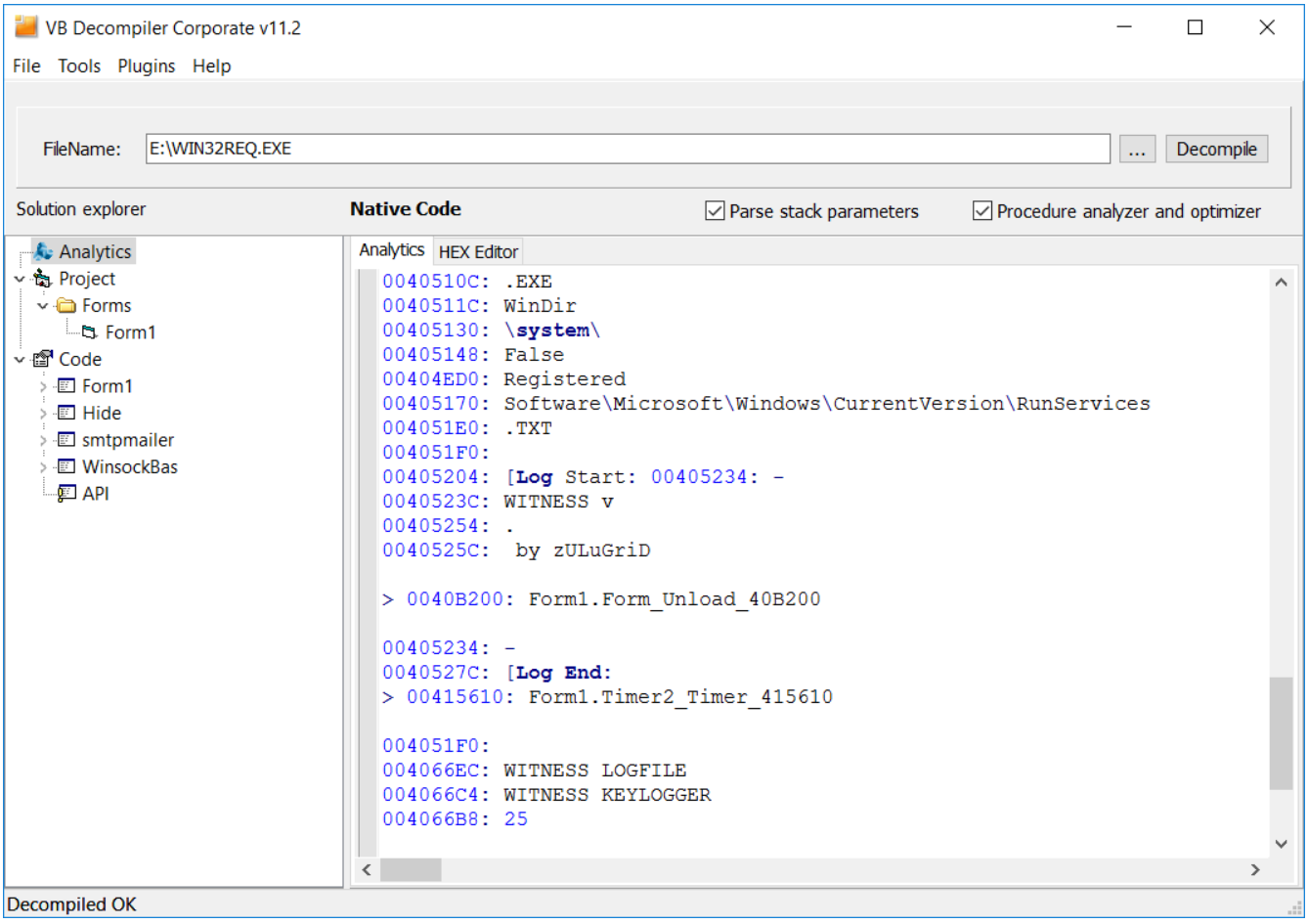
2 ファイルをオープンし、解析が完了するまで待ちます。必要なライセンスと設定がある場合、レポートは自動的に生成され表示されます。

3 カテゴリをレビューし、関連するプロシージャを開いてコードと文字列参照を検査します。

カテゴリは、ファイルおよびボリュームへのアクセス、レジストリへのアクセス、ネットワークアクティビティ、プロセスの作成、ウィンドウ操作、および `CallByName` および `VarPtr` などの選択された VB 内部機能など、認識された操作をカバーします。

## クイックレポートと文字列コンテキスト

クイックレポートは、認識された操作を含むプロシージャのアドレスと名前をリストします。拡張レポートは、それらのプロシージャから利用可能な文字列参照を追加します。このコンテキストは、調査に値する候補パス、レジストリキー、ホスト名、または他の値を開示できます。



暗号化、オブフスケートされた、またはランタイムで生成された文字列は欠落している可能性があります。プロシージャに関連付けられた文字列は有用なコンテキストですが、それが特定の API 呼び出しに渡されることを証明したり、分岐が実行されていることを証明したりするものではありません。

## 結果のナビゲーションと保存

レポート内のプロシージャ名をダブルクリックしてコードを開きます。プロシージャまたは文字列の前にあるアドレスをダブルクリックして、HEX エディタでの位置を検査します。**A** ボタンを使用してオブジェクトツリーで前の位置に戻り、ナビゲーション矢印または **Esc** を使用して検査されたプロシージャを再訪します。

必要に応じてレポートテキストを選択してコピーするか、**ファイル** > **分析レポートの保存** (**File** > **Save analytics report**) を選択してさらに分析のために保存します。

## 発見事項の解釈

レポートは静的に認識されたコードパターンと参照を特定します。これは観察された実行の記録やファイルがマルウェアであるという自動的な判断ではなく、初期調査を支援するツールです。間接呼び出し、未解決の外部インターフェース、破損したメタデータ、オブフスケート、およびランタイムで生成された値はカバレッジを制限できます。重要な発見事項は対応するプロシージャ、逆アセンブル、およびファイルデータを確認してください。

VB5/6 の再構築のより深い検査については トレース を参照してください。リテラル文字列とそのコード位置については 文字列参照 を参照してください。

# 13 難読化の制限

**VB Decompiler** の難読化機能は、**Visual Basic 5.0 および 6.0 アプリケーション** の逆コンパイルに関する研究に基づいて開発されており、コンパイラが実行ファイルに残す情報を効果的に隠蔽します。現代のプログラミング言語は、コンパイルされたファイル内に大量のメタデータやデバッグ情報を格納することが多く、プログラムを解析する第三者が目的の関数を見つけやすくなります。この脅威に対抗するため、**Visual Basic** アプリケーション用に特別に設計された難読化ツールを開発しました。

## 難読化処理の詳細

難読化ツールは、プログラム内のフォーム、モジュール、クラス、およびボタンやテキストボックスなどのコントロールを解析します。各フォームについて、個々のコントロールとそれに関連するイベントを調べます。その後、関数とプロシージャに関する情報、およびプログラム内の各オブジェクトに関連付けられたイベントが収集されます。このプロセス中、難読化ツールはオブジェクト間の依存関係をチェックし、各オブジェクトの名前を整合性を保ちながら変更し、関数がどのモジュールやクラスに属するかを特定しにくくします。

その結果、プログラム内の特定のフォーム（例：**frmTrial** または **frmRegistration**）を見つけることが大幅に困難になります。

## Visual Basic 5.0/6.0 アプリケーションの難読化の制限

難読化ツールはオブジェクト間の依存関係を解析しますが、すべてのパターンに対応しているわけではありません。そのため、難読化の前にプログラムを修正する必要がある場合があります。以下の制限は、Visual Basic 5.0 / 6.0 アプリケーションの難読化にのみ適用されます。

1. 難読化ツールは、設計時に設定された他のオブジェクトへの参照をサポートしていません。このような参照は、難読化後に実行時エラーを引き起こす可能性があります。

例えば、**TreeView** コントロールが関連する **ImageList** に依存する場合を考えます。**ImageList** オブジェクトの名前が **TreeView** のプロパティにハードコーディングされている場合、難読化後も元の名前による参照が残るため、実行時にエラーが発生します。この制限は、**DataSets** などのデータベース関連コンポーネントにも適用されます。

```
Begin ComctlLib.TreeView TreeView1
  Height          =   1932
  Left            =   3600
  TabIndex        =   3
  Top             =   1320
  Width           =   2412
  _ExtentX        =   4255
  _ExtentY        =   3408
  _Version        =   327682
  Style           =   7
  ImageList       =   "ImageList1"
  Appearance      =   1
End
```

この問題は、設計時に設定している参照を **Form_Load** イベント内で実行時に代入するよう変更することで解決できます。

```
Private Sub Form_Load()
  TreeView1.ImageList = ImageList1
End Sub
```

該当するすべての参照に同じ修正が必要です。通常、ほとんどのプロジェクトで修正が必要な箇所は数か所です。

2. As New によるオブジェクト宣言（LateCall 呼び出し）はサポートされていません。

この制限は通常、ほとんどの VB6 プロジェクトには影響しません。ただし、フォームをクラスとして扱い、次のようにインスタンスを作成する場合には注意が必要です。

```
Private Sub Form_Load()
  Dim oForm as New frmMain
  oForm.Show
End Sub
```

このような場合、VB6 コンパイラは **Form_Load** 関数のコードに直接名前 **frmMain** を挿入します。難読化ツールはデータのみを変更しコードを変更しないため、コードに埋め込まれた名前は難読化後も変更されずに残ります。残念ながら、これによりプログラムがエラーで終了してしまいます。

**VB Decompiler** の難読化ツールでこのような呼び出しに対応するには、すべての関数のコードを逆アセンブルおよび逆コンパイルする必要があります。逆コンパイルには時間がかかり、大規模なプロジェクトではコード全体の逆コンパイルだけで30 分以上かかることもあります。さらに、オブジェクトへの参照の解析と名前の変更にも処理時間が必要です。

これらの技術的制約から、**VB Decompiler** の難読化ツールに完全な逆コンパイルとコード解析は実装しないことにしました。この処理が必要になるのは Visual Basic 5.0 / 6.0 アプリケーションのごく一部であり、ほとんどのユーザーにとって利点がないためです。

# 14 テクニカルサポート

プログラムに関するご質問があり、このヘルプガイドで回答が見つからない場合は、テクニカルサポートチームまでご連絡ください。このオプションは登録ユーザーのみが利用可能です。未登録ユーザーについては、**VB Decompiler** のライセンス購入に関するお問い合わせにも喜んで対応いたします。

さらに、プログラム内でバグを発見した場合や、改善への提案がある場合は、テクニカルサポートサービスへフィードバックをお送りください。

# 15 登録

**VB Decompiler** の完全な機能を利用するには、**Business License** を購入する必要があります。Single Developer 用ビジネスライセンス または エンタープライズライセンス を購入することで、以下の機能を取得できます：

**1.** 一般的なサポート対象の **Visual Basic 5.0/6.0 P-Code** アプリケーションで、コードの約 85% を復元する;

**2.** 一般的なサポート対象の **Visual Basic 5.0/6.0 Native Code** アプリケーションで、コードの最大 85% を復元する;
**3.** 一般的なサポート対象で難読化されていない **.NET** アセンブリを **C#** に近いコードへ逆コンパイルし、最大 95% を復元する;
**4.** 常にプログラムの最新バージョンを保有：ライセンス購入から 1 年間は更新が完全に無料です；
**5.** ライセンス購入から 1 年間はメールによる技術サポートを受ける；
**6.** この逆コンパイラの開発を支援する。

さらに、マルウェア分析家およびフォレンジック向けの特別ライセンスの購入を検討することをお勧めします：

**1.** トレース機能 を備えたライセンス
**2.** 分析機能 を備えたライセンス

逆コンパイラの **ライセンス** を今すぐ購入するには、プログラム内の **ヘルプ** (**Help**) メニューで **Pro版を購入** (**Order Pro version**) オプションを選択してください。この時点ではインターネットに接続されている必要があります。ブラウザウィンドウが開き、プログラムのウェブサイトが表示され、そこでライセンスを購入できます。

逆コンパイラの **Business License** は、1 台のコンピューターのみで、かつ **Single Developer**（ライセンス所有者）のみで使用することを目的としています。複数のコンピューターでプログラムを使用する必要がある場合は、各コンピューターごとに別々のライセンスを購入するか、**Business Enterprise license** の購入を検討してください。（最大 20 台のコンピューターで使用可能）。

# 16 開発者について

この逆コンパイラは **DotFix Software** によって開発されています。

製品のウェブサイト: www.vb-decompiler.org
テクニカルサポートのメールアドレス: admin@vb-decompiler.org