DE Decompiler

ユーザーマニュアル

日本語

Delphi 3-13 / C++BuilderのVCLプログラム

DotFix Software

目次

1 はじめに 4

1.1 イントロダクション 4

1.2 クイックスタート 6

1.3 コンパイラの互換性とRTTI 7

1.4 メインウィンドウ 8

1.5 ユニット、クラス、および関数 11

2 開くおよび保存 13

2.1 ファイルを開く 13

2.2 解析データベース 14

2.3 復元されたデータの保存 15

2.4 プロシージャリストの保存 16

3 アプリケーションの調査 17

3.1 コードの表示と比較 17

3.2 アドレスおよびツリーパスによるナビゲーション 20

3.3 プロシージャとテキストの検索 21

3.4 文字列参照 22

3.5 アドレスからの逆コンパイル 23

3.6 バイトの検査とパッチ適用 24

3.7 VCLフォームとビジュアルインスペクション 25

4 オプションとツール 27

4.1 プログラムオプション 27

4.2 逆コンパイル処理 28

4.3 逆アセンブリ表示 30

4.4 テーマとフォント 32

4.5 処理モードとキャッシュ 34

4.6 インターフェース言語 35

4.7 AI ヘルパー 36

4.8 プラグインと SDK 39

4.9 難読化 42

5 リファレンス 43

5.1 復元されたコードの解釈 43

5.2 テクニカルサポート 45

5.3 ライセンス 46

5.4 開発者について 47

1 はじめに

1.1 イントロダクション

Delphiで記述された、またはC++Builderで作成されたVCLプログラム用の専門家向け逆コンパイラ。リバースエンジニアリングやフォレンジックに向け、Win32/Win64のコード、フォーム、ユニット、クラスを解析します。

Delphi 3〜13で記述された、またはC++Builderで作成されたWin32/Win64向けVCLアプリケーションを、Delphi風のPascalに逆コンパイルします。機械語命令と利用可能なメタデータから制御フロー、呼び出し、フォーム、ユニット、クラスを復元し、VCLのイベントから実装へ移動できます。

復元された構造とコードを使用して、プログラムの動作、マルウェア、および潜在的に望ましくないまたは隠蔽された機能を調査してください。VCL イベントをハンドラまで追跡し、制御フローや呼び出しを調べ、逆アセンブリおよび HEX エディタを使用して詳細な検査を行ってください。

ネイティブコードの逆コンパイル

高水準コードリストは、`if` 条件、`case` ブロック、ループ、および認識された VCL メソッドへの呼び出しを含むプログラムロジックを再構築します。認識されたユニット、クラス、および公開関数は構造的なビューを提供し、ヒューリスティックな検出は各ユニット内のプライベート関数も検索します。

フォームとツール

読み取り専用のデザイナーで復元したVCLコントロールとプロパティを調べ、イベントからハンドラへ移動できます。ユニットとフォームはPascal/DFMプロジェクトまたは単一の`.pas`コードリストとして出力します。内蔵の難読化ツールは、DelphiおよびC++BuilderのWin32/Win64向けVCLアプリケーションに対応します。

バイナリに保持されている情報

出力は、元のソーステキストそのものではなく、Delphi風のPascalコードです。ネイティブコンパイルでは、ソースレベルの情報がすべて保持されるわけではありません。ローカル変数名は通常、自動生成されます。データ構造のレイアウトはメモリアクセスのサイズやアドレスから推定される場合があり、識別できない呼び出しは元のVCLメソッド名なしで表示されることがあります。コンパイラごとのRTTIに関する知識を利用して、残存する名前や型を復元します。

高水準コードリストは、その証拠によってサポートされる詳細さでネイティブプログラムロジックを再構築します。推定された型、予期しない式、および重要な結論を対応する命令に対して確認してください。復元されたコードの解釈 を参照してください。

失われたソースの復元を支援する用途にも利用できます。ただし、出力はそのままコンパイルできるソースプロジェクトではありません。復元したコードをIDEで開く前に、逆コンパイルの不正確な箇所や互換性の問題を手動で解析・修正し、不足している依存関係を補う必要があります。

C++Builderで作成されたアプリケーション

DE Decompilerは、C++Builderで作成されたWin32/Win64向けVCLアプリケーションに対応します。フォームの復元と表示、ユニット・クラス・関数の解析、イベントハンドラへの移動に加え、同じワークスペースでその他の解析・出力ツールも利用できます。

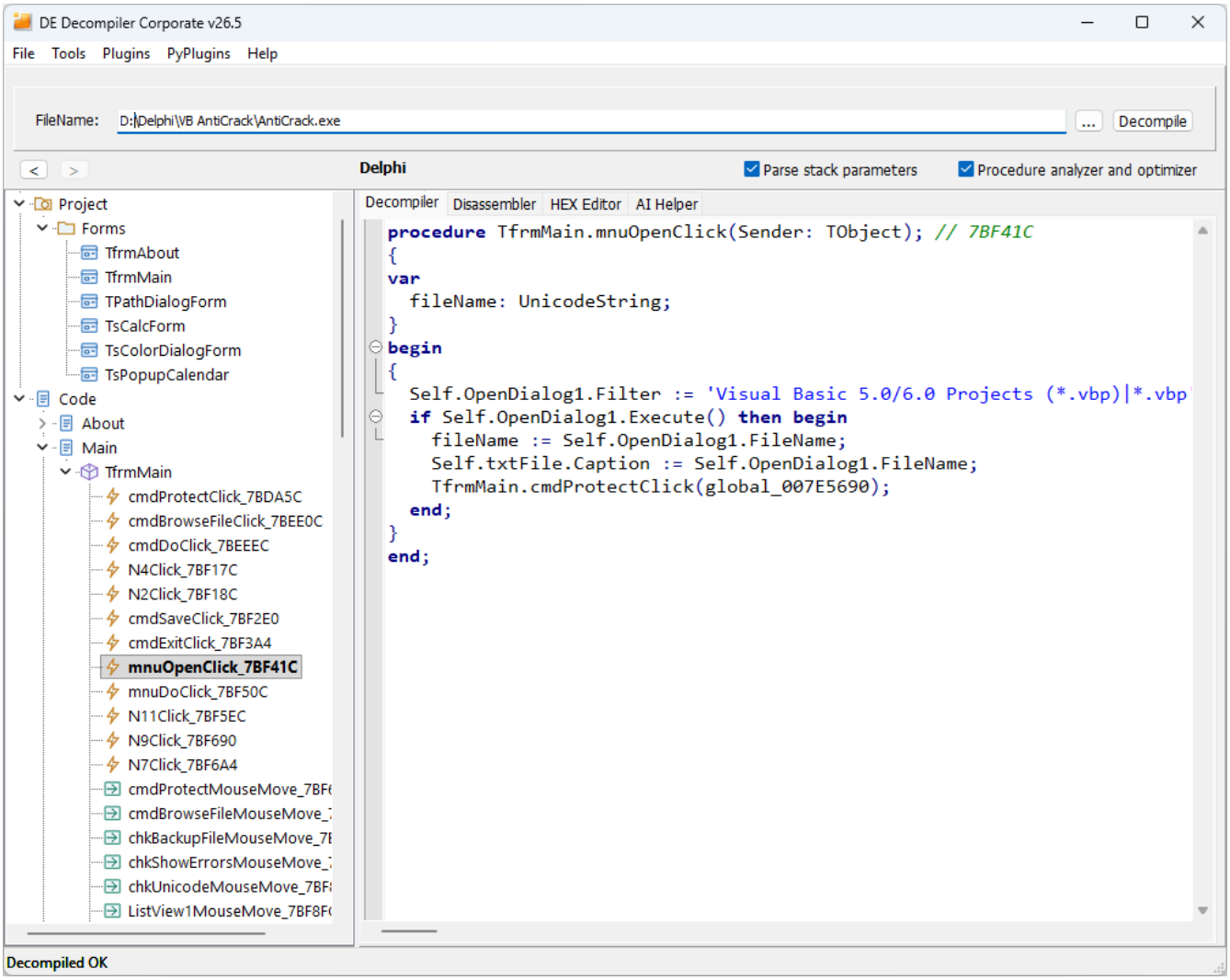
C++Builder固有のライブラリに対応するRTTI知識ベースは、まだ用意されていません。共通のVCLの認識範囲に含まれない関数や、識別できないオブジェクトのプロパティ・メソッドは、ライブラリ内の名前を付け

ずに表示される場合があります。このため、C++Builderのコードの復元品質は、Delphiで記述されたプログラムより明らかに低くなります。 この制約はコードの復元精度とライブラリ内の名前の識別に関するものです。フォーム表示や構造解析など、その他の機能はDelphiで記述されたプログラムと同じエディションの条件で利用できます。

C++BuilderのコードはDelphi風のPascalとして復元し、PAS/DFM形式で出力します。元のC++ソースには戻しません。他のネイティブコード復元と同様、IDEで利用する前に、推定された型、生成された名前、逆コンパイルの不正確な箇所を手作業で確認・修正してください。

エディション

高水準コード逆コンパイル、AI ヘルパー、HEX エディタ、および難読化には、Single Developer または Enterprise ライセンスが必要です。無料の Lite バージョンは VCL フォーム情報を復元し、ネイティブ逆アセンブルと文字列参照を提供します。ライセンス を参照してください。



復元したプロジェクト構造付きの逆コンパイルされたDelphi風のPascalコード。

1.2 クイックスタート

対応するDelphiまたはC++BuilderのWin32/Win64向けVCLアプリケーションを開きます。復元したプロジェクトの構造を調べ、コードと機械語命令を照合してください。高水準コードの逆コンパイル、AI Helper、HEXエディターには商用ライセンスが必要です。Free Liteではフォーム表示、逆アセンブル、文字列参照を利用できます。

1 **ファイル** > **プログラムを開く** (**File** > **Open program**) を選択し、解析対象のバイナリを選択します。DE Decompiler が初期処理を完了するまで待ちます。

2 プロジェクトツリーで**コード**を展開します。逆コンパイル結果で宣言された復元済み宣言を確認するにはユニットを選択し、クラスまたはプロシージャを選択します。

3 **逆コンパイラ** (**Decompiler**)と**逆アセンブラ** (**Disassembler**)を比較します。アドレスまたは命令をファイルのバイト列に対して確認する必要がある場合は、**バイナリエディタ** (**HEX Editor**)を使用してください。

4 **フォーム** (**Forms**)を展開し、復元されたフォームを選択して**デザイナー** (**読み取り専用**) (**Designer** (**read only**))を開きます。そのコントロールとプロパティを検査し、復元されたイベントエントリを使用して対応するハンドラコードを開きます。

5 プロシージャを検索するには**Ctrl**+**T**を使用し、表示されているテキストを検索するには**Ctrl**+**F**を使用し、認識された文字列リテラルを調査するには**ツール** > **文字列の参照** (**Tools** > **String references**)を使用します。

6 後で解析に戻す必要がある場合は、**ファイル** > **データベース** (**File** > **Data Base**)を通じて解析データベースを保存してください。

復元されたデータの保存

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**)を使用して復元されたDelphiプロジェクトをエクスポートするか、結合された.pasリストを取得するには**ファイル** > **1つのモジュールにすべて保存** (**File** > **Save all in one module**)を使用します。IDEで復元されたコードを開く前に、逆コンパイルの不正確さや互換性の問題を手動で修正し、欠落している依存関係を提供してください。調査資料とともに、解析対象のバイナリファイルの改変されていないコピーを保管してください。

処理モードの選択

プロシージャを開くと**高速逆コンパイル** (**Fast decompilation**)は個々のプロシージャを処理します。プロシージャのシグネチャの予備的な解析には、**プロトタイプの分析** (**Analyze prototypes**)を有効にしてください。すべての高レベルテキストを検索する前に、**高速逆コンパイル** (**Fast decompilation**)を無効化してファイルを再度処理してください。プロジェクトおよび結合リストのエクスポートは必要に応じてプロシージャを処理し、完了メッセージを確認します。処理モードとキャッシュを参照してください。

有用な最初の確認手順」を参照してください。

既知のフォームと、それに関連するイベントメソッドまたは公開関数のいずれかから開始します。復元した宣言を確認し、コードを追跡して、いくつかの文を逆アセンブル結果と比較します。これにより、DE Decompilerがその特定のビルドでどの程度の有用な構造を認識しているかが把握できます。

1.3 コンパイラの互換性とRTTI

DE DecompilerはDelphi 3〜13をネイティブに識別し、C++Builderで作成されたWin32/Win64向けVCLアプリケーションにも対応します。

コンパイラバージョンと復元された型

Delphiのリリースによって、実行ファイルの構造やRTTI（実行時型情報）の形式が異なります。DE Decompilerはコンパイラのバージョンに対応する型情報の知識を使って解析します。コンパイラを識別できても、すべての型、ライブラリ、プロシージャを同じ精度で復元できるとは限りません。

認識されたクラスをその宣言、フォームのプロパティ、およびメソッドコードと比較してください。名前、フィールド型、署名、またはレイアウトに矛盾がある場合は、高レベル表現を信頼する前に、ネイティブな命令とファイルバイトを使用して調査を行ってください。

コンパイラ識別とシグネチャデータベース

C++BuilderとFree PascalをDelphiとは区別して識別します。互換シグネチャデータベースの使用時や保存済みプロジェクトの読み込み後も、入力のコンパイラ情報を保持します。長いコンパイラ名やバージョン範囲はメイン画面のラベル内に収まり、詳細はツールチップで確認できます。

入力ファイルのコンパイラは、コード解析に選択したシグネチャデータベースとは独立して識別されます。得られた情報だけでバージョンを区別できない場合は、表示にもその曖昧さが残ります。

商用エディションでは、deディレクトリからシグネチャデータベースを選択します。まずコンパイラのバージョンが完全に一致するものを探し、なければ同じビット数で最も近い新しいバージョン、それもなければ最も近い古いバージョンを選択します。ファイル名は`vcl<version>.bin`または`vcl<version> x64.bin`です。

アスタリスク（*）は、代替データベースを使用している場合、またはデータベースの一致を確実に確認できない場合に表示されます。コンパイラ名にポインタを重ねると、選択されたBINファイル名を確認できます。この印はデータベースの対応関係を示すもので、入力ファイルを解析できないことを意味するものではありません。シグネチャデータベースは商用エディションで利用できます。

C++Builderで作成されたアプリケーション

DE Decompilerは、C++Builderで作成されたWin32/Win64向けVCLアプリケーションに対応します。フォームの復元と表示、ユニット・クラス・関数の解析、イベントハンドラへの移動に加え、同じワークスペースでその他の解析・出力ツールも利用できます。

C++Builder固有のライブラリに対応するRTTI知識ベースは、まだ用意されていません。共通のVCLの認識範囲に含まれない関数や、識別できないオブジェクトのプロパティ・メソッドは、ライブラリ内の名前を付けずに表示される場合があります。このため、C++Builderのコードの復元品質は、Delphiで記述されたプログラムより明らかに低くなります。 この制約はコードの復元精度とライブラリ内の名前の識別に関するものです。フォーム表示や構造解析など、その他の機能はDelphiで記述されたプログラムと同じエディションの条件で利用できます。

C++BuilderのコードはDelphi風のPascalとして復元し、PAS/DFM形式で出力します。元のC++ソースには戻しません。他のネイティブコード復元と同様、IDEで利用する前に、推定された型、生成された名前、逆コンパイルの不正確な箇所を手作業で確認・修正してください。

プライベート関数

逆コンパイラは、各ユニット内のプライベート関数をヒューリスティックに特定しようとします。この発見は認識されたパブリックエントリポイント以外のコードを明らかにするのに役立ちますが、すべての関数境界またはユニット関連が復元されたことを意味するものではありません。

VCLの対応範囲

ビジュアルフォームのワークフローは、Win32 および Win64 の入力ファイルから復元された VCL フォームとコントロールに適用されます。

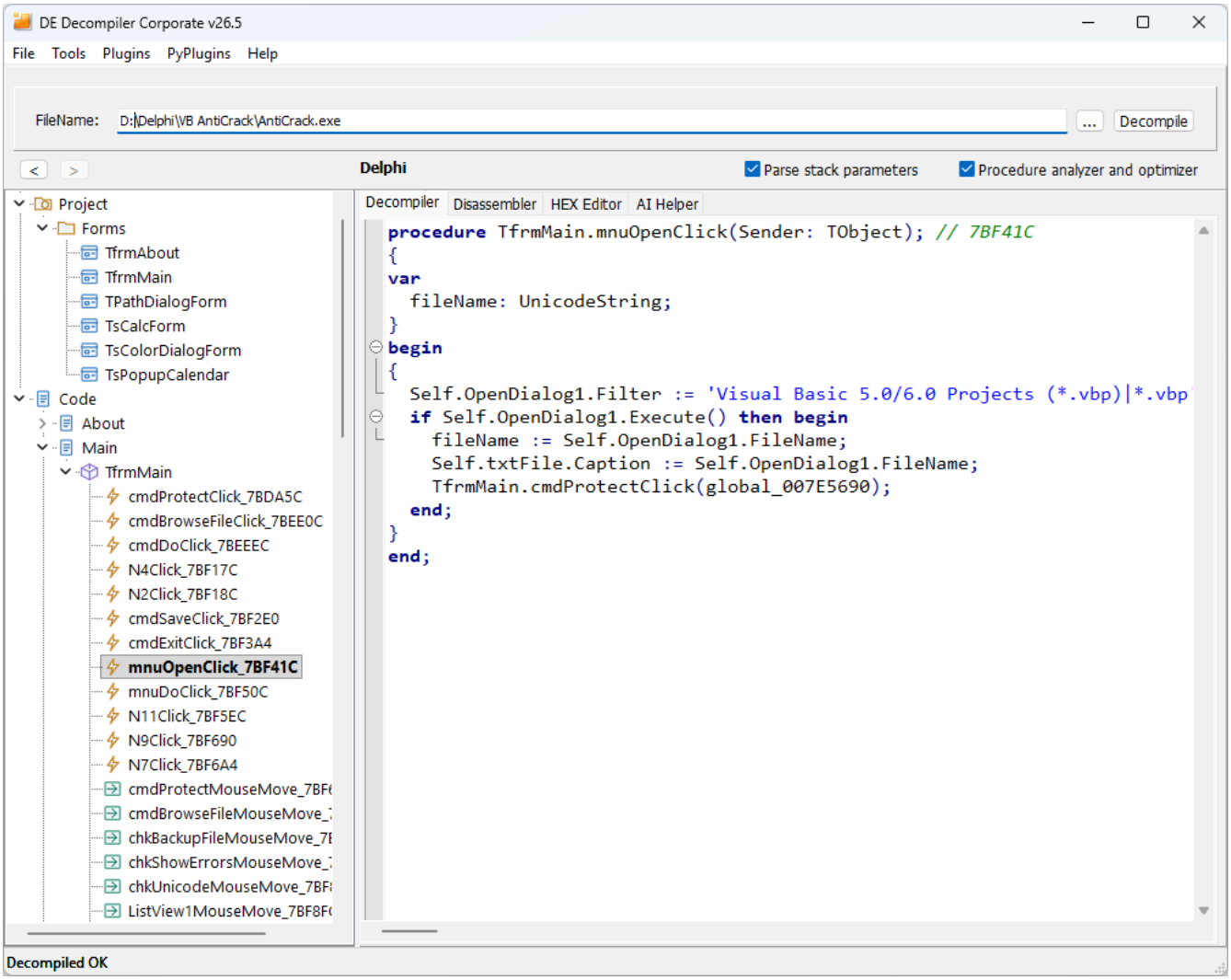
互換性問題の報告

DE Decompiler は継続的に開発されており、コンパイラのサポート、型認識、コード再構築が洗練されるにつれて、逆コンパイルの品質も向上します。

特定の課題について報告したい場合は、問題を再現できるバイナリファイルを送信し、それが発生するフォームまたは関数を特定してください。その名前と、関数の場合は利用可能なアドレスを、DE Decompiler のビルド番号およびエラーの説明と共に含めてください。連絡先詳細については テクニカルサポート を参照してください。

1.4 メインウィンドウ

プロジェクトツリーは、復元されたフォーム、ユニット、クラス、およびプロシージャをコード、宣言、およびバイナリビューと結びつけます。



ライトインターフェーステーマでのDE Decompilerワークスペース。

プロジェクトツリー

フォーム (Forms)には復元されたフォームオブジェクトが含まれています。**コード**は**ユニット** > **クラス** > **プロシージャ**としてグループ化されます。ユニットとクラスは開かれるまで折りたたまれた状態のままになり、1つの可視クラスを持つユニットを展開すると、そのクラスのすべてのプロシージャも表示されます。ユニットを選択すると再構築された宣言が表示され、プロシージャを選択すると対応するコードビューが開きます。

ビューと処理コントロール

- **逆コンパイラ (Decompiler)**は再構築された高レベルステートメントを表示します。
- **逆アセンブラ (Disassembler)**は機械語命令とアドレスを表示します。
- **バイナリエディタ (HEX Editor)**は基盤となるファイルのバイトを表示します。
- **デザイナー (読み取り専用) (Designer (read only))**は復元されたVCLフォームとその利用可能なプロパティを表示します。
- **DFM**は構文強調表示、折りたたみ、検索、コピー機能付きの復元されたDelphiフォームソースを表示します。

- **AI 支援 (AI Helper)**は選択されたモデルまたはAPIサービスを使用して既存の逆コンパイル結果を処理します。

スタックパラメータの解析 (Parse stack parameters)および**プロシージャアナライザとオプティマイザ (Procedure analyzer and optimizer)**は、現在のコードリストの再構築に影響を与えます。出力が不明瞭な場合は、これらの処理オプションを無効にして比較し、対応する逆アセンブリを確認してください。キャッシュされた出力または完全に処理された出力の場合、設定を変更した後にファイルを再度処理してください。

ナビゲーション

ツリー上の矢印ボタンは表示履歴を移動します。**Esc**およびマウスの[戻る]と[進む]ボタンは、コード、宣言、フォームのプレビュー、ソースビュー間で同じ履歴をナビゲートします。フォームに戻ると、選択されたコントロール、ページ、スクロール位置が復元され、コードに戻るとキャレットと表示位置が復元されます。**Alt+1**でツリーとアクティブなビュー間のフォーカスが切り替わり、**Enter**で選択されたツリー項目が開きます。HEXエディタでは、**Tab**で16進フィールドとテキストフィールド間を移動し、その後次のコントロールに移動します。

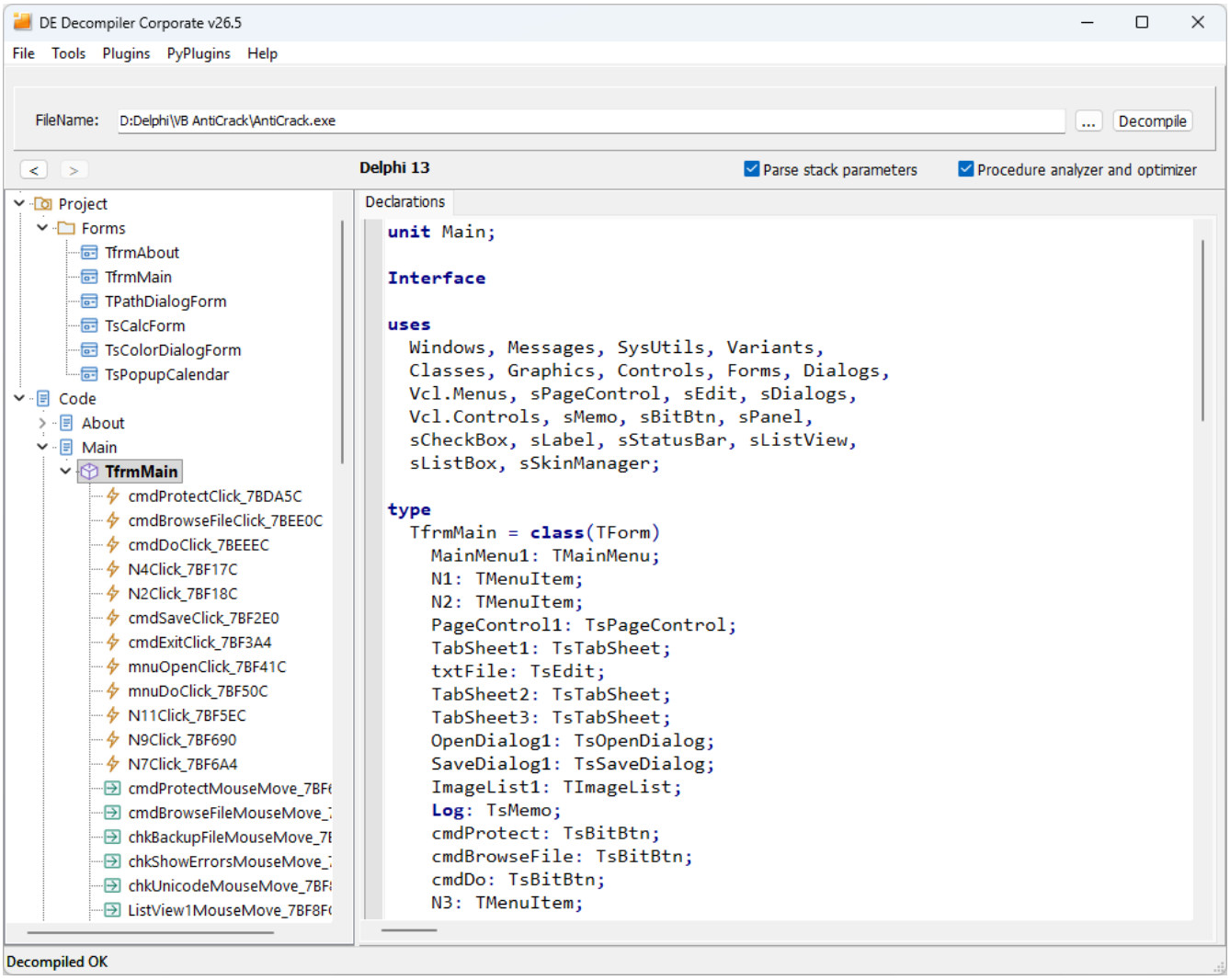
Visual Studioのようなウィンドウスタイル (Window style as in Visual Studio)はコードを左側に、プロジェクトツリーを右側に配置します。利用可能なタブは選択されたオブジェクトによって異なります。コンテキスト固有のショートカットについては、コードの表示と比較を参照してください。

1.5 ユニット、クラス、および関数

復元した宣言を手掛かりに、対応するDelphiおよびC++BuilderのVCLアプリケーションの構造を調べます。

ユニット宣言

Code下のユニットを選択して、その再構築された宣言リストを表示します。ユニット名、利用可能な参照、宣言されたクラス、および可視メンバーを使用して、アプリケーションのどの部分を調査しているかを特定します。



Delphiアプリケーションに対する復元したユニットおよびクラスの宣言。

復元したクラス宣言の一部抜粋：

```
type
  TfrmMain = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    PageControl1: TsPageControl;
```

クラスとメソッド

認識されたクラスを展開し、メソッドまたは関数を選択します。シグネチャと本体を合わせて読み解く必要があります：妥当な本体が得られたとしても、それだけでは推定されたパラメータ型やクラスフィールドの解釈が正しいことを保証するものではありません。

パブリック関数とプライベート関数

認識されたパブリック関数とメソッドは、復元された構造内に表示されます。逆コンパイラはまた、各ユニット内のプライベート関数の特定も試みます。プライベート関数の検出はヒューリスティックな手法によるものであり、予期せぬ境界線、欠落した引数、または矛盾するユニットの関連付けについては、逆アセンブリ結果を用いて調査してください。

コンパイラによって生成された匿名メソッドフレームは、通常のツリーからは非表示になります。検索結果、アドレスジャンプ、および履歴から必要なフレームを特定できます。**VCLクラスを表示 (Show VCL classes)**は完全なクラスビューを提供します。ユニットまたはクラスの宣言を開くために、そのすべてのメソッドをデコードする必要はありません。

名前とアナリストの注釈

バイナリに保持されている名前と、生成された識別子や解析中に割り当てられた名前を区別してください。再構築された名前またはアナリストによって割り当てられた名前は、プロシージャの動作を説明するのに役立つ場合がありますが、それが元のソースコードの識別子であったことを証明するものではありません。

2 開くおよび保存

2.1 ファイルを開く

対応するDelphiまたはC++BuilderのWin32/Win64向けVCLバイナリを開くには、**ファイル** > **プログラムを開く**（**File** > **Open program**）を選択します。

C++Builderで作成されたVCLアプリケーションも同じ方法で開きます。コンパイラ欄ではC++BuilderとFree PascalをDelphiと区別し、ツールチップでコンパイラとシグネチャデータベースの詳細を確認できます。

1 **ファイル** > **プログラムを開く**（**File** > **Open program**）を選択して入力ファイルを選択します。

2 プロジェクト ツリーが構築され、初期処理が完了するまで待機します。

3 個々のプロシージャに進む前に、復元したコード構造とフォームを確認します。

ドラッグ＆ドロップおよびコマンドラインからの開き方

Explorer からバイナリを DE Decompiler にドラッグ＆ドロップするか、コマンドラインでそのパスを渡すことができます。認識はファイルの内容に依存するため、見慣れないファイル拡張子や拡張子の欠如だけでは解析が妨げられるわけではありません。サポートされている Windows ショートカットは、参照先のファイルに解決されます。

保存された解析データベースも、ドラッグ＆ドロップまたはコマンドラインでパスを渡すことで開くことができます。複数のビルドを扱う際は、パスとファイルのバージョンが明確であることを確認してください。

初期解析のキャンセル

大きなアプリケーションの読み込み、クラス検出、フォーム読み取り、全プロシージャの逆コンパイル時のキャンセルを改善しました。キャンセル要求後は以降のプロシージャ処理を停止します。全体逆コンパイルをキャンセルした場合、不完全な結果は破棄します。

任意のプロトタイプ解析をキャンセルしても、解析済みファイルは開いたままとし、検出済みのプロトタイプ、API、文字列を保持します。未完了の解析レポートは表示せず、中断後はデコード設定を復元します。

認識に制限のある入力

利用可能な命令とメタデータは、復元された名前、型、呼び出しの詳細さを決定します。コンパイラの差異、異常な型情報、保護、およびファイルの損傷は認識に影響を与える可能性があります。予期しない結果については機械語命令ビューで確認してください。コンパイラの互換性とRTTI および 復元されたコードの解釈方法 を参照してください。

再現性のある解析を行うには、正確なバイナリを保持し、使用した設定を記録してください。同じファイル名を持つ後のビルドでは、アドレス、レイアウト、動作が異なる場合があります。

2.2 解析データベース

解析データベースは、復元されたプロジェクト情報を保存し、毎回バイナリから開始することなく調査に戻れるようにします。

ファイル > データベース (**File > Data Base**) を使用して解析データを保存または開きます。圧縮された VDD データベースには、復元された構造体、フォーム、リソース、解析者が割り当てた名前、メソッドのメタデータ、および後で確認するためのキャッシュされたリストが格納されます。初期解析に長時間を要する大規模なアプリケーションの場合、完全に処理された出力を保存することが有用です。

ファイル開封ワークフローを通じて、ウィンドウへのドラッグ、またはコマンドラインでのパス指定により、保存されたデータベースを開くこともできます。調査資料をワークステーション間で転送する際は、データベースと共に元のバイナリを一緒に保管してください。

実行ファイルなしでデータベースを確認する場合

元の exe ファイルが存在しなくても、以前にキャッシュされたネイティブコードは引き続き利用可能です。保存されたフォームリソースや復元された構造体も検査できます。保存前にデコードされなかったプロシージャについては、対応する元のイメージが必要です。DE Decompiler は、欠落または変更された実行ファイルに対してデコードを行いません。

利用可能なキャッシュ素材を確認またはエクスポートし、デコードされていないプロシージャを特定するメッセージを確認してください。ポータブルな調査スナップショットを作成するには、保存前に必要なプロシージャを処理し、可能であればデータベースと共に正確な入力バイナリを保管してください。

保存と互換性

データベースの保存はキャンセルをサポートし、既存の VDD は完全な書き込みが完了してから置き換えられます。個々のプロシージャがデコードできない場合でも、データベースには他の解析データを保持できます。保存後に報告される警告を確認してください。

新しいデータベースには拡張モデルやリソースデータが含まれる場合があります。古いビルドでは互換性サブセットのみが表示される可能性があります。すべての保存されたデータを検査するには、最新の DE Decompiler ビルドを使用してください。

保持すべきもの

- 解析データベースと、それが記述する正確なバイナリ。
- DE Decompiler のビルドおよび関連する処理設定。
- 調査で使用された個別のノート、アナリストが割り当てた名前、およびエクスポートされた資料。

データベースは解析アーティファクトです。復元されたソースファイルとは異なり、アドレスとバイト列の源として元の実行ファイルを置き換えるものではありません。

保存された解析結果の確認

別のワークステーションでデータベースを開いた後は、既知のユニット、メソッド、フォームを確認してください。異なる逆コンパイラビルドからの結果を比較する際は、古いデータベースを保持し、各結果を生成したビルドを記録してください。

2.3 復元されたデータの保存

調査に必要な出力を保存しつつ、元のバイナリおよび解析コンテキストを保持します。

プロジェクトおよびモジュールの出力

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)は、復元されたユニットやフォームなど、復元された素材をDelphiプロジェクトとしてエクスポートします。出力フォルダを選択し、保存されたファイルを確認してください：

- `.pas`：各ユニットごとに1つのUTF-8ソースファイルを作成し、復元されたクラスからの宣言とメソッドを結合します。
- `.dfm`：個別のフォーム定義（複数のフォームが1つのユニットに属する場合でも別々のファイルとして保存）。非ASCII文字リテラルはDelphiの#文字コードを使用し、バイナリプロパティデータは保持されます。
- `Project.dpr`：現在のエクスポート中に正常に書き込まれたユニットとフォームリソースに基づいたプロジェクト参照および復元された起動情報。

プライマリフォームは`{$R *.dfm}`を通じてリンクされ、追加のフォームは明示的なリソースディレクティブを使用します。フォームのみの解析では、不足しているPascalクラスや起動コードを推測することなく、確認可能なDFMファイルのエクスポートを行います。不足または競合するリソースについては、エクスポート診断を確認してください。

1つの結合された`.pas`リストの方が確認や比較に便利な場合は**ファイル > 1つのモジュールにすべて保存 (File > Save all in one module)**を使用します。結合されたリストとマルチファイルのDelphiプロジェクトは異なる目的のために使用されるため、調査に必要な出力を選択してください。

コメント付きプロシージャ本体

既定では、メソッドのシグネチャを残し、復元したローカル宣言と文をPascalのコメント内に出力します。プロシージャ本体は引き続き表示、検索、解析の対象になります。この形式で出力されても、ユニットをそのままコンパイルできるわけではありません。

進捗状況、キャンセルおよび失敗

プロジェクトおよび結合コードリストのエクスポート処理は、必要に応じてプロシージャを処理します。デコードできなかったメソッドに関する警告を確認してください。保存されたキャッシュコードは元の実行ファイルなしでエクスポートできますが、デコードされていないプロシージャには対応するバイナリが必要です。

キャンセル (Cancel)を使用してプロシージャ間で停止します。既存のPAS、DFM、DPRおよび結合リストファイルは、完全な書き込み後にのみ置き換えられます。キャンセルされたマルチファイルプロジェクトのエクスポートでは、キャンセル前に完了したファイルが保持されるため、出力フォルダには完了したファイルと古いファイルの混合が含まれる可能性があります。以前のエクスポートを保持する場合は、別の出力フォルダを使用してください。

IDE 用の素材準備

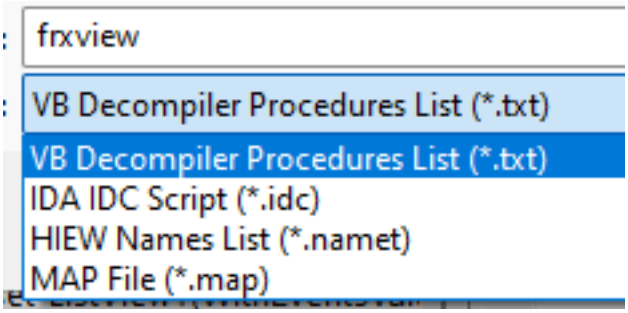
IDE で復元されたコードを開く前に、逆コンパイルの不正確さおよび互換性問題を手動で解決してください。生成された識別子、推測された型およびシグネチャ、未識別の呼び出し、欠落した依存関係を調整してください。コードリストは解析のためにプログラムロジックを再構築しますが、コンパイル可能なソースプロジェクトを再現するものではありません。ビルドには、エクスポートされたコードの手動解析と修正が必要です。

調査記録

セキュリティレビューでは、プロシージャのアドレスおよびバイナリ識別子をエクスポートされたフラグメントとともに保持してください。逆コンパイルされたステートメント、機械語命令、モデル生成の提案、アナリストの結論を、結果として得られるレポートで区別可能に保ってください。

2.4 プロシージャリストの保存

[ファイル] > [プロシージャリストの保存] は、他の解析ワークフローで使用するために認識されたプロシージャ名とアドレスをエクスポートします。



プロシージャリストのエクスポート形式。

保存ダイアログで必要なファイルタイプを選択してください：

形式	「
TXT	レビュー、比較、アドレス/名前の記録用のプレーンテキストのプロシージャリスト。
IDC	認識された関数名とアドレスを転送するための IDA スクリプト。
NAMET	HIEW で使用するための名前リスト。
MAP	互換性のあるデバッガ、逆アセンブラ、シンボルインポートワークフロー用のマップファイル。

すべての形式は完全な Win64 仮想アドレスを保持します。IDC エクスポートは再ベースされた相対アドレスを使用し、ビット数をチェックします。既存の IDA 命令およびデータを保持します。保存はキャンセル可能であり、完全なファイルの置き換えを使用するため、書き込みに失敗した場合でも既存のリストが保持されます。

エクスポートされたリストは認識されたプロシージャを記述します。プライベート関数の検出はヒューリスティックであるため、リストにないことがアドレスに実行可能コードが含まれていないことを示すものではありません。元の入力とそのアドレスコンテキストをエクスポートされたリストとともに保持してください。

3 アプリケーションの調査

3.1 コードの表示と比較

復元したDelphi風のPascalの文でプロシージャの処理を読み取り、逆アセンブル結果でその根拠を確認します。この手順は、対応するDelphiとC++Builderの両方の入力に適用できます。

コードリストの追跡

ツリーからプロシージャを選択します。個々の式を追跡する前に、シグネチャ、ローカル宣言、および制御フローを調べます。生成された名前は、元のソース名が利用できない場合に復元された値を識別するために使用されます。構文強調表示と折りたたみ可能なブロックは、長いプロシージャのナビゲーションに役立ちます。

復元された本体は、エクスポート用に Pascal コメントのラッパーで囲まれる場合があります。DE Decompiler は、それらの本体を識別子の強調表示、参照検索、およびリネームのために依然としてコードとして扱います。文字列リテラルと実際の行コメントは、コード参照の一致から除外されます。

識別子とコードナビゲーション

レジスタ、変数、または関数名にカーソルを置くと、現在のコードリスト内での一致する出現箇所が強調表示されます。あるプロシージャ内のローカルまたはグローバルな識別子を強調表示することは、プロジェクト全体の参照検索とは別個の操作です。

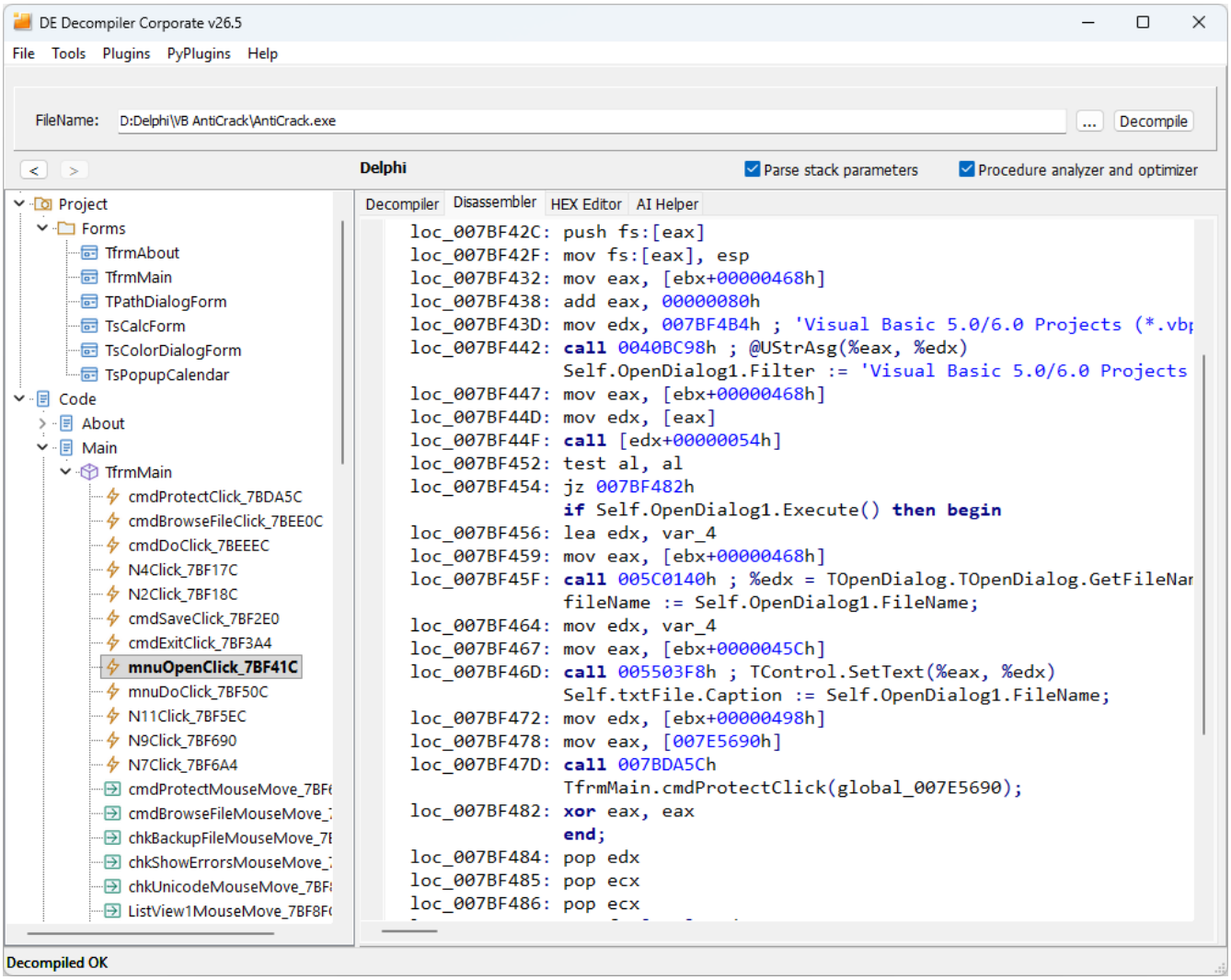
認識されたプロシージャ呼び出しをダブルクリックして、呼び出されたプロシージャを開きます。アドレスのような定数を使用すると、HEX エディタで対応するファイルバイトに移動できます。**Esc** キー、履歴矢印、またはマウスの戻るボタンと進むボタンを使用して戻ります。

アドレスの同期

逆コンパイラと逆アセンブラーの切り替えは、現在のキャレットアドレス（Win64アドレスを含む）に従って行われます。同一の命令ラベルが存在しない場合は、最も近い利用可能なアドレスが選択されます。宛先を含むフォルダのみが開かれます。

ハイブリッドな高レベルおよびネイティブビュー

コード上で右クリックして **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** キーを押します。生成されるビューでは、アドレス付きの再構築されたステートメントが対応する機械語命令の横に表示されます。折りたたまれたソースは元のエディタを展開せずに含まれ、アドレス付けされていない宣言は注釈から省略されます。アドレスベースの比較を使用する際は、**アドレスを追加** (**Add Addresses**) を有効にしてください。



再構築されたDelphi文が機械語命令と交互に配置されます。

参照と名前

ツリーで選択されたプロシージャ上で **参照の検索** (**Find references**) を使用するか、**X** キーを押して、認識された呼び出し元を調べます。結果をダブルクリックして呼び出し元を検査します。参照検索には完全な処理が必要です：必要に応じてプロンプトが表示された場合は **高速逆コンパイル** (**Fast decompilation**) を無効にし、入力を再オープンしてください。デコードされていないキャッシュされたボディに基づく参照リストは不完全です。コードビューでキャレットをグローバル変数に置き、**X** キーを押してその参照を検索します。間接的または動的に解決される使用法は、認識された参照リストに表示されない場合があります。

名前変更 (**Rename variable**) または **N** を使用して意味のある解析名を割り当てます。名前は Pascal 識別子の規則に従います。名前の変更によるマッチングは大文字と小文字を区別せず、パラメータ、ローカル宣言、および予約語との衝突をチェックします。**名前の変更を元に戻す** (**Undo renaming**) または **Ctrl+Shift+Z** を使用すると、通常のエディタの元に戻し操作とは独立して変数の名前変更を元に戻すことができます。解析者が割り当てた名前は、バイナリに保持されている元の名称と区別してください。

キーボードアクセス

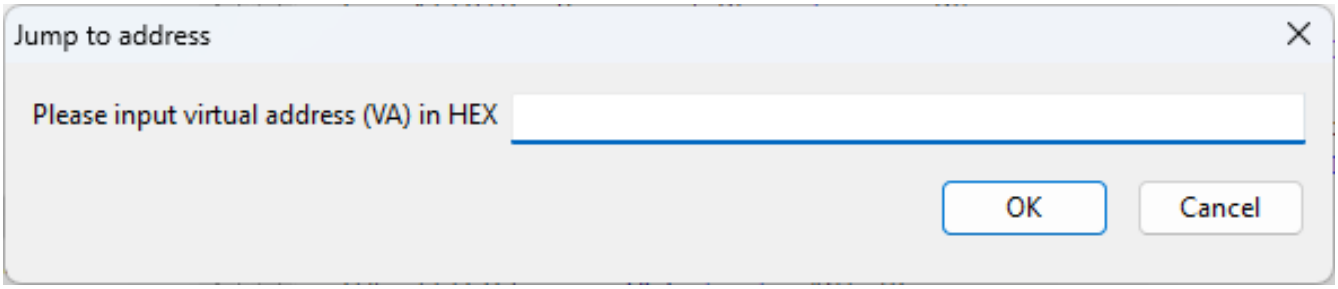
フォーカス	ショートカット	アクション
コード	D	再構築されたコードリストをディスアセンブリと組み合わせる。
コード	N	選択した変数の名前を変更する。
コード	Ctrl+Shift+Z	変数名の変更を元に戻す。
コード	X	カーソル下のグローバル変数の参照を検索。
ツリー	X	選択されたプロシージャの参照を検索。
ツリー	G	セクションパスを使用して移動する。
ツリー	Ctrl+C	選択されたオブジェクトの完全なツリーパスをコピー。
ツリー	Space または Insert	選択された項目をマークまたはアンマーク。
ビュー	Alt+1	ツリーとアクティブなビューの間でフォーカスを切り替える。
履歴	Escキー	直前のプロシージャに戻る。

一文字のショートカットはキーボードのフォーカスに依存します。入力フィールドでの通常のテキスト入力を置き換えるものではありません。

3.2 アドレスおよびツリーパスによるナビゲーション

[ツール] > [仮想アドレスへ移動] を開くと、仮想アドレスによって場所を開きます。

Ctrl+Gを使用するか、**ツール > 仮想アドレスにジャンプ (Tools > Jump to virtual address)**を選択して、必要なアドレスを入力し、確認します。アドレスは現在解析中のビルドを参照します。ファイルオフセットと仮想アドレスは異なる座標系です。リストとファイル内のバイトを比較する際は、この区別を保ってください。



ナビゲーション用の仮想アドレスの入力。

ツリー位置の共有

プロジェクトツリーで、**セクション名をコピー (Copy the name of section)**または**Ctrl+C**を使用して選択したオブジェクトのパスをコピーします。**セクションへジャンプ (Jump to the section)**または**G**を使用して、対応する解析内でコピーされたパスで移動します。

また、**Ctrl+G**を実行する前にリスト内のアドレスを選択することもできます。参照に従った後に戻るには表示履歴を使用します。マウスの［戻る］および［進む］ボタンは、手順内での位置を保持します。他のアナリストと共有されるパスまたはアドレスは、同じバイナリおよび互換性のある復元構造の文脈においてのみ意味を持ちます。

3.3 プロシージャとテキストの検索

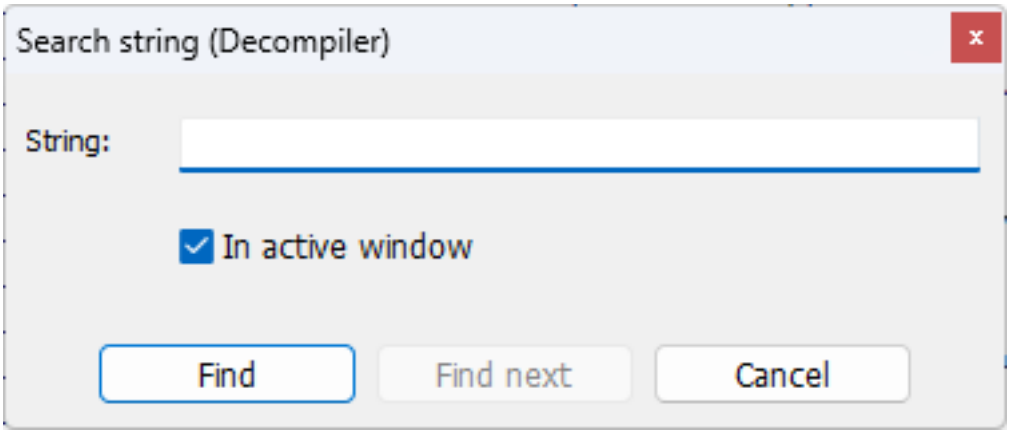
プロシージャ名の検索とコードリストテキストの検索は、異なる目的のために使用されます。

プロシージャを検索

ツール > **検索** (**Tools** > **Find procedure**) または **Ctrl**+**T** を使用して、完全な名前または識別できる一部を入力します。一致は大文字小文字を区別せず、修飾付きプロシージャ名を受け付けます。結果から必要な一致を選択します。

表示されたテキストを検索

アクティブウィンドウ内 (**In active window**) を選択してテキスト検索を開くには、**ツール** > **文字列の検索** または **Ctrl**+**F** を使用します。この場合、検索は現在のコードまたはフォームデータビューに適用されます。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内を検索します。**String** にテキストを入力し、**検索** (**Find**) と **次を検索** (**Find next**) を使用して一致箇所間を移動します。



アクティブなコードビューでのテキスト検索。

プロジェクト全体の上位レベルのテキスト検索には、対応するプロシージャが逆コンパイルされている必要があります。上位レベルのテキスト検索で完全なカバレッジが必要な場合は、**高速逆コンパイル** (**Fast decompilation**) を無効にしてファイルを再度処理してください。

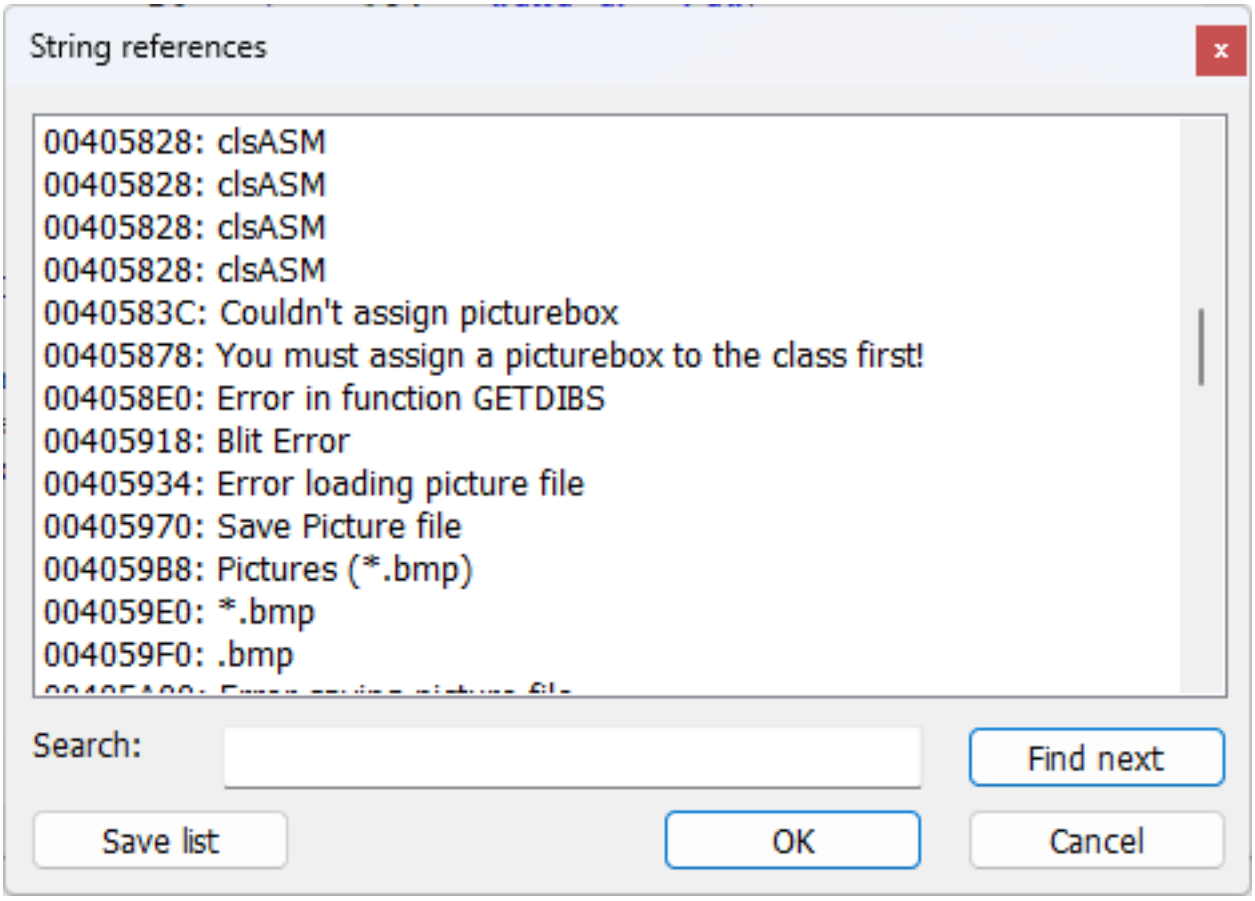
逆コンパイラ、逆アセンブラー、DFMソース、AIビューにおいて、折りたたまれたコンテンツもテキスト検索の対象に含まれます。一致する箇所が見つかった場合、関連しない折りたたみはそのまま閉じた状態が維持されます。

文字列リテラル

認識された文字列リテラルとその場所を調査するには、文字列参照 を使用します。これらの参照は、高速逆コンパイルモードでの予備的な逆アセンブルによって収集できます。すべてのプロシージャに完全な上位レベルのコードリストが必要というわけではありません。

3.4 文字列参照

[ツール] > [文字列参照] は、認識されたリテラルおよびそれに関連するプロシージャの場所を検索するのに役立ちます。文字列参照のブラウジングは、Free Lite および商用エディションで利用可能です。



文字列参照ブラウザ。

興味のあるリテラルを検索するには、**次を検索** (**Find next**) と **検索** を使用します。文字列をダブルクリックすると、それを参照するプロシージャが開きます。再度ダブルクリックすると、その使用箇所をさらに探すことができます。選択した文字列に移動してダイアログを閉じるには **OK** を使用し、ダイアログを閉じるには **キャンセル** (**Cancel**) を使用します。

リストには逆アセンブル中に収集された文字列が含まれています。高速逆コンパイルモードでは、**プロトタイプの分析** (**Analyze prototypes**) が有効な場合、予備的な解析によってこれらの参照が提供されることがあります。DE Decompiler が全体解析が必要であると報告する場合、**高速逆コンパイル** (**Fast decompilation**) を無効にして入力ファイルを再度開きます。Lite では、参照を選択するとネイティブコードの逆アセンブルが開きます。値がどのように使用されているかを判断するために、参照元のプロシージャと対応するバイトを確認します。パス、レジストリ関連の文字列、ネットワークアドレス、コマンド断片は、コードレビューの優先順位付けに役立ちます。

参照は解析の手がかりです。プロシージャ内に文字列が存在することだけでは、それが特定の API に渡されることや、関連する分岐が実行されることを示すものではありません。暗号化されているもの、動的に構築されているもの、またはそれ以外で認識されない文字列は欠落している可能性があります。

認識された文字列の編集

サポートされている文字列を右クリックし、**文字列を編集 (Edit string)** を選択します。ファイルはより大きな値用のスペースを予約していないため、置換元の文字列よりも長くすることはできません。編集を確認すると、ファイルが直接変更されます。この操作ではバックアップが作成されず、文字列編集ダイアログを通じて元に戻すことはできません。コピー上で作業し、変更されていないサンプルを保持してください。

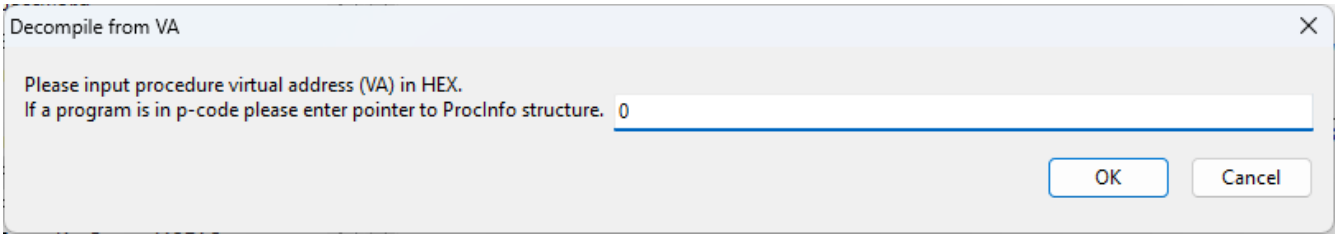
商用エディションでは、コンテキストメニューには文字列のバイトを検査するための **バイナリエディタ (HEX Editor)** も提供されています。HEX Editor は独自の編集に対して元に戻す操作とやり直し操作を提供します。変更後に結果のバイトと影響を受けるコードを確認してください。

3.5 アドレスからの逆コンパイル

[ツール] > [VA から逆コンパイル] は、指定された仮想アドレスからの逆コンパイルを要求します。

このワークフローは、復元されたツリーで期待される手続きとして表現されていないコードを調査する際に有用です。ファイルがすでに開かれている場合は、処理対象の仮想アドレスを入力してください。ファイルが開かれていない場合、DE Decompiler は選択を促し、簡易モードで開きます。

生成された機械語命令と再構築されたコードリストを一緒に確認してください。通常の初期処理が行われない場合、インポートされた呼び出しや型情報が利用できない場合があります。

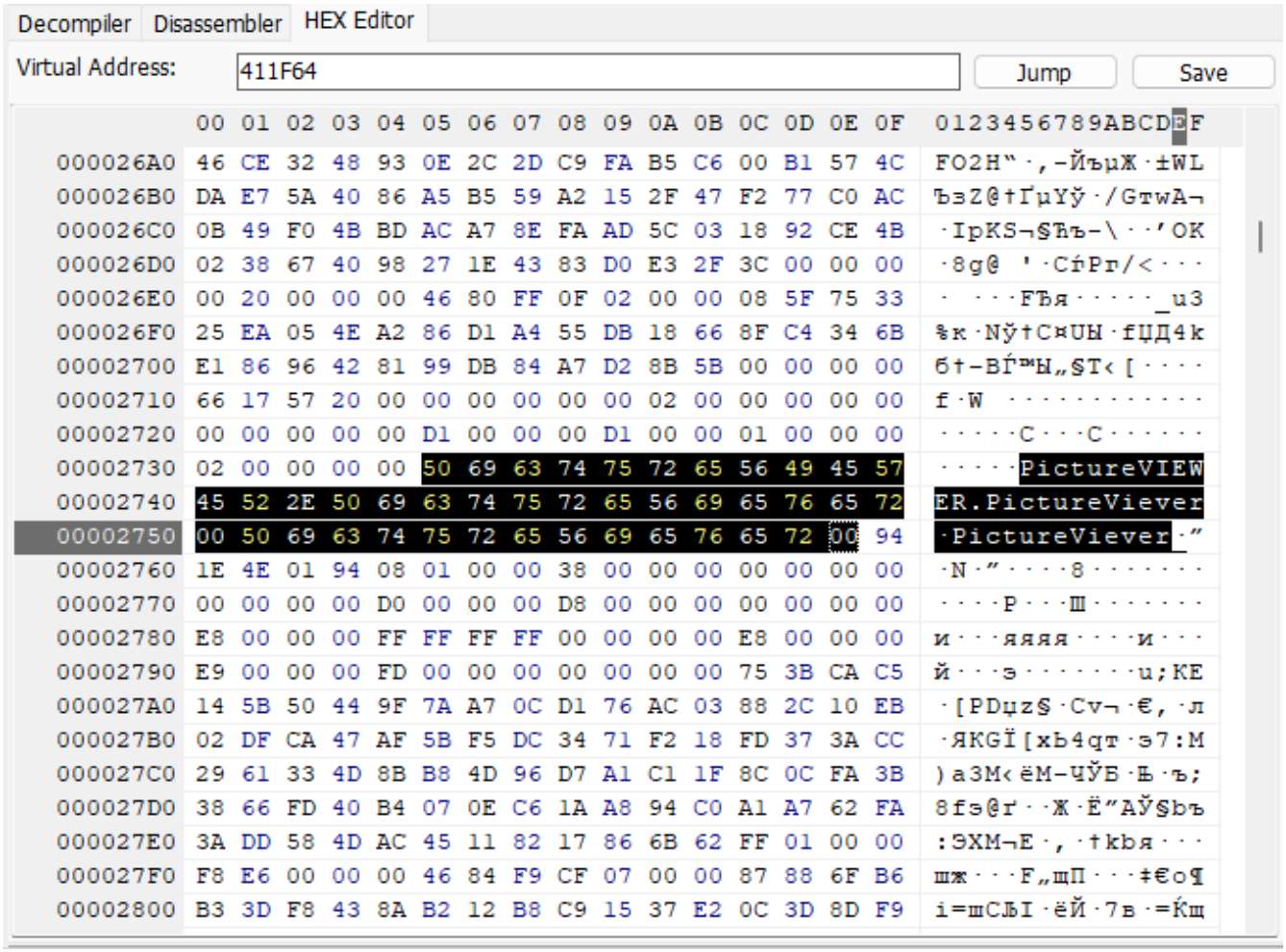


アドレスからの逆コンパイルダイアログ。

アドレスを指定しただけでは、ネイティブ関数の境界やシグネチャは確定しません。開始位置が誤っていると、文や引数を誤って解釈したり、出力が不完全になったりします。復元結果を判断する前に、周囲の命令と認識された呼び出し元を確認してください。

3.6 バイトの検査とパッチ適用

HEX エディタタブを使用して、復元されたコードおよびフォームデータの基盤となるファイルバイトを検査・編集します。



HEX エディタでのバイナリ検査。

ファイルデータの検査

バイナリエディタ (**HEX Editor**) をコードまたはフォームビューと共に開きます。ファイルの先頭から測定したファイルオフセット、**Image Base** を含む仮想アドレスを使用してナビゲートします。表示されたバイトを対応する機械語命令または復元されたフォームデータと比較します。

バイトの編集

HEX エディタは、ファイル内の個々のバイトおよびデータブロックの編集をサポートしています。変更を加える前に、その位置が仮想アドレスとして表現されているかファイルオフセットとして表現されているかを確認してください。

別の解析コピーから作業してください。変更後、必要に応じて影響を受けたコードを再読み込みまたは再処理してください。特にキャッシュされたコードリスト表示が有効な場合に注意してください。動作の比較や調査記録を行う際は、元のバイトと変更履歴の両方を保持してください。

3.7 VCLフォームとビジュアルインスペクション

DE DecompilerはVCLフォーム情報を逆コンパイルし、復元されたコントロールとプロパティを可視化します。

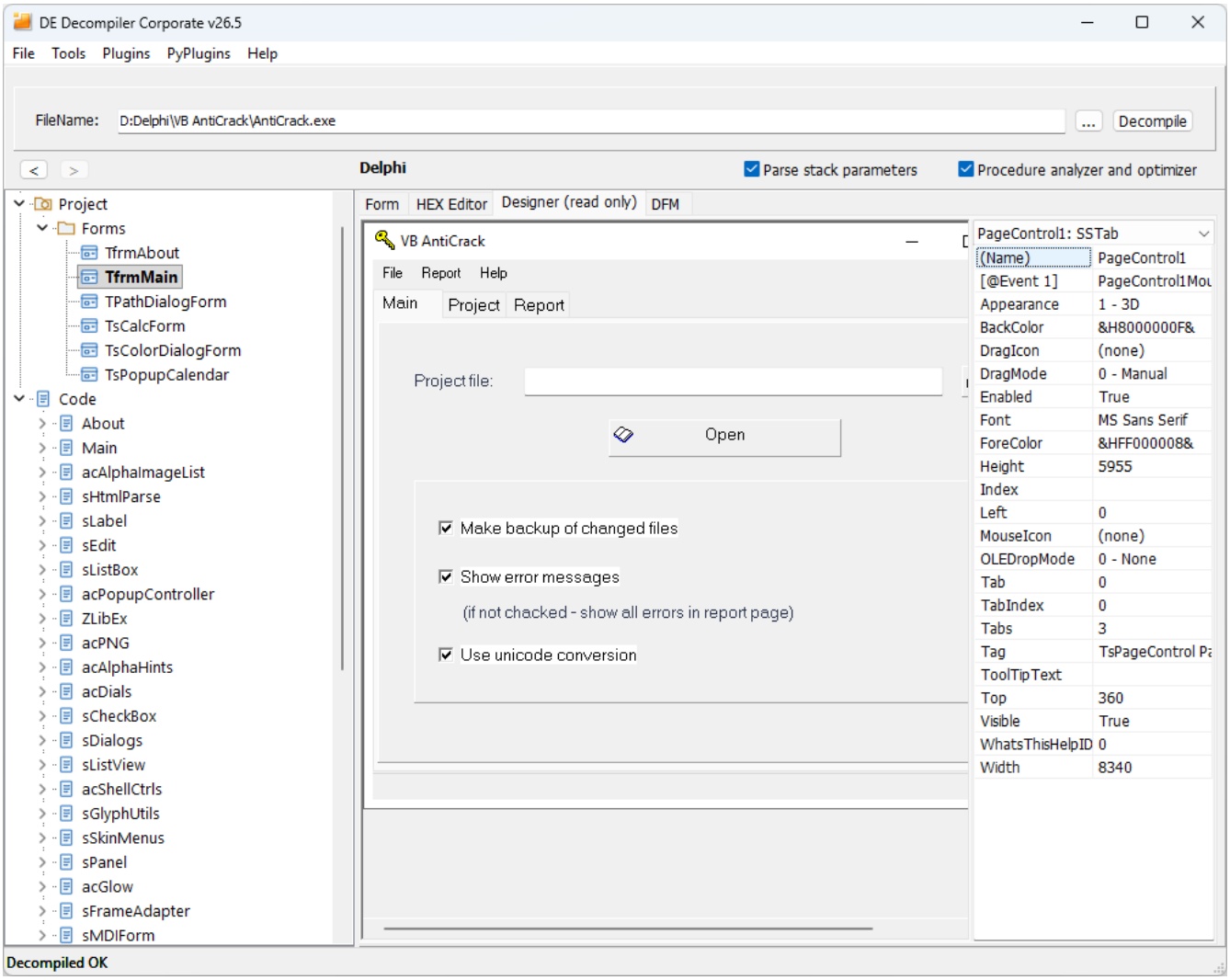
C++Builderで作成されたアプリケーションでも、同じ手順でVCLフォームを調べ、イベントハンドラへ移動できます。

1 プロジェクト・ツリーで**フォーム (Forms)**を展開し、検査するフォームを選択します。

2 **デザイナー (読み取り専用) (Designer (read only))**を開いて、復元されたビジュアルレイアウトを検査します。

3 コントロールを選択し、プロパティエリアで利用可能なプロパティと復元されたイベントエントリを確認します。

4 フォームの表現を、利用可能な**DFM**、フォーム・データ、およびHEXビューと比較します。



再構築されたVCLフォームと、復元されたコンポーネントのプロパティ。

FRX Image Viewerを使用して、登録されたフォーム画像と認識されたネイティブImageListアイコンをプレビューおよびエクスポートします。

DFMソースとビジュアルプレビュー

DFMソース・ビューは、復元されたコンポーネント・クラス名、プロパティ、およびイベント・ハンドラ識別子を保持します。構文強調表示、一致する識別子の強調表示、検索、およびコピーをサポートします。`object`、`inherited`、`inline`ブロック、コレクション項目、バイナリ・プロパティ、および文字列リストを折りたたんで、大きなフォームをナビゲートします。

Formタブには、ビジュアルプレビュー用に変換した内容を表示します。保存されたプロパティ値や元のコンポーネントクラス名は、復元したDFMソースで確認してください。

プレビューは、保存されたページ、メニュー、リスト、グリッドのジオメトリ、ツールバーボタンなどを再構築します。サポートされているローカルImageListのグラフィック、マスク、アルファ透過性も含まれます。サポートされているAlphaSkinsコントロールは、表示用にVCL同等のものにマッピングされますが、DFMクラス名はソースで引き続き利用可能です。解析対象アプリケーションからの描画コードやイベントコードは実行されません。

イベントハンドラをたどる

フォーム表示で復元したコントロールまたはメニューのイベントを選択すると、そのハンドラを開けます。表示履歴で戻ると、選択したコントロール、アクティブなページ、スクロール位置を復元します。イベントの動作を調べる際は、**逆コンパイラ** (**Decompiler**)でプロシージャを確認し、**逆アセンブラ** (**Disassembler**)と照合してください。

フォームデータとオフセット

オフセットを表示 (**Show offsets**)オプションは、フォームデータ表示に位置情報を追加します。復元されたプロパティまたは定義をバイナリ表現に関連付ける場合に使用します。

プレビューの解釈

デザイナーは、復元されたフォームデータの検査用ビューです。元のアプリケーションが初期化コードやイベントコードを実行しているわけではありません。ランタイムで作成されたコントロールや、コードによってのみ割り当てられた値については、対応するプロシージャで調査する必要があります。

カスタムコントロールやバージョン固有の型認識は、フォームの表現方法に影響を与える可能性があります。コンポーネント、プロパティ、レイアウトが不完全に見える場合は、復元されたデータとコードを必ず確認してください。

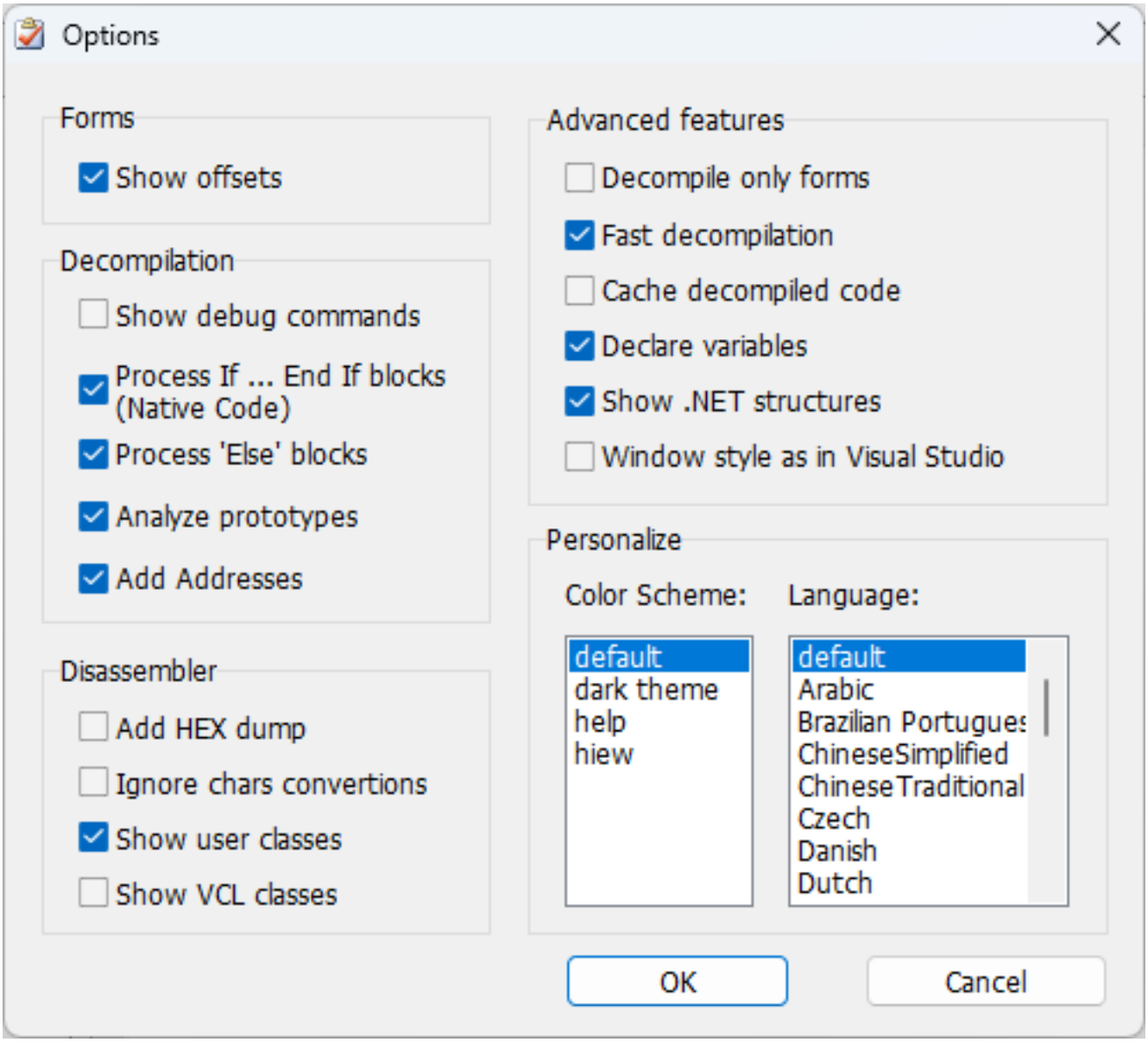
TMSコントロールと静止プレビュー

ビジュアルプレビューでは、認識されたTMSのラベル、ボタン、リスト、グループボックス、スライダー、スクロールバー、タブを、保存された文字列と配置に基づいて表示します。アクティビティ表示は静止プレビューとし、埋め込みWeb領域はページを読み込まずに位置を示します。

4 オプションとツール

4.1 プログラムオプション

[ツール] > [オプション] で、再構築、表示、キャッシュ、およびインターフェース言語を制御します。



ユーザークラスおよび VCL クラスの設定を含む DE Decompiler オプション ダイアログ。

以下では、対応するDelphiおよびC++BuilderのVCLアプリケーションを解析するための設定を説明します。

- フォームと視覚的検査
- 逆コンパイル処理
- 逆アセンブリ表示
- テーマとフォント
- 処理モードとキャッシュ
- インターフェース言語

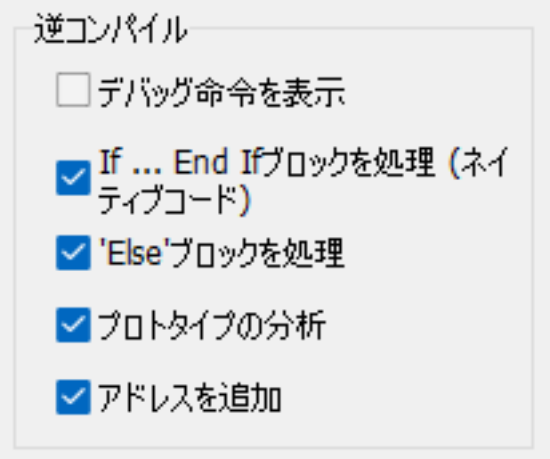
処理設定を比較する際は、入力バイナリを保持し、各コードリストを生成した設定を記録してください。キャッシュされた出力は、変更されたオプションを反映するために再処理が必要になる場合があります。

4.2 逆コンパイル処理

ネイティブコードの逆コンパイルでは、機械語命令と利用可能なメタデータからDelphi風のPascalコードを復元します。同じ処理設定を、対応するDelphiおよびC++Builderのプログラムに適用できます。

再構築されたプログラムロジック

逆コンパイラは条件、`case` ブロック、ループおよび呼び出しを復元し、認識された VCL メソッドも含みます。ソースレベルの名前と型は、バイナリに保持されている情報に依存します。生成された変数名、推測された構造体レイアウト、および認識された VCL 名がない呼び出しは、ネイティブコードの再構築における通常の結果です。詳細については 復元されたコードの解釈 を参照してください。



逆コンパイル処理のコントロール。

Delphi 式とランタイム操作

認識された制御フローには、`if/else`、`case`、`for`、`while` および `repeat/until` が含まれ、ネイティブな分岐がその意味を確立する場所では `Break` および `Continue` が使用されます。

利用可能な RTTI、クラスメタデータおよびシグネチャは、オブジェクトフィールド、プロパティ、仮想メソッド、VCL 呼び出しおよびインポートされた Windows API 呼び出しをサポートします。一般的な RTL 操作は、Delphi の文字列式、`Length`、`Copy`、`Delete`、`Insert`、配列のサイズ変更とインデックス付け、バリアント配列へのアクセス、および `TextFile` 入力操作として表現できます。

これらの変換は、認識された型、呼び出し規約、およびデータフローに依存します。証拠が不十分な場合、コードリストには明示的な呼び出し、ポインタ、または分岐が保持されます。生成された意味的な変数名は推測された役割を記述するものであり、復元された元の識別子ではありません。

スタックパラメータと最適化

スタックパラメータの解析 (Parse stack parameters) は呼び出しと高レベル式の再構築に寄与します。**プロシージャアナライザとオプティマイザ (Procedure analyzer and optimizer)** は認識された式および制御フローパターンを簡素化します。式や分岐が不明確な場合は、これらの制御を無効にして結果のコードリストと比較してください。

制御フロー処理

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)**) は、認識された条件分岐を高レベルの条件ブロックとして再構築します。**'Else'ブロックを処理 (Process 'Else' blocks)** は Else ブランチのヒューリスティックを追加し、If ブロックの処理を必要とします。あいまいな分岐を調査する際に、これらのコントロールの有効化状態と無効化状態を比較してください。

プロトタイプとアドレス

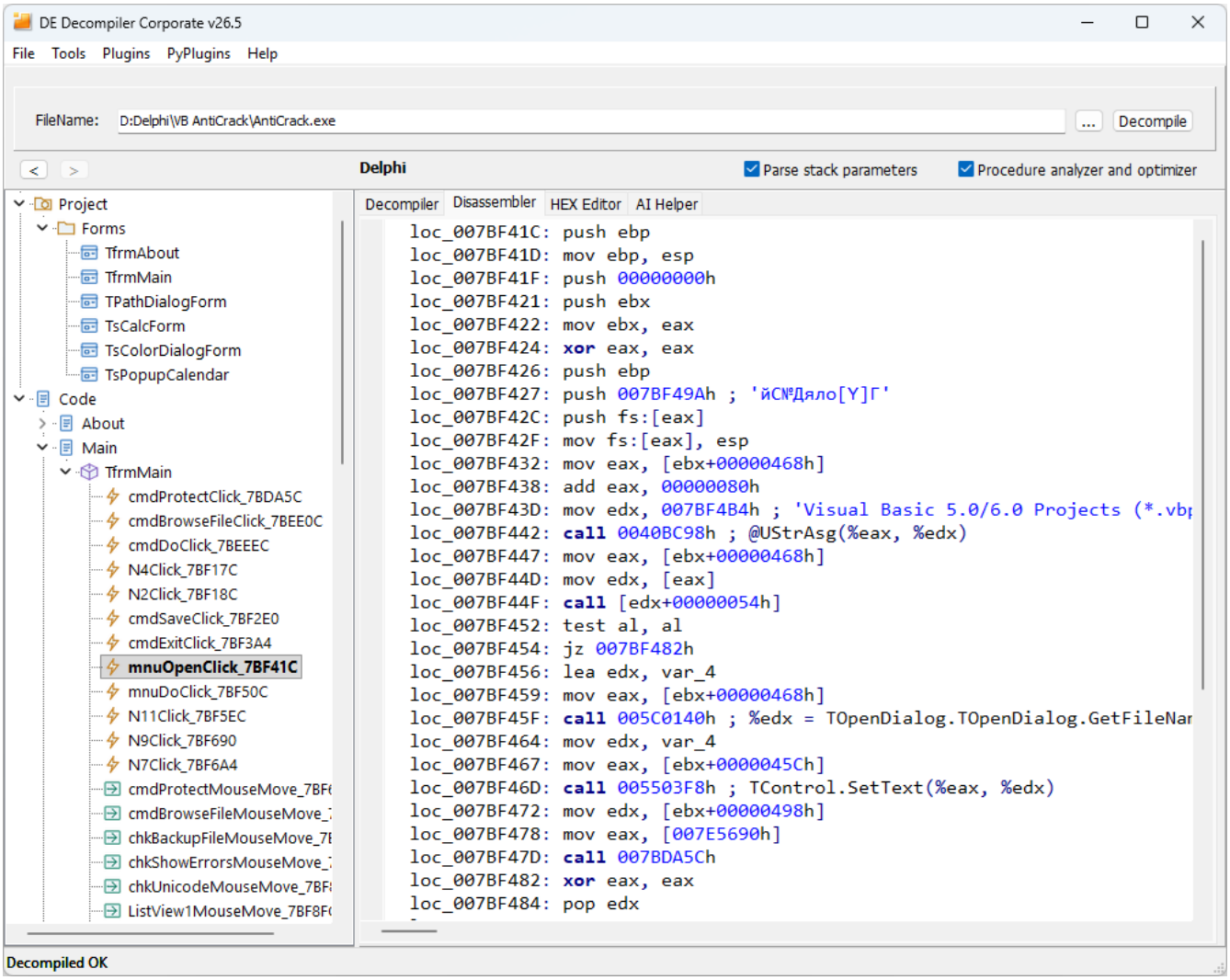
プロトタイプの分析 (Analyze prototypes) は、プロシージャの引数と戻り値の予備的なヒューリスティック解析を実行します。これは、プライベート関数に明示的な署名メタデータがない場合に高速逆コンパイルモードで有用です。この解析には追加の初期処理時間がかかります。解析の結論に影響を与える可能性がある場合、推定された署名を呼び出し元や機械語命令に対して確認してください。

アドレスを追加 (Add Addresses) は、比較およびナビゲーションのために高水準コードリストに仮想アドレスを保持します。**逆アセンブラにコピー (Copy to disassembler)** を有効化している場合はこれを有効のままにしてください。逆アセンブラは、この設定とは独立してアドレスを表示します。

出力がキャッシュされている場合、またはプロジェクト全体がすでに処理済みの場合は、関連する設定を変更した後に再読み込みまたは再処理を行ってください。

4.3 逆アセンブリ表示

逆アセンブル表示では、対応するDelphiおよびC++Builderのプログラムから復元したコードを、機械語命令と照合できます。



復元したDelphiプロシージャの機械語命令。

表示オプション

16進ダンプを追加 (Add HEX dump) は、各命令の前に機械語バイトを表示します。**文字変換を無視 (Ignore chars convertions)** は、文字を数値の Unicode 値に変換するのではなく、非英語文字列を読み可能な状態に保ちます。数値表現が必要な場合はこれを無効化してください。コードアドレスを使用して、コードリストと HEX エディタ間を移動します。

ユーザークラスとVCLクラス

ツール > オプション > 逆アセンブラ (Tools > Options > Disassembler)において、**ユーザークラスを表示 (Show user classes)**はクラス処理で発見されたアプリケーション・クラスを含め、認識されたライブラリ・ユニットと継承メソッドをフィルタリングします。名前がライブラリにも存在するアプリケーション・クラスは、そのユニットの同一性によって区別される場合に保持されます。**VCLクラスを表示 (Show VCL classes)**は内部VCLライブラリ・クラスと、コンパイラ生成のフレームを含む完全なクラスビューも含みます。後者は解析に多くのライブラリ・クラスを追加する可能性があります。アプリケーション自体の実装に集中したい場合は、これを無効のままにしてください。

ライブラリ実装も調査対象とする場合にVCLクラス処理を有効にします。ファイルを開くか再処理する前に、必要なクラスオプションを設定してください。両方のコントロールはプログラム・オプションに表示されます。

解釈の比較

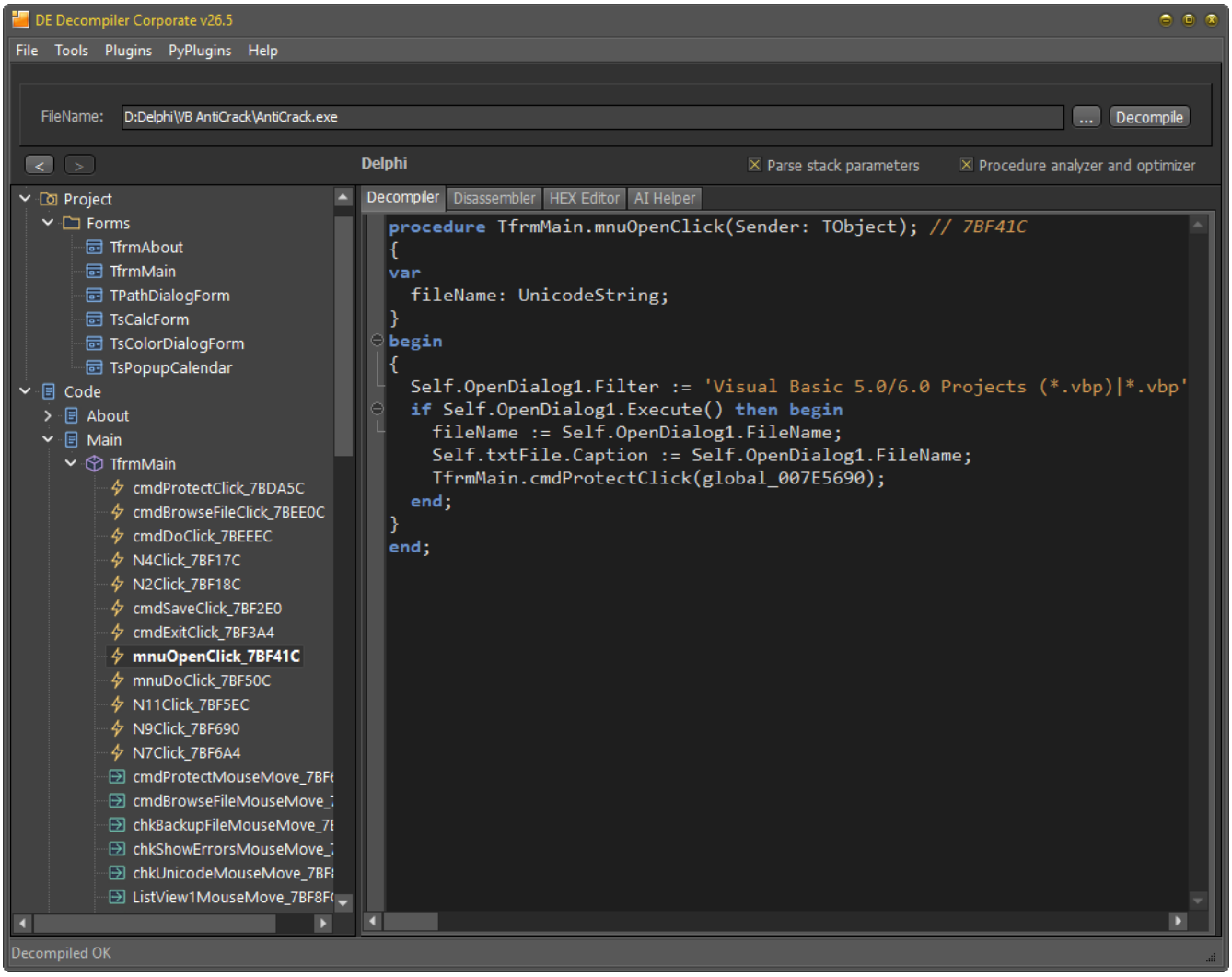
引数の準備、フィールドアクセス、条件分岐、呼び出し、および返された値を機械語命令に対して確認します。可読性の高い高レベルステートメントには、セキュリティ調査において重要な詳細が省略されている場合があります。

逆アセンブラにコピー (Copy to disassembler)のワークフローは、再構築されたステートメントと機械語命令を組み合せます。コードの表示と比較を参照してください。

4.4 テーマとフォント

カラースキームリストは、利用可能なダークテーマを含む、コードおよびバイナリビューの表示を変更します。

ツール > オプション > 個人設定 (**Tools > Options > Personalize**) の下のスキームを選択します。長時間のレビュー セッションのために、前景、背景、および構文の色の一貫して読みやすい組み合わせを選んでください。



ダークインターフェーステーマにおける同じ Delphi コードのワークフロー。

カスタムテーマファイル

プリセットは、インストール ディレクトリ内の `colors` フォルダーにある `.ini` ファイルです。ファイル名がプリセット名を提供します。既存のプリセットをコピーして独自のものを作成し、変更しないキーはそのままにしてください。選択したテーマは、アプリケーションの再起動なしで適用できます。INI ファイルを編集した後でも同様です。

`[Colors]` セクションは 10 進数の色値を使用します。そのキーには `Background`、`Foreground`、`KeyWords`、`LocationsAndArguments`、`Procedures`、`UnderCursorBackground` および `UnderCursorForeground` が含まれます。`UseDarkMode` は、ライト (`0`) またはダーク (`1`) インターフェースを選択します。`Colors2=1` は拡張色セクションを有効にします。

拡張コードビュー設定を使用した抜粋：

```
[Colors]
UseDarkMode=1
Colors2=1

[Colors2]
TVFontName=Segoe UI, Tahoma, Arial
TVFontSize=10
FontName=Consolas, Courier New, monospace
FontSize=11
Background=1E1E1E
Foreground=DFDFE0
Key=CD9069
String=569CD6
HexFontName=Consolas, Courier New, monospace
HexFontSize=11
HexBackground=1E1E1E
HexForeground=DFDFE0
```

フォントおよび構文設定

基本の [Font] セクションには、Name、Size、KeyWordsBold、LocationsAndArgumentsBold および ProceduresBold が含まれます。[Colors2] では、フォント名にコンマ区切りのフォールバックを含めることができます。ツリー フォントには TVFontName と TVFontSize を使用し、コード ビューには FontName と FontSize を使用します。HEX エディターには HexFontName と HexFontSize が使用されます。

拡張色は16進数値を使用します。構文キーには、Key、Comment、Char、Float、Hex、Number、String および Symbol が含まれます。HEX エディタのキーには、HexBackground、HexForeground、HexGrid、HexActiveFieldBackground、HexOffsetBackground、HexCurrentOffset、HexOddColumn、HexEvenColumn および HexOffset が含まれます。カーソル下の色は、コード内の一致する識別子の強調表示を制御します。

4.5 処理モードとキャッシュ

高速逆コンパイル、完全処理、およびキャッシュ出力のサポートにより、さまざまな解析ワークフローに対応します。

フォームのみを逆コンパイル

プロシージャ本体の処理を行わずにVCLフォーム情報を復元する必要がある場合は、**フォームのみを逆コンパイル** (**Decompile only forms**)を有効にします。

高速逆コンパイル

高速逆コンパイル (**Fast decompilation**)では、初期処理でプロジェクト構造が確立され、プロシージャ本体は開くたびに処理されます。これは、特定のユニットやメソッドを調査する際に有用です。キャッシュが無効になっている場合、処理オプションを変更した後にプロシージャを再度開くと、新しいコードリストが取得できます。

プロトタイプの分析 (**Analyze prototypes**)は予備的なヒューリスティックなシグネチャ解析を追加します。この予備的な逆アセンブル処理により、文字列参照も収集されます。完全な文字列参照パスが実行されていない場合、ダイアログで高速逆コンパイルを無効化し、入力を再処理するよう促されます。

完全処理

高速逆コンパイル (**Fast decompilation**)を無効化してファイルを再度処理すると、プロシージャのコードリストを事前に生成できます。これには時間がかかりますが、生成されたテキストをプロジェクト全体の検索や参照解析に利用可能になります。プロジェクトおよび結合コードリストのエクスポートでも、必要に応じてプロシージャを処理できます。逆コンパイル設定を変更した場合は、ファイルを再度処理する必要があります。

逆コンパイルされたコードをキャッシュ

逆コンパイルされたコードをキャッシュする (**Cache decompiled code**)は生成されたプロシージャのコードリストを保持します。再構築オプションを対話的に比較する際は、キャッシュを無効にしてください。キャッシュされた結果は、設定が変更された効果を自動的に示すものではありません。

変数を宣言

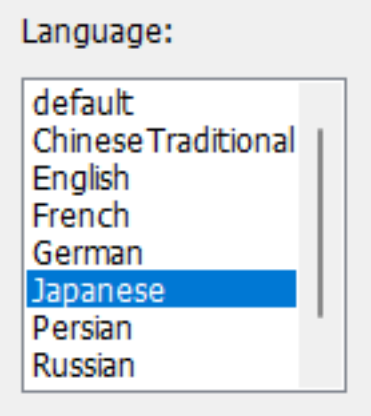
変数の宣言 (**Declare variables**) は、コードリストに復元された変数宣言を追加します。利用可能な RTTI 知識、呼び出し元、および機械語命令の文脈において推論された型を読み取ってください。

ウィンドウレイアウト

Visual Studioのようなウィンドウスタイル (**Window style as in Visual Studio**) は左側にコードを、右側にプロジェクトツリーを配置します。これはワークスペースのレイアウトを変更するものであり、基盤となる解析内容を変更するものではありません。

4.6 インターフェース言語

[ツール] > [オプション] の［パーソナライズ］セクションでインターフェース言語を選択します。



インターフェース言語セレクター。

利用可能なローカライゼーションを選択し、DE Decompiler を再起動して、選択した言語がウィンドウ全体で一貫して適用されるようにしてください。このマニュアルでは英語のインターフェースキャプションを使用しています。

ローカルおよびオンラインドキュメント

英語のマニュアルはオンラインおよびPDFで利用可能です。分離された解析環境で作業する際は、ローカルコピーを保管してください。ローカルマニュアルには、開発者についてに記載されている製品ウェブサイトとサポートメールが含まれています。

選択した言語のヘルプがない場合は、英語のPDFまたはWebサイトを開きます。ローカルのPDFやCHMを開けない場合は、対応するオンラインドキュメントを表示します。

4.7 AI ヘルパー

AI Helperは、対応するDelphiまたはC++Builderのプログラムから DE Decompilerが復元した、Delphi風のPascalコードを処理します。Ollama経由のローカルモデルと、OpenAI API連携を利用するOpenAI（ChatGPT）およびDeepSeekのオンラインモデルに対応しています。

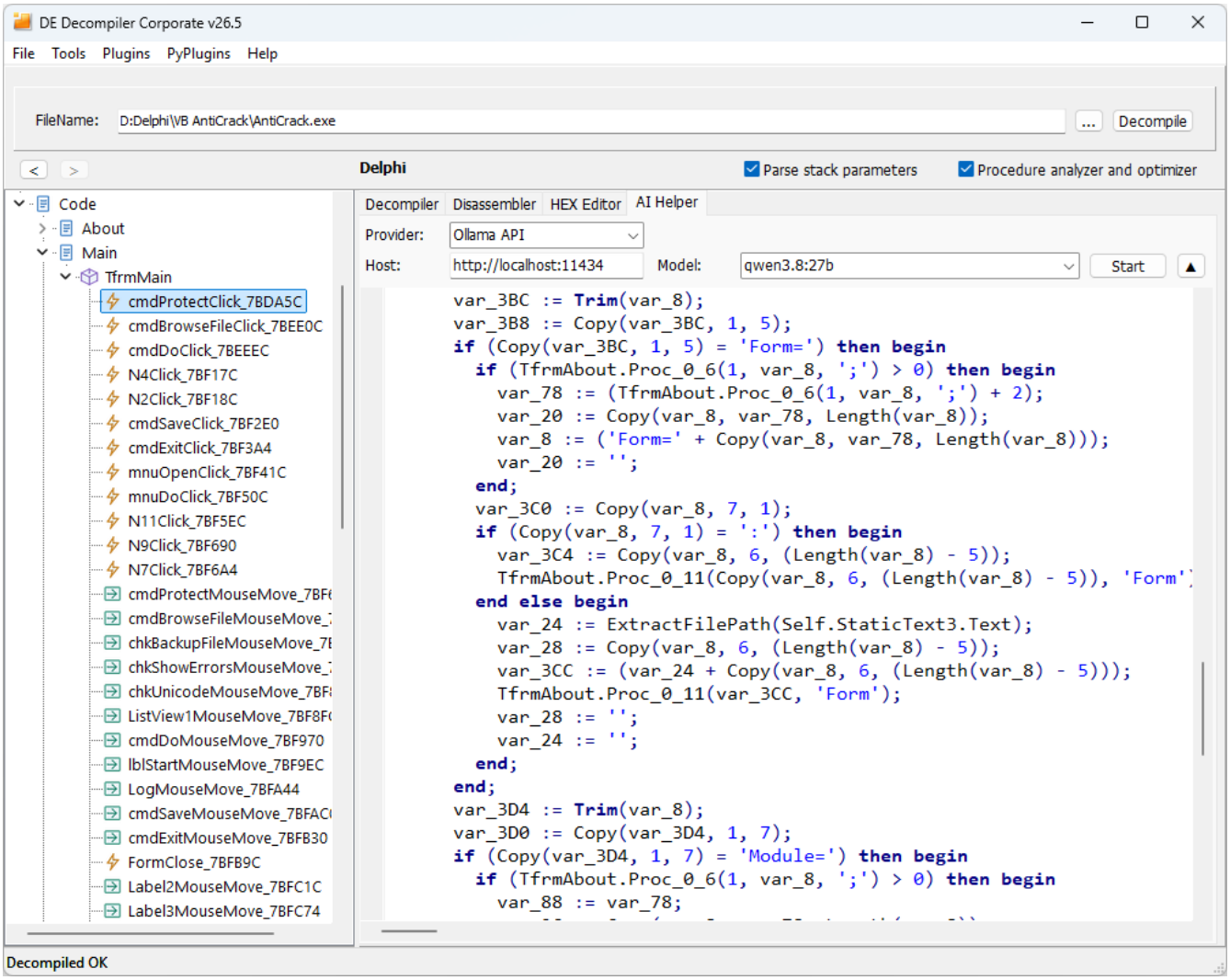
1 プロシージャを開き、その**逆コンパイラ (Decompiler)** 出力を確認します。

2 選択 **AI 支援 (AI Helper)**.

3 **Provider** を選択し、サーバーアドレスを設定してモデルを選択します。

4 処理を開始し、提案された結果を元の逆コンパイルコードリストおよび機械語命令と比較します。

処理は別スレッドで実行され、折りたたまれたコードを含む表示中のプロシージャ全体を使用します。プロンプトではObject Pascalでの出力を指定し、未解決の呼び出し先や、出力用コメント内の復元コードを保持します。時間のかかるリクエストは**キャンセル (Cancel)**でキャンセルできます。補助的な接続設定を折りたたむと、コードの表示領域を広く取れます。



Ollamaを介して復元されたDelphiプロシージャを処理中のAI Helper。

Ollama

互換性のあるワークステーションまたはサーバーで Ollama を実行し、必要なモデルをインストールしてから、AI Helper をそのインスタンスに接続します。ローカルの Ollama サーバーは通常 `http://localhost:11434` を使用します。モデルは、**Host** フィールドによって選択されたインスタンスにインストールされている必要があります。

モデルの選択とGPUメモリ

`qwen3:8b` のような軽量モデルは、復元された Delphi コードの処理において限られた改善しか提供しません。このマニュアルでは、より大きな `qwen3.8:27b` モデルが示されています。

このローカル `qwen3.8:27b` 設定には、以下の GPU メモリに関するガイダンスを使用してください：

構成	GPUメモリ	想定される用途
最小限	16 GB VRAM	処理は非常に遅くなる可能性があります。これはエントリーレベルの構成であり、通常のモデル使用に適した快適なワークステーションではありません。
推奨	32 GB VRAM	図示されたモデルをローカルで通常使用するための最適な構成です。

これらの数値は、Ollama によってサポートされる GPU の専用ビデオメモリを指します。システム RAM および古い統合グラフィックス アダプターで報告される共有メモリ数値は、専用 VRAM と同等ではありません。古い統合グラフィックスを搭載した Pentium M 時代のコンピュータは、このセットアップには適していません。GPU を選択する前に、Ollama ハードウェア互換性 を確認してください。

要件はOllamaを実行しているコンピュータに適用されます。AI Helperが別のOllamaサーバーに接続する場合、そのサーバーがモデルリソースを提供する必要があります。

コンテキストとその他のモデル

コンテキスト長はOllamaの設定画面のスライダーで選択します。この構成に外部拡張機能のインストールは必要ありません。コンテキストはトークン数で表され、モデルへの指示、入力コード、生成される回答を含みます。コンテキストを長くすると必要なメモリも増えるため、上記の容量であらゆるモデル形式やコンテキスト長を利用できるとは限りません。

設定したプロバイダーが対応する他のモデルも選択できます。モデルによって出力の品質や書式が異なり、モデル固有の補助テキストが結果に残る場合があります。生成されたコードは、使用前に確認してください。

オンラインモデル: OpenAI (ChatGPT) および DeepSeek

プロバイダー で **OpenAI API** を選択して、OpenAI モデル (ChatGPT) または DeepSeek オンラインを使用します。選択したサービスに対して **ホスト**、**API キー**、および **モデル** を設定します。ホスト、API キー、およびモデル識別子は同じサービスに属している必要があり、その API キーで利用可能なモデルを使用してください。

DeepSeek は同じ **OpenAI API** プロバイダーを使用します。DeepSeek のホスト、API キー、およびモデル識別子を対応するフィールドに入力します。オンライン推論はプロバイダーのインフラストラクチャ上で実行されるため、これらのサービスを使用する場合、ローカルの Ollama GPU メモリ推奨事項は解析ワークステーションには適用されません。

データ処理

API キーをスクリーンショットや共有された調査資料から除外してください。

AI Helper は、現在逆コンパイルされたコードを選択したサービスに送信します。別のワークステーション上のサーバーまたは外部 API が、そのコードを解析者のコンピュータの外に配置する可能性があります。独自のソース、サンプル、および調査証拠の取り扱い要件に従ってエンドポイントを選択してください。

提案の確認

AI はより明確な名前、宣言、または制御フローを提案できますが、提案はバイナリの動作の証明ではありません。変更された呼び出し、条件、および型を逆アセンブリコードと関連するプロシージャに対して確認してください。モデル支援による解釈を記録する際は、元の逆コンパイル結果を保持してください。

接続とモデルのトラブルシューティング

- モデルが欠落している場合は、**ホスト** によって選択された Ollama インスタンスにそのモデルがインストールされていることを確認してください。
- によって選択された 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てください。応答がない場合は、サービスに到達可能であることを確認し、ホスト、API キー、およびモデル識別子が同じプロバイダーを指していることを確認してください。
- 処理が遅すぎる場合は、リクエストをキャンセルし、Ollama ホストのリソース、モデルサイズ、コンテキスト設定を確認してください。
- 見慣れないモデルが補助タグやサービステキストを残す場合、結果を確認しながらそのテキストを削除するか、別の適切なモデルを選択してください。

4.8 プラグインと SDK

FRX イメージビューアを使用して、復元されたフォームグラフィックスを検査・抽出するか、共有 SDK を通じてカスタムプラグインを開発します。

FRX Image Viewer

FRX Image Viewer 1.2.0は、VB DecompilerまたはDE Decompilerが登録した画像を表示・抽出します。VB DecompilerはVB5/6と.NETのフォーム画像を、DE Decompilerは対応するDelphiおよびC++BuilderのVCLアプリケーションのバイナリDFM画像を提供します。BMP、PNG、JPEG、GIF、ICO、CURに対応しています。

インストールと起動

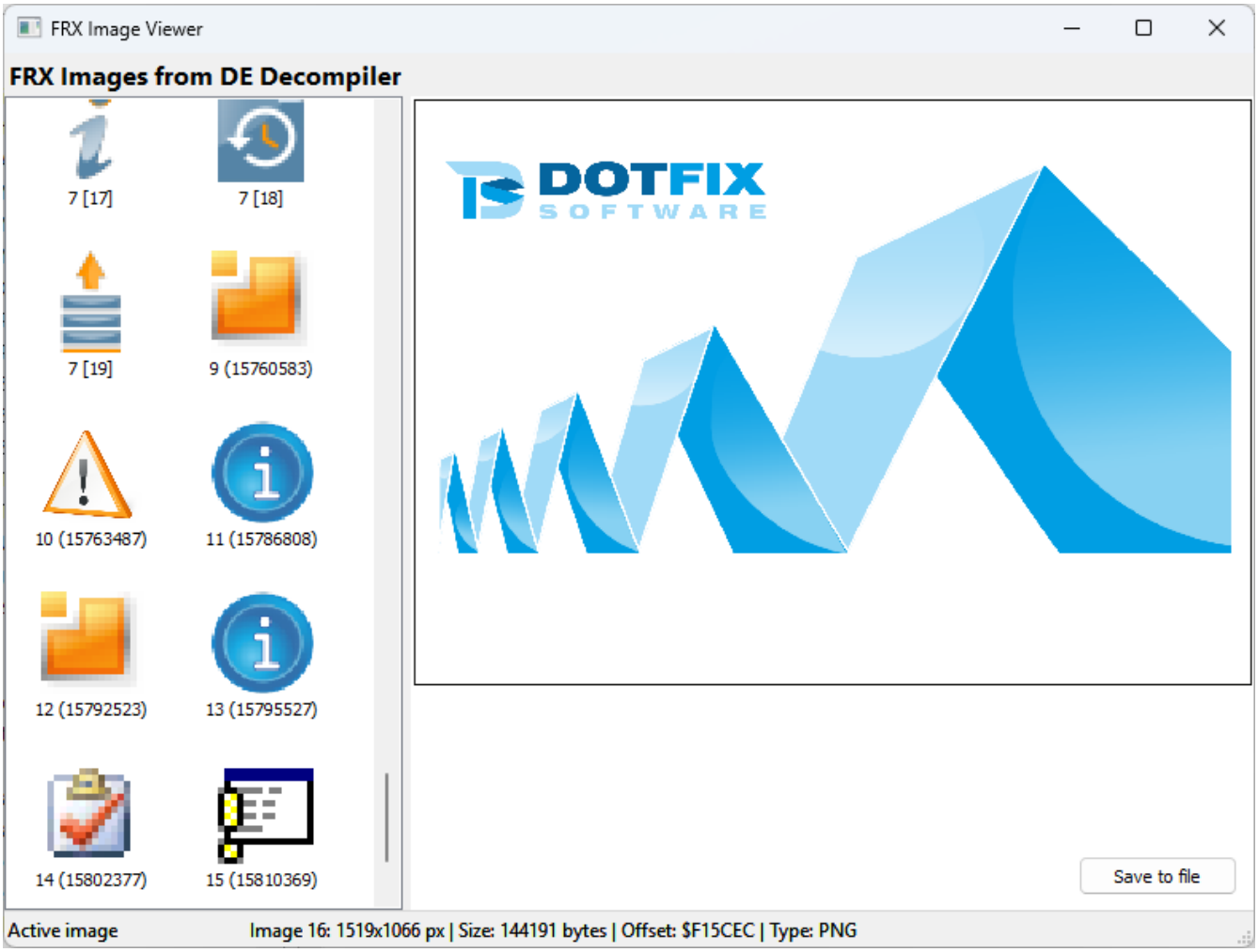
1. `FRXViewer.dll` をホストインストールの `plugins` サブディレクトリにコピーし、逆コンパイラを再起動します。DLL のアーキテクチャは解析対象ファイルのアーキテクチャに関係なく、ホストアプリケーションと一致している必要があります。
2. EXE または DLL を開き、フォームを解析してホストに画像を登録させます。
3. **プラグイン** (**Plugins**) > **FRX Image Viewer** を選択し、イメージを検査するサムネイルを選択します。

1.2.0 ダウンロード には x86 ホスト用の DLL が含まれています。x64 ホストには一致するビルドを使用してください。このプラグインには Windows 10/11 が必要です。ソースリポジトリ にはビルド手順が含まれています。

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。

選択した項目をエクスポートするには、**ファイルに保存**を選択します。スタンドアロンの画像は元のバイトとフォーマットを保持します。個々の ImageList セルは、マスクとアルファ透明度を含む ICO ファイルとしてエクスポートされ、これらの ICO ファイルはソースのビットマップシートから派生しています。



DE Decompiler の FRX イメージビューアでは、Delphi DFM グラフィックスと個別の ImageList アイコンが表示されます。

ネイティブ Delphi ImageList と互換性

対応するホストでは、認識されたネイティブVCLのImageListを個々のアイコンとして表示します。コレクション項目内の独立した画像は、通常の個別画像として扱います。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、画像全体の表示・抽出を維持します。

VB Decompiler 26.6 および DE Decompiler 26.6 でサポートされる共有 SDK リビジョン 26.6 は、オプションの `GetFrxImageListInfo` (63) と `GetProductName` (64) を追加します。古いホストは元の画像 API と全体画像の動作を保持します。機能検出とオフセットルールについては、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 (Plugins) メニューは、インストールされたネイティブ DLL プラグインのエントリポイントです。共有コールバックインターフェースは、カスタム拡張機能に対してサポートされているプロジェクト情報とコードを公開します。利用可能な DLL プラグインは起動時に初期化され、対象のホストアーキテクチャと Windows 環境向けにビルドする必要があります。

プラグインの開発

SDK には Visual C++、Rust、Delphi、Free Pascal の例が含まれています。ワークフローで必要な操作を実装するには、エクスポートされたエントリポイントとホストコールバックを使用してください。

また、AI コーディングエージェントにプラグインの実装を依頼することもできます。SDK および API リファレンスへのリンクと、必要な動作の説明を提供し、DE Decompiler と代表的な入力ファイルに対して結果の DLL をレビュー、ビルド、テストしてください。

インターフェースの互換性

文書化された SDK 呼び出し規約、エクスポートされたエントリポイントおよびデータ表現を使用します。共有 API 識別子は既存の名前を維持し、`VBDecompilerPlugin` プレフィックスを含みます。エクスポートされたシンボル内の製品名を変更すると、ABI が変更されます。

インターフェースが共通でも、すべてのVB Decompiler用プラグインがDE Decompilerのデータに適しているとは限りません。想定するオブジェクト型、復元コードの言語、現在の項目で利用できる操作を確認してください。

開発者リファレンス

製品ウェブサイトのプラグイン API リファレンスを参照してください。そこでは、エクスポートされたエントリポイント、コマンド識別子、コールバックデータおよびバッファ要件が説明されています。SDK はアクティブビューとプロジェクトレベルのコードアクセスを提供します。すべての逆コンパイル済みプロシージャを検索する操作には、対応する逆コンパイル結果（コードリスト）の利用が必要です。コード照会コールバックは、一致する入力ファイルが利用可能な場合に個々のプロシージャを処理できます。

表示中の項目、関数マーク、アドレス

GetDisplayedTreeItem（65）で表示中のフォーム、ユニット、クラス、プロシージャを取得できます。GetModuleFunctionMark（66）とSetModuleFunctionMark（67）で関数のマークを取得・変更し、FindModuleFunctionByVA（68）で仮想アドレスから関数を検索できます。GetModuleFunctionVA（69）は64ビットを含む完全な仮想アドレスを返します。既存のコールバックシグネチャとコマンド番号は維持しています。

4.9 難読化

内蔵の難読化ツールは、Delphiで記述された、またはC++Builderで作成されたWin32/Win64向けVCLアプリケーションに対応します。

この手順は、対応するC++BuilderのWin32/Win64向けVCLアプリケーションにも適用されます。

自身が開発した対応VCLアプリケーションを配布する際に難読化を利用します。この処理はコンパイル済みバイナリを変更します。復元したPascalコードの出力とは別の操作です。

アプリケーションの難読化

1 元のコンパイル済みアプリケーションを保持し、別のコピーに対して作業を行ってください。

2 **ファイル** > **プログラムを開く** (**File** > **Open program**)を通じてアプリケーションを開きます。

3 **ツール** > **プロジェクトを難読化** (**Tools** > **Obfuscate project**) を選択し、出力ダイアログの指示に従ってください。

4 この操作は、書き込み前にネイティブ入力をバックアップし、保存後に復元したモデルを再構築します。変換、書き込み、または再読み込みに失敗した場合、元のファイルとモデルの復元を試みます。復元が完了できない場合は、復元用コピーのパスを報告します。結果得られたアプリケーションは、その intended なデプロイメントで使用されるコンポーネント、データ、およびワークフローでテストしてください。

結果の確認

リリース記録において、元のビルドと難読化されたビルドを区別できるようにしてください。結果得られたアプリケーションにおいて、フォームの作成、コンポーネント間の相互作用、およびイベント駆動型の動作を確認します。難読化は解析を困難にしますが、プログラムが検査や逆コンパイルの対象にならないことを保証するものではありません。

5 リファレンス

5.1 復元されたコードの解釈

DE Decompilerは、機械語命令と利用可能なメタデータからDelphi風のPascalによるプログラムロジックを再構築します。バイナリに保持されている情報が結果の詳細度を決定します。

高水準コードリストが表すもの

ネイティブコードの逆コンパイルでは、式、`if`条件、`case`ブロック、ループ、認識されたVCLメソッドなどの呼び出しを復元します。これにより、リバースエンジニアリングやセキュリティ解析でプログラムのロジックを調べられます。コードリストは実装を再構築したものであり、元のソースコードそのものではありません。

名前、データ構造、呼び出し

- **変数名:** 元のローカル名はバイナリに通常含まれていません。生成された識別子は復元された値を表し、アナリストによって割り当てられた名前を使用するとその役割が明確になる場合があります。
- **型とレイアウト:** RTTIは利用可能な場合に型情報を提供します。他の構造体はアクセスサイズとアドレスから推測されるため、フィールドの型、境界、およびシグネチャにはさらなる解釈が必要になる場合があります。
- **呼び出し:** 認識されたVCLメソッドは名前で表示されることがあります。識別できない呼び出しは、元のVCL名なしでメソッドベースまたはアドレスベースの呼び出しとして残る場合があります。
- **コンパイラの差異:** コンパイラの世代、ライブラリ、および最適化は出力される命令とメタデータに影響を与えます。バージョン固有のRTTI知識はそれらを解釈するのに役立ちますが、推測された型や式には修正が必要になる場合があります。

重要な結論を確認する

1. 解析対象の正確なバイナリ、ユニット、クラス、およびプロシージャを特定します。
2. 機械語命令を使用して不確実な式や型を解決しながら、逆コンパイルされたロジックと宣言を読み込みます。
3. 呼び出し元、引数の準備、条件分岐、および関連する文字列参照を調査します。
4. 命令または復元された値をファイルデータと比較して確認する必要がある場合は、HEX エディタを使用します。
5. 直接観察された内容と、推論に基づく解釈の両方を記録します。

セキュリティ調査

復元された構造とコードは、フォレンジック調査官、アンチウイルス解析担当者、および企業セキュリティチームが、不審なロジック、潜在的なバックドア、および不要な機能の位置を特定するのに役立ちます。結論を出す前に、値や呼び出しがどのように使用されているかを確立してください。文字列や API 名だけでは、悪意のある動作を示すものでもなく、特定の分岐が実行されたことを証明するものでもありません。

ソース復元

復元されたフォーム、宣言、およびプロシージャコードは、手動での復元作業を削減できます。ネイティブコードの逆コンパイルは、コンパイル可能なソースプロジェクトを再現するものではありません。IDE でエクスポートされたコードを開く前に、逆コンパイルの不正確さや互換性の問題をマニュアルで解析・修正し、推論された型と未識別の呼び出しを統合し、欠落しているユニット、コンポーネント、および依存関係を追加してください。復元された素材をビルドするには、この手動作業が必要です。

C++Builderで作成されたアプリケーション

C++Builder固有のライブラリに対応するRTTI知識ベースは、まだ用意されていません。共通のVCLの認識範囲に含まれない関数や、識別できないオブジェクトのプロパティ・メソッドは、ライブラリ内の名前を付けずに表示される場合があります。このため、C++Builderのコードの復元品質は、Delphiで記述されたプログラムより明らかに低くなります。 この制約はコードの復元精度とライブラリ内の名前の識別に関するものです。フォーム表示や構造解析など、その他の機能はDelphiで記述されたプログラムと同じエディションの条件で利用できます。

C++BuilderのコードはDelphi風のPascalとして復元し、PAS/DFM形式で出力します。元のC++ソースには戻しません。他のネイティブコード復元と同様、IDEで利用する前に、推定された型、生成された名前、逆コンパイルの不正確な箇所を手作業で確認・修正してください。

5.2 テクニカルサポート

焦点を絞ったレポートは、認識、フォーム、または逆コンパイルの問題の再現に役立ちます。

DE Decompilerのビルド番号、オペレーティングシステム、入力ファイル名、正確なビルドID、可能な場合はコンパイラバージョン、および関連するユニット、クラス、フォーム、またはプロシージャのアドレスを含めてください。問題の再現に必要な処理オプションと手順を明記してください。

共有が許可されている場合にのみ、最小限の例または元のバイナリファイルを添付してください。機密性の高い調査の場合は、まず問題を説明し、資料を提供する適切な方法について合意してください。APIキーや無関係な顧客情報を決して含めないでください。

現在の連絡先情報はサポートページをご覧ください。

サポートメール： support@de-decompiler.com

5.3 ライセンス

DE Decompilerは、Free Lite版と、Single DeveloperおよびEnterpriseの商用ライセンスで提供されます。

エディション

エディション	適用範囲
Free Lite	VCLフォーム情報、ネイティブWin32およびWin64逆アセンブリ、構文強調表示、プロシージャ検索、文字列参照。高水準コードの再構築は含まれません。
Single Developerライセンス	1人の特定の開発者が、1台の物理コンピュータまたはローカルホストの仮想PCで使用する商用製品機能。
Enterpriseライセンス	ライセンスされた企業内において、最大20名のユーザーが、最大20台の物理コンピュータまたはローカルホストの仮想PCで使用する商用製品機能。

AI Helper、HEXエディタ、および難読化処理には商用ライセンスが必要です。

アップデート、サポート、および更新

新しい商用ライセンスには、1年間のアップデートと技術サポートが含まれます。Single Developerライセンスおよびライセンスの両方について、1年間の更新が可能です。

アップデートおよびサポート期間が終了しても、インストールされたバージョンはアクティベーションが行われたコンピュータ上で引き続き動作します。新しいバージョンへのアクセス、技術サポート、および別のコンピュータへのアクティベーションの移行には、有効なアップデートおよびサポート期間が必要です。

割り当てとアクティベーション

Single Developer ライセンスは、名義登録された開発者個人に固有のものであり、開発者間で共有したり、チームメンバー間でローテーションさせたり、他の開発者に譲渡・再割り当てすることはできません。

ライセンスを他の個人や企業へ譲渡または再登録することはできません。本ライセンスはフローティング型や同時実行型ライセンスではありません。エンタープライズ版のアクティベーションは、ライセンス対象企業のユーザー数およびコンピュータ数の制限内で割り当てられます。

現在の価格や購入リンクについてはライセンスページを、Free Liteについてはダウンロードページをご覧ください。

5.4 開発者について

DE DecompilerはDotFix Softwareによって開発されています。

DotFix Softwareは2001年からソフトウェアツールの開発を行っています。会社ウェブサイトおよびVB Decompilerをご覧ください。

DE Decompilerは2007〜2008年に一般向けに販売されました。その後も社内用途向けに開発を続け、一般向けのライセンス販売を再開しました。主な用途は、Delphi 3〜13で記述されたWin32/Win64向けVCLプログラムのネイティブコード解析です。

C++Builderで作成されたVCLアプリケーションにも対応します。同じワークスペースでフォーム、ユニット、クラス、関数を調べ、その他のツールも利用できます。コンパイラ固有のRTTIに関する制約が影響するのはコードの復元品質であり、これらの機能の利用可否ではありません。

製品サイト: www.de-decompiler.com
サポート窓口: support@de-decompiler.com