VB Decompiler

Benutzerhandbuch

Deutsch

Visual Basic 5.0/6.0 · .NET

DotFix Software

Inhalt

1 Beschreibung 4

2 Schnellstart 6

3 Hauptfenster 7

4 Code anzeigen 9

5 KI-gestützte Codeverbesserung 12

6 Öffnen und Speichern 16

6.1 Datei öffnen 16

6.2 Mit Analysedatenbanken arbeiten 18

6.3 Prozedurliste speichern 19

7 Programmfunktionen 20

7.1 Zu einer Adresse springen 20

7.2 Zeichenfolgen suchen 21

7.3 Zeichenfolgenreferenzen 22

7.4 Dekompilierung ab einer Adresse 24

7.5 Daten bearbeiten 25

7.6 Obfuskation 27

8 Programmeinstellungen 28

8.1 Programmeinstellungen 28

8.2 Formulare 29

8.3 Dekompilierung 30

8.4 Disassembler 32

8.5 Farbschema 33

8.6 Erweiterte Funktionen 35

8.7 Sprachauswahl 39

9 Plugins 40

10 Visuelle Formularrekonstruktion 44

11 Tracing 48

12 Analysefunktionen 50

13 Einschränkungen der Obfuskation 52

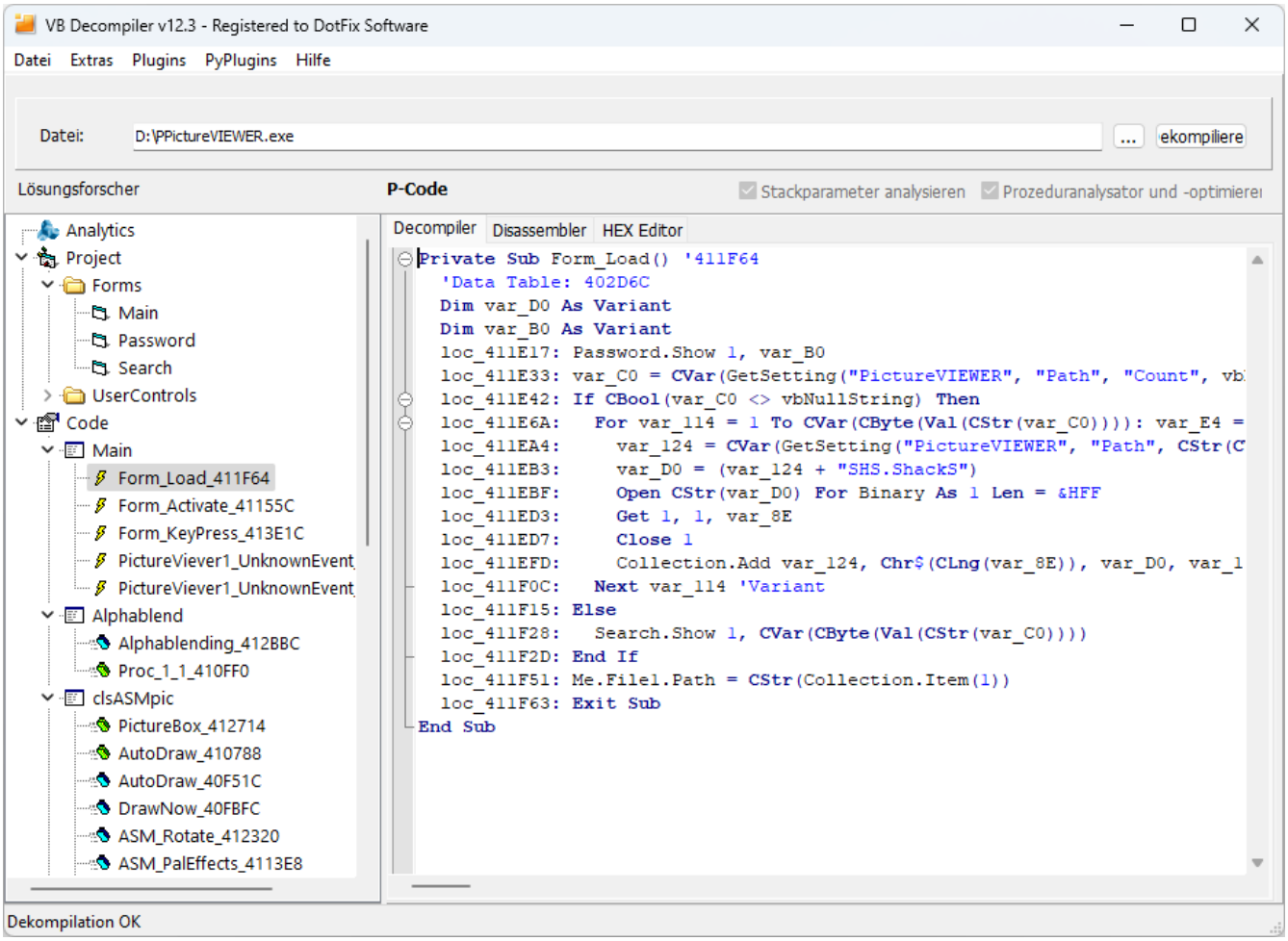
14 Technischer Support 54

15 Registrierung 54

16 Über den Entwickler 55

1 Beschreibung

VB Decompiler ist ein Werkzeug für Reverse Engineering und Quellcode-Wiederherstellung für Visual Basic 5.0/6.0-Programme und .NET-Assemblys. Es kombiniert hochstufige Dekompilierung, Disassemblierung auf Befehlsebene, Metadaten-Inspektion, einen integrierten HEX-Editor und die Wiederherstellung der visuellen Benutzeroberfläche. Typische Anwendungen umfassen die Wartung von Legacy-Software, Incident Response, Malware-Analyse und die Untersuchung nicht dokumentierten Anwendungsverhaltens.



Unterstützte Code-Darstellungen

- **VB5/6 P-Code:** rekonstruiert Visual Basic-Anweisungen aus den P-Code-Befehlen des Compilers und Laufzeitoperationen.
- **VB5/6 Native Code:** disassembliert x86-Befehle und verwendet einen internen Emulator, Laufzeit-Signaturen und Heuristiken zur Rekonstruktion von Visual Basic-ähnlichem Code.
- **.NET:** disassembliert Common Intermediate Language (CIL, auch MS IL genannt) und dekompiliert unterstützte IL-Muster in C#-ähnlichen Code, unabhängig von der ursprünglichen .NET-Quellsprache. Auch nativer x86- und x64-Code in .NET-Assemblys kann disassembliert werden.

Die .NET-Analyse-Engine erfordert weder die .NET-Laufzeitumgebung noch Reflection für die Code-Analyse oder visuelle Formular-Wiederherstellung.

Analyse-Tools

Verwenden Sie Prozedur- und Textsuche, Querverweise und Variablenumnennung sowie Zeichenfolgenreferenzen, um das Verhalten zu untersuchen. Der visuelle Viewer rekonstruiert klassische VB-Formulare, .NET Windows Forms und WPF-Fenster, Seiten und benutzerdefinierte Steuerelemente, die in kompilierten BAML-Ressourcen gespeichert sind.

Lizenzen mit Tracing-Funktionen bieten eine schrittweise Emulation von VB5/6 Native Code und P-Code. Lizenzen mit Analysefunktionen bieten kategorisierte Berichte für VB5/6- und .NET-Anwendungen. Der KI-Hilfe (AI Helper) kann wiederhergestellten VB5/6- oder C#-Code mit Ollama oder der OpenAI-API-Integration verarbeiten.

Auslegung von wiederhergestelltem Code

Kompilierung entfernt Informationen wie Kommentare, viele lokale Variablennamen und einige Typ- und Prozedur-Metadaten. Optimierung und Obfuskation können die Code-Struktur weiter verändern. Die rekonstruierte Auflistung unterstützt die Code-Analyse und -Wiederherstellung. Überprüfen Sie wichtige Schlussfolgerungen anhand des disassemblierten Codes, der Metadaten, der Zeichenfolgenreferenzen und des Codes der zugehörigen Prozeduren.

2 Schnellstart

1. Wählen Sie **Datei > Programmdatei öffnen (File > Open program)** aus und öffnen Sie die EXE-, DLL- oder OCX-Datei zur Analyse. Die Verarbeitung startet automatisch. Warten Sie, bis der Projektbaum gefüllt ist und die Verarbeitung abgeschlossen ist.
2. Öffnen Sie eine Prozedur in **Lösungsforscher (Solution explorer)**. Vergleichen Sie die Ansichten **Decompiler** und **Disassembler**; verwenden Sie **HEX Editor**, um die entsprechenden Bytes zu untersuchen.
3. Drücken Sie **Ctrl+T**, um eine Prozedur nach seinem Namen oder einem Teil seines Namens zu finden. Verwenden Sie **Ctrl+F**, um angezeigten Text zu suchen, oder **Extras > Zeichenfolgenverweise (Tools > String references)**, um Zeichenfolgen und die Prozeduren zu untersuchen, die diese verwenden.
4. Öffnen Sie ein wiederhergestelltes Formular und wählen Sie **Designer (schreibgeschützt) (Designer (read only))** für eine visuelle Vorschau. Für klassisches VB, Windows Forms und WPF verwenden Sie wiederhergestellte Ereignisse, um zum entsprechenden Handler-Code zu navigieren.
5. Speichern Sie die Analyse über **Datei > Datenbank (File > Data Base)**, wenn Sie sie später fortsetzen oder die Ergebnisse mit einem anderen Analysten teilen möchten.

Auswahl eines Dekompilierungsmodus

Schnelle Dekompilierung verarbeitet Prozeduren, wenn Sie sie öffnen. Aktivieren Sie **Prototypen analysieren (Analyze prototypes)** für eine zusätzliche vorläufige Analyse von Signaturen als Private deklarierter Prozeduren. Für die Suche im gesamten dekompilierten Text oder zum Exportieren eines vollständigen Projekts deaktivieren Sie die Schnelle Dekompilierung und verarbeiten Sie die Datei erneut.

Speichern wiederhergestellten Quellcodes

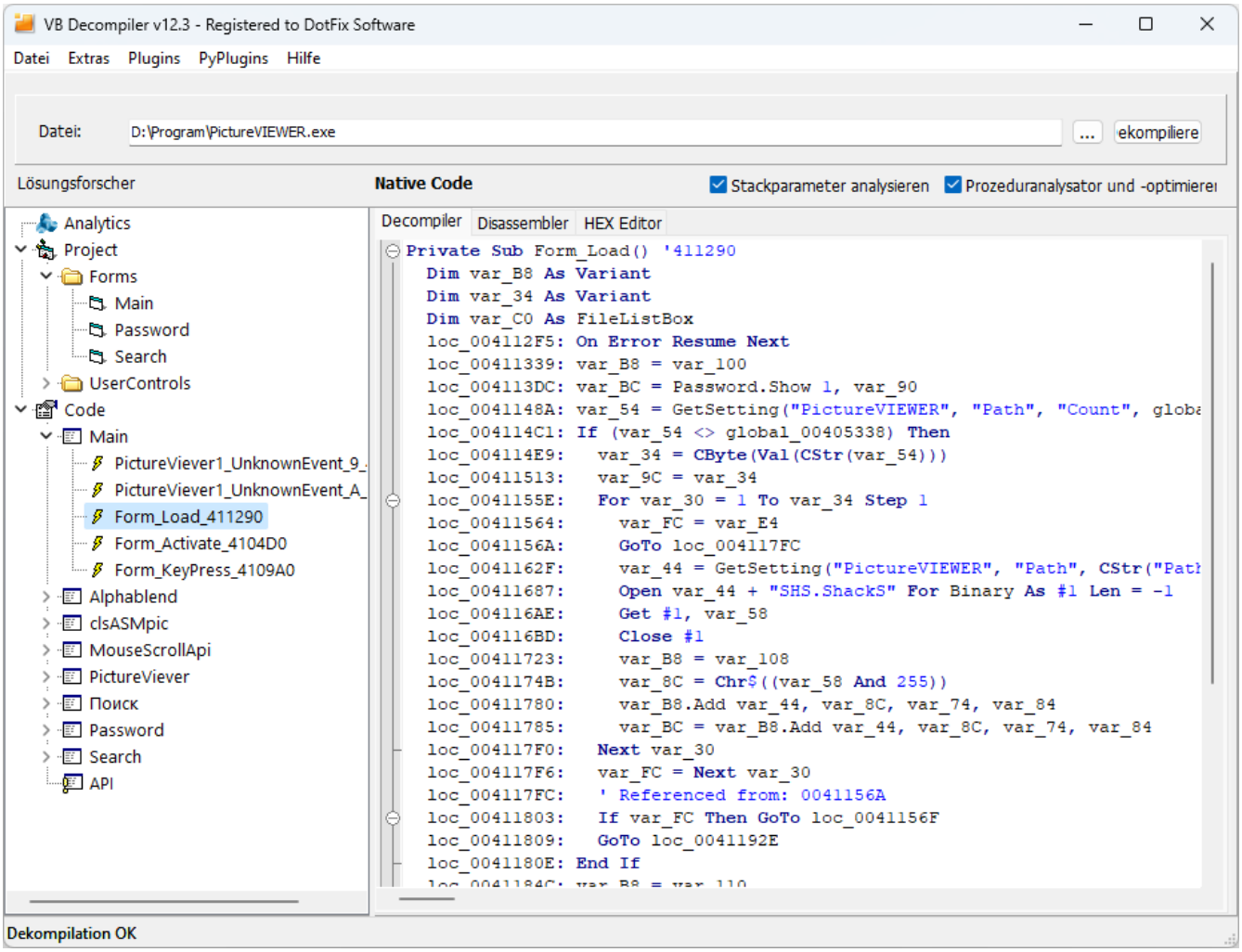
Um VB5/6 Dekompilierungsergebnisse zu exportieren, wählen Sie **Datei > Dekompiliertes Projekt speichern (File > Save decompiled project)** und wählen Sie einen Ausgabepfad. Der Export enthält wiederhergestellte Formulare und Module; FRX-Dateien enthalten zugehörige Bilder und andere binäre Formulardaten.

Korrigieren Sie vor dem Öffnen des exportierten Codes in einer IDE manuell alle Dekompilierungsfehler und Inkompatibilitäten im wiederhergestellten Quellcode.

Siehe Visuelle Formularrekonstruktion, Code anzeigen und KI-gestützte Codeverbesserung für die detaillierten Arbeitsabläufe.

3 Hauptfenster

Lösungsforscher (Solution explorer) organisiert das wiederhergestellte Projekt in Formulare, Module, Klassen und Prozeduren. Die angrenzende Ansicht zeigt den ausgewählten Code, das Formular oder die Metadaten an. Die Option **Fensterstil wie in Visual Studio (Window style as in Visual Studio)** platziert den Baum rechts und den Code links.



Code- und Datenansichten

- **Decompiler:** hochstufig rekonstruierter Code.
- **Disassembler:** die entsprechenden x86/x64-Befehle, VB P-Code oder .NET IL mit Adressen.
- **HEX Editor:** die zugrunde liegenden Dateibytes, mit Navigation über virtuelle Adresse oder Datei-Offset.
- **Designer (schreibgeschützt) (Designer (read only)):** ein rekonstruiertes Formular und seine wiederhergestellten Eigenschaften, sofern unterstützt.
- **KI-Hilfe (AI Helper):** optionale KI-Verarbeitung der aktuell dekompilierten Prozedur.

Die verfügbaren Registerkarten hängen vom ausgewählten Objekt ab. Der Code-Typ-Indikator unterscheidet **P-Code**, **Native Code** und **.NET Assembly**.

Stackparameter analysieren (Parse stack parameters)

Für P-Code kombiniert dieses Steuerelement Stapeloperationen und Aufrufe zu Ausdrücken mit Argumenten. Wenn es deaktiviert wird, werden Zwischenoperationen sichtbar und dies kann helfen, wenn ein unbekannter Prototyp unerwartete Stapelwerte verbraucht. Für Native Code führt das Aktivieren dazu, dass der Emulator Ausdrücke auf höherer Abstraktionsebene rekonstruiert; das Deaktivieren macht die Ausgabe auf Assemblerebene sichtbar. Für .NET trägt die Stapelverarbeitung zur Rekonstruktion von IL als C#-ähnliche Ausdrücke bei.

Prozeduranalysator und -optimierer (Procedure analyzer and optimizer)

Dieses Steuerelement vereinfacht die rekonstruierte Auflistung, indem es Ausdrücke und erkannte Kontrollflussmuster kombiniert und unterstützte Formen von Compiler-Gerüstcode entfernt. Es gilt für VB5/6 und .NET-Code. Der .NET-Optimierer kann unterstützte `while`, `do...while` und `foreach`-Muster, verschachtelte Bedingungen, Operatorausdrücke und interpolierte-String-Handler-Muster rekonstruieren. Diese Transformationen werden vom Decompiler selbst durchgeführt und erfordern keinen KI-Hilfe.

Ist ein Ausdruck oder eine Verzweigung unklar, vergleichen Sie die Ausgabe mit deaktivierter Optimierung und untersuchen Sie die Disassemblierung. Im **Schnelle Dekompilierung (Fast decompilation)**-Modus ohne Caching können diese Steuerelemente geändert werden, während Sie eine Prozedur inspizieren. Bei vollständiger Dekompilierung oder gecachter Ausgabe verarbeiten Sie die Datei erneut nach dem Ändern der Einstellungen.

Navigation und Tastaturzugriff

Die Pfeilschaltflächen über dem Baum navigieren durch zuvor angezeigte Prozeduren; **Esc** kehrt zur vorherigen Prozedur zurück. Die Maus-Tasten Zurück und Vorwärts verwenden dieselbe Historie, einschließlich der gespeicherten Position in einer Prozedur.

Drücken Sie **Alt+1**, um den Fokus zwischen Lösungsforscher und der aktiven Ansicht zu wechseln. Drücken Sie **Enter** im Baum, um das ausgewählte Objekt zu öffnen. Im HEX-Editor wechselt der Fokus mit **Tab** zwischen den hexadezimalen und Textfeldern und dann zum nächsten Steuerelement. Beschriftungen der Bedienelemente sind mit ihren Eingabefeldern für Screen-Reader-Navigation verknüpft.

Siehe Code anzeigen für kontextsensitive Tastaturkürzel, Querverweise und Umbenennung sowie Analysefunktionen für die Analyse (Analytics)-Ansicht.

4 Code anzeigen

Die Code-Ansichten kombinieren Syntaxhervorhebung, zusammenklappbare Blöcke, adressbasierte Navigation und Querverweise. Verwenden Sie das Listing in Hochsprachencode, um eine Prozedur zu verstehen, und die Disassemblierung sowie Dateibytes, um zu überprüfen, wie das Ergebnis erzielt wurde.

Bezeichner hervorheben und zu ihnen navigieren

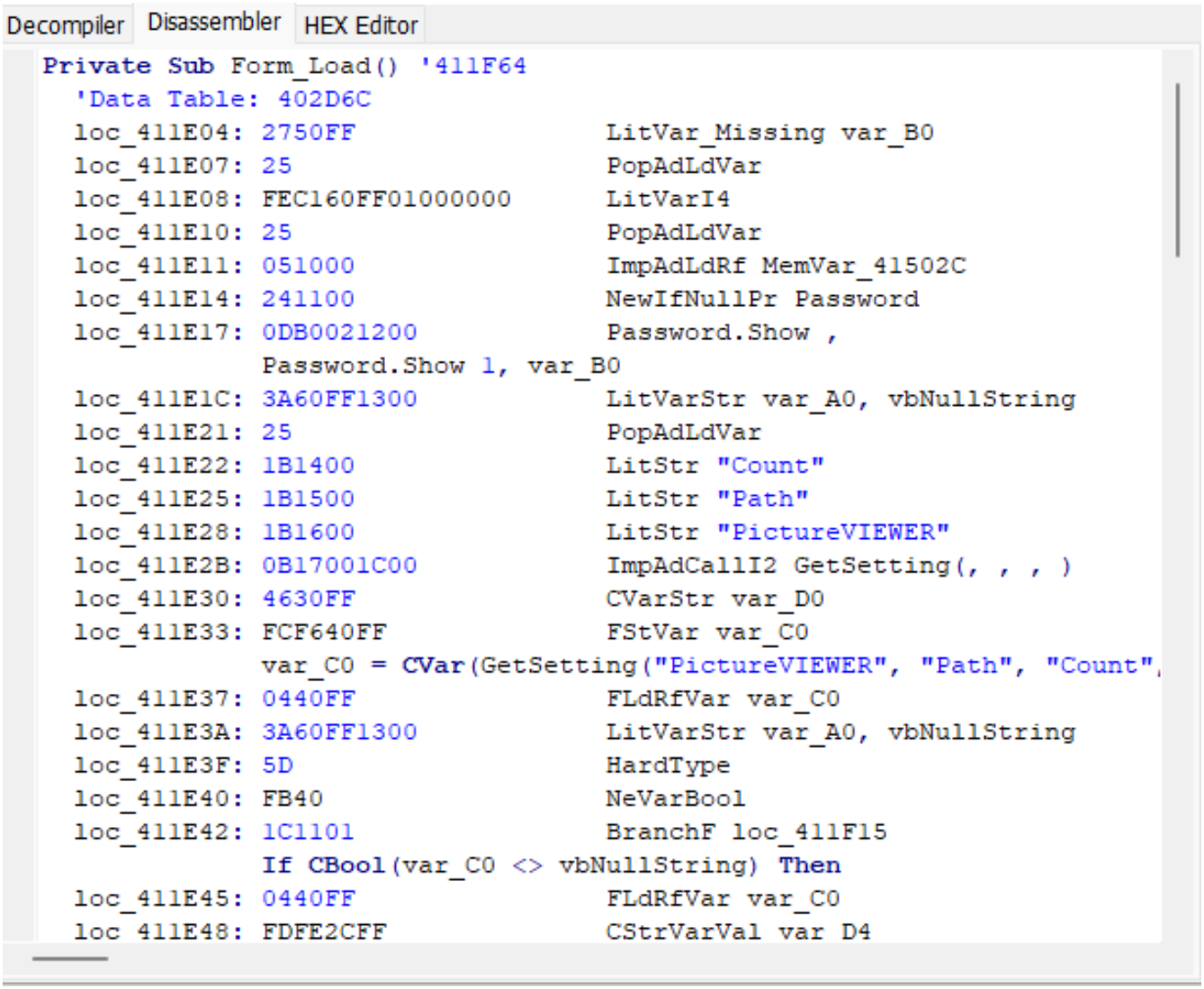
Setzen Sie den Cursor auf ein Register, eine Variable oder einen Funktionsnamen, um übereinstimmende Vorkommen in der aktuellen Auflistung hervorzuheben. Lokale Variablen wie `var_XX` und `str_XX` werden ebenfalls hervorgehoben, wenn sie als Basis eines Member-Ausdrucks verwendet werden, beispielsweise `str_1.Value`. Das Markieren eines `global_XX`-Identifikators hebt seine Vorkommen innerhalb der aktuellen Prozedur hervor; diese Hervorhebung ist getrennt von einer projektweiten Referenzsuche.

Doppelklicken Sie auf einen erkannten Prozeduraufruf, um die aufgerufene Prozedur zu öffnen. Adressähnliche Konstanten können stattdessen zu den entsprechenden Bytes im HEX-Editor führen. Verwenden Sie **Esc**, die Navigationspfeile oder die Maus-Rückwärts/Vorwärts-Tasten, um durch die Anzeigehistorie zu navigieren.

Dekompilierten und disassemblierten Code zusammenführen

Klicken Sie mit der rechten Maustaste in das dekompilierte Listing und wählen Sie **Kopieren Sie zu dem Disassembler (Copy to disassembler)**, oder drücken Sie **D** in der Code-Ansicht. Dies verbindet die rekonstruierten Anweisungen mit dem disassemblierten Listing für die aktuelle Prozedur. Um die aktuelle Position beizubehalten, setzen Sie den Cursor auf die relevante `loc_`-Adresse, bevor Sie den Befehl aufrufen.

Lassen Sie **Adressen hinzufügen (Add Addresses)** in den Dekompilierungseinstellungen aktiv, wenn Sie diesen Workflow verwenden. Der Disassembler zeigt immer Adressen an; Adressen hinzufügen steuert die Decompiler-Ansicht.



Umbenennen von Variablen

Bei VB5/6 P-Code und Native Code klicken Sie mit der rechten Maustaste auf eine Variable und wählen Sie **Umbenennen (Rename variable)**, oder setzen Sie den Cursor auf die Variable und drücken Sie **N**. Geben Sie einen aussagekräftigen Namen ein, der die bei der Analyse ermittelte Bedeutung beschreibt. Der Decompiler prüft auf Namenskollisionen und reservierte Schlüsselwörter. Um eine Umbenennung rückgängig zu machen, verwenden Sie **Umben. zurück (Undo renaming)** oder **Ctrl+Shift+Z**. Dieser Befehl ist unabhängig von der gewöhnlichen Rückgängig-Funktion der Textbearbeitung.

Durch Umbenennen lässt sich das rekonstruierte Listing leichter analysieren; es wird kein Name wiederhergestellt, der durch Kompilierung entfernt wurde. Behandeln Sie vom Analysten vergebene Namen als Annotationen, wenn Sie Analyseergebnisse weitergeben.

Auffinden von aufrufenden Prozeduren und globalen Variablenreferenzen

Um aufrufende Prozeduren zu finden, klicken Sie im Lösungsforscher (Solution explorer) mit der rechten Maustaste auf eine Prozedur und wählen Sie **Referenzen finden (Find references)**, oder drücken Sie **X**, nachdem Sie diese Prozedur im Baum ausgewählt haben. Die Ergebnisse listen die Prozeduren auf, die erkannte Aufrufe der ausgewählten Prozedur enthalten. Doppelklicken Sie auf ein Ergebnis, um die aufrufende Prozedur zu untersuchen. Dieser Workflow unterstützt VB5/6 und .NET; .NET-Unterstützung wurde in Version 12.4 hinzugefügt.

Um eine globale Variable zu untersuchen, setzen Sie den Cursor auf ihren Identifikator in der Code-Ansicht und drücken Sie **X**. Die Taste X sucht somit im Baum nach Prozedurreferenzen und in der Code-Ansicht nach Referenzen auf globale Variablen. Eine Liste erkannter Referenzen schließt zusätzliche indirekte, dynamisch aufgelöste oder durch Obfuskation verborgene Zugriffe nicht aus.

Kontextsensitive Tastenkürzel

Fokus	Tastenkürzel	Aktion
Code	N	Umbenennen der Variable unter dem Cursor (VB5/6).
Code	X	Referenzen auf die globale Variable unter dem Cursor suchen.
Code	D	Aktuellen dekompilierten Code in den Disassembler kopieren.
Code	F4	Tracing bis zur Adresse in der aktuellen Zeile ausführen; erfordert Tracing-Funktionen.
Code	Ctrl+Shift+Z	Umbenennung der Variable unter dem Cursor rückgängig machen.
Baum	X	Auffinden von Referenzen zur ausgewählten Prozedur.
Baum	G	Dialog zum Wechsel zu einem Abschnitt öffnen.
Baum	Ctrl+C	Vollständigen Pfad des ausgewählten Objekts im Baum kopieren.
Baum	Space or Insert	Markieren oder Entfernen der Markierung des ausgewählten Elements.

Einzelbuchstaben-Tastenkürzel hängen von der fokussierten Ansicht ab und ersetzen nicht das normale Tippen in Eingabefeldern. Verwenden Sie **Alt+1**, um den Fokus zu ändern. Sehen Sie Zeichenfolgen-Suche für Ctrl+T und Ctrl+F, Sprung zur Adresse für Baumwege und Adressen, und Tracing für Emulator-Beschränkungen.

5 KI-gestützte Codeverbesserung

KI-Hilfe (AI Helper) verarbeitet Code, den VB Decompiler bereits rekonstruiert hat. Er unterstützt dekompilierten C#-Code aus .NET-Assemblys und Visual Basic 5.0/6.0-Code aus P-Code oder Native Code. Sie können ein lokales Modell über Ollama verwenden oder sich mit einem Dienst verbinden, indem Sie die OpenAI-API-Integration nutzen.

KI kann klarere Kontrollflüsse, Variablennamen, Deklarationen und Ausdrücke vorschlagen. KI kann Informationen nicht zuverlässig wiederherstellen, die in der Binärdatei fehlen, oder Fragmente reparieren, die der Decompiler falsch interpretiert hat. Vergleichen Sie Vorschläge mit dem ursprünglichen dekompilierten Listing, der Disassemblierung und den umgebenden Prozeduren, bevor Sie sie im wiederhergestellten Quellcode oder in einem Untersuchungsbericht verwenden.

Verarbeitung einer Prozedur

1. Öffnen Sie die erforderliche Prozedur im Lösungsforscher (Solution explorer) und überprüfen Sie deren **Decompiler**-Listing.
2. Wechseln Sie zu **KI-Hilfe** und konfigurieren Sie Anbieter, Host und Modell wie unten beschrieben.
3. Starten Sie die Verarbeitung mit der Schaltfläche neben der Modellauswahl.
4. Überprüfen Sie den zurückgegebenen Code im KI-Hilfe und vergleichen Sie ihn mit dem Ergebnis des Decompilers.

KI-Verarbeitung läuft in einem separaten Thread. Verwenden Sie **Abbrechen (Cancel)**, wenn das Modell zu lange zum Antworten benötigt. Zusätzliche Verbindungseinstellungen können zusammengeklappt werden, um mehr Platz für Code zu lassen.

Nutzung eines lokalen Modells über Ollama

1. Installieren Sie Ollama von dessen offizieller Downloadseite und starten Sie es.
2. Laden Sie das empfohlene Modell herunter, indem Sie `ollama pull qwen3:8b` in einem Terminal ausführen.
3. Wählen Sie den lokalen LLM/Ollama-Anbieter im KI-Hilfe aus. Stellen Sie **Ollama-Host (Ollama host)** auf die Serveradresse; die lokale Standardadresse ist `http://localhost:11434`.
4. Wählen Sie das installierte Modell in **Mod. (Model)** aus und starten Sie die Verarbeitung.

Wir empfehlen `qwen3:8b`. Die integrierten Prompts von KI-Hilfe sind auf dieses leichtgewichtige Modell abgestimmt. Neuere Modelle können für die Codeverarbeitung länger benötigen, ohne das Ergebnis zu verbessern.

Die Modellauswahl ist nicht eingeschränkt: Sie können jedes Modell verwenden, das in der ausgewählten Ollama-Instanz installiert ist. Bei einem unbekannten Antwortformat kann der Antwortparser modellspezifische Tags oder anderen Hilfstext im angezeigten Ergebnis belassen.

Hardware für die lokale Ausführung mit Ollama

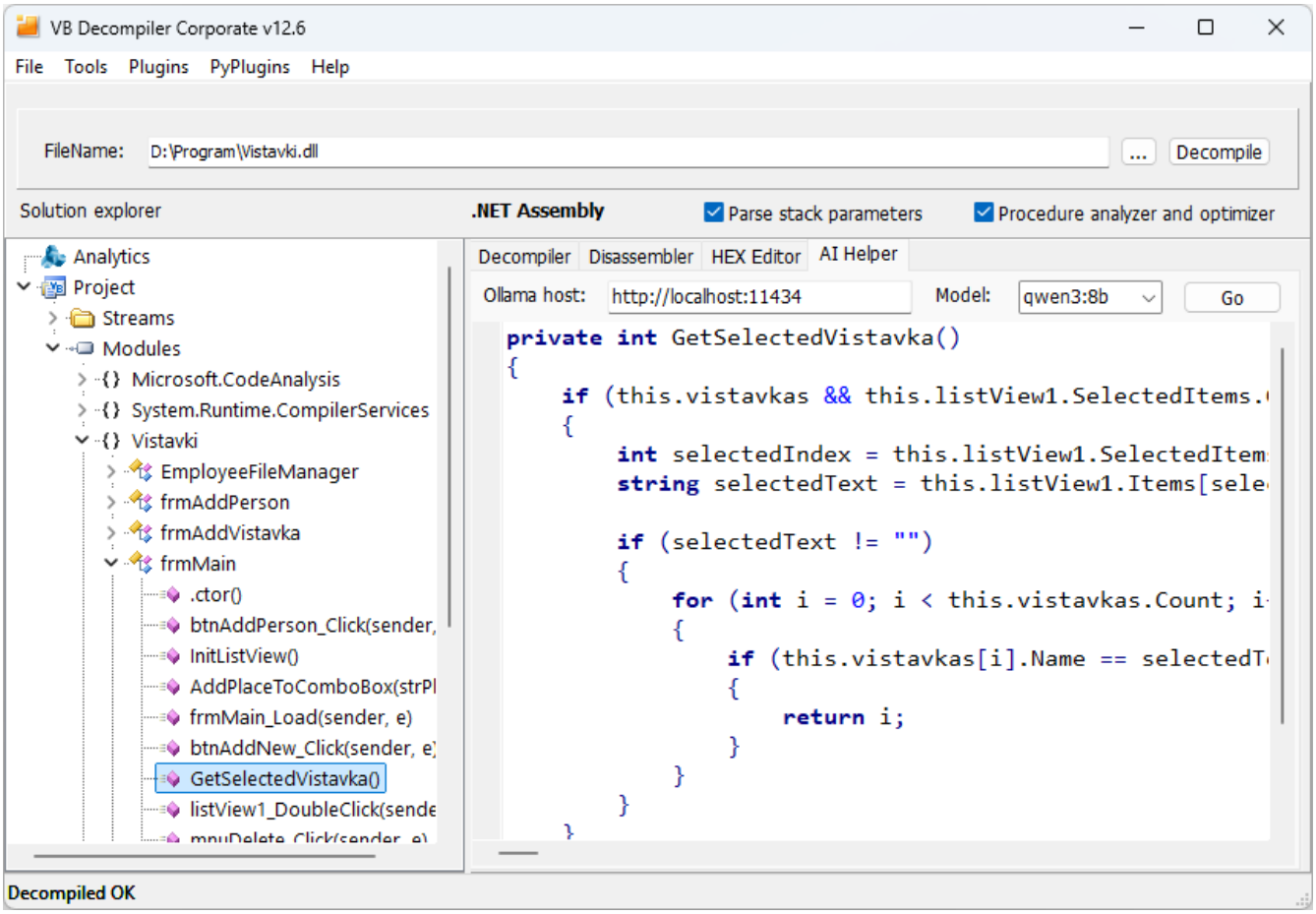
Die lokale KI-Verarbeitung erfolgt auf dem Rechner, auf dem Ollama läuft, und benötigt erheblich mehr Ressourcen als die Dekompilierung. Die folgenden praxisbezogenen Konfigurationen gelten für `qwen3:8b` mit `Q4_K_M`-Gewichten, dem standardmäßigen `f16`-Kontextcache und jeweils einer Anfrage. Der Kontext wird in Tokens gemessen: 4K entspricht 4096 Tokens, 128K entspricht 131072 Tokens. Dazu zählen die Anweisungen, der Quellcode und die erzeugte Antwort.

Komponente	Praktische Mindestkonfiguration: 4K	Empfohlene Konfiguration: 128K
Betriebssystem	Windows 10 (22H2) oder Windows 11, jeweils 64 Bit	Windows 10 (22H2) oder Windows 11, jeweils 64 Bit
Prozessor	Moderner x64-Prozessor mit AVX2 und mindestens 4 physischen Kernen	Moderner x64-Prozessor mit AVX2 und mindestens 8 physischen Kernen
Arbeitsspeicher (RAM)	16 GB	64 GB
Grafikkarte	Ollama-kompatible GPU mit mindestens 8 GB dediziertem VRAM	Ollama-kompatible GPU mit mindestens 32 GB dediziertem VRAM
Datenträger	SSD mit mindestens 20 GB freiem Speicher für Ollama und dieses Modell	SSD mit mindestens 20 GB freiem Speicher für Ollama und dieses Modell

Wählen Sie die Kontextgröße mit dem Schieberegler in den Ollama-Einstellungen. Eine für 4K geeignete Grafikkarte mit 8 GB reicht für die oben beschriebene 128K-Konfiguration nicht aus. Der Modelldownload umfasst etwa 5.2 GB; der gesamte Speicherbedarf ist höher. Bei 128K benötigt allein der `f16`-Kontextcache ungefähr 18 GiB, zuzüglich Modellgewichten und Arbeitsbereichen. Halten Sie Speicher für Windows und andere Anwendungen frei.

Bei ausreichendem RAM kann Ollama auf der CPU rechnen; die Codeverarbeitung kann dabei jedoch sehr langsam sein. Pentium-M-Systeme und ähnlich alte Rechner sind für die lokale Ausführung von `qwen3:8b` ungeeignet. Der gemeinsam genutzte Speicher älterer Intel-HD-Grafik ersetzt den oben angegebenen dedizierten VRAM nicht. Diese Anforderungen gelten für den Ollama-Rechner; ein entfernter Server oder externer API-Dienst verwendet seine eigene Hardware.

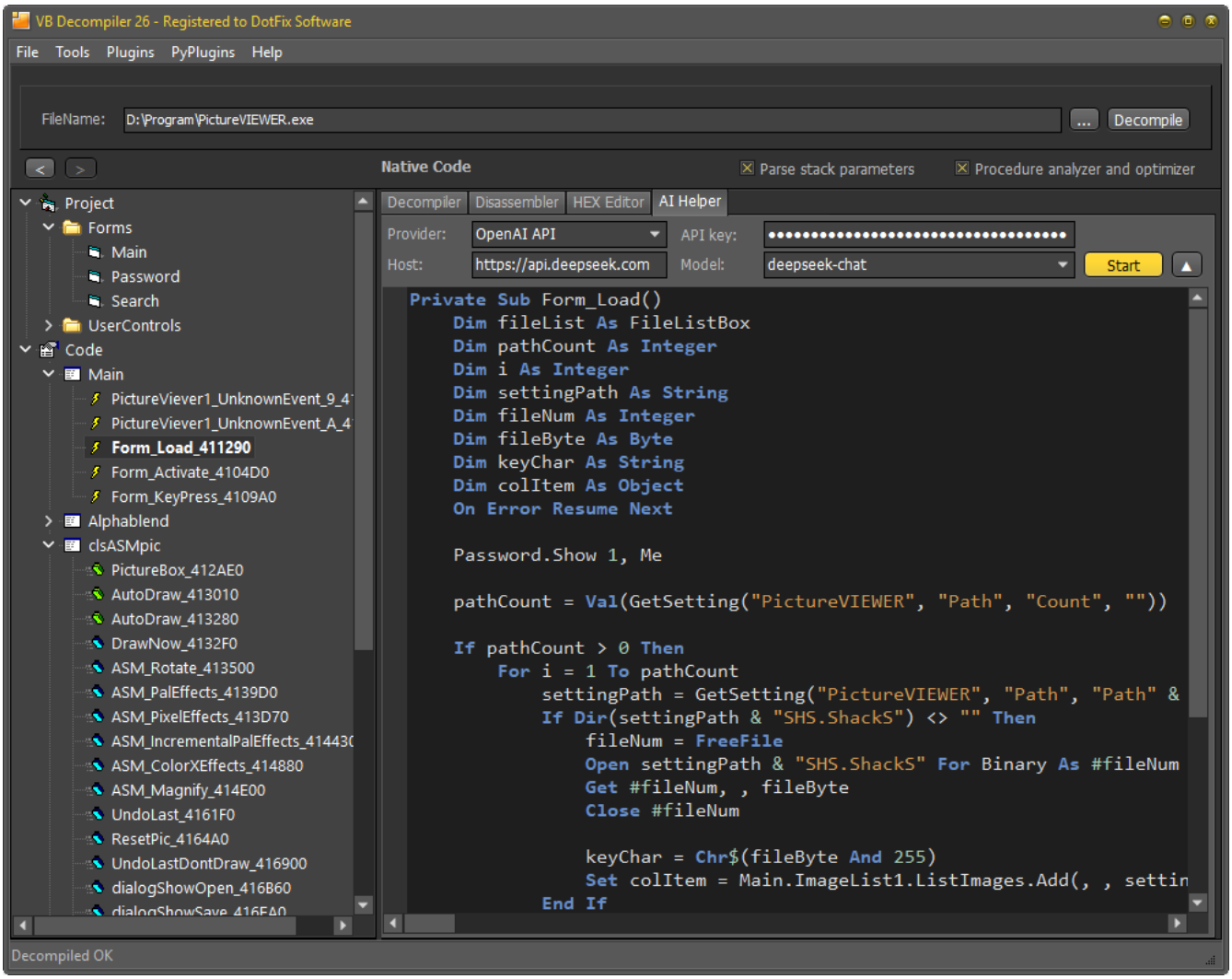
Beginnen Sie bei kurzen Prozeduren mit 4K. Ein größerer Kontext erhöht den Speicherbedarf und die Verarbeitungszeit langer Eingaben. Brechen Sie eine zu langsame Verarbeitung ab, verkleinern Sie den Kontext und prüfen Sie mit `ollama ps` die GPU-Nutzung. Der tatsächliche Speicherbedarf hängt von Ollama-Version, Backend und anderen laufenden Aufgaben ab.



Nutzung der OpenAI-API-Integration

Stellen Sie **Anbieter (Provider)** auf **OpenAI API**. Konfigurieren Sie **Host**, **API-Key (API key)** und **Mod.** für den beabsichtigten Dienst. Geben Sie einen gültigen API-Schlüssel und eine Modell-ID ein, auf die Sie mit diesem Schlüssel zugreifen können. Halten Sie API-Schlüssel aus Screenshots und geteiltem Analysematerial heraus.

KI-Hilfe unterstützt auch DeepSeek über den **OpenAI API**-Anbieter. Geben Sie den DeepSeek-Host, den API-Schlüssel und die Modell-ID in den entsprechenden Feldern ein. Der folgende Screenshot zeigt diese Konfiguration.



Wo der Code verarbeitet wird

KI-Hilfe sendet den aktuellen dekompilierten Code an den konfigurierten Dienst. Für einen rein lokalen Workflow verwenden Sie einen lokal laufenden Ollama-Server und ein lokal ausgeführtes Modell. Ein Ollama-Host auf einem anderen Rechner, ein Ollama-Cloud-Modell oder ein externer API-Dienst kann Code außerhalb des Analyse-Arbeitsplatzes übertragen.

Für vertrauliche Quellcode-Wiederherstellung oder Sicherheitsuntersuchungen wählen Sie den Endpunkt gemäß den Datenbehandlungsanforderungen Ihrer Organisation. Schließen Sie nicht allein aus der Anwesenheit von Ollama oder einer localhost-URL auf einen Offline-Betrieb.

Fehlerbehebung

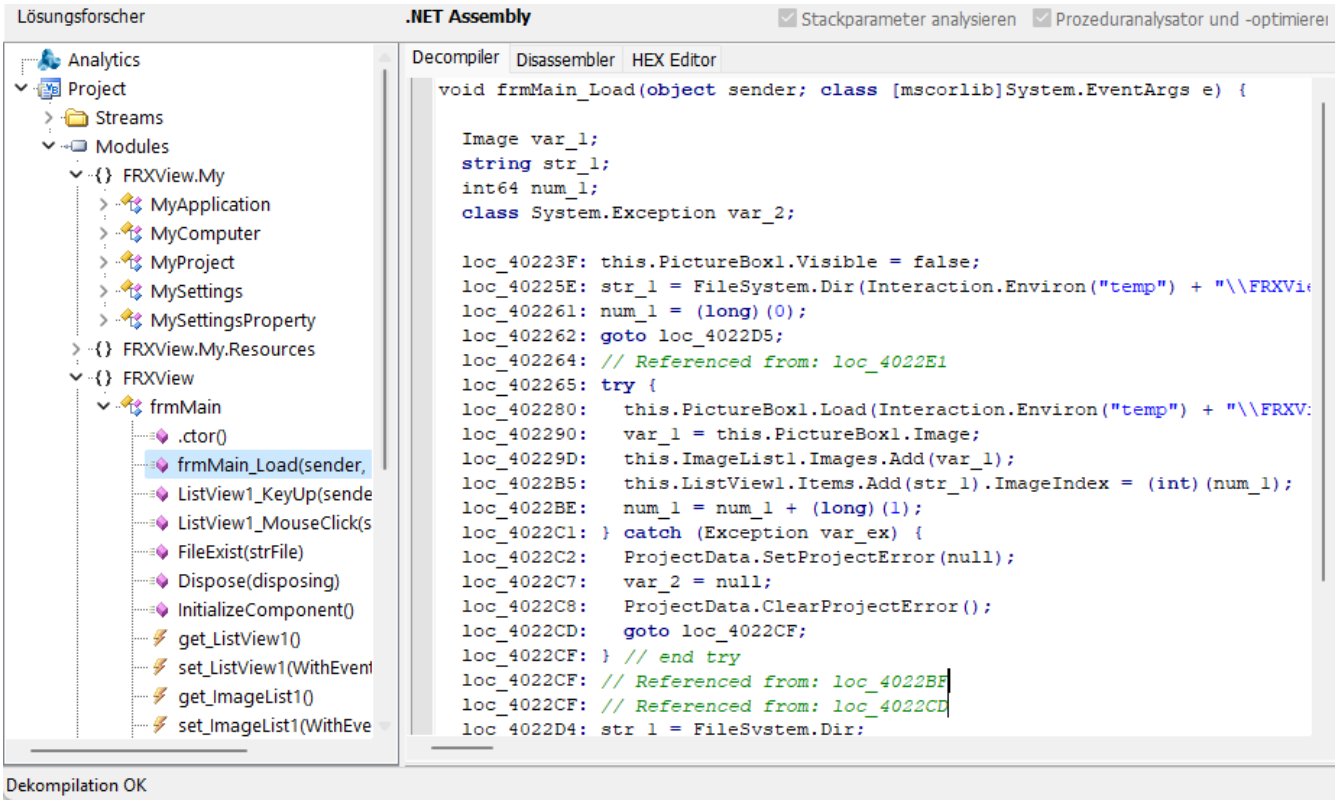
- Wenn ein lokales Modell fehlt, prüfen Sie, ob es in der von dem Host-Feld angegebenen Ollama-Instanz installiert ist.
- Wenn keine Antwort vorliegt, überprüfen Sie, ob der ausgewählte Dienst läuft und erreichbar ist, und ob Endpunkt, Zugangsdaten und Modell-ID demselben Anbieter entsprechen.
- Wenn die Verarbeitung zu langsam ist, brechen Sie sie ab. Wenn Sie ein anderes lokales Modell verwenden, versuchen Sie das empfohlene `qwen3:8b`-Modell.
- Wenn ein unbekanntes Modell Tags oder andere Hilfstexte in seiner Antwort belässt, versuchen Sie `qwen3:8b` oder entfernen Sie diesen Text manuell, während Sie das Ergebnis überprüfen.
- Wenn das Ergebnis eine Bedingung, einen Aufruf, eine Konstante oder einen Typ unerwartet ändert, prüfen Sie es anhand des Low-Level-Listings. Bessere Lesbarkeit allein belegt kein gleichwertiges Verhalten.

Ollama und externe Modellanbieter sind Produkte Dritter. Ihre Verfügbarkeit, das Verhalten der Modelle und die Dienstrichtlinien sind unabhängig von VB Decompiler.

6 Öffnen und Speichern

6.1 Datei öffnen

Bevor Sie den Prozess der Dekompilierung starten, müssen Sie eine Datei zur Analyse auswählen. Zu diesem Zweck gibt es einen **Programmdatei öffnen (Open program)**-Bereich im Menü **Datei (File)**. **VB Decompiler** unterstützt das Öffnen von EXE/OCX/DLL-Dateien, die in **Visual Basic**-Versionen 5.0 oder 6.0 erstellt wurden, sowie Anwendungen, die in **Visual Studio .NET** kompiliert wurden. Im letzteren Fall ist die Quellensprache irrelevant, da Visual Studio .NET Anwendungen unabhängig von der ursprünglichen Sprache in ein einziges internes Format – **MS IL** – kompiliert.



Nach dem Öffnen der Datei überprüft der Decompiler, ob das Programm zu den angegebenen Entwicklungsumgebungen gehört. Wenn das geöffnete Programm in einer nicht unterstützten Umgebung entwickelt wurde (z. B. **Delphi** oder **Visual C++**), erscheint eine Meldung, die besagt, dass diese Umgebung nicht unterstützt wird und das Programm nicht dekompiliert werden kann. Falls diese Meldung erscheint, können Sie uns kontaktieren, und wir könnten möglicherweise dabei helfen, Ihr Programm zu dekompilieren. Wir verfügen über interne Entwicklungen im Bereich der Dekompilierung von **Delphi** und **FreePascal**-Anwendungen und können Decompilerdienste für solche Dateien als Service anbieten.

Wenn das geöffnete Programm mit einem **EXE-Packer**, Obfuskator oder Schutz versehen ist, wird der Decompiler anbieten, diese Datei zu entpacken. Diese Funktion steht nur in 32-Bit-Versionen von Windows zur Verfügung und wird ohne jegliche Garantien angeboten. Wenn das Entpacken möglich ist, wird die Datei gestartet, im Speicher entpackt, auf der Festplatte mit einem anderen Namen gespeichert und zum Dekompilieren geöffnet. **Achtung!** Versuchen Sie niemals, bösartige Programme oder anderen Code, der Ihr System schädigen könnten, mit dem integrierten Entpacker zu entpacken, da beim Entpacken das geöffnete Programm ausgeführt wird und seine Ausführung Ihr System oder Ihre Daten beschädigen kann.

Wichtiger Hinweis: sollten Sie unter keinen Umständen versuchen, Software ohne die erforderlichen Rechte zu dekompilieren oder zu entpacken, und wenn der Inhaber der Urheberrechte die Dekomprimierung und Analyse dieser Datei durch Lizenz untersagt. Dies kann gegen die Gesetze Ihres Landes verstoßen.

*Visual Studio .NET, Visual C++ und Visual Basic sind eingetragene Marken der Microsoft Corporation. Delphi ist eine eingetragene Marke von Embarcadero.

Weitere Möglichkeiten zum Öffnen von Dateien

Sie können eine Datei aus dem Explorer in das Fenster von VB Decompiler ziehen. Dateien mit einer unbekannten Erweiterung oder ohne Erweiterung können ebenfalls auf diese Weise übermittelt werden; die Erkennung hängt weiterhin vom tatsächlichen Inhalt der Datei ab. Unterstützte Windows **.lnk**-Verknüpfungen können auf ihre referenzierten Dateien aufgelöst werden. Gespeicherte VDD-Datenbanken können ebenfalls durch Drag-and-Drop oder über die Befehlszeile geöffnet werden.

6.2 Mit Analysedatenbanken arbeiten

Diese Funktion ist über den Bereich **Datenbank (Data Base)** im Menü **Datei (File)** verfügbar und dient dazu, die Ergebnisse der Dekompilierung in eine separate komprimierte Datei zu speichern. Dies kann nützlich sein, um zukünftig die Ladezeit der Dekompilierungsergebnisse zu verkürzen, ohne den Code erneut dekompilieren zu müssen.

Sollten Sie zur Analyse des Programms zurückkehren oder diese Ergebnisse mit einem Kollegen teilen wollen, ist eine erneute Dekompilierung nicht erforderlich. Im Fall einer vollständigen Dekompilierung (mit der **Schnelle Dekompilierung (Fast decompilation)** Option deaktiviert) kann dieser Prozess länger dauern und zusätzlichen Zeitbedarf verursachen. Es ist deutlich bequemer, den dekompilierten Code im Voraus in der **Database** zu speichern.

VB Decompiler verwendet ein internes komprimiertes Datenbankformat, das eine erhebliche Optimierung von des Speicherbedarfs ermöglicht. Zusätzlich besteht Kompatibilität mit Datenbanken, die von den meisten vorherigen Versionen von VB Decompiler gespeichert wurden, sodass Sie Informationen speichern können, ohne das Risiko einzugehen, dass eine solche Datenbank in der neuen Version des Produkts nicht geöffnet werden kann.

Öffnen und Übertragen von VDD-Datenbanken

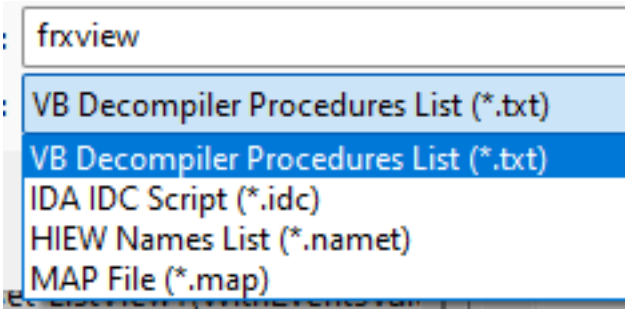
Gespeicherte Analysedatenbanken verwenden die **.vdd**-Erweiterung. Sie können eine VDD-Datenbank über den Dateioffnungsablauf öffnen, sie in das Fenster des Decompilers ziehen oder ihren Pfad über die Befehlszeile angeben.

Beim Auffinden der ursprünglichen analysierten Binärdatei prüft VB Decompiler sowohl seinen gespeicherten Pfad als auch das Verzeichnis, das die Datenbank enthält. Für die Zusammenarbeit sollten Sie die entsprechende Binärdatei neben der VDD-Datei belassen, wenn ihr ursprünglicher Pfad nicht verfügbar ist. Bewahren Sie die exakte analysierte Build-Version bei, damit Adressen und Dateibytes weiterhin übereinstimmen.

6.3 Prozedurliste speichern

Diese Operation wird durch Aufruf der Option **Prozeduren Liste speichern (Save procedures list)** im Menü **Datei (File)** ausgeführt und ist ein leistungsfähiges Werkzeug zur Analyse von Programmen, um VB Decompiler mit externen Analysewerkzeugen zu ergänzen.

Derzeit können Adressen, Funktionsnamen und in dem verarbeiteten Programm verwendete Prozeduren in mehreren Formaten gespeichert werden. Sie können jede dieser Optionen wählen, indem Sie den Dateityp im Standard-Speicherdialog angeben.



Speichern im TXT-Format ermöglicht das Speichern der Prozedurliste in einer einfachen Textdatei. Dies kann nützlich sein, um Prozeduren zwischen verschiedenen Versionen eines Programms zu vergleichen, Änderungen von Adressen zu erkennen und so weiter.

Speichern als IDC-Skript ermöglicht das Laden von Funktions- und Prozeduradressen in den **IDA Pro**-Disassembler. In Fällen, in denen das analysierte Programm in **Native Code** kompiliert wurde, kann diese Funktion sehr nützlich sein.

Speichern im NAMET-Format ermöglicht die Vereinfachung der Analyse mit dem Dienstprogramm **HIEW**. Kopieren Sie die erhaltene Datei in den Ordner des analysierten Programms. Öffnen Sie dann das Programm in HIEW und drücken Sie die Taste F12. Ein Fenster mit einer leeren Prozedurliste erscheint. Drücken Sie erneut die Taste F12, wenn die Adresse der NAMET-Datei der erstellten entspricht – drücken Sie einfach Enter. Die Adressen der Prozeduren werden in HIEW geladen, wodurch Sie durch Methoden navigieren können, indem Sie die F12-Taste drücken.

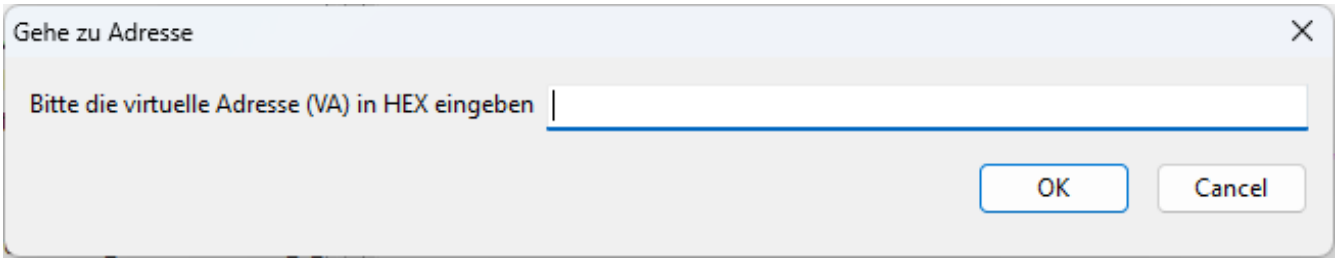
Speichern im MAP-Format generiert eine Standard-Map-Datei, die normalerweise von Compilern erstellt wird. Diese Datei kann mit vielen Debuggern und Disassemblern verwendet werden, um den Start von Programmfunktionen zu markieren. Da Programme in Visual Basic häufig von Drittanbieter-Software falsch disassembliert werden, vereinfacht die von **VB Decompiler** erzeugte MAP-Datei die Analyse erheblich. Dieses Format wird durch Plugins in verschiedenen Produkten wie **OllyDbg**, **IDA Pro** und anderen unterstützt.

7 Programmfunktionen

7.1 Zu einer Adresse springen

Zu einer virtuellen Adresse springen

Wählen Sie **Extras > Gehe zu virtuellen Adresse (Tools > Jump to virtual address)** aus oder drücken Sie **Ctrl+G**. Geben Sie die virtuelle Adresse des zu untersuchenden Codes ein. Sie können auch eine Adresse in der Auflistung auswählen und Strg+G aufrufen. Der Decompiler öffnet den mit dieser Adresse verknüpften Code.



Eine virtuelle Adresse (VA) identifiziert einen Speicherort im Adressraum des ausführbaren Bildes und enthält dessen Basisadresse. Ein Dateioffset identifiziert eine Byte-Position in der Datei auf dem Datenträger. Ersetzen Sie keines durch das andere: Der HEX-Editor unterstützt beide Navigationsformen.

Zurückkehren über die Navigationshistorie

Verwenden Sie die Pfeilschaltflächen oberhalb des Baums, um rückwärts und vorwärts durch zuvor angezeigte Prozeduren zu navigieren. **Esc** kehrt zur vorherigen Prozedur zurück. Die Maus-Tasten Zurück und Vorwärts nutzen ebenfalls diese Historie, wobei die Position innerhalb der Prozedur beibehalten wird.

Einen Speicherort über seinen Baumweg teilen

Klicken Sie im Lösungsforscher (Solution explorer) mit der rechten Maustaste auf das erforderliche Objekt und wählen Sie **Sektionsname kopieren (Copy the name of section)** aus, oder drücken Sie **Ctrl+C**, während der Baum den Fokus hat. Dies kopiert den vollständigen Pfad zum ausgewählten Knoten.

Um diesen Speicherort erneut aufzurufen, wählen Sie **Zum Abschnitt (Jump to the section)** im Kontextmenü des Baums aus oder drücken Sie **G** im Baum und fügen Sie den kopierten Pfad in das Dialogfeld ein. Dies ist nützlich, wenn eine Untersuchung an einen anderen Analysten übergeben wird oder man zu einem großen Projekt zurückkehrt. Verwenden Sie den Pfad mit dem entsprechenden analysierten Datei; ein Pfad aus einem anderen Build kann möglicherweise nicht mehr denselben Code identifizieren.

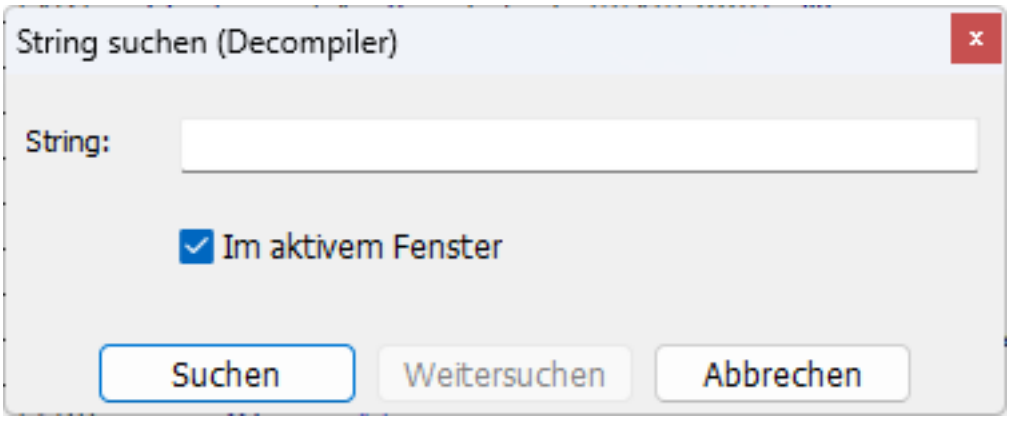
7.2 Zeichenfolgen suchen

Prozedur anhand ihres Namens suchen

Wählen Sie **Extras > Prozedursuche (Tools > Find procedure)** aus oder drücken Sie **Ctrl+T**. Geben Sie einen vollständigen Prozedurnamen oder einen charakteristischen Teil davon ein. Die Suche umfasst den Projektbaum, sodass Sie nicht wissen müssen, in welchem Modul oder welcher Klasse die Prozedur enthalten ist. Wenn mehrere Namen übereinstimmen, enthält die Ergebnisliste alle übereinstimmenden Einträge; doppelklicken Sie auf den gewünschten Eintrag, um ihn zu öffnen.

Suche nach angezeigtem Text oder im gesamten Projekt

Wählen Sie **Extras > Find string (Tools > Find string)** aus oder drücken Sie **Ctrl+F**, um das Dialogfeld für die Textsuche zu öffnen. Wenn **Im aktivem Fenster (In active window)** ausgewählt ist, gilt die Suche für den aktuell angezeigten Code oder Formulardaten. Deaktivieren Sie dieses Kontrollkästchen, um im gesamten dekompilierten Projekt zu suchen.



Geben Sie den Text in **String** ein und wählen Sie **Suchen (Find)** aus. Verwenden Sie **Weitersuchen (Find next)**, um die Suche nach demselben Text fortzusetzen. **Abbrechen (Cancel)** schließt das Dialogfeld.

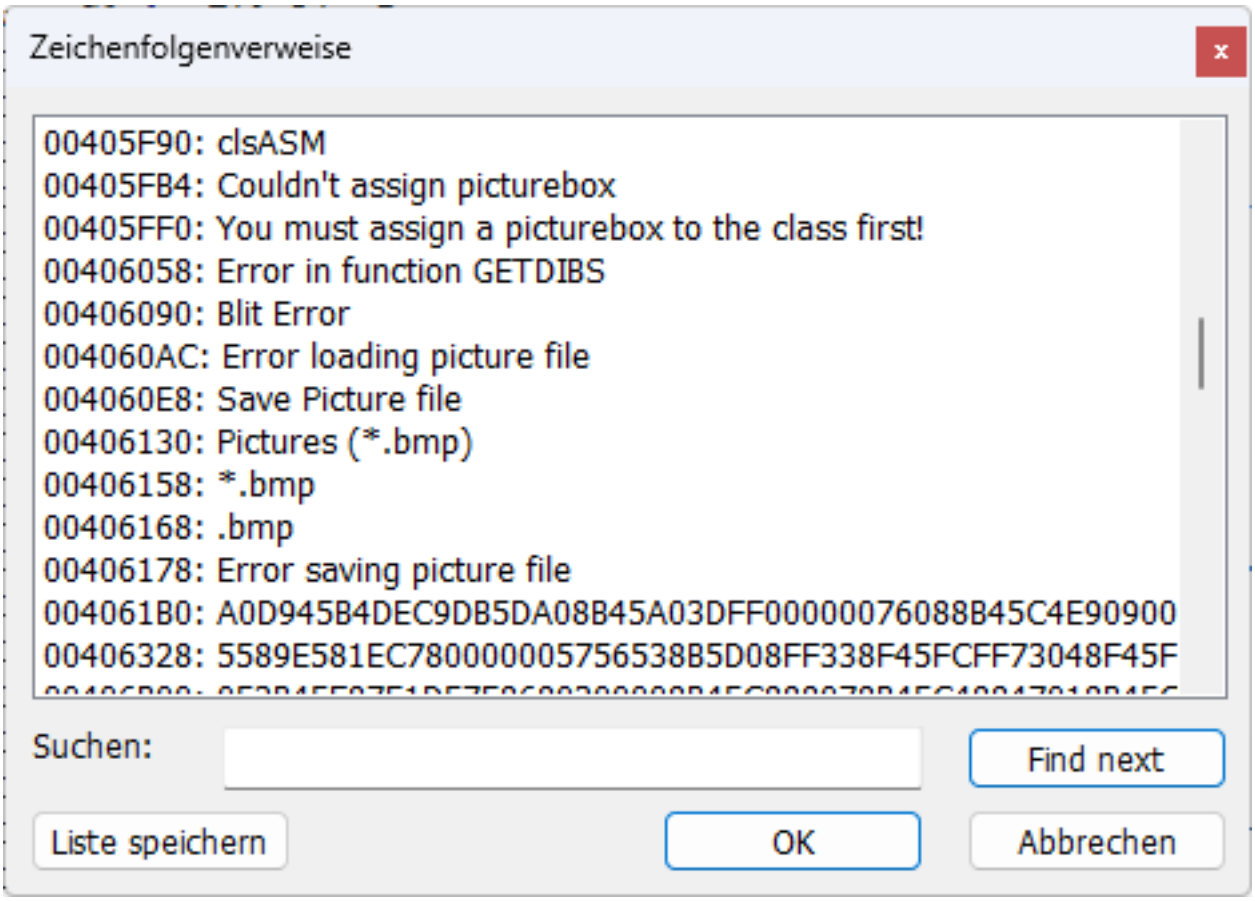
Die Suche in allen dekompilierten Texten erfordert eine vollständige Dekompilierung. Deaktivieren Sie **Schnelle Dekompilierung (Fast decompilation)** unter Erweiterte Funktionen und verarbeiten Sie die Datei zunächst erneut. Die Suche nach Prozedurnamen durchsucht hingegen nur die Projektstruktur.

Suche nach Zeichenfolgenreferenzen

Falls Sie Literalzeichenfolgen und den Code benötigen, der diese verwendet, öffnen Sie Zeichenfolgenreferenzen. Das Feld **Suchen (Search)** und die Schaltfläche **Weitersuchen** durchsuchen die Zeichenfolgenliste. Zeichenfolgenreferenzen können durch eine vorläufige Disassemblierung im Modus für schnelle Dekompilierung gesammelt werden; dafür ist kein vollständig dekompilierter Prozedurcode erforderlich.

7.3 Zeichenfolgenreferenzen

Im Code eines Programms begegnen Sie häufig Zeichenketten-Daten (Registrierungsschlüssel, Dateinamen, Fehlermeldungen usw.). Es kann sehr nützlich sein, die Liste der von Prozeduren und Funktionen in dekompilierten Anwendungen verwendeten Zeichenketten zu sehen. Genau dafür ist unsere **Zeichenfolgenverweise (String references)**-Funktion konzipiert, die über das Menü **Extras (Tools)** zugänglich ist. Um sie zu verwenden, navigieren Sie einfach zu **Extras** -> **Zeichenfolgenverweise**, und wenn Ihr Programm Zeichenketten-Daten enthält, werden diese wie unten dargestellt angezeigt:

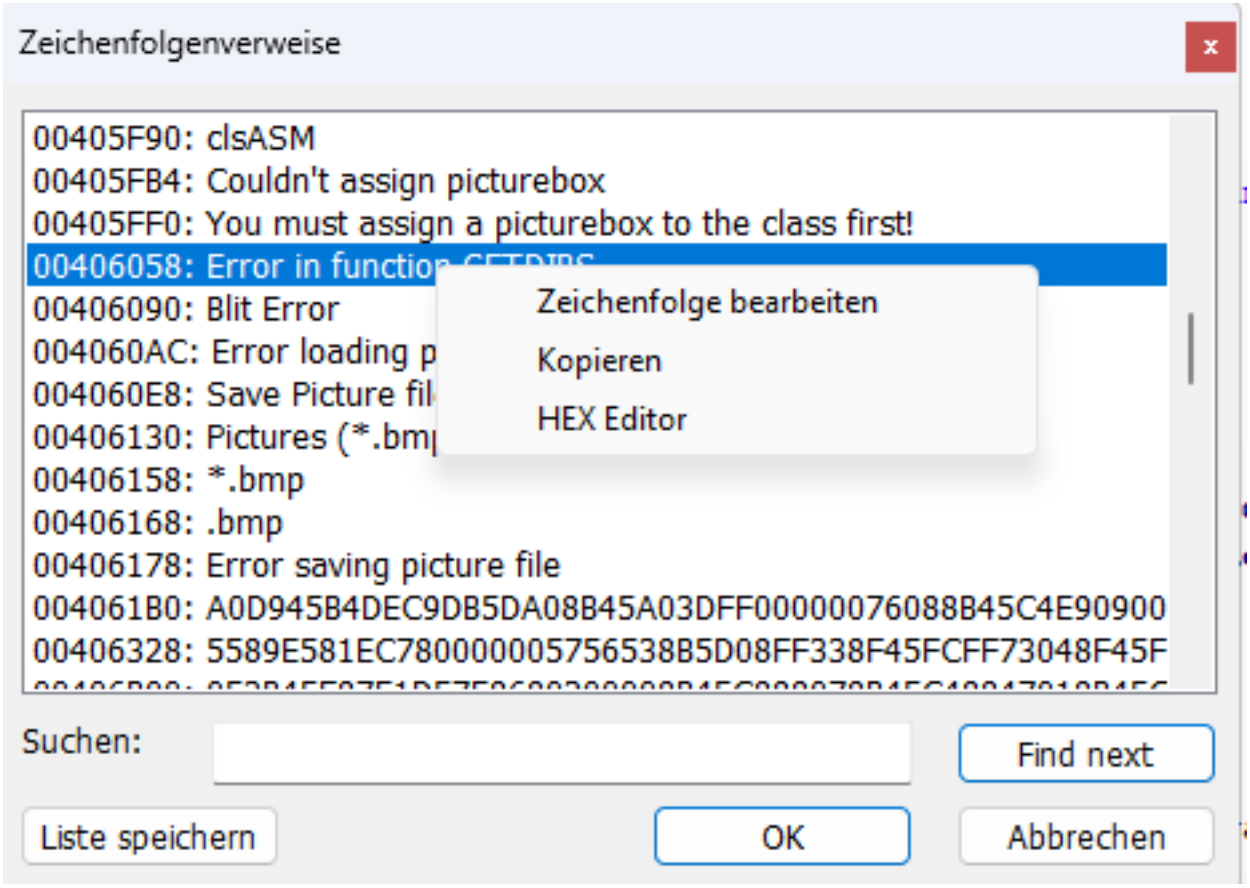


Finden Sie die gewünschte Zeile in der Liste und doppelklicken Sie darauf. Der Codeabschnitt, der diese Zeile enthält, wird lokalisiert, und die entsprechende Prozedur wird im Code-Fenster geöffnet. Ein weiterer Doppelklick auf die Zeile setzt die Suche nach dieser Zeile innerhalb des Programmcodes fort. Klicken Sie auf 'OK', um die hervorgehobene Zeile im Code zu finden und das Dialogfeld zur Auswahl der Zeile zu schließen. Abbrechen schließt einfach das Dialogfeld.

Die Funktion zum Anzeigen von **Zeichenfolgenverweise** ist nur in der lizenzierten Version der Software verfügbar und für die **Lite**-Version nicht zugänglich.

Wichtiger Hinweis: Zeichenketten werden nicht nur für das aktive Modul wie in Programmen der Version 1.x angezeigt, sondern für das gesamte Programm. Diese Änderung wurde vorgenommen, um die Analyse von Programmen mit einer großen Anzahl von Formularen und Modulen zu erleichtern.

VB Decompiler unterstützt auch das Bearbeiten von Zeichenketten. Um das Dialogfeld zum Bearbeiten der Zeichenkette aufzurufen, klicken Sie mit der rechten Maustaste auf die Zeichenkette, die Sie ändern möchten, und wählen Sie **Zeichenfolge bearbeiten (Edit string)** aus dem Kontextmenü, das erscheint. Ein Dialogfeld zum Eingeben einer neuen Zeichenkette wird angezeigt. Nach dem Ändern der Zeichenkette klicken Sie auf **Ok**, und die Zeichenkette wird direkt in der bearbeiteten Datei geändert. Wir empfehlen, eine Sicherungskopie der zu untersuchenden Datei vorher anzulegen, da während dieser Aktion keine Sicherung erstellt wird und diese Operation nicht rückgängig gemacht werden kann.



Wichtiger Hinweis: Denken Sie beim Bearbeiten von Zeichenketten daran, dass die Länge der neuen Zeichenkette die Länge der alten nicht überschreiten sollte. Dies hängt mit dem Format von binären Anwendungsdateien zusammen, bei denen kein Platz für das Einfügen einer längeren Zeichenkette reserviert ist.

Sie können die Zeichenkette auch im HEX-Editor anzeigen und teilweise bearbeiten. Dazu klicken Sie mit der rechten Maustaste auf die Zeichenkette und wählen HEX Editor aus dem Menü, das erscheint. Im HEX-Editor gibt es Funktionen zum Rückgängigmachen und Wiederherstellen von Änderungen, was das Experimentieren beim Ändern von Zeichenketten-Daten von Anwendungen vereinfacht.

Das Fenster mit der Liste der Zeichenketten verfügt auch über ein Textfeld namens **Suchen (Search)** und eine **Weitersuchen (Find next)**-Schaltfläche. Diese Funktionalität ist nützlich, um Zeichenketten innerhalb der Liste zu finden, insbesondere bei großen Projekten, bei denen eine manuelle visuelle Suche durch die Liste extrem schwierig ist.

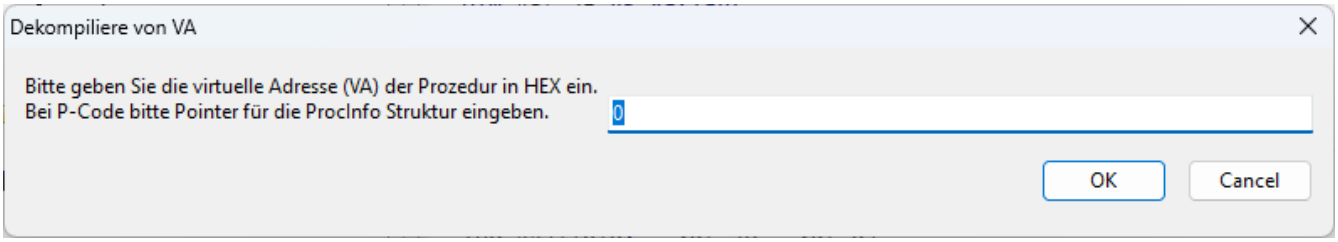
Schnelle Dekompilierung und große Zeichenkettenlisten

Zeichenfolgenreferenzen können durch vorläufige Disassemblierung gesammelt werden, einschließlich im **Schnelle Dekompilierung (Fast decompilation)**-Modus. Dies unterscheidet sich von der Suche nach vollständig dekompiliertem Text. Version 26.2 erweitert das Dialogfeld über 32,000-Referenzen hinaus und verbessert die Speicherverwaltung für große Listen.

Für .NET-Assemblys können Benutzerzeichenketten aus dem #US-Heap mit ihren Positionen in der Datei überprüft werden. Die Liste stellt Zeichenketten dar, die der statischen Analyse zugänglich sind; verschlüsselte oder dynamisch generierte Zeichenketten können eine separate Untersuchung erfordern.

7.4 Dekompilierung ab einer Adresse

Im Menü **Extras (Tools)** befindet sich eine sehr nützliche Funktion - **Dekompiliere von VA (Decompile from VA)** (Code ab einer angegebenen virtuellen Adresse dekompilieren). Das Hauptmerkmal dieser Funktion ist, dass sie nicht erfordert, dass die Datei, die dekompiliert wird, eine korrekte Struktur aufweist und mit Standardverfahren verarbeitet wurde. Natürlich hat der Code in diesem Fall eine Reihe von Einschränkungen. Zum Beispiel, wenn importierte Funktionen oder OLE-Objekte nicht vorverarbeitet wurden, ist der Code an diesen Stellen uninformativ.



Um einen bestimmten Codeabschnitt innerhalb einer Datei zu dekompilieren, müssen Sie einfach die virtuelle Adresse dieses Codes angeben (wenn wir über **Native Code** oder **.NET-Assembly** sprechen), oder eine **ProcInfo**-Strukturreferenz bereitstellen (wenn wir über klassische **Visual Basic P-Code** sprechen). Die Einzigartigkeit der Arbeit mit solchen **P-Code** besteht darin, dass die **ProcInfo**-Struktur ein Array von Daten enthält, das vom Code verwendet wird. Ohne dieses Array ist es unmöglich zu bestimmen, mit welchen Funktionen der Code arbeitet, auf welche Strings er sich bezieht usw. Im Fall der Arbeit mit **.NET-Assemblys** bietet der Decompiler nach Eingabe der Adresse eine Wahl zwischen dem Dekompilieren des Codes als **MS IL**-Zwischencode oder dem Disassemblieren dieses Codes. Diese Option ist aufgrund der Tatsache, dass .NET-Assemblys sowohl MS IL-Zwischencode (**verwalteter Code**) als auch Maschinencode (sogenannter **nicht verwalteter Code**) enthalten können.

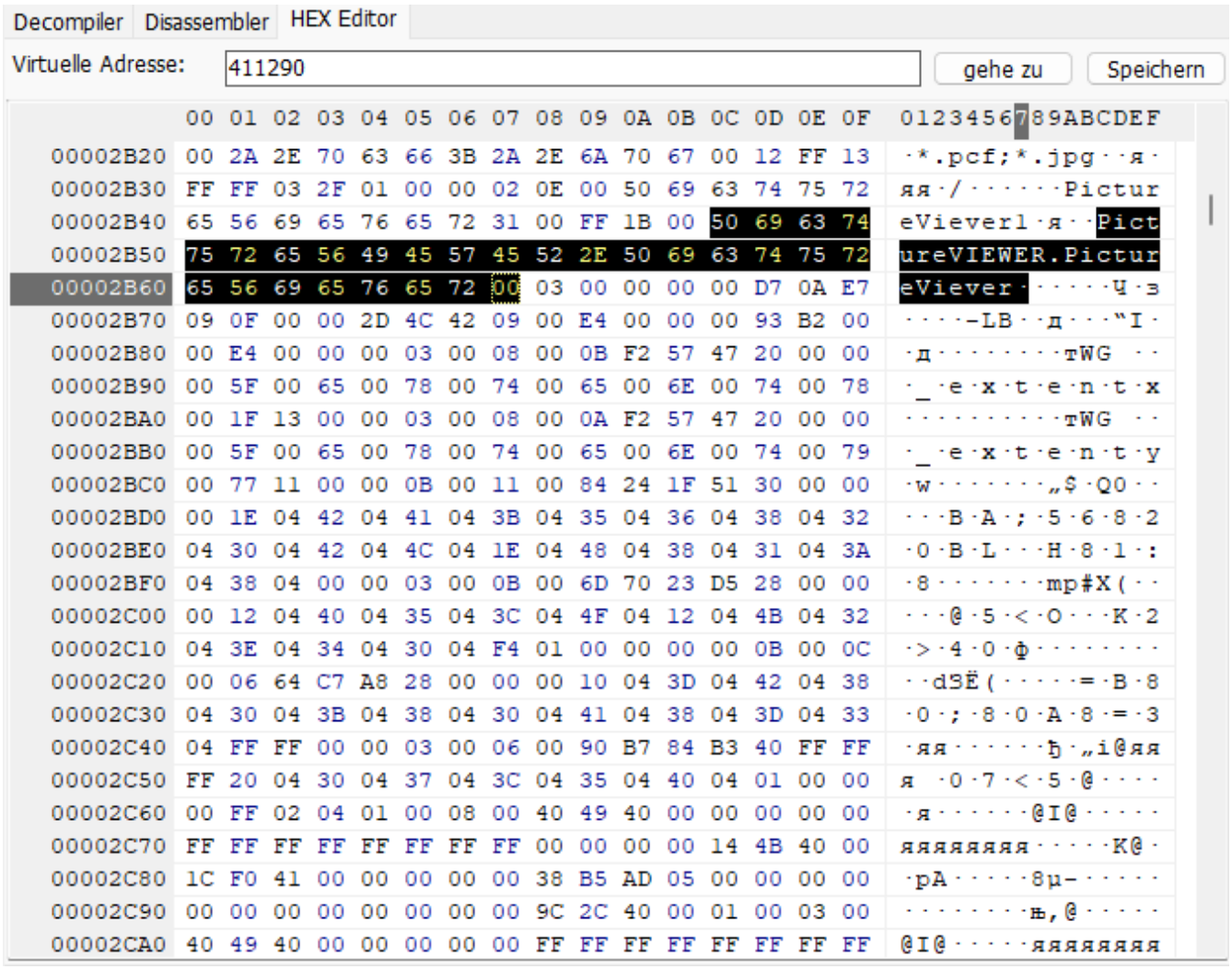
Wenn die Datei bereits im Decompiler geöffnet ist, müssen Sie einfach die **Dekompiliere von VA**-Funktion aufrufen und die Adresse für die Verarbeitung angeben. Wenn die Datei noch nicht geöffnet wurde, wird dies automatisch vorgeschlagen, und die Datei wird im vereinfachten Modus geöffnet.

Die **Dekompiliere von VA**-Funktion ist besonders nützlich, wenn der gewünschte Code nicht automatisch dekompiliert wird oder wenn die Datei teilweise beschädigt ist, sodass sie mit dem normalen Analyseverfahren nicht verarbeitet werden kann. Natürlich sollten Sie verstehen, dass das Ergebnis der Dekompilierung in diesem Fall schlechter ist als bei einer Standard-Dekompilierung.

7.5 Daten bearbeiten

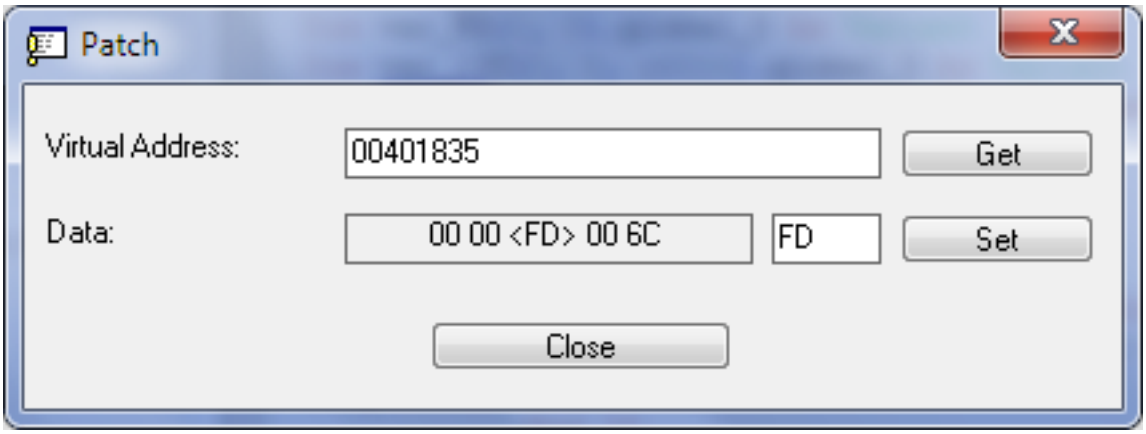
Ab Version 9.4 verfügt **VB Decompiler** über ein modernes und benutzerfreundliches Werkzeug zum Bearbeiten von Binärdaten innerhalb von Dateien – einen vollwertigen **HEX Editor**. Er ist aus dem Hauptfenster des Programms zugänglich, neben dekompiliertem Code im Reiter **HEX Editor** für jede Prozedur und jedes Formular.

Der **HEX Editor** ermöglicht es Ihnen, die Datei Byte für Byte sowie in Blöcken zu bearbeiten. Um zur gewünschten Adresse innerhalb der untersuchten Datei zu navigieren, können Sie entweder virtuelle Adressen (einschließlich **Image Base**) oder Dateioffsets (eine reguläre Adresse, die den Offset der benötigten Daten relativ zum Anfang der Datei angibt) verwenden.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
00002B20	00	2A	2E	70	63	66	3B	2A	2E	6A	70	67	00	12	FF	13	·*.pcf;*.jpg··я·
00002B30	FF	FF	03	2F	01	00	00	02	0E	00	50	69	63	74	75	72	яя·/·····Pictur
00002B40	65	56	69	65	76	65	72	31	00	FF	1B	00	50	69	63	74	eVieverl·я··Pict
00002B50	75	72	65	56	49	45	57	45	52	2E	50	69	63	74	75	72	ureVIEWER.Pictur
00002B60	65	56	69	65	76	65	72	00	03	00	00	00	00	D7	0A	E7	eViever·····Ч·з
00002B70	09	0F	00	00	2D	4C	42	09	00	E4	00	00	00	93	B2	00	····-LB··д···"I·
00002B80	00	E4	00	00	00	03	00	08	00	0B	F2	57	47	20	00	00	·д·····тWG ··
00002B90	00	5F	00	65	00	78	00	74	00	65	00	6E	00	74	00	78	·_·e·x·t·e·n·t·x
00002BA0	00	1F	13	00	00	03	00	08	00	0A	F2	57	47	20	00	00	·····тWG ··
00002BB0	00	5F	00	65	00	78	00	74	00	65	00	6E	00	74	00	79	·_·e·x·t·e·n·t·y
00002BC0	00	77	11	00	00	0B	00	11	00	84	24	1F	51	30	00	00	·w·····„$·Q0··
00002BD0	00	1E	04	42	04	41	04	3B	04	35	04	36	04	38	04	32	···B·A·;·5·6·8·2
00002BE0	04	30	04	42	04	4C	04	1E	04	48	04	38	04	31	04	3A	·0·B·L···H·8·1·:
00002BF0	04	38	04	00	00	03	00	0B	00	6D	70	23	D5	28	00	00	·8·····mp#X(··
00002C00	00	12	04	40	04	35	04	3C	04	4F	04	12	04	4B	04	32	···@·5·<·O···K·2
00002C10	04	3E	04	34	04	30	04	F4	01	00	00	00	00	0B	00	0C	·>·4·0·ф·····
00002C20	00	06	64	C7	A8	28	00	00	00	10	04	3D	04	42	04	38	··dЗЁ(·····=·B·8
00002C30	04	30	04	3B	04	38	04	30	04	41	04	38	04	3D	04	33	·0·;·8·0·A·8·=·3
00002C40	04	FF	FF	00	00	03	00	06	00	90	B7	84	B3	40	FF	FF	·яя·····ђ·„і@яя
00002C50	FF	20	04	30	04	37	04	3C	04	35	04	40	04	01	00	00	я ·0·7·<·5·@····
00002C60	00	FF	02	04	01	00	08	00	40	49	40	00	00	00	00	00	·я·····@I@·····
00002C70	FF	FF	FF	FF	FF	FF	FF	FF	00	00	00	00	14	4B	40	00	яяяяяяяя·····K@·
00002C80	1C	F0	41	00	00	00	00	00	38	B5	AD	05	00	00	00	00	·pA·····8µ-·····
00002C90	00	00	00	00	00	00	00	00	9C	2C	40	00	01	00	03	00	·····ќ,@·····
00002CA0	40	49	40	00	00	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	@I@·····яяяяяяяя

In früheren Versionen des Decompilers (vor Version 9.4) wurde die Funktion zum Bearbeiten von Dateien mit einem primitiveren Werkzeug angewendet: Data Patcher. Es funktionierte wie folgt. Im Menü **Extras (Tools)** -> **Patche Daten (Patch data)** gab es ein Instrument namens Patcher, das in Fällen verwendet werden konnte, in denen nach der Analyse des Programmcodes einige Teile geändert werden mussten. Um die benötigten Bytes ohne externe Werkzeuge zu ändern, verfügte **VB Decompiler** über eingebaute Funktionalität. Das Fenster des Patchers sah wie folgt aus:



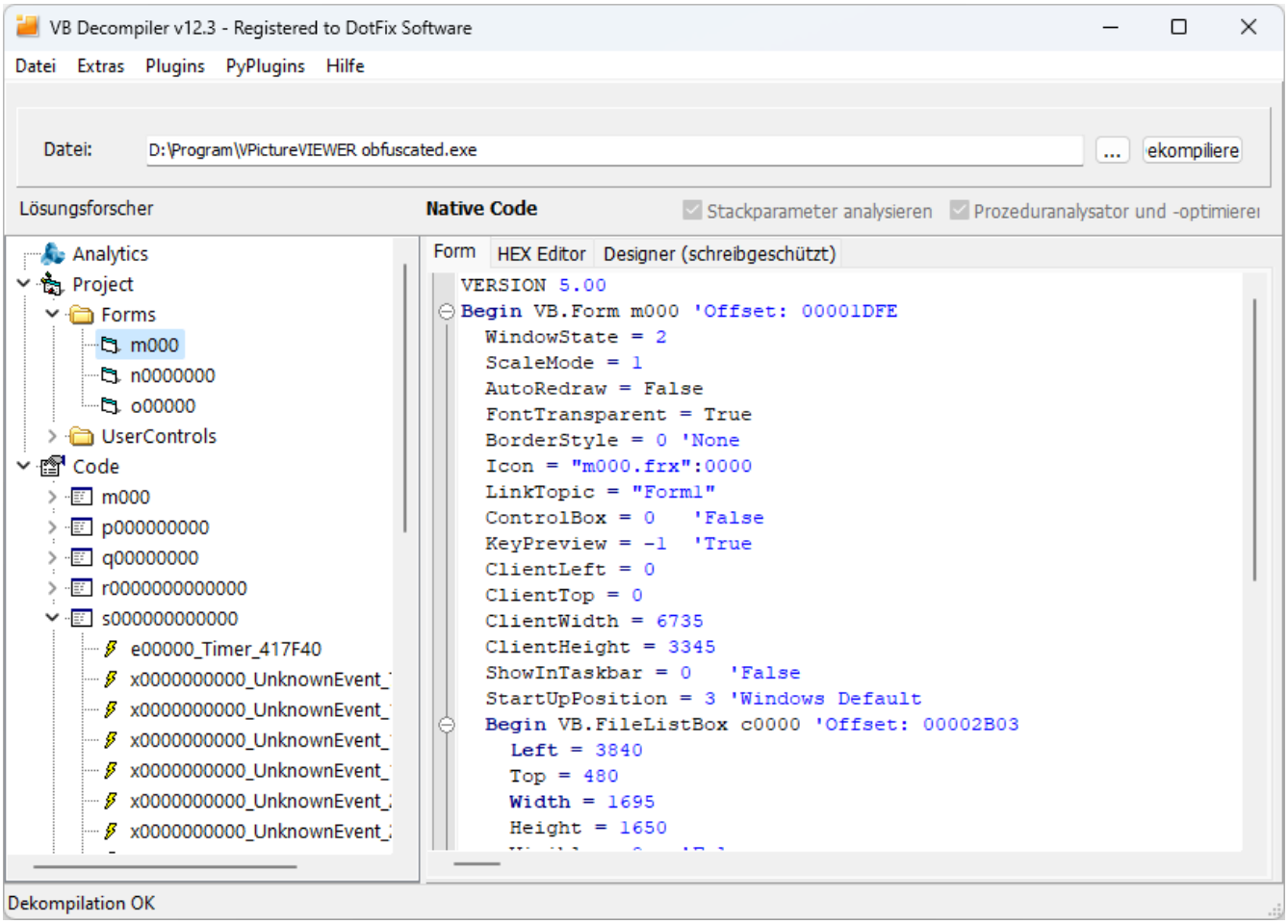
Die Arbeit mit dem Patcher war extrem einfach. Im Feld **Virtuelle Adresse (Virtual Address)** gaben Sie die Adresse zum Bearbeiten eines Bytes ein, drückten dann entweder **Enter** oder klickten auf **Ermitteln (Get)**. Die Datenbytes in der Nähe der ausgewählten Adresse wurden im Feld Data angezeigt, und das ausgewählte Byte konnte geändert werden. Anschließend mussten Sie auf **Setzen (Set)** klicken, und das eingegebene Byte wurde geändert.

Wenn in Programmeinstellungen die Option **Schnelle Dekompilierung (Fast decompilation)** aktiviert und die Option **Dekompilierten Code zwischenspeichern (Cache decompiled code)** deaktiviert war, dann wurden alle Änderungen in der Funktion angezeigt, in der die Bytes modifiziert wurden. Um dies zu tun, genügte es, dieselbe Funktion durch Doppelklicken auf ihren Namen im Baum wiederzuöffnen.

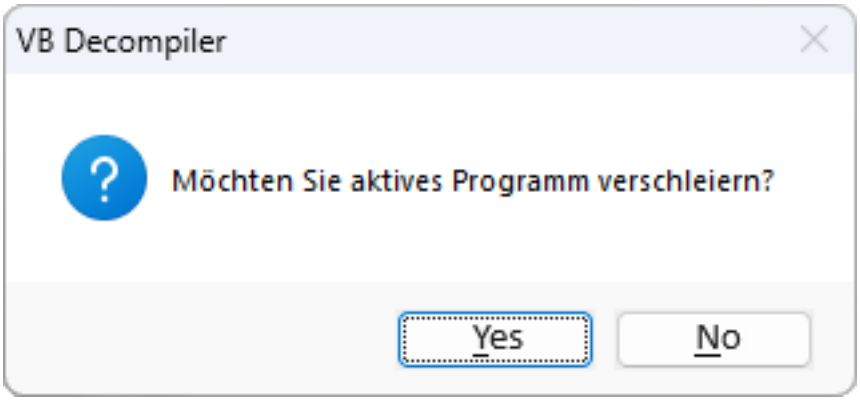
Wichtiger Hinweis: Diese Funktionalität ist ausschließlich zum Bearbeiten Ihrer eigenen Programmdateien oder solcher Dateien gedacht, für die Sie die entsprechenden Rechte besitzen, sowie zur Erforschung von bösartigem Code in Antivirenlabors. Bevor Sie den Code anderer Personen ändern – überprüfen Sie unbedingt die Lizenzvereinbarung hinsichtlich des Vorhandenseins eines solchen Rechts.

7.6 Obfuskation

Wenn Sie Programme in **Visual Basic 5.0/6.0** entwickeln, können Sie deren Schutz vor Dekompilierung erhöhen. Um dies zu erreichen, enthält der Decompiler einen **Obfuscator**, der alle Funktions-, Formular-, Modul- und Objektnamen aus der Binärdatei Ihres Programms sowie verschiedene Signaturen entfernen kann. Dies erschwert die Analyse des Programms, unabhängig davon, ob der bereitgestellte Decompiler oder andere Lösungen verwendet werden. Darüber hinaus werden die meisten Drittanbieterprodukte diese Datei insgesamt nicht dekompilieren.



Um diesen Schutz anzuwenden, öffnen Sie Ihr Projekt im Decompiler und wählen **Projekt verschleiern (Obfuscate) (Obfuscate project)** aus dem Menü **Extras (Tools)**. Dieser Prozess dauert nicht viel Zeit, daher empfehlen wir, Ihre Projekte vor der Veröffentlichung zu obfuskieren.



Diese Funktion ist auch in unserer umfassenden Lösung zum Schutz von Software integriert - **DotFix NiceProtect**, die von www.niceprotect.com heruntergeladen werden kann.

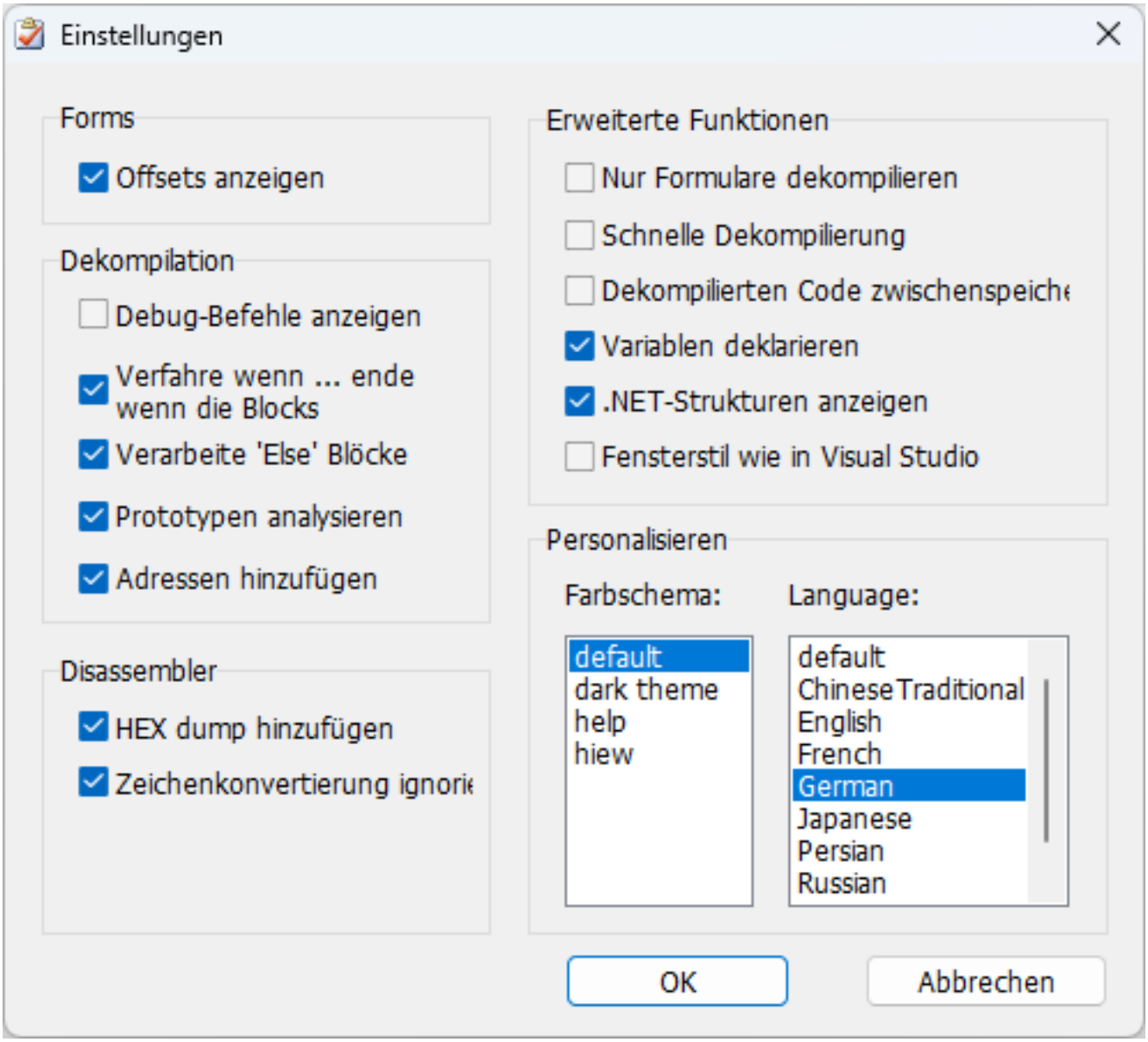
Obfuskation steht nur Benutzern mit einer Business-Lizenz des **VB Decompiler** zur Verfügung.

Obfuskationseinschränkungen

8 Programmeinstellungen

8.1 Programmeinstellungen

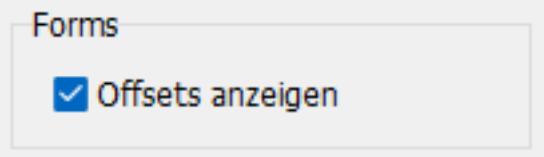
Um die Programmeinstellungen zu ändern, gehen Sie zum Menü **Extras (Tools)** > **Einstellungen (Options)**. Im daraufhin geöffneten Dialog sehen Sie folgende Optionen:



Formulare
Dekompilierung
Disassembler
Farbschema Erweiterte Funktionen Sprachauswahl

8.2 Formulare

Sie können die folgenden Dekompilierungseinstellungen für Formulare ändern:



In diesem Bereich der Einstellungen können Sie die Anzeige von Dateioffsets neben Formularobjekten aktivieren. Dies kann beispielsweise erforderlich sein, wenn ein Programm in einer anderen Sprache lokalisiert wird, ohne Zugriff auf den Quellcode. **Offsets anzeigen (Show offsets)** ermöglicht es Ihnen, den Ort innerhalb des Programms zu finden, an dem sich diese Objekte befinden, sowie deren Namen und Eigenschaften.

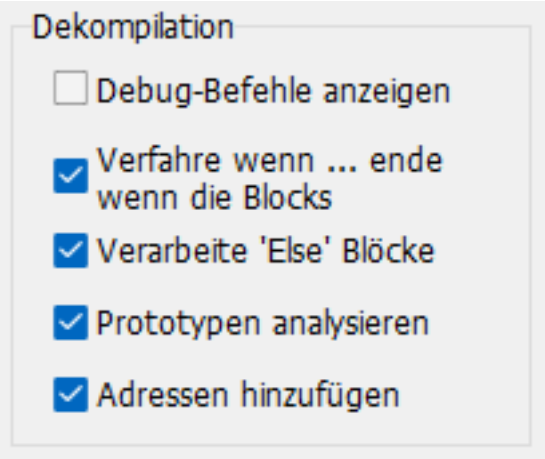
Anschließend können Sie mit einem HEX-Editor diese Werte ändern. Beachten Sie, dass es empfohlen wird, diese Option nicht zu aktivieren, wenn Ihr Wissen auf dem Niveau eines grundlegenden Benutzers beschränkt ist und Sie mit den inneren Abläufen von Binärdateien nicht vertraut sind. Jegliche falsche Änderungen an der Datei machen sie unbrauchbar, und unnötige Adressen werden Ihnen lediglich hinderlich sein.

Visuelle Vorschau und Ereignisnavigation

Formularoptionen sind vom **Designer (schreibgeschützt) (Designer (read only))**-Viewer getrennt. Siehe Visuelle Formularrekonstruktion für klassische VB-Formulare, Windows Forms, WPF/BAML-Unterstützung, Ereignis-Handler-Navigation und die Einschränkungen jeder Schnittstellentechnologie.

8.3 Dekompilierung

Öffnen Sie **Extras > Einstellungen (Tools > Options)**, um die Dekompilierung zu konfigurieren. Beim interaktiven Vergleich von Einstellungen verwenden Sie Schnelle Dekompilierung ohne Caching; andernfalls verarbeiten Sie die Datei erneut nach Änderung einer Option.



Debug-Befehle anzeigen (Show debug commands)

Für klassisches VB P-Code umfasst dies Compiler- und Debugger-Operationen, die sonst aus der hochsprachlichen Auflistung ausgeschlossen würden. Aktivieren Sie diese Option bei der Inspektion der zugrunde liegenden Instruktionssequenz anstelle nur der rekonstruierten Programmlogik.

Verfahre wenn ... ende wenn die Blocks (Process If ... End If blocks (Native Code))

Rekonstruiert erkannte bedingte Sprünge in Native Code als `If...End If`-Blöcke. Deaktivieren Sie diese Option, wenn Sie das ursprüngliche Verzweigungsmuster inspizieren oder eine Prozedur untersuchen müssen, deren Struktur fehlerhaft erscheint.

Verarbeite 'Else' Blöcke (Process 'Else' blocks)

Verwendet zusätzliche Heuristiken, um Else-Zweige von verschachtelten Bedingungen und GoTo-Übergängen zu unterscheiden. Bei Native Code muss dafür auch die Verarbeitung von If-Blöcken aktiviert sein. Lassen Sie diese Option für normales Lesen aktiviert; vergleichen Sie sie mit deaktivierter Einstellung bei unklaren rekonstruierten Zweigen.

Prototypen analysieren (Analyze prototypes)

Führt eine vorläufige heuristische Analyse von Prozedurargumenten und Rückgabewerten durch. Dies ist besonders nützlich im Modus für schnelle Dekompilierung: Private Funktionen und Prozeduren in Standardmodulen können fehlende explizite Signatur-Metadaten aufweisen. Die vorläufige Analyse benötigt zusätzliche Zeit, vermeidet aber die Behandlung eines Aufrufs ausschließlich gemäß dem derzeit bekannten Stapelinhalt. Eine heuristisch ermittelte Signatur bleibt eine Annahme, insbesondere wenn Metadaten, externe Typen oder Code unvollständig sind.

Adressen hinzufügen (Add Addresses)

Zeigt virtuelle Adressen neben rekonstruierten Anweisungen an. Lassen Sie diese Option aktiviert, um Code mit Disassemblierung, Dateibytes und Ergebnissen anderer Tools zu korrelieren. Für **Kopieren Sie zu dem Disassembler (Copy to disassembler)** ist dies zudem erforderlich, um hochsprachliche Anweisungen mit ihren niedrigsprachlichen Gegenstücken zu verschmelzen. Der Disassembler zeigt Adressen unabhängig von dieser Einstellung immer an.

Verständnis der Native Code-Rekonstruktion

Die Native-Code-Engine erkennt unterstützte `Select Case`-Muster, mehrdimensionale Array-Zugriffe, COM-Aufrufe und VB-Runtime-Operationen. Fehlerbehandlung kann als `On Error Resume Next`, `On Error GoTo` und `On Error GoTo 0` rekonstruiert werden. Compiler-Optimierung und fehlende Typinformationen können das Ergebnis weiterhin begrenzen; inspizieren Sie das Disassemblierung, wenn die hochsprachliche Ausgabe unvollständig ist.

Version 26.2 erweiterte Variant-Typisierung, mehrdimensionale SafeArray-Lese- und Schreiboperationen sowie `VarPtr`-Rekonstruktion für Array-Elemente. Die Ausgabe kann eine Array-Referenz als `SafeArray` identifizieren, was die von der Analyse-Engine erkannte Laufzeitstruktur widerspiegelt. Getrennte Zustandsverfolgung in Verzweigungen und Analyse von Temporärvariablen verbessern zudem die Rekonstruktion komplexer If/Else-Codes. Version 26.5 verbessert weiterhin Select Case und VB5/VB6-Runtime-Funktionsbehandlung.

COM- und ActiveX-Typinformationen

VB Decompiler verwendet zunächst seine integrierte Wissensdatenbank für bekannte Schnittstellen. Für ein unbekanntes externes COM-Objekt kann es Bibliotheks-, Klassen- und Schnittstellen-Identifikatoren über die Windows-Registrierung auflösen und die entsprechende DLL/OCX-Typbibliothek parsen. Dies liefert Methoden- und Eigenschaftssignaturen, einschließlich zurückgegebener Schnittstellentypen, für early-bound und unterstützte late-bound-Aufrufe.

Fehlt eine erforderliche Typbibliothek oder kann ihre Registrierung nicht aufgelöst werden, können einige Aufrufe unidentifiziert bleiben, beispielsweise als `UnkVCall`. Fehlende Abhängigkeiten sind daher relevant beim Vergleich von Ergebnissen zwischen Analyse-Maschinen. Der erweiterte IID-Wissensbestand umfasst Office-Automatisierung, ADO/DAO, XML und HTTP, Skripting, Shell, Messaging und Verzeichnisdienst-Schnittstellen; dies impliziert nicht, dass jeder mögliche COM-Aufruf aufgelöst werden kann.

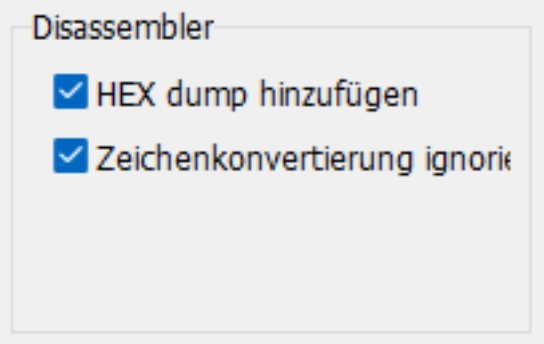
In manchen VB-Binärdateien bleiben nur wenige UDT-Metadaten erhalten. Der Decompiler kann Informationen wie die Größe einer Struktur und die Anzahl der String-Felder wiederherstellen, ohne das vollständige ursprüngliche Feldlayout zu rekonstruieren. Interpretieren Sie eine teilweise Deklaration nicht als vollständige Definition im ursprünglichen Quellcode.

.NET-Rekonstruktion und Optimierung

Unterstütztes IL wird als C#-ähnliche Ausdrücke und Steuerfluss rekonstruiert. Die **Prozeduranalysator und -optimierer (Procedure analyzer and optimizer)**-Steuerung im Hauptfenster kann unterstützte Muster für Schleifen, Bedingungen, Operatoren und temporäre Variablen vereinfachen. Prüfen Sie IL und Metadaten, wenn ein Fragment nicht unterstützt wird oder obfuskiert ist. Diese Verarbeitung ist unabhängig vom KI-Hilfe (AI Helper).

8.4 Disassembler

Dieser Block von Optionen ist für die Disassemblierung von Maschinencode zuständig:



HEX dump hinzufügen (Add HEX dump) zeigt Maschinencodes vor jedem disassemblierten Befehl an, was möglicherweise nur erforderlich ist, wenn die Binärdatei nach der Analyse in einem HEX-Editor modifiziert wird.

Zeichenkonvertierung ignorieren (Ignore chars convertions) dient zum Deaktivieren des Konvertierens von nicht-englischen Stringdaten in numerische Unicode-Werte. Das Aktivieren dieser Option macht die angezeigten Strings leichter lesbar. Deaktivieren Sie sie, wenn Sie die numerische Unicode-Darstellung benötigen.

Benutzerklassen anzeigen (Show user classes) beinhaltet das Parsen von Benutzerklassen innerhalb **Delphi** / **Free Pascal**-Anwendungen. Diese Option wird nicht für **Visual Basic** und **C#**-Anwendungen verwendet.

VCL-Klassen anzeigen (Show VCL classes) beinhaltet das Dekompilieren aller internen Klassen der **VCL**-Bibliothek. Es wird empfohlen, diese Option zu deaktivieren, damit der Disassembler nicht Tausende von Serviceklassen analysiert, die bei der Analyse von Programmen in der Regel unnötig sind. Diese Option wird nur für das Dekompilieren von **Delphi**-Anwendungen zu internen Zwecken verwendet und wurde zur Vereinheitlichung mit anderen Produkten erstellt, ist jedoch nicht in **VB Decompiler** implementiert.

Die partielle Dekompilierung von **Delphi** und **Free Pascal**-Anwendungen wird derzeit ausschließlich als separate Dienstleistung (nicht als Software) angeboten.

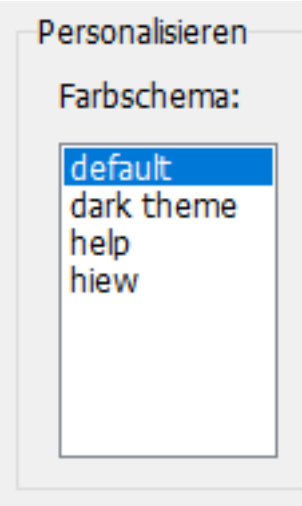
VB P-Code und .NET IL

HEX dump hinzufügen unterstützt auch das P-Code-Listing auf niedriger Ebene. Verwenden Sie **Kopieren Sie zu dem Disassembler (Copy to disassembler)** in der Decompiler-Ansicht, um rekonstruierte Anweisungen mit P-Code oder nativen Befehlen in einem gemeinsamen Listing zu vergleichen.

Für .NET-Assemblys kann der Disassembler IL oder native x86/x64-Methoden anzeigen, je nach Anwendung. Wiederhergestellte lokale Variablensignaturen können Arrays, Generics, Werttypen und TypeSpec-Referenzen umfassen. Aktivieren Sie **.NET-Strukturen anzeigen (Show .NET structures)**, wenn Sie Metadatentabellen und -Heaps neben dem Code inspizieren müssen. Spätere Versionen erweitern die Unterstützung für obfuskierte Metadaten, einschließlich der FieldPtr-Tabelle; dies garantiert nicht die erfolgreiche Wiederherstellung jeder beschädigten oder obfuskierten Assembly.

8.5 Farbschema

Dieser Abschnitt definiert das Farbschema und die Syntaxhervorhebung beim Anzeigen von Code in einem Dekompilator:



Voreinstellungen werden im Ordner "colors" innerhalb der .ini-Dateien gespeichert, die sich im Verzeichnis befinden, in dem VB Decompiler installiert wurde. Die Dateinamen bestimmen die Voreinstellungsbezeichnungen im Einstellungsbereich des Programms. Diese Dateien haben ein einfaches Format, und Sie können sie leicht selbst erstellen oder vorhandene bearbeiten.

Das Konfigurationsformat ist wie folgt:

```
[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)
```

Der Farbcode wird im 32-Bit True-Color-Format angegeben. Dezimal in einem Colors-Abschnitt (zur Rückwärtskompatibilität mit älteren Versionen von VB Decompiler) und hexadezimal im Abschnitt Colors2.

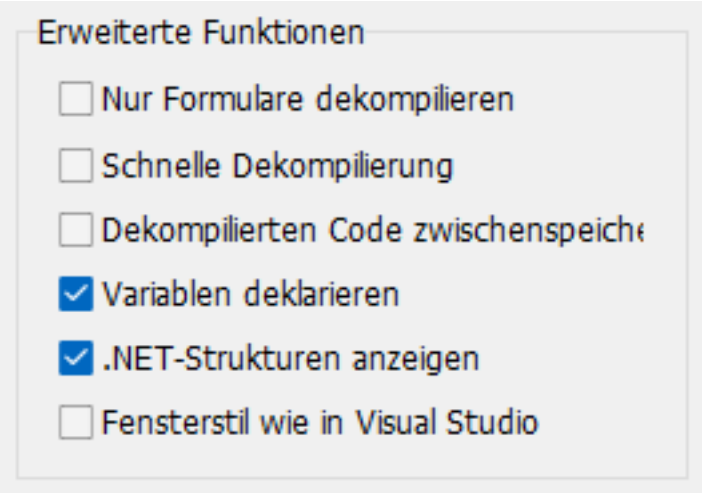
Wechseln von Themes während der Codeanalyse

Das Theme-System unterstützt den Dark Mode und kann ein ausgewähltes Theme anwenden, ohne VB Decompiler neu zu starten, einschließlich nach dem Bearbeiten seiner INI-Datei. Schriftart-Einstellungen in der erweiterten Konfiguration können durch Komma getrennte Fallback-Namen enthalten, was hilft, wenn äquivalente Schriftarten zwischen Windows und Kompatibilitätsumgebungen unterschiedlich sind.

Die Farben unter dem Cursor steuern die Hervorhebung von Identifikatoren. Siehe Anzeigen von Code für die Hervorhebung lokaler Variablen, von Objektmembern und globaler Variablen.

8.6 Erweiterte Funktionen

Diese Einstellungen steuern, wie das Projekt verarbeitet wird und wie die wiederhergestellten Informationen dargestellt werden.



Nur Formulare dekompilieren (Decompile only forms)

Schaltet die Dekompilierung von Code aus, wenn nur wiederhergestellte Formular- und UserControl-Definitionen benötigt werden. Für den visuellen Viewer und seine technologie-spezifischen Fähigkeiten siehe Visuelle Formularrekonstruktion.

Schnelle Dekompilierung (Fast decompilation)

Wenn aktiviert, verarbeitet VB Decompiler zuerst die Projektstruktur und dekompiliert einzelne Prozeduren, wenn Sie sie öffnen. Dies reduziert die anfängliche Wartezeit bei der Untersuchung ausgewählter Prozeduren. Bei deaktiviertem **Dekompilierten Code zwischenspeichern (Cache decompiled code)** können Sie Analyse-Einstellungen ändern und eine Prozedur erneut öffnen, um eine neue Auflistung zu erhalten.

Aktivieren Sie **Prototypen analysieren (Analyze prototypes)** in den Dekompilierungseinstellungen, um eine vorläufige heuristische Analyse der Signaturen privater Prozeduren durchzuführen. Dies verlängert die anfängliche Verarbeitung, kann aber die Argumentwiederherstellung verbessern, ohne eine vollständige Dekompilierung zu erfordern.

Schnelle Dekompilierung bietet keine vollständige durchsuchbare hochstufige Auflistung des Projekts. Deaktivieren Sie sie und verarbeiten Sie die Datei erneut, bevor Sie eine projektweite Textsuche durchführen oder ein vollständig dekompiliertes Projekt exportieren. Zeichenfolgenreferenzen werden unabhängig davon ermittelt: Sie können durch vorläufige Disassemblierung gesammelt werden und stehen im Modus für schnelle Dekompilierung zur Verfügung.

Vollständige Dekompilierung

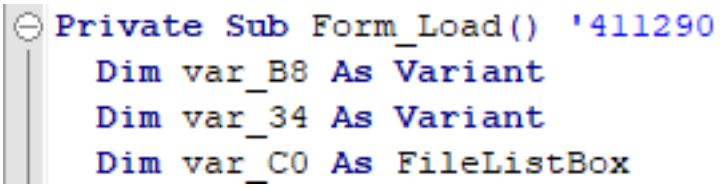
Bei deaktivierter schneller Dekompilierung verarbeitet der Decompiler die Prozeduren des Projekts im Voraus. Dies dauert länger, macht aber die vollständige generierte Auflistung für Textsuche und Export verfügbar. Änderungen an Dekompilierungseinstellungen erfordern das erneute Verarbeiten der Datei.

Dekompilierten Code zwischenspeichern

Bewahrt bereits generierte Auflistungen im Modus für schnelle Dekompilierung. Dies vermeidet wiederholtes Dekompilieren einer geöffneten Prozedur, aber zwischengespeicherte Ergebnisse spiegeln spätere Änderungen der Analyse-Einstellungen nicht wider. Wenn Sie die Auswirkungen der Einstellungen **Stackparameter analysieren (Parse stack parameters)** oder **Prozeduranalysator und -optimierer (Procedure analyzer and optimizer)** interaktiv vergleichen, deaktivieren Sie die Zwischenspeicherung.

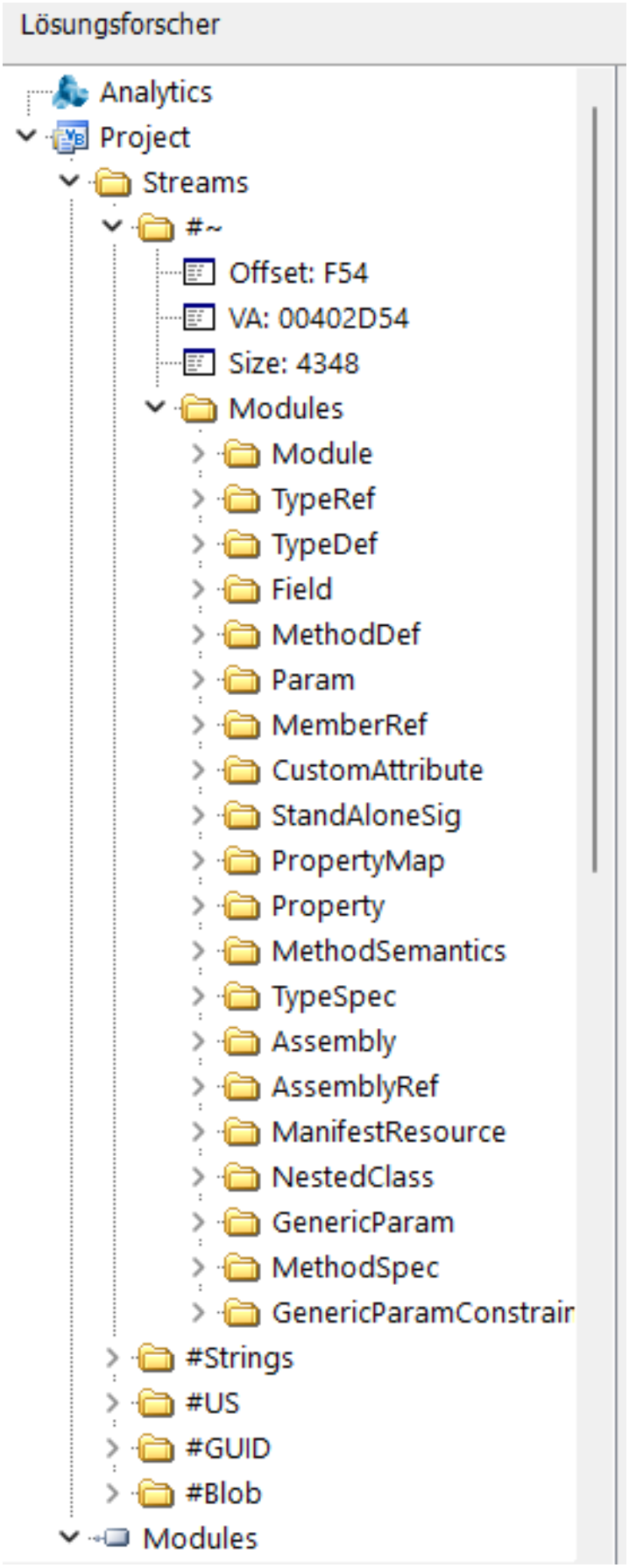
Variablen deklarieren (Declare variables)

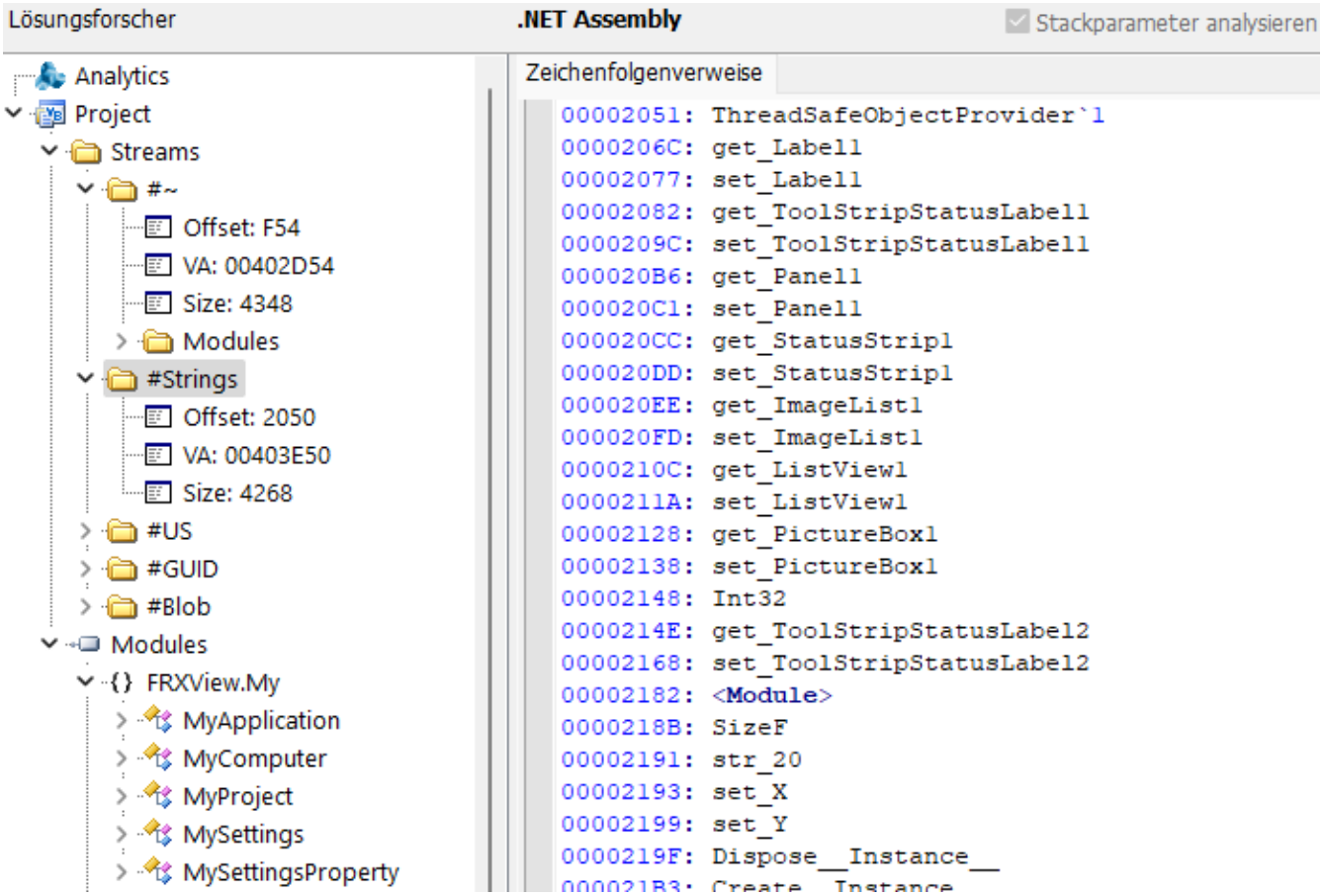
Fügt einen Deklarationsblock für wiederhergestellte Variablen am Anfang jeder dekompilierten Prozedur hinzu. Dies ist nützlich für Quellwiederherstellung und Typprüfung. Für einen knappen Überblick über die Programmlogik können Sie die Deklarationen ausblenden. Wiederhergestellte Typen unterliegen den Metadaten und Analysen, die dem Decompiler zur Verfügung stehen.



.NET-Strukturen anzeigen (Show .NET structures)

Zeigt .NET-Metadatentabellen und -Heaps im Lösungsforscher (Solution explorer) an. Verwenden Sie es, um Strukturen und Signaturen neben dem Code zu inspizieren. Das Auswählen einer Tabelle zeigt ihren Inhalt an; der Heap #US enthält Benutzerzeichenfolgen. Deaktivieren Sie die Option, wenn Sie keine Metadaten-Navigation benötigen.


Lösungsforscher
Analytics
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
Modules



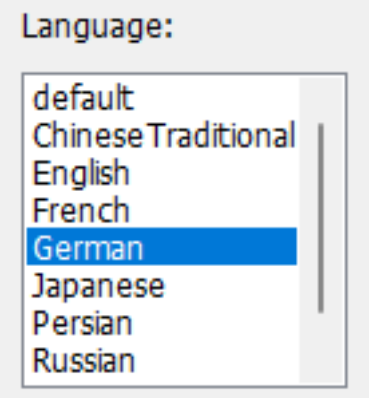
Fensterstil wie in Visual Studio (Window style as in Visual Studio)

Platziert den Code links und den Lösungsforscher rechts. Dies ändert das Arbeitsbereichs-Layout, ohne den wiederhergestellten Code zu ändern.

8.7 Sprachauswahl

VB Decompiler bietet Schnittstellenlokalisierungen für mehr als 20 weltweit verwendete Sprachen. Dokumentation ist ebenfalls in mehreren Sprachen verfügbar.

Sie können die bevorzugte Sprache der Programmoberfläche im Menü **Extras (Tools)** -> **Einstellungen (Options)**, im Bereich **Personalisieren (Personalize)**, auswählen. Nach der Sprachauswahl müssen Sie **VB Decompiler** neu starten, um die Sprachkonfigurationen im gesamten Programm ordnungsgemäß anzuwenden. Wenn das Neustarten des Programms unerwünscht ist, können Sie **VB Decompiler** weiterhin verwenden; einige Texte können jedoch noch in der vorherigen Sprache angezeigt werden.



Die Hilfesprache wird automatisch basierend auf der Systemsprache Ihres Computers ausgewählt. Um das Handbuch in einer anderen Sprache zu lesen, öffnen Sie die entsprechende Hilfe-Datei aus dem Ordner, in dem **VB Decompiler** installiert ist. PDF ist das primäre Format für lokalisierte Handbücher; CHM-Hilfe-Dateien bleiben für einige Sprachen verfügbar.

Das englische Handbuch ist die Quelle für alle Übersetzungen. Wenn Sie einen Fehler oder eine Inkonsistenz in einem lokalisierten Handbuch feststellen, melden Sie dies bitte an den technischen Support.

Zugriff auf das Handbuch

Falls eine lokale Hilfe-Datei oder der erforderliche PDF-Viewer nicht verfügbar ist, kann VB Decompiler die Online-Dokumentation öffnen. Für ein Offline-Untersuchungsumfeld sollten Sie die erforderlichen lokalen Hilfe-Dateien im Voraus verfügbar halten.

9 Plugins

Plugins erweitern VB Decompiler über native DLLs. Installierte Plugins sind im Menü **Plugins** verfügbar. Die Callback-API bietet Zugriff auf wiederhergestellte Projektinformationen und Code, sodass ein Plugin unterstützte interne Strukturen inspizieren oder modifizieren kann.

VB/DE Decompiler MCP Server 2.0.2

Verbinden Sie einen MCP-Client mit einem bereits geöffneten Projekt. Sie können Module, Formulare, Prozeduren und rekonstruierten Code lesen, ohne das analysierte Programm auszuführen.

MCP Server 2.0.2 für x86 herunterladen

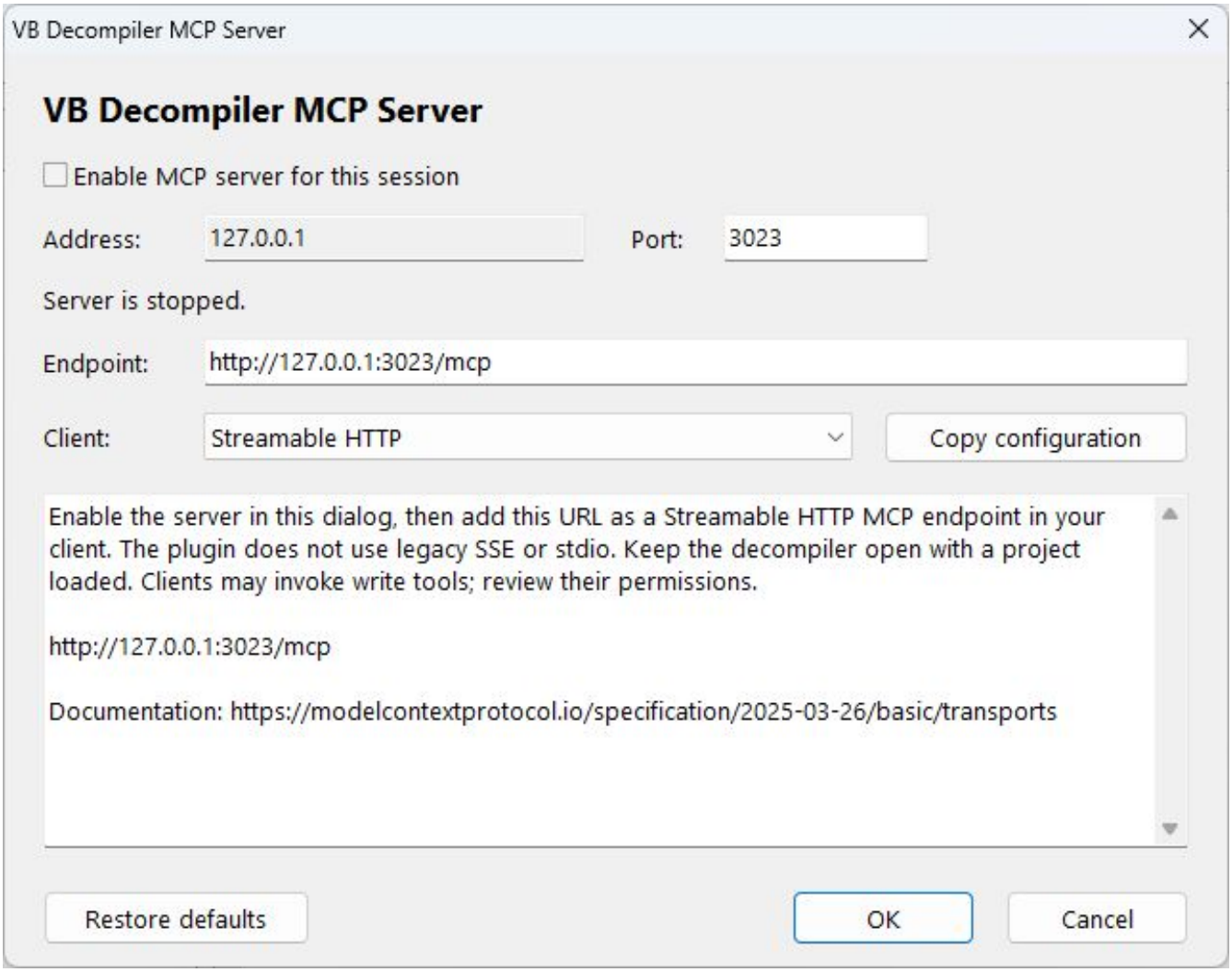
`vb_get_form` gibt den Formularindex `form_index`, den Namen `form_name` und die Eigenschaften `form_text` zurück. Wenn kein Name verfügbar ist, bleibt `form_name` leer. MCP-Befehlsreferenz und Beispiele

Voraussetzungen

Das Paket enthält ein x86-Plugin für die 32-Bit-Ausgabe des Decompilers unter Windows. Die verfügbaren Werkzeuge hängen von Version, Ausgabe und geöffnetem Projekt ab. Bei älteren Ausgaben stehen teilweise nur bereits zwischengespeicherte Listings zur Verfügung.

Installation und Start

1. Beenden Sie den Decompiler. Entpacken Sie das Archiv und kopieren Sie `vbmcp.dll` in den Ordner `plugins` der Installation. Ersetzen Sie gegebenenfalls die frühere DLL.
2. Starten Sie den Decompiler und öffnen Sie ein Projekt. Wählen Sie **Plugins** → **VB/DE Decompiler MCP Server**.
3. Aktivieren Sie **Enable MCP server for this session**, klicken Sie auf **OK** und verwenden Sie die angezeigte Adresse.



MCP-Server-Einstellungen in VB Decompiler. Das Plugin erkennt den Decompiler, der es geladen hat, und zeigt dessen Produktnamen im Fenstertitel an.

MCP-Client verbinden

Verwenden Sie Streamable HTTP. Die Standardadresse lautet `http://127.0.0.1:3023/mcp`. Nutzen Sie bei parallelen Sitzungen unterschiedliche Ports.

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

In VS Code können Sie **MCP: Add Server** aufrufen und HTTP wählen.

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

Client-Dokumentation · Codex · Claude Code · VS Code

Verbindung prüfen

Öffnen Sie ein Beispiel mit Formularen und rufen Sie `vb_list_forms` mit `limit` gleich `5` auf. Die Werkzeuge dieser Version beginnen mit `vb_`.

Lokaler Zugriff und Projektänderungen

Der Server ist nur über `127.0.0.1` erreichbar und hat keine Benutzerauthentifizierung. Lokale Clients können Projektdaten lesen sowie Listings, Namen und Markierungen ändern. Behalten Sie Schreibbestätigungen im Client bei. Entfernen Sie zum Beenden das Häkchen und klicken Sie auf **OK**. Bei jedem Programmstart ist der Server deaktiviert.

Der Plugin-Server arbeitet lokal. Der MCP-Client kann Ergebnisse entsprechend seiner Einstellungen an seinen Modelldienst übertragen.

Originalprojekt von july008 (MIT).

FRX Image Viewer

FRX Image Viewer 1.2.0 zeigt und extrahiert Bilder, die VB Decompiler oder DE Decompiler registriert hat. Verwenden Sie VB Decompiler für Formulargrafiken aus VB5/6 und .NET und DE Decompiler für Grafiken aus binären Delphi-DFM-Daten. Unterstützt werden BMP, PNG, JPEG, GIF, ICO und CUR.

Installation und Aufruf

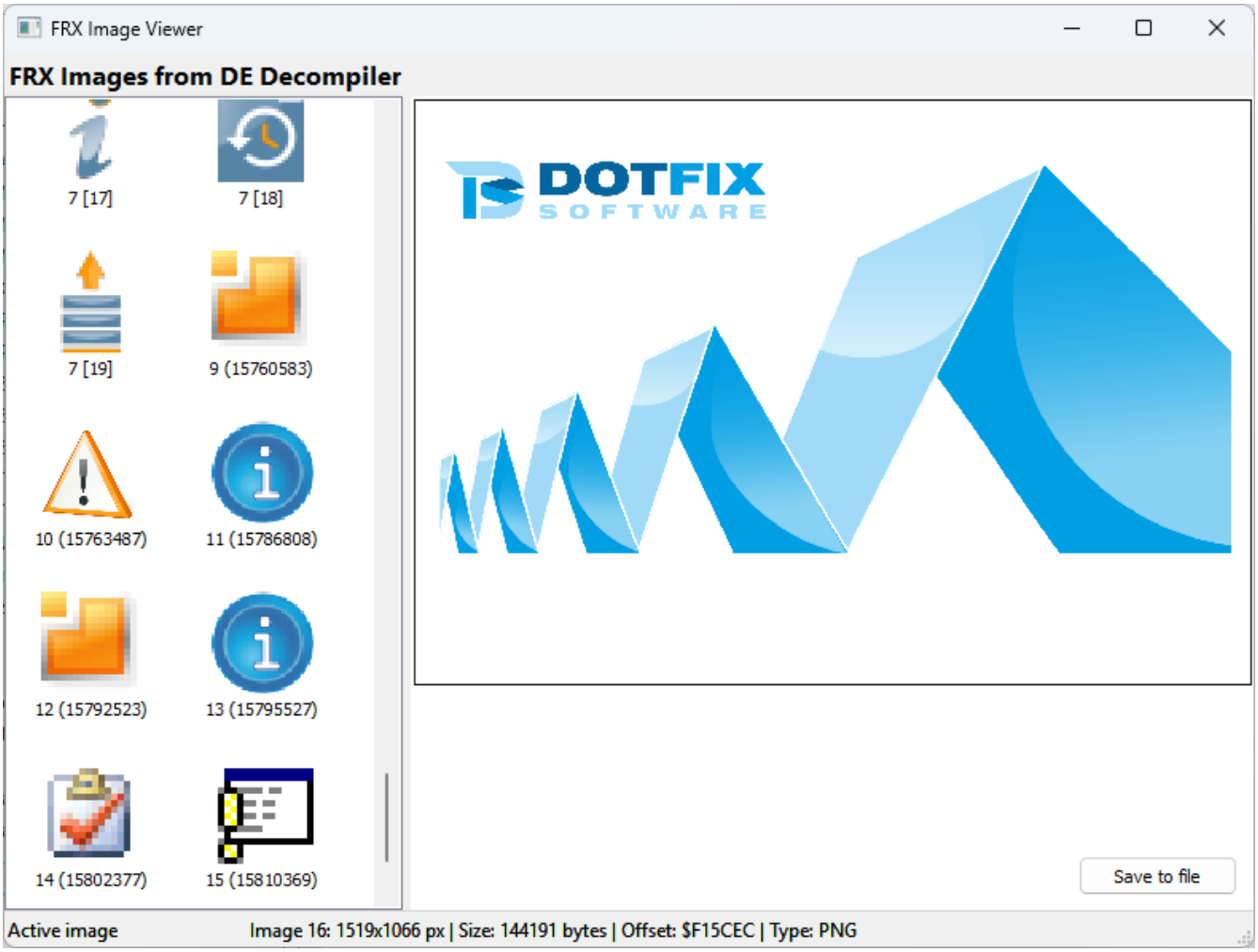
1. Kopieren Sie `FRXViewer.dll` in das Unterverzeichnis `plugins` der Hostinstallation und starten Sie den Decompiler neu. Die DLL-Architektur muss zur Hostanwendung passen, unabhängig von der Architektur der untersuchten Datei.
2. Öffnen Sie die EXE- oder DLL-Datei und lassen Sie ihre Formulare dekodieren, damit der Host die verfügbaren Bilder registriert.
3. Wählen Sie **Plugins** > **FRX Image Viewer** und anschließend ein Vorschaubild.

Der Download für 1.2.0 enthält eine DLL für einen x86-Host. Ein x64-Host benötigt einen passenden Build. Das Plugin erfordert Windows 10/11; Buildanweisungen stehen im Quellcode-Repository.

Vorschau und Export

Galerie und Vorschau zeigen die verfügbaren Bilder einschließlich zusammengesetzter ICO/CUR-Dateien. Der Statusbereich zeigt Element, Abmessungen, Größe oder Offset der Quellbitmap und Format. Das Plugin übernimmt das Hostdesign und unterstützt den dunklen Modus.

Wählen Sie **Save to file**, um das ausgewählte Element zu exportieren. Eigenständige Bilder behalten ihre ursprünglichen Bytes und ihr Format. Einzelne ImageList-Zellen werden als ICO mit Masken und Alphatransparenz exportiert; diese ICO-Dateien werden aus den Quellbitmaps erzeugt.



FRX Image Viewer in DE Decompiler mit Delphi-DFM-Grafiken und einzelnen ImageList-Symbolen.

Native Delphi-ImageLists und Kompatibilität

Mit einem kompatiblen Host erscheinen erkannte native Delphi-ImageLists als einzelne Symbole. Separat kodierte Bilder in Collection-Einträgen bleiben eigenständige Bildeinträge. Nicht unterstützte Streams behalten die Ganzbilddarstellung, sofern ursprüngliche Bildeinträge verfügbar sind.

Die gemeinsame SDK-Revision 26.6, unterstützt von VB Decompiler 26.6 und DE Decompiler 26.5, ergänzt die optionalen Befehle `GetFrxImageListInfo` (63) und `GetProductName` (64). Ältere Hosts behalten die bisherige Bild-API und Ganzbilddarstellung. Details zur Fähigkeitserkennung und zu Offsets enthält die SDK-Referenz.

Vorschau und Extraktion führen die untersuchte Datei nicht aus und laden deren Komponenten nicht. Animierte GIFs zeigen das erste Bild. Verfügbar sind nur Bilder, die der Host erfolgreich dekodiert und registriert hat.

Entwicklung eines Plugins

Das SDK enthält Beispiele für Visual C++, Rust, Delphi und Free Pascal. Sehen Sie sich die VB Decompiler Plugin-API-Referenz für exportierte Einstiegspunkte, Callback-Signaturen, Befehlsidentifikatoren, Datenrepräsentationen und Links zu den Beispielprojekten an.

Ein DLL-Plugin liefert seine Namen- und Autorinformationen und empfängt den Host-Callback über `VBDecompilerPluginLoad`. Der Callback wird verwendet, um unterstützte Projektdaten abzurufen oder zu aktualisieren. Befolgen Sie die Aufrufkonvention, Zeiger-, Zeichenkodierungs- und Pufferanforderungen des SDK bei der Implementierung dieser Schnittstelle; stellen Sie bei der Verwendung einer verwalteten Sprache eine kompatible Schnittstelle zum nativen Code sicher.

VB Decompiler initialisiert verfügbare DLL-Plugins beim Start. Erstellen Sie Plugins für die beabsichtigte Host-Architektur und Windows-Umgebung und überprüfen Sie die erzeugte DLL in dieser Umgebung. Das SDK stellt sowohl Code-Zugriff für den aktiven Ansichtsbereich als auch projektabhängigen Code-Zugriff bereit; Operationen, die alle dekompilierten Prozeduren erfordern, benötigen eine vollständige Dekompilierung statt nur eines bei Bedarf erstellten Listings.

10 Visuelle Formularrekonstruktion

Der Tab **Designer (schreibgeschützt) (Designer (read only))** zeigt eine visuelle Rekonstruktion unterstützter Schnittstellen aus der analysierten Binärdatei an. Er hilft dabei, Schnittstellenelemente mit dem Anwendungsverhalten zu verknüpfen, ohne jede wiederhergestellte Eigenschaft oder Initialisierungsanweisung manuell interpretieren zu müssen.

Öffnen einer wiederhergestellten Formulars

1. Öffnen Sie die Anwendung und lassen Sie VB Decompiler ihre Projektstruktur wiederherstellen.
2. Wählen Sie ein wiederhergestelltes Formular im Lösungsforscher (Solution explorer) aus und öffnen Sie **Designer (schreibgeschützt)**.
3. Wählen Sie ein Steuerelement aus, um seine verfügbaren Eigenschaften und wiederhergestellte Ereignisinformationen zu untersuchen. Verwenden Sie die Ereignisse, um zum entsprechenden Handler-Code zu navigieren.

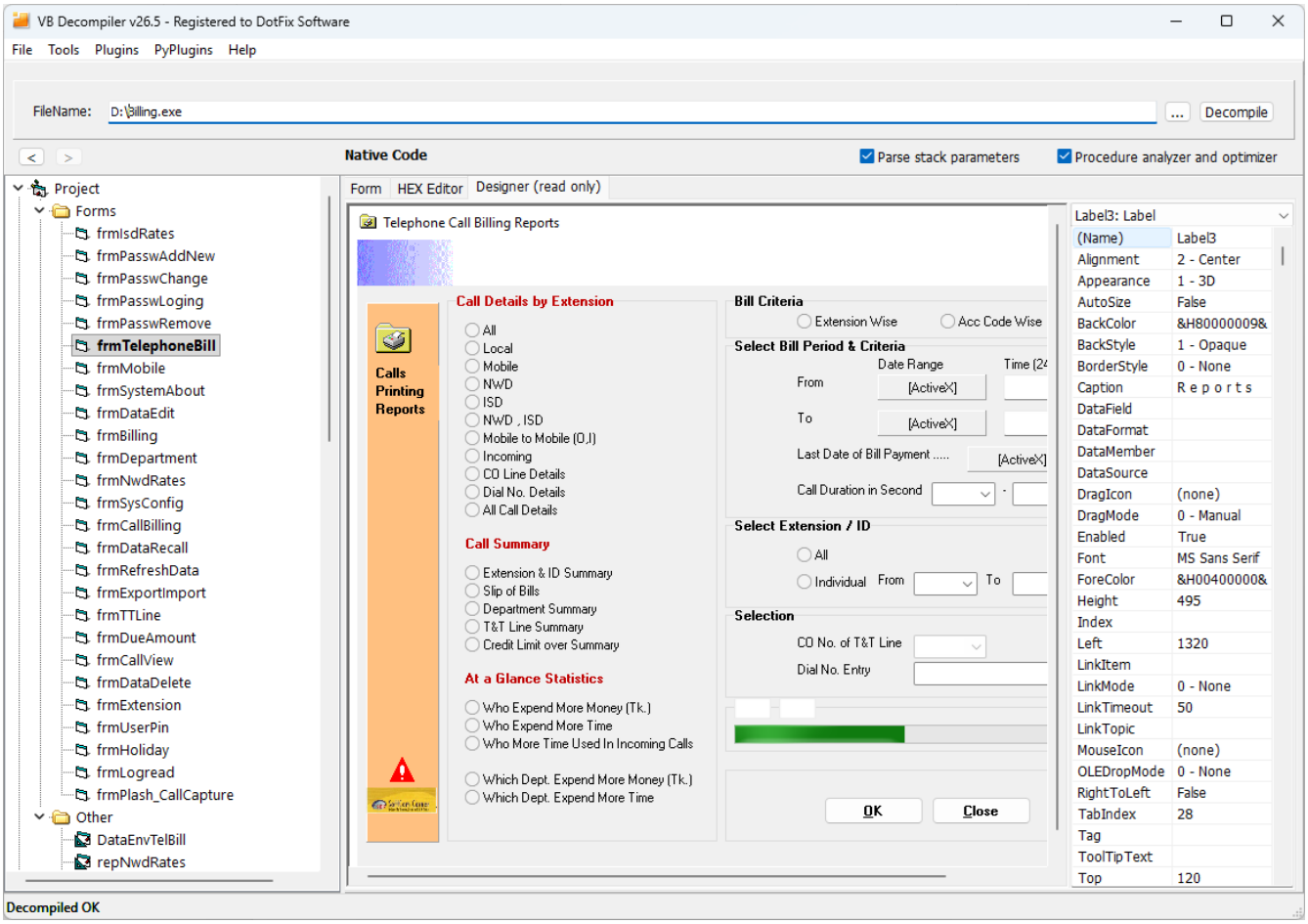
Die Vorschau wird aus gespeicherten Informationen rekonstruiert. Es handelt sich nicht um eine laufende Instanz der ursprünglichen Schnittstelle. Nicht unterstützte Steuerelemente, zur Laufzeit generierte Inhalte, fehlende Ressourcen und benutzerdefinierte Layoutlogik können unvollständige oder angenäherte Ergebnisse hervorrufen.

Visual Basic 5.0/6.0

Der Viewer rekonstruiert Formulare und UserControls, einschließlich unterstützter Steuerelementeigenschaften, Schriftarten, Farben und Bilder. Er liest Binärformdaten, die von Listen, Kombinationsfeldern und anderen Steuerelementen verwendet werden. Bilder können Symbole, Cursor, Bitmaps sowie JPEG-, PNG- und GIF-Daten umfassen.

Die Rekonstruktion des visuellen Menüs unterstützt sowohl P-Code- als auch Native Code-Anwendungen. Unterstützte Steuerelemente sind `SSTab`, `MSComctlLib.StatusBar`, `MSComctlLib.ListView`, `MSComctlLib.ProgressBar` und `MSComctlLib.TreeView`.

Um den Handler eines klassischen VB-Steuerelements zu untersuchen, wählen Sie das Steuerelement aus und doppelklicken Sie auf sein Feld **_Event** in der Eigenschaftsliste. Wählen Sie das erforderliche Ereignis aus der resultierenden Liste aus, um den Handler zu öffnen.

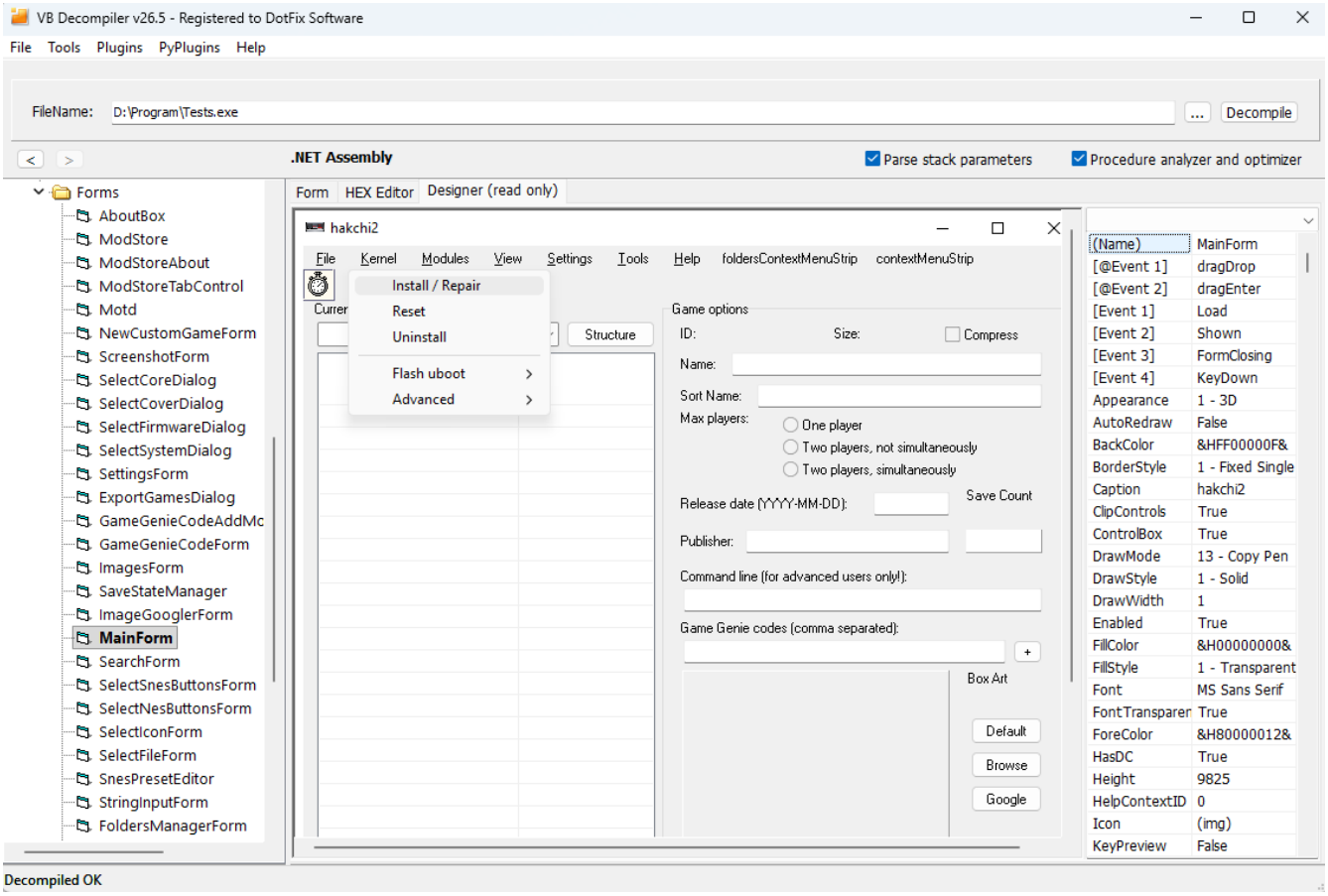


.NET Windows Forms

VB Decompiler analysiert `InitializeComponent` statisch und rekonstruiert unterstützte Windows-Form-Steuerelemente sowie deren Layout. Der Parser und der Renderer erfordern keine .NET-Runtime oder die Ausführung der Zielanwendung. Unterstützte wiederhergestellte Bilder können ebenfalls auf Steuerelementen und Form-Hintergründen angezeigt werden.

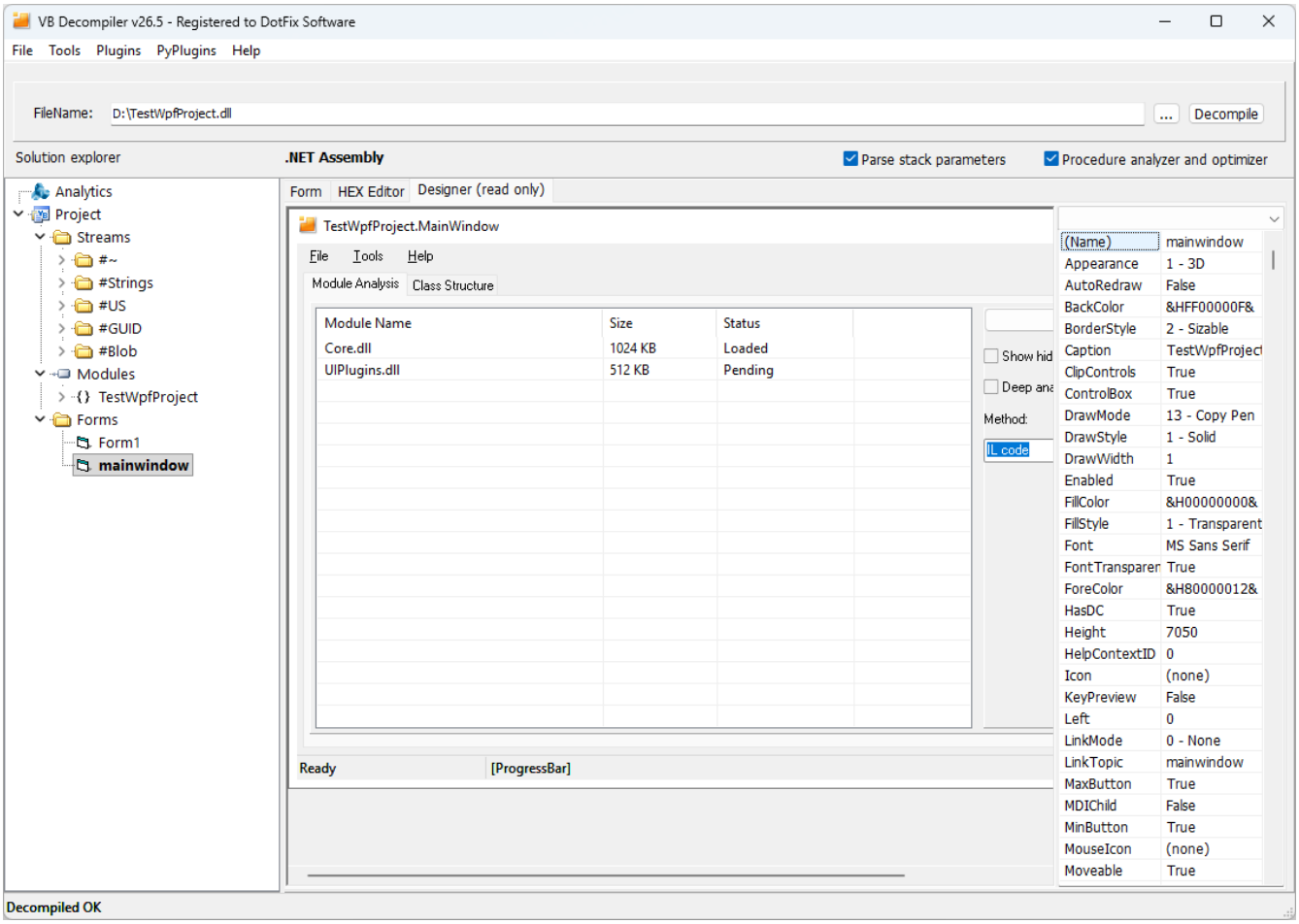
Wählen Sie ein Steuerelement aus, um seine wiederhergestellten Ereignisse zu untersuchen, und doppelklicken Sie dann auf ein Ereignis, um den entsprechenden Handler zu öffnen. Der Viewer rekonstruiert Menü- und Kontextmenühierarchien; das Klicken auf ein wiederhergestelltes Menüelement öffnet sein entsprechendes Click-Ereignis. Benutzerdefinierte Windows-Form-Ereignisse werden mit dem Präfix **@** angezeigt.

Der Viewer verarbeitet die Z-Reihenfolge von Steuerelementen, vererbte Vordergrund- und Hintergrundfarben, `Color.FromArgb`-Werte, numerische `FontStyle`-Flags, boolesche Ressourcen-Eigenschaften sowie `MinimumSize` und `ClientSize`.



WPF-Fenster, Seiten und benutzerdefinierte Steuerelemente

Der Viewer rekonstruiert unterstützte WPF-Fenster, Seiten und benutzerdefinierte Steuerelemente aus kompilierten XAML-Ressourcen, die als **BAML** gespeichert sind.



Zu den unterstützten Layout-Containern gehören **Grid** mit festen, Auto- und proportionalen (*) Zeilen- und Spaltengrößen, **StackPanel**, **DockPanel** und **Canvas**, einschließlich verschachtelter Layouts. Die Rekonstruktion berücksichtigt unterstützte Größen, Abstände, Innenabstände, Ausrichtung, Andocken sowie das Verbinden von Grid-Zeilen und -Spalten.

Neben gängigen Eingabesteuerelementen kann der Viewer Registerkarten, Gruppenkästen, Erweiterer, Menühierarchien und Trenner, Statusleisten, ListView-Spalten und unterstützte statische Daten, statische TreeView-Knoten sowie statische Kombinationsfeld-Einträge rekonstruieren. Ribbon-Schnittstellen, Navigationsbereiche und andockbare Bereiche können vereinfachte Darstellungen aufweisen.

Soweit Ressourcendaten verfügbar sind, rekonstruiert der Viewer Namen, sichtbaren Text, Tooltips, Schriftarten, Hintergrundfarben, Aktivierungs- und Sichtbarkeitszustände sowie eingebettete Bilder, einschließlich unterstützter PNG-Transparenz und Hintergrunddeckkraft. Nicht unterstützte benutzerdefinierte Steuerelemente können als Platzhalter erscheinen, die den ursprünglichen WPF-Typ identifizieren.

Der WPF-Viewer verknüpft wiederhergestellte Steuerelementereignisse mit den zugehörigen Handlern. Verwenden Sie die Ereignisinformationen im visuellen Viewer, um den entsprechenden dekompilierten Code zu öffnen.

Grenzen der Rekonstruktion: Die visuelle Vorschau ist angenähert. Vorlagen, Styles, Bindungen, benutzerdefinierte Steuerelemente und dynamisch generierter Inhalt können teilweise dargestellt werden.

Untersuchen und Extrahieren von Formdaten

Siehe Formulare für die Einstellung Offsets anzeigen (Show offsets) und verknüpfte Binäruntersuchung. Verwenden Sie das FRX-Image-Viewer-Plugin, um unterstützte wiederhergestellte Bildassets zu untersuchen und zu exportieren.

11 Tracing

Tracing features ermöglicht die schrittweise Emulation von Native Code und P-Code aus Visual Basic 5.0/6.0. Dafür ist eine Lizenz mit Tracing-Funktionen erforderlich. Der Tracer arbeitet mit der internen Darstellung einer Prozedur im Emulator; er führt die Zielprozedur nicht auf der CPU des Analysearbeitsplatzes aus.

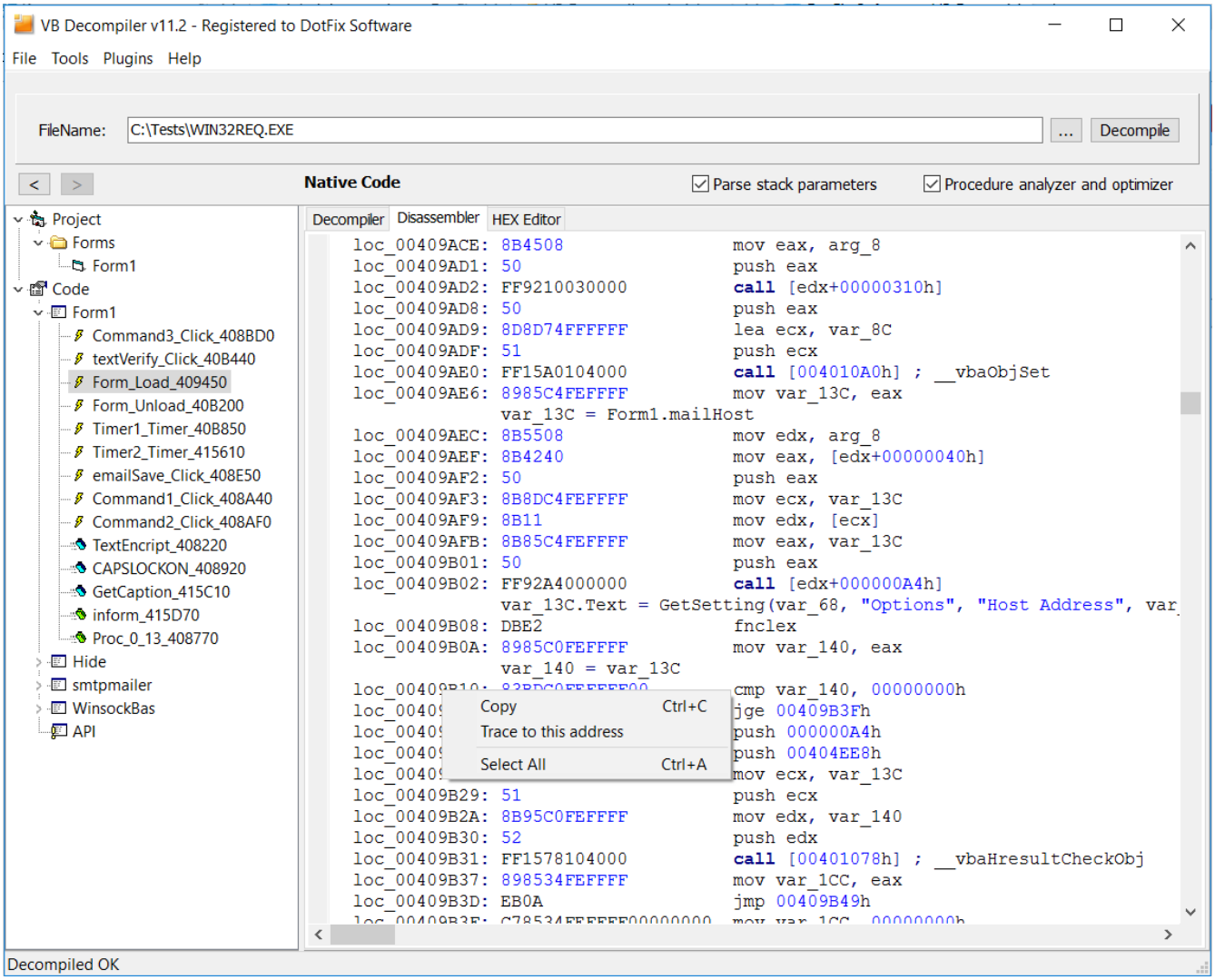
Einen Trace starten

1 Öffnen Sie eine VB5/6-Prozedur und lassen Sie die Adressanzeige im Listing aktiviert.

2 Rufen Sie auf der zu untersuchenden Zeile **Bis zur Adresse tracen (Trace to this address)** aus dem Code-Kontextmenü auf, oder drücken Sie **F4**.

3 Untersuchen Sie die Code- und Status-Anzeigen des Tracers und fahren Sie mit **Next** fort. Beenden Sie den Trace, wenn er abgeschlossen ist.

Ab Version 26.4 kann der Befehl die führende Adresse `loc_` auch bei führenden Leerzeichen auflösen. Sie können den Cursor an anderer Stelle auf dieser Zeile platzieren, statt präzise auf ihre Adressbezeichnung.



Zustand bei Native-Code-Tracing

Der Tracer zeigt CPU- und FPU-Register, den Stack, Variablen und den rekonstruierten Code an, während die Emulation fortschreitet. Verwenden Sie diese Ansichten, um zu untersuchen, wie ein Wert oder Ausdruck abgeleitet wurde. Die Suche in der Variablenliste und die Anzeige des wiederhergestellten Typs des aktuellen Objekts können bei größeren Prozeduren hilfreich sein.

P-Code Tracing

P-Code Tracing, eingeführt in Version 11, zeigt, wie sich Variableninformationen während der Verarbeitung ändern und dekompilierte Anweisungen entstehen. Es kann helfen, Unterschiede zwischen den P-Code-Zwischenoperationen und der optimierten hochsprachlichen Auflistung zu erklären.

Bereich und Einschränkungen

- Tracing bezieht sich auf eine einzelne Prozedur. Es beginnt mit einem leeren Zustand der emulierten Register, des Stacks und der Variablen, nicht mit dem Zustand eines laufenden Prozesses.
- Nicht unterstützte Native Code-Befehle können übersprungen werden. Aufrufe externer Funktionen werden nicht ausgeführt; die Unterstützung von Aufrufkonventionen kann den modellierten Stapel anpassen, ohne das Verhalten der externen Funktion nachzubilden.
- Der Tracer bietet keine vollständige Anzeige von Prozessor-Flags. Interpretieren Sie seinen Status nicht als vollständigen Hardware-Ausführungstrace.
- Das dokumentierte Modell ist eine vorwärts gerichtete schrittweise Emulation, ohne Rückwärts-Tracing oder wiederholte Ausführung von Schleifeniterationen.
- Dieser Workflow beschreibt VB5/6 Tracing. Das Vorhandensein einer .NET Disassemblierung impliziert keinen .NET Runtime Debugger.

Verwenden Sie Tracing, um die Rekonstruktion des Decompilers und Zwischenwerte zu untersuchen. Ein emulierter Wert allein belegt nicht den Wert, der von einem tatsächlichen Prozess mit initialisierten Globals, externen Abhängigkeiten und Eingabedaten erzeugt wird.

12 Analysefunktionen

Lizenzen mit **Analytic features** erzeugen kategorisierte Berichte über erkannte Operationen im analysierten Programm. Diese Berichte helfen Sicherheitsanalysten, Prozeduren zu identifizieren, die vor dem Lesen der gesamten Codebasis untersucht werden sollten. Sie unterstützen Visual Basic 5.0/6.0-Anwendungen und ab Version 11.7 auch .NET-Assemblys.

Bericht erstellen

1 Aktivieren Sie **Prototypen analysieren (Analyze prototypes)** in **Extras > Einstellungen (Tools > Options)**.

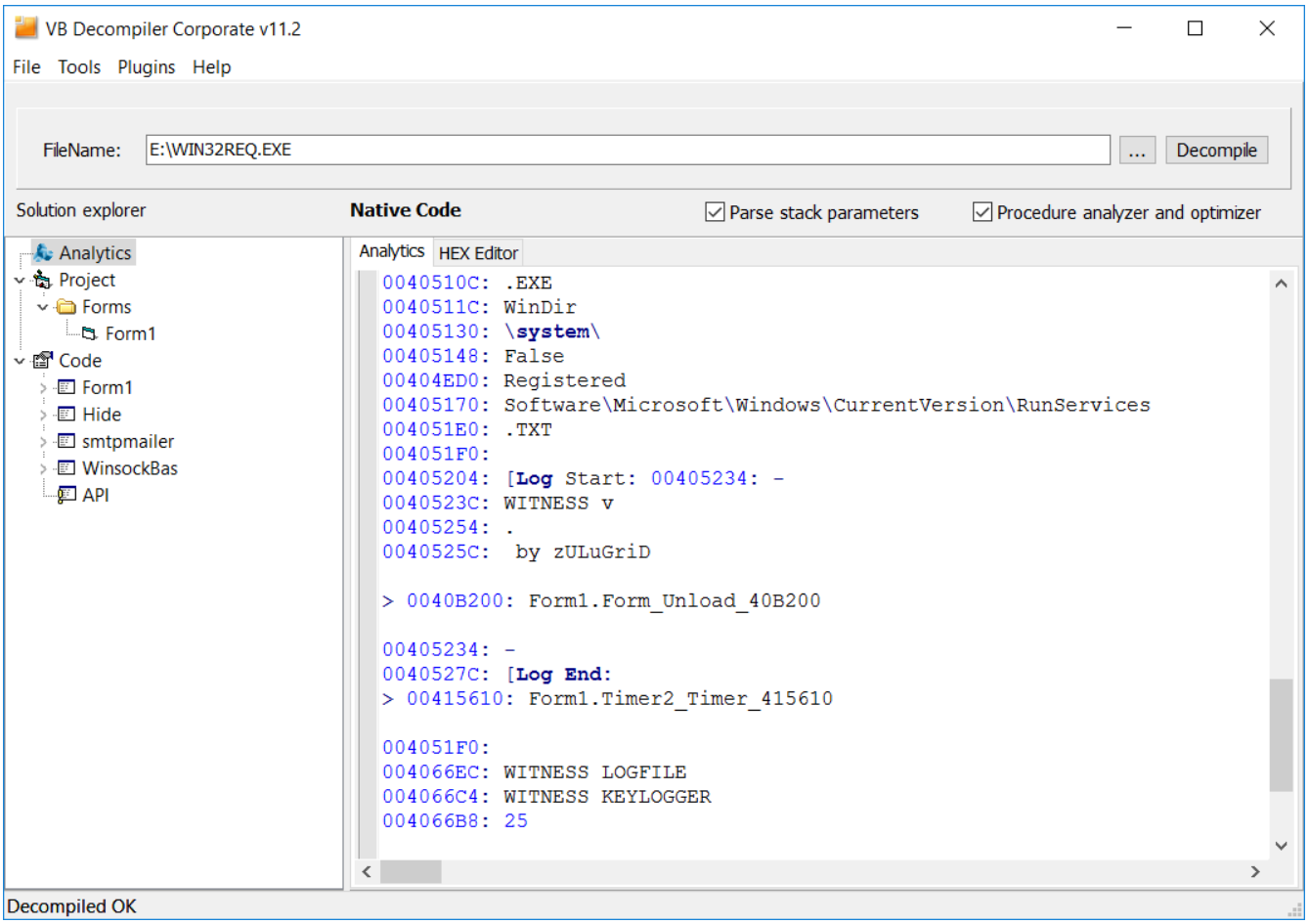
2 Öffnen Sie die Datei und warten Sie, bis die Analyse abgeschlossen ist. Mit der erforderlichen Lizenz und den Einstellungen wird der Bericht automatisch erstellt und angezeigt.

3 Überprüfen Sie die Kategorien, öffnen Sie dann relevante Prozeduren und prüfen Sie deren Code und Zeichenfolgenreferenzen.

Kategorien umfassen erkannte Operationen wie Datei- und Datenträgerzugriff, Registrierungszugriff, Netzwerkaktivität, Prozesserstellung, Fensteroperationen sowie ausgewählte interne VB-Funktionen wie `CallByName` und `VarPtr`.

Schnellbericht und String-Kontext

Der Schnellbericht listet Adressen und Namen von Prozeduren auf, die erkannte Operationen enthalten. Der erweiterte Bericht fügt die verfügbaren Zeichenfolgenreferenzen aus diesen Prozeduren hinzu. Dieser Kontext kann Kandidatenpfade, Registrierungsschlüssel, Hostnamen oder andere Werte offenbaren, die einer Prüfung bedürfen.



Verschlüsselte, obfuskierte oder zur Laufzeit generierte Strings können fehlen. Eine einer Prozedur zugeordnete Zeichenfolge liefert nützlichen Kontext, beweist aber für sich genommen nicht, dass diese Zeichenfolge an einen bestimmten API-Aufruf übergeben wird oder dass eine Verzweigung ausgeführt wird.

Navigieren und Ergebnisse speichern

Doppelklicken Sie auf einen Prozedurnamen im Bericht, um dessen Code zu öffnen. Doppelklicken Sie auf eine Adresse vor einer Prozedur oder Zeichenfolge, um die entsprechende Position im HEX-Editor zu prüfen. Verwenden Sie die **A**-Schaltfläche oberhalb des Objektbaums, um zum Bericht an der vorherigen Position zurückzukehren, und die Navigationspfeile oder **Esc**, um geprüfte Prozeduren erneut aufzurufen.

Wählen Sie den Berichts-Text aus und kopieren Sie ihn bei Bedarf oder wählen Sie **Datei > Analysebericht speichern (File > Save analytics report)**, um ihn für weitere Analysen zu speichern.

Auswertung der Ergebnisse

Der Bericht identifiziert statisch erkannte Code-Muster und Referenzen. Er dient als Triage-Hilfe, nicht als Aufzeichnung beobachteter Ausführungen oder als automatisches Urteil, dass eine Datei bösartig ist. Indirekte Aufrufe, ungelöste externe Schnittstellen, beschädigte Metadaten, Obfuskation und zur Laufzeit generierte Werte können die Abdeckung einschränken. Bestätigen Sie wesentliche Ergebnisse in den entsprechenden Prozeduren, der Disassemblierung und den Dateidaten.

Für eine tiefere Inspektion von VB5/6-Rekonstruktionen siehe Tracing. Für Literalstrings und deren Code-Positionen siehe Zeichenfolgenreferenzen.

13 Einschränkungen der Obfuskation

VB Decompiler, ausgestattet mit einer einzigartigen Obfuskationstechnologie, verdeckt effektiv die vom Compiler hinterlassenen Daten in unserer Forschung zur Dekompilierung von **Visual-Basic-5.0- und 6.0-Anwendungen**. Moderne Programmiersprachen speichern häufig erhebliche Mengen an Metadaten und Debugging-Informationen in kompilierten Dateien, was es Angreifern erleichtert, gewünschte Funktionen innerhalb Ihres Programms zu lokalisieren. Um dieser Bedrohung entgegenzuwirken, haben wir einen Obfuskator entwickelt, der speziell für **Visual Basic**-Anwendungen konzipiert ist.

Der Obfuskationsprozess im Detail

Der Obfuskator funktioniert, indem er das gesamte Programm in seine Bestandteile zerlegt, einschließlich Formulare, Module, Klassen und Steuerelemente wie Schaltflächen und Textfelder. Jedes Formular wird dann in einzelne Steuerelemente mit zugehörigen Ereignissen zerlegt. Anschließend werden Informationen über Funktionen und Prozeduren sowie Ereignisse, die an jedes Objekt im Programm gebunden sind, gesammelt. Während dieses Prozesses prüft der Obfuskator auf Abhängigkeiten zwischen Objekten und ändert deren Namen konsistent, um es schwierig zu machen, festzustellen, welche Funktion zu einem bestimmten Modul oder einer Klasse gehört.

Infolgedessen wird das Auffinden spezifischer Formulare wie **frmTrial** oder **frmRegistration** innerhalb des Programms erheblich erschwert.

Einschränkungen bei der Obfuskation von Visual-Basic-5.0/6.0-Anwendungen

Der vom Obfuskator verwendete Algorithmus ist komplex und dient der Analyse von Objektabhängigkeiten, unterstützt aber dennoch nicht alles. Folglich müssen Sie in einigen Fällen Ihr Programm vorab für den Obfuskationsprozess vorbereiten. Die unten diskutierten Punkte gelten ausschließlich für die Obfuskation von Visual Basic 5.0 / 6.0-Anwendungen.

1. Der Obfuskator unterstützt keine Design-Zeit-Referenzen auf andere Objekte, was zu Problemen zur Laufzeit führen kann.

Betrachten Sie beispielsweise eine Situation, in der Sie ein **TreeView**-Steuerelement haben, das von einem zugehörigen **ImageList** abhängt. Wenn der Name dieses **ImageList**-Objekts in den Eigenschaften Ihres **TreeView** hartcodiert ist, führen alle Versuche zur Obfuskation zu Fehlern, da die ursprüngliche Referenz intakt bleibt. Diese Einschränkung gilt auch für datenbankbezogene Komponenten wie **DataSets**

```
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
```

Dieses Problem kann leicht gelöst werden, indem die Zuweisungen von der Design-Zeit zur Laufzeit verschoben werden, einfach indem diese Zuweisungen in das **Form_Load**-Ereignis verschoben werden:

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

Die gleiche Korrektur muss für jeden gleichartigen Fall angewendet werden. Zum Glück gibt es in den meisten Projekten normalerweise nur wenige solcher Fälle.

2. Objektdeklarationen mit As New werden nicht unterstützt (LateCall-Aufrufe)

Diese Einschränkung betrifft die Mehrheit der VB6-Projekte in der Regel nicht, ist aber wichtig im Hinterkopf zu behalten. Wenn Sie mit Formularen als Klassen arbeiten und Instanzen von Formularen erstellen, könnten Sie mit diesem Szenario vertraut sein:

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

In solchen Fällen fügt der VB6-Compiler den Namen **frmMain** direkt in den Code der **Form_Load**-Funktion ein. Da unser Obfuskator nur Daten und keinen Code ändert, bleibt dieser Name im Code nach der Obfuskation unverändert. Leider führt dies zu einem Programmabbruch mit Fehler.

Leider ist es für uns unmöglich, solche Aufrufe im Obfuskator von **VB Decompiler** zu unterstützen, da wir den gesamten Code aller Funktionen zerlegen und dekompilieren müssen. Trotz dieser Fähigkeit ist die Dekompilierung ein extrem zeitaufwändiger Prozess. Für große Projekte kann es über eine halbe Stunde dauern, nur um die gesamte Codebasis zu analysieren. Darüber hinaus müssen wir auch Referenzen auf Objekte analysieren und umbenennen, was noch mehr Verarbeitungszeit hinzufügt.

Angesichts dieser technischen Einschränkungen haben wir uns entschieden, keine vollständige Dekompilierung und Codeanalyse im Obfuskator von **VB Decompiler** zu implementieren, insbesondere da dies das Problem nur nur in seltenen Fällen bei Visual Basic 5.0 / 6.0-Apps lösen würde und den meisten Kunden keinen Nutzen bieten würde.

14 Technischer Support

Wenn Sie Fragen zum Programm haben und die Antworten in dieser Hilfedokumentation nicht finden können, wenden Sie sich bitte an unser technisches Support-Team. Diese Option steht nur registrierten Benutzern zur Verfügung. Für nicht registrierte Benutzer stehen wir gerne bei Anfragen zur Lizenzbeschaffung für **VB Decompiler** zur Verfügung.

Darüber hinaus können Sie sich bei technischen Problemen oder Verbesserungsvorschlägen gerne an unseren technischen Support wenden, um uns Ihr Feedback zu übermitteln.

15 Registrierung

Um die vollen Fähigkeiten des **VB Decompiler** zu nutzen, müssen Sie eine **Business License** erwerben. Durch den Kauf einer Business-Lizenz für einzelne Entwickler oder einer Unternehmenslizenz erhalten Sie die Möglichkeit:

1. Annähernd 85% des Codes typischer unterstützter **Visual Basic 5.0/6.0 P-Code**-Anwendungen wiederherstellen;
2. Bis zu 85% des Codes typischer unterstützter **Visual Basic 5.0/6.0 Native Code**-Anwendungen wiederherstellen;
3. Typische unterstützte, nicht obfuskierte **.NET**-Assemblys in **C#**-ähnlichen Code dekompilieren, mit einer Wiederherstellung von bis zu 95%;
4. Immer die neueste Version des Programms haben: Updates sind absolut kostenlos für ein Jahr nach dem Kauf der Lizenz;
5. Technischen Support per E-Mail für ein Jahr nach dem Kauf der Lizenz erhalten;
6. Die Entwicklung dieses Decompilers unterstützen.

Darüber hinaus empfehlen wir, den Kauf spezieller Lizenzen für Malware-Analysten und Forensik in Betracht zu ziehen:

1. Lizenz mit Tracing-Funktionen
2. Lizenz mit Analysefunktionen

Um eine **Lizenz** des Decompilers jetzt zu erwerben, wählen Sie die Option **Pro Version kaufen (Order Pro version)** im Menü **Hilfe (Help)** des Programms. Zu diesem Zeitpunkt müssen Sie eine Internetverbindung haben. Ein Browserfenster öffnet sich zur Website des Programms, wo Sie die Lizenz kaufen können.

Die **Business License** des Decompilers ist für den Einsatz nur auf einem einzigen Computer und nur durch einen **Single Developer** (Lizenzinhaber) konzipiert. Wenn Sie das Programm auf mehreren Computern verwenden müssen, müssen Sie eine separate Lizenz für jeden erwerben oder die **Business Enterprise license** in Betracht ziehen (für bis zu 20 Computer).

16 Über den Entwickler

Der Decompiler wird von **DotFix Software** entwickelt.

Produktwebsite: www.vb-decompiler.org
Support-E-Mail: admin@vb-decompiler.org