DE Decompiler

Руководство пользователя

Русский

Программы Delphi 3-13 / C++Builder · VCL

DotFix Software

Содержание

1 Начало работы 4

1.1 Введение 4

1.2 Быстрый старт 6

1.3 Совместимость компиляторов и RTTI 7

1.4 Главное окно 8

1.5 Модули, классы и функции 10

2 Открытие и сохранение 12

2.1 Открытие файла 12

2.2 Базы анализа 13

2.3 Сохранение результатов 14

2.4 Сохранение списка процедур 15

3 Изучение приложения 16

3.1 Просмотр и сравнение кода 16

3.2 Навигация по адресу и пути дерева 19

3.3 Поиск процедур и текста 20

3.4 Ссылки на строки 21

3.5 Декомпиляция по адресу 22

3.6 Просмотр и изменение байтов 23

3.7 Формы VCL и визуальный просмотр 24

4 Параметры и инструменты 26

4.1 Параметры программы 26

4.2 Параметры декомпиляции 27

4.3 Отображение дизассемблированного кода 29

4.4 Темы оформления и шрифты 31

4.5 Режимы обработки и кэширование 33

4.6 Язык интерфейса 34

4.7 AI Helper 35

4.8 Плагины и SDK 38

4.9 Обфускация 41

5 Справочник 42

5.1 Интерпретация восстановленного кода 42

5.2 Техническая поддержка 44

5.3 Лицензирование 45

5.4 О разработчике 46

1 Начало работы

1.1 Введение

Профессиональный декомпилятор VCL-программ, написанных на Delphi или созданных в C++Builder: код Win32/Win64, формы, модули и классы для реверс-инжиниринга и компьютерной криминалистики.

Декомпилируйте **VCL-приложения, написанные на Delphi 3–13 или созданные в C++Builder, для Win32 и Win64** в Delphi-подобный Pascal. Восстанавливайте поток управления, вызовы, формы, модули и классы по машинным инструкциям и доступным метаданным; переходите от событий VCL к их обработчикам.

Используйте восстановленную структуру и код для исследования поведения программы, вредоносного ПО и потенциально нежелательной или скрытой функциональности. Отслеживайте события VCL до их обработчиков, изучайте поток управления и вызовы, а также используйте дизассемблер и HEX-редактор для детального анализа.

Нативная декомпиляция

Высокоуровневый листинг восстанавливает логику программы, включая `if` условия, `case` блоки, циклы и вызовы распознанных методов VCL. Распознанные модули, классы и публичные функции обеспечивают структурный обзор; эвристический поиск также выявляет приватные функции внутри каждого модуля.

Формы и инструменты

Изучайте восстановленные элементы управления VCL и их свойства в режиме просмотра формы и переходите от событий к обработчикам. Экспортируйте модули и формы в проект Pascal/DFM или объединённый листинг `.pas`. Встроенный обфускатор поддерживает VCL-приложения, написанные на Delphi или созданные в C++Builder, для Win32 и Win64.

Информация, сохраняемая в бинарном файле

Результат представляет собой Delphi-подобный код, восстановленный по машинным инструкциям и метаданным. При компиляции сохраняется не вся информация исходного кода. Имена локальных переменных обычно генерируются автоматически, структура данных может определяться косвенно по размерам обращений к памяти и адресам, а нераспознанные вызовы могут отображаться без исходного имени метода VCL. Базы знаний о RTTI конкретных версий компилятора помогают восстановить сохранившиеся имена и типы.

Высокоуровневый листинг восстанавливает логику нативной программы для анализа с детализацией, поддерживаемой имеющимися данными. Проверяйте предполагаемые типы, неожиданные выражения и важные выводы, сопоставляя их с соответствующими инструкциями. См. Интерпретация восстановленного кода.

Восстановление утраченного исходного кода — дополнительный сценарий использования. Полученный результат требует доработки и не является готовым к компиляции проектом. Перед открытием восстановленного кода в IDE вручную проанализируйте и исправьте неточности декомпиляции и несовместимости, а также восстановите недостающие зависимости.

Приложения, созданные в C++Builder

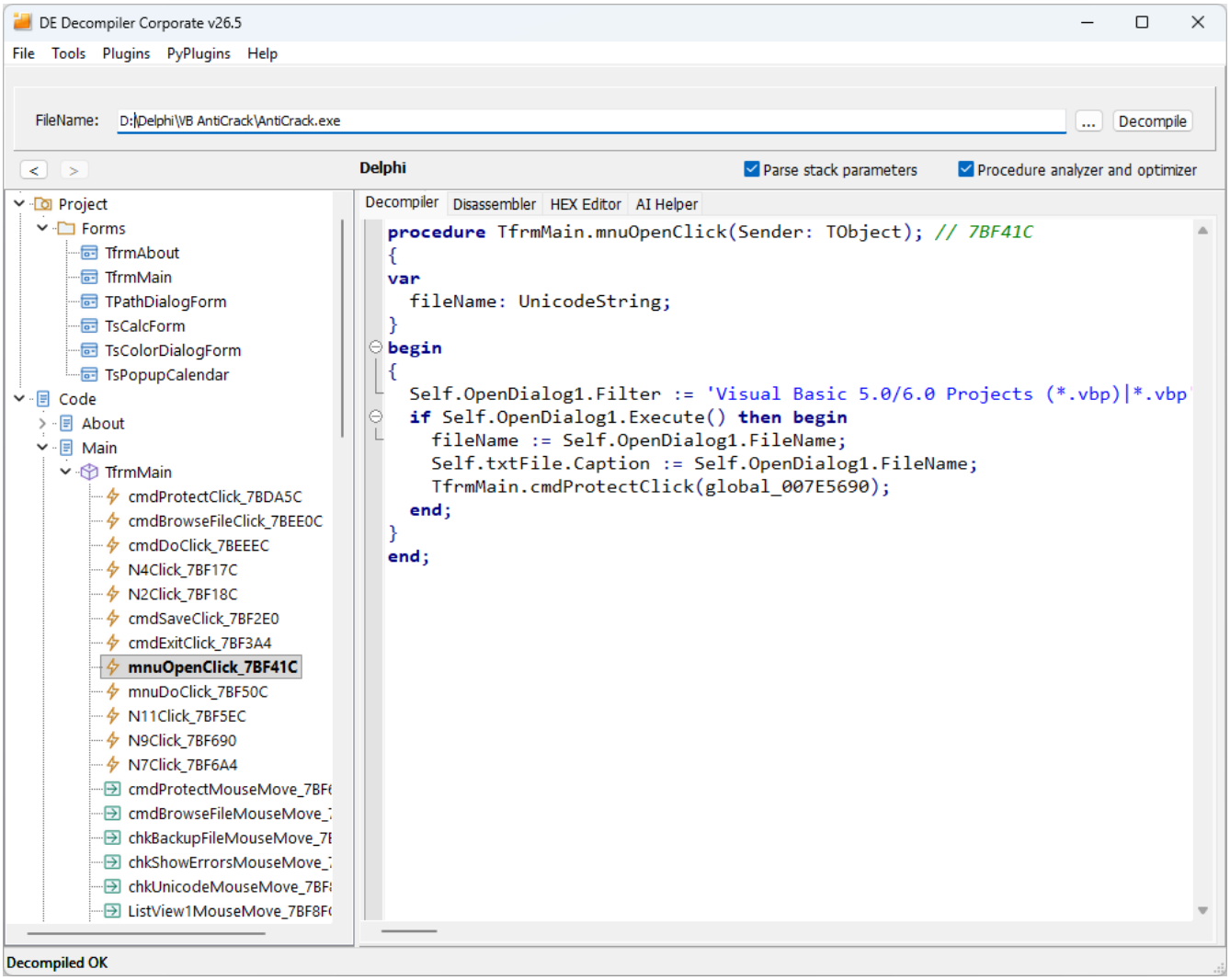
DE Decompiler поддерживает VCL-приложения, созданные в C++Builder, для Win32 и Win64. Доступны восстановление и визуализация форм, разбор модулей, классов и функций, переходы к обработчикам событий, а также остальные инструменты анализа и экспорта.

Баз знаний RTTI для библиотек, специфичных для C++Builder, пока нет. Функции вне общей базы VCL, а также нераспознанные свойства и методы объектов могут отображаться без библиотечных имён. Поэтому качество декомпиляции кода C++Builder заметно ниже, чем для программ, написанных на Delphi. Это ограничение касается качества восстановления кода и распознавания библиотечных имён. Остальные возможности доступны для C++Builder по тем же правилам редакций, что и для программ, написанных на Delphi.

Код C++Builder восстанавливается в виде Delphi-подобного Pascal с экспортом в PAS/DFM. Исходный текст на C++ не восстанавливается. Как и при другом восстановлении нативного кода, перед использованием результата в IDE вручную проверьте и исправьте предполагаемые типы, сгенерированные имена и погрешности декомпиляции.

Редакции

Декомпиляция высокоуровневого кода, AI Helper, HEX-редактор и работа с обфускацией требуют лицензии Single Developer или Enterprise. Free Lite восстанавливает информацию о формах VCL и обеспечивает нативное дизассемблирование и ссылки на строки. См. Лицензирование.



Декомпилированный код, похожий на Delphi, с восстановленной структурой проекта.

1.2 Быстрый старт

Откройте поддерживаемое VCL-приложение для Win32 или Win64, написанное на Delphi или созданное в C++Builder. Изучите восстановленную структуру проекта и сверяйте код с машинными инструкциями. Для декомпиляции кода на язык высокого уровня, AI Helper и HEX-редактора нужна коммерческая лицензия. В Free Lite доступны просмотр форм, дизассемблирование и ссылки на строки.

1. Выберите **«Файл > Открыть программу» (File > Open program)** и выберите бинарный файл для изучения. Дайте DE Decompiler завершить начальную обработку.
2. Расширьте **Код** в дереве проекта. Выберите модуль для изучения его восстановленных объявлений, затем выберите класс или процедуру.
3. Сравнить **«Декомпилятор» (Decompiler)** и **«Дизассемблер» (Disassembler)**. Используйте **«HEX Редактор» (HEX Editor)** когда адрес или инструкция требует проверки против байтов файла.
4. Расширьте **«Формы» (Forms)**, выберите восстановленную форму и откройте **«Дизайнер (только чтение)» (Designer (read only))**. Изучите её элементы управления и свойства, а также используйте восстановленные записи событий для открытия соответствующего обработчика кода.
5. Используйте **Ctrl+T** для поиска процедуры, **Ctrl+F** для поиска отображаемого текста, и **«Утилиты > Все строки программы» (Tools > String references)** для исследования распознанных строковых литералов.
6. Сохраните базу данных анализа через **«Файл > База Данных» (File > Data Base)** когда вам нужно вернуться к анализу позже.

Сохранение результатов

Используйте **«Файл > Сохранить декомпилированный проект» (File > Save decompiled project)** для экспорта восстановленного Delphi проекта, или **«Файл > Сохранить все в один модуль» (File > Save all in one module)** для объединённого `.pas` листинга. Перед открытием восстановленного кода в IDE вручную исправьте неточности декомпиляции и несовместимости, а также обеспечьте недостающие зависимости. Храните неизменную копию проанализированного бинарного файла вместе с материалами расследования.

Выбор режима обработки

«Быстрая декомпиляция» (Fast decompilation) обрабатывает отдельные процедуры при их открытии. Включите **«Анализ прототипов» (Analyze prototypes)** для предварительного анализа сигнатур процедур. Перед поиском по всему тексту высокого уровня отключите **«Быстрая декомпиляция»** и повторно обработайте файл. Экспорт проектов и объединённых листингов обрабатывает процедуры по мере необходимости; ознакомьтесь с сообщениями о завершении. См. Режимы обработки и кэширование.

Полезная первая проверка

Начните с известной формы и одного связанного с событием метода или публичной функции. Проверьте восстановленные объявления, проследите за кодом и сравните несколько операторов с дизассемблированием. Это позволит установить, какую полезную структуру DE Decompiler распознал в данной конкретной сборке.

1.3 Совместимость компиляторов и RTTI

DE Decompiler распознаёт версии компилятора Delphi с 3 по 13 и поддерживает VCL-приложения, созданные в C++Builder, для Win32 и Win64.

Версии компилятора и восстановленные типы

Версии Delphi различаются структурой исполняемых файлов и форматом RTTI — информации о типах времени выполнения. Для интерпретации этих различий DE Decompiler использует знания о типах, соответствующие версии компилятора. Распознанная версия компилятора не гарантирует одинакового качества восстановления всех типов, библиотечного кода и процедур.

Проверяйте распознанный класс по его объявлениям, свойствам форм и коду методов. Если имя, тип поля, сигнатура или макет выглядят несогласованно, используйте нативные инструкции и байты файла для исследования перед тем, как полагаться на высокоуровневое представление.

Определение компилятора и базы сигнатур

C++Builder и Free Pascal определяются отдельно от Delphi. Сведения о компиляторе остаются корректными при использовании совместимой базы сигнатур и после загрузки сохранённого проекта. Длинные названия и диапазоны версий помещаются в поле главного окна; полные сведения доступны в подсказке.

Название компилятора определяется по входному файлу независимо от базы сигнатур, выбранной для анализа кода. Если имеющихся данных недостаточно для точного определения версии, неоднозначность сохраняется в отображаемом названии.

В коммерческих версиях DE Decompiler выбирает базу из каталога `de`: сначала для точной версии компилятора, затем для ближайшей более новой и, если её нет, для ближайшей более старой версии той же разрядности. Имена файлов имеют вид `vcl<version>.bin` или `vcl<version> x64.bin`.

Звездочка (*) обозначает использование заменяющей базы или базы, соответствие которой не удалось установить с достаточной уверенностью. Наведите указатель на название компилятора, чтобы увидеть имя выбранного BIN-файла. Маркер относится к подбору базы и сам по себе не означает, что файл невозможно проанализировать. Базы сигнатур доступны в коммерческих версиях.

Приложения, созданные в C++Builder

DE Decompiler поддерживает VCL-приложения, созданные в C++Builder, для Win32 и Win64. Доступны восстановление и визуализация форм, разбор модулей, классов и функций, переходы к обработчикам событий, а также остальные инструменты анализа и экспорта.

Баз знаний RTTI для библиотек, специфичных для C++Builder, пока нет. Функции вне общей базы VCL, а также нераспознанные свойства и методы объектов могут отображаться без библиотечных имён. Поэтому качество декомпиляции кода C++Builder заметно ниже, чем для программ, написанных на Delphi. Это ограничение касается качества восстановления кода и распознавания библиотечных имён. Остальные возможности доступны для C++Builder по тем же правилам редакций, что и для программ, написанных на Delphi.

Код C++Builder восстанавливается в виде Delphi-подобного Pascal с экспортом в PAS/DFM. Исходный текст на C++ не восстанавливается. Как и при другом восстановлении нативного кода, перед использованием результата в IDE вручную проверьте и исправьте предполагаемые типы, сгенерированные имена и погрешности декомпиляции.

Приватные функции

Декомпилятор пытается эвристически выявлять приватные функции внутри каждого модуля. Такое обнаружение помогает выявить код за пределами распознанных точек входа, но не гарантирует, что все границы функций или ассоциации с модулями были восстановлены.

Формы VCL

Рабочий процесс визуальных форм применяется к формам и элементам управления VCL, восстановленным из входных файлов Win32 и Win64.

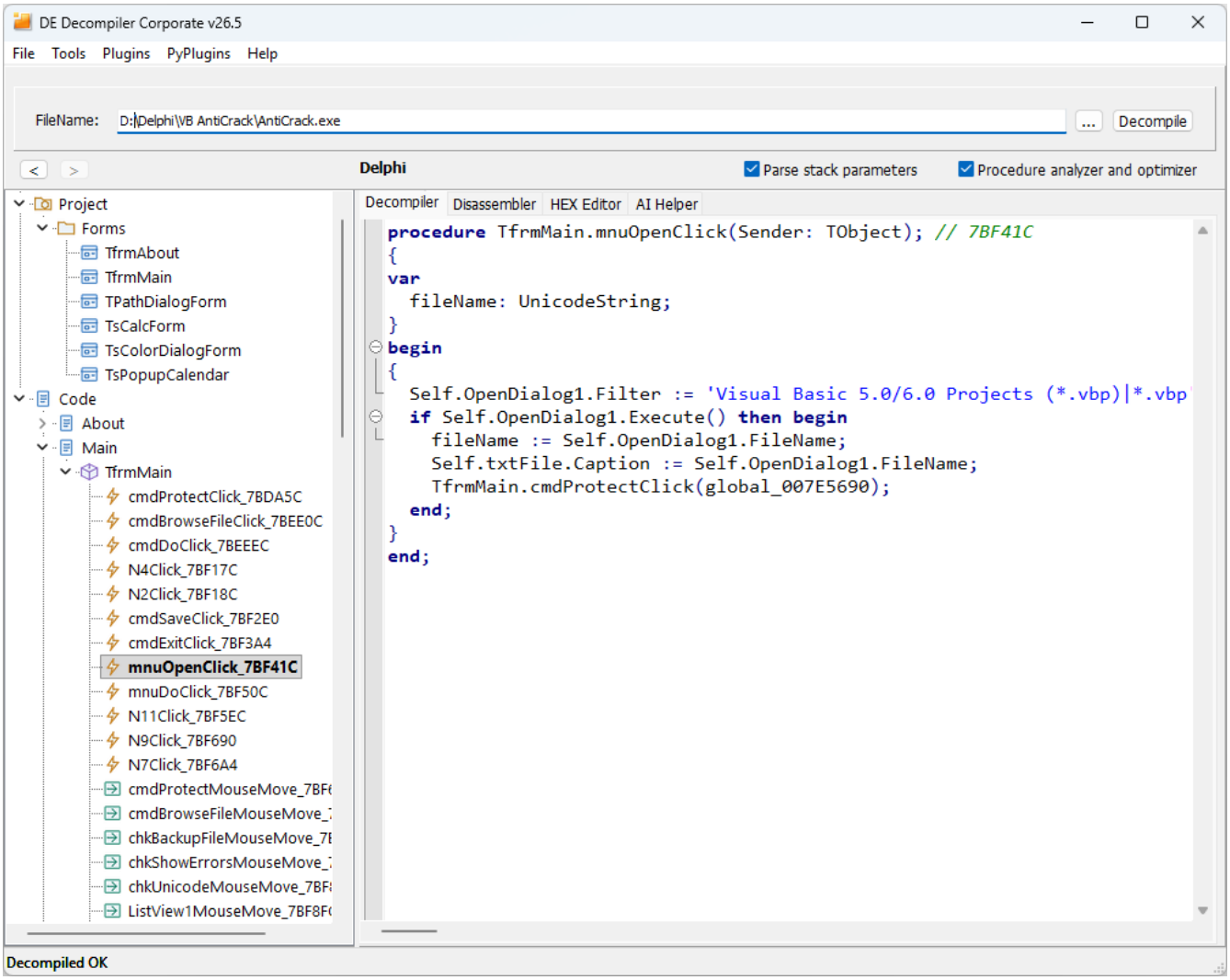
Сообщение о проблеме совместимости

DE Decompiler постоянно развивается, а качество декомпиляции улучшается по мере совершенствования поддержки компиляторов, распознавания типов и реконструкции кода.

Если вы хотите сообщить о конкретной проблеме, отправьте двоичный файл, воспроизводящий ошибку, и укажите форму или функцию, в которой она возникает. Включите ее имя и, для функции, ее адрес (если доступен), вместе с версией сборки DE Decompiler и описанием ошибки. См. Техническая поддержка для контактных данных.

1.4 Главное окно

Дерево проекта связывает восстановленные формы, модули, классы и процедуры с представлениями кода, объявлений и бинарных данных.



Рабочее пространство DE Decompiler в светлой теме интерфейса.

Дерево проекта

«Формы» (Forms) содержит восстановленные объекты форм. **Код** группируется как **Модуль > Класс > Процедура**. Модули и классы остаются свёрнутыми до момента их раскрытия; раскрытие модуля с одним видимым классом также показывает процедуры этого класса. Выбор модуля отображает его восстановленные объявления; выбор процедуры открывает соответствующие представления кода.

Представления и элементы управления обработкой

- **«Декомпилятор» (Decompiler)** отображает восстановленные высокоуровневые операторы.
- **«Дизассемблер» (Disassembler)** отображает нативные инструкции и адреса.
- **«HEX Редактор» (HEX Editor)** отображает исходные байты файла.
- **«Дизайнер (только чтение)» (Designer (read only))** отображает восстановленную VCL форму и её доступные свойства.
- **DFM** отображает исходный код восстановленной Delphi формы с подсветкой синтаксиса, сворачиванием, поиском и копированием.
- **«ИИ Помощник» (AI Helper)** обрабатывает существующий декомпилированный листинг с использованием выбранной модели или сервиса API.

«Парсить парам. из стека» (Parse stack parameters) и **«Анализ и оптимизация кода» (Procedure analyzer and optimizer)** влияет на восстановление текущего листинга. Если вывод неясен, сравните его с отключёнными параметрами обработки и проанализируйте соответствующее дизассемблирование. При наличии кэшированного или полностью обработанного вывода выполните повторную обработку файла после изменения настроек.

Навигация

Кнопки со стрелками над деревом перемещают по истории просмотров. **Esc** а также кнопки мыши «Назад» и «Вперёд» осуществляют навигацию по одной и той же истории в представлениях кода, объявлений, форм и исходного текста. Возврат к форме восстанавливает выбранный элемент управления, страницу и прокрутку; возврат к коду восстанавливает положение курсора и области просмотра. **Alt+1** переключает фокус между деревом и активным представлением, а также **Enter** открывает выбранный элемент дерева. В HEX-редакторе, **Tab** перемещает между полями шестнадцатеричного и текстового представлений, а затем к следующему элементу управления.

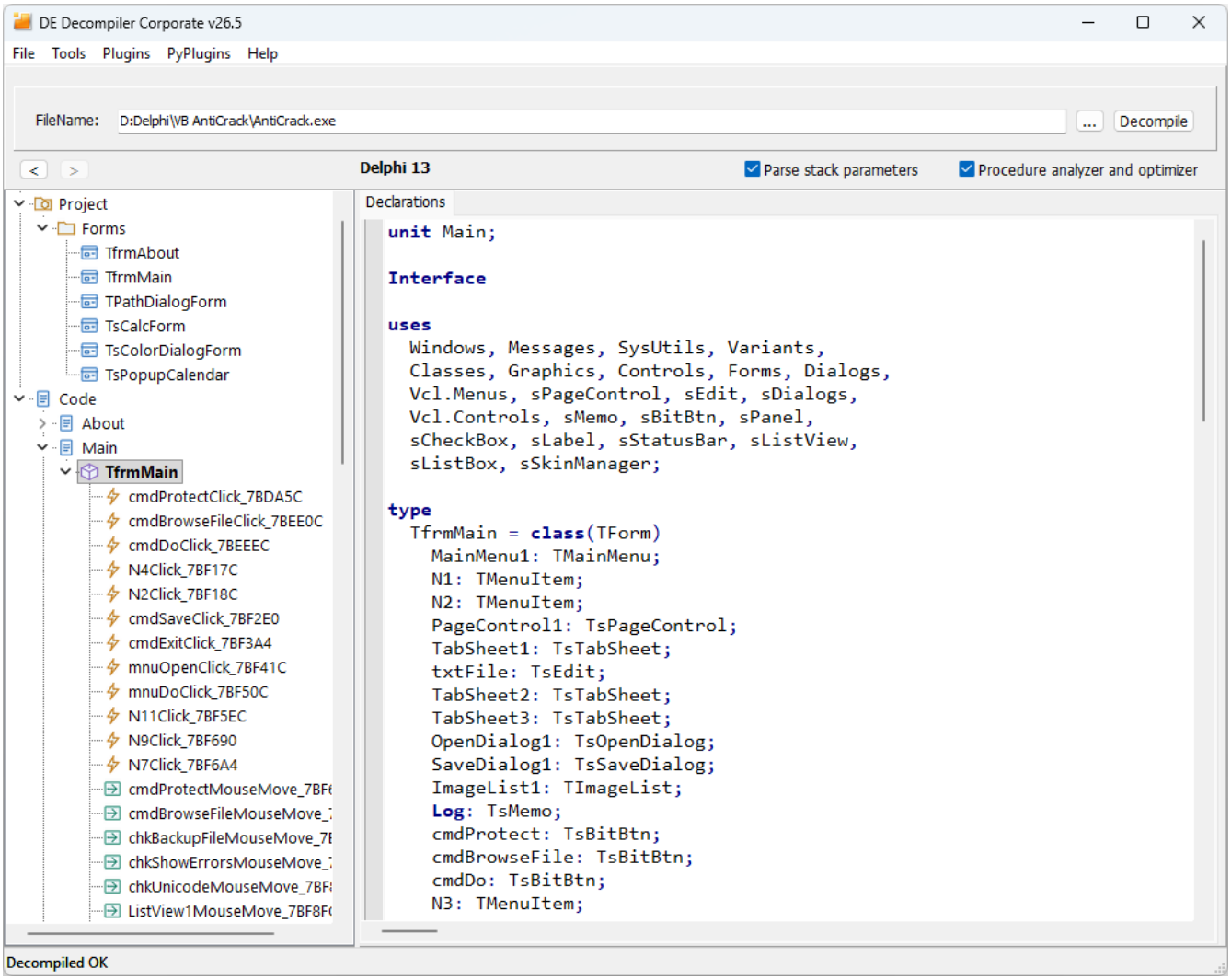
«Стиль окна как в Visual Studio» (Window style as in Visual Studio) размещает код слева, а дерево проекта — справа. Доступные вкладки зависят от выбранного объекта. См. Просмотр и сравнение кода для контекстно-зависимых сочетаний клавиш.

1.5 Модули, классы и функции

Восстановленные объявления помогают разобраться в структуре поддерживаемых VCL-приложений, написанных на Delphi или созданных в C++Builder.

Объявления модулей

Выберите модуль под **Код** для просмотра его восстановленного листинга объявлений. Используйте имя модуля, доступные ссылки, объявленные классы и видимые члены, чтобы определить, какую часть приложения вы изучаете.



Восстановленные объявления модулей и классов для приложения Delphi.

Выдержка из восстановленного объявления класса:

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

Классы и методы

Разверните распознанный класс и выберите метод или функцию. Читайте сигнатуру и тело вместе: правдоподобное тело само по себе не подтверждает предполагаемый тип параметра или интерпретацию поля класса.

Публичные и приватные функции

Распознанные публичные функции и методы отображаются в восстановленной структуре. Декомпилятор также пытается найти приватные функции в каждом модуле. Поиск приватных элементов эвристический; исследуйте неожиданные границы, отсутствующие аргументы или несогласованные ассоциации модулей с помощью дизассемблирования.

Кадровые структуры анонимных методов, сгенерированных компилятором, скрыты от обычного дерева. Результаты поиска, переходы по адресам и история могут выявить необходимый кадр. **«Показывать VCL классы» (Show VCL classes)** предоставляет полный вид класса. Открытие объявления модуля или класса не требует декодирования всех его методов.

Имена и аннотации аналитика

Различайте имена, сохранённые в бинарном файле, от сгенерированных идентификаторов и имён, назначенных во время анализа. Восстановленное или назначенное аналитиком имя может помочь объяснить процедуру, но не доказывает, что это был исходный идентификатор.

2 Открытие и сохранение

2.1 Открытие файла

Выберите **«Файл > Открыть программу» (File > Open program)**, чтобы открыть бинарный файл поддерживаемого VCL-приложения для Win32 или Win64, написанного на Delphi или созданного в C++Builder.

VCL-приложения, созданные в C++Builder, открываются тем же способом. В поле компилятора C++Builder и Free Pascal отличаются от Delphi; подсказка показывает полные сведения о компиляторе и базе сигнатур.

1 Выберите **«Файл > Открыть программу» (File > Open program)** и укажите входной файл.

2 Дождитесь заполнения дерева проекта и завершения начальной обработки.

3 Проверьте восстановленную структуру кода и форм перед переходом к отдельным процедурам.

Перетаскивание и открытие через командную строку

Вы можете перетащить бинарный файл из Проводника на DE Decompiler или передать его путь в командной строке. Распознавание зависит от содержимого файла, поэтому незнакомое расширение файла или его отсутствие само по себе не препятствует анализу. Поддерживаемые ярлыки Windows могут быть разрешены до ссылаемых файлов.

Сохраненные базы анализа также можно открыть путем перетаскивания или передачи их пути в командной строке. При работе с несколькими сборками сохраняйте однозначность путей и версий файлов.

Отмена начального анализа

Улучшена отмена открытия больших приложений, поиска классов, чтения форм и полной декомпиляции. После запроса отмены дальнейшая обработка процедур прекращается; неполный результат полной декомпиляции по-прежнему отбрасывается.

При отмене необязательного анализа прототипов разобранный файл остаётся открытым, а уже найденные прототипы, API и строки сохраняются. Незавершённые отчёты не выводятся; после прерывания восстанавливаются параметры декодирования.

Входные данные с ограниченным распознаванием

Доступные инструкции и метаданные определяют детализацию восстановленных имен, типов и вызовов. Вариации компилятора, необычная информация о типах, защита и повреждение файла могут повлиять на распознавание; проверьте неожиданный результат в представлении нативных инструкций. См. Совместимость компиляторов и RTTI и Интерпретация восстановленного кода.

Для воспроизводимого анализа сохраняйте точный бинарный файл и фиксируйте использованные настройки. Более поздняя сборка с тем же именем файла может иметь другие адреса, структуру и поведение.

2.2 Базы анализа

База данных анализа сохраняет восстановленную информацию о проекте, чтобы вы могли вернуться к расследованию без необходимости каждый раз начинать работу с бинарного файла.

Используйте **«Файл > База Данных» (File > Data Base)** для сохранения или открытия результатов анализа. Сжатая база VDD содержит восстановленную структуру приложения, формы и ресурсы, имена, заданные аналитиком, метаданные методов и кэшированные листинги. Сохранение результатов полной обработки особенно полезно, если первоначальный анализ крупного приложения занимает много времени.

Вы также можете открыть сохраненную базу данных через рабочий процесс открытия файлов, перетащив ее на окно или передав путь в командной строке. При передаче материалов расследования между рабочими станциями храните соответствующий исходный бинарный файл вместе с базой данных.

Просмотр базы данных без исполняемого файла

Кэшированный нативный код остаётся доступен и без исходного исполняемого файла. Также можно просматривать сохранённые ресурсы форм и восстановленную структуру. Если процедура не была декомпилирована до сохранения базы, для неё потребуется соответствующий исходный файл. DE Decompiler не будет декомпилировать её при отсутствии или изменении этого файла.

Просмотрите или экспортируйте доступные кэшированные данные и проверьте любые сообщения, указывающие на процедуры, которые не были декодированы. Для создания переносного снимка расследования обрабатывайте необходимые процедуры перед сохранением и по возможности храните точный входной бинарный файл вместе с базой данных.

Сохранение и совместимость

Сохранение базы данных поддерживает отмену и заменяет существующую базу VDD только после полного завершения записи. Если отдельную процедуру не удается декодировать, база данных все равно может сохранять остальные данные анализа; просмотрите предупреждения, отображаемые после сохранения.

Новые базы могут содержать расширенные данные модели и ресурсов. Более старая сборка может отображать только совместимую с ней часть данных. Для просмотра всех сохранённых сведений используйте актуальную сборку DE Decompiler.

Что следует сохранять

- Аналитическую базу данных и точный бинарный файл, который она описывает.
- Сборку DE Decompiler и соответствующие настройки обработки.
- Любые отдельные заметки, имена, присвоенные аналитиком, и экспортированные материалы, используемые в расследовании.

База данных является артефактом анализа. Она отличается от восстановленных исходных файлов и не заменяет оригинальный исполняемый файл как источник адресов и байтов.

Проверка сохраненного анализа

После открытия базы данных на другой рабочей станции проверьте известный модуль, метод и форму. При сравнении результатов от разных сборок декомпилятора сохраняйте более раннюю базу данных и фиксируйте, какая сборка сгенерировала каждый результат.

2.3 Сохранение результатов

Сохраняйте выходные данные, необходимые для расследования, сохраняя при этом исходный бинарный файл и контекст анализа.

Вывод проекта и модуля

«Файл > Сохранить декомпилированный проект» (File > Save decompiled project) экспортирует восстановленный материал в проект Delphi, включая восстановленные модули и формы. Выберите папку вывода и проверьте сохранённые файлы:

- `.pas`: один исходный файл UTF-8 для каждого модуля, объединяющий объявления и методы из его восстановленных классов.
- `.dfm`: отдельные определения форм, включая раздельные файлы, когда несколько форм принадлежат одному модулю. Строковые литералы не-ASCII используют коды символов Delphi # , а двоичные данные свойств сохраняются.
- `Project.dpr`: ссылки проекта и восстановленная информация о запуске на основе успешно сохранённых во время текущего экспорта модулей и ресурсов форм.

Основная форма связана через `{$R *.dfm}`; дополнительные формы используют явные директивы ресурсов. Анализ только форм экспортирует доступные файлы DFM для проверки без создания отсутствующих классов Pascal или кода запуска. Проверьте диагностику экспорта на наличие отсутствующих или конфликтующих ресурсов.

Используйте **«Файл > Сохранить все в один модуль» (File > Save all in one module)** когда единый объединённый `.pas` листинг удобнее для просмотра или сравнения. Объединённый листинг и многофайловый проект Delphi служат разным целям; выбирайте вывод, требуемый расследованием.

Тела процедур с комментариями

По умолчанию восстановленные локальные объявления и операторы помещаются в комментарий Pascal, а сигнатура метода сохраняется. Тело процедуры остаётся доступным для изучения, поиска и анализа. Такой формат не означает, что экспортированный модуль готов к компиляции.

Прогресс, отмена и ошибки

При экспорте проекта или объединённого листинга процедуры обрабатываются по мере необходимости. Проверяйте предупреждения о методах, которые не удалось декомпилировать. Сохранённый кэшированный код можно экспортировать без исходного исполняемого файла. Для ещё не декомпилированных процедур потребуется соответствующий бинарный файл.

Используйте **«Отмена» (Cancel)** для остановки между процедурами. Существующие файлы PAS, DFM, DPR и объединённого листинга заменяются только после полного завершения записи новых файлов. При отмене экспорта многофайлового проекта уже записанные файлы сохраняются, поэтому в каталоге могут оказаться как новые, так и прежние файлы. Чтобы сохранить предыдущий экспорт отдельно, используйте другой каталог.

Подготовка материала для IDE

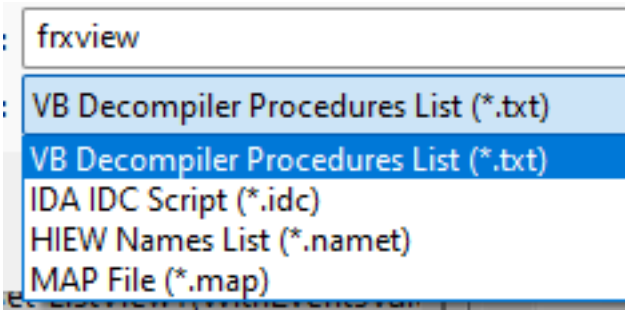
Перед открытием восстановленного кода в IDE вручную устраните неточности декомпиляции и несовместимости. Приведите в соответствие сгенерированные идентификаторы, предполагаемые типы и сигнатуры, неразрешённые вызовы и отсутствующие зависимости. Листинг восстанавливает логику программы для анализа; он не воспроизводит исходный проект, готовый к компиляции. Сборка требует ручного анализа и исправления экспортированного кода.

Записи расследования

Для проверки безопасности сохраняйте адрес процедуры и идентификацию двоичного файла вместе с экспортированными фрагментами. Убедитесь, что в итоговом отчёте различимы декомпилированные операторы, нативные инструкции, предложения, сгенерированные моделью, и выводы аналитика.

2.4 Сохранение списка процедур

Файл > Сохранить список процедур экспортирует распознанные имена процедур и адреса для использования в других рабочих процессах анализа.



Форматы экспорта списка процедур.

Выберите требуемый тип файла в диалоге сохранения:

Формат	Используйте
`TXT`	Простой текстовый список процедур для обзора, сравнения и записи адресов/имён.
`IDC`	Скрипт IDA для передачи распознанных имён функций и адресов.
`NAMET`	Список имён для использования с HIEW.
`MAP`	Файл карты для совместимых отладчиков, дизассемблеров и рабочих процессов импорта символов.

Во всех форматах сохраняются полные виртуальные адреса Win64. При экспорте IDC используются относительные адреса с учётом новой базы загрузки и проверяется разрядность. Существующие инструкции и данные IDA сохраняются. Сохранение можно отменить; файл заменяется только после полной записи, поэтому при ошибке прежний список остаётся доступен.

Экспортированный список описывает распознанные процедуры. Обнаружение частных функций эвристическое, поэтому отсутствие адреса в списке не гарантирует, что по этому адресу нет исполняемого кода. Сохраняйте исходный входной файл и его контекст адресов вместе с экспортированным списком.

3 Изучение приложения

3.1 Просмотр и сравнение кода

Изучайте восстановленные операторы Delphi-подобного Pascal, чтобы понять работу процедуры, и сверяйте их с машинными инструкциями. Этот порядок анализа применим к поддерживаемым программам, созданным как в Delphi, так и в C++Builder.

Следование по листингу

Выберите процедуру в дереве. Изучите сигнатуру, локальные объявления и поток управления перед переходом к отдельным выражениям. Сгенерированные имена идентифицируют восстановленные значения там, где недоступно оригинальное имя из исходного кода. Подсветка синтаксиса и сворачиваемые блоки помогают ориентироваться в длинных процедурах.

Восстановленные тела могут быть заключены в обёртки комментариев Pascal для экспорта. DE Decompiler по-прежнему рассматривает эти тела как код для подсветки идентификаторов, поиска ссылок и переименования. Строковые литералы и обычные комментарии остаются исключёнными из совпадений ссылок на код.

Идентификаторы и навигация по коду

Установите курсор на имя регистра, переменной или функции, чтобы выделить все совпадающие вхождения в текущем листинге. Выделение локального или глобального идентификатора в одной процедуре отделено от поиска ссылок во всём проекте.

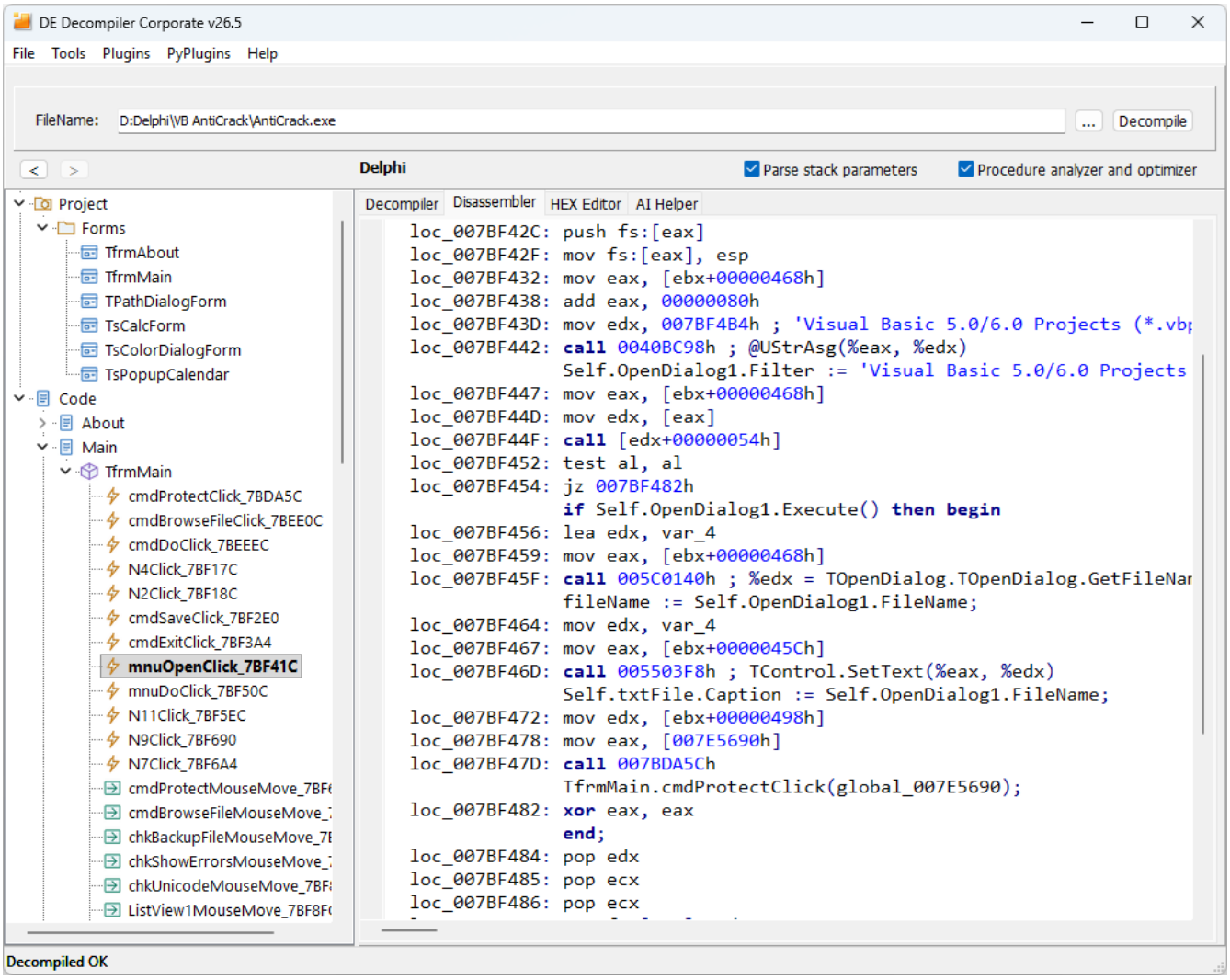
Дважды щёлкните по распознанному вызову процедуры, чтобы открыть вызываемую процедуру. Константы, похожие на адреса, могут привести к соответствующим байтам файла в HEX-редакторе. Используйте **Esc**, стрелки истории или кнопки мыши «Назад» и «Вперёд», чтобы вернуться.

Синхронизация адресов

Переключение между Декомпилятором и Дизассемблером следует текущему адресу курсора, включая полные Win64 адреса. Если отсутствует метка идентичной инструкции, выбирается ближайший доступный адрес. Открываются только свёртки, содержащие пункт назначения.

Смешанные представления высокого уровня и нативного кода

Щёлкните правой кнопкой мыши по коду и выберите **«Скопировать в дизассемблер» (Copy to disassembler)**, или нажмите **D** в представлении кода. Полученное представление размещает восстановленные операторы с метками адресов рядом с соответствующими нативными инструкциями. Свёрнутый исходный код включается без разворачивания оригинального редактора, а объявления без адресов исключаются из аннотаций. Сохраняйте **«Добавлять адреса» (Add Addresses)** включённым при использовании сравнения на основе адресов.



Восстановленные Delphi операторы переплетены с нативными инструкциями.

Ссылки и имена

Используйте **«Найти ссылки» (Find references)** на процедуре, выбранной в дереве, или нажмите **X**, чтобы просмотреть распознанные вызывающие объекты. Дважды щёлкните по результату для анализа вызывающего объекта. Поиск ссылок требует полной обработки: отключите **«Быстрая декомпиляция» (Fast decompilation)** и повторно откройте входные данные, если будет предложено. Список ссылок на основе не декодированных кэшированных тел является неполным. В представлении кода установите курсор на глобальную переменную и нажмите **X** для поиска её ссылок. Косвенные или динамически разрешаемые использования могут отсутствовать в списке распознанных ссылок.

Используйте **«Переименовать переменную» (Rename variable)** или **N** для присвоения осмысленного имени анализа. Имена подчиняются правилам идентификаторов Pascal. Переименование с учётом совпадений нечувствительно к регистру и проверяет коллизии с параметрами, локальными объявлениями и зарезервированными ключевыми словами. **«Отменить переименование» (Undo renaming)** или **Ctrl+Shift+Z** отменяет переименование переменной независимо от обычного отмены редактирования текста. Сохраняйте имена, присвоенные аналитиком, отличными от оригинальных имён, сохранённых в бинарном файле.

Клавиатурный доступ

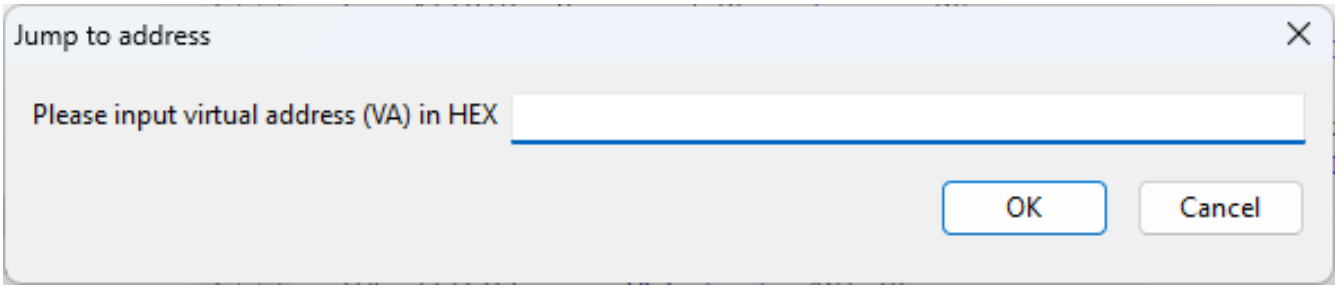
Фокус	Сочетание клавиш	Действие
Код	D	Объедините восстановленный листинг с дизассемблированием.
Код	N	Переименовать выбранную переменную.
Код	Ctrl+Shift+Z	Отменить переименование переменной.
Код	X	Найти ссылки на глобальную переменную под курсором.
Дерево	X	Найти ссылки на выбранную процедуру.
Дерево	G	Переход по пути раздела.
Дерево	Ctrl+C	Скопировать полный путь к выбранному объекту в дереве.
Дерево	Пробел или Insert	Отметить или снять отметку с выбранного элемента.
Представления	Alt+1	Переключить фокус между деревом и активным представлением.
История	Esc	Вернуться к предыдущей процедуре.

Однобуквенные сочетания зависят от клавиатурного фокуса. Они не заменяют обычное вводе текста в полях ввода.

3.2 Навигация по адресу и пути дерева

Инструменты > Перейти к виртуальному адресу открывает позицию по её виртуальному адресу.

Используйте **Ctrl+G** или выберите **«Утилиты > Перейти на виртуальный адрес» (Tools > Jump to virtual address)**, введите требуемый адрес и подтвердите. Адреса относятся к текущему анализируемому сборке. Файловые смещения и виртуальные адреса представляют различные системы координат; сохраняйте это различие при сравнении листинга с байтами в файле.



Ввод виртуального адреса для навигации.

Обмен позицией в дереве

В дереве проекта используйте **«Скопировать имя секции» (Copy the name of section)** или **Ctrl+C** чтобы скопировать путь выбранного объекта. Используйте **«Перейти используя путь секции» (Jump to the section)** или **G** для навигации по скопированному пути в соответствующем анализе.

Вы также можете выбрать адрес в листинге перед вызовом **Ctrl+G**. Используйте историю просмотра для возврата после перехода по ссылке; кнопки мыши «Назад» и «Вперёд» сохраняют позицию внутри процедуры. Путь или адрес, переданный другому аналитику, имеет смысл только в контексте той же бинарной модели и совместимой восстановленной структуры.

3.3 Поиск процедур и текста

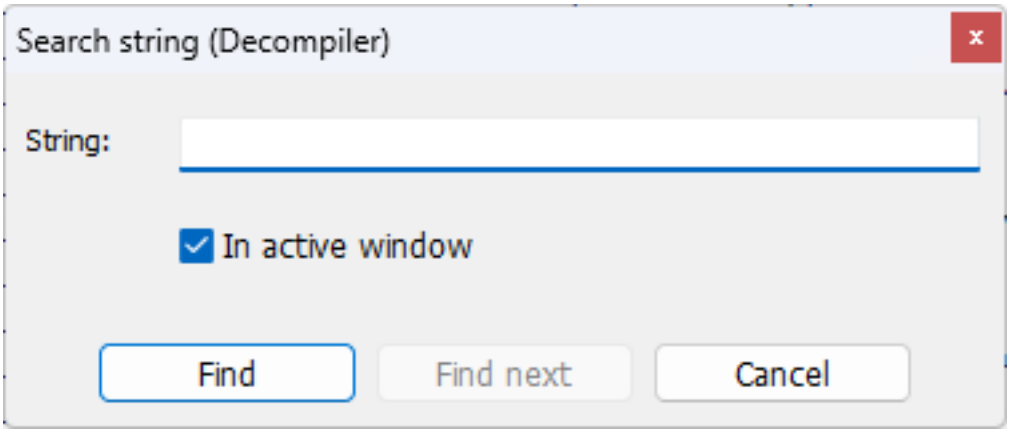
Поиск по имени процедуры и поиск по тексту листинга служат разным целям.

Найти процедуру

Используйте **«Утилиты > Найти процедуру» (Tools > Find procedure)** или **Ctrl+T** и введите полное имя или отличительную часть. Совпадение не зависит от регистра и принимает квалифицированные имена процедур. Выберите требуемое совпадение из результатов.

Найти отображаемый текст

Используйте **Инструменты > Найти строку** или **Ctrl+F** для открытия поиска текста. С **«В активном окне» (In active window)** выбранным, поиск применяется к текущему виду кода или данных формы. Снимите этот флажок, чтобы искать в декомпилированном проекте. Введите текст в поле **Строка**, затем используйте **«Искать» (Find)** и **«Искать далее» (Find next)** для перехода между совпадениями.



Поиск текста в активном представлении кода.

Для высокоразмерного поиска текста по всему проекту требуется предварительная декомпиляция соответствующих процедур. Отключите **«Быстрая декомпиляция» (Fast decompilation)** и обработайте файл повторно, когда вам необходимо полное покрытие для высокоразмерного поиска текста.

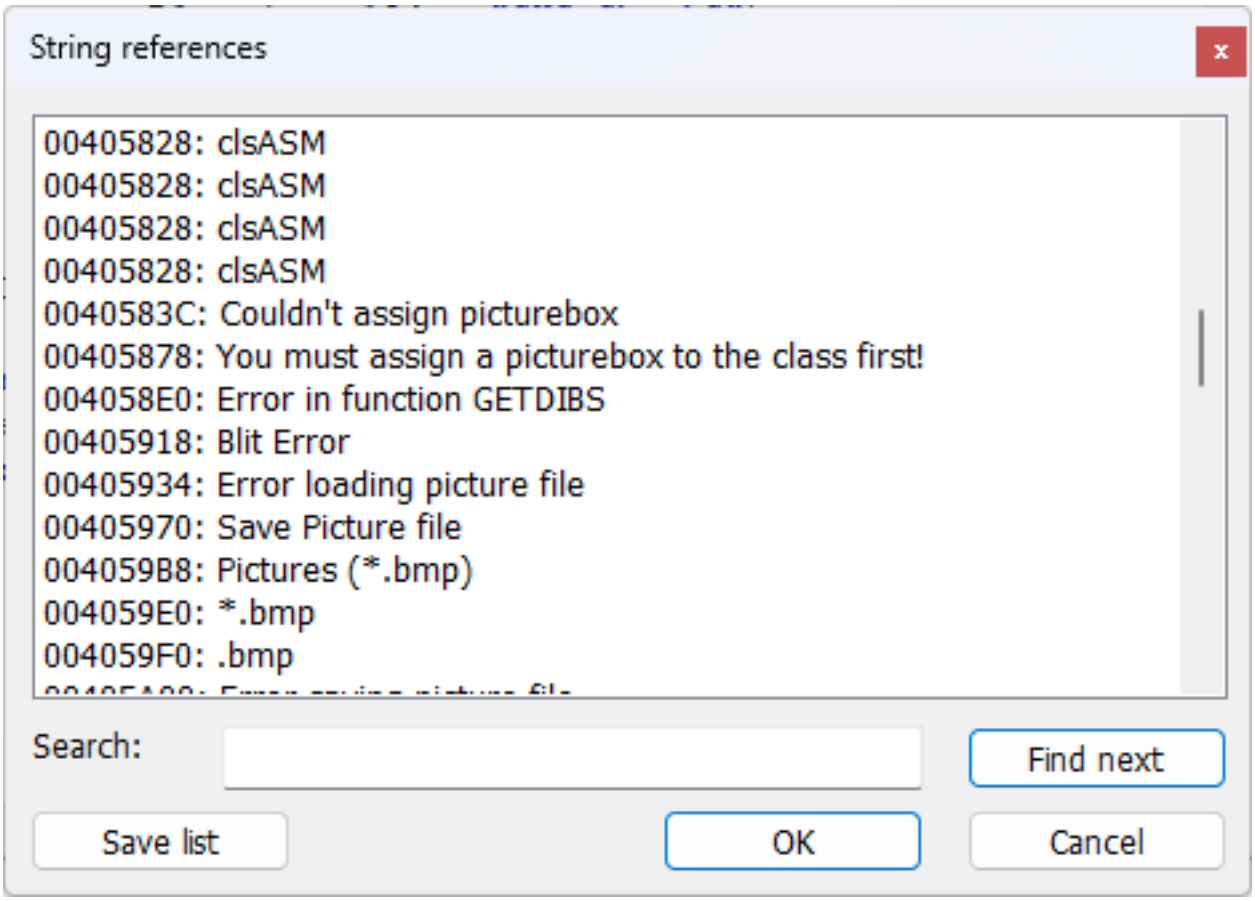
Поиск текста включает свёрнутое содержимое в представлениях Decompiler, Disassembler, DFM source и AI. Найденное совпадение отображается, тогда как несвязанные свёртки остаются закрытыми.

Строковые литералы

Используйте Ссылки на строки для исследования распознанных строковых литералов и их расположения. Эти ссылки могут быть собраны посредством предварительного дизассемблирования в режиме быстрой декомпиляции; для этого не требуется наличие полного высокоразмерного листинга для каждой процедуры.

3.4 Ссылки на строки

Инструмент Tools > String references помогает находить распознанные литералы и связанные с ними процедуры. Просмотр ссылок на строки доступен в Free Lite и коммерческих изданиях.



Браузер ссылок на строки.

Используйте **Поиск** и **«Искать далее» (Find next)** для поиска интересного литерала. Дважды щёлкните по строке, чтобы открыть процедуру, которая на неё ссылается; повторное двойное щелчок продолжит поиск её использований. **OK** переходит к выбранной строке и закрывает диалог; **«Отмена» (Cancel)** закрывает диалог.

Список содержит строки, собранные во время дизассемблирования. В режиме быстрой декомпиляции предварительный анализ может предоставить эти ссылки при **«Анализ прототипов» (Analyze prototypes)** включено. Если DE Decompiler сообщает о необходимости полного прохода, отключите **«Быстрая декомпиляция» (Fast decompilation)** и повторно откройте входной файл. В Lite выбор ссылки открывает нативное дизассемблирование. Изучайте процедуры-источники и соответствующие байты, чтобы определить, как используется значение. Пути, строки, связанные с реестром, сетевые адреса и фрагменты команд могут помочь расставить приоритеты при обзоре кода.

Ссылка является указателем анализа. Наличие строки в процедуре не гарантирует, что она передаётся в определённый API или что соответствующий ветвь выполняется. Зашифрованные, динамически сконструированные или иным образом нераспознанные строки могут отсутствовать.

Редактирование распознанной строки

Щёлкните правой кнопкой мыши по поддерживаемой строке и выберите **«Редактировать строку» (Edit string)**. Заменяемая строка не должна быть длиннее исходной, поскольку файл не резервирует место для большего значения. Подтверждение редактирования изменяет файл напрямую; эта операция не создаёт резервную копию и не может быть отменена через диалог редактирования строк. Работайте с копией и сохраняйте неизменный образец.

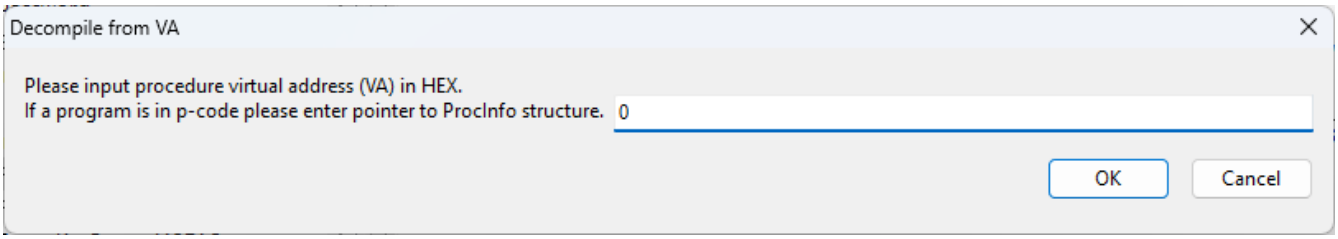
В коммерческих изданиях контекстное меню также предоставляет **«HEX Редактор» (HEX Editor)** для просмотра байтов строки. HEX-редактор поддерживает отмену и повтор своих изменений. Проверяйте получившиеся байты и затронутый код после внесения изменений.

3.5 Декомпиляция по адресу

Инструменты > Декомпилировать по VA (виртуальному адресу) запрашивает декомпиляцию с указанного виртуального адреса.

Этот рабочий процесс полезен при исследовании кода, который не представлен в виде ожидаемой процедуры в восстановленном дереве. Если файл уже открыт, введите виртуальный адрес для обработки. Если файл не открыт, DE Decompiler предложит выбрать его и откроет в упрощенном режиме.

Изучайте полученные нативные инструкции и восстановленный листинг совместно. Без нормальной начальной обработки некоторая информация о импортируемых вызовах и типах может быть недоступна.

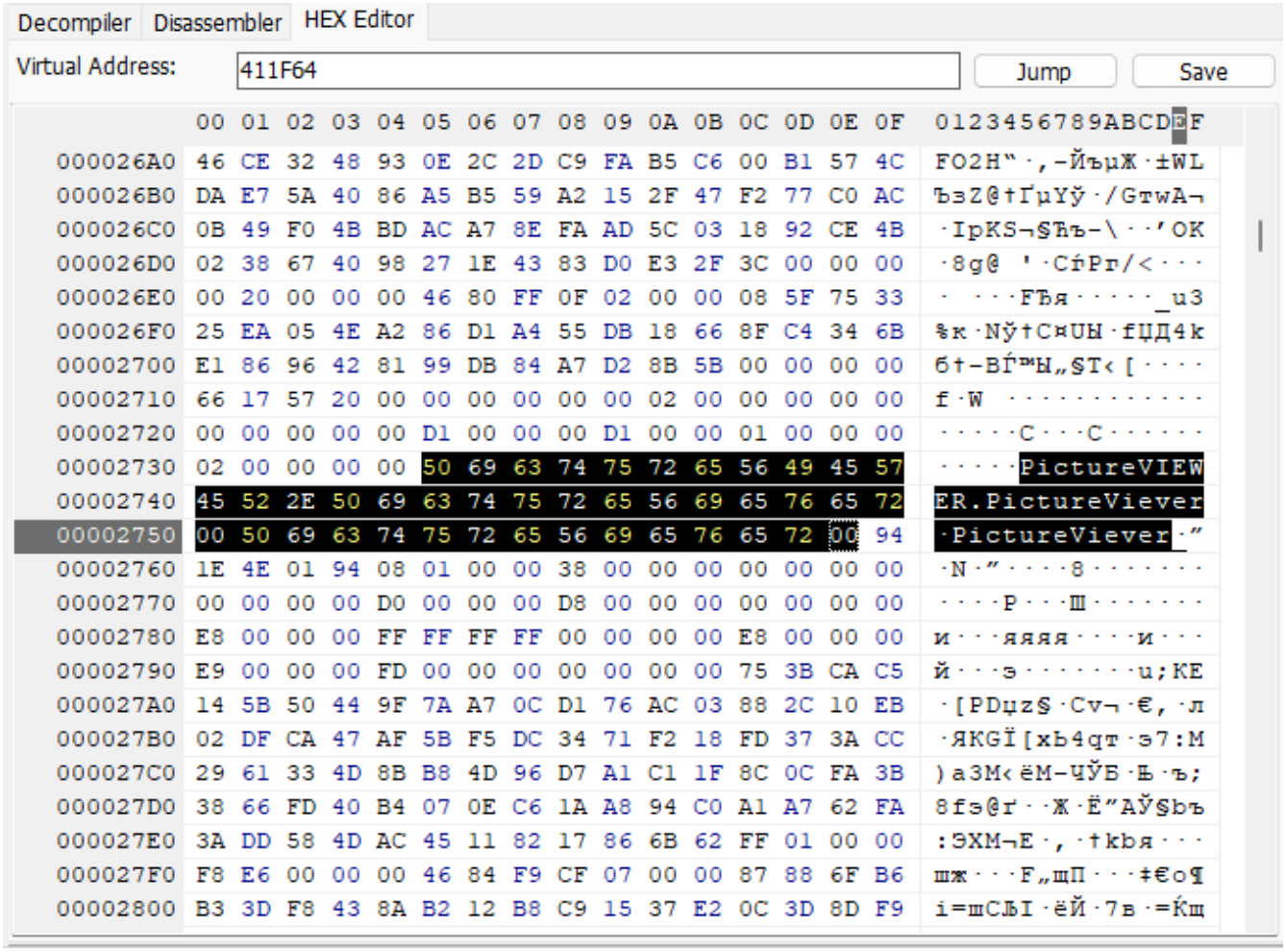


Диалоговое окно декомпиляции по адресу.

Сам по себе выбор адреса не подтверждает границу функции или её сигнатуру. Неверная точка входа может привести к ошибочным операторам, интерпретации аргументов или неполному результату. Проверьте окружающие инструкции и распознанные места вызова, прежде чем полагаться на восстановленный код.

3.6 Просмотр и изменение байтов

Используйте вкладку HEX-редактор для исследования и редактирования байтов файла, лежащих в основе восстановленного кода и данных форм.



Инспекция бинарных данных в HEX-редакторе.

Исследование данных файла

Откройте **«HEX Редактор» (HEX Editor)** рядом с представлением кода или формы. Навигация осуществляется с помощью виртуального адреса, включая **Базовый адрес образа**, или смещение файла, отсчитываемое от начала файла. Сравните отображаемые байты с соответствующими машинными инструкциями или восстановленными данными формы.

Редактирование байтов

HEX-редактор поддерживает редактирование отдельных байтов и блоков данных внутри файла. Перед внесением изменений убедитесь, что адрес задан как виртуальный адрес или смещение файла.

Работайте с отдельной копией для анализа. При необходимости перезагрузите или повторно обработайте затронутый код после внесения изменений, особенно если включено кэширование листингов. Сохраняйте как исходные байты, так и запись об изменении при сравнении поведения или документировании расследования.

3.7 Формы VCL и визуальный просмотр

DE Decompiler декомпилирует информацию о форме VCL и визуализирует восстановленные элементы управления и свойства.

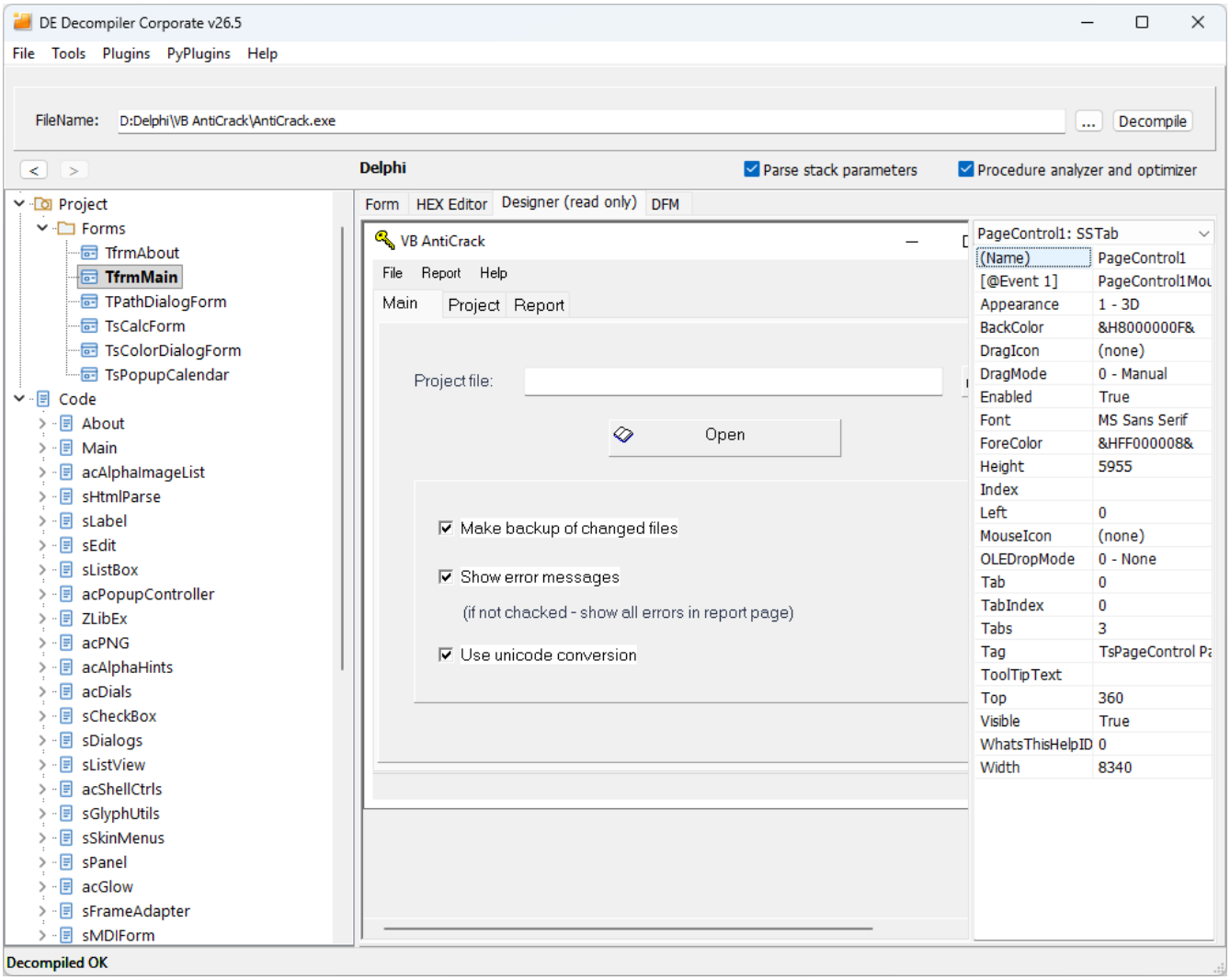
Тот же порядок просмотра форм VCL и переходов к обработчикам событий доступен для приложений, созданных в C++Builder.

1 Расширьте **«Формы» (Forms)** в дереве проекта и выберите форму для изучения.

2 Откройте **«Дизайнер (только чтение)» (Designer (read only))** для просмотра восстановленной визуальной компоновки.

3 Выберите элемент управления и изучите его доступные свойства и восстановленные записи событий в области свойств.

4 Сравните представление формы с доступными **DFM**данными формы и видами HEX.



Восстановленная форма VCL с ее восстановленными свойствами компонентов.

Используйте FRX Image Viewer для предварительного просмотра и экспорта зарегистрированных изображений форм и распознанных нативных значков ImageList.

DFM исходный код и визуальный предварительный просмотр

DFM представление исходного кода сохраняет восстановленные имена классов компонентов, свойства и идентификаторы обработчиков событий. Оно поддерживает подсветку синтаксиса, подсветку совпадающих идентификаторов, поиск и копирование. Сверните `object`, `inherited` и `inline` блоки, элементы коллекций, двоичные свойства и списки строк для навигации по большим формам.

Вкладка **Form** содержит преобразованное представление, используемое для визуализации. Сохранённые значения свойств и исходные имена классов компонентов смотрите в восстановленном тексте DFM.

Предпросмотр восстанавливает сохранённые страницы, меню, списки, геометрию сеток и кнопки панелей инструментов, включая поддерживаемую графику ImageList, маски и альфа-прозрачность. Поддерживаемые элементы управления AlphaSkins отображаются с их VCL-эквивалентами, при этом имена классов DFM остаются доступными в исходном коде. Код отрисовки и обработчики событий анализируемого приложения не выполняются.

Переход к обработчикам событий

Выберите восстановленное событие элемента управления или меню в представлении формы, чтобы открыть обработчик. Возврат по истории восстанавливает выбранный элемент, активную вкладку и положение прокрутки. Изучайте процедуру на вкладке **«Декомпилятор» (Decompiler)** и сверяйте её с представлением **«Дизассемблер» (Disassembler)**, чтобы понять работу обработчика.

Данные формы и смещения

«Показывать смещения» (Show offsets) добавляет информацию о расположении к представлению данных формы. Используйте эту опцию при сопоставлении восстановленного свойства или определения с его бинарным представлением.

Интерпретация предпросмотра

Конструктор представляет собой режим просмотра для анализа восстановленных данных формы. Это не оригинальное приложение, выполняющее инициализацию или код событий. Элементы управления, созданные во время выполнения, и значения, присваиваемые только программным путём, требуют исследования в соответствующих процедурах.

Пользовательские элементы управления и распознавание типов, специфичных для версии, могут влиять на представление формы. Изучайте восстановленные данные и код, если компонент, свойство или макет выглядят неполными.

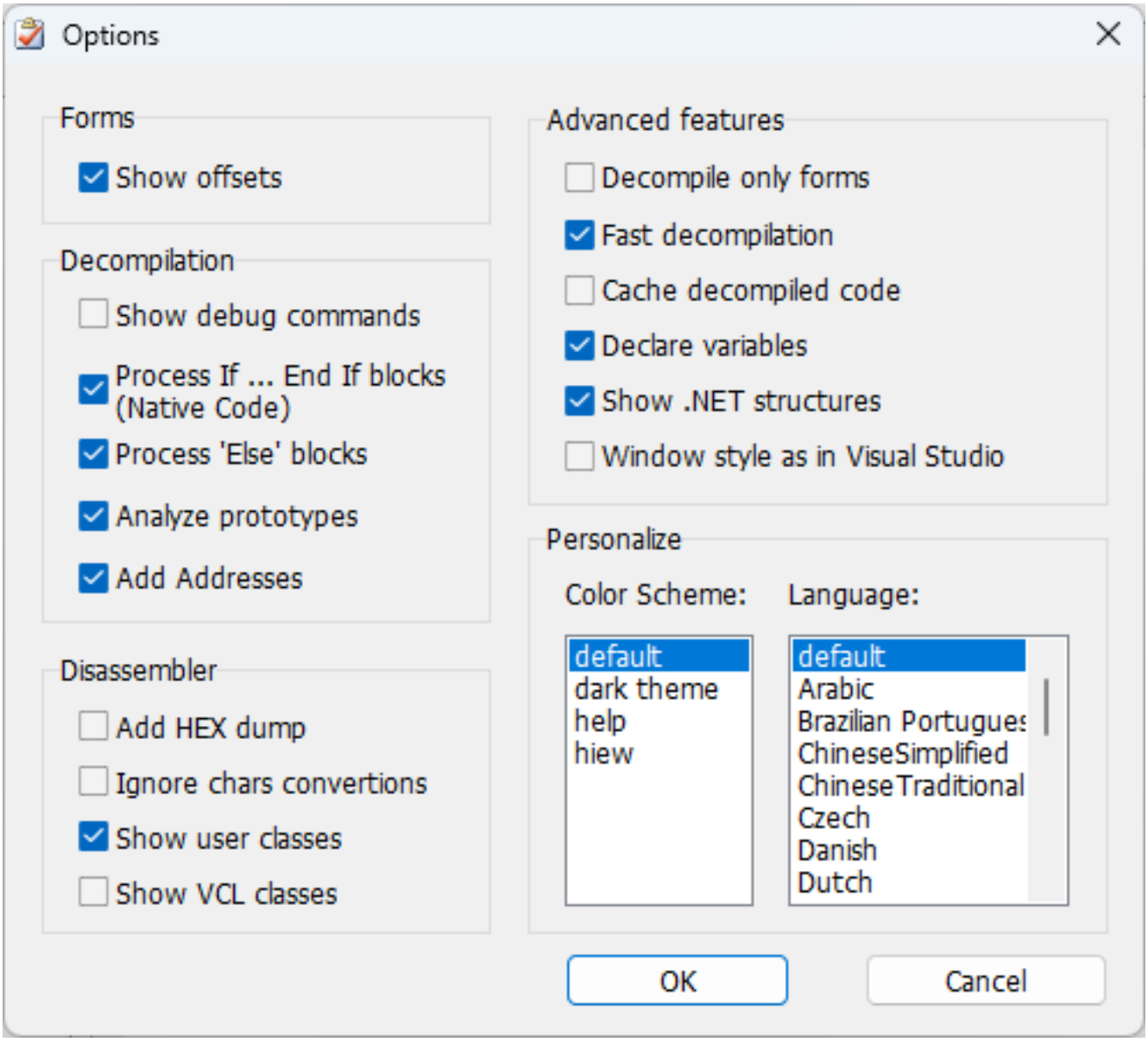
Компоненты TMS и статическая визуализация

Визуализация поддерживает распознанные надписи, кнопки, списки, группы, ползунки, полосы прокрутки и вкладки TMS с учётом сохранённых текста и расположения. Индикаторы активности показываются статически, а встроенные веб-области обозначаются без загрузки страниц.

4 Параметры и инструменты

4.1 Параметры программы

Инструменты > Параметры управляют реконструкцией, представлением, кэшированием и языком интерфейса.



Диалог параметров DE Decompiler, включая настройки пользовательского класса и VCL-класса.

Следующие разделы описывают параметры анализа поддерживаемых VCL-приложений, написанных на Delphi или созданных в C++Builder:

- Формы и визуальный осмотр
- Параметры декомпиляции
- Отображение дизассемблированного кода
- Темы оформления и шрифты
- Режимы обработки и кэширование
- Язык интерфейса

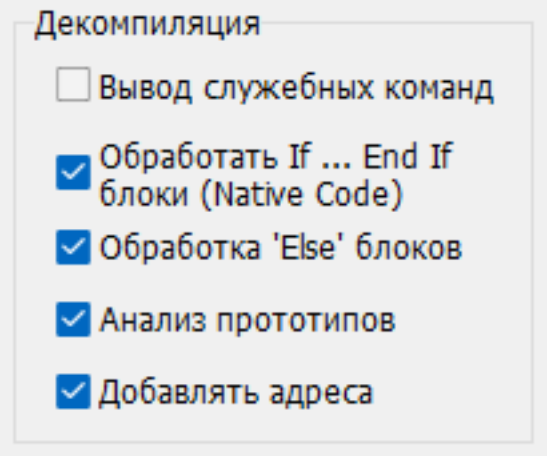
При сравнении настроек обработки сохраняйте входной двоичный файл и записывайте, какие настройки дали каждый листинг. Кэшированный вывод может потребовать повторной обработки, чтобы отразить измененную опцию.

4.2 Параметры декомпиляции

При декомпиляции нативного кода Delphi-подобный Pascal восстанавливается по машинным инструкциям и доступным метаданным. Эти параметры обработки применимы к поддерживаемым программам, написанным на Delphi или созданным в C++Builder.

Восстановленная логика программы

Декомпилятор восстанавливает условия, `case` блоки, циклы и вызовы, включая распознанные методы VCL. Имена и типы на уровне исходного кода зависят от информации, сохраненной в бинарном файле. Сгенерированные имена переменных, выведенные структуры расположения полей и вызовы без распознанных имен VCL являются нормальными последствиями реконструкции нативного кода. См. Интерпретация восстановленного кода для подробностей.



Элементы управления обработкой декомпиляции.

Выражения Delphi и операции времени выполнения

Распознанный поток управления включает `if/else`, `case`, `for`, `while` и `repeat/until`, с `Break` и `Continue` где нативные ветви устанавливают свое значение.

Доступные данные RTTI, метаданные классов и сигнатуры поддерживают поля объектов, свойства, виртуальные методы, вызовы VCL и импортированные вызовы Windows API. Общие операции RTL могут быть представлены как выражения строк Delphi, `Length`, `Copy`, `Delete`, `Insert`, изменение размера массива и индексация, доступ к Variant-массивам и `TextFile` операции ввода.

Эти преобразования зависят от распознанных типов, соглашений вызова и потока данных. Если доказательств недостаточно, листинг сохраняет явные вызовы, указатели или ветвления. Сгенерированные семантические имена переменных описывают выведенные роли и не являются восстановленными оригинальными идентификаторами.

Параметры стека и оптимизация

«Парсить парам. из стека» (Parse stack parameters) способствует реконструкции вызовов и выражений высокого уровня. **«Анализ и оптимизация кода» (Procedure analyzer and optimizer)** упрощает распознавание паттернов выражений и потока управления. Сравнивайте полученный листинг с отключенными элементами управления, если выражение или ветвление неясны.

Обработка потока управления

«Обработать If ... End If блоки (Native Code)» (Process If ... End If blocks (Native Code)) реконструирует распознанные условные переходы в высокоуровневые условные блоки. **«Обработка 'Else' блоков» (Process 'Else' blocks)** добавляет эвристики для ветвей Else и требует обработки блоков If. Сравнивайте эти элементы управления включенными и выключенными при расследовании неоднозначного ветвления.

Прототипы и адреса

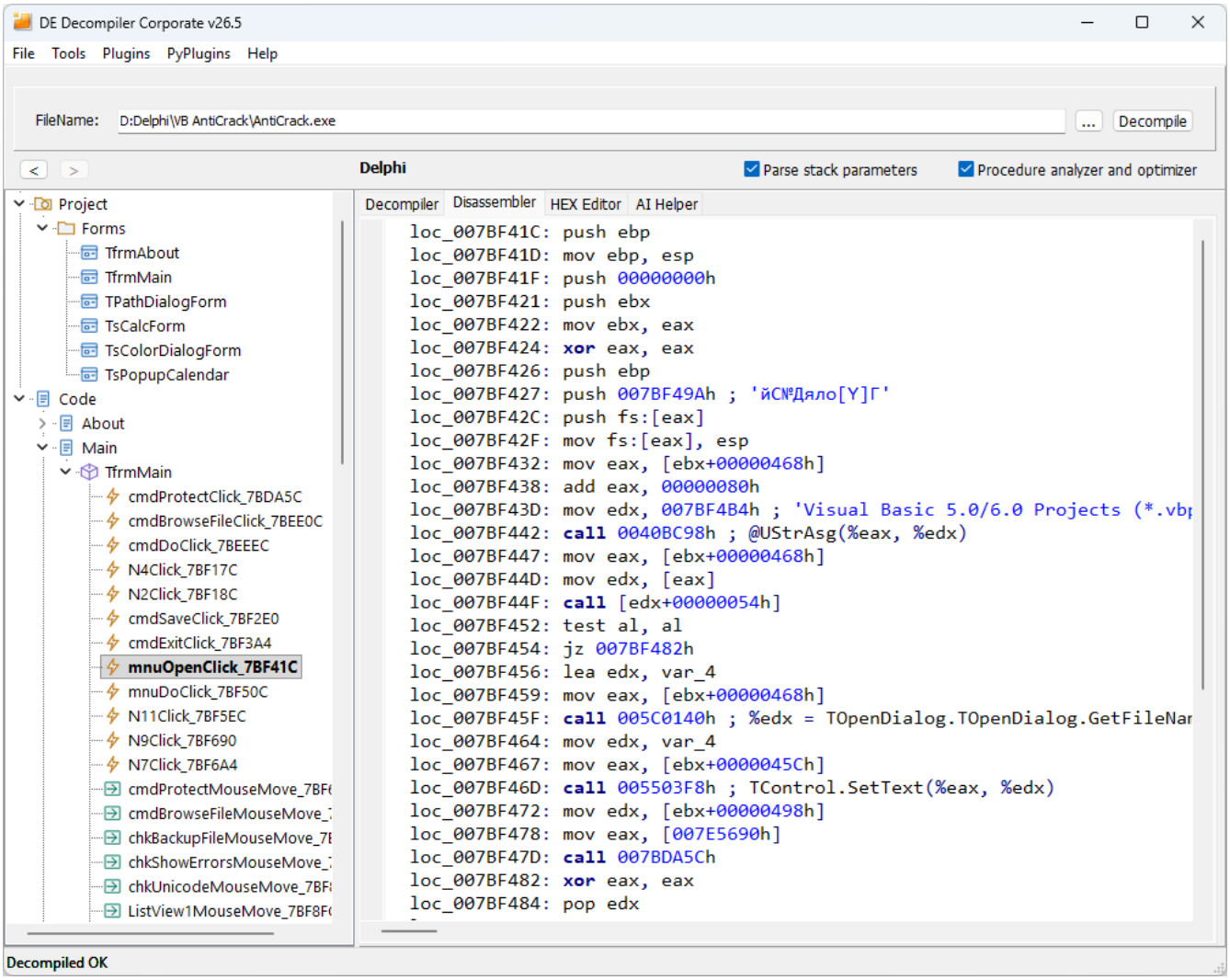
«Анализ прототипов» (Analyze prototypes) выполняет предварительный эвристический анализ аргументов процедур и возвращаемых значений. Это полезно в режиме быстрой декомпиляции, когда для частных функций отсутствует явная метаданных сигнатуры. Этот анализ требует дополнительного начального времени обработки. Проверяйте выведенные сигнатуры по вызывающим объектам и нативным инструкциям, если они влияют на вывод анализа.

«Добавлять адреса» (Add Addresses) сохраняет виртуальные адреса в высокоуровневом листинге для сравнения и навигации. Оставьте его включенным для **«Скопировать в дизассемблер» (Copy to disassembler)**. Дизассемблер отображает адреса независимо от данной настройки.

Если вывод кэширован или весь проект уже был обработан, после изменения соответствующих настроек заново откройте или переработайте входные данные.

4.3 Отображение дизассемблированного кода

Дизассемблер показывает машинные инструкции для проверки кода, восстановленного из поддерживаемых программ, написанных на Delphi или созданных в C++Builder.



Нативные инструкции для восстановленной процедуры Delphi.

Параметры отображения

«Добавлять HEX данные» (Add HEX dump) отображает байты машинного кода перед каждой инструкцией. **«Не преобраз. символы» (Ignore chars convertions)** сохраняет неанглоязычные строки читаемыми, вместо того чтобы преобразовывать их символы в числовые значения Unicode. Отключите эту опцию, если вам нужно получить числовое представление. Используйте адреса кода для перемещения между листингом и HEX-редактором.

Пользовательские классы и классы VCL

При **«Утилиты > Опции > Дизассемблер» (Tools > Options > Disassembler)**, **«Показывать польз. классы» (Show user classes)** включает обнаруженные классы приложения в обработку классов, фильтруя распознанные библиотеки и унаследованные методы. Классы приложения с именами, совпадающими с именами в библиотеках, сохраняются, если их принадлежность к модулю позволяет их различать. **«Показывать VCL классы» (Show VCL classes)** также включает внутренние классы библиотеки VCL и полный вид классов, включая сгенерированные компилятором фреймы. Последние могут добавить множество библиотечных классов в анализ; оставьте эту опцию отключенной, если вы хотите сосредоточиться на собственной реализации приложения.

Включите обработку классов VCL, если расследование требует анализа реализации библиотеки. Установите необходимые параметры классов перед открытием или повторной обработкой файла. Оба элемента управления отображаются в Параметры программы.

Сравнение интерпретаций

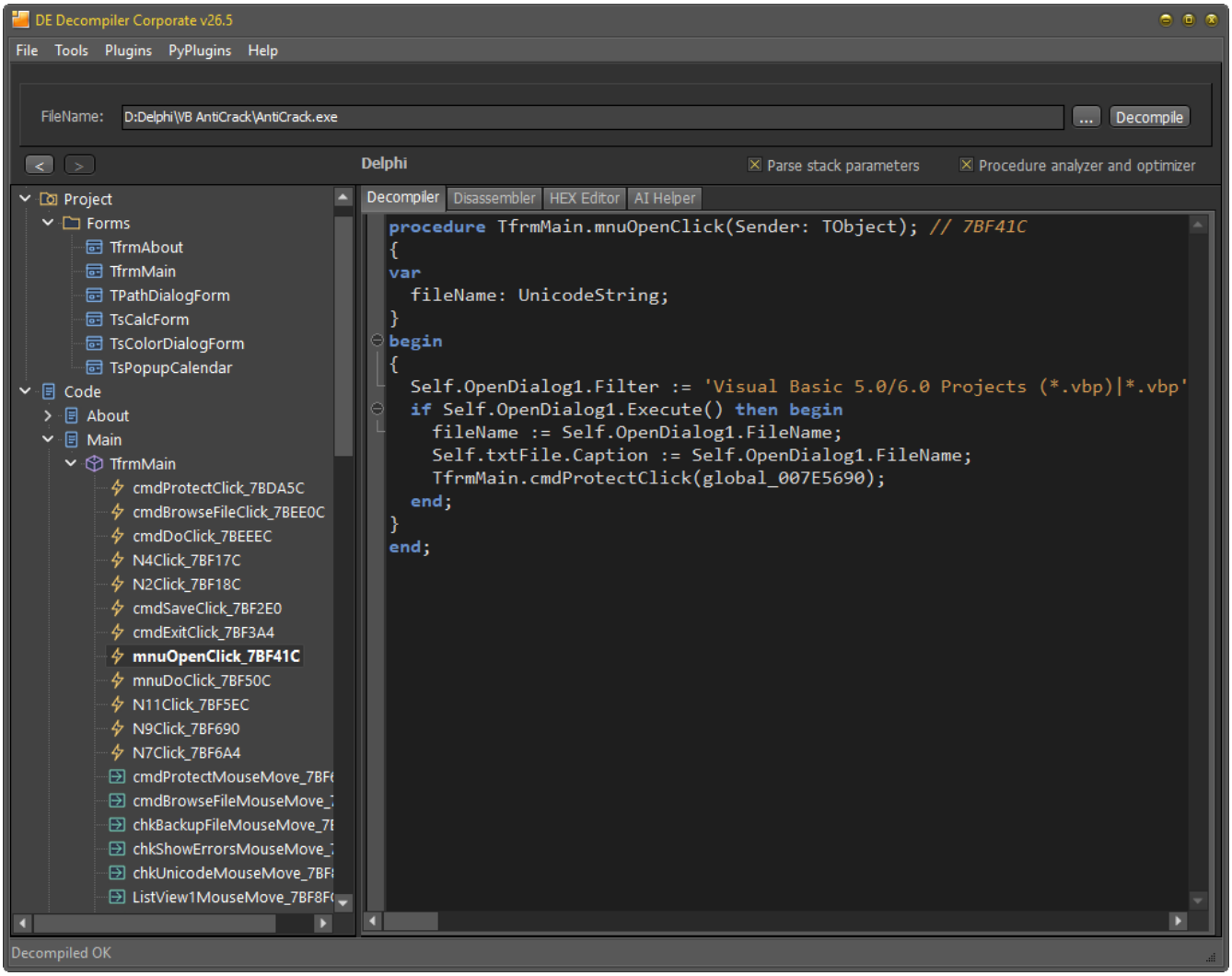
Проверяйте подготовку аргументов, доступ к полям, условные переходы, вызовы и возвращаемые значения в соответствии с нативными инструкциями. Читаемое высокоуровневое утверждение может опускать деталь, имеющую значение для расследования безопасности.

«Скопировать в дизассемблер» (Copy to disassembler) рабочий процесс объединяет восстановленные утверждения с нативными инструкциями. См. Просмотр и сравнение кода.

4.4 Темы оформления и шрифты

Список цветовых схем изменяет отображение кода и бинарных представлений, включая доступную темную тему.

Выберите схему в разделе **«Утилиты > Опции > Персональные настройки» (Tools > Options > Personalize)**. Выберите удобочитаемое сочетание цветов переднего плана, фона и синтаксиса для длительных сеансов просмотра.



Тот же рабочий процесс Delphi с кодом в темной теме интерфейса.

Файлы пользовательских тем

Предустановки находятся в `.ini` файлах в `colors` папка в каталоге установки. Имя файла определяет имя предустановки. Скопируйте существующую предустановку, чтобы создать свою собственную, и сохраните ключи, которые вы не намерены изменять. Выбранную тему можно применить без перезапуска приложения, в том числе после редактирования её файла INI.

`[Colors]` секция использует десятичные значения цветов. Её ключи включают `Background`, `Foreground`, `KeyWords`, `LocationsAndArguments`, `Procedures`, `UnderCursorBackground` и `UnderCursorForeground`. `UseDarkMode` выбирает светлый (`0`) или тёмный (`1`) интерфейс. `Colors2=1` включает расширенную секцию цветов.

Выдержка с использованием расширенных настроек просмотра кода:

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

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

Настройки шрифта и синтаксиса

Базовая `[Font]` секция предоставляет `Name`, `Size`, `KeyWordsBold`, `LocationsAndArgumentsBold` и `ProceduresBold`. В `[Colors2]`имена шрифтов могут содержать разделённые запятыми резервные варианты. Шрифты дерева используют `TVFontName` и `TVFontSize`; представления кода используют `FontName` и `FontSize`; HEX-редактор использует `HexFontName` и `HexFontSize`.

Расширенные цвета используют шестнадцатеричные значения. Ключи синтаксиса включают `Key`, `Comment`, `Char`, `Float`, `Hex`, `Number`, `String` и `Symbol`. Ключи HEX-редактора включают `HexBackground`, `HexForeground`, `HexGrid`, `HexActiveFieldBackground`, `HexOffsetBackground`, `HexCurrentOffset`, `HexOddColumn`, `HexEvenColumn` и `HexOffset`. Цвета под курсором управляют подсветкой совпадающих идентификаторов в коде.

4.5 Режимы обработки и кэширование

Быстрая декомпиляция, полная обработка и поддержка закэшированного вывода обеспечивают различные рабочие процессы анализа.

Декомпилировать только формы

Включить **«Декомпилировать только формы» (Decompile only forms)** при необходимости восстановления информации о форме VCL без обработки тел процедур.

Быстрая декомпиляция

При **«Быстрая декомпиляция» (Fast decompilation)**, начальная обработка устанавливает структуру проекта, а тела процедур обрабатываются по мере их открытия. Это полезно при исследовании выбранных модулей или методов. При отключенном кэшировании повторно откройте процедуру после изменения параметров обработки, чтобы получить новый листинг.

Параметр **«Анализ прототипов» (Analyze prototypes)** добавляет предварительный эвристический анализ сигнатур. Предварительное дизассемблирование также может собирать ссылки на строки. Если для ссылок на строки требуется полный проход, диалог предложит отключить быструю декомпиляцию и повторно обработать входной файл.

Полная обработка

Отключить **«Быстрая декомпиляция»** и обработать файл заново для предварительного генерирования листингов процедур. Это занимает больше времени, но делает сгенерированный текст доступным для поиска по всему проекту и анализа ссылок. Экспорт проекта и объединенных листингов также может обрабатывать процедуры по требованию. Изменение настроек декомпиляции требует повторной обработки файла.

Кэшировать декомпилированный код

«Кэшировать декомпилированный код» (Cache decompiled code) сохраняет сгенерированные листинги процедур. Отключите кэширование при интерактивном сравнении вариантов реконструкции. Кэшированные результаты не демонстрируют автоматически эффект от изменения настроек.

Объявлять переменные

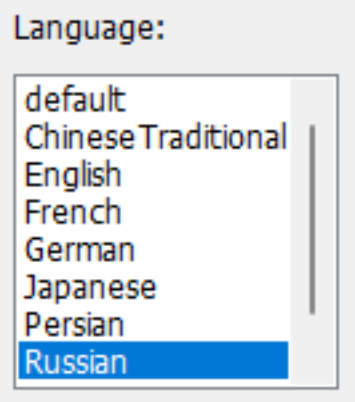
«Объявлять переменные» (Declare variables) добавляет в листинг восстановленные объявления переменных. Проверяйте предполагаемые типы с учётом доступных сведений RTTI, вызывающих процедур и машинных инструкций.

Расположение окон

Параметр **«Стиль окна как в Visual Studio» (Window style as in Visual Studio)** размещает код слева, а дерево проекта — справа. Он влияет на расположение окон, сохраняя результаты анализа.

4.6 Язык интерфейса

Выберите язык интерфейса в разделе Персонализация > Инструменты > Параметры.



Переключатель языка интерфейса.

Выберите доступную локализацию и перезапустите DE Decompiler, чтобы выбранный язык применялся последовательно во всех окнах. В данном руководстве используются подписи интерфейса на английском языке.

Локальная и онлайн-документация

Англоязычное руководство доступно онлайн и в формате PDF. Сохраняйте локальную копию при работе в изолированной среде анализа. Локальное руководство содержит ссылку на сайт продукта и адрес электронной почты поддержки в О разработчике.

Если справка на выбранном языке отсутствует, DE Decompiler открывает английский PDF или сайт. Если локальный PDF или CHM открыть не удаётся, используется соответствующая онлайн-документация.

4.7 AI Helper

AI Helper обрабатывает Delphi-подобный Pascal, уже восстановленный DE Decompiler из поддерживаемых программ, написанных на Delphi или созданных в C++Builder. Доступны локальные модели через Ollama и онлайн-модели OpenAI (ChatGPT) и DeepSeek через интеграцию OpenAI API.

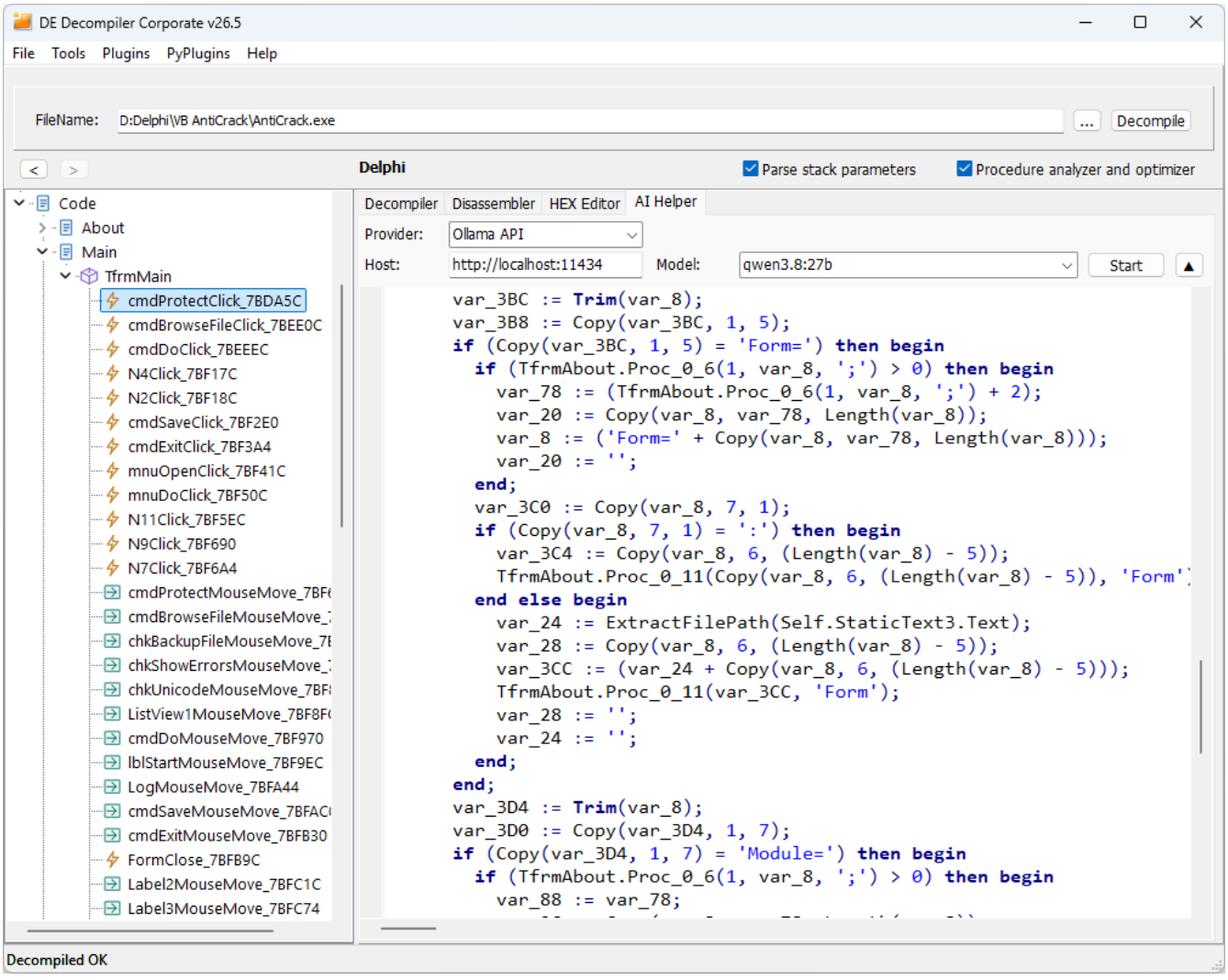
1 Откройте процедуру и просмотрите ее **«Декомпилятор» (Decompiler)** вывод.

2 Выберите **«ИИ Помощник» (AI Helper)**.

3 Выберите **Поставщик**установите адрес сервера и выберите модель.

4 Запустите обработку и сравните предложение с исходным декомпилированным листингом и нативными инструкциями.

Обработка выполняется в отдельном потоке и использует текущую процедуру целиком, включая свёрнутый код. Промпт запрашивает результат на Object Pascal и сохраняет обозначения нераспознанных вызовов и восстановленный код внутри обрамляющих комментариев экспорта. Для отмены долгого запроса используйте **«Отмена» (Cancel)**. Дополнительные настройки подключения можно свернуть, освободив место для кода.



AI Helper обрабатывает восстановленную процедуру Delphi через Ollama.

Ollama

Запустите Ollama на совместимой рабочей станции или сервере, установите там необходимую модель и подключите AI Helper к этому экземпляру. Локальный сервер Ollama обычно использует `http://localhost:11434`. Модель должна быть установлена в экземпляре, выбранном в поле **Хост** .

Выбор модели и объём памяти GPU

Лёгкие модели, такие как `qwen3:8b` обеспечивают лишь ограниченное улучшение при обработке восстановленного кода Delphi. Более крупная `qwen3.8:27b` модель показана в данном руководстве.

Для данной локальной `qwen3.8:27b` конфигурации используйте следующие рекомендации по объёму памяти GPU:

Конфигурация	Память GPU	Ожидаемое использование
Минимальные	16 ГБ видеопамяти	Обработка может быть очень медленной. Это конфигурация начального уровня, не подходящая для комфортной рабочей станции при регулярном использовании модели.
Рекомендуемые	32 ГБ видеопамяти	Предпочтительная конфигурация для регулярного локального использования указанной модели.

Эти значения относятся к выделенной видеопамяти на GPU, поддерживаемом Ollama. Оперативная память системы и показатель общей памяти, отображаемый для старого интегрированного графического адаптера, не эквивалентны выделенной видеопамяти. Компьютер эпохи Pentium M со старым интегрированным графическим чипом не подходит для этой настройки. Проверьте Ollama совместимость оборудования перед выбором GPU.

Требования относятся к компьютеру, на котором выполняется Ollama. Если AI Helper подключается к отдельному серверу Ollama, этот сервер должен предоставлять ресурсы модели.

Контекст и другие модели

Размер контекста задаётся ползунком в настройках Ollama; дополнительные расширения Ollama для этого не требуются. Контекст измеряется в токенах и включает инструкции модели, входной код и ответ. При увеличении контекста требуется больше памяти, поэтому приведённые выше объёмы не гарантируют, что любая конфигурация модели и контекста поместится в видеопамять.

Вы можете выбрать другие модели, поддерживаемые настроенным провайдером. Качество вывода и форматирование могут отличаться, а специфичный для модели служебный текст может остаться в результате. Просматривайте сгенерированный листинг перед использованием.

Онлайн-модели: OpenAI (ChatGPT) и DeepSeek

Выберите **OpenAI API** в **Поставщик** для использования онлайн-моделей OpenAI (ChatGPT) или DeepSeek. Установите **Хост**, **Ключ API** и **Модель** для выбранного сервиса. Хост, ключ API и идентификатор модели должны принадлежать одному сервису; используйте модель, доступную по этому ключу API.

DeepSeek использует тот же **OpenAI API** провайдер. Введите хост DeepSeek, ключ API и идентификатор модели в соответствующие поля. Онлайн-инференс выполняется на инфраструктуре провайдера, поэтому локальные рекомендации по объему памяти GPU Ollama не применяются к рабочей станции анализа при использовании этих сервисов.

Обработка данных

Не допускайте попадания ключей API в скриншоты и материалы совместного расследования.

AI Helper отправляет текущий декомпилированный код в выбранный сервис. Сервер на другой рабочей станции или внешний API может поместить этот код за пределы компьютера аналитика. Выбирайте конечную точку в соответствии с требованиями вашей организации к обработке проприетарного исходного кода, образцов и доказательств расследования.

Проверка предложений

ИИ может предлагать более понятные имена, объявления или структуру управления, но предложение не является доказательством поведения бинарного файла. Проверяйте измененные вызовы, условия и типы на основе дизассемблирования и связанных процедур. Сохраняйте исходный результат декомпиляции при записи интерпретации, полученной с помощью модели.

Устранение неполадок подключения и моделей

- Если модель отсутствует, убедитесь, что она установлена в экземпляре Ollama, выбранном **Хост**.
- Если ответ не поступает, проверьте доступность сервиса и то, что хост, ключ API и идентификатор модели относятся к одному провайдеру.
- Если обработка выполняется слишком медленно, отмените запрос и проверьте ресурсы хоста Ollama, размер модели и настройки контекста.
- Если незнакомая модель оставляет вспомогательные теги или служебный текст, удалите этот текст при просмотре результата или выберите другую подходящую модель.

4.8 Плагины и SDK

Используйте FRX Image Viewer для просмотра и извлечения восстановленной графики форм или разработайте собственный плагин через общий интерфейс SDK.

FRX Image Viewer

FRX Image Viewer 1.2.0 показывает и извлекает изображения, зарегистрированные VB Decompiler или DE Decompiler. VB Decompiler предоставляет графику форм VB5/6 и .NET, а DE Decompiler — изображения из бинарных DFM поддерживаемых VCL-приложений, написанных на Delphi или созданных в C++Builder. Поддерживаются BMP, PNG, JPEG, GIF, ICO и CUR.

Установка и запуск

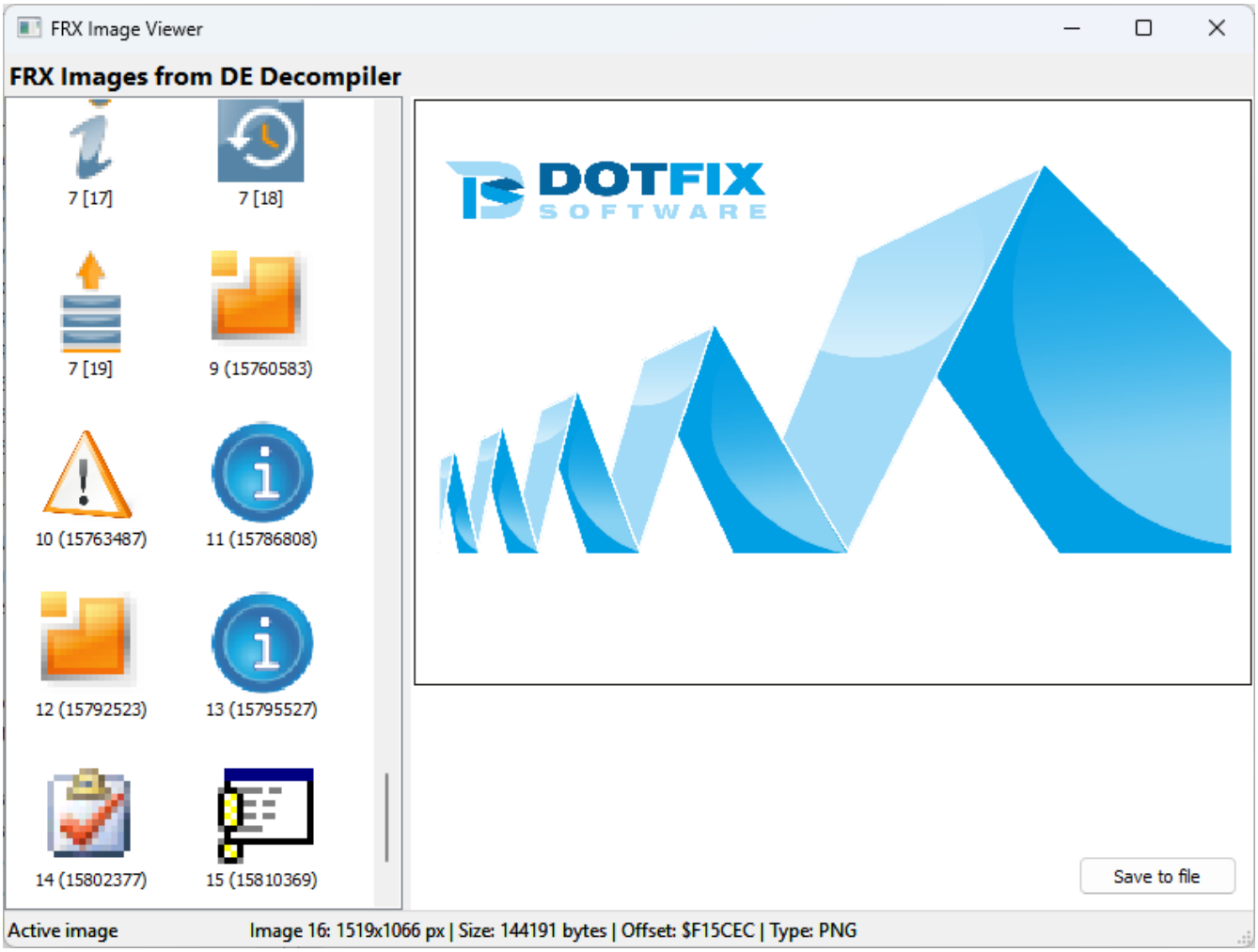
1. Копировать `FRXViewer.dll` в подкаталог установки хоста `plugins` и перезапустите декомпилятор. Архитектура DLL должна соответствовать архитектуре приложения-хоста, независимо от архитектуры анализируемого файла.
2. Откройте EXE или DLL и выполните разбор форм, чтобы декомпилятор зарегистрировал доступные изображения.
3. Выберите **«Плагины» (Plugins)** > **FRX Image Viewer**, затем выберите миниатюру для просмотра изображения.

Архив версии 1.2.0 содержит DLL для хоста x86. Для хоста x64 используйте сборку соответствующей архитектуры. Плагину требуется Windows 10/11. Инструкции по сборке доступны в репозитории исходного кода.

Просмотр и экспорт

Галерея и область предварительного просмотра показывают доступные изображения, в том числе составные ICO/CUR. В строке состояния указаны выбранный элемент, размеры, объём данных или смещение исходного растра и формат. Плагин использует тему декомпилятора и поддерживает тёмное оформление.

Выберите **Сохранить в файл** для экспорта выбранного элемента. Самостоятельные изображения сохраняют исходные байты и формат. Ячейки ImageList экспортируются как файлы ICO с масками и альфа-прозрачностью; эти файлы ICO получены из исходных битовых карт.



FRX Image Viewer в DE Decompiler, отображающая графику Delphi DFM и отдельные значки ImageList.

Нативные ImageList Delphi и совместимость

При совместимом хосте распознанные нативные списки изображений VCL отображаются как отдельные значки. Изображения с самостоятельной кодировкой внутри элементов коллекций остаются отдельными обычными изображениями. Для неподдерживаемых потоков сохраняется работа с изображением целиком, если доступны исходные записи изображений.

Ревизия общего SDK 26.6 поддерживается VB Decompiler 26.6 и DE Decompiler 26.6 и добавляет необязательные команды `GetFrxImageListInfo` (63) и `GetProductName` (64). Для старых хостов сохраняются прежний API изображений и просмотр изображения целиком. Обнаружение возможностей и правила интерпретации смещений описаны в справочнике SDK.

При просмотре и извлечении исследуемый файл не выполняется, его компоненты не загружаются. Для анимированных GIF отображается первый кадр. Доступность ресурсов зависит от успешного разбора и регистрации изображений декомпилятором.

Меню **«Плагины» (Plugins)** позволяет запускать установленные нативные DLL-плагины. Общий интерфейс обратного вызова предоставляет расширениям поддерживаемые сведения о проекте и коде. Доступные DLL-плагины инициализируются при запуске. Собирайте их для архитектуры и среды Windows, в которых работает приложение-хост.

Разработка плагина

SDK включает примеры для Visual C++, Rust, Delphi и Free Pascal. Используйте его экспортируемые точки входа и обратный вызов хоста для реализации операций, необходимых вашему рабочему процессу.

Разработку плагина можно поручить ИИ-агенту: предоставьте ссылку на справочник SDK и API и опишите требуемое поведение. Затем проверьте и соберите полученный код, протестируйте DLL в DE Decompiler на характерных входных файлах.

Совместимость интерфейса

Соблюдайте описанные в SDK соглашение о вызовах, имена экспортируемых точек входа и представления данных. Общие идентификаторы API сохраняют прежние имена, включая префикс `VBDecompilerPlugin`. Замена имени продукта в экспортируемом символе нарушит совместимость ABI.

Общий интерфейс не означает, что каждый плагин VB Decompiler подходит для данных DE Decompiler. Проверьте ожидаемые типы объектов, язык восстановленного кода и операции, доступные для текущего элемента.

Справочник разработчика

См. справочник API плагинов на сайте продукта. В нём описаны экспортируемые точки входа, идентификаторы команд, данные обратного вызова и требования к буферам. SDK предоставляет доступ к коду активного представления и всего проекта. Для операций поиска по всем декомпилированным процедурам необходимы соответствующие листинги. При наличии нужного входного файла обратные вызовы получения кода могут обрабатывать отдельные процедуры.

Открытые элементы, отметки функций и адреса

Плагины могут определять открытую форму, модуль, класс или процедуру через GetDisplayedTreeItem (65), читать и менять отметки функций через GetModuleFunctionMark (66) и SetModuleFunctionMark (67), а также искать функции по виртуальному адресу через FindModuleFunctionByVA (68). GetModuleFunctionVA (69) возвращает полный виртуальный адрес, включая 64-битный. Существующие сигнатуры callback и номера команд сохранены.

4.9 Обфускация

Встроенный обфускатор поддерживает VCL-приложения, написанные на Delphi или созданные в C++Builder, для Win32 и Win64.

Этот порядок работы также применим к поддерживаемым VCL-приложениям C++Builder для Win32 и Win64.

Обфускация предназначена для подготовки собственного поддерживаемого VCL-приложения к распространению. Она изменяет скомпилированный бинарный файл; экспорт восстановленного кода Pascal — отдельная операция.

Обфускация приложения

1. Сохраняйте оригинальное скомпилированное приложение и работайте с отдельной копией.
2. Откройте приложение через **«Файл > Открыть программу» (File > Open program)**.
3. Выберите **«Утилиты > Обфускация проекта» (Tools > Obfuscate project)** и следуйте инструкциям диалога вывода.
4. Операция создаёт резервную копию нативного входного файла перед записью и перестраивает восстановленную модель после сохранения. При сбое записи или повторного открытия во время конвертации система пытается восстановить исходный файл и модель; если восстановление завершить не удаётся, она сообщает путь к копии восстановления. Протестируйте полученное приложение с использованием тех же компонентов, данных и рабочих процессов, которые предусмотрены для его целевого развертывания.

Проверка результата

Сохраняйте возможность различать оригинальные и обфусцированные сборки в записях релизов. Проверяйте создание форм, взаимодействие компонентов и поведение, управляемое событиями, в полученном приложении. Обфускация затрудняет анализ, но не гарантирует невозможность исследования или декомпиляции программы.

5 Справочник

5.1 Интерпретация восстановленного кода

DE Decompiler восстанавливает Delphi-подобную логику программы из нативных инструкций и доступных метаданных. Детализация результата определяется информацией, сохранённой в бинарном файле.

Что представляет собой высокоуровневый листинг

Декомпиляция нативного кода восстанавливает выражения, условия `if`, блоки `case`, циклы и вызовы, включая распознанные методы VCL. Это помогает изучать логику программы при реверс-инжиниринге и анализе безопасности. Листинг представляет восстановленную реализацию, а не исходный текст программы.

Имена, структуры данных и вызовы

- **Имена переменных:** оригинальные локальные имена обычно отсутствуют в бинарном файле. Сгенерированные идентификаторы представляют восстановленные значения; присвоенные аналитиком имена могут прояснить их роль.
- **Типы и структуры:** RTTI предоставляет информацию о типах там, где она доступна. Другие структуры могут быть выведены из размеров доступа и адресов, поэтому типы полей, границы и сигнатуры могут требовать дополнительной интерпретации.
- **Вызовы:** распознанные методы VCL могут отображаться по имени. Вызов, который не удалось идентифицировать, может остаться вызовом метода или адреса без его оригинального VCL имени.
- **Вариации компилятора:** поколения компиляторов, библиотеки и оптимизация влияют на генерируемые машинные инструкции и метаданные. Специфичная для версии информация RTTI помогает в их интерпретации, однако выведенный тип или выражение все еще может требовать коррекции.

Проверяйте существенные выводы

1. Определите точный бинарный файл, модуль, класс и процедуру, которые рассматриваются.
2. Изучите декомпилированную логику и объявления, используя нативные инструкции для разрешения неопределенных выражений или типов.
3. Исследуйте вызывающие объекты, подготовку аргументов, условные переходы и связанные строки.
4. Используйте HEX-редактор, когда инструкцию или восстановленное значение необходимо проверить по данным файла.
5. Фиксируйте то, что было замечено непосредственно, и то, что остается выведенной интерпретацией.

Расследование безопасности

Восстановленная структура и код помогают следователям по инцидентам, аналитикам антивирусов и корпоративным командам безопасности находить подозрительную логику, потенциальные бэкдоры и нежелательный функционал. Установите, как используется значение или вызов, прежде чем делать выводы. Сама по себе строка или имя API не доказывают вредоносное поведение или факт выполнения ветви.

Восстановление исходного кода

Восстановленные формы, объявления и код процедур могут сократить объём ручной работы по восстановлению. Декомпиляция нативного кода не воспроизводит компилируемый исходный проект. Перед открытием экспортированного кода в IDE выполните ручной анализ и исправьте неточности декомпиляции и несовместимости, согласуйте предполагаемые типы и неустановленные вызовы, а также обеспечьте наличие недостающих модулей, компонентов и зависимостей. Сборка восстановленного материала требует выполнения ручных работ.

Приложения, созданные в C++Builder

Баз знаний RTTI для библиотек, специфичных для C++Builder, пока нет. Функции вне общей базы VCL, а также нераспознанные свойства и методы объектов могут отображаться без библиотечных имён. Поэтому качество декомпиляции кода C++Builder заметно ниже, чем для программ, написанных на Delphi. Это ограничение касается качества восстановления кода и распознавания библиотечных имён. Остальные возможности доступны для C++Builder по тем же правилам редакций, что и для программ, написанных на Delphi.

Код C++Builder восстанавливается в виде Delphi-подобного Pascal с экспортом в PAS/DFM. Исходный текст на C++ не восстанавливается. Как и при другом восстановлении нативного кода, перед использованием результата в IDE вручную проверьте и исправьте предполагаемые типы, сгенерированные имена и погрешности декомпиляции.

5.2 Техническая поддержка

Сфокусированный отчёт помогает воспроизвести проблему распознавания, формы или декомпиляции.

Укажите DE Decompiler сборку, операционную систему, имя входного файла и точный идентификатор сборки, версию компилятора (если известна), а также соответствующий модуль, класс, форму или адрес процедуры. Укажите параметры обработки и шаги, необходимые для воспроизведения проблемы.

Прикрепляйте минимальный пример или исходный бинарный файл только при наличии разрешения на их распространение. Для конфиденциальных расследований сначала опишите проблему и согласуйте подходящий способ предоставления материалов. Никогда не включайте Ключ APIи или несвязанную информацию о клиентах.

См. страница поддержки для получения текущей контактной информации.

Поддержка по электронной почте: support@de-decompiler.com

5.3 Лицензирование

DE Decompiler предлагается как Free Lite и с коммерческими лицензиями Single Developer и Enterprise.

Редакции

Редакция	Область применения
Free Lite	VCL информация о форме, нативное Win32 и Win64 дизассемблирование, подсветка синтаксиса, поиск процедур и ссылки на строки. Восстановление кода на высокоуровневом языке недоступно.
Лицензия Single Developer	Коммерческие функции продукта для одного конкретного разработчика на одном физическом компьютере или локально размещенном виртуальном ПК.
Лицензия Enterprise	Коммерческие функции продукта для до 20 пользователей на до 20 физических компьютеров или локально размещенных виртуальных ПК в рамках лицензированной компании.

AI Helper, HEX-редактор и обфускация требуют коммерческой лицензии.

Обновления, поддержка и продление

Новые коммерческие лицензии включают один год обновлений и технической поддержки. Продление на один год доступно как для Single Developer, так и для Enterprise.

После окончания периода обновлений и поддержки установленная версия продолжает работать на компьютере, где она была активирована. Доступ к новым версиям, техническая поддержка и перенос активации на другой компьютер требуют наличия активного периода обновлений и поддержки.

Привязка и активация

Лицензия Single Developer является персональной для указанного разработчика. Она не может передаваться между разработчиками, ротироваться среди членов команды, переноситься или переоформляться на другого разработчика.

Лицензии не могут быть переданы или переоформлены на другое лицо или компанию. Лицензии не являются плавающими или конкурентными. Активации Enterprise распределяются в пределах лимитов пользователей и компьютеров лицензированной компании.

См. страница лицензирования для получения текущих цен и ссылок на покупку, а также страница загрузки для Free Lite.

5.4 О разработчике

Декомпилятор разрабатывается Чубченко Сергеем Георгиевичем (**DotFix Software**).

DotFix Software разрабатывает программные инструменты с 2001 года. См. сайт DotFix Software и VB Decompiler.

Лицензии на DE Decompiler впервые поступили в открытую продажу в 2007–2008 годах. Затем продукт продолжал развиваться для внутренних задач и позднее вернулся в открытую продажу. Основное направление — анализ нативного кода VCL-программ, написанных на Delphi 3–13, для Win32 и Win64.

Также поддерживаются VCL-приложения, созданные в C++Builder. В том же рабочем пространстве доступны формы, модули, классы, функции и остальные инструменты. Ограничения баз RTTI влияют на качество восстановленного кода, а не на доступность этих возможностей.

Сайт продукта: www.de-decompiler.com
Электронная почта поддержки: support@de-decompiler.com