Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования   
«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Отчет

По лабораторной работе №4

**«Статический и динамический анализ»**

по дисциплине «Тестирование и отладка программного обеспечения»

Студенты группы ПИ-12 Черкасов С.В.

Доцент, Ерёмин Я.Б.

Барнаул 2025

**Цель работы:** Научиться делать статический и динамический анализ проектов.

**Задание:**

Порядок выполнения лабораторной работы #4

1. Взять какой-нибудь курсовой проект, натравить на него любой статический анализатор: https://ru.wikipedia.org/wiki/Статический\_анализ\_кода#Инструменты\_статического\_анализа
2. Посмотреть на результаты, и описать самое полезное предупреждение. Описать ложноположительное. Если ничего интересного нет, добавить какой-нибудь явно некорректный код, например ошибки при невнимательном редактировании после копирования/вставки, недостижимые ни при каких условиях участки кода, и посмотреть, что он скажет.

Примеры багов из-за копипасты: <https://pvs-studio.com/ru/blog/posts/cpp/0260/>.

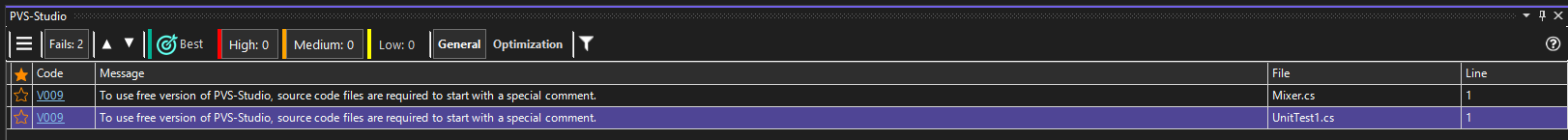
У PVS-Studio ещё и на Хабре отличный блог, вот, например, их ТОП-10 багов в ASP.NET: <https://habr.com/ru/company/pvs-studio/blog/654395/>

1. Прогнать какой-нибудь проект любым динамическим анализатором https://ru.wikipedia.org/wiki/Динамический\_анализ\_кода#Примеры\_утилит и, если утечек памяти там нет, организовать её искусственно и поймать. Если утечки памяти для вашего любимого языка программирования не очень актуальны, можно попробовать какую-нибудь другую полезную функциональность.
2. Сделать отчет о проделанной работе, залить на LMS.

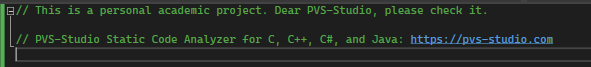
**Выполнение работы:**

Для выполнения данной работы возьму третью лабораторную работу по тестированию.

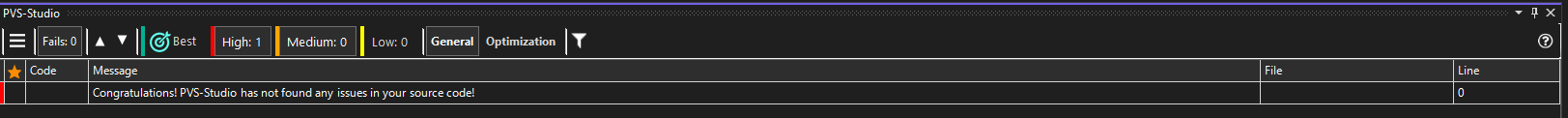
Статический анализ

Проект написан на языке C# с использованием Visual Studio, поэтому вполне логично было выбрать PVS-Studio, так как он добавляется в виде расширения. После установки и настройки запускаю проверку:  


Добавляю соответствующие комментарии во все файлы:

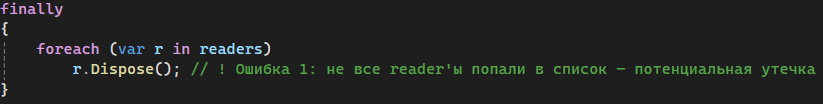


И снова запускаю:

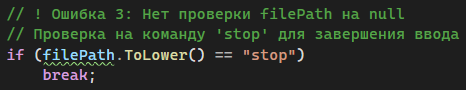


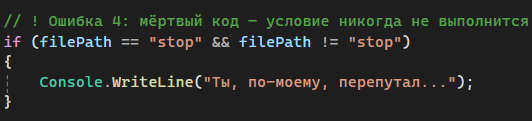
На удивление, все хорошо.

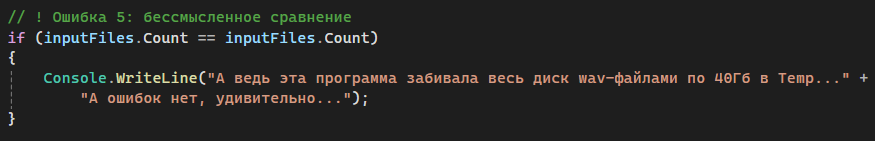
Добавим ошибок (будто нам их итак не хватает):  

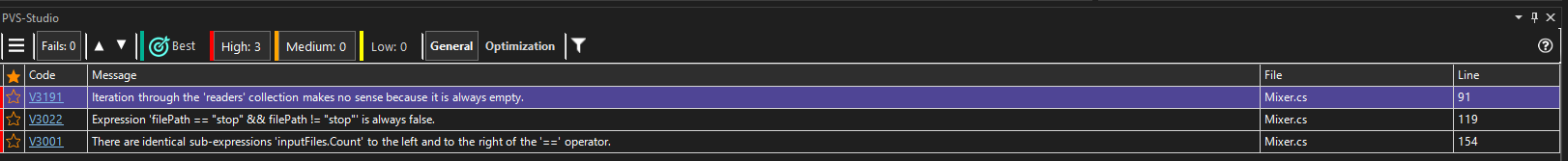








И вот результат:



Из 5 ошибок только 3 найдено.  
Ошибка 1: «Итерация через коллекцию 'readers' не имеет смысла, потому что она всегда пуста.». Да, так оно и есть, ведь я закомментировал добавление элементов в readers.

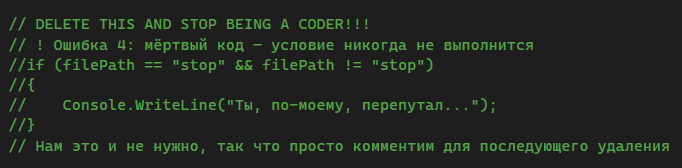
Ошибка 4: Условие всегда ложно. Комментировать не нужно.

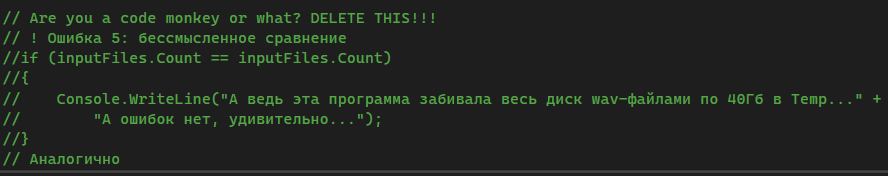
Ошибка 5: «Идентичные подвыражения 'inputFiles.Count' слева и справа от оператора '=='.». Все верно, это ошибка.

А вот ошибки 2 и 3 (копипаста и отсутствие проверки filePath на null) были пропущены. Ошибка 3 особенно важна, ибо на моей практике часто возникают проблемы именно с попыткой обратиться к пустому значению переменной. Не всегда такие проверки нужны (в случае, если мы **точно** уверены, что значение пустым не будет), но все же не лишним было бы отметить это как ошибку. А копипаста могла входить в замыслы разработчика (вот захотел он одно и то же в консоль вывести аж 2 раза!).

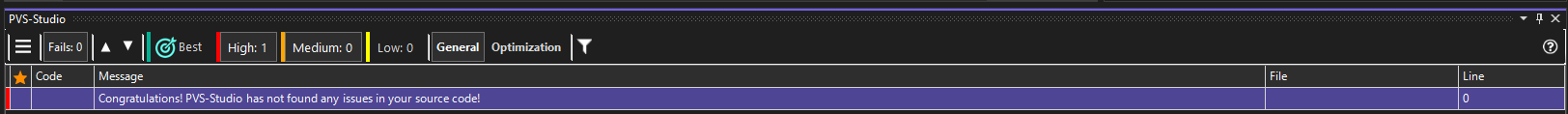
Теперь исправим все ошибки, на которые «поругался» анализатор:







Еще раз запускаем:



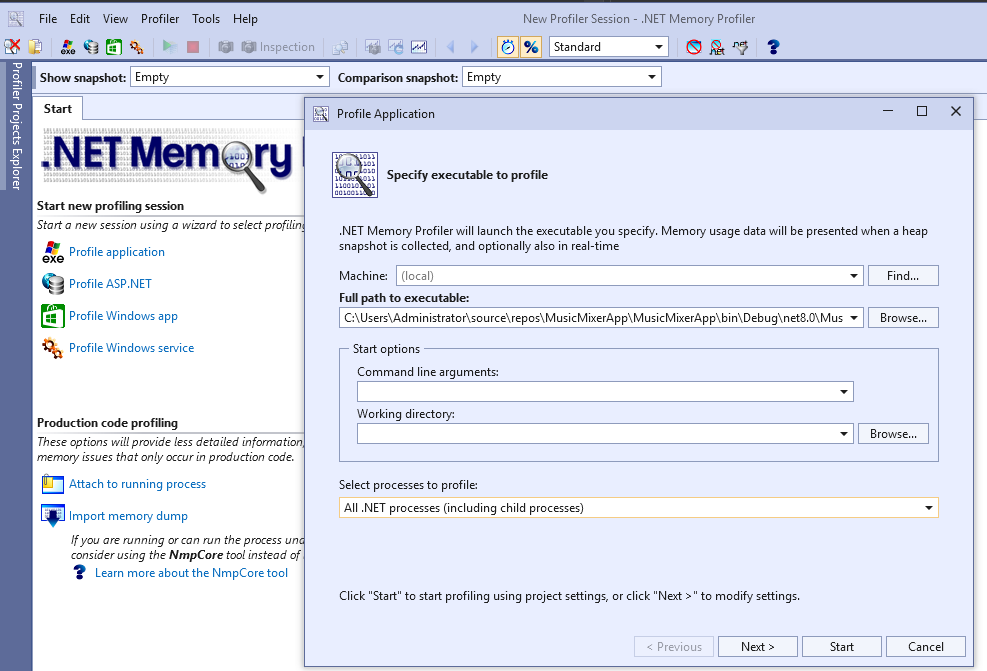
Сами накодили себе наголову – сами исправили. Все честно.

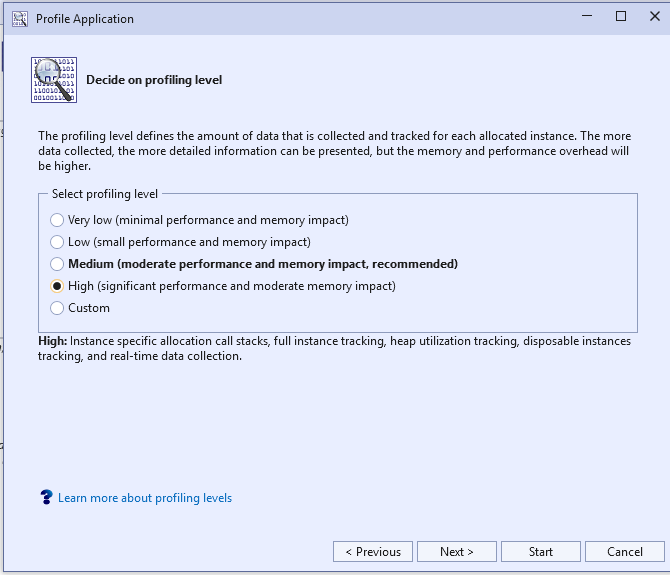
Наиболее значимыми, на мой взгляд, являются предупреждения, связанные с отсутствием доступа к файлам, включённым в проект. Согласно информации из открытых источников, при удалении файла из проекта, статический анализатор сообщает об ошибке, указывая на его недоступность.

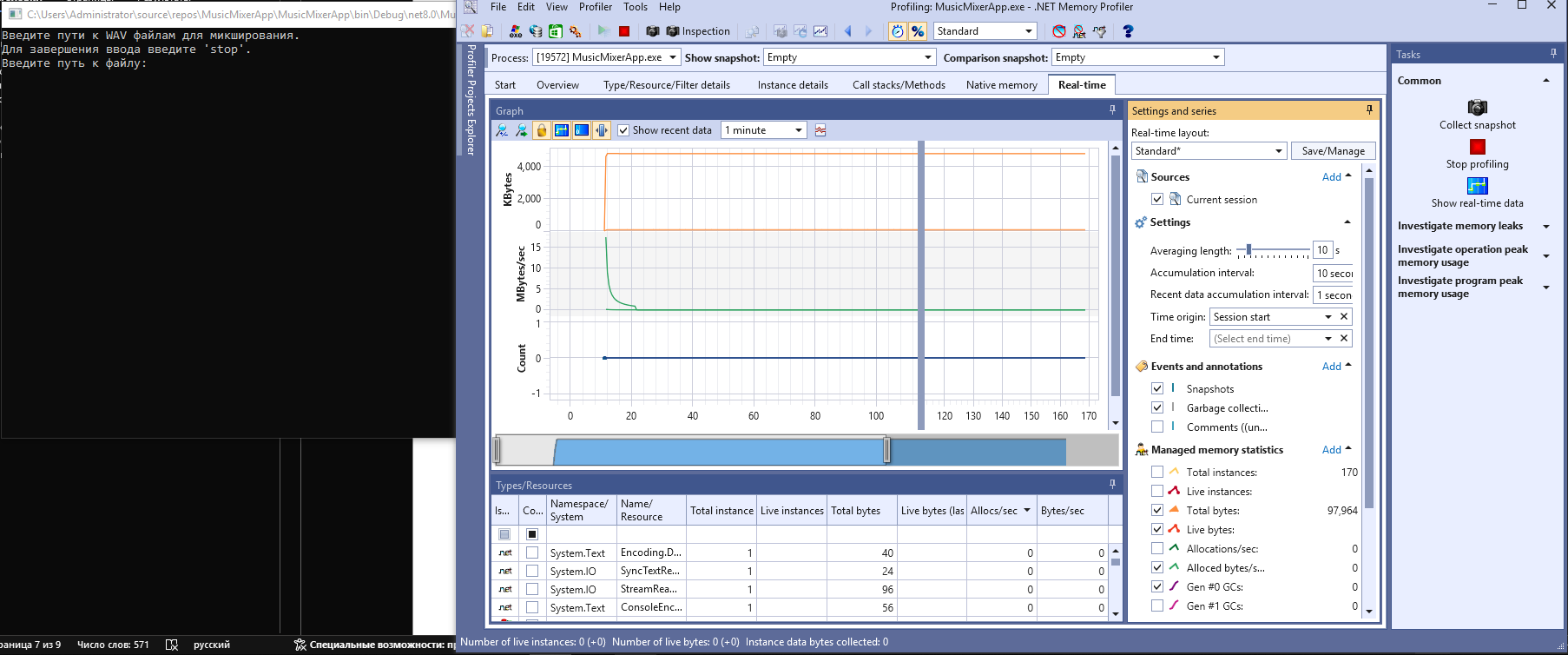
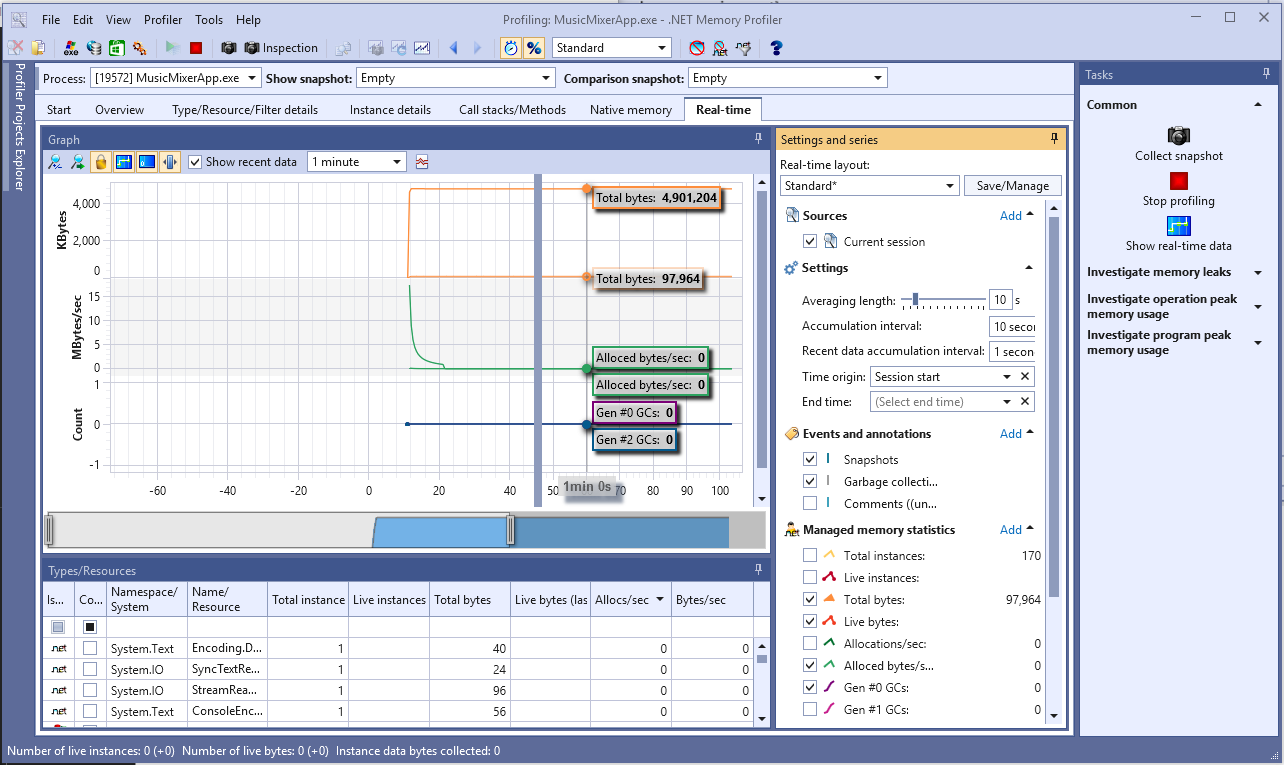
Динамический анализ

Выбрал SciTech .NET Memory Profiler (интеграция с Visual Studio, работа с .NET, детальный анализ распределения памяти.

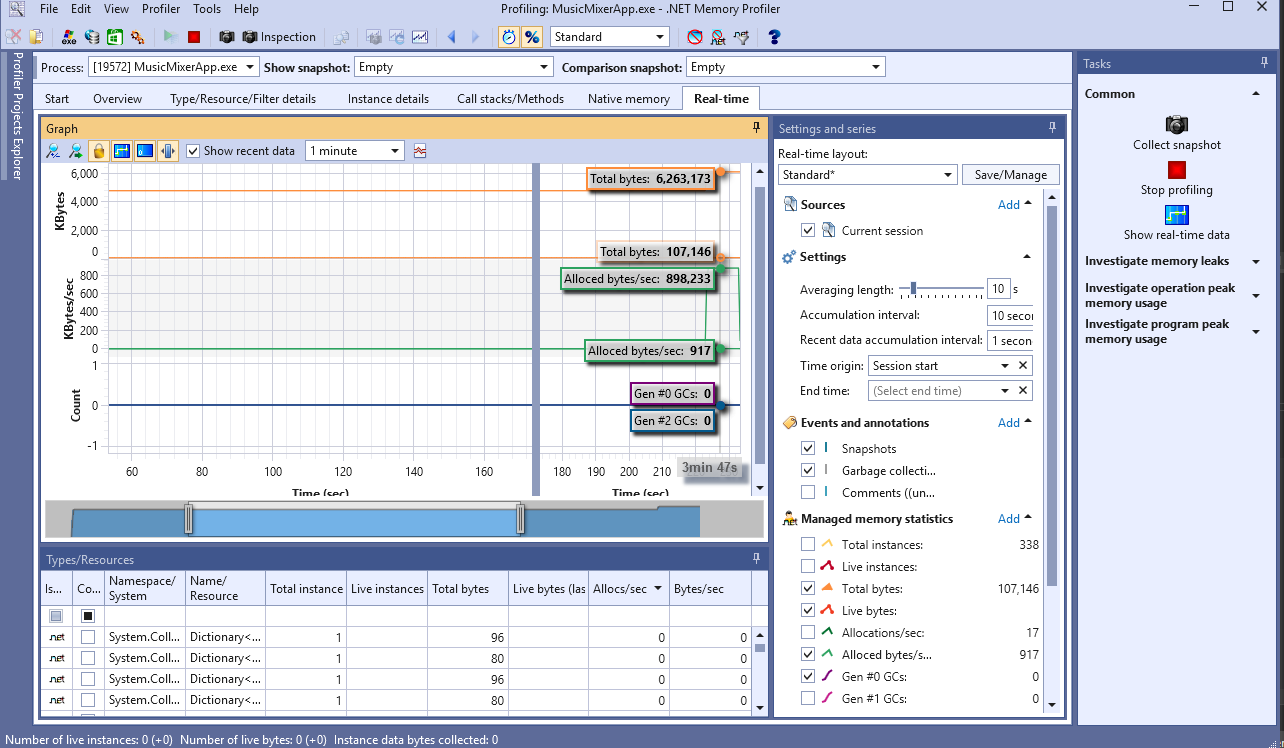
Анализирую тот же проект. Особый интерес представляет утечка памяти на диске (из-за раздувания wav-файлов, которое происходит в случае попытки прочитать данные аудио потока, даже если там уже ничего нет).

Запускаем:  


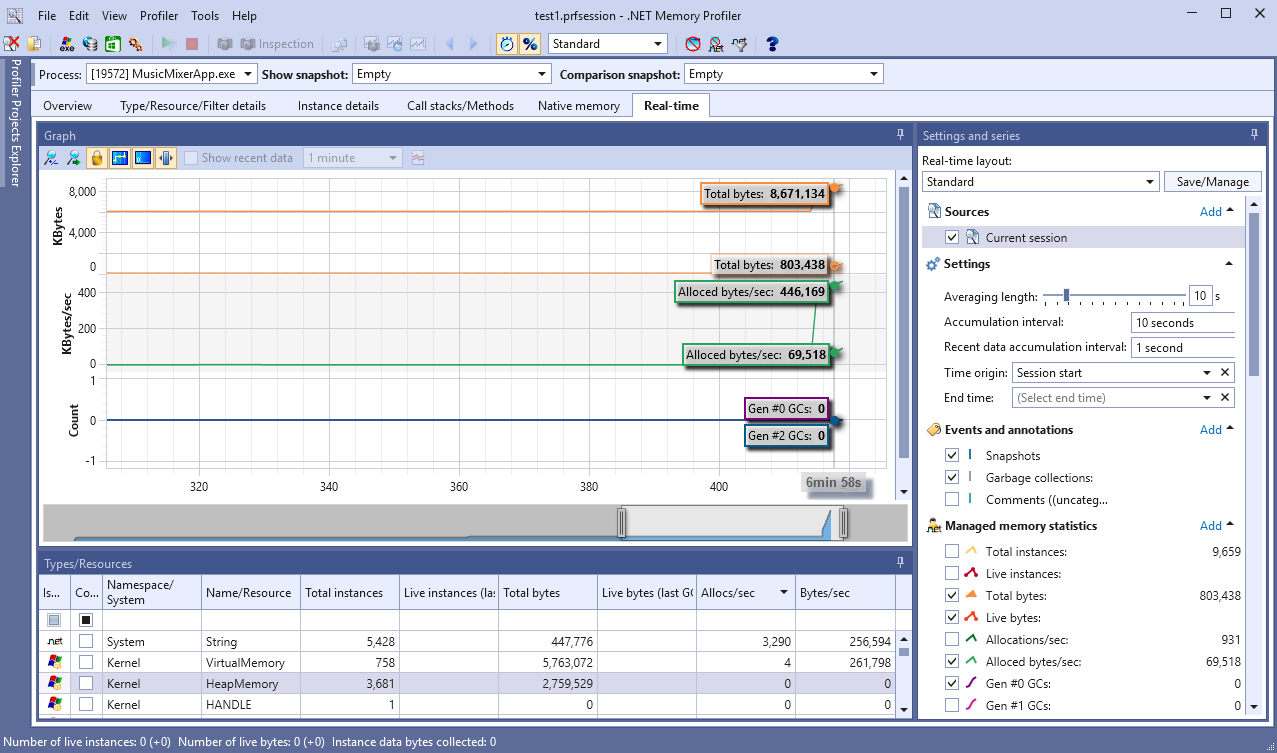
  
При запуске появился график. Зафиксируем значения:

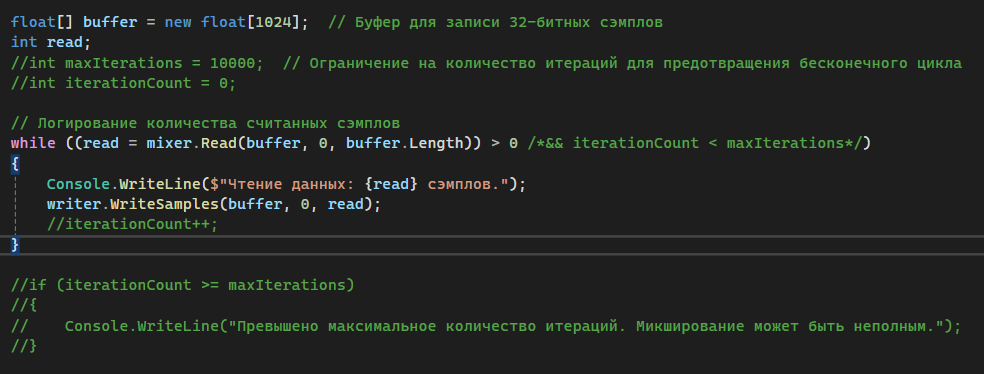
Ввод первого файла:

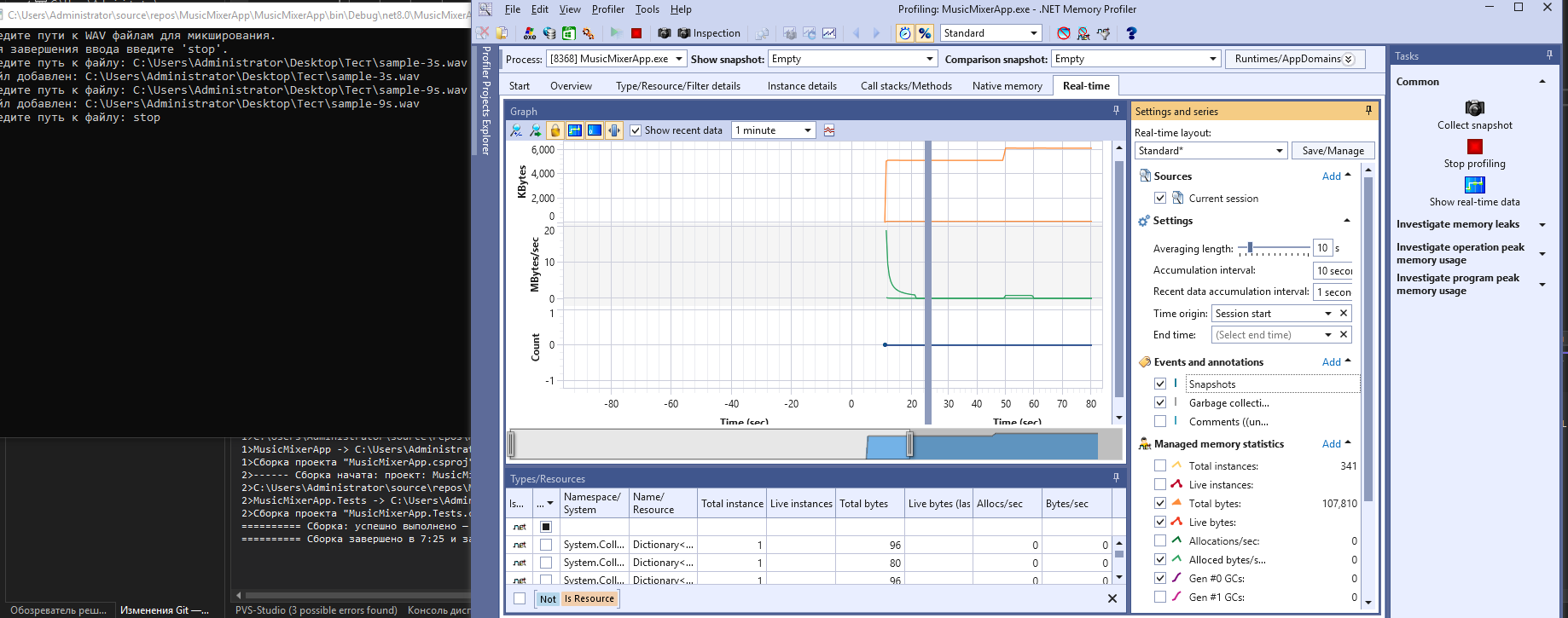


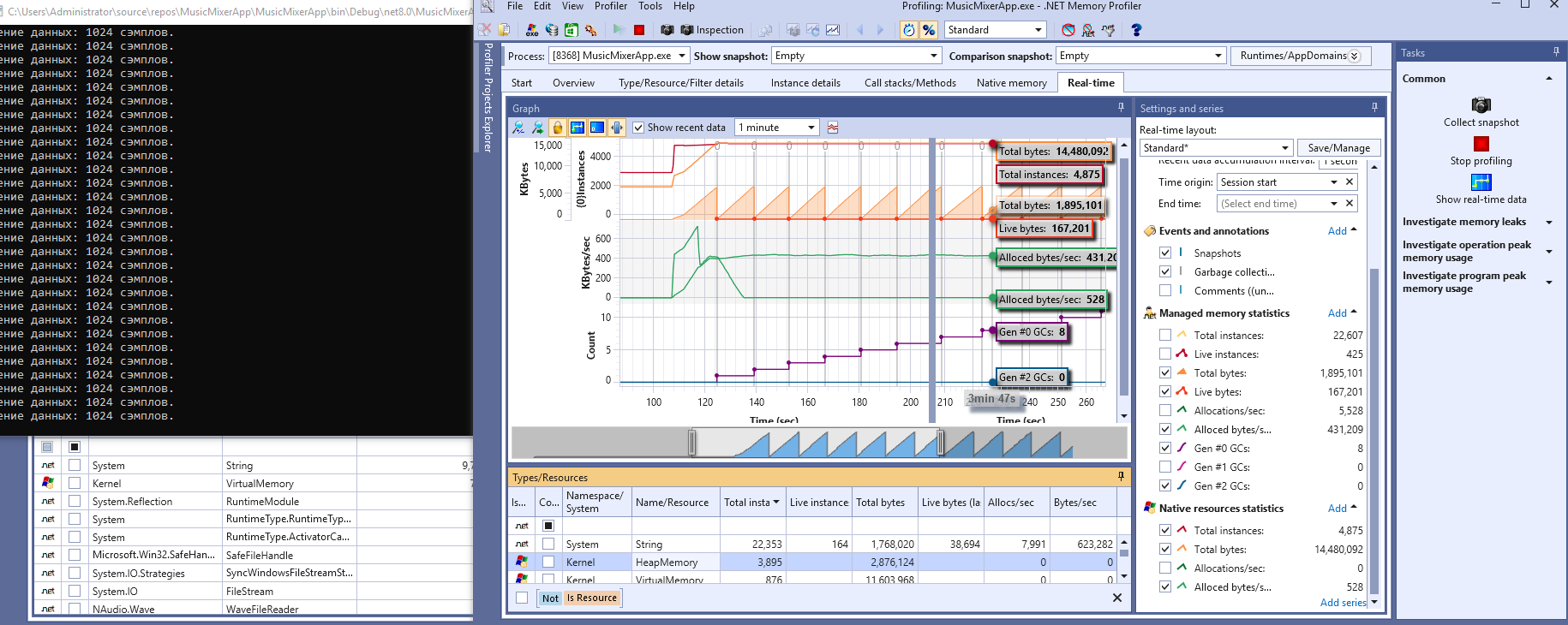
Ввод второго файла дал лишь небольшой прирост, поэтому сразу перейдем к началу миксования:



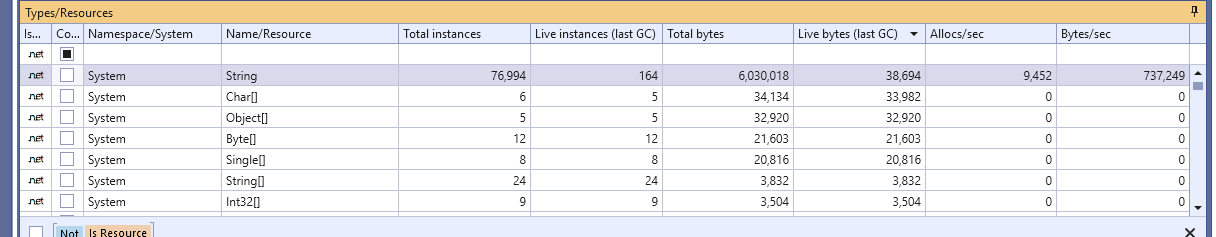
Резкий скачок по памяти. Теперь уберем заглушку на количество итераций в цикле сэмплирования:

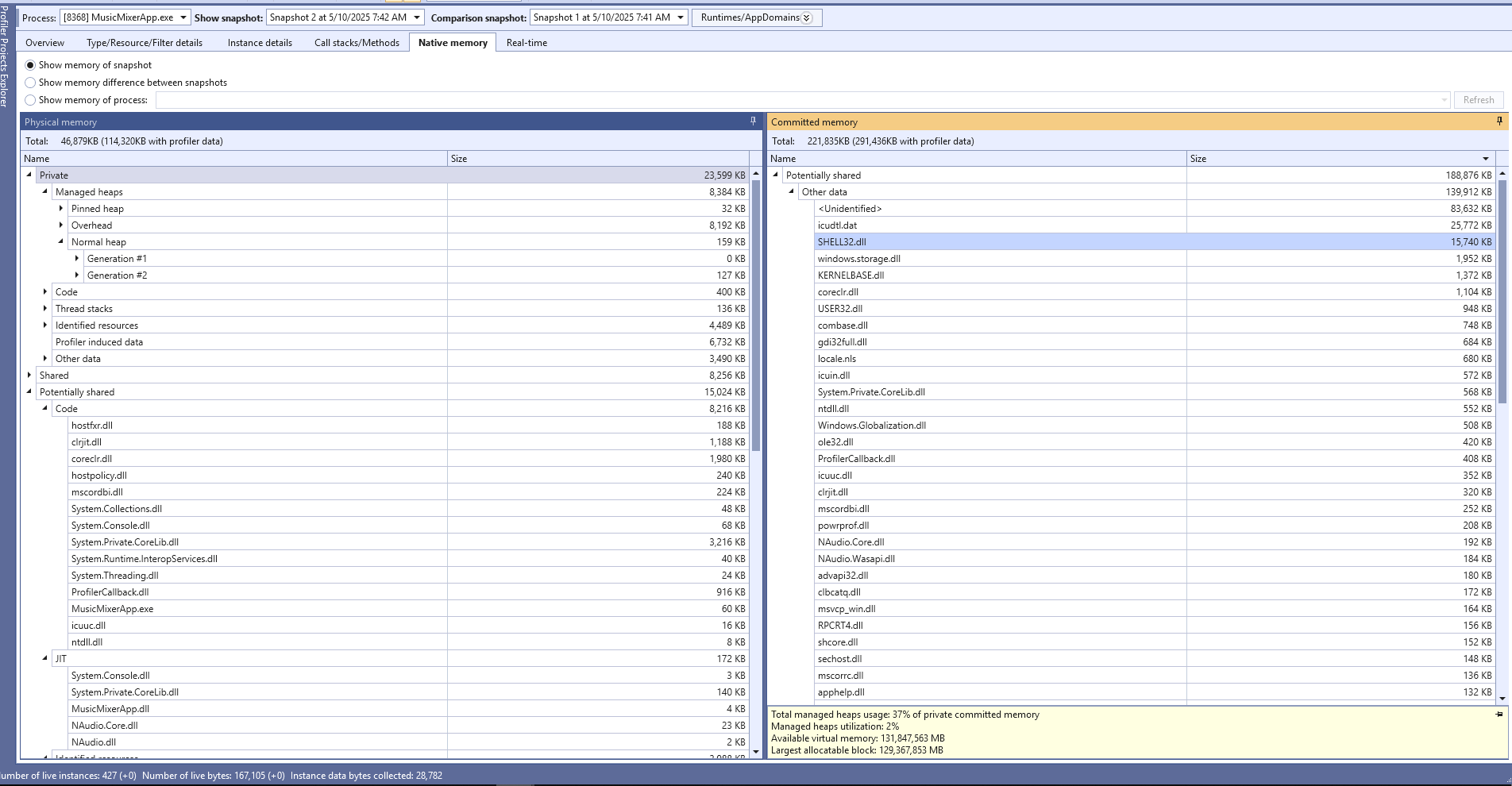


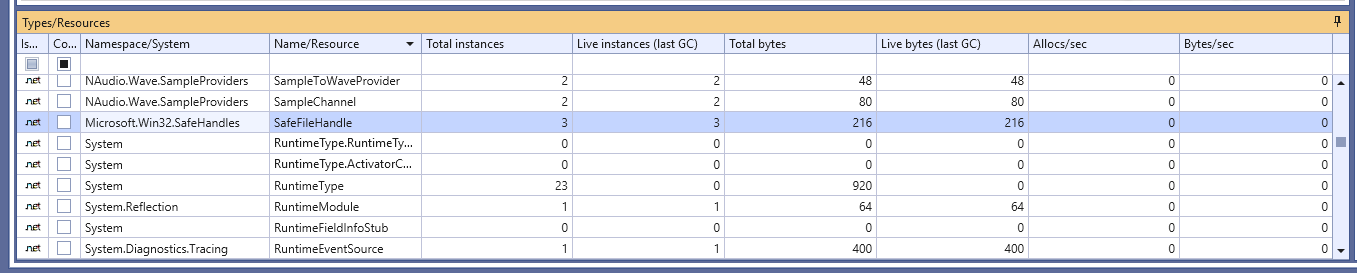




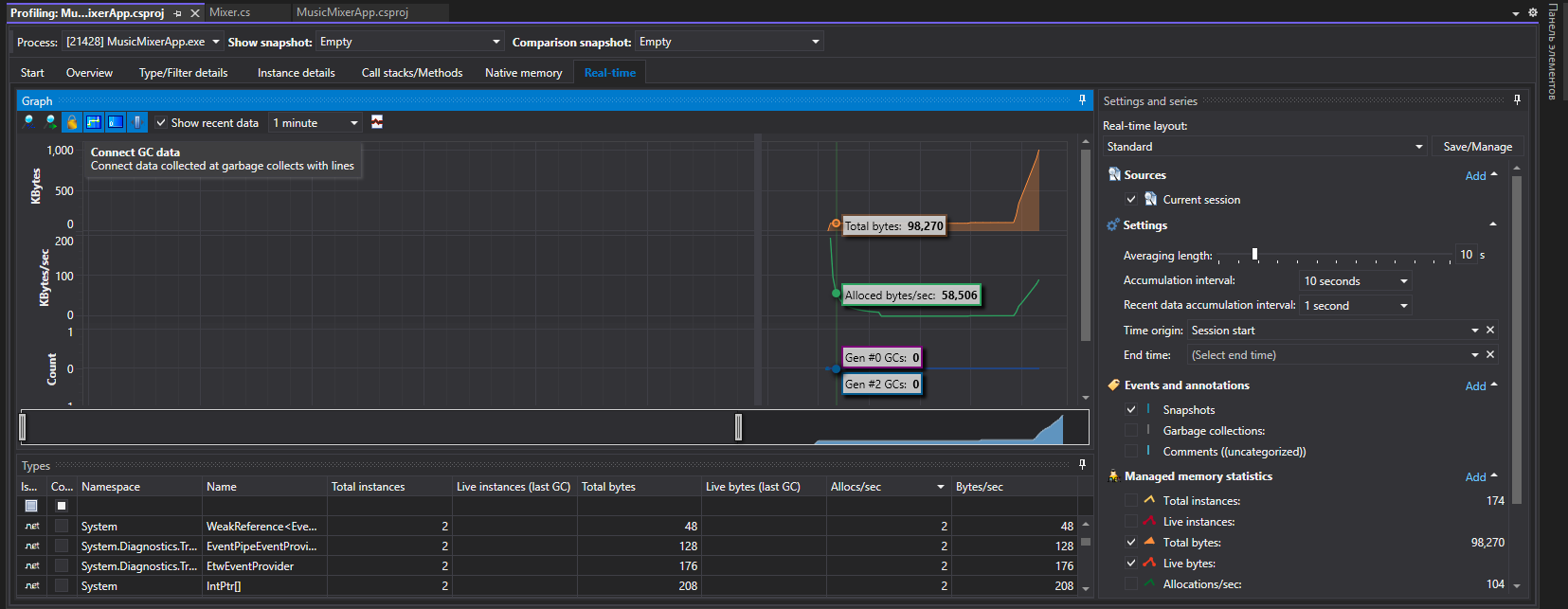
Отсортируем по Live Bytes (last GC):





Теперь видим, что утечки managed ресурсов не наблюдается. А вот unmanaged присутствует, о чем нам говорит Microsoft.Win32.SafeHandles.  
  
Их всего 3 объекта, весят не так много, но в бесконечном цикле именно они отвечают за раздувание файлов (так как они отвечают за сохранение файлов и не освобождаются). Вот так косвенными методами можно обнаружить, по какому типу ресурсов утечка и по каким возможным объектам.

Можно и через Visual Studio:



**Ссылка на работу:** <https://drive.google.com/drive/folders/15UBdaMHHQZKnwSLox4_jG6pskCxzEI3F?usp=sharing>

**Ссылка на GitHub:** <https://github.com/Dev-saint/MusicMixerAppTDD/tree/Analyzer>

**Вывод:** В процессе выполнения лабораторной работы были освоены методы синтаксического и динамического анализа программного кода, направленные на выявление потенциальных ошибок и повышение его надёжности.