csvHelper Library 15.09.25  
В начале ничего не работает. Работает толькео в конце  
F:\Projects\VS\CsvSln\CsvParserLibrary07\CsvParserLibrary07.csproj

Надо посмотреть, что в начале, там есть интересный код.  
Есть не упорядоченный фрагменты кода, которыя я писал давно и совсем забыл, что они делают. Необходимо проанализировать мой прошлый код. Взять оттуда все лучшее. Убрать худшее. Возможно что-то добавить от себя И сделать небольшую библиотеку с# lib для парсинга с правильными функциями. Выкладываю код. using CsvHelper; using CsvHelper.Configuration; using System; using System.Collections.Generic; using System.Globalization; using System.IO; // Класс Person public class Person { public int Id { get; set; } public string Name { get; set; } public int Age { get; set; } public override string ToString() { return $"Id: {Id}, Name: {Name}, Age: {Age}"; } } // Класс для обработки маппинга CSV-файла (опционально, если требуется ручная настройка столбцов) public sealed class PersonMap : ClassMap<Person> { public PersonMap() { // Соответствие столбцов CSV и свойств класса Map(m => m.Id).Name("Id"); Map(m => m.Name).Name("Name"); Map(m => m.Age).Name("Age"); } } // Основной код public class Program { /// <summary> /// Общий метод для парсинга файлов CSV в список объектов переданного типа /// </summary> /// <typeparam name="T">Тип целевых объектов</typeparam> /// <param name="filename">Имя файла CSV</param> /// <returns>Список объектов типа T</returns> public static List<T> ParseCsvToObjects<T>(string filename) where T : class { // Результирующий список var result = new List<T>(); // Чтение файла using (var reader = new StreamReader(filename)) using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture)) { // Если требуется кастомное маппирование, оно регистрируется в этом месте csv.Context.RegisterClassMap<PersonMap>(); // Читаем данные и преобразуем их в список объектов типа T result = new List<T>(csv.GetRecords<T>()); } return result; } /// <summary> /// Главный метод программы /// </summary> /// <param name="args">Аргументы командной строки</param> public static void Main(string[] args) { try { // Укажите путь к вашему CSV-файлу var filename = "example.csv"; // Парсим файл CSV и создаём объекты Person var people = ParseCsvToObjects<Person>(filename); // Вывод данных в консоль foreach (var person in people) { Console.WriteLine(person.ToString()); } } catch (FileNotFoundException) { Console.WriteLine("CSV файл не найден. Проверьте указанный путь."); } catch (Exception ex) { Console.WriteLine($"Произошла ошибка при обработке CSV файла: {ex.Message}"); } } }

Давайте создадим улучшенную библиотеку для парсинга CSV-файлов на C#. Я проанализирую ваш код и предложу более гибкое и надежное решение.

**Основные улучшения:**

* Добавление обработки различных форматов CSV
* Улучшение обработки ошибок
* Добавление поддержки разных кодировок
* Создание более гибкой системы маппинга
* Добавление возможности настройки разделителей
* Реализация асинхронных методов

Вот улучшенная версия библиотеки:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Text;

using System.Threading.Tasks;

namespace CsvParserLibrary

{

// Базовый класс для маппинга CSV

public abstract class CsvClassMap<T> : ClassMap<T> where T : class

{

protected CsvClassMap()

{

AutoMap();

}

}

// Основной класс для работы с CSV

public static class CsvParser

{

/// <summary>

/// Синхронный метод парсинга CSV файла

/// </summary>

public static List<T> ParseCsv<T>(string filePath, Encoding encoding = null,

char delimiter = ',', bool hasHeader = true) where T : class

{

return ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader);

}

/// <summary>

/// Асинхронный метод парсинга CSV файла

/// </summary>

public static async Task<List<T>> ParseCsvAsync<T>(string filePath, Encoding encoding = null,

char delimiter = ',', bool hasHeader = true) where T : class

{

return await ParseCsvInternalAsync<T>(filePath, encoding, delimiter, hasHeader);

}

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding,

char delimiter, bool hasHeader) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding,

char delimiter, bool hasHeader) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return await csv.GetRecords<T>().ToListAsync();

}

}

}

// Пример использования

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public override string ToString()

{

return $"Id: {Id}, Name: {Name}, Age: {Age}";

}

}

public class PersonMap : CsvClassMap<Person>

{

public PersonMap()

{

Map(m => m.Id).Name("Id");

Map(m => m.Name).Name("Name");

Map(m => m.Age).Name("Age");

}

}

}

**Как использовать библиотеку:**

csharp

try

{

// Синхронный способ

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';',

hasHeader: true);

// Асинхронный способ

var peopleAsync = await CsvParser.ParseCsvAsync<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';',

hasHeader: true);

foreach (var person in people)

{

"Как использовать библиотеку" - этот раздел не вместился на страницу. Выложу еще раз этот раздел в этот новый абзац.:

**Как использовать библиотеку**

**Базовый пример использования:**

csharp

try

{

// Синхронный способ

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';',

hasHeader: true);

// Асинхронный способ

var peopleAsync = await CsvParser.ParseCsvAsync<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';',

hasHeader: true);

foreach (var person in people)

{

Console.WriteLine(person);

}

}

catch (FileNotFoundException ex)

{

Console.WriteLine($"Файл не найден: {ex.Message}");

}

catch (CsvHelper.Exceptions.CsvException ex)

{

Console.WriteLine($"Ошибка парсинга CSV: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

**Дополнительные возможности:**

1. **Настройка маппинга:**

csharp

public class CustomPersonMap : CsvClassMap<Person>

{

public CustomPersonMap()

{

Map(m => m.Id).Name("UserID");

Map(m => m.Name).Name("FullName");

Map(m => m.Age).Name("Years");

}

}

// Использование кастомного маппинга

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = ';',

HasHeaderRecord = true

};

config.RegisterClassMap<CustomPersonMap>();

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';',

hasHeader: true);

1. **Работа с разными кодировками:**

csharp

// Для Windows-1251

var peopleWindows = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.GetEncoding(1251),

delimiter: ';');

// Для UTF-8

var peopleUtf8 = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';');

1. **Обработка CSV без заголовков:**

csharp

var peopleWithoutHeader = CsvParser.ParseCsv<Person>(

"example.csv",

hasHeader: false);

**Важные замечания:**

* Библиотека автоматически определяет типы данных
* Поддерживает различные разделители (по умолчанию - запятая)
* Имеет встроенную обработку ошибок
* Позволяет настраивать кодировку чтения файла
* Поддерживает как синхронные, так и асинхронные операции

**Рекомендации по использованию:**

* Всегда указывайте явную кодировку файла
* Проверяйте существование файла перед парсингом
* Используйте try-catch для обработки возможных ошибок
* При работе с большими файлами рекомендуется использовать асинхронные методы

ЕСть ошибки 1. AutoMap() - не находит protected CsvClassMap() { AutoMap(); } Severity Code Description Project File Line Suppression State Details Error (active) CS1501 No overload for method 'AutoMap' takes 0 arguments CsvParseLib02 F:\Projects\VS\CsvSln\CsvParseLib02\Class1.cs 14

Давайте исправим ошибку с методом AutoMap(). В CsvHelper метод AutoMap требует указания типа конфигурации. Также я внесу дополнительные улучшения в код:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Text;

using System.Threading.Tasks;

namespace CsvParserLibrary

{

// Базовый класс для маппинга CSV

public abstract class CsvClassMap<T> : ClassMap<T> where T : class

{

protected CsvClassMap()

{

// Указываем конфигурацию для AutoMap

AutoMap(new CsvConfiguration(CultureInfo.InvariantCulture));

}

}

// Основной класс для работы с CSV

public static class CsvParser

{

/// <summary>

/// Синхронный метод парсинга CSV файла

/// </summary>

public static List<T> ParseCsv<T>(string filePath, Encoding encoding = null,

char delimiter = ',', bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

/// <summary>

/// Асинхронный метод парсинга CSV файла

/// </summary>

public static async Task<List<T>> ParseCsvAsync<T>(string filePath, Encoding encoding = null,

char delimiter = ',', bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return await ParseCsvInternalAsync<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding,

char delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

// Возможность дополнительной настройки конфигурации

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding,

char delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return await csv.GetRecords<T>().ToListAsync();

}

}

}

// Пример использования

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public override string ToString()

{

return $"Id: {Id}, Name: {Name}, Age: {Age}";

}

}

public class PersonMap : CsvClassMap<Person>

{

public PersonMap()

{

Map(m => m.Id).Name("Id");

Map(m => m.Name).Name("Name");

Map(m => m.Age).Name("Age");

}

}

}

Теперь метод AutoMap работает корректно, так как мы указали конфигурацию. Также я добавил дополнительную возможность настройки конфигурации через делегат Action<CsvConfiguration>, что делает библиотеку еще более гибкой.

Пример использования с новой возможностью настройки:

csharp

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ';',

hasHeader: true,

config =>

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding, char delimiter, bool hasHeader) where T : class { var config = new CsvConfiguration(CultureInfo.InvariantCulture) { 44 Delimiter = delimiter, HasHeaderRecord = hasHeader }; using (var stream = new FileStream(filePath, FileMode.Open)) using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8)) using (var csv = new CsvReader(reader, config)) { return csv.GetRecords<T>().ToList(); } } Severity Code Description Project File Line Suppression State Details Error (active) CS0029 Cannot implicitly convert type 'char' to 'string' CsvParseLib02 F:\Projects\VS\CsvSln\CsvParseLib02\Class1.cs 44

Проблема в том, что свойство **Delimiter** в CsvConfiguration ожидает строку (string), а не символ (char). Давайте исправим это:

csharp

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding,

char delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter.ToString(), // Преобразуем char в string

HasHeaderRecord = hasHeader

};

// Возможность дополнительной настройки конфигурации

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding,

char delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter.ToString(), // Преобразуем char в string

HasHeaderRecord = hasHeader

};

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return await csv.GetRecords<T>().ToListAsync();

}

}

Также можно оставить параметр delimiter как string:

csharp

// Изменяем сигнатуру методов

public static List<T> ParseCsv<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

// И соответственно в асинхронной версии

public static async Task<List<T>> ParseCsvAsync<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

Теперь использование будет выглядеть так:

csharp

// Пример использования

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ";", // Теперь указываем строку

hasHeader: true);

// Или с кастомной конфигурацией

var peopleWithConfig = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ";",

hasHeader: true,

config => {

config.RegisterClassMap<PersonMap>();

config.ShouldQuote = args => true;

});

Таким образом мы избегаем проблем с типами данных и делаем библиотеку более универсальной.

Выложи весь полный код, а то у меня опять ошибки появились при редактирвании.

Вот полный исправленный код библиотеки для парсинга CSV:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Text;

using System.Threading.Tasks;

namespace CsvParserLibrary

{

// Базовый класс для маппинга CSV

public abstract class CsvClassMap<T> : ClassMap<T> where T : class

{

protected CsvClassMap()

{

// Указываем конфигурацию для AutoMap

AutoMap(new CsvConfiguration(CultureInfo.InvariantCulture));

}

}

// Основной класс для работы с CSV

public static class CsvParser

{

/// <summary>

/// Синхронный метод парсинга CSV файла

/// </summary>

public static List<T> ParseCsv<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

/// <summary>

/// Асинхронный метод парсинга CSV файла

/// </summary>

public static async Task<List<T>> ParseCsvAsync<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return await ParseCsvInternalAsync<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding,

string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

// Возможность дополнительной настройки конфигурации

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding,

string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return await csv.GetRecords<T>().ToListAsync();

}

}

}

// Пример использования

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public override string ToString()

{

return $"Id: {Id}, Name: {Name}, Age: {Age}";

}

}

public class PersonMap : CsvClassMap<Person>

{

public PersonMap()

{

Map(m => m.Id).Name("Id");

Map(m => m.Name).Name("Name");

Map(m => m.Age).Name("Age");

}

}

}

**Как использовать библиотеку**

**Базовый пример использования:**

csharp

try

{

// Синхронный способ

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ";",

hasHeader: true);

// Асинхронный способ

var peopleAsync = await CsvParser.ParseCsvAsync<Person>(

"example.csv",

Encoding.UTF8,

del

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding, string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class { var config = new CsvConfiguration(CultureInfo.InvariantCulture) { Delimiter = delimiter, HasHeaderRecord = hasHeader }; configAction?.Invoke(config); using (var stream = new FileStream(filePath, FileMode.Open)) using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8)) using (var csv = new CsvReader(reader, config)) 80 { return await csv.GetRecords<T>().ToListAsync(); } } ----------- Severity Code Description Project File Line Suppression State Details Error (active) CS1061 'IEnumerable<T>' does not contain a definition for 'ToListAsync' and no accessible extension method 'ToListAsync' accepting a first argument of type 'IEnumerable<T>' could be found (are you missing a using directive or an assembly reference?) CsvParseLib02 F:\Projects\VS\CsvSln\CsvParseLib02\Class1.cs 80

Проблема в том, что метод **ToListAsync()** не является встроенным для IEnumerable. Нужно добавить необходимые using и использовать правильный способ асинхронного преобразования.

Вот исправленный код:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq; // Добавляем это using

using System.Text;

using System.Threading.Tasks;

namespace CsvParserLibrary

{

// Базовый класс для маппинга CSV

public abstract class CsvClassMap<T> : ClassMap<T> where T : class

{

protected CsvClassMap()

{

// Указываем конфигурацию для AutoMap

AutoMap(new CsvConfiguration(CultureInfo.InvariantCulture));

}

}

// Основной класс для работы с CSV

public static class CsvParser

{

/// <summary>

/// Синхронный метод парсинга CSV файла

/// </summary>

public static List<T> ParseCsv<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

/// <summary>

/// Асинхронный метод парсинга CSV файла

/// </summary>

public static async Task<List<T>> ParseCsvAsync<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return await ParseCsvInternalAsync<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding,

string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

// Возможность дополнительной настройки конфигурации

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding,

string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

// Используем правильный способ асинхронного чтения

var records = csv.GetRecords<T>();

var list = new List<T>();

await foreach (var record in records)

{

list.Add(record);

}

return list;

}

}

}

// Пример использования

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public override string ToString()

{

return $"Id: {Id}, Name: {Name}, Age: {Age}";

}

}

public class PersonMap : CsvClassMap<Person>

{

public PersonMap()

{

Map(m => m.Id).Name("Id");

Map(m => m.Name).Name("Name");

Map(m => m.Age).Name("Age");

}

}

}

Основные изменения:

1. Добавлен using System.Linq
2. Изменен способ асинхронного чтения данных:

private static async Task<List<T>> ParseCsvInternalAsync<T>(string filePath, Encoding encoding, string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class { var config = new CsvConfiguration(CultureInfo.InvariantCulture) { Delimiter = delimiter, HasHeaderRecord = hasHeader }; configAction?.Invoke(config); using (var stream = new FileStream(filePath, FileMode.Open)) using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8)) using (var csv = new CsvReader(reader, config)) { // Используем правильный способ асинхронного чтения var records = csv.GetRecords<T>(); var list = new List<T>(); 85 await foreach (var record in records) { list.Add(record); } return list; } } --------------------- Severity Code Description Project File Line Suppression State Details Error (active) CS8415 Asynchronous foreach statement cannot operate on variables of type 'IEnumerable<T>' because 'IEnumerable<T>' does not contain a public instance or extension definition for 'GetAsyncEnumerator'. Did you mean 'foreach' rather than 'await foreach'? CsvParserLibrary04 F:\Projects\VS\CsvSln\CsvParserLibrary04\Class1.cs 85

Давайте исправим асинхронную часть кода, так как CsvReader не предоставляет асинхронного перечисления. Мы можем использовать обычный foreach и обернуть его в асинхронную операцию:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsvParserLibrary

{

// Базовый класс для маппинга CSV

public abstract class CsvClassMap<T> : ClassMap<T> where T : class

{

protected CsvClassMap()

{

// Указываем конфигурацию для AutoMap

AutoMap(new CsvConfiguration(CultureInfo.InvariantCulture));

}

}

// Основной класс для работы с CSV

public static class CsvParser

{

/// <summary>

/// Синхронный метод парсинга CSV файла

/// </summary>

public static List<T> ParseCsv<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

/// <summary>

/// Асинхронный метод парсинга CSV файла

/// </summary>

public static async Task<List<T>> ParseCsvAsync<T>(string filePath, Encoding encoding = null,

string delimiter = ",", bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return await Task.Run(() =>

ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction));

}

private static List<T> ParseCsvInternal<T>(string filePath, Encoding encoding,

string delimiter, bool hasHeader, Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

// Возможность дополнительной настройки конфигурации

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

}

// Пример использования

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public override string ToString()

{

return $"Id: {Id}, Name: {Name}, Age: {Age}";

}

}

public class PersonMap : CsvClassMap<Person>

{

public PersonMap()

{

Map(m => m.Id).Name("Id");

Map(m => m.Name).Name("Name");

Map(m => m.Age).Name("Age");

}

}

}

Основные изменения:

1. Убрали асинхронное перечисление
2. Обернули синхронную операцию в Task.Run() для асинхронного выполнения
3. Упростили код, так как асинхронное чтение CSV напрямую не поддерживается CsvHelper

Теперь асинхронный метод будет работать корректно, выполняя операцию в фоновом потоке.

Это весь код или его часть ?

Это полный код библиотеки для парсинга CSV-файлов. Давайте я представлю его в структурированном виде с пояснениями:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsvParserLibrary

{

// Базовый класс для маппинга CSV

public abstract class CsvClassMap<T> : ClassMap<T> where T : class

{

protected CsvClassMap()

{

// Настройка автоматического маппинга

AutoMap(new CsvConfiguration(CultureInfo.InvariantCulture));

}

}

// Основной класс для работы с CSV

public static class CsvParser

{

/// <summary>

/// Синхронный метод парсинга CSV файла

/// </summary>

public static List<T> ParseCsv<T>(string filePath,

Encoding encoding = null,

string delimiter = ",",

bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

return ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction);

}

/// <summary>

/// Асинхронный метод парсинга CSV файла

/// </summary>

public static async Task<List<T>> ParseCsvAsync<T>(string filePath,

Encoding encoding = null,

string delimiter = ",",

bool hasHeader = true,

Action<CsvConfiguration> configAction = null) where T : class

{

// Выполняем операцию в фоновом потоке

return await Task.Run(() =>

ParseCsvInternal<T>(filePath, encoding, delimiter, hasHeader, configAction));

}

private static List<T> ParseCsvInternal<T>(string filePath,

Encoding encoding,

string delimiter,

bool hasHeader,

Action<CsvConfiguration> configAction) where T : class

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = delimiter,

HasHeaderRecord = hasHeader

};

// Возможность дополнительной настройки конфигурации

configAction?.Invoke(config);

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

}

}

// Пример использования

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public override string ToString()

{

return $"Id: {Id}, Name: {Name}, Age: {Age}";

}

}

public class PersonMap : CsvClassMap<Person>

{

public PersonMap()

{

Map(m => m.Id).Name("Id");

Map(m => m.Name).Name("Name");

Map(m => m.Age).Name("Age");

}

}

}

Этот код включает:

1. Базовую структуру для маппинга CSV-данных
2. Синхронные и асинхронные методы парсинга
3. Гибкую систему настройки параметров чтения
4. Пример класса для демонстрации работы
5. Маппинг для примера

Вы можете использовать эту библиотеку следующим образом:

csharp

try

{

// Синхронный способ

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ";",

hasHeader: true);

// Асинхронный способ

var peopleAsync = await CsvParser.ParseCsvAsync<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ";",

hasHeader: true);

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

static async Task Main() { Console.OutputEncoding = Encoding.UTF8; try { // Синхронный способ var people = CsvParser.ParseCsv<Person>( "example.csv", Encoding.UTF8, delimiter: ",", hasHeader: true); foreach (var record in people) { Console.WriteLine($"{record.Id}, {record.Name}, {record.Age}"); } // Асинхронный способ var peopleAsync = await CsvParser.ParseCsvAsync<Person>( "example.csv", Encoding.UTF8, delimiter: ",", hasHeader: true); 71 await foreach (var record in peopleAsync) { Console.WriteLine($"{record.Id}, {record.Name}, {record.Age}"); } } catch (Exception ex) { Console.WriteLine($"Error in CsvParser: {ex.Message}"); } Console.WriteLine("Hello, World!"); } ------ Severity Code Description Project File Line Suppression State Details Error (active) CS8415 Asynchronous foreach statement cannot operate on variables of type 'List<Person>' because 'List<Person>' does not contain a public instance or extension definition for 'GetAsyncEnumerator'. Did you mean 'foreach' rather than 'await foreach'? CaParserClient05 F:\Projects\VS\CsvSln\CaParserClient05\Program.cs 71 Почему 71 ошибка ?

Ошибка возникает потому, что вы пытаетесь использовать await foreach с обычной коллекцией List<T>, которая не поддерживает асинхронное перечисление. Давайте исправим код:

csharp

static async Task Main()

{

Console.OutputEncoding = Encoding.UTF8;

try

{

// Синхронный способ

var people = CsvParser.ParseCsv<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ",",

hasHeader: true);

foreach (var record in people)

{

Console.WriteLine($"{record.Id}, {record.Name}, {record.Age}");

}

// Асинхронный способ

var peopleAsync = await CsvParser.ParseCsvAsync<Person>(

"example.csv",

Encoding.UTF8,

delimiter: ",",

hasHeader: true);

// Исправленный foreach без await

foreach (var record in peopleAsync)

{

Console.WriteLine($"{record.Id}, {record.Name}, {record.Age}");

}

}

catch (Exception ex)

{

Console.WriteLine($"Error in CsvParser: {ex.Message}");

}

Console.WriteLine("Hello, World!");

}

Основные изменения:

1. Заменено await foreach на обычный foreach для работы с List<Person>
2. Всё остальное остаётся без изменений

Теперь код будет работать корректно. await foreach используется только с асинхронными перечислениями (например, с IAsyncEnumerable), а List<T> — это обычная коллекция, которая не требует асинхронного перечисления.

Также напомню, что асинхронный метод ParseCsvAsync в нашей библиотеке на самом деле выполняет операцию синхронно в фоновом потоке через Task.Run(), поэтому использование await перед ним всё ещё имеет смысл, если вы хотите, чтобы основной поток не блокировался во время чтения файла.

Вот этот код в библиотеке Что делает ? // Возможность дополнительной настройки конфигурации configAction?.Invoke(config); using (var stream = new FileStream(filePath, FileMode.Open)) using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8)) using (var csv = new CsvReader(reader, config)) { return csv.GetRecords<T>().ToList(); }

Давайте разберем этот код по частям:

1. configAction?.Invoke(config);

Это необязательный параметр делегата Action<CsvConfiguration>, который позволяет пользователю дополнительно настроить конфигурацию CsvReader перед началом чтения файла.

Пример использования:

csharp

var people = CsvParser.ParseCsv<Person>(

"example.csv",

config: config => {

config.RegisterClassMap<PersonMap>();

config.ShouldQuote = args => true;

config.Delimiter = ";";

});

1. Блок using с потоками:

csharp

using (var stream = new FileStream(filePath, FileMode.Open))

using (var reader = new StreamReader(stream, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, config))

{

return csv.GetRecords<T>().ToList();

}

Этот блок отвечает за:

* Открытие файла через FileStream
* Создание StreamReader для чтения потока с указанной кодировкой (или UTF-8 по умолчанию)
* Создание CsvReader с заданной конфигурацией
* Автоматическое освобождение ресурсов благодаря конструкции using

Пояснение работы:

1. FileStream открывает файл для чтения
2. StreamReader преобразует байты в читаемый текст с учетом кодировки
3. CsvReader парсит текст как CSV согласно заданной конфигурации
4. GetRecords<T>() преобразует строки CSV в объекты типа T
5. .ToList() преобразует результат в список

Преимущества такого подхода:

* Автоматическое управление ресурсами (файлы и потоки закрываются автоматически)
* Гибкая настройка через делегат configAction
* Типобезопасное преобразование CSV в объекты
* Возможность обработки больших файлов благодаря потоковому чтению

Если вам нужно добавить какую-то специфическую логику обработки CSV, вы можете использовать параметр configAction для настройки дополнительных опций CsvHelper.

Продолжим, то есть начнем немного с начала. По-прежнему используем CsvHelper - Nuget package У меня есть два простых кода - Write CSV - ReadCsv ---------------------------- Мы пополняем нашу библиотеку новыми фукциями Необходимо взять их код и инкапсулировать его в две функции (каждый код в свою функцию) с соответствующими входными параматрами (имя файла, итд) Наверное все-таки лучше работать не List, все-таки с IEnumerable Сделать обычную и async версии Код REadCSV ---------------------- using CsvHelper; using System.Globalization; using System.IO; using System.Linq; public class Person { public int Id { get; set; } public string Name { get; set; } public int Age { get; set; } } class Program { static void Main() { using (var reader = new StreamReader("F:\\Projects\\VS\\CsvSln\\data.csv")) using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture)) { var records = csv.GetRecords<Person>().ToList(); foreach (var record in records) { Console.WriteLine($"{record.Id}, {record.Name}, {record.Age}"); } } } } ----------------------- Код WriteCSv ------------------------ using CsvHelper; using System.Formats.Asn1; using System.Globalization; using System.IO; public class Person { public int Id { get; set; } public string Name { get; set; } public int Age { get; set; } } class Program { static void Main() { var people = new List<Person> { new Person { Id = 1, Name = "John Doe", Age = 30 }, new Person { Id = 2, Name = "Jane Smith", Age = 25 }, new Person { Id = 3, Name = "Bob Johnson", Age = 40 } }; using (var writer = new StreamWriter("F:\\Projects\\VS\\CsvSln\\data.csv")) using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture)) { csv.WriteRecords(people); } } } ---------------- Если задача понятна, то создаем код, Если не понятно, то заавайте вопросы

Давайте создадим библиотеку с методами для чтения и записи CSV файлов.

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsvLibrary

{

// Базовый класс для работы с CSV

public static class CsvOperations

{

// Чтение CSV - синхронная версия

public static IEnumerable<T> ReadCsv<T>(string filePath, Encoding encoding = null)

{

using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))

{

return csv.GetRecords<T>();

}

}

// Чтение CSV - асинхронная версия

public static async Task<IEnumerable<T>> ReadCsvAsync<T>(string filePath, Encoding encoding = null)

{

using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))

{

var records = new List<T>();

while (await csv.ReadAsync())

{

records.Add(csv.GetRecord<T>());

}

return records;

}

}

// Запись CSV - синхронная версия

public static void WriteCsv<T>(string filePath, IEnumerable<T> records, Encoding encoding = null)

{

using (var writer = new StreamWriter(filePath, false, encoding ?? Encoding.UTF8))

using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture))

{

csv.WriteRecords(records);

}

}

// Запись CSV - асинхронная версия

public static async Task WriteCsvAsync<T>(string filePath, IEnumerable<T> records, Encoding encoding = null)

{

using (var writer = new StreamWriter(filePath, false, encoding ?? Encoding.UTF8))

using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture))

{

await csv.WriteRecordsAsync(records);

}

}

}

// Пример класса для демонстрации

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

}

}

Пример использования:

csharp

class Program

{

static async Task Main()

{

// Пример записи CSV

var people = new List<Person>

{

new Person { Id = 1, Name = "John Doe", Age = 30 },

new Person { Id = 2, Name = "Jane Smith", Age = 25 },

new Person { Id = 3, Name = "Bob Johnson", Age = 40 }

};

// Синхронная запись

CsvOperations.WriteCsv("data.csv", people);

// Асинхронная запись

await CsvOperations.WriteCsvAsync("data.csv", people);

// Пример чтения CSV

// Синхронное чтение

var readPeople = CsvOperations.ReadCsv<Person>("data.csv");

foreach (var person in readPeople)

{

Console.WriteLine($"{person.Id}, {person.Name}, {person.Age}");

}

// Асинхронное чтение

var readPeopleAsync = await CsvOperations.ReadCsvAsync<Person>("data.csv");

foreach (var person in readPeopleAsync)

{

Console.WriteLine($"{person.Id}, {person.Name}, {person.Age}");

}

}

}

Основные особенности реализации:

1. Методы работают с IEnumerable<T> вместо List<T> для более эффективной работы с большими данными
2. Добавлены как синхронные, так и асинхронные версии методов
3. Поддержка различных кодировок через параметр Encoding
4. Автоматическое управление ресурсами через using
5. Гибкая настройка через обобщенные типы

Вы можете расширить функциональность, добавив дополнительные параметры конфигурации (разделитель, наличие заголовков и т.д.) по необходимости.

Почему то выдает исключение в синхронном методе Console.WriteLine("\nSimple fucntions"); // Пример записи CSV var persons = new List<Person> { new Person { Id = 1, Name = "John Doe", Age = 32 }, new Person { Id = 2, Name = "Jane Smith", Age = 25 }, new Person { Id = 3, Name = "Bob Johnson", Age = 42 } }; // Синхронная запись CsvOperations.WriteCsv("data.csv", people); // Асинхронная запись await CsvOperations.WriteCsvAsync("data.csv", people); // Пример чтения CSV // Синхронное чтение var readPeople = CsvOperations.ReadCsv<Person>("data.csv"); foreach (var person in readPeople) <<<<<<<<<<<<<<<<< Вот здесь выдает исключение { Console.WriteLine($"{person.Id}, {person.Name}, {person.Age}"); } // Асинхронное чтение var readPeopleAsync = await CsvOperations.ReadCsvAsync<Person>("data.csv"); foreach (var person in readPeopleAsync) { Console.WriteLine($"{person.Id}, {person.Name}, {person.Age}"); } Исключение System.ObjectDisposedException: 'GetRecords<T>() returns an IEnumerable<T> that yields records. This means that the method isn't actually called until you try and access the values. e.g. .ToList() Did you create CsvReader inside a using block and are now trying to access the records outside of that using block? Object name: 'CsvReader'.'

Проблема возникает потому, что CsvReader уничтожается до того, как мы успеваем получить все записи. Это происходит из-за того, что GetRecords<T>() возвращает отложенное выполнение (deferred execution).

Давайте исправим метод чтения:

csharp

// Чтение CSV - синхронная версия

public static IEnumerable<T> ReadCsv<T>(string filePath, Encoding encoding = null)

{

using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))

{

// Сохраняем записи в список перед возвратом

var records = csv.GetRecords<T>().ToList();

return records;

}

}

// Чтение CSV - асинхронная версия

public static async Task<IEnumerable<T>> ReadCsvAsync<T>(string filePath, Encoding encoding = null)

{

using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))

{

var records = new List<T>();

while (await csv.ReadAsync())

{

records.Add(csv.GetRecord<T>());

}

return records;

}

}

Теперь код будет работать корректно, потому что:

1. В синхронной версии мы сразу преобразуем результат в список
2. В асинхронной версии мы также собираем все записи в список

Полный исправленный код библиотеки:

csharp

using CsvHelper;

using CsvHelper.Configuration;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsvLibrary

{

public static class CsvOperations

{

// Чтение CSV - синхронная версия

public static IEnumerable<T> ReadCsv<T>(string filePath, Encoding encoding = null)

{

using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))

{

return csv.GetRecords<T>().ToList();

}

}

// Чтение CSV - асинхронная версия

public static async Task<IEnumerable<T>> ReadCsvAsync<T>(string filePath, Encoding encoding = null)

{

using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8))

using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))

{

var records = new List<T>();

while (await csv.ReadAsync())

{

records.Add(csv.GetRecord<T>());

}

return records;

}

}

// Запись CSV - синхронная версия

public static void WriteCsv<T>(string filePath, IEnumerable<T> records, Encoding encoding = null)

{

using (var writer = new StreamWriter(filePath, false, encoding ?? Encoding.UTF8))

using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture))

{

csv.WriteRecords(records);

}

}

// Запись CSV - асинхронная версия

public static async Task WriteCsvAsync<T>(string filePath, IEnumerable<T> records, Encoding encoding = null)

{

using (var writer = new StreamWriter(filePath, false, encoding ?? Encoding.UTF8))

using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture))

{

await csv.WriteRecordsAsync(records);

}

}

}

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

}

}

Теперь вы можете использовать этот код без ошибок:

csharp

Console.WriteLine("\nSimple functions");

// Пример записи CSV

var persons = new List<Person>

{

new Person { Id = 1, Name = "John Doe", Age = 32 },

new Person { Id = 2, Name = "Jane Smith", Age = 25 },

new Person { Id = 3, Name = "Bob Johnson", Age = 42

using CsvHelper; using CsvHelper.Configuration; using System; using System.Collections.Generic; using System.Globalization; using System.IO; using System.Linq; using System.Text; using System.Threading.Tasks; namespace CsvLibrary { public static class CsvOperations { // Чтение CSV - синхронная версия public static IEnumerable<T> ReadCsv<T>(string filePath, Encoding encoding = null) { using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8)) using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture)) { return csv.GetRecords<T>().ToList(); } } // Чтение CSV - асинхронная версия public static async Task<IEnumerable<T>> ReadCsvAsync<T>(string filePath, Encoding encoding = null) { using (var reader = new StreamReader(filePath, encoding ?? Encoding.UTF8)) using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture)) { var records = new List<T>(); while (await csv.ReadAsync()) { records.Add(csv.GetRecord<T>()); } return records; } } // Запись CSV - синхронная версия public static void WriteCsv<T>(string filePath, IEnumerable<T> records, Encoding encoding = null) { using (var writer = new StreamWriter(filePath, false, encoding ?? Encoding.UTF8)) using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture)) { csv.WriteRecords(records); } } // Запись CSV - асинхронная версия public static async Task WriteCsvAsync<T>(string filePath, IEnumerable<T> records, Encoding encoding = null) { using (var writer = new StreamWriter(filePath, false, encoding ?? Encoding.UTF8)) using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture)) { await csv.WriteRecordsAsync(records); } } } public class Person { public int Id { get; set; } public string Name { get; set; } public int Age { get; set; } } } напиши пример использования вот этого коды

Вот несколько примеров использования данной библиотеки CsvLibrary:

**1. Простой пример записи и чтения данных синхронно:**

csharp

var persons = new List<Person>

{

new Person { Id = 1, Name = "Иван Иванов", Age = 30 },

new Person { Id = 2, Name = "Пётр Петров", Age = 25 },

new Person { Id = 3, Name = "Сергей Сидоров", Age = 35 }

};

// Запись в CSV-файл синхронно

CsvOperations.WriteCsv("persons.csv", persons);

// Чтение из CSV-файла синхронно

var readPersons = CsvOperations.ReadCsv<Person>("persons.csv");

foreach (var person in readPersons)

{

Console.WriteLine($"{person.Id} - {person.Name}, {person.Age} лет");

}

**2. Пример использования асинхронных методов:**

csharp

async Task Main()

{

var persons = new List<Person>

{

new Person { Id = 4, Name = "Анна Смирнова", Age = 28 },

new Person { Id = 5, Name = "Ольга Волкова", Age = 40 }

};

// Асинхронная запись

await CsvOperations.WriteCsvAsync("persons2.csv", persons);

// Асинхронное чтение

var asyncReadPersons = await CsvOperations.ReadCsvAsync<Person>("persons2.csv");

foreach (var person in asyncReadPersons)

{

Console.WriteLine($"{person.Id} - {person.Name}, {person.Age} лет");

}

}

**3. Пример с указанием кодировки:**

csharp

// Указываем кодировку UTF-16

var encoding = Encoding.Unicode;

// Запись с указанной кодировкой

CsvOperations.WriteCsv("persons\_utf16.csv", persons, encoding);

// Чтение с указанной кодировкой

var utf16Persons = CsvOperations.ReadCsv<Person>("persons\_utf16.csv", encoding);

**4. Пример комбинированного использования:**

csharp

async Task Main()

{

// Создаём список объектов

var people = new List<Person>();

// Заполняем данными...

// Сначала записываем синхронно

CsvOperations.WriteCsv("sync\_persons.csv", people);

// Затем читаем асинхронно

var asyncPersons = await CsvOperations.ReadCsvAsync<Person>("sync\_persons.csv");

// Обрабатываем данные

foreach (var person in asyncPersons)

{

Console.WriteLine(person.Name);

}

}

**5. Пример обработки ошибок:**

csharp

try

{

var persons = await CsvOperations.ReadCsvAsync<Person>("non\_existent\_file.csv");

}

catch (FileNotFoundException ex)

{

Console.WriteLine($"Файл не найден: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

Эти примеры демонстрируют основные сценарии использования библиотеки: запись и чтение CSV-файлов синхронно и асинхронно, работу с разными кодировками и обработку возможных ошибок.

using CsvParserLibrary07; using System.Text; async Task Main() { Console.OutputEncoding = Encoding.UTF8; var persons = new List<Person> { new Person { Id = 4, Name = "Анна Смирнова", Age = 28 }, new Person { Id = 5, Name = "Ольга Волкова", Age = 40 } }; // Асинхронная запись await CsvOperations.WriteCsvAsync("persons2.csv", persons); // Асинхронное чтение var asyncReadPersons = await CsvOperations.ReadCsvAsync<Person>("persons2.csv"); foreach (var person in asyncReadPersons) { Console.WriteLine($"{person.Id} - {person.Name}, {person.Age} лет"); } } ----- не входим в main() потому что async Task Main() Как написать чтобы код входил в Main() ?

Есть несколько способов решить проблему с входом в Main() для асинхронного кода:

**1. Использовать**async Main**(C# 7.1 и выше)**

Если вы используете актуальную версию C# (7.1 или выше), можно просто объявить Main как async Task:

csharp

using CsvParserLibrary07;

using System.Text;

public static async Task Main()

{

Console.OutputEncoding = Encoding.UTF8;

var persons = new List<Person>

{

new Person { Id = 4, Name = "Анна Смирнова", Age = 28 },

new Person { Id = 5, Name = "Ольга Волкова", Age = 40 }

};

// Асинхронная запись

await CsvOperations.WriteCsvAsync("persons2.csv", persons);

// Асинхронное чтение

var asyncReadPersons = await CsvOperations.ReadCsvAsync<Person>("persons2.csv");

foreach (var person in asyncReadPersons)

{

Console.WriteLine($"{person.Id} - {person.Name}, {person.Age} лет");

}

}

**2. Обернуть асинхронный код в синхронный**Main

Если версия C# не позволяет использовать async Main, можно обернуть асинхронный код в синхронный:

csharp

using CsvParserLibrary07;

using System.Text;

public static void Main()

{

MainAsync().GetAwaiter().GetResult();

}

private static async Task MainAsync()

{

Console.OutputEncoding = Encoding.UTF8;

var persons = new List<Person>

{

new Person { Id = 4, Name = "Анна Смирнова", Age = 28 },

new Person { Id = 5, Name = "Ольга Волкова", Age = 40 }

};

// Асинхронная запись

await CsvOperations.WriteCsvAsync("persons2.csv", persons);

// Асинхронное чтение

var asyncReadPersons = await CsvOperations.ReadCsvAsync<Person>("persons2.csv");

foreach (var person in asyncReadPersons)

{

Console.WriteLine($"{person.Id} - {person.Name}, {person.Age} лет");

}

}

**3. Использовать**ConfigureAwait(false)

Иногда помогает добавить ConfigureAwait(false) для избежания проблем с контекстом синхронизации:

csharp

await CsvOperations.WriteCsvAsync("persons2.csv", persons).ConfigureAwait(false);

var asyncReadPersons = await CsvOperations.ReadCsvAsync<Person>("persons2.csv").ConfigureAwait(false);

**4. Проверить настройки проекта**

Убедитесь, что:

* в проекте включена поддержка асинхронного Main;
* используется актуальная версия .NET;
* в файле проекта (csproj) нет ограничений на версию языка.

**5. Использовать**.Wait()**(не рекомендуется)**

Этот способ менее предпочтителен, так как может привести к блокировке потоков:

csharp

public static void Main()

{

Console.OutputEncoding = Encoding.UTF8;

var persons = new List<Person>

{

// ...

};

CsvOperations.WriteCsvAsync("persons2.csv", persons).Wait();

var asyncReadPersons = CsvOperations.ReadCsvAsync<Person>("persons2.csv").Result;

// ...

}

**Рекомендация:** лучше всего использовать первый вариант (async Task Main), так как он является современным и рекомендуемым подходом в C#.