Introduction

CodaParser is designed to simplify the process of parsing Belgian CODA banking files within .NET applications. By leveraging modern C# features and adhering to the latest .NET standards, CodaParser ensures both performance and reliability.

Originally forked from supervos/coda-parser, CodaParser enhances functionality and updates dependencies to better fit the evolving .NET ecosystem.

What is CODA?

CODA (Common Organization Data Access) is a standardized format used by Belgian banks to communicate transaction data. Parsing CODA files is essential for applications that handle financial data, enabling them to process transactions, balances, and other relevant information efficiently.

Why CodaParser?

- **Reliability:** Based on a well-established project, ensuring accurate and dependable parsing.
- **Extensibility:** Designed to be easily extendable, allowing developers to add custom features or processing logic.
- **Modern Standards:** Utilizes the latest C# and .NET features for optimal performance and maintainability.

Whether you're building a personal finance app or an enterprise-level financial system, CodaParser provides the tools necessary to handle CODA files seamlessly.

Getting Started

This guide will help you set up and start using CodaParser in your .NET projects.

Prerequisites

- **.NET SDK:** Ensure you have the .NET 8 SDK installed.
- **Git:** Required for cloning the repository and version control.
- **Optional Tools:**
  - **DocFX:** For generating documentation.
  - **Nuke:** For build automation (if using the provided Nuke build script).

Installation

NuGet Package

CodaParser is available as a NuGet package. Install it using the following commands:

Using .NET CLI

```
dotnet add package CodaParser
```

Using Package Manager Console

```
Install-Package CodaParser
```

Building from Source

If you prefer to build the library yourself, follow these steps:

1. **Clone the Repository**

   ```
   git clone https://github.com/phmatray/CodaParser.git
   cd CodaParser
   ```

2. **Restore Dependencies**

   Ensure you have the .NET SDK installed. Then run:

   ```
   dotnet restore
   ```

3. **Build the Project**

```
dotnet build
```

4. **Use the Library**

   Reference the built DLLs in your project by adding them to your project's dependencies.

Basic Usage

Below is a simple example demonstrating how to parse a CODA file and iterate through the statements and transactions using the `IParser<T>` interface.

```
using CodaParser;
using System.Collections.Generic;

// Instantiate the parser (assuming a concrete implementation is available)
IParser<Statement> parser = new Parser();

// Parse the CODA file
IEnumerable<Statement> statements = parser.ParseFile("path/to/coda-file.cod");

// Iterate through the statements and transactions
foreach (var statement in statements)
{
    Console.WriteLine($"Date: {statement.Date:yyyy-MM-dd}");

    foreach (var transaction in statement.Transactions)
    {
        Console.WriteLine($"{transaction.Account.Name}: {transaction.Amount}");
    }

    Console.WriteLine($"New Balance: {statement.NewBalance}");
    Console.WriteLine(new string('-', 40));
}
```

Minimal API Example (.NET 8)

Leveraging .NET 8's minimal API features, you can create a streamlined application without explicit `Program` and `Main` methods:

```
using CodaParser;
using System.Collections.Generic;

IParser<Statement> parser = new Parser();
```

```
IEnumerable<Statement> statements = parser.ParseFile("path/to/coda-file.cod");

foreach (var statement in statements)
{
    Console.WriteLine($"Date: {statement.Date:yyyy-MM-dd}");

    foreach (var transaction in statement.Transactions)
    {
        Console.WriteLine($"{transaction.Account.Name}: {transaction.Amount}");
    }

    Console.WriteLine($"New Balance: {statement.NewBalance}");
    Console.WriteLine(new string('-', 40));
}
```

Running the Example

1. **Create a New .NET Project**

   ```
   dotnet new console -n CodaParserDemo
   cd CodaParserDemo
   ```

2. **Add CodaParser Package**

   ```
   dotnet add package CodaParser
   ```

3. **Replace `Program.cs` with the Example Code**

4. **Run the Application**

   ```
   dotnet run
   ```

Ensure that the path to your CODA file is correct. The application will parse the file and display the transactions in the console.

Features

CodaParser offers a robust set of features designed to facilitate the parsing of Belgian CODA banking files within .NET applications.

Comprehensive .NET Standard 2.0 Support

- **Compatibility:** Compatible with both .NET Framework and .NET Core, ensuring broad usability across different project types and environments.
- **Modern C# Features:** Utilizes the latest C# language enhancements, providing improved performance and code readability.

Reliable Parsing Mechanism

- **Accurate Data Extraction:** Precisely parses CODA files, extracting essential information such as transactions, balances, and account details.
- **Error Handling:** Robust error handling mechanisms to manage and report parsing issues effectively.

Easy Integration

- **Simple API:** Offers a straightforward API (`IParser<T>`) that can be easily integrated into existing projects.
- **Extensible:** Designed to be easily extendable, allowing developers to add custom features or processing logic as needed.

Updated Dependencies

- **Stability and Security:** Leverages the latest NuGet packages, ensuring better stability and security for your applications.
- **Active Maintenance:** Regular updates to dependencies to keep the library aligned with the evolving .NET ecosystem.

Extensibility

- **Custom Processing:** Allows for custom processing of parsed data, enabling developers to tailor the library to specific project requirements.
- **Modular Design:** Modular architecture facilitates easy extension and maintenance.

Based on a Proven Foundation

- **Forked from a Trusted Source:** Based on the well-established supervos/coda-parser project by Christophe Devos.

- **Community-Driven:** Benefits from community contributions and support, ensuring continuous improvement and feature enhancements.

CodaParser is committed to providing a reliable and efficient solution for parsing CODA files, empowering developers to manage financial data with ease and confidence.

API Reference

CodaParser provides a clean and intuitive API for parsing CODA files into structured data. Below is an overview of the key components and interfaces.

`IParser<T>` Interface

The `IParser<T>` interface defines the contract for parsing CODA files into a specific type.

```
namespace CodaParser;

/// <summary>
/// Interface for parsing a list of strings or a file to a specific type.
/// </summary>
/// <typeparam name="T">The result type of the parsing.</typeparam>
public interface IParser<out T>
{
    /// <summary>
    /// Parses a collection of CODA lines into their matching type.
    /// </summary>
    /// <param name="codaLines">The strings to parse.</param>
    /// <returns>The parsed result.</returns>
    IEnumerable<T> Parse(IEnumerable<string> codaLines);

    /// <summary>
    /// Reads the content of the specified file and parses it into the
matching type.
    /// </summary>
    /// <param name="codaFile">Location of the file to parse.</param>
    /// <returns>The parsed result.</returns>
    IEnumerable<T> ParseFile(string codaFile);
}
```

Implementing `IParser<T>`

To create a custom parser, implement the `IParser<T>` interface:

```
using CodaParser;
using System.Collections.Generic;

public class CustomParser : IParser<Statement>
{
    public IEnumerable<Statement> Parse(IEnumerable<string> codaLines)
    {
        // Implement your parsing logic here
    }

    public IEnumerable<Statement> ParseFile(string codaFile)
    {
        var lines = File.ReadAllLines(codaFile);
        return Parse(lines);
    }
}
```

`Parser` Class

The `Parser` class is a concrete implementation of the `IParser<Statement>` interface, providing methods to parse CODA files into `Statement` objects.

Methods

- **`IEnumerable<Statement> Parse(IEnumerable<string> codaLines)`**

  Parses a collection of CODA lines and returns the corresponding `Statement` objects.

- **`IEnumerable<Statement> ParseFile(string codaFile)`**

  Reads the content of the specified CODA file and parses it into `Statement` objects.

`Statement` Class

Represents a single banking statement extracted from a CODA file.

Properties

- **`Date`** (`DateTime`): The date of the statement.
- **`Account`** (`Account`): The account information associated with the statement.
- **`InitialBalance`** (`decimal`): The initial balance of the account at the start of the statement period.
- **`NewBalance`** (`decimal`): The balance of the account after all transactions have been processed.
- **`InformationalMessage`** (`string`): Any informational messages included in the statement.
- **`Transactions`** (`IEnumerable<Transaction>`): A collection of transactions executed during the statement period.

Example

```
public class Statement
{
    public DateTime Date { get; set; }
    public Account Account { get; set; }
    public decimal InitialBalance { get; set; }
    public decimal NewBalance { get; set; }
    public string InformationalMessage { get; set; }
    public IEnumerable<Transaction> Transactions { get; set; }
}
```

`Account` Class

Represents account details within a statement.

Properties

- **`Name`** (`string`): The name of the account holder.
- **`Number`** (`string`): The account number.
- **`Type`** (`string`): The type of account (e.g., Checking, Savings).

`Transaction` Class

Represents a single transaction within a statement.

Properties

- **`Date`** (`DateTime`): The date of the transaction.
- **`Description`** (`string`): A description of the transaction.
- **`Amount`** (`decimal`): The amount involved in the transaction.
- **`Balance`** (`decimal`): The account balance after the transaction.

For more detailed information and advanced usage, refer to the official documentation.

Contributing

We welcome contributions to the CodaParser project! By contributing, you help improve the library and assist other developers in effectively parsing CODA banking files.

Code of Conduct

Please adhere to the Code of Conduct when interacting with the project.

How to Contribute

Follow these steps to contribute to CodaParser:

1. Fork the Repository

Start by forking the CodaParser repository to your GitHub account.

2. Clone Your Fork

Clone the forked repository to your local machine:

```
git clone https://github.com/phmatray/CodaParser.git
cd CodaParser
```

3. Create a Feature Branch

Create a new branch for your feature or bug fix:

```
git checkout -b feature/YourFeature
```

4. Make Your Changes

Implement your feature or fix the bug. Ensure your code adheres to the project's coding standards and includes appropriate tests.

5. Commit Your Changes

Use conventional commit messages to describe your changes. This helps maintain a consistent commit history.

Example Commit Message

```
feat(parser): add support for new transaction types
```

6. Push to Your Fork

Push your changes to your forked repository:

```
git push origin feature/YourFeature
```

7. Open a Pull Request

Navigate to the original CodaParser repository and open a pull request from your forked repository's feature branch.

8. Respond to Feedback

Maintain an open line of communication with the project maintainers. Address any feedback or requested changes promptly.

Reporting Issues

If you encounter any issues or bugs, please report them using the issue tracker. Include detailed information to help us reproduce and address the problem effectively.

Development Guidelines

- **Coding Standards:** Follow the existing coding conventions used throughout the project.
- **Testing:** Write unit tests for new features and ensure existing tests pass.
- **Documentation:** Update documentation as necessary to reflect your changes.
- **Commit Messages:** Use clear and descriptive commit messages following the Conventional Commits guidelines.

Thank You!

Your contributions are greatly appreciated and help make CodaParser a valuable tool for the developer community. Thank you for your interest and support!

License

Acknowledgements

CodaParser is built upon the foundation laid by several projects and communities. We extend our gratitude to the following:

- supervos/coda-parser by Christophe Devos for providing the original implementation and inspiration.
- wimverstuyf/php-coda-parser project for additional insights and features.
- **Open-Source Community:** Thanks to the open-source community for their invaluable contributions, feedback, and support.
- **Developers and Contributors:** Special thanks to all the contributors who have helped improve CodaParser through code, documentation, and testing.

Your collective efforts have been instrumental in shaping CodaParser into a reliable and efficient tool for parsing CODA banking files.