@fin.cx/mt940parser

Documentation for @fin.cx/mt940parser

- readme.md for @fin.cx/mt940parser
- changelog.md for @fin.cx/mt940parser

readme.md for @fin.cx/mt940parser

A parser to process and extract statements from MT940 format files.

Install

To install the `@fin.cx/mt940parser` package, you can use npm or yarn. Here is the command to install via npm:

```
npm install @fin.cx/mt940parser
```

Alternatively, if you are using yarn:

```
yarn add @fin.cx/mt940parser
```

Usage

`@fin.cx/mt940parser` is a library for parsing MT940 files, which are commonly used by banks for electronic account statements. This module enables you to easily parse the MT940 files into JavaScript objects for further processing. Below, we'll walk through various scenarios and features to help you get the most out of this library. We will use modern ECMAScript Module (ESM) syntax and TypeScript for code examples.

Basic Usage

To start parsing MT940 files using `@fin.cx/mt940parser`, you need to first import the library and create an instance of `Mt940Parser`. Here's a basic example of how to parse an MT940 file string:

```
import { Mt940Parser } from '@fin.cx/mt940parser';


const parser = new Mt940Parser();
```

```
const mt940FileString = `
// your MT940 file content here
`;

async function parseMT940() {
  try {
    const statements = await parser.parseMt940FileString(mt940FileString);
    console.log(statements);
  } catch (error) {
    console.error('Error parsing MT940 file:', error);
  }
}

parseMT940();
```

Reading from a File

Often, you'll want to read MT940 data from a file. Below is an example of how to do that using Node.js built-in filesystem (fs) module:

```
import { promises as fs } from 'fs';
import { Mt940Parser } from '@fin.cx/mt940parser';

async function parseMT940FromFile(filePath: string) {
  try {
    const fileContent = await fs.readFile(filePath, 'utf-8');
    const parser = new Mt940Parser();
    const statements = await parser.parseMt940FileString(fileContent);
    console.log(statements);
  } catch (error) {
    console.error('Error parsing MT940 file:', error);
  }
}

parseMT940FromFile('./path/to/your/mt940file.txt');
```

Parsing a Buffer

If you have the MT940 data in a buffer, you can use the `parseMt940Buffer` method to process it. Here's an example to demonstrate this:

```
import { promises as fs } from 'fs';
import { Mt940Parser } from '@fin.cx/mt940parser';

async function parseMT940FromBuffer(filePath: string) {
  try {
    const fileBuffer = await fs.readFile(filePath);
    const parser = new Mt940Parser();
    const statements = await parser.parseMt940Buffer(fileBuffer.buffer);
    console.log(statements);
  } catch (error) {
    console.error('Error parsing MT940 buffer:', error);
  }
}

parseMT940FromBuffer('./path/to/your/mt940file.txt');
```

Advanced Usage

Parsing with Custom Processing

The `parseMt940Buffer` method allows for further custom processing of parsed MT940 data. By leveraging this function, you can modify certain fields or process the data before utilizing it. Here's an example:

```
import { Mt940Parser } from '@fin.cx/mt940parser';

const customParser = new Mt940Parser();

customParser.parseMt940Buffer = async function (fileBuffer) {
  const parsedStatements = await customParser.parseMt940Buffer(fileBuffer);

  // Custom processing: Example, removing all transactions with amount 0
  for (const statement of parsedStatements) {
    statement.transactions = statement.transactions.filter(transaction => transaction.amount
!== 0);
  }
```

```
  return parsedStatements;
};

// Usage remains the same
async function parseWithCustomProcessing(mt940FileString: string) {
  const textEncode = new TextEncoder();
  const fileBuffer = textEncode.encode(mt940FileString).buffer;
  const statements = await customParser.parseMt940Buffer(fileBuffer);
  console.log(statements);
}

const mt940FileString = `
// your MT940 file content here
`;

parseWithCustomProcessing(mt940FileString);
```

Combining Multiple MT940 Files

It is common to have multiple MT940 files covering different periods. You can combine the data from multiple files and parse them together. Here is an example:

```
import { promises as fs } from 'fs';
import { Mt940Parser } from '@fin.cx/mt940parser';

async function mergeMT940Files(filePaths: string[]) {
  const parser = new Mt940Parser();
  const allStatements = [];

  for (const filePath of filePaths) {
    try {
      const fileContent = await fs.readFile(filePath, 'utf-8');
      const statements = await parser.parseMt940FileString(fileContent);
      allStatements.push(...statements);
    } catch (error) {
      console.error(`Error parsing file ${filePath}:`, error);
    }
  }
```

```
  return allStatements;
}

// Example file paths array
const filePaths = ['./path/to/first.mt940', './path/to/second.mt940'];

async function main() {
  const combinedStatements = await mergeMT940Files(filePaths);
  console.log(combinedStatements);
}

main();
```

Filtering and Analyzing Transactions

Once you have parsed the MT940 file into JavaScript objects, you might want to filter or analyze the transactions contained in the statements. Here's how you can filter transactions that meet specific criteria:

```
import { Mt940Parser } from '@fin.cx/mt940parser';

const parser = new Mt940Parser();

async function findLargeTransactions(mt940FileString: string, minAmount: number) {
  const statements = await parser.parseMt940FileString(mt940FileString);
  const largeTransactions = [];

  for (const statement of statements) {
    for (const transaction of statement.transactions) {
      if (transaction.amount >= minAmount) {
        largeTransactions.push(transaction);
      }
    }
  }

  return largeTransactions;
}

const mt940FileString = `
// your MT940 file content here
```

```
`;

async function main() {
  const largeTransactions = await findLargeTransactions(mt940FileString, 1000); // Replace
1000 with your desired minimum amount
  console.log('Large Transactions:', largeTransactions);
}

main();
```

Extending the Parser

If you need additional customization beyond what is provided, you can extend the `Mt940Parser` class and override the methods you need to change.

```
import { Mt940Parser } from '@fin.cx/mt940parser';

class CustomMt940Parser extends Mt940Parser {
  async parseMt940FileString(fileString: string) {
    const statements = await super.parseMt940FileString(fileString);

    // Custom logic: Example, add a new property customField to each statement
    for (const statement of statements) {
      statement.customField = 'your custom value';
    }

    return statements;
  }
}

// Usage remains the same
async function parseWithCustomParser(mt940FileString: string) {
  const customParser = new CustomMt940Parser();
  const statements = await customParser.parseMt940FileString(mt940FileString);
  console.log(statements);
}

const mt940FileString = `
// your MT940 file content here
`;
```

```
parseWithCustomParser(mt940FileString);
```

Error Handling and Validation

Proper error handling and validation are crucial when processing financial data. Here’s how to catch and handle errors effectively:

```
import { Mt940Parser } from '@fin.cx/mt940parser';

const parser = new Mt940Parser();

async function parseMT940WithErrorHandling(mt940FileString: string) {
  try {
    const statements = await parser.parseMt940FileString(mt940FileString);

    // Example validation: Ensure that all statements have transactions
    for (const statement of statements) {
      if (!statement.transactions || statement.transactions.length === 0) {
        throw new Error(`Statement with reference ${statement.referenceNumber} has no
transactions.`);
      }
    }

    console.log('Parsed Statements:', statements);
  } catch (error) {
    console.error('Error parsing MT940 file:', error.message);
  }
}

const mt940FileString = `
// your MT940 file content here
`;

parseMT940WithErrorHandling(mt940FileString);
```

Comprehensive Feature Demonstration

To give you a comprehensive view, here’s a complete application example demonstrating multiple features — reading from a file, combining multiple files, custom processing, error handling,

filtering, and extending the parser.

```
import { promises as fs } from 'fs';
import { Mt940Parser } from '@fin.cx/mt940parser';

class ComprehensiveMt940Parser extends Mt940Parser {
  async parseMt940FileString(fileString: string) {
    const statements = await super.parseMt940FileString(fileString);

    // Custom logic: Example, add a new property customField to each statement
    for (const statement of statements) {
      statement.customField = 'custom value';
    }

    return statements;
  }
}

async function parseAndProcessFiles(filePaths: string[], minAmount: number) {
  const parser = new ComprehensiveMt940Parser();
  const allStatements = [];

  for (const filePath of filePaths) {
    try {
      const fileContent = await fs.readFile(filePath, 'utf-8');
      const statements = await parser.parseMt940FileString(fileContent);
      allStatements.push(...statements);
    } catch (error) {
      console.error(`Error parsing file ${filePath}:`, error);
    }
  }

  const largeTransactions = [];

  for (const statement of allStatements) {
    for (const transaction of statement.transactions) {
      if (transaction.amount >= minAmount) {
        largeTransactions.push(transaction);
      }
    }
```

```
  }

  return { allStatements, largeTransactions };
}

const filePaths = ['./path/to/first.mt940', './path/to/second.mt940'];
const minAmount = 1000;

async function main() {
  try {
    const { allStatements, largeTransactions } = await parseAndProcessFiles(filePaths,
minAmount);
    console.log('All Statements:', allStatements);
    console.log('Large Transactions:', largeTransactions);
  } catch (error) {
    console.error('Error in processing:', error);
  }
}

main();
```

This example demonstrates a holistic approach to working with MT940 files using the `@fin.cx/mt940parser` library. It covers file reading, custom processing, error handling, filtering transactions, and extending the parser for additional functionality. This should give you a solid foundation to build upon and tailor the parser to your specific requirements. undefined

changelog.md for @fin.cx/mt940parser

2024-07-06 - 1.1.0 - feat(core)

Added new afterburner logic for specific bank reference

- Ensured proper accountId formatting for statements with referenceNumber 'BUNQ BV'
- Added support for parsing MT940 files from buffers and strings
- Enhanced testing suite with new test cases
- Included detailed usage examples in readme

2024-07-02 - 1.0.6 - fix(core)

Ensured proper accountId formatting for statements with referenceNumber 'BUNQ BV'

- Added condition to modify accountId format if referenceNumber is 'BUNQ BV' in parseMt940Buffer method

2023-11-15 - 1.0.5 - Minor Fixes

- Fixes in core components for stability

2023-11-15 - 1.0.4 - Minor Fixes

- Fixes in core components for stability

2023-11-15 - 1.0.3 - Minor Fixes

- Fixes in core components for stability

2023-11-15 - 1.0.2 - Minor Fixes

- Fixes in core components for stability

2022-12-06 - 1.0.2 - Minor Fixes

- Fixes in core components for stability