Introduction

BtcWalletLibrary is a C# library designed to simplify Bitcoin wallet development. It abstracts away all transaction-handling complexities, allowing developers to focus solely on UI customization. With BtcWalletLibrary, you don't need to manage address derivation, getting user's transaction history, or fund transfers—everything is handled for you. Just integrate the library, and your wallet is ready to go with minimal effort.

Key Features

- Automatic address and transaction management
- Built-in support for fetching transaction history
- Simplified fund transfers and fee estimation
- Event-driven architecture for real-time updates
- Customizable UI while handling all backend logic automatically

Key Principles

BtcWalletLibrary is built with the following key principles in mind:

- **Abstraction:** It abstracts away the complexities of Bitcoin protocol details and low-level cryptography, allowing developers to focus on the core logic of their wallet applications.
- **Modularity:** The library is designed with a modular architecture, making it easy to integrate specific components into your projects and customize functionality as needed.
- **Event-Driven:** The library leverages an event-driven architecture, enabling real-time updates and reactive UI development through events like transaction fetches, transaction broadcasts, confirmations, and balance changes.
- **Extensibility:** Designed to be extensible, the library allows developers to integrate with different Bitcoin network providers, storage mechanisms, and UI frameworks.
- **DI Container Agnostic:** BtcWalletLibrary adopts a dependency injection (DI) container-agnostic approach using Microsoft.Extensions.DependencyInjection. This ensures that developers can choose their preferred DI container without being locked into a specific implementation.
- **Type-Safety:** Built with C# and strong typing in mind, it provides a type-safe environment, reducing runtime errors and improving code maintainability.

Feature Details

BtcWalletLibrary offers a wide range of features to accelerate your Bitcoin wallet development:

- **Transaction History:** Fetches transaction history by querying addresses until the maximum number of empty addresses is reached (configured in the settings file). For improved user experience, addresses are queried in parallel so that newly found transactions or updated information (such as transaction confirmation status) are broadcast via events.

- **Address Management:** Generates and manages Bitcoin addresses (both main and change addresses).
- **Balance Tracking:** Tracks wallet balances and manages UTXOs.
- **Transaction Construction:** Provides tools for building and signing Bitcoin transactions, including coin selection and fee estimation.
- **Transaction Broadcasting:** Enables broadcasting transactions to the Bitcoin network.
- **Validation:** Ensures robust input validation (addresses, amounts, and selected UTXOs) to prevent errors during transaction creation.
- **Error Handling:** Implements well-defined error handling mechanisms using custom exception classes for specific Bitcoin-related issues.
- **Data Mapping:** Efficiently maps between NBitcoin library types and internal wallet models for seamless integration.
- **UTXO Management:** Supports managing and selecting Unspent Transaction Outputs (UTXOs).

Getting Started with BTCWalletLibrary

Prerequisites

- .NET Standard 2.0+ SDK
- Basic knowledge of Dependency Injection in .NET

Step 1: Install NuGet Package

Install the BTCWalletLibrary NuGet package in your .NET project:

```
dotnet add package BtcWalletLibrary
```

Step 2: Configure Settings

Create an `appsettings.json` file in your project root, example file:

```
{
  "CryptoWalletLibrary": {
    "NodeConfiguration": {
      "Url": "testnet.aranguren.org",
      "Port": 51001,
      "UseSSL": false,
      "Network": "TestNet",
      "MaxRangeEmptyAddr": 2
    },
    "BlockchainTxFeeApi": {
      "BaseUrl": "https://api.blockchain.info",
      "FeePath": "/mempool/fees"
    }
  }
}
```

Note: Set the file's **Copy to Output Directory** property to `Copy Always`.

Step 3: Initialize the Library

Configure the library using Microsoft.Extensions.DependencyInjection:

```
// Build configuration
var configuration = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
    .Build();

// Configure wallet library config
services.Configure<WalletConfig>(configuration.GetSection("CryptoWalletLibrary"));

// Setup dependency injection
var services = new ServiceCollection();

// Configure logging (Serilog shown here as example)
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Verbose()
    .WriteTo.Console()
    .CreateLogger();
services.AddSingleton<ILoggingService, SerilogLoggingService>();


// Initialize with test mnemonic for demo purposes (replace with secure key management
in production)
const string testMnemonic =
    "asthma attend bus original science leaf deputy nuclear ticket valley vacuum tornado";
services.AddBtcWalletLibraryServices(testMnemonic);

// Build service provider
var serviceProvider = services.BuildServiceProvider();
```

Demo App

Example usage in simple UI application:

link: https://github.com/ramazan199/BtcWalletUI

Implementation Notes

- The example uses:
  - `Microsoft.Extensions.DependencyInjection` for DI
  - `Serilog` for logging (optional)
  - .NET's built-in configuration system
- Works in any .NET Standard 2.0+ compatible project:
  - Console apps

- ASP.NET Core services
- Desktop applications
- Mobile apps (via .NET MAUI/Xamarin)
