Namespace BtcWalletLibrary

Classes

WalletInit

Sets up the Bitcoin wallet library services for dependency injection.

Class WalletInit

Namespace: BtcWalletLibrary

Assembly: BtcWalletLibrary.dll

Sets up the Bitcoin wallet library services for dependency injection.

```
public static class WalletInit
```

Inheritance

object ← WalletInit

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AddBtcWalletLibraryServices(IServiceCollection, string)

Adds Bitcoin services to service collection.

```
public static void AddBtcWalletLibraryServices(this IServiceCollection services,
string mnemonicWords)
```

Parameters

`services` IServiceCollection

`mnemonicWords` string

passphrase for wallet address generation

Namespace BtcWalletLibrary.Configuration

Classes

BlockchainTxFeeApiConfigSection

Represents the configuration section for the Blockchain transaction fee API.

NodeConfigSection

Represents the configuration section for the Bitcoin node.

WalletConfig

Represents the overall wallet configuration.

Enums

NetworkType

Enum representing Bitcoin network types.

Class BlockchainTxFeeApiConfigSection

Namespace: BtcWalletLibrary.Configuration

Assembly: BtcWalletLibrary.dll

Represents the configuration section for the Blockchain transaction fee API.

```
public class BlockchainTxFeeApiConfigSection
```

Inheritance

object ← BlockchainTxFeeApiConfigSection

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

BaseUrl

Base URL of the Blockchain transaction fee API.

```
public string BaseUrl { get; set; }
```

Property Value

string

FeePath

Path to the fee endpoint in the Blockchain transaction fee API.

```
public string FeePath { get; set; }
```

Property Value

string

Enum NetworkType

Namespace: BtcWalletLibrary.Configuration

Assembly: BtcWalletLibrary.dll

Enum representing Bitcoin network types.

```
public enum NetworkType
```

Fields

`Main = 0`

`RegTest = 2`

`TestNet = 1`

Class NodeConfigSection

Namespace: BtcWalletLibrary.Configuration

Assembly: BtcWalletLibrary.dll

Represents the configuration section for the Bitcoin node.

```
public class NodeConfigSection
```

Inheritance

object ← NodeConfigSection

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

MaxRangeEmptyAddr

Maximum range of empty addresses to scan during address discovery.

```
public int MaxRangeEmptyAddr { get; set; }
```

Property Value

int

Network

Bitcoin network type (Main, TestNet, RegTest).

```
public NetworkType Network { get; set; }
```

Property Value

NetworkType

Port

Port number of the Bitcoin node.

```
public int Port { get; set; }
```

Property Value

int

Url

URL of the Bitcoin node.

```
public string Url { get; set; }
```

Property Value

string

UseSsl

Indicates whether to use SSL for connecting to the Bitcoin node.

```
public bool UseSsl { get; set; }
```

Property Value

bool

Class WalletConfig

Namespace: BtcWalletLibrary.Configuration

Assembly: BtcWalletLibrary.dll

Represents the overall wallet configuration.

```
public class WalletConfig
```

Inheritance

object ← WalletConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

BlockchainTxFeeApi

Configuration section for the Blockchain transaction fee API.

```
public BlockchainTxFeeApiConfigSection BlockchainTxFeeApi { get; set; }
```

Property Value

BlockchainTxFeeApiConfigSection

NodeConfiguration

Configuration section for the Bitcoin node.

```
public NodeConfigSection NodeConfiguration { get; set; }
```

Property Value

NodeConfigSection

Namespace BtcWalletLibrary.DTOs.Responses

Classes

TransactionFetchResult

Represents the result of a wallet transaction fetching operation. Fetched transactions are published individually via the EventDispatcher (since they are processed in parallel). To access complete transaction lists, use the TxHistoryService's Transactions property.

TransferResult

Represents the result of a transfer operation after sending bitcoins to a recipient.

TxFeeResult

Represents the result of a transaction fee retrieval operation. This class encapsulates the outcome of attempting to fetch or calculate a transaction fee, including success status, whether a default fee was used, the fee amount, and any error messages.

Class TransactionFetchResult

Namespace: BtcWalletLibrary.DTOs.Responses

Assembly: BtcWalletLibrary.dll

Represents the result of a wallet transaction fetching operation. Fetched transactions are published individually via the EventDispatcher (since they are processed in parallel). To access complete transaction lists, use the TxHistoryService's Transactions property.

```
public class TransactionFetchResult
```

Inheritance

object ← TransactionFetchResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

HasNetworkErrors

Indicates whether network errors occurred during the transaction fetch operation.

```
public bool HasNetworkErrors { get; }
```

Property Value

bool

Class TransferResult

Namespace: BtcWalletLibrary.DTOs.Responses

Assembly: BtcWalletLibrary.dll

Represents the result of a transfer operation after sending bitcoins to a recipient.

```
public class TransferResult
```

Inheritance

object ← TransferResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

OperationError

Error information in case the transfer operation failed. Null if the transfer was successful. If not null, this property will contain a TransferError object providing details about the error.

```
public TransferError OperationError { get; }
```

Property Value

TransferError

Success

Indicates whether the transfer operation was successful.

```
public bool Success { get; }
```

Property Value

bool

TransactionId

The transaction ID of the transfer, if successful. Null or empty if the transfer failed.

```
public string TransactionId { get; }
```

Property Value

string

Class TxFeeResult

Namespace: BtcWalletLibrary.DTOs.Responses

Assembly: BtcWalletLibrary.dll

Represents the result of a transaction fee retrieval operation. This class encapsulates the outcome of attempting to fetch or calculate a transaction fee, including success status, whether a default fee was used, the fee amount, and any error messages.

```
public class TxFeeResult
```

Inheritance

object ← TxFeeResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Fee

Gets or internal sets the transaction fee amount. This property contains the recommended or calculated transaction fee as a NBitcoin.Money object. The fee unit is typically in satoshis.

```
public Money Fee { get; }
```

Property Value

Money

IsDefault

Gets or internal sets a value indicating whether a default fee value was used. True if a default or fallback fee was used (typically when network fee retrieval fails), false if a network-derived fee was used.

```
public bool IsDefault { get; }
```

Property Value

bool

IsSuccess

Gets or internal sets a value indicating whether the network fee retrieval operation was successful. True if the fee was successfully fetched from the network, false otherwise (e.g., due to network errors).

```
public bool IsSuccess { get; }
```

Property Value

bool

OperationError

Gets or internal sets the error information in case the fee retrieval operation was not successful (Is Success is false). Null if the operation was successful or if no specific error occurred. If not null, this property will contain a TransactionFeeError object providing details about the error.

```
public TransactionFeeError OperationError { get; }
```

Property Value

TransactionFeeError

Namespace BtcWalletLibrary.Events.Arguments

Classes

AddressTxsFetchedEventArgs

Event arguments for when transaction fetching for a specific address is completed. This event is published via the IEventDispatcher after the wallet has finished fetching transaction history for a single Bitcoin address. Note that addresses are queried individually for their transaction history, such as during wallet synchronization or initial address discovery.

BalanceUpdatedEventArgs

Event arguments for when the wallet's balance is updated. This event is published via the IEvent Dispatcher **after the wallet's balance has been recalculated.** Balance recalculation is triggered by processing transaction fetching events, such as AddressTxsFetchedEventArgs. The event is published by services like the IBalanceService when it has updated the wallet's balance based on new transaction data. Subscribers can listen for this event to update their UI or application state to reflect the latest confirmed and unconfirmed wallet balances.

FetchingCompletedEventArgs

Event arguments for when a fetching process of all addresses inside wallet is completed. This event is published via the IEventDispatcher to signal the completion of a data fetching / synchronizing operation of the wallet transactions. Subscribers can listen for this event to be notified when a fetching process has finished. !!! Note that you need to wait for this event to be raised before sending a new transaction, to ensure that the wallet is up-to-date.

FetchingStartedEventArgs

Event arguments for when a process of fetching all addresses inside wallet is completed. This event is published via the IEventDispatcher to signal the beginning of a data fetching / synchronizing operation of the wallet transactions. Subscribers can listen for this event to be notified when a fetching process commences.

NewAddressesFoundEventArgs

Event arguments base class for events indicating that new addresses have been found (derived and added to the wallet's address lists). This is an abstract base class; concrete implementations like New MainAddressesFoundEventArgs and NewChangeAddressesFoundEventArgs are used for specific address types. These events are published via the IEventDispatcher

NewChangeAddressesFoundEventArgs

Event arguments for when new change addresses have been found and added to the wallet. This event is published via the IEventDispatcher during initial address discovery or gap limit handling. Subscribers can listen for this event to be notified when the wallet expands its set of available change addresses.

NewMainAddressesFoundEventArgs

Event arguments for when new main (receiving) addresses have been found and added to the wallet. This event is published via the IEventDispatcher during initial address discovery or gap limit handling. Subscribers can listen for this event to be notified when the wallet expands its set of available main addresses.

TransactionAddedEventArgs

Event arguments for when a new Bitcoin transaction is added to the wallet's transaction history. This event is published via the IEventDispatcher after a new transaction is discovered or synchronized with the blockchain and added to the local storage. Subscribers can listen for this event to be notified of newly added transactions, enabling real-time updates to the user interface or application logic.

TransactionBroadcastedEventArgs

Event arguments for when a transaction is successfully broadcasted to the Bitcoin network. This event is published via the IEventDispatcher after the wallet has successfully submitted a transaction to the Bitcoin network for broadcasting. Subscribers can listen for this event (to update UI e.g.) to be notified when a transaction initiated by the wallet is successfully sent.

TransactionConfirmedEventArgs

Event arguments for when a transaction is considered confirmed on the Bitcoin network. This event is published via the IEventDispatcher when a transaction, relevant to the wallet, reaches a certain confirmation threshold on the blockchain. Subscribers can listen for this event(e.g. to Update UI) to be notified when a transaction's status changes to confirmed, indicating a higher degree of security and finality.

TransactionDateUpdatedEventArgs

Event arguments for when the date of a transaction is updated in the wallet's transaction history. This event is published via the IEventDispatcher when the date associated with a transaction is updated due to synchronization with the blockchain. Subscribers can listen for this event to update transaction displays or time-sensitive logic based on the corrected transaction date.

TxInputAddrMarkedAsUserAddrEventArgs

Event arguments for when an input address in a transaction is identified as belonging to the user's wallet addresses. This event is published via the IEventDispatcher when the wallet's services determine that an input address of a specific transaction corresponds to an address managed by this wallet.

UI Usage: User interfaces can subscribe to this event to:

- Identify transactions where funds are being sent from the user's wallet.
- Update transaction displays to visually distinguish outgoing transactions based on user-owned input addresses.
- Potentially calculate and display the amount sent in outgoing transactions.

Subscribers can listen for this event to track and react to transactions where the wallet's addresses are used as inputs, indicating funds are being spent from the wallet.

TxOutputAddrMarkedAsUserAddrEventArgs

Event arguments for when an output address in a transaction is identified as belonging to the user's wallet addresses. This event is published via the IEventDispatcher when the wallet's services determine that an output address of a specific transaction corresponds to an address managed by this wallet.

UI Usage: User interfaces can subscribe to this event to:

- Identify transactions where funds are being sent to the user's wallet (incoming transactions).
- Update transaction displays to visually distinguish incoming transactions based on user-owned output addresses.
- Calculate and display the amount received in incoming transactions.

Subscribers can listen for this event to track and react to transactions where the wallet's addresses are used as outputs, indicating funds are being received by the wallet.

Class AddressTxsFetchedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when transaction fetching for a specific address is completed. This event is published via the IEventDispatcher after the wallet has finished fetching transaction history for a single Bitcoin address. Note that addresses are queried individually for their transaction history, such as during wallet synchronization or initial address discovery.

```
public class AddressTxsFetchedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← AddressTxsFetchedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Address

Gets the Bitcoin address for which transactions were fetched.

```
public string Address { get; }
```

Property Value

string

Transactions

Gets the list of transactions that were fetched for the specified address.

```
public List<Transaction> Transactions { get; }
```

Property Value

List <Transaction>

Class BalanceUpdatedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when the wallet's balance is updated. This event is published via the IEvent Dispatcher **after the wallet's balance has been recalculated.** Balance recalculation is triggered by processing transaction fetching events, such as AddressTxsFetchedEventArgs. The event is published by services like the IBalanceService when it has updated the wallet's balance based on new transaction data. Subscribers can listen for this event to update their UI or application state to reflect the latest confirmed and unconfirmed wallet balances.

```
public class BalanceUpdatedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← BalanceUpdatedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

NewConfirmedBalance

Gets the new confirmed balance of the wallet.

```
public double NewConfirmedBalance { get; }
```

Property Value

double

NewUnconfirmedBalance

Gets the new unconfirmed balance of the wallet.

```
public double NewUnconfirmedBalance { get; }
```

Property Value

double

Class FetchingCompletedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when a fetching process of all addresses inside wallet is completed. This event is published via the IEventDispatcher to signal the completion of a data fetching / synchronizing operation of the wallet transactions. Subscribers can listen for this event to be notified when a fetching process has finished. !!! Note that you need to wait for this event to be raised before sending a new transaction, to ensure that the wallet is up-to-date.

```
public class FetchingCompletedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← FetchingCompletedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class FetchingStartedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when a process of fetching all addresses inside wallet is completed. This event is published via the IEventDispatcher to signal the beginning of a data fetching / synchronizing operation of the wallet transactions. Subscribers can listen for this event to be notified when a fetching process commences.

```
public class FetchingStartedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← FetchingStartedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class NewAddressesFoundEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments base class for events indicating that new addresses have been found (derived and added to the wallet's address lists). This is an abstract base class; concrete implementations like NewMainAddressesFoundEventArgs and NewChangeAddressesFoundEventArgs are used for specific address types. These events are published via the IEventDispatcher

```
public abstract class NewAddressesFoundEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← NewAddressesFoundEventArgs

Derived

NewChangeAddressesFoundEventArgs, NewMainAddressesFoundEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NewAddressesFoundEventArgs(uint)

```
protected NewAddressesFoundEventArgs(uint newIdx)
```

Parameters

`newIdx` uint

Properties

NewLastAddrIdx

Gets the index of the last newly found address in the respective address list (main or change). This index typically represents the highest index that has been derived and added during the address discovery process.

```
public uint NewLastAddrIdx { get; }
```

Property Value

uint

Class NewChangeAddressesFoundEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when new change addresses have been found and added to the wallet. This event is published via the IEventDispatcher during initial address discovery or gap limit handling. Subscribers can listen for this event to be notified when the wallet expands its set of available change addresses.

```
public class NewChangeAddressesFoundEventArgs : NewAddressesFoundEventArgs
```

Inheritance

object ← EventArgs ← NewAddressesFoundEventArgs ← NewChangeAddressesFoundEventArgs

Inherited Members

NewAddressesFoundEventArgs.NewLastAddrIdx , EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class NewMainAddressesFoundEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when new main (receiving) addresses have been found and added to the wallet. This event is published via the IEventDispatcher during initial address discovery or gap limit handling. Subscribers can listen for this event to be notified when the wallet expands its set of available main addresses.

```
public class NewMainAddressesFoundEventArgs : NewAddressesFoundEventArgs
```

Inheritance

object ← EventArgs ← NewAddressesFoundEventArgs ← NewMainAddressesFoundEventArgs

Inherited Members

NewAddressesFoundEventArgs.NewLastAddrIdx , EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class TransactionAddedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when a new Bitcoin transaction is added to the wallet's transaction history. This event is published via the IEventDispatcher after a new transaction is discovered or synchronized with the blockchain and added to the local storage. Subscribers can listen for this event to be notified of newly added transactions, enabling real-time updates to the user interface or application logic.

```
public class TransactionAddedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← TransactionAddedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TransactionAddedEventArgs(Transaction)

Initializes a new instance of the TransactionAddedEventArgs class.

```
public TransactionAddedEventArgs(Transaction bitcoinTransaction)
```

Parameters

`bitcoinTransaction` Transaction

The Transaction object representing the newly added Bitcoin transaction.

Properties

BitcoinTransaction

Gets the newly added Bitcoin transaction information.

```
public Transaction BitcoinTransaction { get; }
```

Property Value

Transaction

Class TransactionBroadcastedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when a transaction is successfully broadcasted to the Bitcoin network. This event is published via the IEventDispatcher after the wallet has successfully submitted a transaction to the Bitcoin network for broadcasting. Subscribers can listen for this event (to update UI e.g.) to be notified when a transaction initiated by the wallet is successfully sent.

```
public class TransactionBroadcastedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← TransactionBroadcastedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

TransactionForStorage

Gets the TransactionForStorage object representing the transaction that was broadcasted. This object contains details of the broadcasted transaction, including its transaction ID and other relevant information.

```
public Transaction TransactionForStorage { get; }
```

Property Value

Transaction

Class TransactionConfirmedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when a transaction is considered confirmed on the Bitcoin network. This event is published via the IEventDispatcher when a transaction, relevant to the wallet, reaches a certain confirmation threshold on the blockchain. Subscribers can listen for this event(e.g. to Update UI) to be notified when a transaction's status changes to confirmed, indicating a higher degree of security and finality.

```
public class TransactionConfirmedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← TransactionConfirmedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TransactionConfirmedEventArgs(string)

```
public TransactionConfirmedEventArgs(string txId)
```

Parameters

`txId` string

Properties

TxId

Gets the transaction ID (TxId) of the confirmed transaction.

```
public string TxId { get; }
```

Property Value

string

Class TransactionDateUpdatedEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when the date of a transaction is updated in the wallet's transaction history. This event is published via the IEventDispatcher when the date associated with a transaction is updated due to synchronization with the blockchain. Subscribers can listen for this event to update transaction displays or time-sensitive logic based on the corrected transaction date.

```
public class TransactionDateUpdatedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← TransactionDateUpdatedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TransactionDateUpdatedEventArgs(string, DateTime)

```
public TransactionDateUpdatedEventArgs(string txId, DateTime date)
```

Parameters

`txId` string

`date` DateTime

Properties

Date

Gets the new DateTime representing the updated date of the transaction.

```
public DateTime Date { get; }
```

Property Value

DateTime

TxId

Gets the transaction ID (TxId) of the transaction that has its date updated.

```
public string TxId { get; }
```

Property Value

string

Class TxInputAddrMarkedAsUserAddrEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when an input address in a transaction is identified as belonging to the user's wallet addresses. This event is published via the IEventDispatcher when the wallet's services determine that an input address of a specific transaction corresponds to an address managed by this wallet.

UI Usage: User interfaces can subscribe to this event to:

- Identify transactions where funds are being sent from the user's wallet.
- Update transaction displays to visually distinguish outgoing transactions based on user-owned input addresses.
- Potentially calculate and display the amount sent in outgoing transactions.

Subscribers can listen for this event to track and react to transactions where the wallet's addresses are used as inputs, indicating funds are being spent from the wallet.

```
public class TxInputAddrMarkedAsUserAddrEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← TxInputAddrMarkedAsUserAddrEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Input

Gets the TransactionInput object representing the input that was marked as a user address. This object provides details about the transaction input, including the input address and related information.

```
public TransactionInput Input { get; }
```

Property Value

TransactionInput

TxId

Gets the transaction ID (TxId) of the transaction containing the input address that was marked as a user address.

```
public string TxId { get; }
```

Property Value

string

Class TxOutputAddrMarkedAsUserAddrEventArgs

Namespace: BtcWalletLibrary.Events.Arguments

Assembly: BtcWalletLibrary.dll

Event arguments for when an output address in a transaction is identified as belonging to the user's wallet addresses. This event is published via the IEventDispatcher when the wallet's services determine that an output address of a specific transaction corresponds to an address managed by this wallet.

UI Usage: User interfaces can subscribe to this event to:

- Identify transactions where funds are being sent to the user's wallet (incoming transactions).
- Update transaction displays to visually distinguish incoming transactions based on user-owned output addresses.
- Calculate and display the amount received in incoming transactions.

Subscribers can listen for this event to track and react to transactions where the wallet's addresses are used as outputs, indicating funds are being received by the wallet.

```
public class TxOutputAddrMarkedAsUserAddrEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← TxOutputAddrMarkedAsUserAddrEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Output

Gets the TransactionOutput object representing the output that was marked as a user address. This object provides details about the transaction output, including the output address, value, and related information.

```
public TransactionOutput Output { get; }
```

Property Value

TransactionOutput

TxId

Gets the transaction ID (TxId) of the transaction containing the output address that was marked as a user address.

```
public string TxId { get; }
```

Property Value

string

Namespace BtcWalletLibrary.Exceptions

Classes

OperationError

Base class for representing operation error messages within the BtcWalletLibrary. This class provides a common structure for encapsulating error details as objects, allowing for more type-safe and organized error handling compared to using plain strings for error messages. Derived classes, such as TransactionFeeError and TransferError, can inherit from this class to represent specific types of errors with potentially additional properties in the future.

TransactionBuildError

Represents an error message specific to Bitcoin transaction building operations. This class inherits from OperationError and is used to provide detailed error information when the process of building a Bitcoin transaction (e.g., coin selection, output creation, fee calculation) fails. Instances of this class are used to indicate specific transaction building errors and can be used to provide user-friendly error messages or for logging and debugging purposes.

TransactionFeeError

Represents an error message specific to transaction fee retrieval operations. This class inherits from OperationError and is used to provide more specific error details when fetching or calculating transaction fees fails. Instances of this class are returned within the TxFeeResult class, specifically in its OperationError property, when a fee retrieval operation is unsuccessful.

TransferError

Represents an error message specific to Bitcoin transfer operations. This class inherits from Operation Error and is used to provide detailed error information when a Bitcoin transfer (sending funds) fails. Instances of this class are returned within the TransferResult class, specifically in its OperationError property, when a transfer operation is unsuccessful.

Enums

TransactionBuildErrorCode

Enumeration of error codes related to transaction building process.

Class OperationError

Namespace: BtcWalletLibrary.Exceptions

Assembly: BtcWalletLibrary.dll

Base class for representing operation error messages within the BtcWalletLibrary. This class provides a common structure for encapsulating error details as objects, allowing for more type-safe and organized error handling compared to using plain strings for error messages. Derived classes, such as TransactionFeeError and TransferError, can inherit from this class to represent specific types of errors with potentially additional properties in the future.

```
public class OperationError
```

Inheritance

object ← OperationError

Derived

TransactionBuildError, TransactionFeeError, TransferError

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Message

Gets the error message string. This property contains the human-readable description of the error that occurred.

```
public string Message { get; }
```

Property Value

string

Class TransactionBuildError

Namespace: BtcWalletLibrary.Exceptions

Assembly: BtcWalletLibrary.dll

Represents an error message specific to Bitcoin transaction building operations. This class inherits from OperationError and is used to provide detailed error information when the process of building a Bitcoin transaction (e.g., coin selection, output creation, fee calculation) fails. Instances of this class are used to indicate specific transaction building errors and can be used to provide user-friendly error messages or for logging and debugging purposes.

```
public class TransactionBuildError : OperationError
```

Inheritance

object ← OperationError ← TransactionBuildError

Inherited Members

OperationError.Message , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TransactionBuildError(string)

```
public TransactionBuildError(string message)
```

Parameters

`message` string

Enum TransactionBuildErrorCode

Namespace: BtcWalletLibrary.Exceptions

Assembly: BtcWalletLibrary.dll

Enumeration of error codes related to transaction building process.

```
public enum TransactionBuildErrorCode
```

Fields

`InsufficientFunds = 1005`

Error code indicating insufficient funds to build the transaction.

`InvalidAddress = 1003`

Error code indicating an invalid Bitcoin address format.

`InvalidAmount = 1001`

Error code indicating an invalid transaction amount.

`InvalidCustomFee = 1002`

Error code indicating an invalid custom fee value.

`NoUtxosSelected = 1004`

Error code indicating that no Unspent Transaction Outputs (UTXOs) were selected for the transaction.

`None = 0`

No error. Default value indicating no specific transaction build error.

`TransactionBuildFailed = 1006`

Error code indicating a general failure during transaction building process.

Class TransactionFeeError

Namespace: BtcWalletLibrary.Exceptions

Assembly: BtcWalletLibrary.dll

Represents an error message specific to transaction fee retrieval operations. This class inherits from OperationError and is used to provide more specific error details when fetching or calculating transaction fees fails. Instances of this class are returned within the TxFeeResult class, specifically in its OperationError property, when a fee retrieval operation is unsuccessful.

```
public class TransactionFeeError : OperationError
```

Inheritance

object ← OperationError ← TransactionFeeError

Inherited Members

OperationError.Message , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class TransferError

Namespace: BtcWalletLibrary.Exceptions

Assembly: BtcWalletLibrary.dll

Represents an error message specific to Bitcoin transfer operations. This class inherits from OperationError and is used to provide detailed error information when a Bitcoin transfer (sending funds) fails. Instances of this class are returned within the TransferResult class, specifically in its OperationError property, when a transfer operation is unsuccessful.

```
public class TransferError : OperationError
```

Inheritance

object ← OperationError ← TransferError

Inherited Members

OperationError.Message , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Namespace BtcWalletLibrary.Interfaces

Interfaces

IAddressService

Interface for services that manage Bitcoin addresses within the wallet.

IBalanceService

Interface for services that manages and provides wallet balance information.

ICommonService

Interface for common wallet services providing access to shared configurations and resources.

IEventDispatcher

Interface for an event dispatcher, enabling loosely coupled communication between components. This interface allows for subscribing/unsubscribing to specific event types in your ViewModels or other classes.

ILoggingService

ITransferService

Interface for services responsible for transferring Bitcoin, specifically broadcasting transactions to the network.

ITxBuilderService

Interface for services responsible for building Bitcoin transactions. This service handles the logic of constructing a transaction, including coin selection, fee calculation, and output creation.

ITxFeeService

Interface for services responsible for calculating Bitcoin transaction fees. This service provides functionalities to estimate transaction fees based on different parameters and retrieve fee recommendations.

ITxHistoryService

Interface for services responsible for managing and synchronizing the wallet's transaction history. This service provides access to the list of transactions and functionality to synchronize transaction history with the blockchain.

ITxMapper

Interface for mapping Bitcoin transaction data between different formats and models. This interface defines methods for converting transaction representations from external libraries (like NBitcoin) into the library's internal models optimized for storage and processing (Transaction) and for working with UTXOs. Implementations of this interface are responsible for handling the data transformation logic.

ITxValidator

Interface for validating various aspects of Bitcoin transactions before they are built or broadcasted. This interface defines a set of methods to perform common validation checks on transaction-related data, such as recipient addresses, amounts, fees, and the sufficiency of funds. Implementations of this interface are responsible for enforcing transaction validity rules and providing specific error codes via the TransactionBuildErrorCode enum when validation fails.

Interface IAddressService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for services that manage Bitcoin addresses within the wallet.

```
public interface IAddressService
```

Properties

ChangeAddress

Gets the current change address.

```
BitcoinAddress ChangeAddress { get; }
```

Property Value

BitcoinAddress

ChangeAddresses

Gets the list of all derived change addresses.

```
IReadOnlyList<BitcoinAddress> ChangeAddresses { get; }
```

Property Value

IReadOnlyList<BitcoinAddress>

LastChangeAddrIdx

Gets the index of the last derived change address.

```
int LastChangeAddrIdx { get; }
```

Property Value

int

LastMainAddrIdx

Gets the index of the last derived main address.

```
int LastMainAddrIdx { get; }
```

Property Value

int

MainAddress

Gets the current main (receiving) address.

```
BitcoinAddress MainAddress { get; }
```

Property Value

BitcoinAddress

MainAddresses

Gets the list of all derived main (receiving) addresses.

```
IReadOnlyList<BitcoinAddress> MainAddresses { get; }
```

Property Value

IReadOnlyList<BitcoinAddress>

QueriedChangeAddresses

Gets the list of change addresses that have been queried from the blockchain.

```
IReadOnlyList<BitcoinAddress> QueriedChangeAddresses { get; }
```

Property Value

IReadOnlyList<BitcoinAddress>

QueriedMainAddresses

Gets the list of main addresses that have been queried from the blockchain.

```
IReadOnlyList<BitcoinAddress> QueriedMainAddresses { get; }
```

Property Value

IReadOnlyList<BitcoinAddress>

Methods

AddAddressToQueriedChangeAddrLst(BitcoinAddress)

Adds a change address to the list of queried change addresses.

```
void AddAddressToQueriedChangeAddrLst(BitcoinAddress address)
```

Parameters

`address` BitcoinAddress

The Bitcoin address to add to the queried change addresses list.

AddAddressToQueriedMainAddrLst(BitcoinAddress)

Adds a main address to the list of queried main addresses.

```
void AddAddressToQueriedMainAddrLst(BitcoinAddress address)
```

Parameters

`address` BitcoinAddress

The Bitcoin address to add to the queried main addresses list.

DerivChangeAddr(uint)

Derives a change address at a specific index.

```
BitcoinAddress DerivChangeAddr(uint addrIdx)
```

Parameters

`addrIdx` uint

The address index to derive.

Returns

BitcoinAddress

The derived Bitcoin change address.

DeriveMainAddr(uint)

Derives a main (receiving) address at a specific index.

```
BitcoinAddress DeriveMainAddr(uint addrIdx)
```

Parameters

`addrIdx` uint

The address index to derive.

Returns

BitcoinAddress

The derived Bitcoin main address.

DeriveNewChangeAddr()

Derives a new change address.

```
BitcoinAddress DeriveNewChangeAddr()
```

Returns

BitcoinAddress

The newly derived Bitcoin change address.

DeriveNewMainAddr()

Derives a new main (receiving) address.

```
BitcoinAddress DeriveNewMainAddr()
```

Returns

BitcoinAddress

The newly derived Bitcoin main address.

IsInQueiredAddresses(BitcoinAddress)

Checks if a given Bitcoin address is in the list of queried addresses (main or change).

```
bool IsInQueiredAddresses(BitcoinAddress bitcoinAddress)
```

Parameters

`bitcoinAddress` BitcoinAddress

The Bitcoin address to check.

Returns

bool

True if the address is in the queried addresses list, otherwise false.

Interface IBalanceService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for services that manages and provides wallet balance information.

```
public interface IBalanceService
```

Properties

TotalConfirmedBalance

Gets the total confirmed balance of the wallet across all addresses.

```
double TotalConfirmedBalance { get; }
```

Property Value

double

TotalUncnfirmedBalance

Gets the total unconfirmed balance of the wallet across all addresses.

```
double TotalUncnfirmedBalance { get; }
```

Property Value

double

Utxos

Gets a list of all Unspent Transaction Outputs (UTXOs) for the wallet, aggregated from all addresses.

```
IEnumerable<UtxoDetailsElectrumx> Utxos { get; }
```

Property Value

IEnumerable<UtxoDetailsElectrumx>

UtxosPerAddress

Gets a dictionary of Unspent Transaction Outputs (UTXOs) grouped by address. The key is the Bitcoin address, and the value is a list of UtxoDetailsElectrumx for that address.

```
IReadOnlyDictionary<string, IReadOnlyList<UtxoDetailsElectrumx>> UtxosPerAddress { get; }
```

Property Value

IReadOnlyDictionary<string, IReadOnlyList<UtxoDetailsElectrumx>>

Interface ICommonService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for common wallet services providing access to shared configurations and resources.

```
public interface ICommonService
```

Properties

BitcoinNetwork

Gets the Bitcoin network the wallet is configured to operate on (e.g., Mainnet, Testnet).

```
Network BitcoinNetwork { get; }
```

Property Value

Network

ChangeAddressesParentExtKey

Gets the extended private key used as the parent for deriving change addresses.

```
ExtKey ChangeAddressesParentExtKey { get; }
```

Property Value

ExtKey

ChangeAddressesParentExtPubKey

Gets the extended public key derived from the change addresses parent extended private key. This public key can be used for watch-only wallets or sharing purposes.

```
ExtPubKey ChangeAddressesParentExtPubKey { get; }
```

Property Value

ExtPubKey

MainAddressesParentExtKey

Gets the extended private key used as the parent for deriving main (receiving) addresses.

```
ExtKey MainAddressesParentExtKey { get; }
```

Property Value

ExtKey

MainAddressesParentExtPubKey

Gets the extended public key derived from the main addresses parent extended private key. This public key can be used for watch-only wallets or sharing purposes.

```
ExtPubKey MainAddressesParentExtPubKey { get; }
```

Property Value

ExtPubKey

MaxEmptyAddrRange

Gets the maximum range of consecutive empty addresses to check during address discovery. This is used to determine when to stop searching for used addresses.

```
int MaxEmptyAddrRange { get; }
```

Property Value

int

Interface IEventDispatcher

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for an event dispatcher, enabling loosely coupled communication between components. This interface allows for subscribing/unsubscribing to specific event types in your ViewModels or other classes.

```
public interface IEventDispatcher
```

Methods

Publish<TEventArgs>(object, TEventArgs)

Publishes an event of type TEventArgs to all subscribers.

```
void Publish<TEventArgs>(object sender, TEventArgs eventData) where TEventArgs : EventArgs
```

Parameters

`sender` object

The object that is raising the event.

`eventData` TEventArgs

The event arguments to be passed to subscribers.

Type Parameters

`TEventArgs`

The type of the event arguments, must inherit from EventArgs.

Exceptions

ArgumentNullException

Thrown if eventData is null.

Subscribe<TEventArgs>(EventHandler<TEventArgs>)

Subscribes a handler to events of type TEventArgs.

```
void Subscribe<TEventArgs>(EventHandler<TEventArgs> handler) where TEventArgs : EventArgs
```

Parameters

`handler` EventHandler<TEventArgs>

The event handler delegate to be invoked when an event of type TEventArgs is published.

Type Parameters

`TEventArgs`

The type of the event arguments to subscribe to, must inherit from EventArgs.

Unsubscribe<TEventArgs>(EventHandler<TEventArgs>)

Unsubscribes a handler from events of type TEventArgs.

```
void Unsubscribe<TEventArgs>(EventHandler<TEventArgs> handler) where TEventArgs : EventArgs
```

Parameters

`handler` EventHandler<TEventArgs>

The event handler delegate to be removed from the subscribers list for events of type TEventArgs.

Type Parameters

`TEventArgs`

The type of the event arguments to unsubscribe from, must inherit from EventArgs.

Interface ILoggingService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

```
public interface ILoggingService
```

Methods

LogError(Exception, string)

```
void LogError(Exception ex, string message)
```

Parameters

`ex` Exception

`message` string

LogInformation(string)

```
void LogInformation(string message)
```

Parameters

`message` string

LogWarning(string)

```
void LogWarning(string message)
```

Parameters

`message` string

Interface ITransferService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for services responsible for transferring Bitcoin, specifically broadcasting transactions to the network.

```
public interface ITransferService
```

Methods

BroadcastTransactionAsync(Transaction)

Broadcasts a Bitcoin transaction to the network asynchronously.

```
Task<TransferResult> BroadcastTransactionAsync(Transaction tx)
```

Parameters

`tx` Transaction

The NBitcoin.Transaction to broadcast.

Returns

Task<TransferResult>

A Task<TResult> representing the asynchronous operation. The task result contains a TransferResult indicating the success or failure of the broadcast. Potential failures could be due to network issues or transaction rejection by the network.

Interface ITxBuilderService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for services responsible for building Bitcoin transactions. This service handles the logic of constructing a transaction, including coin selection, fee calculation, and output creation.

```
public interface ITxBuilderService
```

Methods

TryBuildTx(Money, string, out Transaction, out Money, out List<Coin>, out TransactionBuildErrorCode, List<UnspentCoin>, Money)

Attempts to build a Bitcoin transaction.

```
bool TryBuildTx(Money amount, string destinationAddr, out Transaction tx, out Money
calculatedFee, out List<Coin> autoSelectedCoins, out TransactionBuildErrorCode
txBuildErrorCode, List<UnspentCoin> selectedUnspentCoins = null, Money customFee = null)
```

Parameters

`amount` Money

The amount to send in the transaction.

`destinationAddr` string

The destination Bitcoin address for the transaction.

`tx` Transaction

When this method returns successfully, contains the built NBitcoin.Transaction.

`calculatedFee` Money

When this method returns successfully, contains the calculated transaction fee.

`autoSelectedCoins` List<Coin>

When this method returns successfully, contains the list of coins that were automatically selected for the transaction if no specific coins were provided.

`txBuildErrorCode` TransactionBuildErrorCode

When this method returns false, contains the TransactionBuildErrorCode indicating the reason for transaction building failure.

`selectedUnspentCoins` List<UnspentCoin>

Optional. A list of specific UnspentCoin to use as inputs for the transaction. If null, coins will be automatically selected.

`customFee` Money

Optional. A custom fee to use for the transaction. If null, the fee will be automatically calculated.

Returns

bool

True if the transaction was built successfully, false otherwise. Check `txBuildErrorCode` for details on failure.

Interface ITxFeeService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for services responsible for calculating Bitcoin transaction fees. This service provides functionalities to estimate transaction fees based on different parameters and retrieve fee recommendations.

```
public interface ITxFeeService
```

Methods

CalculateTransactionFee(List<Coin>, List<Key>, IDestination, Money)

Calculates the transaction fee for a given set of inputs and outputs.

```
Money CalculateTransactionFee(List<Coin> selectedUnspentCoins, List<Key> signingKeys,
IDestination destinationAddress, Money amount)
```

Parameters

`selectedUnspentCoins` List<Coin>

List of NBitcoin.Coin coins selected as inputs for the transaction.

`signingKeys` List<Key>

List of NBitcoin.Key keys corresponding to the input coins, used for fee calculation based on signature size.

`destinationAddress` IDestination

The NBitcoin.IDestination address to which the funds are being sent. Used for output size estimation.

`amount` Money

The amount of Bitcoin to send. Used to determine the output value.

Returns

Money

A NBitcoin.Money object representing the calculated transaction fee.

GetRecommendedBitFeeAsync()

Asynchronously retrieves the recommended transaction fee rate from an external source.

```
Task<TxFeeResult> GetRecommendedBitFeeAsync()
```

Returns

Task <TxFeeResult>

A Task<TResult> representing the asynchronous operation. The task result contains a TxFeeResult object with recommended fee rates in satoshis per byte. This method might involve network requests and could potentially fail, resulting in a failed task or exceptions encapsulated in the TxFeeResult.

Interface ITxHistoryService

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for services responsible for managing and synchronizing the wallet's transaction history. This service provides access to the list of transactions and functionality to synchronize transaction history with the blockchain.

```
public interface ITxHistoryService
```

Properties

Transactions

Gets the list of transactions currently stored in the wallet's history. This list is updated during synchronization operations. Note that during transaction fetching, individual transactions and updates are published in real-time via the IEventDispatcher, not directly through this property for immediate consumption.

```
IReadOnlyList<Transaction> Transactions { get; }
```

Property Value

IReadOnlyList <Transaction>

Methods

SyncTransactionsAsync()

Asynchronously synchronizes the wallet's transaction history with the blockchain (for main and change addresses). During synchronization, new transactions related to the wallet's addresses are fetched and added to the history.

Real-time Updates via Events: As the synchronization process runs, the IEventDispatcher publishes the following events to provide real-time updates and process lifecycle notifications:

- FetchingStartedEventArgs: Published when the transaction fetching process begins.
- TransactionAddedEventArgs: For each newly discovered transaction.
- TransactionConfirmedEventArgs: When a transaction reaches a confirmation threshold.
- TransactionDateUpdatedEventArgs: If a transaction's date information is updated.
- FetchingCompletedEventArgs: Published when the entire transaction fetching and synchronization process is complete.

These events allow subscribers to receive immediate notifications of transaction activity and the overall synchronization status. The Transactions property provides access to the complete, synchronized transaction history *after* the synchronization process finishes.

```
Task<TransactionFetchResult> SyncTransactionsAsync()
```

Returns

Task <TransactionFetchResult>

A Task<TResult> representing the asynchronous operation. The task result contains a TransactionFetchResult indicating whether the synchronization process encountered network errors. To access the updated list of transactions after synchronization, use the Transactions property.

Interface ITxMapper

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for mapping Bitcoin transaction data between different formats and models. This interface defines methods for converting transaction representations from external libraries (like NBitcoin) into the library's internal models optimized for storage and processing (Transaction) and for working with UTXOs. Implementations of this interface are responsible for handling the data transformation logic.

```
public interface ITxMapper
```

Methods

NBitcoinTxToBtcTxForStorage(Transaction)

Asynchronously maps an NBitcoin NBitcoin.Transaction object to a Transaction object. This method is used to convert a transaction representation from the NBitcoin library into the library's internal format suitable for storage in the wallet's transaction history and further processing within the application.

```
Task<Transaction> NBitcoinTxToBtcTxForStorage(Transaction tx)
```

Parameters

`tx` Transaction

The NBitcoin NBitcoin.Transaction object to be mapped.

Returns

Task<Transaction>

A Task<TResult> representing the asynchronous operation. The task result is a Transaction object that is the mapped representation of the input NBitcoin transaction.

UtxoToCoin(UtxoDetailsElectrumx)

Maps a UtxoDetailsElectrumx object (representing UTXO details from Electrumx) to an NBitcoin Coin object. This method converts UTXO data retrieved from an Electrumx server into the NBitcoin Coin format, which is commonly used within the NBitcoin library for representing transaction inputs and outputs.

```
Coin UtxoToCoin(UtxoDetailsElectrumx utxo)
```

Parameters

`utxo` UtxoDetailsElectrumx

The UtxoDetailsElectrumx object containing UTXO details from Electrumx.

Returns

Coin

An NBitcoin Coin object representing the UTXO, suitable for use with NBitcoin transaction operations.

Interface ITxValidator

Namespace: BtcWalletLibrary.Interfaces

Assembly: BtcWalletLibrary.dll

Interface for validating various aspects of Bitcoin transactions before they are built or broadcasted. This interface defines a set of methods to perform common validation checks on transaction-related data, such as recipient addresses, amounts, fees, and the sufficiency of funds. Implementations of this interface are responsible for enforcing transaction validity rules and providing specific error codes via the TransactionBuildErrorCode enum when validation fails.

```
public interface ITxValidator
```

Methods

ValidateAddress(string, out TransactionBuildErrorCode)

Validates a Bitcoin address string to ensure it is a valid and supported address format.

```
bool ValidateAddress(string destinationAddr, out TransactionBuildErrorCode txBuildErrorCode)
```

Parameters

`destinationAddr` string

The Bitcoin address string to validate.

`txBuildErrorCode` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the address is valid, false otherwise.

ValidateAddress(string, out TransactionBuildErrorCode, out BitcoinAddress)

Validates a Bitcoin address string and attempts to parse it into a NBitcoin.BitcoinAddress object.

```
bool ValidateAddress(string destinationAddr, out TransactionBuildErrorCode txBuildErrorCode,
out BitcoinAddress bitcoinDestinationAddr)
```

Parameters

`destinationAddr` string

The Bitcoin address string to validate.

`txBuildErrorCode` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

`bitcoinDestinationAddr` BitcoinAddress

Output parameter that will be populated with the parsed NBitcoin.BitcoinAddress object if validation is successful. Will be null if validation fails.

Returns

bool

True if the address is valid and parsed successfully, false otherwise.

ValidateAmount(Money, out TransactionBuildErrorCode)

Validates an amount represented as NBitcoin.Money to ensure it is a valid Bitcoin amount. Checks for conditions such as negative amounts or amounts exceeding maximum limits.

```
bool ValidateAmount(Money amount, out TransactionBuildErrorCode txBuildError)
```

Parameters

`amount` Money

The NBitcoin.Money amount to validate.

`txBuildError` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the amount is valid, false otherwise.

ValidateAmount(decimal, out TransactionBuildErrorCode)

Validates a decimal amount to ensure it is a valid Bitcoin amount. Checks for conditions such as negative amounts or amounts exceeding maximum limits.

```
bool ValidateAmount(decimal amount, out TransactionBuildErrorCode txBuildError)
```

Parameters

`amount` decimal

The decimal amount to validate.

`txBuildError` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the amount is valid, false otherwise.

ValidateCustomFee(Money, out TransactionBuildErrorCode)

Validates a custom fee amount provided as NBitcoin.Money. Checks if the custom fee is within acceptable ranges (e.g., non-negative).

```
bool ValidateCustomFee(Money customFee, out TransactionBuildErrorCode txBuildError)
```

Parameters

`customFee` Money

The NBitcoin.Money custom fee amount to validate.

`txBuildError` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the custom fee is valid, false otherwise.

ValidateCustomFee(decimal, out TransactionBuildErrorCode)

Validates a custom fee amount provided as a decimal. Checks if the custom fee is within acceptable ranges (e.g., non-negative).

```
bool ValidateCustomFee(decimal customFee, out TransactionBuildErrorCode txBuildError)
```

Parameters

`customFee` decimal

The decimal custom fee amount to validate.

`txBuildError` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the custom fee is valid, false otherwise.

ValidateFundSufficiency(Money, Money, List<UnspentCoin>, out TransactionBuildErrorCode)

Validates if the selected unspent coins (UnspentCoin) are sufficient to cover the transaction amount and the transaction fee.

```
bool ValidateFundSufficiency(Money amount, Money fee, List<UnspentCoin>
selectedUnspentCoins, out TransactionBuildErrorCode txBuildErrorCode)
```

Parameters

`amount` Money

The transaction amount (excluding fee) as NBitcoin.Money.

`fee` Money

The transaction fee as NBitcoin.Money.

`selectedUnspentCoins` List<UnspentCoin>

The list of UnspentCoin objects selected for the transaction inputs.

`txBuildErrorCode` TransactionBuildErrorCode

Output parameter that will be set to InsufficientFunds if funds are insufficient, or another relevant error code if validation fails for other reasons. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the selected unspent coins are sufficient, false otherwise.

ValidateSelectedUnspentCoins(List<UnspentCoin>, Money, out TransactionBuildErrorCode)

Validates the selected unspent coins in relation to the specified transaction amount (NBitcoin.Money). This validation might include checks beyond fund sufficiency, such as ensuring that the selected coins are valid for the given amount in other transaction-specific ways if needed.

```
bool ValidateSelectedUnspentCoins(List<UnspentCoin> selectedUnspentCoins, Money amount, out
TransactionBuildErrorCode txBuildError)
```

Parameters

`selectedUnspentCoins` List<UnspentCoin>

The list of UnspentCoin objects selected for the transaction inputs.

`amount` Money

The transaction amount (excluding fee) as NBitcoin.Money.

`txBuildError` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the selected unspent coins are valid, false otherwise.

ValidateSelectedUnspentCoins(List<UnspentCoin>, decimal, out TransactionBuildErrorCode)

Validates the selected unspent coins in relation to the specified transaction amount (decimal). This validation might include checks beyond fund sufficiency, such as ensuring that the selected coins are valid for the given amount in other transaction-specific ways if needed.

```
bool ValidateSelectedUnspentCoins(List<UnspentCoin> selectedUnspentCoins, decimal amount,
out TransactionBuildErrorCode txBuildError)
```

Parameters

`selectedUnspentCoins` List<UnspentCoin>

The list of UnspentCoin objects selected for the transaction inputs.

`amount` decimal

The transaction amount (excluding fee) as a decimal.

`txBuildError` TransactionBuildErrorCode

Output parameter that will be set to a TransactionBuildErrorCode value if validation fails. If validation is successful, this parameter may be set to None or left unchanged.

Returns

bool

True if the selected unspent coins are valid, false otherwise.

Namespace BtcWalletLibrary.Models

Classes

Coin

Represents a Bitcoin Coin (UTXO) within the wallet library, extending the base NBitcoin.Coin class with wallet-specific information. This class inherits from NBitcoin.Coin and adds a Confirmed property to track the confirmation status of the UTXO. It is used to represent spendable coins within the wallet, incorporating both the cryptographic details from NBitcoin and the wallet's internal tracking of confirmation status.

Transaction

Represents a Bitcoin transaction within the wallet library. This class models a Bitcoin transaction, encapsulating key details such as the raw transaction hex, date, transaction ID, outputs, inputs, and confirmation status. It implements the ICloneable interface to allow for creating copies of transaction objects and is marked as SerializableAttribute to enable serialization.

TransactionInput

Represents an input within a Bitcoin transaction. Transaction inputs detail the source of funds for a transaction, referencing outputs from previous transactions. This class encapsulates the specifics of a transaction input, including the originating transaction, the output index being spent, the associated address, user address status, and the amount contributed as input. Implements the ICloneable interface and is SerializableAttribute.

TransactionOutput

Represents an output of a Bitcoin transaction. Transaction outputs define where bitcoins are sent as a result of a transaction. Each output specifies a destination address and the amount of bitcoins being transferred. This class details the components of a transaction output, including the recipient's address, the value of bitcoins being sent, and whether the address belongs to the user's wallet. Implements the ICloneable interface and is marked as SerializableAttribute.

UnspentCoin

Represents an Unspent Transaction Output (UTXO), also known as an unspent coin, in the Bitcoin system. UTXOs are the fundamental units of spendable Bitcoin. Each UTXO represents a certain amount of bitcoin that is associated with a specific address and can be used as input in a new transaction. This class encapsulates the key details of a UTXO, including its value, the transaction it originated from, the address it is associated with, and its confirmation status on the blockchain.

UtxoDetailsElectrumx

Represents detailed information about an Unspent Transaction Output (UTXO) as retrieved from an ElectrumX server. ElectrumX is an open-source Electrum server implementation that provides APIs to query Bitcoin blockchain data. This class encapsulates specific UTXO details obtained from ElectrumX,

such as the transaction ID and position of the output, the raw transaction hex, confirmation status, and the associated address. It is used to represent UTXOs in a format compatible with data returned by ElectrumX.

Class Coin

Namespace: BtcWalletLibrary.Models

Assembly: BtcWalletLibrary.dll

Represents a Bitcoin Coin (UTXO) within the wallet library, extending the base NBitcoin.Coin class with wallet-specific information. This class inherits from NBitcoin.Coin and adds a Confirmed property to track the confirmation status of the UTXO. It is used to represent spendable coins within the wallet, incorporating both the cryptographic details from NBitcoin and the wallet's internal tracking of confirmation status.

```
public class Coin : Coin, ICoin, ICoinable
```

Inheritance

object ← Coin ← Coin

Implements

ICoin, ICoinable

Inherited Members

Coin._OverrideScriptCode , Coin.GetScriptCode() , Coin.GetHashVersion() , Coin.ToScriptCoin(Script) , Coin.TryToScriptCoin(Script) , Coin.TryToScriptCoin(PubKey) , Coin.ToColoredCoin(AssetId, ulong) , Coin.ToColoredCoin(BitcoinAssetId, ulong) , Coin.ToColoredCoin(AssetMoney) , Coin.OverrideScriptCode(Script) , Coin.IsMalleable , Coin.CanGetScriptCode , Coin.Outpoint , Coin.TxOut , Coin.Amount , Coin.ScriptPubKey , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Coin(Transaction, uint, bool)

```
public Coin(Transaction fromTx, uint fromOutputIndex, bool confirmed)
```

Parameters

`fromTx` Transaction

`fromOutputIndex` uint

`confirmed` bool

Properties

Confirmed

Gets a value indicating whether the coin (UTXO) is confirmed on the blockchain. True if the transaction that created this coin has reached a sufficient number of confirmations, false otherwise. This property is specific to the wallet's management of UTXOs and is not part of the base NBitcoin.Coin class.

```
public bool Confirmed { get; }
```

Property Value

bool

Class Transaction

Namespace: BtcWalletLibrary.Models

Assembly: BtcWalletLibrary.dll

Represents a Bitcoin transaction within the wallet library. This class models a Bitcoin transaction, encapsulating key details such as the raw transaction hex, date, transaction ID, outputs, inputs, and confirmation status. It implements the ICloneable interface to allow for creating copies of transaction objects and is marked as SerializableAttribute to enable serialization.

```
[Serializable]
public class Transaction : ICloneable
```

Inheritance

object ← Transaction

Implements

ICloneable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Confirmed

Indicates if the transaction has reached a sufficient number of confirmations. When true, the transaction is considered confirmed on the blockchain; otherwise, it is false.

```
public bool Confirmed { get; set; }
```

Property Value

bool

Date

Represents the date and time when the transaction was first seen or included in a block. This DateTime object stores the transaction's timestamp.

```
public DateTime Date { get; set; }
```

Property Value

DateTime

Inputs

Includes a list of transaction inputs. This list comprises TransactionInput objects, each signifying an input to this transaction, and referencing the UTXOs (Unspent Transaction Outputs) being spent.

```
public List<TransactionInput> Inputs { get; set; }
```

Property Value

List <TransactionInput>

Outputs

Contains a list of transaction outputs. Each TransactionOutput object in this list represents an output of this transaction, detailing recipient addresses and amounts.

```
public List<TransactionOutput> Outputs { get; set; }
```

Property Value

List <TransactionOutput>

TransactionHex

Provides the raw transaction data in hexadecimal format. This string holds the serialized Bitcoin transaction in its hexadecimal encoding.

```
public string TransactionHex { get; set; }
```

Property Value

string

TransactionId

Uniquely identifies the transaction within the Bitcoin blockchain. This is the transaction hash (TxId) represented as a string.

```
public string TransactionId { get; set; }
```

Property Value

string

Methods

Clone()

Creates a deep clone of the Transaction object. This method implements the Clone() interface and creates a new Transaction object with all properties copied from the original, including deep clones of the Outputs and Inputs lists and their contents.

```
public object Clone()
```

Returns

object

A new Transaction object that is a clone of the current instance.

Class TransactionInput

Namespace: BtcWalletLibrary.Models

Assembly: BtcWalletLibrary.dll

Represents an input within a Bitcoin transaction. Transaction inputs detail the source of funds for a transaction, referencing outputs from previous transactions. This class encapsulates the specifics of a transaction input, including the originating transaction, the output index being spent, the associated address, user address status, and the amount contributed as input. Implements the ICloneable interface and is SerializableAttribute.

```
[Serializable]
public class TransactionInput : ICloneable
```

Inheritance

object ← TransactionInput

Implements

ICloneable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Address

Bitcoin address associated with this transaction input. This address typically corresponds to the recipient of the bitcoins in the referenced transaction output.

```
public string Address { get; set; }
```

Property Value

string

Amount

Represents the Bitcoin amount being spent by this transaction input. This value reflects the worth of the referenced output (TrId:OutputIdx) being used as an input in the current transaction.

```
public double Amount { get; set; }
```

Property Value

double

IsUsersAddress

Indicates whether the address of this input is one managed by the user's wallet. When true, it signifies that the Address belongs to the wallet owner; otherwise, it's false.

```
public bool IsUsersAddress { get; set; }
```

Property Value

bool

OutputIdx

Specifies the index of the output within the referenced transaction that is being utilized as input. This index points to a specific output in the outputs collection of the transaction identified by TrId.

```
public int OutputIdx { get; set; }
```

Property Value

int

TrId

Identifies the transaction from which this input is spending an output. This property holds the transaction ID (TxId) of the prior transaction being referenced.

```
public string TrId { get; set; }
```

Property Value

string

Methods

Clone()

Creates a new object that is a deep copy of the current TransactionInput instance. Implements the Clone() method to facilitate the creation of exact replicas of TransactionInput objects.

```
public object Clone()
```

Returns

object

A new TransactionInput object that is a clone of this instance.

Class TransactionOutput

Namespace: BtcWalletLibrary.Models

Assembly: BtcWalletLibrary.dll

Represents an output of a Bitcoin transaction. Transaction outputs define where bitcoins are sent as a result of a transaction. Each output specifies a destination address and the amount of bitcoins being transferred. This class details the components of a transaction output, including the recipient's address, the value of bitcoins being sent, and whether the address belongs to the user's wallet. Implements the ICloneable interface and is marked as SerializableAttribute.

```
[Serializable]
public class TransactionOutput : ICloneable
```

Inheritance

object ← TransactionOutput

Implements

ICloneable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Address

Bitcoin address to which bitcoins are being sent. This property contains the recipient's Bitcoin address for this output.

```
public string Address { get; set; }
```

Property Value

string

Amount

Defines the quantity of bitcoins being transferred in this output. This value represents the amount of bitcoins, typically in satoshis, designated for the recipient address.

```
public double Amount { get; set; }
```

Property Value

double

IsUsersAddress

Flags if the output address is associated with the user's wallet. When true, it indicates that the Address is an address managed by the current wallet; otherwise, it is false.

```
public bool IsUsersAddress { get; set; }
```

Property Value

bool

Methods

Clone()

Creates a new object that is a deep copy of the current TransactionOutput instance. Implements the Clone() method to enable the creation of exact copies of TransactionOutput objects.

```
public object Clone()
```

Returns

object

A new TransactionOutput object that is a clone of this instance.

Class UnspentCoin

Namespace: BtcWalletLibrary.Models

Assembly: BtcWalletLibrary.dll

Represents an Unspent Transaction Output (UTXO), also known as an unspent coin, in the Bitcoin system. UTXOs are the fundamental units of spendable Bitcoin. Each UTXO represents a certain amount of bitcoin that is associated with a specific address and can be used as input in a new transaction. This class encapsulates the key details of a UTXO, including its value, the transaction it originated from, the address it is associated with, and its confirmation status on the blockchain.

```
public class UnspentCoin
```

Inheritance

object ← UnspentCoin

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Address

Bitcoin address associated with this UTXO. This address is the recipient address to which the bitcoins were sent in the output that created this UTXO.

```
public string Address { get; set; }
```

Property Value

string

Amount

Amount of bitcoins this UTXO represents. This value is typically expressed in satoshis (the smallest unit of Bitcoin) as a decimal.

```
public decimal Amount { get; set; }
```

Property Value

decimal

Confirmed

Boolean indicating whether the transaction that created this UTXO is confirmed on the blockchain. True if the transaction has reached a sufficient number of confirmations, making this UTXO spendable with a higher degree of certainty, false otherwise.

```
public bool Confirmed { get; set; }
```

Property Value

bool

TransactionId

Transaction ID (TxId) of the transaction that created this UTXO. This property refers to the transaction hash of the transaction where this unspent output was originally created.

```
public string TransactionId { get; set; }
```

Property Value

string

Class UtxoDetailsElectrumx

Namespace: BtcWalletLibrary.Models

Assembly: BtcWalletLibrary.dll

Represents detailed information about an Unspent Transaction Output (UTXO) as retrieved from an ElectrumX server. ElectrumX is an open-source Electrum server implementation that provides APIs to query Bitcoin blockchain data. This class encapsulates specific UTXO details obtained from ElectrumX, such as the transaction ID and position of the output, the raw transaction hex, confirmation status, and the associated address. It is used to represent UTXOs in a format compatible with data returned by ElectrumX.

```
public class UtxoDetailsElectrumx
```

Inheritance

object ← UtxoDetailsElectrumx

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Address

Bitcoin address associated with this UTXO, as reported by ElectrumX. This is the address to which the bitcoins in this UTXO were sent.

```
public string Address { get; }
```

Property Value

string

Confirmed

Value indicating whether the transaction that created this UTXO is considered confirmed by ElectrumX. True if the transaction has reached a sufficient number of confirmations as determined by the ElectrumX server, false otherwise.

```
public bool Confirmed { get; }
```

Property Value

bool

TransactionHex

Raw transaction data in hexadecimal format for the transaction that created this UTXO, as reported by ElectrumX. This can be used to access the full transaction details if needed.

```
public string TransactionHex { get; }
```

Property Value

string

TransactionId

Transaction ID (TxId) of the transaction that created this UTXO, as reported by ElectrumX. This is the hash of the transaction in which this UTXO was an output.

```
public string TransactionId { get; }
```

Property Value

string

TransactionPos

Output position (index) within the transaction (TransactionId) that created this UTXO, as reported by ElectrumX. This index identifies the specific output in the transaction's output list that corresponds to

this UTXO.

```
public int TransactionPos { get; }
```

Property Value

int