

# 1. Конфигурация BtcLib

Файл config.json на FTP описывает настройки, которые понадобятся для работы с нодой по RPC.

```
{
  "host": "127.0.0.1",
  "port": "18332",
  "rpcuser": "people_bitcoins",
  "rpcpassword": "MW6EJqKCWe"
}
```

Важно: BtcLib не уважно, что это - testnet или mainnet. Как правило, отличается порт, но вся работа с нодой происходит без знаний о типе сети.

Важно: testnet использует отдельный список аккаунта(кошельков) на ноде. Создав на одной сети кошелек, он не будет доступен на другой сети.

Вывод: создавая кошельки нужно помнить, что они жестко привязаны к datadir-ноде и используемой сети (testnet/mainnet/..)

---

## Создание BtcLib объекта - для выполнения команд к ноде (index.php)

```
// Подключение namespace BtcPhp
require "vendor/autoload.php";

// BUG некоторые классы почему-то требуют отдельной загрузки
require_once __DIR__ . '/src/PeopleBitcoins/BtcPhp/TxInfo.php';
require_once __DIR__ . '/src/PeopleBitcoins/BtcPhp/TxInfoExtended.php';

use PeopleBitcoins\BtcPhp\BtcLib;

if(!file_exists( filename: 'config.json')) {
    throw new Exception( message: "No config file on disk");
}

// Загрузка $config из json-файла (host, port, rpcuser, rpcpassword)
$config = json_decode(file_get_contents( filename: 'config.json'), associative: true);

// Клиент для работы с нодой (настройки из нашего конфига)
$btclib = new BtcLib($config['host'], $config['port'], $config['rpcuser'], $config['rpcpassword']);
```

## 2. Интерфейс для теста

1 - Отображается % синхронизации с нодой. Что бы работать с блокчейном, нужно всегда быть на 99-100%.

2 - отображаются кошельки, которые загружены из DAT-файла на FTP (wallets..IP:port ноды.....dat). На ноде могут быть и другие, но мы просто показываем массив WalletInfo загруженный из файла.

3 - название аккаунта кошелька на ноде. Что бы увидеть bitcoin-адрес, нужно кликнуть по квадратику, адрес появится в п. 5

4 - кнопка для создания нового кошелька на ноде. Создает аккаунт кошелька со случайным именем по RPC, шифрует его случайным паролем, добавляет к уже имеющемуся списку WalletInfo и перезаписывает DAT-файл на FTP, где хранятся кошельки.

5 - поле, куда подставляется адрес кошелька для дальнейших команд (balance, transactions). Здесь можно только наш кошелек использовать (из списка выше) (для которого есть аккаунт на ноде и известен passphrase).

6 - скопировать в буфер обмена выбранный кошелек (адрес bitcoin)

7 - запросить баланс по кошельку. Показывается исходя из минимума 6 подтверждений по транзакциям (может запаздывать, если деньги еще "идут")

8 - загружает таблицу с информацией о транзакциях (макс. 10 первых записей, от старых к новым). Транзакции могут быть Receive или Send, у них будет Txid, Amount (сумма зачисления) и Fee (коммисия)

9. Можно загружать только одну транзакцию в таблицу, введя ее txid хэш. Она должна быть связана с выбранным кошельком - просто так информация по случайной транзакции может быть не доступна (без txindex=1 на ноде).

The screenshot shows a web browser window with the address bar displaying 'work.people-bit...'. The page title is 'Sync: 100%'. The main content area is titled 'Wallets' and contains two red buttons labeled 'wIC5ZGxA' and 'wIDGPIoA'. Below the 'Wallets' section is a '+ Create wallet' button. The 'Wallet info' section has an 'Address' field and a 'Copy' button. The 'Balance' section has a 'Load' button. The 'Transactions' section has a 'Load' button. The 'Single transaction' section has an 'Address' field and a 'Load' button. The 'Create transaction' section has 'From', 'To', and 'Amount' fields, with 'Amount' set to '0.0001', and 'Send' and 'Clear' buttons.

1 - Sync: 100%

2 - Wallets

3 - wIC5ZGxA, wIDGPIoA

4 - + Create wallet

5 - Address

6 - Copy

7 - Load

8 - Load

9 - Single transaction

10 - Create transaction

10. Создать перевод (транзакцию). From - кошелек, с которого отправляются деньги. Это должен быть наш кошелек, который есть на ноде и мы знаем passphrase (WalletInfo, загруженные из DAT-файла). To - любой адрес bitcoin. Amount - сумма, которую нужно зачислить. Помимо этой суммы спишется небольшая комиссия, которая потом отображается как transaction fee.

Нода создает транзакцию и будет предлагать нодам в сети ее подтвердить. Это займет время. В таблице транзакций можно видеть сколько подтверждений у каждой - 3-6 подтверждений уже считается “общепризнанная транзакция”.



### 3. Как использовать на сайте?

Подключить autoload, что бы можно было использовать классы. Например, скопировать файлы из папки src (FTP) и добавить в Composer.json подгрузку оттуда:

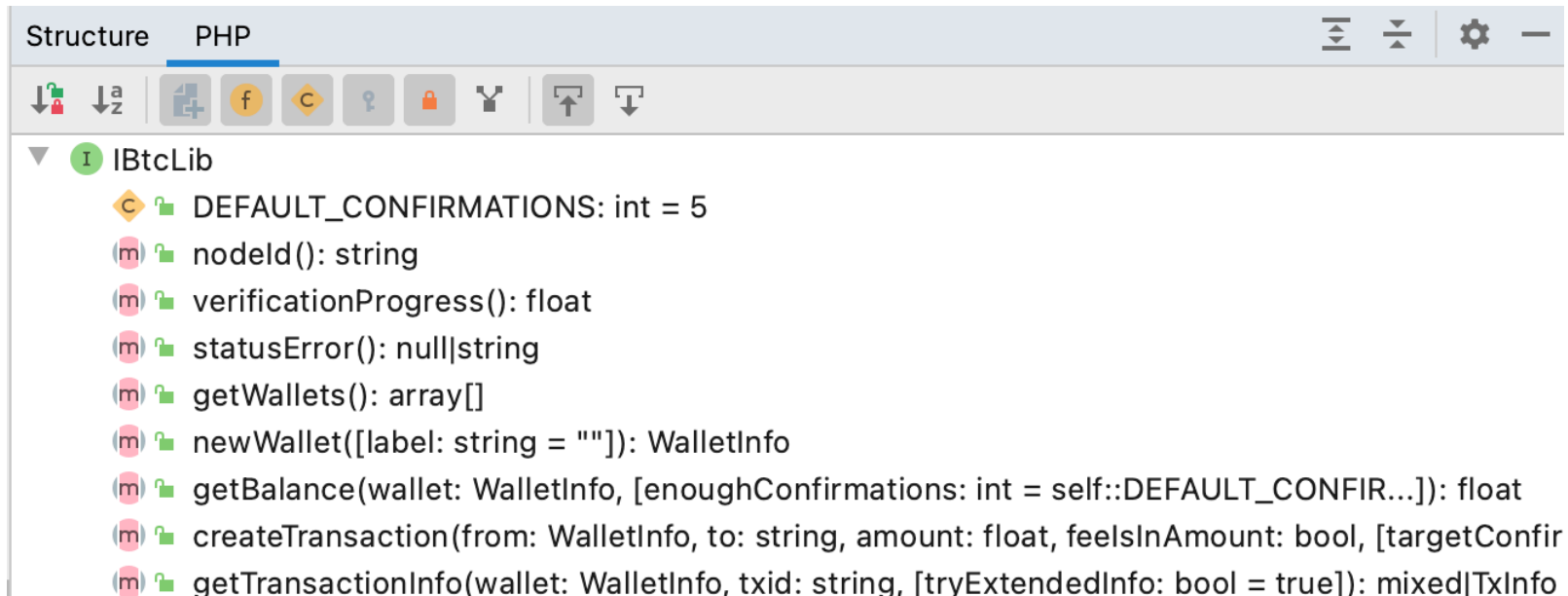
```
{
  "require": {
    "denpa/php-bitcoinrpc": "^2.1", v2.1.3
    "ext-json": "*"
  },
  "autoload": {
    "classmap": ["src/"]
  }
}
```

composer dump-autoload после этого.

Еще установить зависимости: composer require denpa/php-bitcoinrpc

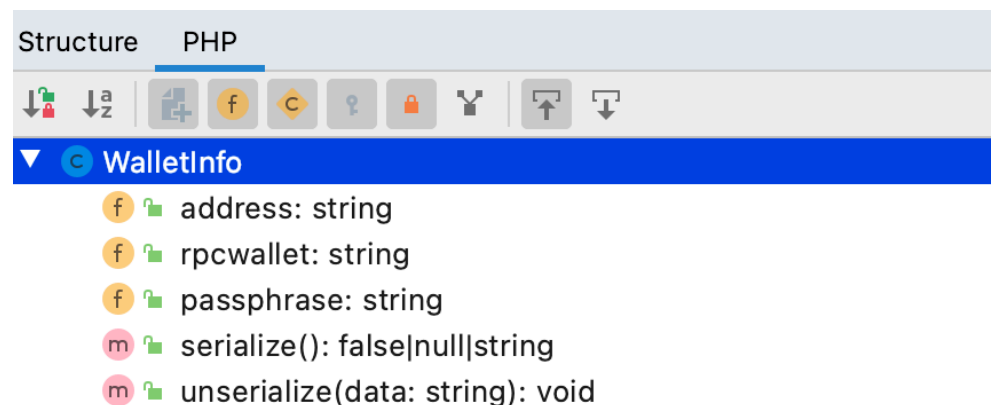
Теперь код из п. 1 должен работать на сайте (начиная с use PeopleBitcoins\..)

Создали объект \$btclib с нужными параметрами подключения к RPC, теперь нам доступны методы IBtcLib:



Все методы выполняют HTTP (JSON-RPC) запрос к ноде. Самое интересное, что у нас есть класс WalletInfo.

Например, выполнив newWallet() мы получаем объект WalletInfo. Он содержит название сгенерированного аккаунта, bitcoin-адрес, и пароль для доступа.



Этот аккаунт и его приватный ключ хранится на самой ноде. На диске аккаунт зашифрован паролем passphrase. Поэтому на сайте для каждого

Объект WalletInfo можно сохранить в базе, в колонке в виде строки. Или в файле хранить (как мы делаем в тестовом приложении). Пример:

```
$userWallet = $btclib->newWallet( label: "userwallets"); // WalletInfo
$strwallet = serialize($userWallet); // string
$userWallet = unserialize($strwallet); // WalletInfo
```

Ну и собственно мы можем легко выполнять операции между кошельками. Или отправить деньги на любой Bitcoin кошелек.

```
$w1 = $btclib->newWallet();  
$w2 = $btclib->newWallet();  
  
$txid = $btclib->createTransaction($w1, $w2->address, amount: 0.0001, feelsInAmount: false);  
  
$txInfo = $btclib->getTransactionInfo($w1, $txid);  
  
echo "Money send! Currently, confirmations: {$txInfo->confirmations}";
```

Особенности работы с BtcLib - это:

- Методы возвращают разные данные в разное время. Хотя можно сериализовывать и хранить TxInfo объекты (данные о транзакциях), это имеет мало смысла, если транзакции “свежие”.
- Могут выбрасываться исключения при вызове методов.
- Стоит открыть IBtcLib.php и почитать описание нужного метода.
- Стоит хранить WalletInfo в базе привязанные к пользователям, потому что данные в WalletInfo - суперценные (название аккаунта на ноде + пароль для доступа к его балансу)

Для тестовой сети можно получить крипту в bitcoin test faucet, мне помогли эти:

- <https://testnet-faucet.mempool.co/>
- <https://bitcoinafaucet.uo1.net/send.php>
- <https://coinafaucet.eu/en/btc-testnet/>

Возможность использования faucet обычно 1 раз в сутки

## Пример: построение массива всех транзакций в убывающем порядке

Сам по себе метод `getTransactions` (и `getTransactionsExtended`) возвращает ограниченное количество транзакций и начинается с самой старой, скорее всего понадобится полный массив в обратном порядке. Вот пример.

```
$userWallet = $btclib->newWallet();

// предположим, что прошло много транзакций по кошельку,
// мы хотим отобразить полную таблицу от новых к старым
$transactions = [];

$pageSize = 10;
for($offset = 0; ; ) {
    // важно: эта версия возвращает iterable, а не массив (лечится x = array(x))
    $items = $btclib->getTransactionsExtended($userWallet, $pageSize, $offset);
    foreach($items as $item) {
        $transactions[] = $item;
        $offset++;
    }
}

// развернем порядок, что бы самая последняя была сверху, а не снизу
$transactions = array_reverse($transactions);
```

Для

копипасты:

```
$userWallet = $btclib->newWallet();
```

```
// предположим, что прошло много транзакций по кошельку,
```



```
// мы хотим отобразить полную таблицу от новых к старым
```

```
$transactions = [];
```

```
$pageSize = 10;
```

```
for($offset = 0; ; ) {
```

```
// важно: эта версия возвращает iterable, а не массив (лечится x = array(x))
```

```
$items = $btclib->getTransactionsExtended($userWallet, $pageSize, $offset);
```

```
foreach($items as $item) {
```

```
    $transactions[] = $item;
```

```
    $offset++;
```

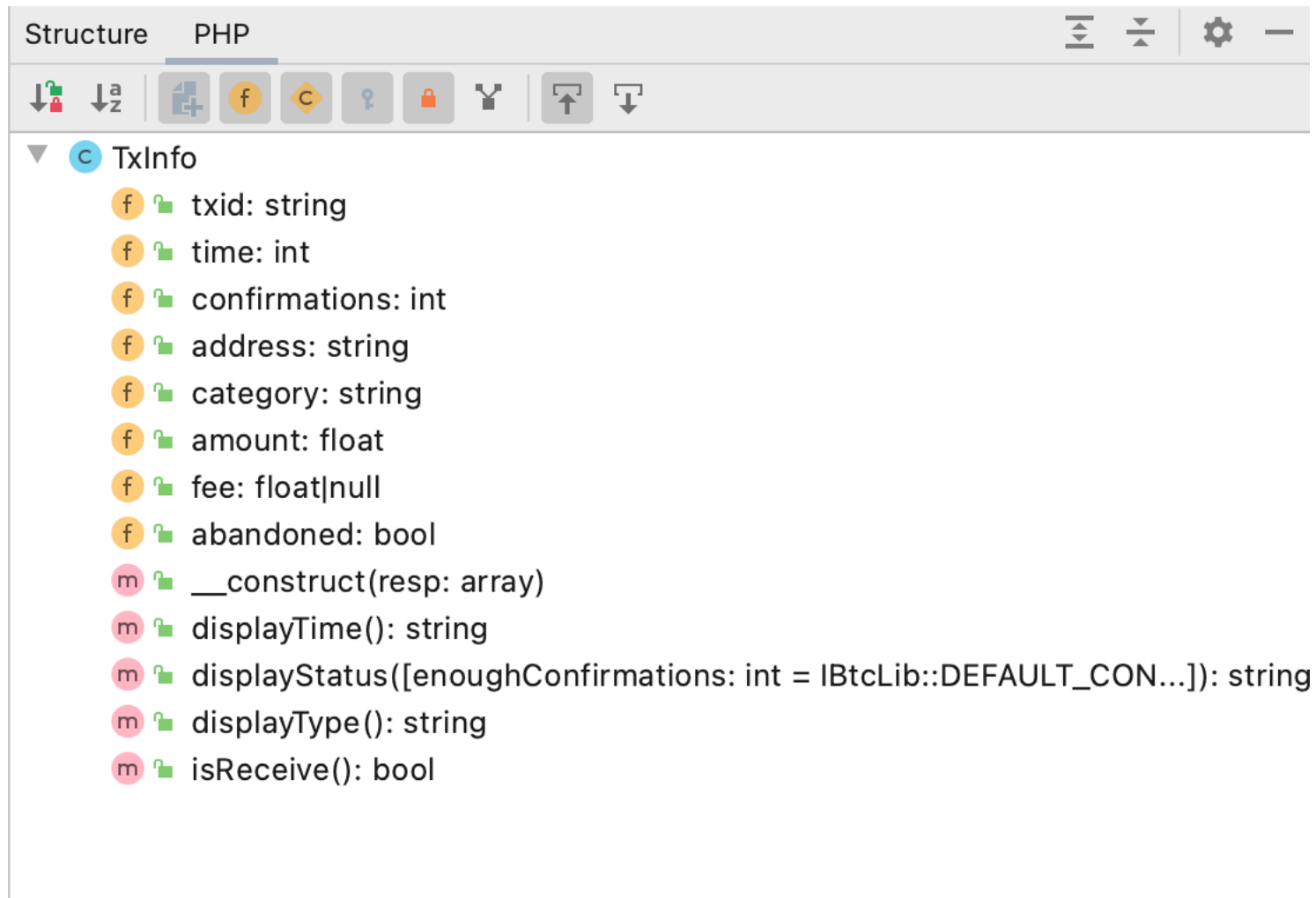
```
}
```

```
}
```

```
// развернем порядок, что бы самая последняя была сверху, а не снизу
```

```
$transactions = array_reverse($transactions);
```

Класс TxInfo, информация, обычно доступная для любой транзакции:



The screenshot shows an IDE window with the 'Structure' tab selected. The class 'TxInfo' is expanded, showing its properties and methods. The properties are: txid: string, time: int, confirmations: int, address: string, category: string, amount: float, fee: float|null, and abandoned: bool. The methods are: \_\_construct(resp: array), displayTime(): string, displayStatus([enoughConfirmations: int = IBtcLib::DEFAULT\_CON...]): string, displayType(): string, and isReceive(): bool.

```
Structure  PHP
```

▼ TxInfo

- f txid: string
- f time: int
- f confirmations: int
- f address: string
- f category: string
- f amount: float
- f fee: float|null
- f abandoned: bool
- m \_\_construct(resp: array)
- m displayTime(): string
- m displayStatus([enoughConfirmations: int = IBtcLib::DEFAULT\_CON...]): string
- m displayType(): string
- m isReceive(): bool

---

## TxInfoExtended - дополнительное поле и зачем оно нужно

Методы для получения транзакции(й) вернут TxInfoExtended, если:

- getTransactionInfo с параметром tryExtendedInfo (true - по умолчанию)
- getTransactionsExtended

В этом объекте будет поле senderAddress. Если транзакция isReceive() - нам передали деньги - то senderAddress содержит адрес, который перевел нам деньги.

Для этого функционала бота должна работать с txindex=1.

TxInfo::\$address - это всегда адрес того, КОМУ переводили деньги.