Technical description

Contents

1. Technologies used in the client-side module

Component receiving data related to desired loan conditions
\components\MainpageCtrl.class.php
class MainpageCtrl extends admin\lib\frontend\base\Component {
public function actDefault() {//Main website page with a loan calculator
public function actNewOrder() {//Acceptance of conditions, record into session, redirect to questionnaire
function actInfo() {//information pages

Customer registration component, filling out personal details
Controller class, event handler
class NeworderCtrl extends admin\lib\frontend\base\Component {
function actDefault() { display questionnaire
function actStepAll() { receive and record data
function actSmsOferta() { event for sending an SMS verification code
function actCheckSmsCode() { SMS code verification
function actGo() { create a loan application
function actPm1() { method selection interface for the receipt of funds
function actSocial() { connect social network to the account

Customer registration component model. Process business logic.
model\NewOrder.php

```
namespace model;

class NewOrder {
static function smsoferta() {     //Send code
static function checkSmsCode() {       //Validate code
static function preSaveAll() {     //Preliminarily save personal data
static function step1Check() {     //Validate data
if ($_POST['surname'] == '') {
        $error['surname'] = "Este espacio se require completar";   //"This field needs to be completed.";
   }
   if ($_POST['name'] == '') {
        $error['name'] = "Este espacio se require completar";   //" This field needs to be completed.";
   }
...
static function step1Save() {   //Save data
static function upload() {      //Load required documents
static function create() {      //Create application
```

Base functionality of the system

core\class.auth.php
class authStd {
User authentication by social network account or by mobile phone number. User registration in the system.
Logout from the system
Receve user details, current state, credit status

function registration($fields, $values = array()) { //Registration
static function addUser($data = array()) { //Add user
function userData_update($fields, $values) { //Update customer data
function userFamily_update($family = array()) { // Update additional customer data

core\class.order.php
class order {
function isTakeOrder() {//Possibility to receive new application
static function addOrder($sum, $count, $duration, $percent, $user_id, $rank_id = 0) { //Add application into the system
public static function addOrderProlongation($order_id, $days) {//Add loan extension agreement
function getOrders() { //Application list
function getPayments($order_id) { //List of payments for an application
function getAccept($order_id) //Information on signing
static function getTae($days){ Cost of credit calculation

SMS notification management system
core\plugins\sms.php
require_once ROOT . "/core/class.plugin.php";

class SMS extends PLUGIN {

  protected $smsprovider;

  function __construct() {
    parent::__construct();
    $cfg = Cfg::get('sms');
    if ($cfg['provider'] == 'smsru') {
      require_once(ROOT . '/core/plugins/smsru.php');
      $this->smsprovider = new smsru($cfg);
    }
…
Select SMS provider based on settings and send notifications
Perform delivery through the chosen service provider

class smsapi {

  /**
   * @var array(login,password)

```
 */
private $cfg = array();

function __construct($cfg) {
   $this->cfg = $cfg;
}

function send($phone, $text) {
   $params = $params = array(
      'username' => $this->cfg['login'],
      'password' => $this->cfg['password'],
      'to' => $phone,
      'from' => $this->cfg['from'],
      'eco' => 0,
      'message' => $text,
   );
   return self::sms_send($params);
}

function sms_send($params, $backup = false) {

   static $content;

   if ($backup == true) {
      $url = 'https://api2.smsapi.com/sms.do';
   } else {
      $url = 'https://api.smsapi.com/sms.do';
   }

   $c = curl_init();
   curl_setopt($c, CURLOPT_URL, $url);
   curl_setopt($c, CURLOPT_POST, true);
   curl_setopt($c, CURLOPT_POSTFIELDS, $params);
   curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
   curl_setopt($c, CURLOPT_SSL_VERIFYPEER, false);
   curl_setopt($c, CURLOPT_SSL_VERIFYHOST, false);

   $content = curl_exec($c);
   $err= curl_error($c);
   $http_status = curl_getinfo($c, CURLINFO_HTTP_CODE);

   if ($http_status != 200 && $backup == false) {
      $backup = true;
      self::sms_send($params, $backup);
   }
```

```
    curl_close($c);
    return $content;
  }

  static function sms_send_with_token($params, $token, $backup = false) {

    static $content;

    if ($backup == true) {
      $url = 'https://api2.smsapi.com/sms.do';
    } else {
      $url = 'https://api.smsapi.com/sms.do';
    }

    $c = curl_init();
    curl_setopt($c, CURLOPT_URL, $url);
    curl_setopt($c, CURLOPT_POST, true);
    curl_setopt($c, CURLOPT_POSTFIELDS, $params);
    curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($c, CURLOPT_HTTPHEADER, array(
      "Authorization: Bearer $token"
    ));

    $content = curl_exec($c);
    $http_status = curl_getinfo($c, CURLINFO_HTTP_CODE);

    if ($http_status != 200 && $backup == false) {
      $backup = true;
      sms_send($params, $token, $backup);
    }

    curl_close($c);
    return $content;
  }
}
```

Component of the User Area

Controller class, event handler

```
class CabinetCtrl extends admin\lib\frontend\base\Component {

function actAccept()          //  Process initial login to the User Area
function actDefault()         //  Obtain information on customer's loans
```

function actPay() // Repay loan
function actProlongation() // Prolong loan
function actSmsoferta() // Sign loan agreement
function actOrders() // Loan documents
function actAnketa() // Customer's personal data
function actChange_mail() // Change email address
function actChange_phone() // Change phone number
function actChangePass() // Change authentication password
function actChangePayType() // Change loan disbursement method
function actPayType() // Change loan repayment method
function actNeworder() // Create repeated loan application

Customer's User Area component model

namespace model;

class Cabinet

function accept() // Save information on first login to the User Area
function getAllOrders() // Obtain data on customer's loans
function getCards() // Obtain information on customer's cards
function newCreditSumAgreement() // Agree to loan conditions
function newCreditSumDisagreement() // Decline loan
function getPayData() // Obtain current debt data
function prolongDaysData // Obtain prolongation data

Authentication page component

Controller class, event handler
class LoginCtrl extends admin\lib\frontend\base\Component

function actRecovery() // Recover authentication password
function actLoginEmail() // Email authentication
function actLoginPhone() // Mobile number authentication

Calculation functionality
core\orders\UserOrder.php
Class responsible for choice of lending policy for a specific loan agreement

admin\lib\orders\UserOrderStandart.php
Class responsible for calculation

```
class UserOrderStandart {

  public static function getCfgParams() {
    return array(
      'amnesty' => 'Days of penalties not charged and percent after past due',
      'fine' => array(
        'single_pay' => 'One-time payment', //
        'pday' => 'Penalty per day, percent', // Penalty per annum, percent
        'pyear' => ' Penalty per annum, percent ', // Penalty per annum, percent
      )
    );
  }
public function getInfo() { //General loan information
public function getRefundDate($date = null) { //Calculate refund date for dates
/**
   * Total debt amount
   *
   * @return float
   */
  public function getTotalDebtAtTodaysDate($param_date = null, $param = false) {
    return $this->_calcTotalDebtAtTodaysDate($param_date, $param);
  }

  //  sba 04.08.2015 Total penalty for a specific date
  public function getTotalDebtAtDate($param_date, $param = false) {
    $this->_calcTotalDebtAtTodaysDate($param_date, $param);
    return $this->oFineDebt;
  }

  //  sba 12.08.2015 Total interest for a specific date
  public function calcTotalProcentstToDates($param_date = null, $param = false) {
    $this->_calcTotalDebtAtTodaysDate($param_date, $param);
    return $this->oPercentsDebt;
  }

  //  sba 26.08.2015 Total principal debt for a specific date
  public function getDebtAtDate($param_date, $param = false) {
    $this->_calcTotalDebtAtTodaysDate($param_date, $param);
    return $this->oMainDebt;
  }
```

2. Description of technologies used in the administrator control panel

admin\plugins\results\back\settings\results.php

Front controller

admin\plugins\results\back\settings\Result.class.php

Loan management component

```
class Result extends AdminComponent {
function actDefault() {//Loan list

public function actChangeStatus() {//Change loan status
    global $db;
    $lRet = ['status' => 0];

    if (isUserHavePermissionForSectionLinkUsing(SECTION_FIRST_LOAN, LINK_CHANGE_POINTS_AND_STATUS,
null)) {
        $order_id = (int) $_POST['order_id'];
        $rs = DB::select("SELECT o.*, u.phone, u.check_mail, u.email FROM orders AS o LEFT JOIN users AS u ON
o.user_id=u.id WHERE o.id=$order_id");
        if ($rs->next()) {

...
function actGetUserComments() {//Comments on customer
    global $db;
    $id = $_POST['id'];

…
function actSendUserComments() {//Send a comment on customer
    global $db;
    $id = $_POST['id'];
    $desctext = $_POST['desctext'];
…
public function actProlongateOrder() {//Prolong loan
    global $db;
    $result = array('status' => 0, 'message' => Success!');
    $order_id = $_POST['oid'];
…
public function actPay() {//Loan repayment interface
    $sectionId = $_POST['sec_id'];
    $orderId = $_POST['oid'];
```

```
…
public function actSendPay() {//Send payment data
    global $db;
    $result = array('status' => 0, 'message' => Success!');
    $order_id = $_POST['oid'];
    $order_sum = $_POST['osum'];
    $pay_from = $_POST['ofrom'];
…
function changeStatus2() {//Disburse loan, change payment disbursement status

function correct_super() {//Adjust loan conditions

function takemoneynow() {//Disburse funds through acquiring
function backmoney() {//Return funds through acquiring
function openPicData() {//Document overview

function getanketadata() {//View personal details

function modifyanketadata() {//Change personal details

function openLoanData() {//View loan data

function scorringDetails() {//Scoring details

admin\plugins\results\back\js\res.js
```

Script handling events in the browser on client's side

```
Result = {
  load: function (onresplag) { //Call loan list loading
    jQuery.ajax({
      type: "POST",
      url: "/admin/scr/loadPlugSettings.scr.php",
      data: "plugName=results&onresplag=" + onresplag,
      success: function (html) {
        CURRENT_LOAN_PLUGIN = onresplag === undefined ? 1 : onresplag;
        jQuery("#main-content").html(html);
        reset_result_window();
```

```
        }
      });
  },
```

admin\plugins\results\back\settings\parts\result.tpl.php

Configure access permissions in the CRM

admin\plugins\admins_and_permissions\UsersAccess.class.php

Class for management of permissions and operators' access to the CRM

```
class UsersAccess extends AdminComponent {

  function getRoles() {//Roles
    return DB::select("SELECT * FROM adm_users_role")->toArray();
  }
function actRoles() {
    $data['rs'] = $this->getRoles();
    $this->display($data, dirname(__FILE__) . '/roles.tpl.php');
    exit;
  }

  function actSaveRoles() {
    $cfg = '';
    if (!empty($_POST['cfg'])) {
      $cfg = json_encode($_POST['cfg']);
    }
    $data = array('name' => $_POST['name'], 'cfg' => $cfg);
    DB::update('adm_users_role', $data, "id=" . $_POST['id']);
    echo json_encode(array('msg' => Accepted'));
    exit;
  }
```

Browser-side handler

admin\plugins\admins_and_permissions\js\main.js

```
var UsersAccess = {
  roles: function () {
    $.ajax({
      type: "POST",
      url: '/admin/plugins/admins_and_permissions/?act=roles',
      data: "",
      success: function (html) {
        $("#main-content").html(html);
        UsersAccess.initRoles();
```

```
        }
    });
  },
  initRoles: function () {
    $('#create_role_form').submit(function () {
      $.post($(this).attr('action'), $(this).serialize(), function (res) {
        if (!!res.err) {
          alert(res.err);
        }
        if (!!res.msg) {
          alert(res.msg);
          UsersAccess.roles();
```

Notification distribution system and configuration

admin\plugins\notice\Notice.class.php

```
class Notice extends AdminComponent {

  private function getTemplateList() {//List of templates
    $cond = "1=1 AND not_editable <> '1'";
    return DB::select("SELECT * FROM sms_template WHERE $cond")->toArray();
  }

  private function getMailTemplatePath() {
    return ROOT . "/storage/mail/simple.html";
  }

  function getSndType($key=null){
    $snd_type=  array(0 => 'Email sms', 1 => 'sms', 2 => 'Email');
    if($key===null){
      return $snd_type;
    }
    if(isset($snd_type[$key])){
      return $snd_type[$key];
    }
  }



  function actTemplate() { //Interface for input and configuration of notification templates
    $data['rs'] = $this->getTemplateList();
    $data['date_default'] = date('Y-m-d');
    $data['hour_default'] = 12;
    $data['mail_template'] = file_get_contents($this->getMailTemplatePath());

    $this->display($data, dirname(__FILE__) . '/template.tpl.php');
```

```
  }
```
admin\plugins\notice\notice.js

Browser-side event handler

```
Notice = {
  report: function () {
    $.ajax({
      type: "POST",
      url: '/admin/plugins/notice/?act=report',
      data: "",
      success: function (html) {
        $("#main-content").html(html);
        Notice.reportInit();
      }
    });
  },
  reportInit: function () {
    $("input.date").datetimepicker({
      timepicker: false,
```

3. **Using .Net to enable cryptographic protection (validate server certificates, adding/removing signature for requests/responses).**

```
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
...

Configuring application
namespace EqufaxCH
{
   public partial class Form1 : Form
   {
```

```
    public Form1()
    {
      InitializeComponent();
      textBox2.Text = ConfigurationManager.AppSettings["kbki_file"];
…
```

Finding and obtaining certificates:

```
      public X509Certificate2 GetCert(string subjectName)
    {

      var store = new X509Store(StoreName.My,
       StoreLocation.CurrentUser);
      store.Open(OpenFlags.ReadOnly);

      try{var certs = store.Certificates.Find(X509FindType.FindBySerialNumber,subjectName, false);...
```

Data access over secure ssh protocol:

```
PasswordConnectionInfo connectionInfo = new PasswordConnectionInfo(textBox8.Text,
Convert.ToInt32(textBox7.Text), textBox6.Text, textBox5.Text);
            connectionInfo.Timeout = TimeSpan.FromSeconds(30);
            using (var client = new SshClient(connectionInfo))
            {
              try
              {
                this.SetText("Attempting SSH connection ... \r\n");
                client.Connect();

…
```

Read and write MySQL data

```
          MySqlConnection conn;
          MySqlCommand comm;
          conn = new MySqlConnection(textBox3.Text);
```

Remote server connection:

```
          HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
          request.Proxy.Credentials = CredentialCache.DefaultCredentials;
          request.CachePolicy = new HttpRequestCachePolicy(HttpRequestCacheLevel.NoCacheNoStore);
          Stream RequestStream = (Stream)request.GetRequestStream();
```

4. Initial token issue

The functionality of initial token issue conforms to the ERC20 standard.

Contract description

```
contract ERC20 {

  uint public totalSupply;                  // Current number of coins issued:
  function balanceOf(address who) constant returns (uint);   // Balance by address
  function transfer(address to, uint value);  // Token transfer
  function allowance(address owner, address spender) constant returns (uint); // Configure permission

  function transferFrom(address from, address to, uint value);  // Transfer another user's tokens
  function approve(address spender, uint value); // Approval of token usage

  event Transfer(address indexed from, address indexed to, uint value); // Token transfer events
  event Approval(address indexed owner, address indexed spender, uint value);  //Approval events
}
```

Contract implementation

```
contract StandardToken is ERC20 {

  string public constant name = "Token Name";  // Full name of token
  string public constant symbol = "TKN";             // Short name of token
  uint8 public constant decimals = 18;
  mapping (address => mapping (address => uint)) allowed;
  mapping (address => uint) balances;
  function transferFrom(address _from, address _to, uint _value) {
  var _allowance = allowed[_from][msg.sender];

    // Check is not needed because safeSub(_allowance, _value) will already throw if this condition is not met
    // if (_value > _allowance) throw;
    balances[_to] += _value;
    balances[_from] -= _value;
    allowed[_from][msg.sender] -= _value;
    Transfer(_from, _to, _value);
  }

  function approve(address _spender, uint _value) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
  }

  function allowance(address _owner, address _spender) constant returns (uint remaining) {
    return allowed[_owner][_spender];
  }

  function transfer(address _to, uint _value) {
```

```
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    Transfer(msg.sender, _to, _value);
  }
  function balanceOf(address _owner) constant returns (uint balance) {
    return balances[_owner];
  }
}
```

5. BTC receipt

5.1. In order to connect to a node and receive data from the BTC blockchain, JSON-RPC API 2.0 is used.

```
class BTC
{

   protected $url = null, $is_debug = false, $parameters_structure = 'array';

  // Standard CURL configuration
   protected $curl_options = array(
      CURLOPT_CONNECTTIMEOUT => 8,
      CURLOPT_TIMEOUT => 8
   );

   / / Error processing
   private $httpErrors = array(
      400 => '400 Bad Request',
      401 => '401 Unauthorized',
      403 => '403 Forbidden',
      404 => '404 Not Found',
      405 => '405 Method Not Allowed',
      406 => '406 Not Acceptable',
      408 => '408 Request Timeout',
      500 => '500 Internal Server Error',
      502 => '502 Bad Gateway',
      503 => '503 Service Unavailable'
   );

   // Receive connection parameters

   public function __construct($pUrl = null)
   {
      $this->validate(false === extension_loaded('curl'), 'The curl extension must be loaded for using this class!');
      $this->validate(false === extension_loaded('json'), 'The json extension must be loaded for using this class!');

      // set an url to connect to
      $this->url = $pUrl;
   }

   // Error messages
   private function getHttpErrorMessage($pErrorNumber)
   {
      return isset($this->httpErrors[$pErrorNumber]) ? $this->httpErrors[$pErrorNumber] : null;
   }
```

```
// Debug mode
public function setDebug($pIsDebug)
{
   $this->is_debug = !empty($pIsDebug);
   return $this;
}

// Set structure used for parameters
public function setParametersStructure($pParametersStructure)
{
   if (in_array($pParametersStructure, array('array', 'object')))
   {
      $this->parameters_structure = $pParametersStructure;
   }
   else
   {
      throw new UnexpectedValueException('Invalid parameters structure type.');
   }

   return $this;
}

// Configure additional CURL options
public function setCurlOptions($pOptionsArray)
{
   if (is_array($pOptionsArray))
   {
      $this->curl_options = $pOptionsArray + $this->curl_options;
   }
   else
   {
      throw new InvalidArgumentException('Invalid options type.');
   }
   return $this;
}

// Perform requests and obtain results
public function __call($pMethod, $pParams)
{
   static $requestId = 0;

   // generating uniuqe id per process
   $requestId++;

   // check if given params are correct
   $this->validate(false === is_scalar($pMethod), 'Method name has no scalar value');
   $this->validate(false === is_array($pParams), 'Params must be given as array');

   // send params as an object or an array
   $pParams = ($this->parameters_structure == 'object') ? $pParams[0] : array_values($pParams);
```

```
        // Request (method invocation)
        $request = json_encode(array('jsonrpc' => '2.0', 'method' => $pMethod, 'params' => $pParams, 'id' =>
$requestId));

        // if is_debug mode is true then add url and request to is_debug
        $this->debug('Url: ' . $this->url . "\r\n", false);
        $this->debug('Request: ' . $request . "\r\n", false);

        $responseMessage = $this->getResponse($request);

        // if is_debug mode is true then add response to is_debug and display it
        $this->debug('Response: ' . $responseMessage . "\r\n", true);

        // decode and create array ( can be object, just set to false )
        $responseDecoded = json_decode($responseMessage, true);

        // check if decoding json generated any errors
        $jsonErrorMsg = $this->getJsonLastErrorMsg();
        $this->validate( !is_null($jsonErrorMsg), $jsonErrorMsg . ': ' . $responseMessage);

        // check if response is correct
        $this->validate(empty($responseDecoded['id']), 'Invalid response data structure: ' . $responseMessage);
        $this->validate($responseDecoded['id'] != $requestId, 'Request id: ' . $requestId . ' is different from Response
id: ' . $responseDecoded['id']);
        if (isset($responseDecoded['error']))
        {
            $errorMessage = 'Request have return error: ' . $responseDecoded['error']['message'] . '; ' . "\n" .
                'Request: ' . $request . '; ';

            if (isset($responseDecoded['error']['data']))
            {
                $errorMessage .= "\n" . 'Error data: ' . $responseDecoded['error']['data'];
            }

            $this->validate( !is_null($responseDecoded['error']), $errorMessage);
        }

        return $responseDecoded['result'];
    }

    // Wait for service response

    protected function & getResponse(&$pRequest)
    {
        // do the actual connection
        $ch = curl_init();
        if ( !$ch)
        {
            throw new RuntimeException('Could\'t initialize a cURL session');
        }

        curl_setopt($ch, CURLOPT_URL, $this->url);
```

```
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $pRequest);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-type: application/json'));
    curl_setopt($ch, CURLOPT_ENCODING, 'gzip,deflate');
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    if ( !curl_setopt_array($ch, $this->curl_options))
    {
        throw new RuntimeException('Error while setting curl options');
    }

    $response = curl_exec($ch);

    // check http status code
    $httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    if (isset($this->httpErrors[$httpCode]))
    {
        throw new RuntimeException('Response Http Error - ' . $this->httpErrors[$httpCode]);
    }
    // check for curl error
    if (0 < curl_errno($ch))
    {
        throw new RuntimeException('Unable to connect to '.$this->url . ' Error: ' . curl_error($ch));
    }
    // close the connection
    curl_close($ch);

    return $response;
}

// Validation
protected function validate($pFailed, $pErrMsg)
{
    if ($pFailed)
    {
        throw new RuntimeException($pErrMsg);
    }
}

// Debugging
protected function debug($pAdd, $pShow = false)
{
    static $debug, $startTime;

    // is_debug off return
    if (false === $this->is_debug)
    {
        return;
    }
    // add
    $debug .= $pAdd;
    // get starttime
```

```
    $startTime = empty($startTime) ? array_sum(explode(' ', microtime())) : $startTime;
    if (true === $pShow and !empty($debug))
    {
      // get endtime
      $endTime = array_sum(explode(' ', microtime()));
      // performance summary
      $debug .= 'Request time: ' . round($endTime - $startTime, 3) . ' s Memory usage: ' .
round(memory_get_usage() / 1024) . " kb\r\n";
      echo nl2br($debug);
      // send output imidiately
      flush();
      // clean static
      $debug = $startTime = null;
    }
  }

  // Obtain last error via JSON
  function getJsonLastErrorMsg()
  {
    if (!function_exists('json_last_error_msg'))
    {
      function json_last_error_msg()
      {
        static $errors = array(
          JSON_ERROR_NONE         => 'No error',
          JSON_ERROR_DEPTH        => 'Maximum stack depth exceeded',
          JSON_ERROR_STATE_MISMATCH => 'Underflow or the modes mismatch',
          JSON_ERROR_CTRL_CHAR    => 'Unexpected control character found',
          JSON_ERROR_SYNTAX       => 'Syntax error',
          JSON_ERROR_UTF8         => 'Malformed UTF-8 characters, possibly incorrectly encoded'
        );
        $error = json_last_error();
        return array_key_exists($error, $errors) ? $errors[$error] : 'Unknown error (' . $error . ')';
      }
    }


    if (function_exists('json_last_error'))
    {
      return json_last_error() ? json_last_error_msg() : null;
    }
    else
    {
      return null;
    }
  }
}
```

5.2. BTC blockchain interaction commands

Command	Parameters	Description	Requires unlocked wallet? (v0.4.0+)
`addmultisigaddress`	<nrequired> <'["key","key"]'> [account]	Add a nrequired-to-sign multisignature address to the wallet. Each key is a bitcoin address or hex-encoded public key. If [account] is specified, assign address to [account]. Returns a string containing the address.	N
`addnode`	<node> <add/remove/onetry>	**version 0.8** Attempts add or remove <node> from the addnode list or try a connection to <node> once.	N
`backupwallet`	<destination>	Safely copies wallet.dat to destination, which can be a directory or a path with filename.	N
`createmultisig`	<nrequired> <'["key,"key"]'>	Creates a multi-signature address and returns a json object	
`createrawtransaction`	[{"txid":txid,"vout":n},...] {address:amount,...}	**version 0.7** Creates a raw transaction spending given inputs.	N
`decoderawtransaction`	<hex string>	**version 0.7** Produces a human-readable JSON object for a raw transaction.	N

`dumpprivkey`	<bitcoinaddress>	Reveals the private key corresponding to <bitcoinaddress>	Y
`encryptwallet`	<passphrase>	Encrypts the wallet with <passphrase>.	N
`getaccount`	<bitcoinaddress>	Returns the account associated with the given address.	N
`getaccountaddress`	<account>	Returns the current bitcoin address for receiving payments to this account. If <account> does not exist, it will be created along with an associated new address that will be returned.	N
`getaddednodeinfo`	<dns> [node]	**version 0.8** Returns information about the given added node, or all added nodes (note that onetry addnodes are not listed here) If dns is false, only a list of added nodes will be provided, otherwise connected information will also be available.	
`getaddressesbyaccount`	<account>	Returns the list of addresses for the given account.	N
`getbalance`	[account] [minconf=1]	If [account] is not specified, returns the server's total available balance. If [account] is specified, returns the balance in the account.	N
`getbestblockhash`		**version 0.9** Returns the hash of the best (tip) block in the longest block chain.	N

`getblock`	<hash>	Returns information about the block with the given hash.	N
`getblockcount`		Returns the number of blocks in the longest block chain.	N
`getblockhash`	<index>	Returns hash of block in best-block-chain at <index>; index 0 is the genesis block	N
`getblocknumber`		**Deprecated**. **Removed in version 0.7**. Use getblockcount.	N
`getblocktemplate`	[params]	Returns data needed to construct a block to work on. See BIP_0022 for more info on params.	N
`getconnectioncount`		Returns the number of connections to other nodes.	N
`getdifficulty`		Returns the proof-of-work difficulty as a multiple of the minimum difficulty.	N
`getgenerate`		Returns true or false whether bitcoind is currently generating hashes	N
`gethashespersec`		Returns a recent hashes per second performance measurement while generating.	N
`getinfo`		Returns an object containing various state info.	N
`getmemorypool`	[data]	**Replaced in v0.7.0 with getblocktemplate, submitblock, getrawmempool**	N
`getmininginfo`		Returns an object containing mining-related information: • blocks • currentblocksize • currentblocktx • difficulty • errors • generate	N

		• genproclimit • hashespersec • pooledtx • testnet	
`getnewaddress`	[account]	Returns a new bitcoin address for receiving payments. If [account] is specified payments received with the address will be credited to [account].	N
`getpeerinfo`		**version 0.7** Returns data about each connected node.	N
`getrawchangeaddress`	[account]	**version 0.9** Returns a new Bitcoin address, for receiving change. This is for use with raw transactions, NOT normal use.	N
`getrawmempool`		**version 0.7** Returns all transaction ids in memory pool	N
`getrawtransaction`	<txid> [verbose=0]	**version 0.7** Returns raw transaction representation for given transaction id.	N
`getreceivedbyaccount`	[account] [minconf=1]	Returns the total amount received by addresses with [account] in transactions with at least [minconf] confirmations. If [account] not provided return will include all transactions to all accounts. (version 0.3.24)	N
`getreceivedbyaddress`	<bitcoinaddress> [minconf=1]	Returns the amount received by <bitcoinaddress> in transactions with at least [minconf] confirmations. It correctly handles the case where someone has sent to the address in multiple transactions. Keep in mind that addresses are only ever used for receiving transactions. Works only for addresses in the local wallet, external	N

		addresses will always show 0.	

`gettransaction`	<txid>	Returns an object about the given transaction containing:
		• "amount" : total amount of the transaction
		• "confirmations" : number of confirmations of the transaction
		• "txid" : the transaction ID
		• "time" : time associated with the transaction[1].
		• "details" - An array of objects containing:
		  • "account"
		  • "address"
		  • "category"
		  • "amount"
		  • "fee"
			N

`gettxout`	<txid> <n> [includemempool=true]	Returns details about an unspent transaction output (UTXO)	N
`gettxoutsetinfo`		Returns statistics about the unspent transaction output (UTXO) set	N
`getwork`	[data]	If [data] is not specified, returns formatted hash data to work on: • "midstate" : precomputed hash state after hashing the first half of the data • "data" : block data	N

		• "hash1" : formatted hash buffer for second hash • "target" : little endian hash target If [data] is specified, tries to solve the block and returns true if it was successful.	
`help`	[command]	List commands, or get help for a command.	N
`importprivkey`	<bitcoinprivkey> [label] [rescan=true]	Adds a private key (as returned by dumpprivkey) to your wallet. This may take a while, as a rescan is done, looking for existing transactions. **Optional [rescan] parameter added in 0.8.0.**Note: There's no need to import public key, as in ECDSA (unlike RSA) this can be computed from private key.	Y
`invalidateblock`	<hash>	Permanently marks a block as invalid, as if it violated a consensus rule.	N
`keypoolrefill`		Fills the keypool, requires wallet passphrase to be set.	Y
`listaccounts`	[minconf=1]	Returns Object that has account names as keys, account balances as values.	N
`listaddressgroupings`		**version 0.7** Returns all addresses in the wallet and info used for coincontrol.	N
`listreceivedbyaccount`	[minconf=1] [includeempty=false]	Returns an array of objects containing: • "account" : the account of the receiving addresses • "amount" : total amount received by	N

		addresses with this account • "confirmations" : number of confirmations of the most recent transaction included	
`listreceivedbyaddress`	[minconf=1] [includeempty=false]	Returns an array of objects containing: • "address" : receiving address • "account" : the account of the receiving address • "amount" : total amount received by the address • "confirmations" : number of confirmations of the most recent transaction included To get a list of accounts on the system, execute bitcoind listreceivedbyaddress 0 true	N
`listsinceblock`	[blockhash] [target-confirmations]	Get all transactions in blocks since block [blockhash], or all transactions if omitted. [target-confirmations] intentionally **does not** affect the list of returned transactions, but only affects the returned "lastblock" value.[1]	N
`listtransactions`	[account] [count=10] [from=0]	Returns up to [count] most recent transactions skipping the first [from] transactions for account [account]. If [account] not provided it'll return recent	N

		transactions from all accounts.	
`listunspent`	[minconf=1] [maxconf=999999]	**version 0.7** Returns array of unspent transaction inputs in the wallet.	N
`listlockunspent`		**version 0.8** Returns list of temporarily unspendable outputs	
`lockunspent`	<unlock?> [array-of-objects]	**version 0.8** Updates list of temporarily unspendable outputs	
`move`	<fromaccount> <toaccount> <amount> [minconf=1] [comment]	Move from one account in your wallet to another	N
`sendfrom`	<fromaccount> <tobitcoinaddress> <amount> [minconf=1] [comment] [comment-to]	<amount> is a real and is rounded to 8 decimal places. Will send the given amount to the given address, ensuring the account has a valid balance using [minconf] confirmations. Returns the transaction ID if successful (not in JSON object).	Y
`sendmany`	<fromaccount> {address:amount,...} [minconf=1] [comment]	amounts are double-precision floating point numbers	Y
`sendrawtransaction`	<hexstring>	**version 0.7** Submits raw transaction (serialized, hex-encoded) to local node and network.	N
`sendtoaddress`	<bitcoinaddress> <amount> [comment] [comment-to]	<amount> is a real and is rounded to 8 decimal places. Returns the transaction ID <txid> if successful.	Y
`setaccount`	<bitcoinaddress> <account>	Sets the account associated with the given address. Assigning address that is already assigned to the same account will create a new address associated with that account.	N
`setgenerate`	<generate> [genproclimit]	<generate> is true or false to turn generation on or off. Generation is limited to [genproclimit] processors, -1 is unlimited.	N

`settxfee`	<amount>	<amount> is a real and is rounded to the nearest 0.00000001	N
`signmessage`	<bitcoinaddress> <message>	Sign a message with the private key of an address.	Y
`signrawtransaction`	<hexstring> [{"txid":txid,"vout":n,"scriptPubKey":hex},...] [<privatekey1>,...]	**version 0.7** Adds signatures to a raw transaction and returns the resulting raw transaction.	Y/N
`stop`		Stop bitcoin server.	N
`submitblock`	<hex data> [optional-params-obj]	Attempts to submit new block to network.	N
`validateaddress`	<bitcoinaddress>	Return information about <bitcoinaddress>.	N
`verifymessage`	<bitcoinaddress> <signature> <message>	Verify a signed message.	N
`walletlock`		Removes the wallet encryption key from memory, locking the wallet. After calling this method, you will need to call walletpassphrase again before being able to call any methods which require the wallet to be unlocked.	N
`walletpassphrase`	<passphrase> <timeout>	Stores the wallet decryption key in memory for <timeout> seconds.	N
`walletpassphrasechange`	<oldpassphrase> <newpassphrase>	Changes the wallet passphrase from <oldpassphrase> to <newpassphrase>.	N

6. **ETC receipt**

2.1. In order to connect to a node and receive data from the ETC blockchain, JSON-RPC API 2.0 is used.

```
class Eth

   private $uri = '';

   public function __construct() {
```

```
    $this->uri = $url;
}

// Generate ID request
private function generateId() {
    $chars = array_merge(range('A', 'Z'), range('a', 'z'), range(0, 9));
    $id = '';
    for($c = 0; $c < 5; ++$c)
        $id .= $chars[mt_rand(0, count($chars) - 1)];
    return $id;
}

// Generate request, execute, receive response
public function __call($name, $arguments) {
    $id = $this->generateId();

    $request = array(
        'jsonrpc' => '2.0',
        'method'  => $name,
        'params'  => $arguments,
        'id'      => $id
    );

    $jsonRequest = json_encode($request);

    $ctx = stream_context_create(array(
        'http' => array(
            'method'  => 'POST',
            'header'  => 'Content-Type: application/json\r\n',
            'content' => $jsonRequest
        )
    ));

    $jsonResponse = file_get_contents($this->uri, false, $ctx);

    if ($jsonResponse === false)
        throw new JsonRpcFault('file_get_contents failed', -32603);

    $response = json_decode($jsonResponse);

    if ($response === null)
        throw new JsonRpcFault('JSON cannot be decoded', -32603);

    if ($response->id != $id)
        throw new JsonRpcFault('Mismatched JSON-RPC IDs', -32603);

    if (property_exists($response, 'error'))
        throw new JsonRpcFault($response->error->message, $response->error->code);
    else if (property_exists($response, 'result'))
        return $response->result;
    else
        throw new JsonRpcFault('Invalid JSON-RPC response', -32603);
```

```
    }
}
```

6.2. A complete description of commands used to interface with ETH can be found at https://github.com/ethereum/wiki/wiki/JSON-RPC