```
/*jslint node: true */
"use strict";

//exports.port = 6655;
//exports.myUrl = 'wss://mydomain.com/bb';
exports.bServeAsHub = false;
exports.bLight = false;

exports.storage = 'sqlite';

exports.hub = 'victor.BOYU.org/tn';
exports.deviceName = 'Headless';
exports.permanent_pairing_secret = 'randomstring';
exports.control_addresses = ['DEVICE ALLOWED TO CHAT'];
exports.payout_address = 'WHERE THE MONEY CAN BE SENT TO';
exports.KEYS_FILENAME = 'keys.json';

// this is for runnining RPC service only, see play/rpc_service.js
exports.rpcInterface = '127.0.0.1';
exports.rpcPort = '6552';

console.log('finished headless conf');
```

```
{
  "name": "wallet-online",
  "description": "BOYU online wallet",
  "author": "BOYU",
  "version": "0.1.0",
  "main": "start.js",
  "keywords": [
    "wallet",
    "headless",
    "BOYU"
  ],
  "license": "MIT",
  "repository": {
    "url": "https://github.com/BOYU/wallet-online.git",
    "type": "git"
  },
  "bugs": {
    "url": "https://github.com/BOYU/wallet-online/issues"
  },
  "browser": {
    "request": "browser-request",
    "secp256k1": "secp256k1/js"
  },
  "dependencies": {
    "BOYU-common": "git+https://github.com/BOYU/BOYU-common.git",
    "bitcore-lib": "^0.13.14",
    "bitcore-mnemonic": "~1.0.0",
```

```
        "json-rpc2": "^1.0.2"
    }
}
-----------------------------------------------------------
/*jslint node: true */
"use strict";
var constants = require('BOYU-common/constants.js');
var conf = require('BOYU-common/conf.js');
var db = require('BOYU-common/db.js');
var mutex = require('BOYU-common/mutex.js');

const AUTHOR_SIZE = 3 // "sig"
        + 44    // pubkey
        + 88; // signature

const TRANSFER_INPUT_SIZE = 0 // type: "transfer" omitted
        + 44 // unit
        + 8 // message_index
        + 8; // output_index


function readLeastFundedAddresses(asset, wallet, handleFundedAddresses){
        db.query(
                "SELECT address, SUM(amount) AS total \n\
                FROM my_addresses CROSS JOIN outputs USING(address) \n\
                CROSS JOIN units USING(unit) \n\
                WHERE wallet=? AND is_stable=1 AND sequence='good' AND is_spent=0 AND
"+(asset ? "asset="+db.escape(asset) : "asset IS NULL")+" \n\
                        AND NOT EXISTS ( \n\
                                SELECT * FROM units CROSS JOIN unit_authors USING(unit) \n\
                                WHERE is_stable=0 AND unit_authors.address=outputs.address AND
definition_chash IS NOT NULL \n\
                        ) \n\
                GROUP BY address ORDER BY SUM(amount) LIMIT 15",
                [wallet],
                function(rows){
                        handleFundedAddresses(rows.map(row => row.address));
                }
        );
}

function determineCountOfOutputs(asset, wallet, handleCount){
        db.query(
                "SELECT COUNT(*) AS count FROM my_addresses CROSS JOIN outputs
USING(address) JOIN units USING(unit) \n\
                WHERE wallet=? AND is_spent=0 AND "+(asset ? "asset="+db.escape(asset) : "asset IS
NULL")+" AND is_stable=1 AND sequence='good'",
                [wallet],
                function(rows){
                        handleCount(rows[0].count);
```

```
		}
	);
}

function readDestinationAddress(wallet, handleAddress){
	db.query("SELECT address FROM my_addresses WHERE wallet=? ORDER BY is_change DESC, address_index ASC LIMIT 1", [wallet], rows => {
		if (rows.length === 0)
			throw Error('no dest address');
		handleAddress(rows[0].address);
	});
}

function consolidate(wallet, signer){
	var asset = null;
	mutex.lock(['consolidate'], unlock => {
		determineCountOfOutputs(asset, wallet, count => {
			console.log(count+' unspent outputs');
			if (count <= conf.MAX_UNSPENT_OUTPUTS)
				return unlock();
			let count_to_spend = Math.min(count - conf.MAX_UNSPENT_OUTPUTS + 1, constants.MAX_INPUTS_PER_PAYMENT_MESSAGE - 1);
			readLeastFundedAddresses(asset, wallet, arrAddresses => {
				db.query(
					"SELECT address, unit, message_index, output_index, amount \n\
					FROM outputs \n\
					CROSS JOIN units USING(unit) \n\
					WHERE address IN(?) AND is_stable=1 AND sequence='good' AND is_spent=0 AND "+(asset ? "asset="+db.escape(asset) : "asset IS NULL")+" \n\
						AND NOT EXISTS ( \n\
							SELECT * FROM units CROSS JOIN unit_authors USING(unit) \n\
							WHERE is_stable=0 AND unit_authors.address=outputs.address AND definition_chash IS NOT NULL \n\
						) \n\
					ORDER BY amount LIMIT ?",
					[arrAddresses, count_to_spend],
					function(rows){

						// if all inputs are so small that they don't pay even for fees, add one more large input
						function addLargeInputIfNecessary(onDone){
							var target_amount = 1000 + TRANSFER_INPUT_SIZE*rows.length + AUTHOR_SIZE*arrAddresses.length;
							if (input_amount > target_amount)
								return onDone();
							target_amount += TRANSFER_INPUT_SIZE + AUTHOR_SIZE;
							db.query(
								"SELECT address, unit, message_index, output_index,
```

```
amount \n\
                                                FROM my_addresses \n\
                                                CROSS JOIN outputs USING(address) \n\
                                                CROSS JOIN units USING(unit) \n\
                                                WHERE wallet=? AND is_stable=1 AND sequence='good' \n\
                                                        AND is_spent=0 AND "+(asset ? "asset="+db.escape(asset) : "asset IS NULL")+" \n\
                                                        AND NOT EXISTS ( \n\
                                                                SELECT * FROM units CROSS JOIN unit_authors USING(unit) \n\
                                                                WHERE is_stable=0 AND unit_authors.address=outputs.address AND definition_chash IS NOT NULL \n\
                                                        ) \n\
                                                        AND amount>? AND unit NOT IN(?) \n\
                                                LIMIT 1",
                                                [wallet, target_amount - input_amount, Object.keys(assocUsedUnits)],
                                                large_rows => {
                                                        if (large_rows.length === 0)
                                                                return onDone("no large input found");
                                                        let row = large_rows[0];
                                                        assocUsedAddresses[row.address] = true;
                                                        input_amount += row.amount;
                                                        arrInputs.push({
                                                                unit: row.unit,
                                                                message_index: row.message_index,
                                                                output_index: row.output_index
                                                        });
                                                        onDone();
                                                }
                                        );
                                }

                                var assocUsedAddresses = {};
                                var assocUsedUnits = {};
                                var input_amount = 0;
                                var arrInputs = rows.map(row => {
                                        assocUsedAddresses[row.address] = true;
                                        assocUsedUnits[row.unit] = true;
                                        input_amount += row.amount;
                                        return {
                                                unit: row.unit,
                                                message_index: row.message_index,
                                                output_index: row.output_index
                                        };
                                });
                                addLargeInputIfNecessary(err => {
                                        if (err){
                                                console.log("consolidation failed: "+err);
```

```
                                        return unlock();
                                    }
                                    let arrUsedAddresses = Object.keys(assocUsedAddresses);
                                    readDestinationAddress(wallet, dest_address => {
                                        var composer = require('BOYU-common/composer.js');
                                        composer.composeJoint({
                                            paying_addresses: arrUsedAddresses,
                                            outputs: [{address: dest_address, amount: 0}],
                                            inputs: arrInputs,
                                            input_amount: input_amount,
                                            earned_headers_commission_recipients:
[{address: dest_address, earned_headers_commission_share: 100}],
                                            callbacks: composer.getSavingCallbacks({
                                                ifOk: function(objJoint){
                                                    var network =
require('BOYU-common/network.js');
                                                    network.broadcastJoint(objJoint);
                                                    unlock();
                                                    consolidate(wallet, signer); // do
more if something's left
                                                },
                                                ifError: function(err){
                                                    throw Error('failed to compose
consolidation transaction: '+err);
                                                },
                                                ifNotEnoughFunds: function(err){
                                                    throw Error('not enough funds to
compose consolidation transaction: '+err);
                                                }
                                            }),
                                            signer: signer
                                        });
                                    });
                                });
                            }
                        );
                    });
                });
            });
}

exports.consolidate = consolidate;
```

```
/*jslint node: true */
"use strict";
var fs = require('fs');
var crypto = require('crypto');
var util = require('util');
var constants = require('BOYU-common/constants.js');
var conf = require('BOYU-common/conf.js');
```

```
var objectHash = require('BOYU-common/object_hash.js');
var desktopApp = require('BOYU-common/desktop_app.js');
var db = require('BOYU-common/db.js');
var eventBus = require('BOYU-common/event_bus.js');
var ecdsaSig = require('BOYU-common/signature.js');
var Mnemonic = require('bitcore-mnemonic');
var Bitcore = require('bitcore-lib');
var readline = require('readline');

var appDataDir = desktopApp.getAppDataDir();
var KEYS_FILENAME = appDataDir + '/' + (conf.KEYS_FILENAME || 'keys.json');
var PUBLIC_KEYS_FILENAME = appDataDir + '/publickeys.json';
var wallet_id;
var xPrivKey;

function replaceConsoleLog(){
	var log_filename = conf.LOG_FILENAME || (appDataDir + '/log.txt');
	var writeStream = fs.createWriteStream(log_filename);
	console.log('----------');
	console.log('From this point, output will be redirected to '+log_filename);
	console.log("To release the terminal, type Ctrl-Z, then 'bg'");
	console.log = function(){
		writeStream.write(Date().toString()+': ');
		writeStream.write(util.format.apply(null, arguments) + '\n');
	};
	console.warn = console.log;
	console.info = console.log;
}

function readKeys(onDone){
	console.log('---------------');
	if (conf.control_addresses)
		console.log("remote access allowed from devices: "+conf.control_addresses.join(', '));
	if (conf.payout_address)
		console.log("payouts allowed to address: "+conf.payout_address);
	console.log('---------------');
	fs.readFile(PUBLIC_KEYS_FILENAME, 'utf8', function(err, publicdata){
		if (err){ // first start
			console.log('failed to read public keys');
			throw Error("failed to read public keys");
		}

		var publicdata = JSON.parse(publicdata);
		var devicePrivKey = new
Bitcore.PrivateKey(publicdata.device_priv_key).toBuffer({size:32});
		var strXPubKey = publicdata.pub_key;

		fs.readFile(KEYS_FILENAME, 'utf8', function(err, data){

			if (err){ // first start
```

```
                        console.log('failed to read keys, will gen');

                        var deviceTempPrivKey = crypto.randomBytes(32);
                        var devicePrevTempPrivKey = crypto.randomBytes(32);

                        writeKeys(deviceTempPrivKey, devicePrevTempPrivKey, function(){
                                createWallet(strXPubKey, devicePrivKey, function(){
                                        onDone(strXPubKey, devicePrivKey, deviceTempPrivKey,
devicePrevTempPrivKey);
                                });
                        });

                }
                else{ // 2nd or later start
                        var keys = JSON.parse(data);
                        var deviceTempPrivKey = Buffer(keys.temp_priv_key, 'base64');
                        var devicePrevTempPrivKey = Buffer(keys.prev_temp_priv_key, 'base64');
                        determineIfWalletExists(function(bWalletExists){
                                if (bWalletExists)
                                        onDone(strXPubKey, devicePrivKey, deviceTempPrivKey,
devicePrevTempPrivKey);
                                else{
                                        createWallet(strXPubKey, devicePrivKey, function(){
                                                onDone(strXPubKey, devicePrivKey, deviceTempPrivKey,
devicePrevTempPrivKey);
                                        });
                                }
                        });
                }
        });
});
}

function writeKeys(deviceTempPrivKey, devicePrevTempPrivKey, onDone){
        var keys = {
                temp_priv_key: deviceTempPrivKey.toString('base64'),
                prev_temp_priv_key: devicePrevTempPrivKey.toString('base64')
        };
        fs.writeFile(KEYS_FILENAME, JSON.stringify(keys, null, '\t'), 'utf8', function(err){
                if (err)
                        throw Error("failed to write keys file");
                if (onDone)
                        onDone();
        });
}


function createWallet(strXPubKey, devicePrivKey, onDone){
        var device = require('BOYU-common/device.js');
        device.setDevicePrivateKey(devicePrivKey); // we need device address before creating a wallet
```

```
	var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
	walletDefinedByKeys.createWalletByDevices(strXPubKey, 0, 1, [], 'any walletName', function(wallet_id){
		walletDefinedByKeys.issueNextAddress(wallet_id, 0, function(addressInfo){
			onDone();
		});
	});
}

function isControlAddress(device_address){
	return (conf.control_addresses && conf.control_addresses.indexOf(device_address) >= 0);
}

function readSingleAddress(handleAddress){
	db.query("SELECT address FROM my_addresses WHERE wallet=?", [wallet_id], function(rows){
		if (rows.length === 0)
			throw Error("no addresses");
		if (rows.length > 1)
			throw Error("more than 1 address");
		handleAddress(rows[0].address);
	});
}

function prepareBalanceText(handleBalanceText){
	var Wallet = require('BOYU-common/wallet.js');
	Wallet.readBalance(wallet_id, function(assocBalances){
		var arrLines = [];
		for (var asset in assocBalances){
			var total = assocBalances[asset].stable + assocBalances[asset].pending;
			var units = (asset === 'base') ? ' bytes' : (' of ' + asset);
			var line = total + units;
			if (assocBalances[asset].pending)
				line += ' (' + assocBalances[asset].pending + ' pending)';
			arrLines.push(line);
		}
		handleBalanceText(arrLines.join("\n"));
	});
}

function readSingleWallet(handleWallet){
	db.query("SELECT wallet FROM wallets", function(rows){
		if (rows.length === 0)
			throw Error("no wallets");
		if (rows.length > 1)
			throw Error("more than 1 wallet");
		handleWallet(rows[0].wallet);
	});
}
```

```
function determineIfWalletExists(handleResult){
	db.query("SELECT wallet FROM wallets", function(rows){
		if (rows.length > 1)
			throw Error("more than 1 wallet");
		handleResult(rows.length > 0);
	});
}

function signWithLocalPrivateKey(wallet_id, account, is_change, address_index, text_to_sign, handleSig){
	var path = "m/44'/0'/" + account + "'/"+is_change+"/"+address_index;
	var privateKey = xPrivKey.derive(path).privateKey;
	var privKeyBuf = privateKey.bn.toBuffer({size:32}); // https://github.com/bitpay/bitcore-lib/issues/47
	handleSig(ecdsaSig.sign(text_to_sign, privKeyBuf));
}

var signer = {
	readSigningPaths: function(conn, address, handleLengthsBySigningPaths){
		handleLengthsBySigningPaths({r: constants.SIG_LENGTH});
	},
	readDefinition: function(conn, address, handleDefinition){
		conn.query("SELECT definition FROM my_addresses WHERE address=?", [address], function(rows){
			if (rows.length !== 1)
				throw "definition not found";
			handleDefinition(null, JSON.parse(rows[0].definition));
		});
	},
	sign: function(objUnsignedUnit, assocPrivatePayloads, address, signing_path, handleSignature){
		var buf_to_sign = objectHash.getUnitHashToSign(objUnsignedUnit);
		db.query(
			"SELECT wallet, account, is_change, address_index \n\
			FROM my_addresses JOIN wallets USING(wallet) JOIN wallet_signing_paths USING(wallet) \n\
			WHERE address=? AND signing_path=?",
			[address, signing_path],
			function(rows){
				if (rows.length !== 1)
					throw Error(rows.length+" indexes for address "+address+" and signing path "+signing_path);
				var row = rows[0];
				signWithLocalPrivateKey(row.wallet, row.account, row.is_change, row.address_index, buf_to_sign, function(sig){
					handleSignature(null, sig);
				});
			}
		);
	}
};
```

```
if (conf.permanent_pairing_secret)
	db.query(
		"INSERT "+db.getIgnore()+" INTO pairing_secrets (pairing_secret, is_permanent, expiry_date) VALUES (?, 1, '2038-01-01')",
		[conf.permanent_pairing_secret]
	);

setTimeout(function(){
	readKeys(function(strXPubKey, devicePrivKey, deviceTempPrivKey, devicePrevTempPrivKey){
		var saveTempKeys = function(new_temp_key, new_prev_temp_key, onDone){
			writeKeys(new_temp_key, new_prev_temp_key, onDone);
		};
		//var mnemonic = new Mnemonic(mnemonic_phrase);
		// global
		//xPrivKey = mnemonic.toHDPrivateKey(passphrase);
		//var devicePrivKey = xPrivKey.derive("m/1'").privateKey.bn.toBuffer({size:32});
		// read the id of the only wallet
		readSingleWallet(function(wallet){
			// global
			wallet_id = wallet;
			var device = require('BOYU-common/device.js');
			device.setDevicePrivateKey(devicePrivKey);
			let my_device_address = device.getMyDeviceAddress();
			db.query("SELECT 1 FROM extended_pubkeys WHERE device_address=?", [my_device_address], function(rows){
				if (rows.length > 1)
					throw Error("more than 1 extended_pubkey?");
				if (rows.length === 0)
					return setTimeout(function(){
						console.log('passphrase is incorrect');
						process.exit(0);
					}, 1000);
				require('BOYU-common/wallet.js'); // we don't need any of its functions but it listens for hub/* messages
				device.setTempKeys(deviceTempPrivKey, devicePrevTempPrivKey, saveTempKeys);
				device.setDeviceName(conf.deviceName);
				device.setDeviceHub(conf.hub);
				let my_device_pubkey = device.getMyDevicePubKey();
				console.log("====== my device address: "+my_device_address);
				console.log("====== my device pubkey: "+my_device_pubkey);
				if (conf.permanent_pairing_secret)
					console.log("====== my pairing code: "+my_device_pubkey+"@"+conf.hub+"#"+conf.permanent_pairing_secret);
				if (conf.bLight){
					var light_wallet = require('BOYU-common/light_wallet.js');
					light_wallet.setLightVendorHost(conf.hub);
				}
```

```
                        eventBus.emit('headless_wallet_ready');
                        setTimeout(replaceConsoleLog, 1000);
                });
        });
});
}, 1000);


function handlePairing(from_address){
        var device = require('BOYU-common/device.js');
        prepareBalanceText(function(balance_text){
                device.sendMessageToDevice(from_address, 'text', balance_text);
        });
}

function sendPayment(asset, amount, to_address, change_address, device_address, onDone){
        var device = require('BOYU-common/device.js');
        var Wallet = require('common/walletExchange.js');
        Wallet.sendPaymentFromWallet(
                asset, wallet_id, to_address, amount, change_address,
                [], device_address,
                signWithLocalPrivateKey,
                function(err, unit){
                        if (device_address) {
                                if (err)
                                        device.sendMessageToDevice(device_address, 'text', "Failed to pay: " +
err);
                                else
                                // if successful, the peer will also receive a payment notification
                                        device.sendMessageToDevice(device_address, 'text', "paid");
                        }
                        if (onDone)
                                onDone(err, unit);
                }
        );
}

function sendPaymentExchange(asset, amount, to_address, change_address, onDone){
        var walletExchange = require('./common/wallet_exchange.js');
        walletExchange.sendPaymentFromWalletExchange(
                asset, wallet_id, to_address, amount, change_address, [], null,
                signWithLocalPrivateKey,
                function(err, unit){
                        if (onDone)
                                onDone(err, unit);
                }
        );
}

function sendAllBytesFromAddress(from_address, to_address, recipient_device_address, onDone) {
```

```
	var device = require('BOYU-common/device.js');
	var Wallet = require('BOYU-common/wallet.js');
	Wallet.sendMultiPayment({
		asset: null,
		to_address: to_address,
		send_all: true,
		paying_addresses: [from_address],
		arrSigningDeviceAddresses: [device.getMyDeviceAddress()],
		recipient_device_address: recipient_device_address,
		signWithLocalPrivateKey: signWithLocalPrivateKey
	}, (err, unit) => {
		if(onDone)
			onDone(err, unit);
	});
}

function sendAssetFromAddress(asset, amount, from_address, to_address, recipient_device_address, onDone) {
	var device = require('BOYU-common/device.js');
	var Wallet = require('BOYU-common/wallet.js');
	Wallet.sendMultiPayment({
		fee_paying_wallet: wallet_id,
		asset: asset,
		to_address: to_address,
		amount: amount,
		paying_addresses: [from_address],
		change_address: from_address,
		arrSigningDeviceAddresses: [device.getMyDeviceAddress()],
		recipient_device_address: recipient_device_address,
		signWithLocalPrivateKey: signWithLocalPrivateKey
	}, (err, unit) => {
		if (onDone)
			onDone(err, unit);
	});
}


function issueChangeAddressAndSendPayment(asset, amount, to_address, device_address, onDone){
	if (conf.bSingleAddress){
		readSingleAddress(function(change_address){
			sendPayment(asset, amount, to_address, change_address, device_address, onDone);
		});
	}
	else if (conf.bStaticChangeAddress){
		issueOrSelectStaticChangeAddress(function(change_address){
			sendPayment(asset, amount, to_address, change_address, device_address, onDone);
		});
	}
	else{
		var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
```

```
            walletDefinedByKeys.issueOrSelectNextChangeAddress(wallet_id, function(objAddr){
                sendPayment(asset, amount, to_address, objAddr.address, device_address, onDone);
            });
        }
}

function issueChangeAddressAndSendPaymentExchange(asset, amount, to_address, onDone){
            var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
            walletDefinedByKeys.issueOrSelectNextChangeAddress(wallet_id, function(objAddr){
                sendPaymentExchange(asset, amount, to_address, objAddr.address, onDone);
            });
}

function issueOrSelectNextMainAddress(handleAddress){
        var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
        walletDefinedByKeys.issueOrSelectNextAddress(wallet_id, 0, function(objAddr){
            handleAddress(objAddr.address);
        });
}

function issueNextMainAddress(handleAddress){
        var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
        walletDefinedByKeys.issueNextAddress(wallet_id, 0, function(objAddr){
            handleAddress(objAddr.address);
        });
}

function issueOrSelectStaticChangeAddress(handleAddress){
        var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
        walletDefinedByKeys.readAddressByIndex(wallet_id, 1, 0, function(objAddr){
            if (objAddr)
                return handleAddress(objAddr.address);
            walletDefinedByKeys.issueAddress(wallet_id, 1, 0, function(objAddr){
                handleAddress(objAddr.address);
            });
        });
}

function handleText(from_address, text){

        text = text.trim();
        var fields = text.split(/ /);
        var command = fields[0].trim().toLowerCase();
        var params =["",""];
        if (fields.length > 1) params[0] = fields[1].trim();
        if (fields.length > 2) params[1] = fields[2].trim();

        var walletDefinedByKeys = require('BOYU-common/wallet_defined_by_keys.js');
        var device = require('BOYU-common/device.js');
        switch(command){
```

```
		case 'address':
			if (conf.bSingleAddress)
				readSingleAddress(function(address){
					device.sendMessageToDevice(from_address, 'text', address);
				});
			else
				walletDefinedByKeys.issueOrSelectNextAddress(wallet_id, 0,
function(addressInfo){
					device.sendMessageToDevice(from_address, 'text', addressInfo.address);
				});
			break;

		case 'balance':
			prepareBalanceText(function(balance_text){
				device.sendMessageToDevice(from_address, 'text', balance_text);
			});
			break;

		case 'pay':
			analyzePayParams(params[0], params[1], function(asset, amount){
				if(asset===null && amount===null){
					var msg = "syntax: pay [amount] [asset]";
					msg +=	"\namount: digits only";
					msg +=	"\nasset: one of ", 'bytes', 'blackbytes', ASSET_ID";
					msg +=	"\n";
					msg +=	"\nExample 1: 'pay 12345' pays 12345 bytes";
					msg +=	"\nExample 2: 'pay 12345 bytes' pays 12345 bytes";
					msg +=	"\nExample 3: 'pay 12345 blackbytes' pays 12345 blackbytes";
					msg +=	"\nExample 4: 'pay 12345
qO2JsiuDMh/j+pqJYZw3u82O71WjCDf0vTNvsnntr8o=' pays 12345 blackbytes";
					msg +=	"\nExample 5: 'pay 12345 ASSET_ID' pays 12345 of asset
with ID ASSET_ID";
					return device.sendMessageToDevice(from_address, 'text', msg);
				}

				if (!conf.payout_address)
					return device.sendMessageToDevice(from_address, 'text', "payout
address not defined");

				function payout(amount, asset){
					if (conf.bSingleAddress)
						readSingleAddress(function(address){
							sendPayment(asset, amount, conf.payout_address, address,
from_address);
						});
					else
						// create a new change address or select first unused one
						issueChangeAddressAndSendPayment(asset, amount,
conf.payout_address, from_address);
				};
```

```
                        if(asset!==null){
                                db.query("SELECT unit FROM assets WHERE unit=?", [asset], function(rows){
                                        if(rows.length===1){
                                                // asset exists
                                                payout(amount, asset);
                                        }else{
                                                // unknown asset
                                                device.sendMessageToDevice(from_address, 'text', 'unknown asset: '+asset);
                                        }
                                });
                        }else{
                                payout(amount, asset);
                        }

                });
                break;

        default:
                return device.sendMessageToDevice(from_address, 'text', "unrecognized command");
        }
}

function analyzePayParams(amountText, assetText, cb){
        // expected:
        // amountText = amount; only digits
        // assetText = asset; " -> whitebytes, 'bytes' -> whitebytes, 'blackbytes' -> blackbytes, '{asset-ID}' -> any asset

        if (amountText===''&&assetText==='') return cb(null, null);

        var pattern = /^\d+$/;
       if(pattern.test(amountText)){

                var amount = parseInt(amountText);

                var asset = assetText.toLowerCase();
                switch(asset){
                        case '':
                        case 'bytes':
                                return cb(null, amount);
                        case 'blackbytes':
                                return cb(constants.BLACKBYTES_ASSET, amount);
                        default:
                                // return original assetText string because asset ID it is case sensitive
                                return cb(assetText, amount);
                }
```

```
	}else{
		return cb(null, null);
	}
}

// The below events can arrive only after we read the keys and connect to the hub.
// The event handlers depend on the global var wallet_id being set, which is set after reading the keys

function setupChatEventHandlers(){
	eventBus.on('paired', function(from_address){
		console.log('paired '+from_address);
		if (!isControlAddress(from_address))
			return console.log('ignoring pairing from non-control address');
		handlePairing(from_address);
	});

	eventBus.on('text', function(from_address, text){
		console.log('text from '+from_address+': '+text);
		if (!isControlAddress(from_address))
			return console.log('ignoring text from non-control address');
		handleText(from_address, text);
	});
}

exports.readSingleWallet = readSingleWallet;
exports.readSingleAddress = readSingleAddress;
exports.signer = signer;
exports.isControlAddress = isControlAddress;
exports.issueOrSelectNextMainAddress = issueOrSelectNextMainAddress;
exports.issueNextMainAddress = issueNextMainAddress;
exports.issueOrSelectStaticChangeAddress = issueOrSelectStaticChangeAddress;
exports.issueChangeAddressAndSendPayment = issueChangeAddressAndSendPayment;

exports.issueChangeAddressAndSendPaymentExchange                                    =
issueChangeAddressAndSendPaymentExchange;

exports.setupChatEventHandlers = setupChatEventHandlers;
exports.handlePairing = handlePairing;
exports.handleText = handleText;
exports.sendAllBytesFromAddress = sendAllBytesFromAddress;
exports.sendAssetFromAddress = sendAssetFromAddress;

if (require.main === module)
	setupChatEventHandlers();
```