```
//"service.js" - the backend always-on part, buiilt on Minima using
the minima scripting language. promises are not possible here:

//Load dmax.js
MDS.load("dmax.js");

/* eslint-disable no-undef */
var LISTINGSTABLE = 'LISTING';
var SETTINGSTABLE = 'SETTINGS';
var APPLICATION_NAME = 'stampd';

const SERVER_ADDRESS =
'MAX#0x30819F300D06092A864886F70D010101050003818D0030818902818100B4D
30A8C0A1D1EA48DE04CA803D0A9D75453E6E9732D6575F4A06330EEF733DF7DF496E
33BA46BB195C5826ED32264FE69E4C809C544F9859CF543932CB5A6ED347052F33B5
0F3A2D424C1BE384CA9B5E0DD0DFFECE2286E4D0311CDF30F3B5E343369CDA8AC8E5
DBB1B2EDADD7E9053B9393F4AA021224BF4AA41568403D82D0203010001#MxG18HGG
6FJ038614Y8CW46US6G20810K0070CD00Z83282G60G1C0ANS2ENGJEFBYJM2SCQFR3U
3KBJNP1WS9B0KG1Z2QG5T68S6N2C15B2FD7WHV5VYCKBDW943QZJ9MCZ03ESQ0TDR86P
EGUFRSGEJBANN91TY2RVPQTVQSUP26TNR399UE9PPJNS75HJFTM4DG2NZRUDWP06VQHH
VQSGT9ZFV0SCZBZDY0A9BK96R7M4Q483GN2T04P30GM5C10608005FHRRH4@78.141.2
38.36:9001'
const SERVER_WALLET =
'MxG083U3R8H31Z30H7Z6T0KM1Z9GJ978NCDGBJU42JTSZS18ZW4GFDF43EH519U'

//switch on and off logs
var logs = true;

MDS.init(function (msg) {
    switch (msg.event) {
        case "inited":
            setup();
            break;
        case "MAXIMA":
            //if (logs) { MDS.log("MAXIMA EVENT received: "); }
            processMaximaEvent(msg);
            break;
        case "NEWBALANCE":
            //check coins against unconfirmed/pending payemnts
            //if (logs) { MDS.log("NEWBALANCE EVENT received: "); }
            processNewBalanceEvent();
            break;
        case "MINIMALOG":
            processMinimaLogEvent(msg.data);
            break;
        default:
            //if (logs) { MDS.log(JSON.stringify(msg)); }
            break;
    }
});
/*
* Register the store name and public key, if no store then create it
* store name = maxima contact name
* store id = current public key
```

```
*/
function setup() {
    let pk = '';
    getPublicKey(function (res) {
        pk = res;
    });
    let hostName = getMaximaContactName();
    let mls = getMLS();
    const permanentAddress = `MAX#${pk}#${mls}`;

    if (logs) { MDS.log(`Permanent Address: ${permanentAddress}`) }

    //create listing table
    createListingTable(function (result) {
        if (logs) { MDS.log('Listing table created or exists') }
    });

    //add location description column
    addLocationDescriptionColumn(function (result) {
        if (logs) { MDS.log('Added location description Column: ' +
result) }
    });

    //add location description column
    addUnconfirmedCoinColumn(function (result) {
        if (logs) { MDS.log('Added unconfirmed coin Column: ' +
result) }
    });

    addPendingUIDColumn(function (result) {
        if (logs) { MDS.log('Added pending UID Column: ' + result) }
    });


    //create settings table
    createSettingsTable(function (result) {
        if (logs) { MDS.log('Settings table created or exists:' +
hostName); }

        //check if store exists
        createHost(hostName, permanentAddress);
        if (logs) { MDS.log('Local hosting info stored in
database') }

    });

}

/*
************************************************** PROCESS EVENTS
**************************************************
*/

function processMinimaLogEvent(data) {
```

```
    //if we have a new spent coin
    if (data.message.includes("NEW Spent Coin")) {
        //get all pending transactions
        getListingsWithPendingUID(function (listings) {
            if (logs) { MDS.log('Listings Found: ' +
JSON.stringify(listings)); }
            MDS.log('listings is a : ' + typeof (listings));
            if (listings.length > 0) {
                listings.forEach(function (listing) {
                    let cmd = `checkpending uid:$
{listing.pendinguid}`;
                    MDS.cmd(cmd, function (response) {
                        if (response.status === true) {
                            const data = {
                                "type": "purchase_receipt",
                                "buyer_message":
listing.buyer_message,
                                "listing_id": listing.listing_id,
                                "transmission_type":
listing.transmission_type,
                                "buyer_name": listing.buyer_name
                            }
                            MDS.log('SUPER LISTING: ' +
JSON.stringify(listing));
                            sendMaximaMessage(data,
listing.created_by_pk, "stampd", function (result) {
                                if (logs) { MDS.log('Message sent to
seller: ' + JSON.stringify(result)) }
                            });
                        }
                    });
                });
            } else { if (logs) { MDS.log('No listings with pending
UID') } }
        });
    }
}


function processMaximaEvent(msg) {

    //Is it for us.. ?
    if (msg.data.application === "stampd") {


        //Get the data packet..
        var datastr = msg.data.data;
        if (datastr.startsWith("0x")) {
            datastr = datastr.substring(2);
        }
        if (logs) {
            MDS.log("----");
            MDS.log(JSON.stringify(msg.data.data));
            MDS.log("----");
```

```
        }

        var jsonstr = "";
        MDS.cmd("convert from:HEX to:String data:" + msg.data.data,
function (resp) {

MDS.log(JSON.stringify(resp.response.conversion).replace(/'/g, ""));
            jsonstr =
JSON.parse(resp.response.conversion.replace(/'/g, ""));
        });

        if (logs) {
            //And create the actual JSON
            MDS.log(JSON.stringify(jsonstr));
            var entity = jsonstr;
            MDS.log("======");
            MDS.log(entity.type);
            MDS.log("======");
        }

        //determine what type of message you're receiving
        switch (entity.type) {
            case 'availability_check':
                //buyer checks listing availability with seller
                processAvailabilityCheck(entity);
                break;
            case 'availability_response':
                //seller sends status of listing to buyer
                processAvailabilityResponse(entity);
                break;
            case 'listing':
                //a contact has shared a listing with you
                processListing(entity);
                break;
            case 'purchase_receipt':
                //buyer sends seller their address and coin id
                processPurchaseReceipt(entity);
                break;
            case 'collection_confirmation':
                //buyer sends seller their number to arrange
collection
                processCollectionConfirmation(entity);
                break;
            case 'cancel_collection':
                //buyer sends seller their number to arrange
collection
                processCancelCollection(entity);
                break;
            default:
                if (logs) { MDS.log(entity); }
        }
    } else if (msg.data.application === "dmax") {
        //Is it for dmax...
```

```
        //Convert the data..
        MDS.cmd("convert from:HEX to:String data:" + msg.data.data,
function (resp) {

            //And create the actual JSON
            //TODO: Check that conversion is part of the response
            var json = JSON.parse(resp.response.conversion);

            //What type is this..
            var type = json.type;



            if (type === "P2P_RESPONSE") {
                MDS.log("P2P_RESPONSE received:" +
JSON.stringify(json));
                //create two variables for the amount and the
p2pidentity
                var amount = json.data.amount;
                var p2pIdentity = json.data.p2pidentity;

                //set the static MLS
                setStaticMLS(p2pIdentity, function (resp) {
                    MDS.log("Set static MLS");

                    //send amount of money to the server wallet
                    sendMinima(amount, SERVER_WALLET, function
(coinId, error) {
                        if (error) {
                            MDS.log("Error sending Minima: " +
error);
                            //update frontend document with error
                            return;
                        }
                        MDS.log("Sent Minima");
                        //coinID is returned

                        //get the client public key
                        getPublicKey(function (clientPK) {
                            MDS.log("Got public key");

                            //send via maxima coinID, clientPK
                            sendMaximaMessage({ "type":
"PAY_CONFIRM", "data": { "status": "OK", "coin_id": coinId,
"client_pk": clientPK, "amount": amount } }, SERVER_ADDRESS, "dmax",
function (msg) {
                                MDS.log("Sent response to " +
SERVER_ADDRESS);
                            });
                        });
                    });
                });
            }
```

```
            else if (type === "EXPIRY_DATE") {
                //replace user message with the expiry date and
permanent maxima address
                var expiryDate = json.data.expiry_date;
                var permanentAddress = json.data.permanent_address;

                document.getElementById("js-main").innerHTML = `Your
MLS will expire on ${expiryDate}. Your permanent address is $
{permanentAddress}.`;
            } else {
                MDS.log("INVALID message type in dmax server: " +
type);
            }
        });
    }
}
function processListing(entity) {
    if (logs) { MDS.log(`processing listing...${entity}`) }

    //check it's not one of your own
    const host = getHost();
    if (host.pk === entity.created_by_pk) {
        return;
    }

    createListing({
        listingId: entity.listing_id,
        title: entity.title,
        price: entity.price,
        createdByPk: entity.created_by_pk,
        createdByName: entity.created_by_name,
        sentByName: entity.sent_by_name,
        sentByPk: entity.sent_by_pk,
        walletAddress: entity.wallet_address,
        createdAt: entity.created_at,
        image: entity.image,
        description: entity.description,
        collection: entity.collection,
        delivery: entity.delivery,
        location: entity.location,
        locationDescription: entity.location_description,
        shippingCost: entity.shipping_cost,
        shippingCountries: entity.shipping_countries
    });
    if (logs) { MDS.log(`Listing ${entity.title} added!`); }
}


function generateCode(length) {
    let result = '';
    const characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    const charactersLength = characters.length;
```

```
    let counter = 0;
    while (counter < length) {
        result += characters.charAt(Math.floor(Math.random() *
charactersLength));
        counter += 1;
    }
    return result;
}

function processAvailabilityCheck(entity) {
    MDS.log(`received availability check for listing: $
{JSON.stringify(entity)}`);

    const data = {
        "type": "availability_response",
        "status": "unavailable",
        "listing_id": entity.listing_id
    }

    try {
        //is listing available
        const listingStatus = getStatus(entity.listing_id);
        MDS.log(`status of listing is: $
{JSON.stringify(listingStatus)}`);

        if (listingStatus) {
            data.status = listingStatus;
            //generate unique identifier for transaction
            const purchaseCode = generateCode(20);
            //generate purchase code and send to buyer
            MDS.log(`generating purchase code: ${purchaseCode}`);
            data.purchase_code = purchaseCode;

            MDS.log(`sending the responose to buyer..`);
            send(data, entity.buyer_pk);

            MDS.log(`updating listing in db to pending`);
            updateListing(entity.listing_id, { "purchase_code":
purchaseCode });

            //if listing available change to pending to stop other
users buying it
            if (listingStatus === 'available') {
                updateListing(entity.listing_id, { "status":
"pending" });
            }
        };
    } catch (error) {
        if (logs) { MDS.log(`There was an error processing
availability check: ${JSON.stringify(error)}`); }
    };
}

function processNewBalanceEvent() {
```

```
    if (logs) { MDS.log("Processing new balance event"); }
    getHistoryTransactions(function (transactions) {
        if (transactions.length > 0) {

            getListingsWithUnconfirmedCoins(function (listings) {
                if (logs) {
                    MDS.log(`Found ${JSON.stringify(listings.count)}
listings with unconfirmed coins`);
                }

                if (listings.length > 0) {
                    //loop therough coins and check each one against
the coins list returned by mds.cmd('coins')
                    listings.forEach(function (listing) {

                        if (logs) { MDS.log(`Transactions found: $
{JSON.stringify(transactions.length)}`); }
                        confirmCoin(listing.purchase_code,
transactions, function (coin) {

                            if (coin) {
                                MDS.log(`About to check coin amount:
${JSON.stringify(coin)}`);
                                //if coin is confirmed then check
the amount of the coin matches the amount on the listing
                                var totalCost =
parseInt(parseInt(listing.price) + (listing.shipping_cost ?
parseInt(listing.shipping_cost) : 0));
                                MDS.log("listing amount: " +
totalCost);
                                MDS.log("coin amount: " +
coin.amount);
                                MDS.log("total cost: " + totalCost);
                                if (parseInt(coin.amount) ===
totalCost) {
                                    updateListing(id,
                                        {
                                            'buyer_message':
entity.buyer_message,
                                            'status': 'sold',
                                            'unconfirmed_coin':
false,
                                            'notification': true,
                                            'buyer_name':
entity.buyer_name
                                        });
                                } else {
                                    MDS.log("Coin amount does not
match listing amount");
                                }
                            }
                        });
                    });
                }
```

```
            });
        }
    });
}



/*
*   Process a purchase receipt
*   @param {object} entity - the entity object
*/
function processPurchaseReceipt(entity) {
    var id = entity.listing_id;
    if (logs) { MDS.log(`Message received for purchased listing: $
{JSON.stringify(entity)}`); }

    getHistoryTransactions(function (transactions) {
        if (transactions.length > 0) {

            getListingById(id, function (listing) {
                if (logs) { MDS.log(`Listing found: $
{JSON.stringify(listing)}`); }

                if (listing) {
                    if (logs) { MDS.log(`Transactions found: $
{JSON.stringify(transactions.length)}`); }
                    confirmCoin(listing.purchase_code, transactions,
function (coin) {

                        if (coin) {
                            //if coin found check amount of coin
matches amount on listing
                            MDS.log(`About to check coin amount: $
{JSON.stringify(coin)}`);
                            //if coin is confirmed then check the
amount of the coin matches the amount on the listing
                            var totalCost =
parseInt(parseInt(listing.price) + (listing.shipping_cost ?
parseInt(listing.shipping_cost) : 0));
                            MDS.log("listing amount: " + totalCost);
                            MDS.log("coin amount: " + coin.amount);
                            MDS.log("total cost: " + totalCost);
                            if (parseInt(coin.amount) === totalCost)
{
                                MDS.log("coin amount matches total
cost");
                                updateListing(id,
                                    {
                                        'buyer_message':
entity.buyer_message,
                                        'status': 'completed',
                                        'notification': true,
                                        'buyer_name':
entity.buyer_name
```

```
                                    });
                                } else {
                                    MDS.log("coin amount does not match
total cost");
                                }
                            } else {
                                updateListing(id, {
                                    'buyer_message':
entity.buyer_message,
                                    'unconfirmed_coin': true,
                                    'buyer_name': entity.buyer_name
                                });
                                //check for coin when rew balance comes
in
                            }

                        });
                    }
                });
            }
        });
}

/*
* Returns the coin that has the purchase code
* @param {string} purchaseCode - the purchase code
* @param {array} transactions - the transactions array
* @param {function} callback - the callback function
*/
function confirmCoin(purchaseCode, transactions, callback) {
    if (logs) { MDS.log(`Confirming coin for purchase code: $
{purchaseCode}`); }
    var response = null;
    transactions.forEach(function (transaction) {
        if (logs) { MDS.log(`Transaction: $
{JSON.stringify(transaction.body.txn.state)}`); }
        if (transaction.body.txn.state[0]) {
            if (transaction.body.txn.state[0].data) {
                MDS.log(`Transaction: $
{JSON.stringify(transaction.body.txn.state[0].data)}`);

                if (transaction.body.txn.state[0].data === "[" +
purchaseCode + "]") {
                    if (logs) { MDS.log(`Coin confirmed: $
{JSON.stringify(transaction.body.txn.outputs[0].coinid)}`); }
                    response = transaction.body.txn.outputs[0];
                }
            }
        }
    });
    callback(response);
}

function processAvailabilityResponse(entity) {
```

```
    if (logs) { MDS.log(`processing availability response...$
{JSON.stringify(entity)}`); }
    updateListing(entity.listing_id, { "status": entity.status,
"purchase_code": entity.purchase_code });
}

function processCollectionConfirmation(entity) {
    if (logs) { MDS.log(`Message received for collection of listing,
updating..`); }
    updateListing(entity.listing_id, {
        'buyer_message': entity.message,
        'status': 'sold',
        'notification': true,
        'transmission_type': entity.transmission_type,
        'buyer_name': entity.buyer_name
    });
}

function processCancelCollection(entity) {
    //TODO: rewrite function that updates the listing all at once
instead of hitting database x times
    if (logs) { MDS.log(`Message received for cancelling
collection`); }
    const listing = getListingById(entity.listing_id);
    if (listing.buyer_name === entity.buyer_name) {
        updateListing(entity.listing_id, { 'status': 'available' })
    } else {
        if (logs) { MDS.log("buyer name not the same as on listing
so cancel averted!"); }
    }
}


/*
******************************************************** GET FUNCTIONS
********************************************************
*/
function getStatus(listingId) {
    var st = '';
    MDS.sql(`SELECT "status" FROM ${LISTINGSTABLE} WHERE
"listing_id"='${listingId}';`, function (res) {
        if (res) {
            if (logs) { MDS.log(`Response from get status is: $
{JSON.stringify(res)}`); }
            st = res.rows[0].status;
        }
        else {
            if (logs) { MDS.log(`MDS.SQL ERROR, could get status of
listing ${res.error}`); }
        }
    });
    return st;
}
```

```
function getHost() {
    var host = '';
    MDS.sql(`select "pk", "name" FROM SETTINGS;`, function (res) {
        if (res.status && res.count === 1) {
            host = res.rows[0];
        } else if (res.error.includes('Table \"SETTINGS\" not
found')) {
            return "No Tables Created";
        } else {
            return res.error;
        }
    });
    return host;
}
/*
*   Return coin if found
*/
function getCoinById(coinid, callback) {
    MDS.cmd(`coins coinid:${coinid}`, function (res) {
        if (res.status) {
            if (callback) {
                callback(res.response[0]);
            }
        } else {
            if (logs) { MDS.log(`MDS.cmd ERROR, could not get coin $
{res.error}}`); }
            return Error(res.error);
        }
    });
}
/*
*   Return coin if found
*/
function getCoinByIdAndAddress(coinid, address, callback) {
    MDS.cmd(`coins coinid:${coinid} address:${address}`, function
(res) {
        if (res.status) {
            if (callback) {
                callback(res.response[0]);
            }
        } else {
            if (logs) { MDS.log(`MDS.cmd ERROR, could not get coin $
{res.error}}`); }
            return Error(res.error);
        }
    });
}

function getListingById(id, callback) {
    var listings = '';
    MDS.sql(`SELECT * FROM ${LISTINGSTABLE} WHERE "listing_id"='$
{id}';`, function (res) {
        if (res.status) {
            if (res.count > 1) {
```

```
                if (logs) { MDS.log(`More than one listing with id ${id}`); }
                return null;
            } else {
                if (callback) {
                    callback(res.rows[0]);
                } else {
                    listings = res.rows[0];
                }
            }
        } else {
            if (logs) { MDS.log(`MDS.SQL ERROR, could get listing by Id ${res.error}`); }
            return null;
        }
    });
    return listings;
}

function getMLS() {
    var mls = '';
    MDS.cmd('maxima', function (res) {
        if (logs) { MDS.log('Get MLS: ' + JSON.stringify(res)); }
        if (res.status) {
            mls = res.response.mls;
        } else {
            return Error(`Couldn't fetch maxima public key ${res.error}`);
        }
    })
    return mls;
}

function getMaximaContactName() {
    var mcn = '';
    MDS.cmd('maxima', function (res) {
        if (res.status) {
            mcn = res.response.name;
        } else {
            return Error(`Couldn't fetch maxima contact name ${res.error}`);
        }
    })
    return mcn;
}

/*
* Return all listings with unconfirmed coins
*/
function getListingsWithUnconfirmedCoins(callback) {
    if (logs) { MDS.log("Getting unconfirmed coins"); }
    MDS.sql(`SELECT * FROM "${LISTINGSTABLE}" WHERE "unconfirmed_coin" IS true`, function (result) {
        if (result && callback) {
```

```
            callback(result);
        } else {
            callback([]);
            if (logs) { MDS.log("No unconfirmed coins found"); }
        }
    });
}

function getListingsWithPendingUID(callback) {
    if (logs) { MDS.log("Getting pending listings"); }
    MDS.sql(`SELECT * FROM "${LISTINGSTABLE}" WHERE "pendinguid" IS
NOT NULL`, function (result) {
        if (result && callback) {
            callback(result.rows);
        } else {
            callback([]);
            if (logs) { MDS.log("No pending listings found"); }
        }
    });
}



function getHistoryTransactions(callback) {
    if (logs) { MDS.log('Search history for purchase code'); }
    MDS.cmd(`history`, function (res) {
        if (res.status === true) {
            callback(res.response.txpows);
        }
        else { callback([]); }
    });
}
/*
******************************************************* DATABASE
FUNCTIONS *******************************************************
*/

function createListingTable(callback) {
    const Q = `create table if not exists ${LISTINGSTABLE} (
            "listing_id" varchar(666) primary key,
            "title" varchar(50) NOT NULL,
            "price" INT NOT NULL,
            "created_by_pk" varchar(640) NOT NULL,
            "created_by_name" char(50),
            "sent_by_pk" varchar(640),
            "sent_by_name" char(50),
            "created_at" int not null,
            "wallet_address" varchar(80) not null,
            "status" char(12) not null default 'available',
            "buyer_message" varchar(1000),
            "buyer_name" char(50),
            "buyer_pk" varchar(330),
            "purchase_code" varchar(30),
            "pendinguid" varchar(34) default null,
```

```
            "coin_id" varchar(80),
            "unconfirmed_coin" boolean default false,
            "notification" boolean default false,
            "collection" boolean default false,
            "delivery" boolean default false,
            "image"  varchar(max),
            "description" varchar(1500),
            "location" varchar(50),
            "location_description" varchar(150),
            "shipping_cost" int,
            "shipping_countries" varchar(150),
            "transmission_type" varchar(10),
            constraint UQ_listing_id unique("listing_id")
            )`;

    MDS.sql(Q, function (res) {
        if (logs) { MDS.log(`MDS.SQL, ${Q}`); }
        MDS.log(`Creating listing tables ${res.status}`)
        if (res.status && callback) {
            callback(true);
        } else {
            return Error(`Creating listing tables ${res.error}`);
        }
    })
}

function createSettingsTable(callback) {
    const Q = `create table if not exists ${SETTINGSTABLE} (
            "pk" varchar(640),
            "name" varchar(50),
            CONSTRAINT AK_name UNIQUE("name"),
            CONSTRAINT AK_pk UNIQUE("pk")
            )`;

    MDS.sql(Q, function (res) {
        if (logs) { MDS.log(`MDS.SQL, ${Q}`); }
        if (res.status && callback) {
            callback(true);
        } else {
            return Error(`${res.error}`);
        }
    })
}
function createHost(name, pk) {
    let fullsql = `insert into ${SETTINGSTABLE}("name", "pk")
values('${name}', '${pk}');`;
    if (logs) { MDS.log(`Host added to settings table: ${name}`); }
    MDS.sql(fullsql, (res) => {
        if (res.status) {
            return true;
        } else {
            return Error(res.error);
        }
    });
```

```
}

function createListing({
    title,
    price,
    createdByPk,
    createdByName,
    listingId,
    sentByName,
    sentByPk,
    walletAddress,
    createdAt,
    image,
    description,
    collection,
    delivery,
    location,
    locationDescription,
    shippingCost,
    shippingCountries
}) {
    const randomId = Math.trunc(Math.random() * 10000000000000000);
    let pk = '';
    getPublicKey(function (res) {   // get the public key of the
user
        if (res) {
            pk = res;
        } else {
            return Error(`Couldn't fetch public key ${res.error}`);
        }
    });

    const id = `${randomId}${pk}`;
    if (logs) { MDS.log(`the id for the listing is: ${id}`); }
    const timestamp = Math.floor(Date.now() / 1000);

    let fullsql = `insert into ${LISTINGSTABLE}
        (
            "listing_id",
            "title",
            "price",
            "collection",
            "delivery",
            "created_by_pk",
            "created_by_name",
            ${sentByName ? '"sent_by_name",' : ''}
            ${sentByPk ? '"sent_by_pk",' : ''}
            "wallet_address",
             ${sentByPk ? '"status",' : ''}
            "image",
            "description",
            ${location ? '"location",' : ''}
            ${locationDescription ? '"location_description",' : ''}
            ${shippingCost ? '"shipping_cost",' : ''}
```

```
            ${shippingCountries ? '"shipping_countries",' : ''}
            "created_at"
        )

        values(
            ${listingId ? `'${listingId}',` : `'${id}',`}
            '${title}',
            '${price}',
            '${collection}',
            '${delivery}',
            '${createdByPk}',
            '${createdByName}',
            ${sentByName ? `'${sentByName}',` : ''}
            ${sentByPk ? `'${sentByPk}',` : ''}
            '${walletAddress}',
            ${sentByPk ? `'unchecked',` : ''}
            '${image}',
            '${description}',
            ${location ? `'${location}',` : ''}
            ${locationDescription ? `'${locationDescription}',` :
''}
            ${shippingCost ? `'${shippingCost}',` : ''}
            ${shippingCountries ? `'${shippingCountries}',` : ''}
            ${createdAt ? `'${createdAt}'` : `'${timestamp}'`}
        );`;
    MDS.sql(fullsql, (res) => {
        if (logs) { MDS.log(`MDS.SQL, ${fullsql}`); }
        if (res.status) {
            return listingId ? listingId : id;
        } else {
            if (logs) { MDS.log(`MDS.SQL ERROR, could not create
listing ${res.error}}`); }
            return Error(res.error);
        }
    });
}

function addLocationDescriptionColumn(callback) {
    const Q = `alter table ${LISTINGSTABLE} add column if not exists
"location_description" varchar(150);`;
    MDS.sql(Q, function (res) {
        if (logs) { MDS.log(`MDS.SQL, ${Q}`); }
        if (res.status) {
            callback(true)
        } else {
            callback(Error(`Adding location_description column to
listing table ${res.error}`));
        }
    })
}

function addUnconfirmedCoinColumn(callback) {
    const Q = `alter table ${LISTINGSTABLE} add column if not exists
"unconfirmed_coin" boolean default false;`;
```

```
    MDS.sql(Q, function (res) {
        if (logs) { MDS.log(`MDS.SQL, ${Q}`); }
        if (res.status) {
            callback(true)
        } else {
            callback(Error(`Adding unconfirmed_coin column to
listing table ${res.error}`));
        }
    })
}

function addPendingUIDColumn(callback) {
    const Q = `alter table ${LISTINGSTABLE} add column if not exists
"pendinguid" varchar(34) default null;`;
    MDS.sql(Q, function (res) {
        if (logs) { MDS.log(`MDS.SQL, ${Q}`); }
        if (res.status) {
            callback(true)
        } else {
            callback(Error(`Adding pendinguid column to listing
table ${res.error}`));
        }
    })
}

function updateListing(listingId, data) {
    var formattedData = '';

    var keys = Object.keys(data);
    var totalKeys = keys.length;

    for (var i = 0; i < totalKeys; i++) {
        var key = keys[i];

        // Check if it's the last iteration
        if (i === totalKeys - 1) {
            formattedData += `"${key}"='${data[key]}'`;
        } else {
            formattedData += `"${key}"='${data[key]}',`;
        }
    }
    MDS.sql(`UPDATE ${LISTINGSTABLE} SET ${formattedData} WHERE
"listing_id"='${listingId}';`, function (res) {
        if (res.status) {
            if (logs) { MDS.log(`MDS.SQL, UPDATE ${LISTINGSTABLE}
SET ${formattedData} WHERE "listing_id"='${listingId}';`); }
            return res;
        } else {
            if (logs) { MDS.log(`MDS.SQL ERROR, could get update
listing ${res.error}`); }
            return false;
        }
    });
}
```

```
/*
****************************************************** MAXIMA
******************************************************
*/
function send(data, address) {

    //before sending append version number of application

    //Convert to a string..
    var datastr = JSON.stringify(data);
    MDS.log(datastr);
    var hexstr = "";
    const funcC = `convert from:String to:HEX data:'$
{String(datastr)}'`;
    if (logs) { MDS.log(funcC); }
    MDS.cmd(funcC, function (resp) {
        if (logs) { MDS.log(JSON.stringify(resp)); }
        hexstr = resp.response.conversion;
    });
    //And now convert to HEX
    //const hexstr = "0x" + utf8ToHex(datastr).toUpperCase().trim();

    //Create the function..
    let fullfunc = '';
    if (address.includes('@')) {
        fullfunc = `maxima action:send to:${address} poll:true
application:${APPLICATION_NAME} data:${hexstr}`;
    } else {
        fullfunc = `maxima action:send publickey:${address}
poll:true application:${APPLICATION_NAME} data:${hexstr}`;
    }

    //Send the message via Maxima!..
    MDS.cmd(fullfunc, function (resp) {
        if (resp.status === false) {
            if (logs) { MDS.log(JSON.stringify(resp)); }
            return false;
        } else if (resp.response.delivered === false) {
            if (logs) { MDS.log(JSON.stringify(resp)); }
            return false;
        } else if (resp.status === true) {
            return true;
        }
    });
}

//MDS.js: the file written by minima, for minima. this helps run our
software, its a bit like the operating system of the blockchain

/**
* MDS JS lib for MiniDAPPs..
*
* @spartacusrex
```

```
*/

/**
 * The MAIN Minima Callback function
 */
var MDS_MAIN_CALLBACK = null;

/**
 * Main MINIMA Object for all interaction
 */
var MDS = {

        //RPC Host for Minima
        mainhost : "",

        //The MiniDAPP UID
        minidappuid : "",

        //Is logging RPC enabled
        logging : false,

        //When debuggin you can hard set the Host and port
        DEBUG_HOST : null,
        DEBUG_PORT : -1,

        //An allowed TEST Minidapp ID for SQL - can be overridden
        DEBUG_MINIDAPPID : "0x00",

        /**
         * Minima Startup - with the callback function used for all
Minima messages
         */
        init : function(callback){
                //Log a little..
                MDS.log("Initialising MDS..");

                //Is logging enabled.. via the URL
                if(MDS.form.getParams("MDS_LOGGING") != null){
                        MDS.logging = true;
                }

                //Get the host and port..
                var host = window.location.hostname;
                var port =  Math.floor(window.location.port);

                //Get ther MiniDAPP UID
                MDS.minidappuid = MDS.form.getParams("uid");

                //HARD SET if debug mode - running from a file
                if(MDS.DEBUG_HOST != null){
                        MDS.log("DEBUG Settings Found..");

                        host=MDS.DEBUG_HOST;
                        port=MDS.DEBUG_PORT;
```

```
                }

                if(MDS.minidappuid == null){
                        MDS.minidappuid = MDS.DEBUG_MINIDAPPID;
                }
                // env overrides
                if (window.DEBUG) {
                        host = window.DEBUG_HOST;
                        port = Math.floor(window.DEBUG_PORT);
                        MDS.minidappuid = window.DEBUG_UID;
                }
                //Is one specified..
                if(MDS.minidappuid == "0x00"){
                        MDS.log("No MiniDAPP UID specified.. using
test value");
                }

                //The ports..
                var mainport    = port+1;

                MDS.log("MDS FILEHOST  :
https://"+host+":"+port+"/");

                MDS.mainhost    = "https://"+host+":"+mainport+"/";
                MDS.log("MDS MAINHOST : "+MDS.mainhost);

                //Store this for poll messages
                MDS_MAIN_CALLBACK = callback;

                //Start the Long Poll listener
                PollListener();

                //And Post a message
                MDSPostMessage({ "event": "inited" });
        },

        /**
         * Log some data with a timestamp in a consistent manner to
the console
         */
        log : function(output){
                console.log("Minima @ "+new Date().toLocaleString()
+" : "+output);
        },

        /**
         * Notify the User - on Phone it pops up in status bar. On
desktop appears in Logs
         */
        notify : function(output){
                //Send via POST

httpPostAsync(MDS.mainhost+"notify?"+"uid="+MDS.minidappuid,
output);
```

```
        },

        /**
         * Cancel this MiniDAPPs notification
         */
        notifycancel : function(){
                //Send via POST

httpPostAsync(MDS.mainhost+"notifycancel?"+"uid="+MDS.minidappuid,
"*");
        },

        /**
         * Runs a function on the Minima Command Line - same format
as MInima
         */
        cmd : function(command, callback){
                //Send via POST

httpPostAsync(MDS.mainhost+"cmd?"+"uid="+MDS.minidappuid, command,
callback);
        },

        /**
         * Runs a SQL command on this MiniDAPPs SQL Database
         */
        sql : function(command, callback){
                //Send via POST

httpPostAsync(MDS.mainhost+"sql?"+"uid="+MDS.minidappuid, command,
callback);
        },

        /**
         * Network Commands
         */
        net : {

                /**
                 * Make a GET request
                 */
                GET : function(url, callback){
                        //Send via POST

httpPostAsync(MDS.mainhost+"net?"+"uid="+MDS.minidappuid, url,
callback);
                },

                /**
                 * Make a POST request
                 */
                POST : function(url, data, callback){

                        //Create the sinlg eline version..
```

```
                        var postline = url+"&"+data;

                        //Send via POST

httpPostAsync(MDS.mainhost+"netpost?"+"uid="+MDS.minidappuid,
postline, callback);
                }

        },

        /**
         * COMMS - send a message to ALL minidapps or JUST your own
service.js
         */
        comms : {

                /**
                 * PUBLIC message broadcast to ALL (callback is
optional)
                 */
                broadcast : function(msg, callback){

                        //Create the single line
                        var commsline = "public&"+msg;

                        //Send via POST

httpPostAsync(MDS.mainhost+"comms?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * PRIVATE message send just to this MiniDAPP
(callback is optional)
                 */
                solo : function(msg, callback){

                        //Create the single line
                        var commsline = "private&"+msg;

                        //Send via POST

httpPostAsync(MDS.mainhost+"comms?"+"uid="+MDS.minidappuid,
commsline, callback);
                }

        },

        /**
         * File access
         */
        file : {
                /**
                 * List file in a folder .. start at /
```

```
		 */
		list : function(folder, callback){

			//Create the single line
			var commsline = "list&"+folder;

			//Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
		},

		/**
		 * Save text - can be text, a JSON in string format
or hex encoded data
		 */
		save : function(filename, text, callback){

			//Create the single line
			var commsline = "save&"+filename+"&"+text;


			//Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
		},

		/**
		 * Save Binary Data - supply as a HEX string
		 */
		savebinary : function(filename, hexdata, callback){

			//Create the single line
			var commsline =
"savebinary&"+filename+"&"+hexdata;

			//Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
		},

		/**
		 * Load text - can be text, a JSON in string format
or hex encoded data
		 */
		load : function(filename, callback){

			//Create the single line
			var commsline = "load&"+filename;


			//Send via POST
```

```
httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * Load Binary data - returns the HEX data
                 */
                loadbinary : function(filename, callback){

                        //Create the single line
                        var commsline = "loadbinary&"+filename;


                        //Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * Delete a file
                 */
                delete : function(filename, callback){

                        //Create the single line
                        var commsline = "delete&"+filename;


                        //Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * Get the full path - if you want to run a command
on the file / import a txn / unsigned txn etc
                 */
                getpath : function(filename, callback){

                        //Create the single line
                        var commsline = "getpath&"+filename;


                        //Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * Make a directory
                 */
```

```
                makedir : function(filename, callback){

                        //Create the single line
                        var commsline = "makedir&"+filename;


                        //Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * Copy a file
                 */
                copy : function(filename, newfilename, callback){

                        //Create the single line
                        var commsline =
"copy&"+filename+"&"+newfilename;

                        //Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                },

                /**
                 * Move a file
                 */
                move : function(filename, newfilename, callback){

                        //Create the single line
                        var commsline =
"move&"+filename+"&"+newfilename;

                        //Send via POST

httpPostAsync(MDS.mainhost+"file?"+"uid="+MDS.minidappuid,
commsline, callback);
                }

        },

        /**
         * Function for GET parameters..
         */
        form : {

                //Return the GET parameter by scraping the
location..
                getParams : function(parameterName){
                            var result = null,
                        tmp = [];
```

```
                            var items =
window.location.search.substr(1).split("&");
                            for (var index = 0; index <
items.length; index++) {
                                tmp = items[index].split("=");
                                //console.log("TMP:"+tmp);
                                   if (tmp[0] === parameterName)
result = decodeURIComponent(tmp[1]);
                            }
                            return result;
                }
        },

        /**
         * UTILITY functions.. very useful
         */
        util : {

                //Convert HEX to Base 64 - removes the 0x if
necessary
                hexToBase64(hexstring) {
                        //Check if starts with 0x
                        var thex = hexstring;
                        if(hexstring.startsWith("0x")){
                                thex = hexstring.substring(2);
                        }

                    return btoa(thex.match(/\w{2}/g).map(function(a)
{
                        return String.fromCharCode(parseInt(a, 16));
                    }).join(""));
                },

                //Convert Base64 to HEX
                base64ToHex(str) {
                        const raw = atob(str);
                        let result = '';
                        for (let i = 0; i < raw.length; i++) {
                                const hex =
raw.charCodeAt(i).toString(16);
                                result += (hex.length === 2 ? hex :
'0' + hex);
                        }
                        return result.toUpperCase();
                },

                //Convert Base64 to a Uint8Array - useful for Blobs
                base64ToArrayBuffer(base64) {
                    var binary_string = window.atob(base64);
                    var len = binary_string.length;
                    var bytes = new Uint8Array(len);
                    for (var i = 0; i < len; i++) {
                        bytes[i] = binary_string.charCodeAt(i);
                    }
```

```
                    return bytes.buffer;
                },

                //Return a state variable given the coin
                getStateVariable(coin,port){

                        //Get the state vars
                        var statvars = coin.state;
                        var len = statvars.length;
                        for (var i = 0; i < len; i++) {
                                var state = statvars[i];
                                if(state.port == port){
                                        return state.data;
                                }
                        }

                        return undefined;
                }
        }
};

/**
 * Post a message to the Minima Event Listeners
 */
function MDSPostMessage(json){
   //And dispatch
   if(MDS_MAIN_CALLBACK){
                MDS_MAIN_CALLBACK(json);
   }
}


var PollCounter = 0;
var PollSeries  = 0;
function PollListener(){

        //The POLL host
        var pollhost = MDS.mainhost+"poll?"+"uid="+MDS.minidappuid;
        var polldata = "series="+PollSeries+"&counter="+PollCounter;

        httpPostAsyncPoll(pollhost,polldata,function(msg){

                //Are we on the right Series..
                if(PollSeries != msg.series){

                        //Reset to the right series..
                        PollSeries  = msg.series;
                        PollCounter = msg.counter;

                }else{

                        //Is there a message ?
                        if(msg.status == true){
```

```
                                //Get the current counter..
                                PollCounter =
msg.response.counter+1;

                                //And Post the message..

MDSPostMessage(msg.response.message);
                        }
                }

                //And around we go again..
                PollListener();
        });
}

/**
 * Utility function for GET request
 *
 * @param theUrl
 * @param callback
 * @param params
 * @returns
 */
function httpPostAsync(theUrl, params, callback){
        //Do we log it..
        if(MDS.logging){
                MDS.log("POST_RPC:"+theUrl+" PARAMS:"+params);
        }

        var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 && xmlHttp.status == 200){
                        //Do we log it..
                if(MDS.logging){
                        MDS.log("RESPONSE:"+xmlHttp.responseText);
                }

                //Send it to the callback function..
                if(callback){
                        callback(JSON.parse(xmlHttp.responseText));
                }
        }
    }
    xmlHttp.open("POST", theUrl, true); // true for asynchronous
        xmlHttp.overrideMimeType('text/plain; charset=UTF-8');
    //xmlHttp.setRequestHeader('Content-Type', 'application/json');
        xmlHttp.send(encodeURIComponent(params));
        //xmlHttp.send(params);
}

/**
 * Utility function for GET request (UNUSED for now..)
 *
 * @param theUrl
```

```
 * @param callback
 * @returns
 */
/*function httpGetAsync(theUrl, callback)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 && xmlHttp.status == 200){
                if(MDS.logging){
                                console.log("RPC      : "+theUrl);
                                console.log("RESPONSE :
"+xmlHttp.responseText);
                        }

                        //Always a JSON ..
                var rpcjson = JSON.parse(xmlHttp.responseText);

                //Send it to the callback function..
                if(callback){
                        callback(rpcjson);
                }
        }
    }
        xmlHttp.open("GET", theUrl, true); // true for asynchronous
    xmlHttp.send(null);
}*/

function httpPostAsyncPoll(theUrl, params, callback){
        //Do we log it..
        if(MDS.logging){
                MDS.log("POST_POLL_RPC:"+theUrl+" PARAMS:"+params);
        }

        var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 && xmlHttp.status == 200){
                        //Do we log it..
                if(MDS.logging){
                        MDS.log("RESPONSE:"+xmlHttp.responseText);
                }

                //Send it to the callback function..
                if(callback){
                        callback(JSON.parse(xmlHttp.responseText));
                }
        }
    }
    xmlHttp.addEventListener('error', function(ev){
                MDS.log("Error Polling - reconnect in 10s");
                setTimeout(function(){PollListener();},10000);
        });
    xmlHttp.open("POST", theUrl, true); // true for asynchronous
        xmlHttp.overrideMimeType('text/plain; charset=UTF-8');
    xmlHttp.send(encodeURIComponent(params));
```

```
}


//DMAX.js: this is specific to the minidapp within the minidapp.
dmax is a service that we run allowing users to get a permanent
maxima address for a fee of one minima a day
//minima addresses are typically dynamic so we provide a service a
bit like a dns for minima users. giving them a permanent maxima
address

/*
* Set Static MLS
* @param {*} callback
*/
function setStaticMLS(p2pidentity, callback) {
    MDS.log("Setting static MLS to " + p2pidentity);
    var maxcmd = `maxextra action:staticmls host:${p2pidentity}`;
    MDS.cmd(maxcmd, function (msg) {
        MDS.log(JSON.stringify(msg));
        if (callback) {
            callback(msg);
        }

    });
}

/**
 * Send message via Maxima to contat address or permanent address
 * @param {*} message
 * @param {*} address
 * @param {*} app
 * @param {*} callback
 */
function sendMaximaMessage(message, address, app, callback) {
    MDS.log("Sending message to " + address);
    var maxcmd = "maxima action:send poll:true to:" + address + "
application:" + app + " data:" + JSON.stringify(message);
    MDS.log(maxcmd);
    MDS.cmd(maxcmd, function (msg) {
        MDS.log(JSON.stringify(msg));
        if (callback) {
            callback(msg);
        }
    });
}

/**
 * Confirm coin exists and return the coin data response
 * @param {*} coinId
 * @param {*} callback
 * @returns coin data
 */
function confirmPayment(coinId, callback) {
    var maxcmd = "coins coinid:" + coinId;
```

```
    MDS.cmd(maxcmd, function (msg) {
        MDS.log(JSON.stringify(msg));
        if (callback) {
            callback(msg);
        }
    });
}

/**
 * Get Public Key
 * @param {*} callback
*/
function getPublicKey(callback) {
    var maxcmd = "maxima";
    MDS.cmd(maxcmd, function (msg) {
        MDS.log(JSON.stringify(msg));
        if (callback) {
            callback(msg.response.publickey);
        }
    });
}


/**
 * Send minima to address
 * @param {*} amount
 * @param {*} address
 * @param {*} callback
 * @returns coin data
 */
function sendMinima(amount, address, callback) {
    var maxcmd = "send amount:" + amount + " address:" + address;
    MDS.cmd(maxcmd, function (msg) {
        MDS.log(`sendMinima function response: $
{JSON.stringify(msg)}`);
        if (callback) {
            //return the coinid
            if (msg.status) {
                MDS.log(`coinid returned: $
{JSON.stringify(msg.response.body.txn.outputs[0].coinid)}`);
                callback(msg.response.body.txn.outputs[0].coinid);
            } else {
                MDS.log(msg.error);
                callback(false, msg.error)
            }
        }
    });
}

// when you run help in minima it shows you an overview of the
commands you can run and the things you can do when working within
the minidapp system (with MDS rules as outlind in MDS.js).
// this is what it reeurns:
```