Lecture 37 - Things Ethereum Needs

The simple contract that is in most of these examples

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.24 <0.9.0;

contract KVPair {

    event SetKVEvent(bytes32 key, bytes32 value);

    mapping (bytes32 => bytes32) public theData;

    constructor() { }

    function setKVPair(bytes32 key, bytes32 value) external {
        theData[key] = value;
        emit SetKVEvent(key, value);
    }

}
```

The commands to build the .go code (From Makefile)

```
gen_go:
	( cd ./contracts ; \
		solc --abi KVPair.sol -o . --overwrite )
	( cd ./contracts ; \
		solc --bin KVPair.sol -o . --overwrite )
	( cd ./contracts ; \
		abigen --bin=KVPair.bin --abi=KVPair.abi --pkg=KVPair --out=KVPair.go )
```

Events that lead to outside actions

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "math/big"
    "os"
    "strings"

    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/accounts/abi"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/crypto"
    "github.com/ethereum/go-ethereum/ethclient"
    "github.com/pschlump/godebug"

    kv "github.com/Univ-Wyo-Education/S22-4010/class/lect/37/eth/contracts"
```

```
20: )
21:
22: type CfgType struct {
23:     Addrs map[string]string
24: }
25:
26: func main() {
27:
28:     // client, err := ethclient.Dial("wss://rinkeby.infura.io/ws")
29:     client, err := ethclient.Dial("ws://127.0.0.1:8546") // connect to local geth or ganache
30:     if err != nil {
31:         fmt.Fprintf(os.Stderr, "Unable to connect to network/geth/ganache.  Error:%s\n", err)
32:         os.Exit(1)
33:     }
34:
35:     buf, err := ioutil.ReadFile("cfg.json")
36:     if err != nil {
37:         fmt.Fprintf(os.Stderr, "Unable to open cfg.json for read.  Error:%s\n", err)
38:         os.Exit(1)
39:     }
40:     var cfg CfgType
41:     err = json.Unmarshal(buf, &cfg)
42:     if err != nil {
43:         fmt.Fprintf(os.Stderr, "Unable to parse cfg.json.  Error:%s\n", err)
44:         os.Exit(1)
45:     }
46:
47:     contractAddress := common.HexToAddress(cfg.Addrs["KVPair"])
48:     query := ethereum.FilterQuery{
49:         FromBlock: big.NewInt(1),
50:         ToBlock:   big.NewInt(5000000),
51:         Addresses: []common.Address{
52:             contractAddress,
53:         },
54:     }
55:
56:     logs, err := client.FilterLogs(context.Background(), query)
57:     if err != nil {
58:         fmt.Fprintf(os.Stderr, "Error:%s at:%s\n", err, godebug.LF())
59:         os.Exit(1)
60:     }
61:
62:     contractAbi, err := abi.JSON(strings.NewReader(string(kv.KVPairABI)))
63:     if err != nil {
64:         fmt.Fprintf(os.Stderr, "Error:%s at:%s\n", err, godebug.LF())
65:         os.Exit(1)
66:     }
67:
68:     eventSignature := []byte("SetKVEvent(bytes32,bytes32)")
69:     hash := crypto.Keccak256Hash(eventSignature)
70:     fmt.Printf("Event Signature Is: 0x%x\n", hash)
71:
72:     for _, vLog := range logs {
73:         fmt.Printf("BlockHash 0x%x\n", vLog.BlockHash)
74:         fmt.Printf("BlockNumber: %d\n", vLog.BlockNumber)
75:         fmt.Printf("Block Tx: 0x%0x\n", vLog.TxHash)
76:
77:         // if err := it.contract.UnpackLog(it.Event, it.event, log); err != nil {
78:         eventX, err := contractAbi.Unpack("SetKVEvent", vLog.Data)
79:         if err != nil {
80:             fmt.Fprintf(os.Stderr, "Error:%s at:%s\n", err, godebug.LF())
81:             os.Exit(1)
82:         }
83:
84:         fmt.Printf("Event Data: %s\n", godebug.SVarI(eventX))
85:
```

```
        var topics [4]string
        for i := range vLog.Topics {
            topics[i] = vLog.Topics[i].Hex()
        }

        fmt.Printf("Event Topics: %s\n", topics[0])
    }

}
```

```
package main

// Program to expose all of the transfers in a ERC777/ERC20/ERC1155 type contract.
// Watches events for transfers.

import (
    "context"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "math/big"
    "os"
    "strings"

    simple777 "github.com/Univ-Wyo-Education/S22-4010/class/lect/37/eth/Simple777Token"
    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/accounts/abi"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/crypto"
    "github.com/ethereum/go-ethereum/ethclient"
)

// LogTransfer ..
type LogTransfer struct {
    From   common.Address
    To     common.Address
    Tokens *big.Int
}

type CfgType struct {
    Addrs map[string]string
}

// LogApproval ..
type LogApproval struct {
    TokenOwner common.Address
    Spender    common.Address
    Tokens     *big.Int
}

func main() {
    client, err := ethclient.Dial("http://127.0.0.1:8545")
    if err != nil {
        log.Fatal(err)
    }

    buf, err := ioutil.ReadFile("cfg.json")
    if err != nil {
        fmt.Fprintf(os.Stderr, "Unable to open cfg.json for read.  Error:%s\n", err)
        os.Exit(1)
    }
    var cfg CfgType
```

```
54:     err = json.Unmarshal(buf, &cfg)
55:     if err != nil {
56:         fmt.Fprintf(os.Stderr, "Unable to parse cfg.json.  Error:%s\n", err)
57:         os.Exit(1)
58:     }
59:
60:     contractAddress := common.HexToAddress(cfg.Addrs["Simple777Token"])
61:     query := ethereum.FilterQuery{
62:         FromBlock: big.NewInt(5143020),
63:         ToBlock:   big.NewInt(10000000),
64:         Addresses: []common.Address{
65:             contractAddress,
66:         },
67:     }
68:
69:     logs, err := client.FilterLogs(context.Background(), query)
70:     if err != nil {
71:         log.Fatal(err)
72:     }
73:
74:     contractAbi, err := abi.JSON(strings.NewReader(string(simple777.TokenABI)))
75:     if err != nil {
76:         log.Fatal(err)
77:     }
78:
79:     logTransferSig := []byte("Transfer(address,address,uint256)")
80:     LogApprovalSig := []byte("Approval(address,address,uint256)")
81:     logTransferSigHash := crypto.Keccak256Hash(logTransferSig)
82:     logApprovalSigHash := crypto.Keccak256Hash(LogApprovalSig)
83:
84:     for _, vLog := range logs {
85:         fmt.Printf("Log Block Number: %d\n", vLog.BlockNumber)
86:         fmt.Printf("Log Index: %d\n", vLog.Index)
87:
88:         switch vLog.Topics[0].Hex() {
89:         case logTransferSigHash.Hex():
90:             fmt.Printf("Log Name: Transfer\n")
91:
92:             var transferEvent LogTransfer
93:
94:             err := contractAbi.Unpack(&transferEvent, "Transfer", vLog.Data)
95:             if err != nil {
96:                 log.Fatal(err)
97:             }
98:
99:             transferEvent.From = common.HexToAddress(vLog.Topics[1].Hex())
100:            transferEvent.To = common.HexToAddress(vLog.Topics[2].Hex())
101:
102:            fmt.Printf("From: %s\n", transferEvent.From.Hex())
103:            fmt.Printf("To: %s\n", transferEvent.To.Hex())
104:            fmt.Printf("Tokens: %s\n", transferEvent.Tokens.String())
105:
106:        case logApprovalSigHash.Hex():
107:            fmt.Printf("Log Name: Approval\n")
108:
109:            var approvalEvent LogApproval
110:
111:            err := contractAbi.Unpack(&approvalEvent, "Approval", vLog.Data)
112:            if err != nil {
113:                log.Fatal(err)
114:            }
115:
116:            approvalEvent.TokenOwner = common.HexToAddress(vLog.Topics[1].Hex())
117:            approvalEvent.Spender = common.HexToAddress(vLog.Topics[2].Hex())
118:
119:            fmt.Printf("Token Owner: %s\n", approvalEvent.TokenOwner.Hex())
```

```
120:             fmt.Printf("Spender: %s\n", approvalEvent.Spender.Hex())
121:             fmt.Printf("Tokens: %s\n", approvalEvent.Tokens.String())
122:         }
123:
124:         fmt.Printf("\n\n")
125:     }
126: }
```

Scheduler

```
 1: package main
 2:
 3: import (
 4:     "context"
 5:     "crypto/ecdsa"
 6:     "fmt"
 7:     "log"
 8:     "math/big"
 9:     "os"
10:     "time"
11:
12:     "github.com/ethereum/go-ethereum/accounts/abi/bind"
13:     "github.com/ethereum/go-ethereum/common"
14:     "github.com/ethereum/go-ethereum/crypto"
15:     "github.com/ethereum/go-ethereum/ethclient"
16:     "github.com/robfig/cron"
17:     log "github.com/sirupsen/logrus"
18:
19:     kv "github.com/Univ-Wyo-Education/S22-4010/class/lect/37/eth/contracts"
20: )
21:
22: func init() {
23:     log.SetLevel(log.InfoLevel)
24:     log.SetFormatter(&log.TextFormatter{FullTimestamp: true})
25: }
26:
27: func main() {
28:     log.Info("Create new cron")
29:     c := cron.New()
30:     c.AddFunc("0 0 * * *", func() {
31:         log.Info("[Job 1]Every day at midnight\n")
32:         callContract()
33:     })
34:
35:     // Start cron with one scheduled job
36:     log.Info("Start cron")
37:     c.Start()
38:     time.Sleep(1 * time.Minute)
39: }
40:
41: func callContract() {
42:
43:     // client, err := ethclient.Dial("https://rinkeby.infura.io")
44:     client, err := ethclient.Dial("http://127.0.0.1:8545")
45:     if err != nil {
46:         log.Fatal(err)
47:     }
48:
49:     privateKey, err := crypto.HexToECDSA(os.Getenv("PrivateKey_for_127"))
50:     if err != nil {
51:         log.Fatal(err)
```

```
    }

    publicKey := privateKey.Public()
    publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey)
    if !ok {
        log.Fatal("cannot assert type: publicKey is not of type *ecdsa.PublicKey")
    }

    fromAddress := crypto.PubkeyToAddress(*publicKeyECDSA)
    nonce, err := client.PendingNonceAt(context.Background(), fromAddress)
    if err != nil {
        log.Fatal(err)
    }

    gasPrice, err := client.SuggestGasPrice(context.Background())
    if err != nil {
        log.Fatal(err)
    }

    auth := bind.NewKeyedTransactor(privateKey)
    auth.Nonce = big.NewInt(int64(nonce))
    auth.Value = big.NewInt(0)     // in wei
    auth.GasLimit = uint64(600000) // in units
    auth.GasPrice = gasPrice

    address := common.HexToAddress(cfg.Addrs["KVPair"])
    instance, err := kv.NewKVPair(address, client)
    if err != nil {
        log.Fatal(err)
    }

    // Setup to call function that is a tranaction (requires gas) and changes data.

    key := [32]byte{}
    value := [32]byte{}
    copy(key[:], []byte("hello"))
    copy(value[:], []byte("world"))

    tx, err := instance.SetKVPair(auth, key, value)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Printf("tx sent: %s\n", tx.Hash().Hex())

    time.Sleep(1 * time.Minute) // wait for data to be on-chain

    result, err := instance.TheData(nil, key)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Printf("Key Lookup: %s\n", result) // "world"
}
```