

# Auditing System by ISUNCLOUD

Currently, smart contracts are deployed using Remix, and data is input directly into the deployed smart contracts via Etherscan or Remix.

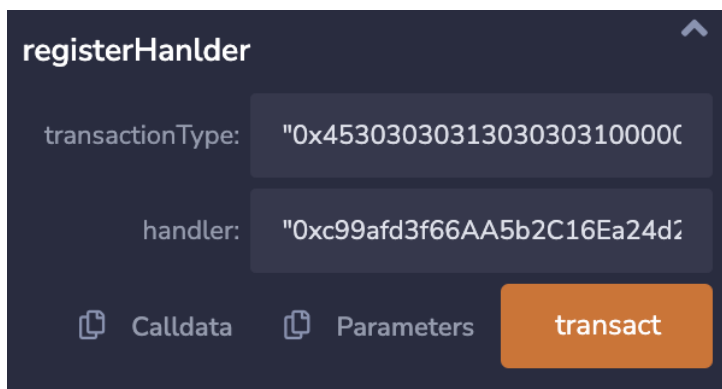
## 1. System Deployment:

**Deploy the smart contracts in the following sequence:**

1. parser.sol
2. reports.sol
3. TransactionContract.sol (parser address)
4. Handlers Contracts (TransactionContract address, parser address)
5. Calculating Contracts (TransactionContract address, Parser address , report address)
6. getTransactionTimeSpan.sol (transaction address, parser address, functioning contracts addresses)

## 2. System Flow:

1. Users start by registering handlers using the registerHandlers function in the **TransactionContract**, inputting the **TransactionType** (of type **bytes32**) and the **handler's** address (of type **address**).



The screenshot shows a dark-themed interface for the `registerHanlder` function. It has two input fields: `transactionType:` with the value `"0x453030303130303031000000"` and `handler:` with the value `"0xc99afd3f66AA5b2C16Ea24d7"`. Below the inputs are three icons: a clipboard for 'Calldata', a document for 'Parameters', and an orange button labeled 'transact'.

2. Record data using a **bytes32** array in the **addRecord** function of the **TransactionContract.sol** contract, where each element has been multiplied by  $10^{18}$ . The first element must be the eventID, and the second should specify the event type. Users must omit the timestamp column to prevent fraudulent events; the system will automatically record the current time."

For example, for

3. In order to create report(s) in a time span, the users first set rates and reportID on "setRate" function in "createTimeSpanReport". The users then interact with the `filterTransactionsInRange` function to set a specific time span. This function employs the `eventID` as primary keys and a `reportID` to organize reports under the `reportID`, preventing any disarray. It then retrieves and returns the transactions that occurred within the specified time span, without providing the full dataset.(S07 - S08)

4. Then pass those transactions to calculating functions (use eventID to determine which one), the calculating functions first use "Iparser.sol" function to change bytes32 into string or int256 and then calculate data with planned formula.
5. After calculating calculating get a 3D array from "Ireports.sol", and then add results into the respective column.
6. We can check the numbers is right or not by calling the function getValue(reportID, reportType, reportColumn) in reports.sol.

### 3. Interact with smart contracts on Ethereum and checking results

1. First, users should download and deploy the "hardhat" environment locally .
2. Run "***npm hardhat run transformReportAPI.js***".
3. This program would help users to interact with reports.sol on ethereum by getting reports data.
4. The program will parse the raw data into planned API format.

### 4. Future Plans:

1. Develop a standardized interface for calculation functions and store it in the interfaces file.
2. Code each function up to Column 7, encompassing all types of deposits and withdrawals.
3. Implement Mistake Proofing mechanisms, such as require statements.
4. Enhance calculation accuracy, given that Solidity lacks a float variable type.
5. Optimize gas fees through careful data structure design.
6. Create an interface for reports.sol and import only the interface.