

LAB 11 Aug, 2023

Events

1 Introduction

A fundamental feature that enables efficient and responsive **communication** within contracts.

Events serve as **notifications triggered** by specific transactions or smart contract actions, allowing developers and users to track important changes and interactions occurring on the **Ethereum Virtual Machine (EVM)**.

Events play a crucial role in informing external applications about the state changes in contracts, facilitating the execution of dependent logic.

Understanding how events are **stored** on the EVM and how they can be verified on platforms like Etherscan is paramount to comprehending the inner workings of decentralized systems.

2 Purpose and Characteristics of Events:

Solidity events serve as **transaction logs** stored on the Ethereum Virtual Machine (EVM). They offer the following characteristics:

Responsiveness: Applications, such as Ether.js, can subscribe and listen to events through the RPC interface. By doing so, they can promptly respond to events at the frontend, enabling real-time updates and interactions with the contract.

Economical Data Storage: Storing data in events is cost-effective, consuming approximately 2,000 gas per event. In contrast, storing new variables on-chain requires at least 20,000 gas. Leveraging events can lead to significant gas savings and optimize contract performance.

3 How To Use Events

3.1 Declaring and Defining Events:

Events are declared using the **event** keyword, followed by the event name and the type and name of each parameter to be recorded.

For example, let's consider the Transfer event from an ERC20 token contract:

```
event Transfer(address indexed from, address indexed to, uint256 value);
```

In this event declaration, 'three' parameters (from, to, and value) are recorded.

The **indexed** keyword is used to mark certain parameters for efficient querying.

3.2 Emitting Events:

Events can be emitted within functions to record specific occurrences.

Each time an event is emitted, the associated parameters are recorded in the transaction logs.

For instance, the following example demonstrates emitting a Transfer event within the `_transfer()` function:

```
// define _transfer function, execute transfer logic

function _transfer(address from, address to, uint256 amount)
external {

    _balances[from] = 10000000; // give some initial tokens to transfer
    address

    _balances[from] -= amount; // "from" address minus the number of
    transfer

    _balances[to] += amount; // "to" address adds the number of transfer

    // emit event

    emit Transfer(from, to, amount); }
```

By emitting events, you can provide crucial information to external applications and enable them to react accordingly.

EXERCISE

Deploy the above smart contract and obtain the logs in json format.

Hint:

You can access the logs by following method:



EXERCISE

- 1) Deploy, observe and explain the following code.
- 2) Access the logs and see how the logs for each **emitted event** are written as.
- 3) Create another event in the 'contract' given below that **emits** the 'quality' (of **Quality** data type), 'spec' (of **specs** data type) and 'ROM' (of **uint** data type).

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Events{

    enum Quality{

        Good, Moderate, Bad

    }

    struct specs{

        string microprocessor;

        uint RAM;

    }

}
```

```

//A single event can have multiple parameters
event multi(string laptop, uint price);

//You can also emit enums and structs as well
//An event can have upto 3 indexed parameters
event multiple(string laptop, uint indexed price, Quality indexed q,
specs indexed s);

//Naming the variable in an event is not necessary
event mu(string, Quality);

//You can also emit empty events
event empty();

function emits() public{
    emit multi("Hp", 150000);
    emit multiple("dell", 100000, Quality.Good, specs("intel", 128));
    emit mu("Lenovo", Quality.Moderate);
    emit empty();
}
}

```

4 Importance of Events

- When an event is emitted, it stores its arguments in the transaction logs. The transaction logs are then stored on the blockchain accessible for anyone who has the address of the contract.
- As the transaction logs are stored in the blockchain hence a history of transaction logs containing every change to the transaction is maintained in the block chain.
- You can emit different deposit and transfer actions, this will maintain a history of those actions in the transaction logs.
- You can use events to test your smart contract for specific variables.
- In terms of gas consumption, they are cheaper than storing data.

EXERCISE

- 1) Create a smart 'contract' named 'Eventss'. Declare an **address** data type state variable in it named 'owner'.
- 2) Make a 'constructor()' for the contract that initialized the value of 'owner' to be **msg.sender**.
- 3) Make a 'NewOwner()' function that takes an **address** data type value as input from the user and assigns it to 'owner'.
- 4) Make an **event** having parameters 'prevowner' and 'newowner', both of **address** data type.
- 5) Modify the 'NewOwner()' function such that it also **emits** the address of both the previous owner and the new owner.

Hint:

You can **emit** the **event** before updating the value of the 'owner'. By doing this you can use 'owner' as 'prevowner' address and the user inputted value as 'newowner' address

EXERCISE

- 1) Create a **struct** named 'PersonInfo', having multiple elements giving information about a person. Also, make a variable named **P1**, for that struct.

```
contract PersonContract {  
  
    struct PersonInfo {...}  
  
    PersonInfo public P1;  
}
```

- 2) Make an **event** having parameters 'OldValue' of **PersonInfo** data type, NewValue of **PersonInfo** data type, 'timestamp' of **uint** data type and 'number' also of **uint** data type. Name the **event** as 'Update'.

```
event Update(  
    PersonInfo oldValue,  
    PersonInfo newValue,  
    uint256 timestamp,  
    uint256 blockNumber  
);
```

- 3) Make a 'setter' function for your **P1** variable. The 'setter' function should be such that it also **emits** the 'old value' of **P1**, the 'new value' of **P1**, 'timestamp' of the block to show when it is emitted and 'number' of the block to show in which block number it is emitted.

```
function setPersonInfo(...) public {  
    PersonInfo memory oldValue = P1;  
    . . .  
    emit Update(oldValue, P1, block.timestamp, block.number);  
}
```

- 4) Deploy the contract and Observe the Logs.

EXERCISE

Create a **contract** that **emits** an **event** when the user assigns a **zero address** (an address consisting of only zeros) to the 'owner' state variable (**address** data type) on deployment of the contract.

Hint:

- Your **event** can have only one parameter of **string** type that **emits** a statement like "Zero Address inputted".

```
event ZeroAddressInputted(string message);
```

- The statement states 'on deployment of contract', so you will make a 'constructor' that takes input from the user.

```
constructor(address _initialOwner)
```

- Using **if** condition, you can check whether user input is a **zero address** in the 'constructor' function.
- Inside the **if** condition you can **emit** your **event**.

```
emit ZeroAddressInputted("Zero Address inputted on deployment of contract");
```

EXERCISE

- 1) Create a **function** named 'numgive()' that takes an **int** type value from the user and **returns** an output value of **int** data type.
- 2) In the 'numgive()' function, declare three **int**-type local variables: 'neg', 'positiveEven', and 'positiveOdd'.
- 3) Within the function, implement an **if/else-if** conditional with the following three conditions:

- a. If the user input is a negative number, store the value in 'neg' and **return** 'neg'.
 - b. If the user input is a positive even number, store the value in 'positiveEven' and **return** 'positiveEven'.
 - c. If the user input is a positive odd number, store the value in 'positiveOdd' and **return** 'positiveOdd'.
- 4) Create an **event** named 'localVar' having a **string** type parameter named 'message'.
- 5) Within the 'numgive()' function's conditions, **emit** the 'localVar' **event** before each **return** statement, including a descriptive message indicating which local variable is returning the value.

Hint:

For the third condition you can write:

```
else if(condition){  
    positiveOdd = n;  
    emit localVar("positiveOdd local variable returned the value");  
    return positiveOdd;  
}
```

How Events are Actually Stored on EVM

1 Introduction

- In this part, we are going to delve into the significance of events, explore their storage mechanisms within the EVM, and provide hands-on guidance on **how to verify** these events using the Etherscan platform.

Note:

Section 2 is related to Creation of MetaMask Wallet, **Section 3** is related to Fetching Some Testnet which you have already done in Lab 01 of Module 02.

If you have no problem with these 2 highlighted topics, jump to **Section 4**.

2 Setup MetaMask Wallet

2.1 Introduction

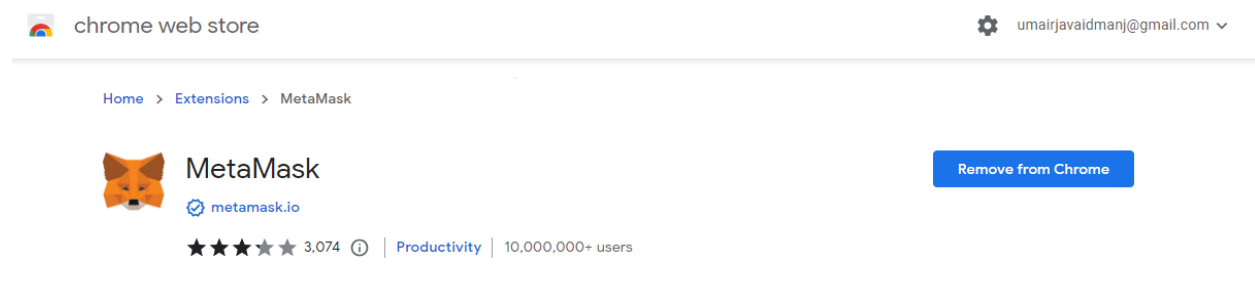
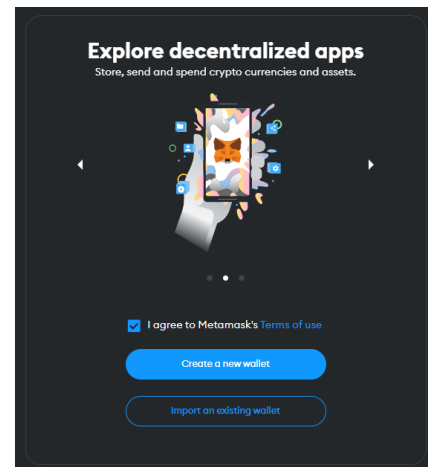
MetaMask is a popular cryptocurrency wallet that enables users to interact with the Ethereum blockchain and manage their Ethereum private keys. Here is a step-by-step guide on how to create a MetaMask wallet:

2.2 Steps Involved

Step 1: Download MetaMask Wallet

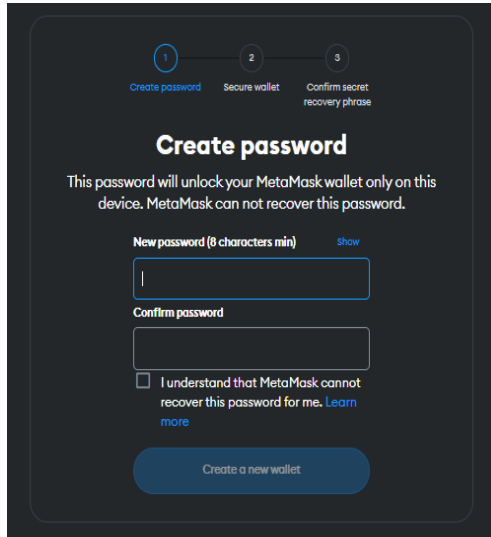
To begin, visit the official MetaMask website at <https://metamask.io/>.

On the homepage, locate the "Download" button and click on it. Choose your preferred browser or mobile application and follow the installation instructions to install the **MetaMask extension**.



Step 2: MetaMask Wallet Installation

Once the MetaMask extension is installed, click on its icon to launch it. On the welcome screen, click on "Get Started." At this point, you have two options: you can either import an existing wallet using the seed phrase or create a new one.



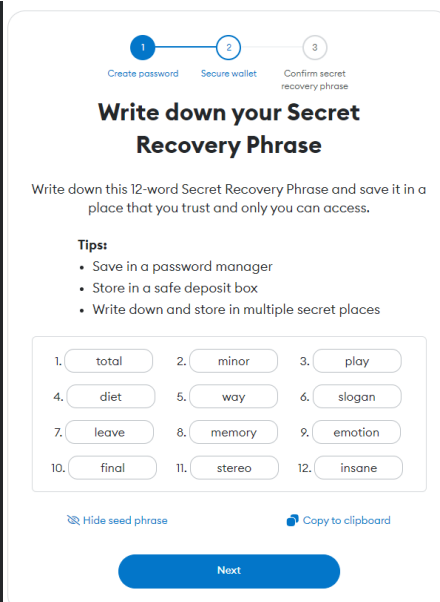
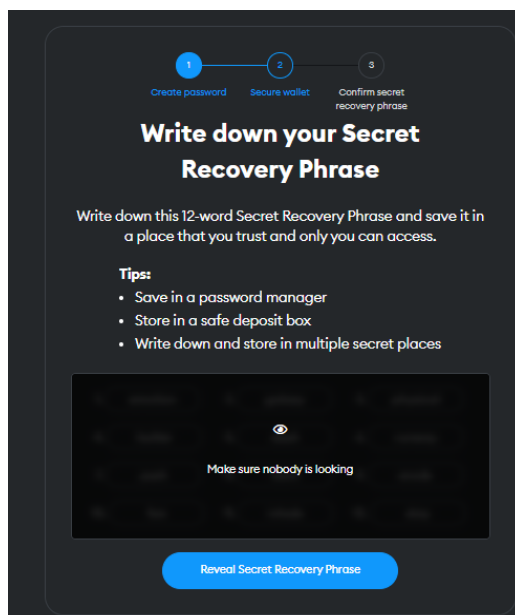
Step 3: Create a Strong Password

To ensure the security of your wallet, it is crucial to create a strong password. Choose a password that is unique, complex, and difficult to guess. Avoid using easily identifiable information and consider combining uppercase and lowercase letters, numbers, and special characters.

Step 4: Securely Store the Seed Phrase

MetaMask requires you to store a seed phrase, also known as a Secret Recovery Phrase, in a safe place. This seed phrase is essential for recovering your funds in case of device failure or browser reset. Click on "**Click**

here to reveal secret words" to display your seed phrase. We highly recommend writing down the 12-word phrase on a piece of paper and storing it in a secure location where only you have access. It is crucial to never share your seed phrase or private key with anyone or any website.



Step 5: Seed Phrase Confirmation

To ensure that you have correctly noted down your seed phrase, MetaMask will ask you to confirm it. On the following screen, click on each word in the order in which they were presented on the previous screen. Once you have selected all the words, click on "Confirm" to proceed.

The image shows two side-by-side screenshots from the MetaMask wallet creation process.

The left screenshot is titled "Confirm Secret Recovery Phrase". At the top, a progress bar shows three steps: 1. Create password, 2. Secure wallet, and 3. Confirm secret recovery phrase. The main heading is "Confirm Secret Recovery Phrase". Below it, the instruction "Confirm Secret Recovery Phrase" is displayed. A grid of 12 word options is shown, numbered 1 through 12. The words are: 1. total, 2. minor, 3. play, 4. diet, 5. way, 6. slogan, 7. leave, 8. memory, 9. emotion, 10. final, 11. stereo, 12. insane. A large blue "Confirm" button is at the bottom.

The right screenshot is titled "Wallet creation successful". It features a colorful confetti icon. The text says: "You've successfully protected your wallet. Keep your Secret Recovery Phrase safe and secret -- it's your responsibility! Remember:". Below this, there are four bullet points: "MetaMask can't recover your Secret Recovery Phrase.", "MetaMask will never ask you for your Secret Recovery Phrase.", "Never share your Secret Recovery Phrase with anyone or risk your funds being stolen", and "Learn more". At the bottom, there is a blue "Got it" button and a link for "Advanced configuration".

Step 6: Congratulations!

Congratulations! You have successfully set up your MetaMask wallet. You can now access your wallet by clicking on the MetaMask icon located in the top-right corner of your preferred browser. From there, you can manage your Ethereum private keys, store Ether and other tokens, and interact with decentralized applications (dapps) on the Ethereum blockchain.

It is important to remember that **MetaMask does not store any personal information such as your email address or password**. You retain full control over your crypto-identity and assets. As a responsible user, always prioritize the security of your wallet by keeping your seed phrase and private key confidential and ensuring that your devices and browsers are adequately protected.

3 Fetch Some TestNet Crypto

3.1 Introduction

Testnet cryptocurrencies play a crucial role in the development and testing of decentralized applications (dapps) and smart contracts.

They provide a **simulated environment** that closely resembles the mainnet (production) blockchain but operates on a separate network specifically designed for testing purposes.

3.2 How to obtain TestNet Crypto

Now, let's discuss the steps to obtain testnet crypto:

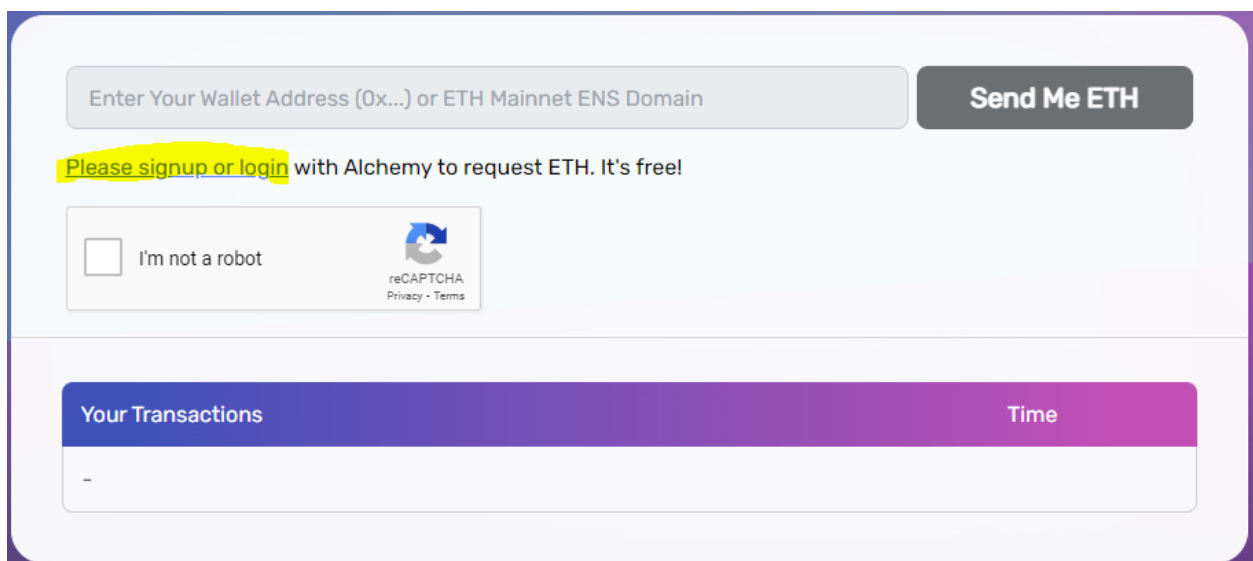
Step 1: Select a Testnet

Choose the testnet that aligns with your development needs. Popular Ethereum testnet include Ropsten, Rinkeby, Kovan, and Goerli. Each testnet has its own characteristics and methods to obtain testnet crypto.

Step 2: Testnet Faucets

Testnet faucets are web platforms that distribute testnet crypto to users. Visit a reliable testnet faucet website, such as <https://sepoliafaucet.com/>, signup using your email and phone number.

Follow their instructions to request testnet crypto for your chosen network.



The screenshot shows the Sepolia faucet website interface. At the top, there is a text input field labeled "Enter Your Wallet Address (0x...) or ETH Mainnet ENS Domain" and a button labeled "Send Me ETH". Below the input field, a message reads "Please signup or login with Alchemy to request ETH. It's free!". Underneath this message is a reCAPTCHA widget with the text "I'm not a robot" and a checkbox. At the bottom, there is a table with the header "Your Transactions" and a column "Time". The table body is currently empty, showing a single row with a hyphen "-" in the "Your Transactions" column.

Your Transactions	Time
-	

Awesome! What type of project would you like to build?

DAOs

Other

Learning

DeFi

NFTs

Analytics

I'm not a developer

Next

Let's choose your chain.

Don't worry, you can always add more later.

MOST POPULAR

Ethereum

Solana

Polygon PoS

Polygon zkEvm

Optimism

Astar

Arbitrum

Starknet

Back

Next

Choose the free payment plan and **skip** the options for financial details.

Choose your plan.

Free

\$0

The most powerful free tier in blockchain.

MOST POPULAR

Growth

\$49/mo

Ultimate scalability and enhanced methods.

Enterprise

Volume pricing, enterprise support & SLAs, on-prem options.

Compare

Back

Next

Add payment info now to unlock powerful benefits OPTIONAL

1,000,000 additional free compute units per month

FIRST NAME

LAST NAME

CARD NUMBER

EXPIRATION

CVV

BILLING ADDRESS

ADDRESS

Back

Skip

Next

Step 3: Provide Wallet Address

Most testnet faucets require you to provide your wallet address to send the testnet crypto. Ensure you have a compatible wallet for the selected testnet (e.g., MetaMask), and copy the address from your wallet to the faucet website.

Account 1

0x76f...130F

0 ETH

\$0.00 USD

+

↗

↻

↺

📈

Buy

Send

Swap

Bridge

Portfolio

0x76f9171551D5648D0A8B7f598ca14c558bEe130F

Send Me ETH

✓

Alchemy account connected, receive 0.5 Sepolia ETH! Fetch any NFT data simply and easily with the [Alchemy.NET API](#)

☐

I'm not a robot

hCAPTCHA

Privacy · Terms

Your Transactions

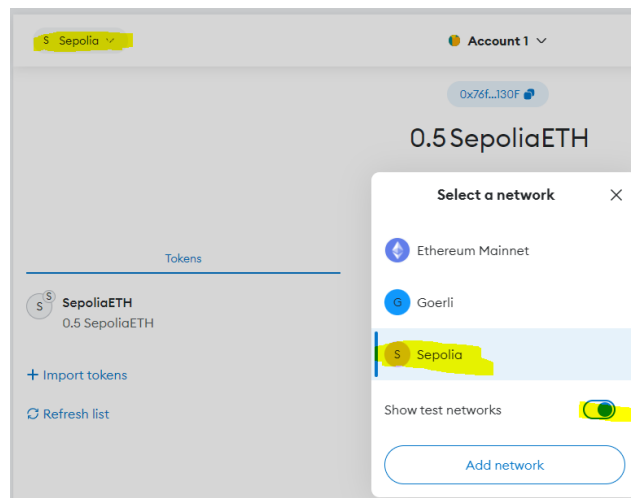
Time

Step 4: Complete Verification (If Required)

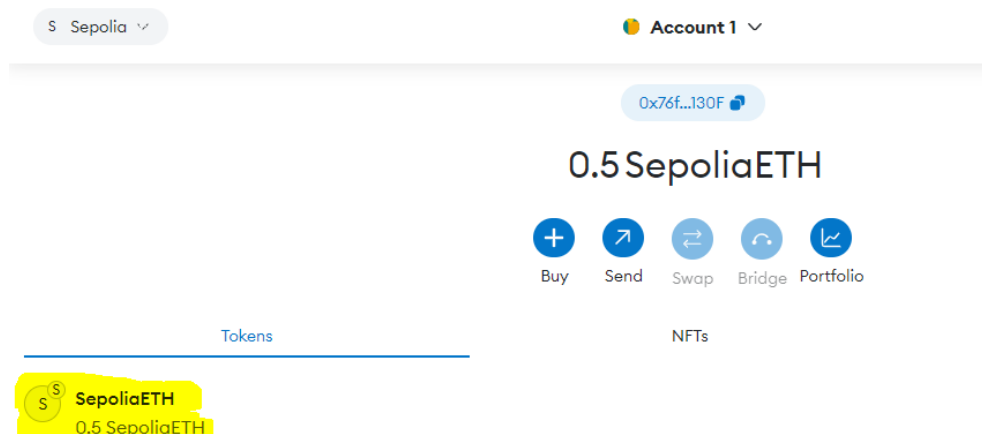
Some testnet faucets may require additional verification steps to prevent abuse. Follow their instructions, such as solving captchas or completing simple tasks, to proceed with the testnet crypto request.

Step 5: Receive Testnet Crypto

Once you have completed the necessary steps, the testnet faucet will process your request and send the testnet crypto to the provided wallet address. The time required for the testnet crypto to arrive may vary depending on the specific testnet and faucet used.



By following these steps, you can obtain testnet crypto and begin testing and developing your applications in a secure and cost-effective environment.



Remember, testnet crypto has no monetary value and cannot be used on the mainnet. It is crucial to differentiate between testnet and mainnet crypto to avoid any confusion or unintentional financial transactions.

4 Generating Logs

Below are the steps to generate and confirm logs (events) in Solidity using the Remix IDE

Step 1: Write Smart Contract Code

```
contract Event {  
    event Log(address indexed sender, string msg);  
    event AnotherLog();  
  
    function test() public {  
        emit Log(msg.sender, "Hello Web3.0");  
        emit Log(msg.sender, "It's all about Decentralization");  
        emit AnotherLog();  
    }  
}
```

1) Event Declaration:

- The **'event'** keyword is used to declare events in Solidity.
- In this code, two events are declared: **Log** and **AnotherLog**.

2) Indexed Parameters: (Optional)

- Events can have parameters, and up to three of these parameters can be indexed.
Indexed parameters are useful for efficient event filtering when querying logs.
- In the Log event, the **address** parameter is indexed using the indexed keyword. This allows for filtering events by the sender's address.

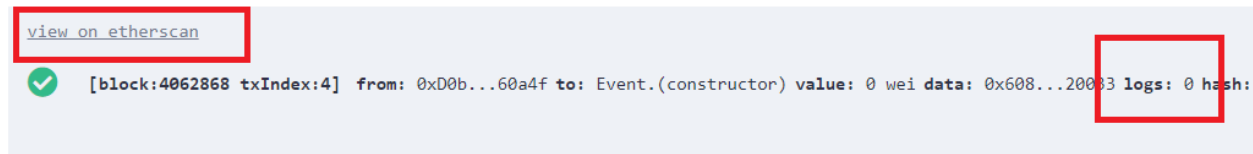
3) Emitting Events:

- Events are emitted using the **'emit'** keyword followed by the event name and its parameters. Emitting an event **creates a log entry on the blockchain**, capturing the event's data.
- In the test function, the **Log** event is emitted twice with different messages and the sender's address (msg.sender). This records two instances of the event with distinct data.
- The **AnotherLog** event is also emitted within the same function, demonstrating the emission of events without parameters.

Step 2: Deploy the Smart Contract

1. After compiling, Go to the "Deploy & Run Transactions" tab in the right panel.
2. Choose the correct environment **Injected Provider - MetaMask**.
3. Click the "Deploy" button to deploy the smart contract.

On successful deployment, you can observe that this **Contract has generated no log**.












The "**View on Etherscan**" feature provides a convenient way to quickly access the Etherscan website and view detailed information about a deployed smart contract.

Etherscan is a widely used blockchain explorer platform that allows users to explore, verify, and analyze activities on the Ethereum blockchain.















When you click "**View on Etherscan**" in Remix after deploying a contract, it typically opens a new tab in your web browser and navigates to the Etherscan page for the specific contract address that was deployed, and displays the **recorded** Transaction Details

[This is a Sepolia **Testnet** transaction only]

Transaction Hash:	0x5355e769d9f5ef1283ad0948f657c27829a9b2b09dbe2489da0cb030419013a 
Status:	 Success
Block:	 4062868 20 Block Confirmations
Timestamp:	 4 mins ago (Aug-10-2023 05:09:00 PM +UTC)
From:	0xD0b34b98E478138832bBbe53b549441Cb2760a4f 
To:	[ 0x4034da9958f4cfc2cc7d1f8dacedd35f481caba5 Created]  
Value:	 0 ETH (\$0.00)
Transaction Fee:	0.000458304472690886 ETH (\$0.00)
Gas Price:	2.689943318 Gwei (0.000000002689943318 ETH)

Transaction Details:

On successful transactions, Remix provides you with a number of parameters, as highlighted below.

status	true Transaction mined and execution succeed
transaction hash	0xd4f53cfb582638105ddf0dddf854e2a3b7fbe1f9a6c877012f5fdd26d7eaa7eb 
block hash	0xa3a34a6100bb31ddb81173af9d71a34d840d34ddfe7f71be2efc5b1dd263b9d7 
block number	4062842 
contract address	0x8dD6e32c1A42728C6703c253B26813F4A75A8a9d 
from	0xD0b34b98E478138832bBbe53b549441Cb2760a4f 
to	Demo.(constructor) 
gas	206682 gas 
transaction cost	206682 gas 
input	0x608...20033 
decoded input	{ } 
decoded output	- 
logs	[]  
val	0 wei 

Here's a brief explanation of each of the important parameters typically provided after a successful deployment of a contract in Solidity:

- **Status:** Indicates whether the contract deployment transaction was successful **(1)** or unsuccessful **(0)**. A successful deployment means the contract's bytecode was successfully added to the blockchain.
- **Transaction Hash:** A unique identifier for the transaction that deployed the contract. This hash can be **used to locate** the transaction and its details on the blockchain.
- **Block Hash:** The hash of the block in which the contract deployment transaction was included. This hash ties the transaction to a specific block in the blockchain.
- **Block Number:** The numerical **position of the block** within the blockchain where the contract deployment transaction was included. It indicates the order of the block in the chain.

- **Contract Address:** The address assigned to the deployed contract. This is the location on the blockchain where the **contract's code** and storage are stored.
- **Gas Used:** The amount of gas consumed by the transaction. Gas is a measure of computational effort required to execute operations on the Ethereum network. Gas used directly affects the transaction's cost.
- ***Logs:** Events emitted during the contract deployment transaction. These events provide additional information about the transaction and contract creation, which can be useful for monitoring and auditing.

Step 3: Interact with the Contract

1. After deployment, the contract's functions will be available for interaction.
2. Run the **"test"** function, and on its successful execution, you can confirm in your terminal that this function has generated **3 logs**.

EVENT AT 0X403...CABA5 (BLOCKCHA

Balance: 0 ETH

test

Low level interactions

CALLDATA

Transact

[view on etherscan](#)

✓ [block:4065656 txIndex:4] from: 0xD0b...60a4f to: Event.test() 0x5b8...f9863 value: 0 wei data: 0xf8a...8fd6d **logs: 3** hash: 0x2d5...eb80f

Step 4: Confirm the Event Log

In the transaction details, you'll see a "logs" section.

Confirm that the event has been logged with the expected values. Here we have 3 logs since we have emitted 3 events in the test function.

Notice that we have logs in the **json** format, and logs have been emitted sequentially.

```
logs [
  {
    "from": "0x4034da9958f4cfc2CC7d1f8DAcedD35F481CAbA5",
    "topic": "0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0",
    "event": "Log",
    "args": {
      "0": "0xD0b34b98E478138832b8be53b549441Cb2760a4f",
      "1": "Hello Web3.0",
      "sender": "0xD0b34b98E478138832b8be53b549441Cb2760a4f",
      "msg": "Hello Web3.0"
    }
  },
  {
    "from": "0x4034da9958f4cfc2CC7d1f8DAcedD35F481CAbA5",
    "topic": "0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0",
    "event": "Log",
    "args": {
      "0": "0xD0b34b98E478138832b8be53b549441Cb2760a4f",
      "1": "It's all about Decentralization",
      "sender": "0xD0b34b98E478138832b8be53b549441Cb2760a4f",
      "msg": "It's all about Decentralization"
    }
  },
  {
    "from": "0x4034da9958f4cfc2CC7d1f8DAcedD35F481CAbA5",
    "topic": "0xfe1a3ad11e425db4b8e6af35d11c50118826a496df73006fc724cb27f2b99946",
    "event": "AnotherLog",
    "args": {}
  }
]
```

5 Verification of Logs on Etherscan

Step 1: Access Etherscan

- 1) On successful execution of **test** function, navigate to the terminal and click on '**view on etherscan**'

- 2) On clicking, you would be navigated to the Etherscan page for this specific transaction, and you can observe the **recorded** Transaction Details.



The page you would be navigated to looks something like this.



- 1) Logs contain the events emitted by the **specific transaction**.
- 2) If you click on 'To' (your contract address), you would get the total logs emitted by your contract.

Step 2: Specific Transaction Details

On clicking '**Logs**', you would get the logs emitted and recorded by the specific transaction. Here, by the **test** function.

Overview
Logs (3)
State

Transaction Receipt Event Logs

12

Address

0x4034da9958f4cfc2cc7d1f8dacedd35f481caba5

Q

▼

Name

Log (index_topic_1 address sender, string message) View Source

Topics

0

0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0

1

Dec ▼ → 0xD0b34b98E478138832b8Be53b549441Cb2760a4f

Data

message: Hello Web3.0

1

13

Address

0x4034da9958f4cfc2cc7d1f8dacedd35f481caba5

Q

▼

Name

Log (index_topic_1 address sender, string message) View Source

Topics

0

0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0

1

Dec ▼ → 0xD0b34b98E478138832b8Be53b549441Cb2760a4f

Data

message: It's all about Decentralization

2

14

Address

0x4034da9958f4cfc2cc7d1f8dacedd35f481caba5

Q

▼

Topics

0

0xfe1a3ad11e425db4b8e6af35d11c50118826a496df73006fc724cb27f2b99946

Data

3

EVM Log:

EVM uses **Log** to store Solidity events. Each log contains two parts: **topics** and **data**.

Address

0x4034da9958f4cfc2cc7d1f8dacedd35f481caba5

Q

▼

Name

Log (index_topic_1 address sender, string message) View Source

Topics

0

0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0

1

Dec ▼ → 0xD0b34b98E478138832b8Be53b549441Cb2760a4f

Data

message: Hello Web3.0

1. Topics

Topics are used to describe events. Each event contains a **maximum of 4** topics. Typically, the first topic is the event hash: **the hash of the event signature**.

The event hash of Transfer event is calculated as follows:

```
keccak256("Transfer(addresses,address,uint256)")
```

Besides event hash, topics can include **3 indexed parameters**, such as the **from** and **to** parameters in Transfer event.

indexed parameters can be understood as the indexed "**key**" for events, which can be easily queried by programs. The size of each indexed parameter is **32 bytes**.

For the parameter, larger than 32 bytes, such as array and string, the hash of the underlying data is stored.

2. Data

Non-indexed parameters will be stored in the data section of the log. They can be interpreted as "**value**" of the event and can't be retrieved directly.

But they can store data with larger sizes. Therefore, **data** section can be used to store **complex data structures**, such as arrays and strings.

Moreover, data consumes **less gas** compared to topic.

Step 3: Contract Transaction Details

In order to view the Transactions Details of the contract, click on the contract address (as Highlight 2).

Overview Logs (3) State

[This is a Sepolia Testnet transaction only]

Transaction Hash: 0x0c166d10775822bdf5956e418842956bd3a337eba6c5756f5b289b1d954f84f8

Status: Success

Block: 4062907 15 Block Confirmations

Timestamp: 4 mins ago (Aug-10-2023 05:18:00 PM +UTC)

From: 0xD0b34b98E478138832bBbe53b549441Cb2760a4f

To: 0x4034da9958f4cfc2CC7d1f8DAcedD35F481CABa5

Value: 0 ETH (\$0.00)

You would be navigated to the page, as below. Click on the '**Events**'.

Overview

ETH BALANCE

0 ETH

More Info

CONTRACT CREATOR

0xD0b34b...b2760a4f at txn 0x5355e769d9f5ef128...

Transactions Token Transfers (ERC-20) Contract Events

Latest 2 from a total of 2 transactions

Transaction Hash	Method	Block	Age	From
0x0c166d10775822bdf...	Test	4062907	9 mins ago	0xD0b34b...b2760a4f
0x5355e769d9f5ef128...	0x60806040	4062868	18 mins ago	0xD0b34b...b2760a4f

On clicking the **'Events'**, you would be displayed all the logs **generated by contract** and got stored on EVM.

[illegible]

Congratulations! You have successfully learnt the importance of Events, and got hands-on experience with how to check generated logs and verify logs on Etherscan.