# 432.https://stackoverflow.com/questions/71038751/how-to-get-a-contract-by-a-transaction-uising-web3j

**T:**How to get a contract by a transaction uising web3j?

**Q:**I need to check if a transaction is a ERC721/ERC1155 transaction and fetch information like receiving address, token address, value, etc. If I understand correctly, I have to load a contract for the transaction and check if it inherits ERC165 in order to confirm that it is a ERC721/ERC1155 transaction.  
  
Problem: I don't understand how I can get a contract having a transaction object. I also haven't found a way to get token address.  
  
I have an Ethereum node on Infura, I read blocks from there and iterate over transactions. I get a transaction and its receipt. My code looks like this:  
  
var tr = web3j.ethGetTransactionByBlockNumberAndIndex(blockIdParam, transactionIndex).sendAsync().get();var hash = tr.getTransaction().get().getHash();var receipt = web3.ethGetTransactionReceipt(hash).send().getTransactionReceipt();  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Right now I am working in the direction of reading transaction logs, checking their topics and verifying if they include Transfer events. But transfer events are also emitted by ERC20 transactions, so I am a bit confused here.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**There is this statement on web3j documentation "It is not possible to return values from transactional functional calls, regardless of the return type of the message signature. However, it is possible to capture values returned by functions using filters. Please refer to the Filters and Events section for details.". Reference : docs.web3j.io/4.8.7/transactions/…. Probably this is the reason for the null value.

1 **Answer**

**A1:**You're going in the right direction checking the Transfer() event logs. Even though both ERC20 and ERC721 use the same event signature, the ERC721 (NFT) standard defines the 3rd topic (token ID) as indexed, which stores its value in the set of indexed topics. While the ERC20 defines the 3rd topic (amount) as non-indexed, making the total length of indexed topics set just 2.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
I picked a random transaction that contains event logs of both ERC20 and ERC721 transfers. Looks like the logic behind it is payment in the form of an ERC20 token to mint a new ERC721 token.  
  
Note: I'm not a Java dev, so I'll use a JS code in my answer that you can hopefully use as a reference to find the correct syntax and methods in the Java implementation of the Web3 library.  
  
const Web3 = require("web3");const web3 = new Web3("<provider\_url>");// keccak256 of string "Transfer(address,address,uint256)"const TRANSFER\_SIGNATURE = "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef";async function run() { const txHash = "0xfb4dc20b4a8d0b72442a396aced0fc537de757c6aac69609ab3d09d19f9c9aa8"; const txReceipt = await web3.eth.getTransactionReceipt(txHash); for (let log of txReceipt.logs) { if (\_isERC20Log(log)) { // Unindexed topics are concatenated in the `data` field. // There's only one unindexed topic in this case, // so you don't need to do any parsing and just use the whole value. const amount = web3.utils.hexToNumberString(log.data); console.log("ERC20 token amount: ", amount); console.log("ERC20 token contract: ", log.address); } else if (\_isERC721Log(log)) { // 3rd indexed topic // index 0 of the array is the event signature, see `\_isERC721Log()` const tokenId = web3.utils.hexToNumberString(log.topics[3]); console.log("ERC721 token ID: ", tokenId); console.log("ERC721 token contract: ", log.address); } }}function \_isERC20Log(log) { return ( log.topics[0] == TRANSFER\_SIGNATURE && log.topics.length == 3 // index 0 is the signature, and then 2 indexed topics );}function \_isERC721Log(log) { return ( log.topics[0] == TRANSFER\_SIGNATURE && log.topics.length == 4 // index 0 is the signature, and then 3 indexed topics );}run();  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Output:  
  
ERC20 token amount: 40000000000000000000ERC20 token contract: 0x54a7cee7B02976ACE1bdd4aFad87273251Ed34CfERC721 token ID: 12013ERC721 token contract: 0x41cB4a771FdD019ADBF4685bd4885fbBeedE1784  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**Thank you for your explanation. It works perfectly well. I will accept your answer as well as send 300 of bounty once the bounty ends. I also want to ask two more questions: 1) Is the condition of 4 topics + the Transfer(address,address,uint256) signature exclusive to ERC721? Could it happen that there is some other ERC that would produce the same transfer signature and provide 4 topics? 2) Do you know how to get the token address (in your example it is etherscan.io/token/0x41cb4a771fdd019adbf4685bd4885fbbeede1784) ?

**C2:**@SashaShpota 1) Theoretically there could be another ERC that produces the same event signature and the same indexed topics. But all ERCs go through a review by the Ethereum Foundation before they're accepted as an official standard. So I'm assuming that the reviewers would point that out, ask the authors to change the event definition, or possibly decline the proposal if it caused conflict with other existing standards... 2) You can get the address of the emitting contract through the log.address property. See the updated code in the answer.

**C3:**As for the 4 topics: There's a technical limit of max 3 indexed (and practically unlimited number of unindexed) topics because of the way they're processed by the EVM. See this answer for more info. But it's not related to an ERC standard.

**C4:**Note that it is not possible to detect ERC721/ERC1155 transaction based on the event logs only. You need to check if the emitting contract is a compatible. This check cannot be done based on on-chain information only, because usually smart contracts do not advertise what standards they support. Thus, unless you are working well-known contracts this kind of checks alone are buggy.

**C5:**Thanks for the note, Mikko. I agree. There are some edge cases where a contract emits this particular event (or even implements the ERC165 supportsInterface() method returning an expected value), but it's not a 100% guarantee that it follows the rest of the token standard... In other words: There are some dishonest token contracts claiming the transfer in event logs but not reflecting the internal balances, as well as other contracts (possibly unintentionally) emitting these "reserved" events even though they're not token contracts.