Truffle now supports console.log in Solidity smart contracts

By Kingsley Arinze

If you used the popular programming language, JavaScript, before Solidity it's possible you've had a hard time debugging your Solidity smart contracts. You might have encountered bugs in your code that a simple console.log() would have solved, but since Solidity has no built-in way of doing this, you had to find alternative ways to fix the bug.

To get around this problem for Ganache users, we've released theconsole.sol library and updated Ganache to be able to parse and automatically log calls to console.sol, to standardize outputs just likeJavaScript's console.log .

To use Ganache'sconsole.sol together with Truffle during development and testing, you must start a standalone Ganache instance and connect Truffle. This also means that logging outputs from your smart contracts are sent to the Ganache terminal.

Since Truffle has its own built-in Ganache instance that you can easily access when you runtruffle develop , we decided to take this a step further by implementing Ganache's console.log support into Truffle directly.

You no longer have to manually install any package or run a separate Ganache instance to get the logging feature. Logging outputs are also sent to the Truffle terminal.

Installation¶

To follow the content of this post, you will need to have the following requirements:

Node.js

Windows, Linux, or macOS

It is recommended that you download Node.js using a Node version manager to avoid permission errors caused by using sudo to download Truffle. Follow the instructionshere to download a Node version manager for your operating system.

With Node.js installed, you can upgrade by first uninstalling Truffle globally and installing the latest version like so:

npm uninstall -g truffle npm install -g truffle Only runnpm install -g truffle if you do not have a previous version of Truffle installed on your machine.

Configuration¶

There are two configuration options you can use to adjust the behavior of the console.log feature and both options can be set in the truffle configuration file i.e truffle-config.js. They are:

1. solidityLog.displayPrefix
2. : This option sets the prefix for every console.log line. The default is "". It is a useful option if you want to differentiate console.log output from extra verbose text output.
3. solidityLog.preventConsoleLogMigration
4. : This option is a safety measure to prevent accidentally migrating contracts that use console.log to MAINNET. It is set to false by default, so you will have to set it to true manually.

module . exports

=

{

.

.

.

solidityLog :

{

displayPrefix :

' :' ,

// defaults to ""

preventConsoleLogMigration :

true ,

// defaults to false

}

Usage¶

To use console.log in your Solidity contract, you will need to import thetruffle/console.sol library and use it as shown in the code sample below. Note that these logs will appear in your terminal when you runtruffle test ortruffle develop .

// SPDX-License-Identifier: MIT pragma solidity

^ 0.8.13 ; import

"truffle/console.sol" ; contract

SampleContract

{

address

public admin ;

constructor (address

_admin)

{

console. log("SampleContract::constructor\n\tadmin: %o" ,

_admin);

admin

_admin;

} } Migration file for SampleContract:

// 1_migration_file.js const

SampleContract

=

artifacts . require (“ SampleContract ”); module . exports

=

function (deployer ,

network ,

accounts){

const

admin

=

accounts [0];

deployer . deploy (sampleContract ,

admin); } Now if you runTruffle develop followed bymigrate , you will see the following printed on you truffle develop console:

This also applies when you run test cases with Truffle using thetruffle test command, outputs from the console.log statements within your smart contract code would be displayed on your terminal along side test results.

As you can see, usage is very similar to Node.js console.log method as it returns a formatted string using the first argument as a printf-like format string which can contain zero or more format specifiers.

Each specifier is replaced with the converted value from the corresponding argument. Some of the supported specifier includes:

- %s for string
- %d for Number
- %j for JSON and lot more, see theNodejs' util.format() documentation

Conclusion¶

At Truffle, we remain committed to improving and simplifying the user experience for dapp developers in the Web3 ecosystem, by creating developer tools, resources, and educational materials.

To find out more about our suite of developer tools, visit theofficial Truffle website . If you have questions, feel free to start a discussion onTruffle Github Discussions .

We also hold weekly, live-streamed sessions: Web3 Unleashed, where we live code and build, interview leaders from across web3 and discuss important developments around the ecosystem. Keep an eye on ourTwitter for updates on the next session.

You can also find past episodes on theTruffle Youtube channel and theunleashed section of our website if you prefer written materials.