Introducing Truffle DB, Part 1 - Artifact archeology

Introducing Truffle DB, part 1: Artifact archeology¶

Truffle is not a small utility: it compiles your contracts, deploys them to multiple networks, runs your automated tests, provides an interactive debugger for when things go wrong, offers a library for interacting with your contracts on the frontend, … (the list goes on.) With just over 20,000 lines of code and close to thirty discrete software packages, Truffle has a lot going on.

Truffle is “not small” on purpose. The tool oversees the development lifecycle in human developer terms. It keeps track of familiar concepts and hides unnecessary, low-level specifics. That’s a big goal.

So how does this work? Well, Truffle tries to be smart about it. It stores information about your contracts so that it can refer to it later. It tracks compilation results alongside the source code you wrote, maintaining a steady mapping betweendata for the blockchain anddata for developers’ brains .

Right now, this system of record-keeping is served through Truffle’scontract artifacts : the large JSON files that get saved in yourbuild/contracts/ directory. Truffle saves your artifacts when you compile and updates your artifacts with address information when you migrate. Pretty much whenever you use any part of Truffle, these artifacts are read from disk and processed, enabling Truffle to paint itself (and you) a picture of what’s happening in human terms.

Truffle’s artifacts have been enormously useful: the tool simply couldn’t exist without this kind of first-class data capturing. Beyond that, though, being able to count on this data has enabled an ecosystem of other tools and utilities that build on top of Truffle’s functionality using Truffle’s artifacts format. Truffle’s artifacts have served as a common language for representing smart contract development concepts.

Unfortunately, Truffle’s artifacts format has a number of core limitations that prevent Truffle from offering ideal behavior in several key situations. This post describes a few of these limitations. In this post’s follow-up, we’ll share information about a new system to manage this record-keeping, which we’re callingTruffle DB .

So what’s the problem now?¶

For starters: artifact files are huge! Too huge! In addition to Truffle reading your contract artifacts, your dapp’s frontend code may read and use your artifacts to let users connect to any deployed network, etc. Problem is—there’s a ton of JSON. It’s a mess, and it’s in your face too much.

For example: Truffle stores the same abstract syntax tree *(AST) in two different formats, in every single artifact! Most applications do not need this.* for the curious: the AST is the reason your code works whether you wrote it all on a single line or not. Source gets parsed, and the AST is the result. Truffle and other tools rely on the AST for things like debugging, code coverage, and linting. By the byte, almost all of the information in each contract artifact is for internal use or for some specific purpose.Only a small subset of the data in each artifact is necessary for your frontend or for you to think about directly. Thus far, users who have struggled with these large files needed to use a tool likejq or write a custom solution.

In addition to these files being too large, the information is grouped purely by the idea of “contract name”. What if you want to have two contracts with the same name? You can’t. What if you want to debug an old version of your contract, deployed to a testnet? I hope you backed up your old artifacts directory somehow.

Although we’ve implemented plenty of workarounds to deal with these limitations, the problems do not go unnoticed. See some long-outstanding open GitHub issues, for example: *Truffle contract artifacts are too big and contain too much sensitive detail to actually deploy on the web (#1269)* Artifact is overwritten if there are two contracts with the same name (#1087) *Source map indexes are missing in artifacts (#1236)*Allow persistency of artifacts remotely (#1790)

Unfortunately, these issues are blocked by the fundamental design of the contract artifacts.

The root of the problem is that Truffle’s artifacts try to serve two conflicting purposes: they try to serve the developer, but they also try to serve the tool itself.

“Stop talking about data formats”, you say.¶

Right? Ideally, Truffle should silently keep complete records of everything you will ever need, and nothing more. And you, the developer, should only ever come face-to-face with the information you need at any given moment.

So, for some time, we have been working on a plan to improve the state of affairs of Truffle’s record-keeping.

We’ll get into it more in the next post, but Truffle DB aims to provide two things: * a data model: a set of terms that collectively represent various, high-level concepts associated with smart contract development * a flexible interface: a way to query and update your Truffle project’s records

In doing this, we're taking a data-first approach, organizing your project's information through immutable records and a clean separation of concerns. Stay tuned for the next post, where we'll talk about this more and outline our release plan for Truffle DB: what we'll be releasing first, what work still needs to be done, and how we plan to handle backwards compatibility over time.