Opaque data types

Data types whose representations are visible are called transparent. These expose the inner structure of the data so that operations on the data can be understood.

Drawbacks to having transparent data types include lesser type safety (or type soundness) - how a programming language discourages or prevents type errors, such as assigning the wrong type to a variable.

Opaque data types present an interface without sharing the actual, concrete data structure. These can only be manipulated by calling subroutines that have access to knowledge about the missing structure.

The principle of information hiding, segregating design decisions from the underlying software implementation, results in more resilient code. Implementations can be improved or changed completely without the fragility that comes from inner details being depended upon. Defensively coding the parts most likely to change results in more robust code overall.

Midnight opaque data types

Opaque types in Compact are a compact type system feature that allow "foreign" JavaScript data to be stored, passed around, and retrieved on behalf of a DApp (but not inspected by Compact code).

Midnight's Compact language currently supports `Opaque<'string'>` and `Opaque<'Uint8Array'>`. These can be stored i
contract's public state.

NOTE: These are opaque only within Compact. They are transparent in a DApp's JavaScript code. Their representation on-chain is **NOT** hidden - `Uint8Array` is represented by the array of bytes and `string` is represented by its UTF-8 encoding.

Examples

The Bulletin Board example DApp, in the developer tutorial, has an example of opaque data type usage.

- The Compact post circuit is a contract entry point that's called from a DApp and passed an `Opaque<'string'>`. Compact code can't look inside this; it does store it in the contract's public state.
- Later, the Compact take_down circuit reads this value and returns it to the JavaScript or TypeScript caller.

Feedback