Proposing some standard events for SUAPPs to signal orderflow via logs. The idea is to emit these events from various SUAPPs, so we have a standard interface for observing orderflow in L1/L2/block-builder/etc applications.

design

The goal was to emit logs automatically in response to calling functions like sendBundle

in suave-std but I couldn't work out how to do it, since emitting logs changes state. We only change state in the callback of a CCR; not in the first step, which is where those functions are called.

So this is an opt-in design. Following that idea, we could host some standard events like this:

github.com/flashbots/suave-std

define standard orderflow events

flashbots:main

← flashbots:brock/observability

opened 11:25PM - 22 May 24 UTC

[

zeroXbrock
](https://github.com/zeroXbrock)

[+9

-0

](https://github.com/flashbots/suave-std/pull/85/files)

Events are defined in an abstract contract (ideally in suave-std):

abstract contract ObservableOrderflow { event SentTransaction(bytes32 txHash); }

Then SUAPP developers import and emit those events in their own callbacks:

contract Suapp is ObservableOrderflow { event MySuappEvent(uint256 x); modifier confidential() { require(Suave.isConfidential(), "must be called confidentially"); _; } function didSomething(bytes32[] memory txHashes, uint256 x) public confidential { emit MySuappEvent(x); emit SentTransactions(txHashes); } function doSomething() public confidential returns (bytes memory) { // pretend these are tx hashes that we're handling in our SUAPP bytes32[] memory txHashes = new bytes32; for (uint256 i = 0; i < txHashes.length; i++) { txHashes[i] = keccak256(abi.encode("tx", i)); } return abi.encodeWithSelector(this.didSomethingWithTxs.selector, txHashes, 9001); } }

This would be easier if we didn't have to do this in a callback.

Mateusz' idea for async chain state transitions has the potential resolve this constraint.

h/t @dmarz for coming up with the idea