**Solidity Interface**

**Introduction:**

In this part, we’ll learn how to convert Ethereum (ETH) into Dollars (USD) and how to use Interfaces.

**Converting Ethereum into USD:**

We begin by trying to convert the `msg.value`, which is now specified in ETH, into USD. This process requires fetching the current USD market price of Ethereum and using it to convert the `msg.value` amount into USD.

// Function to get the price of Ethereum in USD

Function getPrice() public {}

// Function to convert a value based on the price

Function getConversionRate() public {}

**Chainlink Data Feed:**

Our primary source for Ethereum prices is a \*\*Chainlink Data Feed\*\*. Chainlink Data Feed documentation(<https://docs.chain.link/data-feeds/using-data-feeds>) provides an example of how to interact with a Data Feed contract:

1. AggregatorV3Interface: a contract that takes a Data Feed address as input. This contract maintains the ETH/USD price updated.
2. latestRoundData: a function that returns an `answer`, representing the latest Ethereum price.

To utilize the Price Feed Contract, we need its address and its ABI. The address is available in the Chainlink documentation under the Price Feed Contract Addresses (<https://docs.chain.link/data-feeds/price-feeds/addresses>). For our purposes, we’ll use ETH/USD price feed.

**Interface:**

To obtain the ABI, you can import, compile, and deploy the PriceFeed contract itself. In the previous section, we imported the `SimpleStorage` contract into the `StorageFactory` contract, deployed it, and only then we were able to use its functions.

An alternative method involves the use of an Interface, which defines methods signature without their implementation logic. If compiled, the Price Feed Interface, it would return the ABI of the Price Feed contract itself, which was previously deployed on the blockchain. We don’t need to know anything about the function implementations, only knowing the `AggregatorV3Interface` methods will suffice. The Price Feed interface, called `Aggregator V3 Interface`, can be found in Chainlink’s GitHub repository (<https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/shared/interfaces/AggregatorV3Interface.sol>).

**Note:** Interfaces allow different contracts to interact seamlessly by ensuring they share a common set of functionalities.

We can test the Interface usage by calling the `version()` function:

Function getVersion() public view returns (uint256) {

Return AggregatorV3Interface(0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419).version();

}

**Note:** It’s best to work on testnets only after your deployment is complete, as it can be time and resource consuming.

**Conclusion:**

Using interfaces is a common and effective way to interact with external contracts. First, obtain the interface of the external contract, compile it to get the ABI, and then use the deployed contract’s address. This allows you to call any function available at that address seamlessly.