Contract Class ABI

The Contract Class _Application Binary Interface (ABI)_ is the high-level specification of the interface of a contract. It describes the functions that can be called, their expected parameters and return values, along with the types of these parameters and return values. It allows external sources, both from outside the blockchain and other contracts, to communicate with the contract, by encoding and decoding data according to the contract's interface.

Sources outside the blockchain typically use a JSON representation of the ABI to interact with the contract. This JSON representation is generated from the contract class, and contains an array of items that are either types, functions, or events.

Contracts, on the other hand, use the ABI of another contract directly in Cairo through the _dispatcher_ pattern, which is a specific type that implements methods to call the functions of another contract. These methods are auto-generated, and contain the entire logic required to encode and decode the data to be sent to the contract.

When you interact with a smart contract using a block explorer like [Voyager][voyager] or [Starkscan][starkscan], the JSON ABI is used to properly encode the data you send to the contract and decode the data it returns.

[voyager]: https://voyager.online/
[starkscan]: https://starkscan.co/

Entrypoints

All the functions exposed in the ABI of a contract are called _entrypoints_. An entrypoint is a function that can be called from outside the contract class.

There are 3 different types of entrypoints in a Starknet contract:

- [Public functions][public function], the most common entrypoints, exposed either as `view` or `external` depending on their state mutability.

> Note: An entrypoint can be marked as `view`, but might still modify the contract's state when invoked along with a transaction, if the contract uses low-level calls whose immutability is not enforced by the compiler.

- An optional unique [_constructor_][constructor], which is a specific entrypoint that will be called only once during the deployment of the contract.
- L1-Handlers, functions that can only be triggered by the sequencer after receiving a [message][L1-L2 messaging] from the L1 network whose payload contains an instruction to call a contract.

[public function]: ./ch14-02-contract-functions.md#2-public-functions
[constructor]: ./ch14-02-contract-functions.md#1-constructors
[L1-L2 messaging]: ./ch16-04-L1-L2-messaging.md

A function entrypoint is represented by a _selector_ and a `function_idx` in a Cairo contract class.

Function Selector

While functions are defined with a name, entrypoints are identified by their _selector_. The selector is a unique identifier derived from the function name, and is simply computed as `sn_keccak(function_name)`. As overloading a function with different parameters is not possible in Cairo, the hash of the function name is sufficient to uniquely identify the function to be called.

While this process is often abstracted by libraries and when using dispatchers, know that it's possible to call a function directly by providing its selector, for example when

using a low-level system call like `starknet::call_contract_syscall` or when interacting with an RPC.

Encoding

Smart contracts are written in a high-level language like Cairo, using strong types to inform us about the data manipulated. However, the code executed on the blockchain is compiled into a sequence of low-level CASM instructions. The base data type in Starknet is `felt252`, and that's the only data manipulated at the CASM level. As such, all data must be serialized into `felt252` before being sent to the contract. The ABI specifies how types can be encoded into a sequence of `felt252`, and decoded back into their original form.