The Function Context

What is the context

The context is an object that is made available within every function inAztec.nr . As mentioned in the kernel circuit documentation . At the beginning of a function's execution, the context contains all of the kernel information that application needs to execute. During the lifecycle of a transaction, the function will update the context with each of it's side effects (created notes, nullifiers etc.). At the end of a function's execution the mutated context is returned to the kernel to be checked for validity.

Behind the scenes, Aztec.nr will pass data the kernel needs to and from a circuit, this is abstracted away from the developer. In an developer's eyes; the context is a useful structure that allows access and mutate the state of theAztec blockchain.

On this page, you'll learn

- The details and functionalities of the private context in Aztec.nr
- Difference between the private and public contexts and their unified APIs
- Components of the private context, such as inputs and block header.
- Elements like return values, read requests, new note hashes, and nullifiers in transaction processing
- Differences between the private and public contexts, especially the unique features and variables in the public context

Two contexts, one API

TheAztec blockchain contains two environmentspublic and private .

- Private, for private transactions taking place on user's devices.
- Public, for public transactions taking place on the network's sequencers.

As there are two distinct execution environments, they both require slightly differing execution contexts. Despite their differences; the API's for interacting with each are unified. Leading to minimal context switch when working between the two environments.

The following section will cover both contexts.

The Private Context

The code snippet below shows what is contained within the private context.

```
private-context inputs :

PrivateContextInputs , side_effect_counter :

u32 ,

min_revertible_side_effect_counter :

u32 ,

args_hash :

Field , return_values :

BoundedVec < Field ,

RETURN_VALUES_LENGTH

    ,

note_hash_read_requests :

BoundedVec < SideEffect ,

MAX_NOTE_HASH_READ_REQUESTS_PER_CALL

    , nullifier_read_requests :

BoundedVec < ReadRequest ,
```

MAX_NULLIFIER_READ_REQUESTS_PER_CALL

, nullifier_key_validation_requests :

BoundedVec < NullifierKeyValidationRequest ,

MAX_NULLIFIER_KEY_VALIDATION_REQUESTS_PER_CALL

,

new_note_hashes :

BoundedVec < SideEffect ,

MAX_NEW_NOTE_HASHES_PER_CALL

, new_nullifiers :

BoundedVec < SideEffectLinkedToNoteHash ,

MAX_NEW_NULLIFIERS_PER_CALL

,

private_call_stack_hashes :

BoundedVec < Field ,

MAX_PRIVATE_CALL_STACK_LENGTH_PER_CALL

, public_call_stack_hashes :

BoundedVec < Field ,

MAX_PUBLIC_CALL_STACK_LENGTH_PER_CALL

, new_l2_to_l1_msgs :

BoundedVec < L2ToL1Message ,

MAX_NEW_L2_TO_L1_MSGS_PER_CALL

, Source code: noir-projects/aztec-nr/aztec/src/context/private_context.nr#L39-L58

Private Context Broken Down

Inputs

The context inputs includes all of the information that is passed from the kernel circuit into the application circuit. It contains the following values.

private-context-inputs struct

PrivateContextInputs

{ call_context :

CallContext , historical_header :

Header , private_global_variables :

PrivateGlobalVariables , start_side_effect_counter :

u32 , } Source code: noir-projects/aztec-nr/aztec/src/context/inputs/private_context_inputs.nr#L8-L15 As shown in the snippet, the application context is made up of 4 main structures. The call context, the block header, and the private global variables.

First of all, the call context.

call-context struct

CallContext

{ msg_sender :

AztecAddress , storage_contract_address :

AztecAddress , portal_contract_address :

EthAddress ,

function_selector :

FunctionSelector ,

is_delegate_call :

bool , is_static_call :

bool ,

side_effect_counter :

u32 , } Source code: noir-projects/noir-protocol-circuits/crates/types/src/abis/call_context.nr#L7-L20 The call context contains information about the current call being made:

1. Msg Sender* The message sender is the account (Aztec Contract) that sent the message to the current context. In the first call of the kernel circuit (often the account contract call), this value will be empty. For all subsequent calls the value will be the previous call.

The graphic below illustrates how the message sender changes throughout the kernel circuit iterations.

1. Storage contract address
2.
    - This value is the address of the current context's contract address. This value will be the value of the current contract that is being executed except for when the current call is a delegate call (Warning: This is yet to be implemented). In this case the value will be that of the sending contract.
3. Portal Contract Address
4.
    - This value stores the current contract's linkedportal contract
5.
    - address. As a quick recap, this value is the value of the contracts related ethereum l1 contract address, and will be the recipient of any messages that are created by this contract.
6. Flags
7.
    - Furthermore there are a series of flags that are stored within the application context:* is_delegate_call: Denotes whether the current call is a delegate call. If true, then the storage contract address will be the address of the sender.
8.
    -
        - is_static_call: This will be set if and only if the current call is a static call. In a static call, state changing altering operations are not allowed.

Header

Another structure that is contained within the context is the Header object. In the private context this is a header of a block which used to generate proofs against. In the public context this header is set by sequencer (sequencer executes public calls) and it is set to 1 block before the block in which the transaction is included.

header struct

Header

{ last_archive :

AppendOnlyTreeSnapshot , content_commitment :

ContentCommitment , state :

StateReference , global_variables :

GlobalVariables , } Source code: noir-projects/noir-protocol-circuits/crates/types/src/header.nr#L14-L21

Private Global Variables

In the private execution context, we only have access to a subset of the total global variables, we are restricted to those which can be reliably proven by the kernel circuits.

private-global-variables struct

PrivateGlobalVariables

{ chain_id :

Field , version :

Field , } [Source code: noir-projects/aztec-nr/aztec/src/context/globals/private_global_variables.nr#L5-L10]

Args Hash

To allow for flexibility in the number of arguments supported by Aztec functions, all function inputs are reduced to a singular value which can be proven from within the application.

Theargs_hash is the result of pedersen hashing all of a function's inputs.

Return Values

The return values are a set of values that are returned from an applications execution to be passed to other functions through the kernel. Developers do not need to worry about passing their function return values to thecontext directly asAztec.nr takes care of it for you. See the documentation surroundingAztec.nr [macro expansion] for more details.

return_values : BoundedVec,

Read Requests

New Note Hashes

New note hashes contains an array of all of the note hashes created in the current execution context.

New Nullifiers

New nullifiers contains an array of the new nullifiers emitted from the current execution context.

Nullified Note Hashes

Nullified note hashes is an optimization for introduced to help reduce state growth. There are often cases where note hashes are created and nullified within the same transaction. In these cases there is no reason that these note hashes should take up space on the node's commitment/nullifier trees. Keeping track of nullified note hashes allows us to "cancel out" and prove these cases.

Private Call Stack

The private call stack contains all of the external private function calls that have been created within the current context. Any function call objects are hashed and then pushed to the execution stack. The kernel circuit will orchestrate dispatching the calls and returning the values to the current context.

Public Call Stack

The public call stack contains all of the external function calls that are created within the current context. Like the private call stack above, the calls are hashed and pushed to this stack. Unlike the private call stack, these calls are not executed client side. Whenever the function is sent to the network, it will have the public call stack attached to it. At this point the sequencer will take over and execute the transactions.

New L2 to L1 msgs

New L2 to L1 messages contains messages that are delivered to the[l1 outbox] on the execution of each rollup.

Public Context

The Public Context includes all of the information passed from thePublic VM into the execution environment. It is very similar to the[Private Context] , however it has some minor differences (detailed below).

Public Context Inputs

In the current version of the system, the public context is almost a clone of the private execution context.

public-context-inputs struct

PublicContextInputs

{ call_context :

CallContext , historical_header :

Header ,

public_global_variables :

PublicGlobalVariables ,

start_side_effect_counter :

u32 , } Source code: noir-projects/aztec-nr/aztec/src/context/inputs/public_context_inputs.nr#L6-L15

Public Global Variables

The public global variables are provided by the rollup sequencer and consequently contain some more values than the private global variables.

global-variables struct

GlobalVariables

{ chain_id :

Field , version :

Field , block_number :

Field , timestamp :

Field , coinbase :

EthAddress , fee_recipient :

AztecAddress , } Source code: noir-projects/noir-protocol-circuits/crates/types/src/abis/global_variables.nr#L8-L17 Edit this page

Previous Functions Next Public, Private, and Unconstrained Functions