Migration notes

Aztec is in full-speed development. Literally every version breaks compatibility with the previous ones. This page attempts to target errors and difficulties you might encounter when upgrading, and how to resolve them.

0.27.0

initializer

macro replacesconstructor

Before this version, every contract was required to have exactly oneconstructor private function, that was used for deployment. We have now removed this requirement, and madeconstructor a function like any other.

To signal that a function can be used toinitialize a contract, you must now decorate it with the#[aztec(initializer)] attribute. Initializers are regular functions that set an "initialized" flag (a nullifier) for the contract. A contract can only be initialized once, and contract functions can only be called after the contract has been initialized, much like a constructor. However, if a contract defines no initializers, it can be called at any time. Additionally, you can define as many initializer functions in a contract as you want, both private and public.

To migrate from current code, simply add an initializer attribute to your constructor functions.

+

[aztec(initializer)]

[aztec(private)]

fn constructor() { ... } If your private constructor was used to just call a public internal initializer, then remove the private constructor and flag the public function as initializer. And if your private constructor was an empty one, just remove it.

0.25.0

[Aztec.nr]Static calls

It is now possible to perform static calls from both public and private functions. Static calls forbid any modification to the state, including L2->L1 messages or log generation. Once a static context is set through a static all, every subsequent call will also be treated as static via context propagation.

context . static_call_private_function (targetContractAddress , targetSelector , args) ;

context . static_call_public_function (targetContractAddress , targetSelector , args) ;

[Aztec.nr]Introduction toprelude

A newprelude module to include common Aztec modules and types. This simplifies dependency syntax. For example:

use

dep :: aztec :: protocol_types :: address :: AztecAddress ; use

dep :: aztec :: { context :: { PrivateContext ,

Context } ,

note :: { note_header :: NoteHeader , utils as note_utils } , state_vars :: Map } ; Becomes:

use

dep :: aztec :: prelude :: { AztecAddress ,

NoteHeader ,

PrivateContext ,

Map } ; use

dep :: aztec :: context :: Context ; use

dep :: aztec :: notes :: utils as note_utils ; This will be further simplified in future versions (See4496 for further details).

The prelude consists of

prelude use

dep :: protocol_types :: { address :: { AztecAddress ,

EthAddress } ,

abis :: function_selector :: FunctionSelector } ; use

crate :: { state_vars :: { map :: Map ,

private_immutable :: PrivateImmutable ,

private_mutable :: PrivateMutable , public_immutable :: PublicImmutable ,

public_mutable :: PublicMutable ,

private_set :: PrivateSet , shared_immutable :: SharedImmutable } , log :: { emit_unencrypted_log , emit_encrypted_log } ,

context :: PrivateContext , note :: { note_header :: NoteHeader ,

note_interface :: NoteInterface ,

note_getter_options :: NoteGetterOptions , note_viewer_options :: NoteViewerOptions , utils :: compute_note_hash_and_nullifier as utils_compute_note_hash_and_nullifier } } ; Source code: noir-projects/aztec-nr/aztec/src/prelude.nr#L1-L16

internal

is now a macro

Theinternal keyword is now removed from Noir, and is replaced by anaztec(internal) attribute in the function. The resulting behavior is exactly the same: these functions will only be callable from within the same contract.

Before:

[aztec(private)]

internal fn

double (input :

Field)

->

Field

{ input *

2 } After:

[aztec(private)]

[aztec(internal)]

fn

double (input :

Field)

->

Field

{ input *

2 }

[Aztec.nr]No SafeU120 anymore

Noir now have overflow checks by default. So we don't need SafeU120 like libraries anymore.

You can replace it withU128 instead

Before:

SafeU120::new(0) Now:

U128::from_integer(0)

[Aztec.nr]compute_note_hash_and_nullifier

is now autogenerated

Historically developers have been required to include acompute_note_hash_and_nullifier function in each of their contracts. This function is now automatically generated, and all instances of it in contract code can be safely removed.

It is possible to provide a user-defined implementation, in which case auto-generation will be skipped (though there are no known use cases for this).

[Aztec.nr]Updated naming of state variable wrappers

We have decided to change the naming of our state variable wrappers because the naming was not clear. The changes are as follows:

1. Singleton
2. ->PrivateMutable
3. ImmutableSingleton
4. ->PrivateImmutable
5. StablePublicState
6. ->SharedImmutable
7. PublicState
8. ->PublicMutable

This is the meaning of "private", "public" and "shared": Private: read (R) and write (W) from private, not accessible from public Public: not accessible from private, R/W from public Shared: R from private, R/W from public

Note:SlowUpdates will be renamed toSharedMutable once the implementation is ready.

[Aztec.nr]Authwit updates

Authentication Witnesses have been updates such that they are now cancellable and scoped to a specific consumer. This means that theauthwit nullifier must be emitted from the account contract, which require changes to the interface. Namely, theassert_current_call_valid_authwit_public andassert_current_call_valid_authwit inauth.nr willNO LONGER emit a nullifier. Instead it will call aspend_*_authwit function in the account contract - which will emit the nullifier and perform a few checks. This means that theis_valid functions have been removed to not confuse it for a non-mutating function (static). Furthermore, thecaller parameter of the "authwits" have been moved "further out" such that the account contract can use it in validation, allowing scoped approvals from the account POV. For most contracts, this won't be changing much, but for the account contract, it will require a few changes.

Before:

[aztec(public)]

fn

is_valid_public (message_hash :

Field)

->

Field

{ let actions =

AccountActions :: public (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . is_valid_public (message_hash) }

[aztec(private)]

fn

is_valid (message_hash :

Field)

->

Field

{ let actions =

AccountActions :: private (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . is_valid (message_hash) } After:

[aztec(private)]

fn

spend_private_authwit (inner_hash :

Field)

->

Field

{ let actions =

AccountActions :: private (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . spend_private_authwit (inner_hash) }

[aztec(public)]

fn

spend_public_authwit (inner_hash :

Field)

->

Field

{ let actions =

AccountActions :: public (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . spend_public_authwit (inner_hash) }

0.24.0

Introduce Note Type IDs

Note Type IDs are a new feature which enable contracts to have multipleMap s with different underlying note types, something that was not possible before. This is done almost without any user intervention, though some minor changes are required.

The mandatorycompute_note_hash_and_nullifier now has a fifth parameternote_type_id . Use this instead ofstorage_slot to determine which deserialization function to use.

Before:

```
unconstrained fn
compute_note_hash_and_nullifier ( contract_address :
AztecAddress , nonce :
Field , storage_slot :
Field , preimage :
[ Field ;
TOKEN_NOTE_LEN ] )
->
pub
[ Field ;
4 ]
{ let note_header =
NoteHeader :: new ( contract_address , nonce , storage_slot ) ;
if
( storage_slot == storage . pending_shields . get_storage_slot ( ) )
{ note_utils :: compute_note_hash_and_nullifier ( TransparentNote :: deserialize_content , note_header , preimage ) }
else
if
( note_type_id == storage . slow_update . get_storage_slot ( ) )
{ note_utils :: compute_note_hash_and_nullifier ( FieldNote :: deserialize_content , note_header , preimage ) }
else
{ note_utils :: compute_note_hash_and_nullifier ( TokenNote :: deserialize_content , note_header , preimage ) } Now:
unconstrained fn
compute_note_hash_and_nullifier ( contract_address :
AztecAddress , nonce :
Field , storage_slot :
Field , note_type_id :
Field , preimage :
[ Field ;
TOKEN_NOTE_LEN ] )
->
pub
[ Field ;
```

```
4 ]

{ let note_header =

NoteHeader :: new ( contract_address , nonce , storage_slot ) ;

if

( note_type_id ==

TransparentNote :: get_note_type_id ( ) )

{ note_utils :: compute_note_hash_and_nullifier ( TransparentNote :: deserialize_content , note_header , preimage ) }

else

if

( note_type_id ==

FieldNote :: get_note_type_id ( ) )

{ note_utils :: compute_note_hash_and_nullifier ( FieldNote :: deserialize_content , note_header , preimage ) }

else

{ note_utils :: compute_note_hash_and_nullifier ( TokenNote :: deserialize_content , note_header , preimage ) }
```
TheNoteInterface trait now has an additionalget_note_type_id() function. This implementation will be autogenerated in the future, but for now providing any unique ID will suffice. The suggested way to do it is by running the Python command shown in the comment below:

```
impl

NoteInterface < N

for

MyCustomNote

{ fn

get_note_type_id ( )

->

Field

{ // python -c "print(int(''.join(str(ord(c)) for c in 'MyCustomNote')))" 771216711711511611110978111116101 } }
```

[js]Importing contracts in JS

@aztec/noir-contracts is now@aztec/noir-contracts.js . You'll need to update your package.json & imports.

Before:

```
import

{ TokenContract }

from

"@aztec/noir-contracts/Token" ;
```
Now:

```
import

{ TokenContract }

from

"@aztec/noir-contracts.js/Token" ;
```

[Aztec.nr]aztec-nr contracts location change in Nargo.toml

Aztec contracts are now moved outside of theyarn-project folder and intonoir-projects , so you need to update your imports.

Before:

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.23.0" , directory =

"yarn-project/noir-contracts/contracts/easy_private_token_contract" } Now, update theyarn-project folder fornoir-projects :

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.24.0" , directory =

"noir-projects/noir-contracts/contracts/easy_private_token_contract" }

0.22.0

Note::compute_note_hash

renamed toNote::compute_note_content_hash

Thecompute_note_hash function in of theNote trait has been renamed tocompute_note_content_hash to avoid being confused with the actual note hash.

Before:

impl

NoteInterface

for

CardNote

{ fn

compute_note_hash (self)

->

Field

{ pedersen_hash ([self . owner . to_field ( ) ,] ,

0) } Now:

impl

NoteInterface

for

CardNote

{ fn

compute_note_content_hash (self)

->

Field

{ pedersen_hash ([self . owner . to_field ( ) ,] ,

0) }

Introducecompute_note_hash_for_consumption

andcompute_note_hash_for_insertion

Makes a split in logic for note hash computation for consumption and insertion. This is to avoid confusion between the two, and to make it clear that the note hash for consumption is different from the note hash for insertion (sometimes).

compute_note_hash_for_consumption replacescompute_note_hash_for_read_or_nullify .compute_note_hash_for_insertion is new, and mainly used in `lifecycle.nr``

Note::serialize_content

andNote::deserialize_content added to `NoteInterface

TheNoteInterface have been extended to includeserialize_content anddeserialize_content functions. This is to convey the difference between serializing the full note, and just the content. This change allows you to also add aserialize function to support passing in a complete note to a function.

Before:

impl

Serialize < ADDRESS_NOTE_LEN

for

AddressNote

{ fn

serialize (self)

->

[Field ;

ADDRESS_NOTE_LEN] { [self . address . to_field ( ) ,

self . owner . to_field ( ) ,

self . randomness] } } impl

Deserialize < ADDRESS_NOTE_LEN

for

AddressNote

{ fn

deserialize (serialized_note :

[Field ;

ADDRESS_NOTE_LEN])

->

Self

{ AddressNote

{ address :

AztecAddress :: from_field (serialized_note [0]) , owner :

AztecAddress :: from_field (serialized_note [1]) , randomness : serialized_note [2] , header :

NoteHeader :: empty ( ) , } } Now

impl

NoteInterface < ADDRESS_NOTE_LEN

```
for
AddressNote
{ fn
serialize_content ( self )
->
[ Field ;
ADDRESS_NOTE_LEN ] { [ self . address . to_field ( ) ,
self . owner . to_field ( ) ,
self . randomness ] }
fn
deserialize_content ( serialized_note :
[ Field ;
ADDRESS_NOTE_LEN ] )
->
Self
{ AddressNote
{ address :
AztecAddress :: from_field ( serialized_note [ 0 ] ) , owner :
AztecAddress :: from_field ( serialized_note [ 1 ] ) , randomness : serialized_note [ 2 ] , header :
NoteHeader :: empty ( ) , } } ... }
```

[Aztec.nr]No storage.init() andSerialize

,Deserialize ,NoteInterface as Traits, removal of SerializationMethods and SERIALIZED_LEN

Storage definition and initialization has been simplified. Previously:

```
struct
Storage
{ leader :
PublicState < Leader ,
LEADER_SERIALIZED_LEN
        , legendary_card :
Singleton < CardNote ,
CARD_NOTE_LEN
        , profiles :
Map < AztecAddress ,
Singleton < CardNote ,
CARD_NOTE_LEN
                , test :
Set < CardNote ,
```

CARD_NOTE_LEN

, imm_singleton :

PrivateImmutable < CardNote ,

CARD_NOTE_LEN

, }

impl

Storage

{ fn

init (context :

Context)

->

Self

{ Storage

{ leader :

PublicMutable :: new (context , 1 , LeaderSerializationMethods ,) , legendary_card :

PrivateMutable :: new (context ,

2 ,

CardNoteMethods) , profiles :

Map :: new (context , 3 , | context , slot |

{ PrivateMutable :: new (context , slot ,

CardNoteMethods) } ,) , test :

Set :: new (context ,

4 ,

CardNoteMethods) , imm_singleton :

PrivateImmutable :: new (context ,

4 ,

CardNoteMethods) , } } } Now:

struct

Storage

{ leader :

PublicMutable < Leader

, legendary_card :

Singleton < CardNote

, profiles :

Map < AztecAddress ,

Singleton < CardNote

, test :

Set < CardNote

, imm_singleton :

PrivateImmutable < CardNote

, } For this to work, Notes must implement Serialize, Deserialize and NoteInterface Traits. Previously:

use

dep :: aztec :: protocol_types :: address :: AztecAddress ; use

dep :: aztec :: { note :: { note_header :: NoteHeader , note_interface :: NoteInterface , utils :: compute_note_hash_for_read_or_nullify , } , oracle :: { nullifier_key :: get_nullifier_secret_key , get_public_key :: get_public_key , } , log :: emit_encrypted_log , hash :: pedersen_hash , context :: PrivateContext , } ;

// Shows how to create a custom note

global CARD_NOTE_LEN :

Field

=

1 ;

impl

CardNote

{ pub

fn

new (owner :

AztecAddress)

->

Self

{ CardNote

{ owner , } }

pub

fn

serialize (self)

->

[Field ;

CARD_NOTE_LEN]

{ [self . owner . to_field ( )] }

pub

fn

deserialize (serialized_note :

[Field ;

CARD_NOTE_LEN])

->

Self

```
{ CardNote

{ owner :

AztecAddress :: from_field ( serialized_note [ 1 ] ) , } }

pub

fn

compute_note_hash ( self )

->

Field

{ pedersen_hash ( [ self . owner . to_field ( ) , ] , 0 ) }

pub

fn

compute_nullifier ( self , context :

& mut

PrivateContext )

->

Field

{ let note_hash_for_nullify =

compute_note_hash_for_read_or_nullify ( CardNoteMethods ,

self ) ; let secret = context . request_nullifier_secret_key ( self . owner ) ; pedersen_hash ( [ note_hash_for_nullify , secret .
high , secret . low , ] , 0 ) }

pub

fn

compute_nullifier_without_context ( self )

->

Field

{ let note_hash_for_nullify =

compute_note_hash_for_read_or_nullify ( CardNoteMethods ,

self ) ; let secret =

get_nullifier_secret_key ( self . owner ) ; pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }

pub

fn

set_header ( & mut

self , header :

NoteHeader )

{ self . header = header ; }

// Broadcasts the note as an encrypted log on L1. pub

fn

broadcast ( self , context :
```

```
& mut
PrivateContext , slot :
Field )
{ let encryption_pub_key =
get_public_key ( self . owner ) ; emit_encrypted_log ( context , ( * context ) . this_address ( ) , slot , encryption_pub_key ,
self . serialize ( ) , ) ; } }
fn
deserialize ( serialized_note :
[ Field ;
CARD_NOTE_LEN ] )
->
CardNote
{ CardNote :: deserialize ( serialized_note ) }
fn
serialize ( note :
CardNote )
->
[ Field ;
CARD_NOTE_LEN ]
{ note . serialize ( ) }
fn
compute_note_hash ( note :
CardNote )
->
Field
{ note . compute_note_hash ( ) }
fn
compute_nullifier ( note :
CardNote , context :
& mut
PrivateContext )
->
Field
{ note . compute_nullifier ( context ) }
fn
compute_nullifier_without_context ( note :
CardNote )
->
```

Field

{ note . compute_nullifier_without_context ( ) }

fn

get_header (note :

CardNote)

->

NoteHeader

{ note . header }

fn

set_header (note :

& mut

CardNote , header :

NoteHeader)

{ note . set_header (header) }

// Broadcasts the note as an encrypted log on L1. fn

broadcast (context :

& mut

PrivateContext , slot :

Field , note :

CardNote)

{ note . broadcast (context , slot) ; }

global CardNoteMethods

=

NoteInterface

{ deserialize , serialize , compute_note_hash , compute_nullifier , compute_nullifier_without_context , get_header , set_header , broadcast , } ; Now:

use

dep :: aztec :: { note :: { note_header :: NoteHeader , note_interface :: NoteInterface , utils :: compute_note_hash_for_read_or_nullify , } , oracle :: { nullifier_key :: get_nullifier_secret_key , get_public_key :: get_public_key , } , log :: emit_encrypted_log , hash :: pedersen_hash , context :: PrivateContext , protocol_types :: { address :: AztecAddress , traits :: { Serialize ,

Deserialize ,

Empty } } } ;

// Shows how to create a custom note

global CARD_NOTE_LEN :

Field

=

1 ;

impl

```
CardNote

{ pub

fn

new ( owner :

AztecAddress )

->

Self

{ CardNote

{ owner , } } }

impl

NoteInterface

for

CardNote

{ fn

compute_note_content_hash ( self )

->

Field

{ pedersen_hash ( [ self . owner . to_field ( ) , ] , 0 ) }

fn

compute_nullifier ( self , context :

& mut

PrivateContext )

->

Field

{ let note_hash_for_nullify =

compute_note_hash_for_read_or_nullify ( self ) ; let secret = context . request_nullifier_secret_key ( self . owner ) ;
pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }

fn

compute_nullifier_without_context ( self )

->

Field

{ let note_hash_for_nullify =

compute_note_hash_for_read_or_nullify ( self ) ; let secret =

get_nullifier_secret_key ( self . owner ) ; pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }

fn

set_header ( & mut

self , header :

NoteHeader )
```

{ self . header = header ; }

fn

get_header (note :

CardNote)

->

NoteHeader

{ note . header }

fn

serialize_content (self)

->

[Field ;

CARD_NOTE_LEN] { [self . owner . to_field ( )] }

fn

deserialize_content (serialized_note :

[Field ;

CARD_NOTE_LEN])

->

Self

{ AddressNote

{ owner :

AztecAddress :: from_field (serialized_note [0]) , header :

NoteHeader :: empty ( ) , } }

// Broadcasts the note as an encrypted log on L1. fn

broadcast (self , context :

& mut

PrivateContext , slot :

Field)

{ let encryption_pub_key =

get_public_key (self . owner) ; emit_encrypted_log (context , (* context) . this_address ( ) , slot , encryption_pub_key , self . serialize ( ) ,) ; } } Public state must implement Serialize and Deserialize traits.

It is still possible to manually implement the storage initialization (for custom storage wrappers or internal types that don't implement the required traits). For the above example, theimpl Storage section would look like this:

impl

Storage

{ fn

init (context :

Context)

->

Self

{ Storage

{ leader :

PublicMutable :: new (context , 1) , legendary_card :

PrivateMutable :: new (context ,

2) , profiles :

Map :: new (context , 3 , | context , slot |

{ PrivateMutable :: new (context , slot) } ,) , test :

Set :: new (context ,

4) , imm_singleton :

PrivateImmutable :: new (context ,

4) , } } }

0.20.0

[Aztec.nr]Changes toNoteInterface

1. Changingcompute_nullifier()
2. tocompute_nullifier(private_context: PrivateContext)
3. This API is invoked for nullifier generation within private functions. When using a secret key for nullifier creation, retrieve it through:
4. private_context.request_nullifier_secret_key(account_address)
5. The private context will generate a request for the kernel circuit to validate that the secret key does belong to the account.
6. Before:
7. pub
8. fn
9. compute_nullifier
10. (
11. self
12.)
13. ->
14. Field
15. {
16. let
17. secret
18. =
19. oracle
20. .
21. get_secret_key
22. (
23. self
24. .
25. owner
26.)
27. ;
28. pedersen_hash
29. (
30. [
31. self
32. .
33. value
34. ,
35. secret
36. .
37. low
38. ,

39. secret
40. .
41. high
42. ,
43.]
44.)
45. }
46. Now:
47. pub
48. fn
49. compute_nullifier
50. (
51. self
52. ,
53. context
54. :
55. &
56. mut
57. PrivateContext
58.)
59. ->
60. Field
61. {
62. let
63. secret
64. =
65. context
66. .
67. request_nullifier_secret_key
68. (
69. self
70. .
71. owner
72.)
73. ;
74. pedersen_hash
75. (
76. [
77. self
78. .
79. value
80. ,
81. secret
82. .
83. low
84. ,
85. secret
86. .
87. high
88. ,
89.]
90.)
91. }
92. New APIcompute_nullifier_without_context()
93. .
94. This API is used within unconstrained functions where the private context is not available, and using an unverified nullifier key won't affect the network or other users. For example, it's used incompute_note_hash_and_nullifier()
95. to compute values for the user's own notes.
96. pub
97. fn
98. compute_nullifier_without_context
99. (
100. self
101.)
102. ->
103. Field
104. {
105. let

106. secret
107. =
108. oracle
109. .
110. get_nullifier_secret_key
111. (
112. self
113. .
114. owner
115.)
116. ;
117. pedersen_hash
118. (
119. [
120. self
121. .
122. value
123. ,
124. secret
125. .
126. low
127. ,
128. secret
129. .
130. high
131. ,
132.]
133.)
134. }
135. Note that theget_secret_key
136. oracle API has been renamed toget_nullifier_secret_key
137. .

0.18.0

[Aztec.nr]Removeprotocol_types

from Nargo.toml

Theprotocol_types package is now being reexported fromaztec . It can be accessed throughdep::aztec::protocol_types .

aztec

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.28.1" ,

directory

"yarn-project/aztec-nr/aztec"

}

[Aztec.nr]key type definition in Map

TheMap class now requires defining the key type in its declaration whichmust implement theToField trait.

Before:

struct

Storage

{ balances :

Map < PublicMutable < Field ,

FIELD_SERIALIZED_LEN

}

let user_balance = balances . at (owner . to_field ( )) Now:

struct

Storage

{ balances :

Map < AztecAddress ,

PublicState < Field ,

FIELD_SERIALIZED_LEN

}

let user_balance = balances . at (owner)

[js]Updated function names

- waitForSandbox
- renamed towaitForPXE
- in@aztec/aztec.js
- getSandboxAccountsWallets
- renamed togetInitialTestAccountsWallets
- in@aztec/accounts/testing

0.17.0

[js]New@aztec/accounts

package

Before:

import

{ getSchnorrAccount }

from

"@aztec/aztec.js" ;

// previously you would get the default accounts from theaztec.js package: Now, import them from the new package@aztec/accounts

import

{ getSchnorrAccount }

from

"@aztec/accounts" ;

Typed Addresses

Address fields in Aztec.nr now is of typeAztecAddress as opposed toField

Before:

unconstrained fn

compute_note_hash_and_nullifier (contract_address :

Field , nonce :

Field , storage_slot :

Field , serialized_note :

[Field ;

VALUE_NOTE_LEN])

->

[Field ;

4]

{ let note_header =

NoteHeader :: new (_address , nonce , storage_slot) ; ... Now:

unconstrained fn

compute_note_hash_and_nullifier (contract_address :

AztecAddress , nonce :

Field , storage_slot :

Field , serialized_note :

[Field ;

VALUE_NOTE_LEN])

->

pub

[Field ;

4]

{ let note_header =

NoteHeader :: new (contract_address , nonce , storage_slot) ; Similarly, there are changes when using aztec.js to call functions.

To parse aAztecAddress to BigInt, use.inner Before:

const tokenBigInt =

await bridge . methods . token ( ) . view ( ) ; Now:

const tokenBigInt =

(await bridge . methods . token ( ) . view ( )) . inner ;

[Aztec.nr]Addprotocol_types

to Nargo.toml

aztec

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.28.1" ,

directory

"yarn-project/aztec-nr/aztec"

} protocol_types

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.28.1" ,

directory

"yarn-project/noir-protocol-circuits/crates/types" }

[Aztec.nr]moving compute_address func to AztecAddress

Before:

let calculated_address =

compute_address (pub_key_x , pub_key_y , partial_address) ; Now:

let calculated_address =

AztecAddress :: compute (pub_key_x , pub_key_y , partial_address) ;

[Aztec.nr]movingcompute_selector

to FunctionSelector

Before:

let selector =

compute_selector ("_initialize((Field))") ; Now:

let selector =

FunctionSelector :: from_signature ("_initialize((Field))") ;

[js]Importing contracts in JS

Contracts are now imported from a file with the type's name.

Before:

import

{ TokenContract }

from

"@aztec/noir-contracts/types" ; Now:

import

{ TokenContract }

from

"@aztec/noir-contracts/Token" ;

[Aztec.nr]Aztec example contracts location change in Nargo.toml

Aztec contracts are now moved outside of thesrc folder, so you need to update your imports.

Before:

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.16.9" , directory =

"noir-projects/noir-contracts/contracts/easy_private_token_contract" } Now, just remove thesrc folder,:

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.17.0" , directory =

"noir-projects/noir-contracts/contracts/easy_private_token_contract" } Edit this page

Previous Limitations Next Glossary