Cancelling Deposits

A special type of error is anunderpriced transaction - it means that a message is inserted on L1, but the attached fee is too low to be included in a rollup block. In such a case your funds could be stuck in the portal and not minted on L2 (lost forever!)

To address this, the Inbox supports cancelling messages after a deadline. However, this must be called by the portal itself, as it will need to "undo" the state changes is made (for example by sending the tokens back to the user).

In yourTokenPortal.sol smart contract, paste this:

token_portal_cancel /* * *@notice Cancel a public depositToAztec L1 to L2 message * @dev only callable by the* _canceller *of the message * @param _to - The aztec address of the recipient in the original message * @param _amount - The amount to deposit per the original message * @param _deadline - The timestamp after which the entry can be cancelled * @param _secretHash - The hash of the secret consumable message in the original message * @param _fee - The fee paid to the sequencer * @return The key of the entry in the Inbox* / function

cancelL1ToAztecMessagePublic (bytes32 _to , uint256 _amount , uint32 _deadline , bytes32 _secretHash , uint64 _fee)

external

returns

(bytes32)

{ IInbox inbox = registry . getInbox ( ) ; DataStructures . L1Actor memory l1Actor = DataStructures . L1Actor (address (this) , block . chainid) ; DataStructures . L2Actor memory l2Actor = DataStructures . L2Actor (l2TokenAddress ,

1) ; DataStructures . L1ToL2Msg memory message = DataStructures . L1ToL2Msg ({ sender : l1Actor , recipient : l2Actor , content : Hash . sha256ToField (abi . encodeWithSignature ("mint_public(bytes32,uint256,address)" , _to , _amount , msg . sender)) , secretHash : _secretHash , deadline : _deadline , fee : _fee }) ; bytes32 entryKey = inbox . cancelL2Message (message ,

address (this)) ; // release the funds to msg.sender (since the content hash (& entry key) is derived by hashing the caller, // we confirm that msg.sender is same as _canceller supplied when creating the message) underlying . transfer (msg . sender , _amount) ; return entryKey ; }

/* * *@notice Cancel a private depositToAztec L1 to L2 message * @dev only callable by the* _canceller *of the message * @param _secretHashForRedeemingMintedNotes - The hash of the secret to redeem minted notes privately on Aztec * @param _amount - The amount to deposit per the original message * @param _deadline - The timestamp after which the entry can be cancelled * @param _secretHashForL2MessageConsumption - The hash of the secret consumable L1 to L2 message * @param _fee - The fee paid to the sequencer * @return The key of the entry in the Inbox* / function

cancelL1ToAztecMessagePrivate (bytes32 _secretHashForRedeemingMintedNotes , uint256 _amount , uint32 _deadline , bytes32 _secretHashForL2MessageConsumption , uint64 _fee)

external

returns

(bytes32)

{ IInbox inbox = registry . getInbox ( ) ; DataStructures . L1Actor memory l1Actor = DataStructures . L1Actor (address (this) , block . chainid) ; DataStructures . L2Actor memory l2Actor = DataStructures . L2Actor (l2TokenAddress ,

1) ; DataStructures . L1ToL2Msg memory message = DataStructures . L1ToL2Msg ({ sender : l1Actor , recipient : l2Actor , content : Hash . sha256ToField (abi . encodeWithSignature ("mint_private(bytes32,uint256,address)" , _secretHashForRedeemingMintedNotes , _amount , msg . sender)) , secretHash : _secretHashForL2MessageConsumption , deadline : _deadline , fee : _fee }) ; bytes32 entryKey = inbox . cancelL2Message (message ,

address (this)) ; // release the funds to msg.sender (since the content hash (& entry key) is derived by hashing the caller, // we confirm that msg.sender is same as _canceller supplied when creating the message) underlying . transfer (msg . sender , _amount) ; return entryKey ; } Source code: l1-contracts/test/portals/TokenPortal.sol#L104-L184 To cancel a message, the portal must reconstruct it - this way we avoid storing messages in the portal itself. Note that just as with deposits we need to support cancelling messages for minting privately and publicly.

Note that the portal usesmsg.sender as the canceller when computing the secret hash. This is an access control mechanism to restrict only the intended address to cancel a message.

Once the message is cancelled on the inbox, we return the funds back to the user.

The inbox requires each message to provide a deadline by which a message must be consumed. After this time, if the message is still not consumed, the message can be cancelled.

In the next step we will write L1 and L2 logic to withdraw funds from L2 to L1Edit this page

Previous Minting tokens on Aztec Next Withdrawing to L1