Handling Failed xCalls

There are a few failure conditions to watch out for when usingxcall .

High Slippage Conditions

When tokens are bridged through Connext, slippage can impact thexcall during the swap on origin or destination. If slippage is too high on the origin swap, thexcall will just revert. If slippage is too high on the destination swap (after it has already gone through the origin swap), then there are a couple options to consider.

- Cancel the transfer and bridge back to origin (sender will lose any funds related to the origin slippage) [not available yet].
- Wait it out until slippage conditions improve (relayers will continuously re-attempt the transfer execution).
- Increase the slippage tolerance.
-

Increasing Slippage Tolerance

The_delegate parameter ofxcall is an address that has rights to update the original slippage tolerance by calling Connext'sforceUpdateSlippage function with the following signature:

```
Copy functionforceUpdateSlippage(TransferInfocalldata_params,uint256_slippage)external;
```

TheTransferInfo struct that must be supplied:

```
Copy structTransferInfo{ uint32originDomain; uint32destinationDomain; uint32canonicalDomain; addressto; addressdelegate; boolreceiveLocal; bytescallData; uint256slippage; addressoriginSender; uint256bridgedAmt; uint256normalizedIn; uint256nonce; bytes32canonicalId; }
```

The parameters inTransferInfo must match the same parameters used in the originalxcall . It's possible to obtain the original parameters by querying the subgraph (originor destination) with thetransferId associated with thexcall .

The Connext SDK also exposes anupdateSlippage method for this.

Low Relayer Fee

If the estimated relayer fee paid was too low, then users may have toincrease the relayer fee after thexcall has been sent.

Reverts on Receiver Contract

If the call on the receiver contract (also referred to as "target" contract) reverts, funds sent in with the call will end up on the receiver contract. To avoid situations where user funds get stuck on the receivers, developers should build any contract implementingIXReceive defensively.

Ultimately, the goal should be to handle any revert-susceptible code and ensure that the logical owner of fundsalways maintains agency over them.

Try/Catch with External Calls

One way to guard against unexpected reverts is to usetry/catch statements which allow contracts to handle errors on external function calls.

```
Copy contractTargetContract{ ... functionxReceive( bytes32_transferId, uint256_amount, address_asset, address_originSender, uint32_origin, bytesmemory_callData )externalreturns(bytesmemory) { try{ someExternalCall(); }catch{ // Make sure funds are delivered to logical owner on failing external calls } } }
```

Options for Funds on Receiver

We recommend that xApp developers consider recovery options in case of reverting calls on the receiver. For example, there could be an internal accounting structure to recordtransferId s and allow rightfuloriginSender s to rescue their funds from the receiver contract. Note that this approach requires authentication and would causexcall s to go through the slow path.

Alternatively, the protocol can implement an allowlist for addresses that are able to rescue funds and redirect them to users.

Connext is actively researching standards and best practices for receiver contracts. Reach out to us if you questions!

Previous Authentication Next Nested xCalls Last updated9 months ago On this page *High Slippage Conditions * Increasing Slippage Tolerance * Low Relayer Fee * Reverts on Receiver Contract * Try/Catch with External Calls * Options for Funds on Receiver

Edit on GitHub