Entities

Asset

```

Copy typeAsset@entity{ id:ID!"The local asset address." key:Bytes"The key associated with the local asset." canonicalId:Bytes"The canonical identifier for the asset." canonicalDomain:BigInt"The domain of the canonical identifier for the asset." adoptedAsset:Bytes"The address of the adopted asset, if applicable." localAsset:Bytes"The address of the local asset." blockNumber: BigInt "The block # associated with the local asset, needed in case multiple locals are stored under the same canonicalId." status:AssetStatus"The status of the asset, as defined by the AssetStatus enumeration." }

```

AssetStatus

```

Copy typeAssetStatus@entity{ id:ID!"The key associated with the asset status." status: Boolean "The status of the asset, represented as a boolean value. True means the asset is active, while false means it is inactive." }

```

AssetBalance

```

Copy typeAssetBalance@entity{ id:ID!"The unique identifier for the asset balance, constructed as 'key-router_address'." amount:BigInt!"The amount of the asset held in the balance." router:Router!"The router associated with the asset balance." asset:Asset!"The asset associated with the balance." feesEarned:BigInt!"The total amount of fees earned for the asset balance." }

```

Router

```

Copy typeRouter@entity{ id:ID!"The unique identifier for the router." isActive:Boolean!"A boolean indicating whether the router is currently active or not." owner:Bytes"The address of the current owner of the router." recipient:Bytes"The address of the current recipient of router fees." proposedOwner:Bytes"The address of the proposed new owner of the router, if any." proposedTimestamp:BigInt"The timestamp of the proposed ownership transfer, if any." assetBalances: [AssetBalance!]!@derivedFrom(field:"router")"A list of asset balances associated with the router." }

```

Setting

```

Copy typeSetting@entity{ id:ID!"The unique identifier for the setting." maxRoutersPerTransfer:BigInt!"The maximum number of routers allowed in a single transfer." caller:Bytes!"The address of the caller associated with the setting." }

```

Relayer

```

Copy typeRelayer@entity{ id:ID!"The ID of the relayer" isActive:Boolean!"Indicates whether the relayer is currently active" relayer:Bytes"The address of the relayer" }

```

Sequencer

```

Copy typeSequencer@entity{ id:ID!"Unique identifier for the sequencer entity" isActive:Boolean!"Flag indicating if the sequencer is active or not" sequencer:Bytes"Address of the sequencer" }
```

```

TransferStatus

```

Copy enumTransferStatus{ XCalled Executed Reconciled CompletedSlow CompletedFast }

```

OriginTransfer

```

Copy typeOriginTransfer@entity{ id:ID!"The unique identifier for the origin transfer."

Meta

chainId:BigInt"The ID of the chain on which the transfer is taking place." transferId:Bytes"The unique identifier for the transfer." nonce:BigInt"A unique nonce for the transfer, used to prevent replay attacks." status:TransferStatus"The status of the transfer, represented as a TransferStatus enum." messageHash:Bytes"The hash of the transfer message."

CallParams

originDomain:BigInt"The domain of the origin chain." destinationDomain:BigInt"The domain of the destination chain." canonicalDomain:BigInt"The domain of the canonical asset." to:Bytes"The address of the destination contract." delegate:Bytes"The address of the delegate contract." receiveLocal:Boolean"A boolean indicating whether the transfer is receiving a local asset." callData:Bytes"The calldata associated with the transfer." slippage:BigInt"The maximum amount of slippage allowed for the transfer." originSender:Bytes"The address of the sender on the origin chain." bridgedAmt:BigInt"The amount of the asset being transferred." normalizedIn:BigInt"The normalized amount of the asset being transferred." canonicalId:Bytes"The canonical ID of the asset being transferred."

Asset

asset:Asset"The asset being transferred."

Message

message:OriginMessage"The message associated with the transfer."

Relayer Fee paid by user

relayerFee:BigInt"The fee paid by the user for the transfer." bumpRelayerFeeCount:BigInt"The number of times the relayer fee has been bumped."

XCalled Transaction

caller:Bytes"The address of the caller." transactionHash:Bytes"The hash of the transaction." timestamp:BigInt"The timestamp of the transaction." gasPrice:BigInt"The price of gas for the transaction." gasLimit:BigInt"The limit for gas usage in the transaction." blockNumber:BigInt"The number of the block in which the transaction was included." txOrigin:Bytes"The address of the original transaction sender." }

```

DestinationTransfer

```

Copy typeDestinationTransfer@entity{ id:ID!"unique identifier of the destination transfer"

Meta

chainId:BigInt"chain id of the transfer" transferId:Bytes"unique identifier for the transfer" nonce:BigInt"number used to prevent replay attacks" status:TransferStatus"status of the transfer" routers: [Router!]!"list of routers used for the transfer"

CallParams

originDomain:BigInt"domain of the origin chain" destinationDomain:BigInt"domain of the destination chain" canonicalDomain:BigInt"canonical domain of the asset" to:Bytes"recipient address on the destination chain" delegate:Bytes"optional delegate address on the destination chain" receiveLocal:Boolean"whether or not to receive the asset locally on the destination chain" callData:Bytes"optional call data for the recipient" slippage:BigInt"slippage allowance for routers to adjust exchange rate" bumpSlippageCount:BigInt"count of times the slippage allowance was increased" originSender:Bytes"sender address on the origin chain" bridgedAmt:BigInt"amount of asset bridged from the origin chain" normalizedIn:BigInt"amount of asset normalized to its canonical representation" canonicalId:Bytes"unique identifier for the asset"

Asset

asset:Asset"asset being transferred" amount:BigInt"amount of asset being transferred"

calculated

routersFee:BigInt"total fee paid to routers for the transfer"

Executed Transaction

executedCaller:Bytes"address of the user who initiated the transaction" executedTransactionHash:Bytes"hash of the executed transaction" executedTimestamp:BigInt"timestamp of the executed transaction" executedGasPrice:BigInt"gas price used for the executed transaction" executedGasLimit:BigInt"gas limit used for the executed transaction" executedBlockNumber:BigInt"block number of the executed transaction" executedTxOrigin:Bytes"address of the user who initiated the transaction on the destination chain"

Reconciled Transaction

reconciledCaller:Bytes"address of the user who initiated the reconciled transaction" reconciledTransactionHash:Bytes"hash of the reconciled transaction" reconciledTimestamp:BigInt"timestamp of the reconciled transaction" reconciledGasPrice:BigInt"gas price used for the reconciled transaction" reconciledGasLimit:BigInt"gas limit used for the reconciled transaction" reconciledBlockNumber:BigInt"block number of the reconciled transaction" reconciledTxOrigin:Bytes"address of the user who initiated the reconciled transaction on the origin chain" }

```

OriginMessage

```

Copy typeOriginMessage@entity{ id:ID!"unique identifier for each instance of the OriginMessage"

origin transfer data

transferId:Bytes"ID of the origin transfer" destinationDomain:BigInt"domain where the transfer is being sent to"

Dispatch Transaction

leaf:Bytes"leaf of the Merkle tree of the message" index:BigInt"index of the message in the Merkle tree" message:Bytes"the message data" root:Bytes"root of the Merkle tree of the message" transactionHash:Bytes"hash of the transaction that dispatched the message" blockNumber:BigInt"block number of the transaction that dispatched the message"

root count RD

rootCount:RootCount"reference to the root count for this message" }

```

AggregateRoot

```

Copy typeAggregateRoot@entity{ id:ID!"Unique identifier of the entity" root:Bytes!"The root hash of a Merkle tree containing transaction messages" blockNumber:BigInt!"The block number where the root was aggregated" }

```

ConnectorMeta

```

Copy typeConnectorMeta@entity{ id:ID!"ConnectorMeta" spokeDomain:BigInt"domain of the spoke network" hubDomain:BigInt"domain of the hub network"

amb:Bytes"address of the AMB contract used for bridging" rootManager:Bytes"address of the root manager contract used for managing bridge roots" mirrorConnector:Bytes"address of the MirrorConnector contract used for interacting with the Mirror network" }

```

RootCount

```

Copy typeRootCount@entity{ id:ID!"unique identifier for the root count, typically a concatenation of the spoke and hub domain IDs" count:BigInt"the current root count for the given spoke and hub domain pair" }

```

RootMessageSent

```

Copy typeRootMessageSent@entity{ id:ID!"Unique identifier for the root message sent"

spokeDomain:BigInt"Domain ID for the spoke chain" hubDomain:BigInt"Domain ID for the hub chain" root:Bytes"Root hash for the message" count:BigInt"Number of messages in the root"

MessageSent Transaction

caller:Bytes"Address of the transaction sender" transactionHash:Bytes"Hash of the transaction" timestamp:BigInt"Timestamp of the transaction" gasPrice:BigInt"Gas price for the transaction" gasLimit:BigInt"Gas limit for the transaction" blockNumber:BigInt"Block number of the transaction" }

```

Previous Subgraphs Next Sample Queries On this page * Asset * AssetStatus * AssetBalance * Router * Setting * Relayer * Sequencer * TransferStatus * OriginTransfer * DestinationTransfer * OriginMessage * AggregateRoot * ConnectorMeta * RootCount * RootMessageSent

Edit on GitHub
```