Solidity API

EscrowERC20

DPLAT ERC20 escrow abstract contract

treasury

```
address treasury
```

vERC20Addresses

```
mapping(uint256 => address) vERC20Addresses
```

mapping of the vERC20 contract address for the chain

ulAsset

```
contract IERC20 ulAsset
```

The underlying ERC20 token contract

relayWrapper

```
contract IRelayWrapper relayWrapper
```

RelayWrapper contract address

Escrow can only use this trusted RelayWrapper to perform deposit/withdraw

nonce

```
uint256 nonce
```

nonce used for deposit/withdraw operations. Incremented for every successful deposit or withdraw

Action

```
enum Action {
  NONE,
  DEPOSIT,
  WITHDRAW
}
```

PendingAction

```
struct PendingAction {
  enum EscrowERC20.Action action;
  address nAddress;
  address rAddress;
  uint256 chainId;
  uint256 amount;
}
```

pendingAction

```
mapping(bytes32 => struct EscrowERC20.PendingAction) pendingAction
```

mapping of current deposit/withdraw operations for which callback has not yet been received

action: EscrowERC20.Action that is being performed
nAddress: Address from which ERC20 tokens are deposited (for Action.DEPOSIT) or tokens are received into (for Action.WITHDRAW)
rAddress: Address to which vERC20 tokens are deposited (for Action.DEPOSIT) or tokens are received into (for Action.WITHDRAW)
chainId: chain id of the remote chain
amount: Amount of tokens that are deposited or withdrawn
This is updated on successful deposit/withdraw and cleared when callback is received

constructor

```
constructor(address forwarder_, contract IERC20 asset_) internal
```

ZBYT ERC20 Escrow constructor

Parameters

Name	Type	Description
forwarder_	address	Forwarder contact address

Name	Type	Description
asset_	contract IERC20	Underlying ERC20 asset address

receive

```
receive() external payable
```

receive function

onlyRelay

```
modifier onlyRelay()
```

Modifier to enforce call only from valid relay contract

getNonce

```
function getNonce() public view returns (uint256)
```

Get the latest nonce

nonce is incremented for every successful deposit or withdraw

setTreasuryAddress

```
function setTreasuryAddress(address treasury_) public
```

Set the treasury address

Parameters

Name	Type	Description
treasury_	address	Treasury address

_setvERC20Address

```
function _setvERC20Address(address verc20_, uint256 chain_) internal
```

Set the address of vERC20 on a given chain *nonce is incremented for every successful deposit or withdraw*

Parameters

Name	Type	Description
verc20_	address	vERC20 contract address
chain_	uint256	chain id of the chain where vERC2o contract resides

_setRelayWrapperAddress

```
function _setRelayWrapperAddress(address wrapper_) internal
```

Set the address of core relay wrapper

Parameters

Name	Type	Description
wrapper_	address	Core relay wrapper contract address

totalSupplyAllChains

```
function totalSupplyAllChains() public view virtual returns (uint256)
```

Return the amount of vERC20 currently available on all chains

totalSupply

```
function totalSupply(uint256 chain_) public view virtual returns (uint256)
```

Return the amount of vERC20 currently available on a given chain

Parameters

Name	Type	Description
chain_	uint256	The id of the chain of interest

asset

```
function asset() external view virtual returns (address)
```

Return the address of underlying ERC20 contract address

_record

```
function _record(enum EscrowERC20.Action action_, uint256 amount_, uint256
chain_) internal
```

Record and update state on successful deposit/withdraw

Parameters

Name	Type	Description
action_	enum EscrowERC20.Action	deposit or withdraw action
amount_	uint256	amount of tokens deposited or withdrawn
chain_	uint256	target chain id

_deposit

```
function _deposit(uint256 relay_, uint256 chain_, address receiver_,
uint256 amount_) internal returns (bool result)
```

Deposit ERC20 tokens to obtain vERC20 on target chain

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
receiver_	address	Recipient address for vERC20
amount_	uint256	Amount of ERC20 deposited

_withdraw

```
function _withdraw(uint256 relay_, uint256 chain_, address paymaster_,
address receiver_) internal returns (bool result)
```

Withdraw ERC20 tokens by depositing vERC20 on target chain

The paymaster should be a valid paymaster (e.g., forwarder). All vERC20 held by paymaster is destroyed and equal ERC20 is deposited_

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
paymaster_	address	Paymaster address to deposit vERC20
receiver_	address	Recipient address for ERC20

_callbackHandler

```
function _callbackHandler(uint256 chain_, bytes32 ack_, bool success_,
uint256 retval_) internal returns (uint256)
```

callback handler to handle acknowledgement for deposit/withdraw

Parameters

Name	Type	Description
chain_	uint256	Target chain identifier
ack_	bytes32	Unique hash of the submitted deposit/withdraw request
success_	bool	true if the deposit/withdraw was successful on remote
retval_	uint256	The amount of tokens that were deposited/withdrawn

_beforeTokenDeposit

```
function _beforeTokenDeposit(uint256 relay_, uint256 chain_, address
receiver_, uint256 amount_, address verc20_) internal
```

Hook called before token deposit

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
receiver_	address	Recipient address for vERC20
amount_	uint256	Amount of ERC20 deposited

Name	Type	Description
verc20_	address	vERC20 contract address on target chain

_afterTokenDeposit

```
function _afterTokenDeposit(uint256 relay_, uint256 chain_, address
receiver_, uint256 amount_, address verc20_) internal
```

Hook called after token deposit

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
receiver_	address	Recipient address for vERC20
amount_	uint256	Amount of ERC20 deposited
verc20_	address	vERC20 contract address on target chain

_beforeTokenWithdraw

```
function _beforeTokenWithdraw(uint256 relay_, uint256 chain_, address
paymaster_, address receiver_, address verc20_) internal
```

Hook called before token withdraw

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
paymaster_	address	Paymaster address to deposit vERC20
receiver_	address	Recipient address for ERC20
verc20_	address	vERC20 contract address on target chain

_afterTokenWithdraw

```
function _afterTokenWithdraw(uint256 relay_, uint256 chain_, address
paymaster_, address receiver_, address verc20_) internal
```

Hook called after token withdraw

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
paymaster_	address	Paymaster address to deposit vERC20
receiver_	address	Recipient address for ERC20
verc20_	address	vERC20 contract address on target chain

ZbyteEscrow

constructor

```
constructor(address forwarder_, address zbyte_, address treasury_) public
```

deposit

```
function deposit(uint256 relay_, uint256 chain_, address receiver_, uint256
amount_) public returns (bool result)
```

Deposit ERC20 tokens to obtain vERC20 on target chain

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
receiver_	address	Recipient address for vERC20
amount_	uint256	Amount of ERC20 deposited

withdraw

```
function withdraw(uint256 relay_, uint256 chain_, address paymaster_,
address receiver_) public returns (bool result)
```

Withdraw ERC20 tokens by depositing vERC20 on target chain

The paymaster should be a valid paymaster (e.g., forwarder). All vERC20 held by paymaster is destroyed and equal ERC20 is deposited_

Parameters

Name	Type	Description
relay_	uint256	Relay identifier that should be used for the crosschain call
chain_	uint256	Target chain identifier
paymaster_	address	Paymaster address to deposit vERC20
receiver_	address	Recipient address for ERC20

callbackHandler

```
function callbackHandler(uint256 chain_, bytes32 ack_, bool success_,
uint256 retval_) external returns (uint256)
```

callback handler to handle acknowledgement for deposit/withdraw

Parameters

Name	Type	Description
chain_	uint256	Target chain identifier
ack_	bytes32	Unique hash of the submitted deposit/withdraw request
success_	bool	true if the deposit/withdraw was successful on remote
retval_	uint256	The amount of tokens that were deposited/withdrawn

setvERC20Address

```
function setvERC20Address(address verc20_, uint256 chain_) public
```

Set the address of vERC20 on a given chain

nonce is incremented for every successful deposit or withdraw

Parameters

Name	Type	Description
verc20_	address	vERC20 contract address
chain_	uint256	chain id of the chain where vERC2o contract resides

setRelayWrapperAddress

```
function setRelayWrapperAddress(address wrapper_) public
```

Set the address of core relay wrapper

Parameters

Name	Type	Description
wrapper_	address	Core relay wrapper contract address

pause

```
function pause() external
```

Pauses the contract (mint, transfer and burn operations are paused)

unpause

```
function unpause() external
```

Unpauses the paused contract

_msgSender

```
function _msgSender() internal view returns (address sender)
```

ERC2771 _msgSender override

_msgData

```
function _msgData() internal view returns (bytes)
```

ERC2771 _msgData override

ZbyteForwarderCore

The Zbyte core forwarder contract.

ZeroAddress

```
error ZeroAddress()
```

error (0xd92e233d): Address is address(0)

ZbyteAddressSet

```
event ZbyteAddressSet(address)
```

event (0xa6cc9cbb): DPLAT address is set

ZbyteTokenForwarderAddressSet

```
event ZbyteTokenForwarderAddressSet(address)
```

event (0x0a787863): Token forwarder address is set

EscrowAddressSet

```
event EscrowAddressSet(address)
```

event (0x14229a64) Escrow address is set

zByteAddress

```
address zByteAddress
```

DPLAT ERC20 contract address

zbyteTokenForwarder

```
contract MinimalForwarder zbyteTokenForwarder
```

Forwarder of ERC20 token contract

escrowAddress

```
address escrowAddress
```

Escrow contract address

setZbyteAddress

```
function setZbyteAddress(address zbyte_) public
```

Set DPLAT ERC20 address

Parameters

Name	Type	Description
zbyte_	address	DPLAT ERC20 contact address

setZbyteTokenForwarderAddress

```
function setZbyteTokenForwarderAddress(address forwarder_) public
```

Set DPLAT ERC20 Forwarder address

Parameters

Name	Type	Description
forwarder_	address	DPLAT ERC20 forwarder contact address

setEscrowAddress

```
function setEscrowAddress(address escrow_) public
```

Set Zbyte Escrow address

Parameters

Name	Type	Description
escrow_	address	Zbyte Escrow contract address

approveAndDeposit

```
function approveAndDeposit(struct MinimalForwarder.ForwardRequest
reqApprove_, bytes signatureApprove_, struct
MinimalForwarder.ForwardRequest reqDeposit_, bytes signatureDeposit_)
public payable returns (bool success)
```

Perform approve and depost of Zbyte in single call

Allows gasless approve+deposit of DPLAT token to be used at https://dplat.zbyte.io

Parameters

Name	Type	Description
reqApprove_	struct MinimalForwarder.ForwardRequest	ForwardRequest for the approve call
signatureApprove_	bytes	Signature of the approve call params
reqDeposit_	struct MinimalForwarder.ForwardRequest	ForwardRequest for the deposit call
signatureDeposit_	bytes	Signature of the deposit call params

Return Values

Name	Type	Description
success	bool	returns true of approve and deposit are successful

LibDPlatBase

Library for DPlat base storage and functions

Library for DPlat base storage and functions

DiamondStorage

```
struct DiamondStorage {
  address zbyteVToken;
  uint256 zbyteValueInNativeEthGwei;
  uint256 zbyteBurnFactor;
}
```

diamondStorage

```
function diamondStorage() internal pure returns (struct
LibDPlatBase.DiamondStorage ds)
```

Retrieves the DiamondStorage struct for the library.

zbyteVToken: The address of the ZbyteVToken
zbyteValueInNativeEthGwei: The value of Zbyte in native Ether (in Gwei)
zbyteBurnFactor: Burn factor, represents the percent of gas used that will be 'burnt'

_getZbyteVToken

```
function _getZbyteVToken() internal view returns (address)
```

Gets the ZbyteVToken address.

Return Values

Name	Type	Description
[0]	address	The address of the ZbyteVToken.

_getNativeEthEquivalentZbyteValue

```
function _getNativeEthEquivalentZbyteValue(uint256 ethAmount_) internal
view returns (uint256)
```

Calculates the native Ether equivalent value of Zbyte.

ethAmountInGWei = (ethAmountinWei/10 ***9), inZbyte = ethAmountInGWeizbyteValueInNativeEthGwei, inZbyteWei = inZbyte10*** *18*

Parameters

Name	Type	Description
ethAmount_	uint256	The amount in Ether (wei).

Return Values

Name	Type	Description
[0]	uint256	The equivalent value in Zbyte (wei).

_getZbyteBurnFactor

```
function _getZbyteBurnFactor() internal view returns (uint256)
```

Gets the Zbyte burn factor.

Return Values

Name	Type	Description
[0]	uint256	The Zbyte burn factor.

LibDPlatRegistration

Library for DPlat registration storage and functions

Library for DPlat registration storage and functions

DiamondStorage

```
struct DiamondStorage {
  mapping(bytes4 => address) registeredEnterprises;
  mapping(bytes4 => address) registeredEnterprisePolicy;
  mapping(address => bytes4) registeredDapps;
  mapping(address => bytes4) registeredEnterpriseUsers;
  mapping(bytes4 => uint256) enterpriseLimit;
}
```

diamondStorage

```
function diamondStorage() internal pure returns (struct
LibDPlatRegistration.DiamondStorage ds)
```

Retrieves the DiamondStorage struct for the library.

registeredEnterprises: Mapping of registered enterprises by bytes4 ID
registeredEnterprisePolicy: Mapping of enterprise policies by bytes4 ID
registeredDapps: Mapping of registered Dapps by address
registeredEnterpriseUsers: Mapping of registered enterprise users by address
enterpriseLimit: Mapping of enterprise limits by bytes4 ID

_getEnterpriseLimit

```
function _getEnterpriseLimit(bytes4 enterprise_) internal view returns
(uint256)
```

Gets the enterprise limit for a given enterprise ID.

Parameters

Name	Type	Description
enterprise_	bytes4	The enterprise ID.

Return Values

Name	Type	Description
[0]	uint256	The enterprise limit.

_setEntepriseLimit

```
function _setEntepriseLimit(bytes4 enterprise_, uint256 amount_) internal
```

Sets the enterprise limit for a given enterprise ID.

Parameters

Name	Type	Description
enterprise_	bytes4	The enterprise ID.
amount_	uint256	The limit amount to set.

_doesEnterpriseHavePolicy

```
function _doesEnterpriseHavePolicy(bytes4 enterprise_) internal view
returns (bool, address)
```

Checks if an enterprise has a registered policy and retrieves the policy address.

Parameters

Name	Type	Description
enterprise_	bytes4	The enterprise ID.

Return Values

Name	Type	Description
[0]	bool	A tuple indicating whether the enterprise policy exists and the policy address.
[1]	address	

isProviderRegistered

```
function isProviderRegistered(address provider_) internal view returns
(bool)
```

Checks if the given provider is registered

Parameters

Name	Type	Description
provider_	address	The provider address

Return Values

Name	Type	Description
[0]	bool	bool indicating if the provider is registered

isProviderAgentRegistered

```
function isProviderAgentRegistered(address agent_) internal view returns
(address)
```

Checks if the given agent is registered

Parameters

Name	Type	Description
agent_	address	The agent address

Return Values

Name	Type	Description
[0]	address	returns the address of provider if registered, or address(0)

isEnterpriseRegistered

```
function isEnterpriseRegistered(bytes4 enterprise_) internal view returns
(address)
```

Checks if the given enterprise is registered

Parameters

Name	Type	Description
enterprise_	bytes4	The enterprise bytes4 ID

Return Values

Name	Type	Description
[0]	address	returns the address of provider if registered, or address(0)

isEnterpriseUserRegistered

```
function isEnterpriseUserRegistered(address user_) internal view returns
(bytes4)
```

Checks if the given user is registered with an enterprise

Parameters

Name	Type	Description
user_	address	The user address

Return Values

Name	Type	Description
[0]	bytes4	returns the address of provider if registered, or address(0)

isEnterpriseDappRegistered

```
function isEnterpriseDappRegistered(address dapp_) internal view returns
(bytes4)
```

Checks if the given dapp (contract) is registered with an enterprise

Parameters

Name	Type	Description
dapp_	address	The contract address

Return Values

Name	Type	Description

Name	Type	Description
[0]	bytes4	returns the address of provider if registered, or address(0)

LibDPlatProvider

Library for DPlat provider storage and functions

Library for DPlat provider storage and functions

DiamondStorage

```
struct DiamondStorage {
  mapping(address => bool) registeredProviders;
  mapping(address => address) registeredProviderAgent;
}
```

diamondStorage

```
function diamondStorage() internal pure returns (struct
LibDPlatProvider.DiamondStorage ds)
```

Retrieves the DiamondStorage struct for the library.

registeredProviders: Mapping of registered providers by address
registeredProviderAgent: Mapping of registered provider agents by address

ZbyteDPlatBaseFacet

DPlat Base Facet contract

ZbyteVTokenAddressSet

```
event ZbyteVTokenAddressSet(address)
```

event (0x10e1dc22): VZbyte token address is set.

ZbyteValueInNativeEthGweiSet

```
event ZbyteValueInNativeEthGweiSet(uint256)
```

event (0xa0e61546): Zbyte token value in terms of native eth is set.

ZbyteBurnFactorSet

```
event ZbyteBurnFactorSet(uint256)
```

event (0xd7a7cf8c): Zbyte burn factor is set.

setZbyteVToken

```
function setZbyteVToken(address zbyteVToken_) public
```

Sets the address of the ZbyteVToken.

Parameters

Name	Type	Description
zbyteVToken_	address	The address of the ZbyteVToken.

setZbyteBurnFactor

```
function setZbyteBurnFactor(uint256 zbyteBurnFactor_) public
```

Sets the Zbyte burn factor.

Parameters

Name	Type	Description
zbyteBurnFactor_	uint256	Zbyte burn factor

getZbyteVToken

```
function getZbyteVToken() public view returns (address)
```

Gets the address of the ZbyteVToken.

Return Values

Name	Type	Description
[0]	address	The address of the ZbyteVToken.

setZbyteValueInNativeEthGwei

```
function setZbyteValueInNativeEthGwei(uint256 zbyteValueInNativeEthGwei_)
public
```

Sets the value of Zbyte in native Ether (in Gwei).

Parameters

Name	Type	Description
zbyteValueInNativeEthGwei_	uint256	The value of Zbyte in native Ether (in Gwei).

getZbyteValueInNativeEthGwei

```
function getZbyteValueInNativeEthGwei() public view returns (uint256)
```

Gets the value of Zbyte in native Ether (in Gwei).

Return Values

Name	Type	Description
[0]	uint256	The value of Zbyte in native Ether (in Gwei).

getZbyteBurnFactor

```
function getZbyteBurnFactor() public view returns (uint256)
```

Gets the Zbyte burn factor.

Return Values

Name	Type	Description
[0]	uint256	The Zbyte burn factor (0-100).

ZbyteDPlatPaymentFacet

RefundEthToPayer

```
event RefundEthToPayer(address, uint256)
```

events Error(0x187e1a0c) Amount to be refund in terms of Eth to Payer.

getPayer

```
function getPayer(address user_, address dapp_, bytes4 functionSig_,
uint256 amount_) public returns (address)
```

Determines the payer for a transaction.

Parameters

Name	Type	Description
user_	address	The user's address.
dapp_	address	The Dapp's address.
functionSig_	bytes4	The function signature (bytes4).
amount_	uint256	The transaction amount.

Return Values

Name	Type	Description
[0]	address	The payer's address.

preExecute

```
function preExecute(address dapp_, address user_, bytes4 functionSig_,
uint256 ethChargeAmount_) public returns (address _payer)
```

Pre Execution (Finds the payer and charges in ZbyteVToken)

Parameters

Name	Type	Description
dapp_	address	The Dapp's address.
user_	address	The user's address.
functionSig_	bytes4	The function signature (bytes4).
ethChargeAmount_	uint256	The Ether amount to charge.

postExecute

```
function postExecute(address payer_, bool executeResult_, uint256
reqValue_, uint256 gasConsumedEth_, uint256 preChargeEth_) public
```

Executes a transaction and handles Zbyte-related operations.

Parameters

Name	Type	Description
payer_	address	The address of the payer initiating the execution.
executeResult_	bool	A boolean indicating the success of the execution.
reqValue_	uint256	The amount of Ether sent with the execution request.
gasConsumedEth_	uint256	The amount of Ether consumed for gas during execution.
preChargeEth_	uint256	The amount of Ether charged before execution. This function can only be called by the `onlyForwarder` modifier.

ZbyteDPlatRegistrationFacet

Zbyte DPlat Registration Facet

Zbyte DPlat Registration Facet

ZbyteDPlatProviderRegistred

```
event ZbyteDPlatProviderRegistred(address, bool)
```

events event (0x2a3043c9): Zbyte DPlat provider is registered.

ZbyteDPlatProviderAgentRegistered

```
event ZbyteDPlatProviderAgentRegistered(address, address)
```

event (0xb0c62993): Zbyte DPlat provider agent is registered.

ZbyteDPlatEnterpriseRegistered

```
event ZbyteDPlatEnterpriseRegistered(bytes4, address)
```

event (0xa98ff618): Zbyte DPlat enterprise is registered.

ZbyteDPlatEnterpriseUserRegistered

```
event ZbyteDPlatEnterpriseUserRegistered(address, bytes4)
```

event (0x83439d26): Zbyte DPlat enterprise user is registered.

ZbyteDPlatDappRegistered

```
event ZbyteDPlatDappRegistered(address, bytes4)
```

event (0x822d049d): Zbyte DPlat dapp is registered.

ZbyteDPlatEnterpriseLimitSet

```
event ZbyteDPlatEnterpriseLimitSet(bytes4, uint256)
```

event (0xf9d1da16): Zbyte DPlat enterprise limit is set.

ProviderAlreadyRegistered

```
error ProviderAlreadyRegistered(address)
```

errors error (0x74f7822a): Provider already registered.

ProviderNotRegistered

```
error ProviderNotRegistered(address)
```

error (0x232cb27a): Provider not registered.

InvalidEnterprise

```
error InvalidEnterprise(bytes4)
```

error (0x128c088b): Invalid enterprise hash.

ProviderAgentAlreadyRegistered

```
error ProviderAgentAlreadyRegistered(address)
```

error (0xe751ad65): Provider Agent is already registered.

ProviderAgentNotRegistered

```
error ProviderAgentNotRegistered(address)
```

error (0xd0141a6a): Not a registered provider agent.

InvalidProvider

```
error InvalidProvider(address)
```

error (0x96271599): Invalid provider.

EnterpriseAlreadyRegistered

```
error EnterpriseAlreadyRegistered(bytes4)
```

error (0x6d998cea): Enterprise is already registered.

EnterpriseNotRegistered

```
error EnterpriseNotRegistered(bytes4)
```

error (0xbd825961): Enterprise is not registered.

NotARegisteredProvider

```
error NotARegisteredProvider(address)
```

error (0xca61871b): Not a registered provider.

EnterpriseUserAlreadyRegistered

```
error EnterpriseUserAlreadyRegistered(address)
```

error (0x43469070): Enterprise user is already registered.

EnterpriseUserNotRegistered

```
error EnterpriseUserNotRegistered(address)
```

error (0x1b7bfcf8): Enterprise user is not registered.

EnterpriseDappAlreadyRegistered

```
error EnterpriseDappAlreadyRegistered(address)
```

error (0xbcb8afa4): Enterprise dapp is already registered.

EnterpriseDappNotRegistered

```
error EnterpriseDappNotRegistered(address)
```

error (0x31b254a2): Enterprise dapp is not registered.

_setRegisteredProvider

```
function _setRegisteredProvider(address provider_, bool set_) internal
```

Internal function to set the registration status of a provider.

This function is used internally to manage the registration status of providers.

Parameters

Name	Type	Description
provider_	address	The address of the provider whose registration status will be set.
set_	bool	A boolean indicating whether to set the provider as registered or not.

_setRegisteredProviderAgent

```
function _setRegisteredProviderAgent(address agent_, address provider_)
internal
```

Internal function to set the registration of a provider agent.

This function is used internally to manage the registration of provider agents.

Parameters

Name	Type	Description
agent_	address	The address of the agent whose provider registration will be set.
provider_	address	The address of the provider associated with the agent.

_setRegisteredEnterprise

```
function _setRegisteredEnterprise(bytes4 enterprise_, address provider_)
internal
```

Internal function to set the registration status of an enterprise.

This function is used internally to manage the registration status of enterprises.

Parameters

Name	Type	Description
enterprise_	bytes4	The identifier of the enterprise whose registration status will be set.
provider_	address	The address of the provider associated with the enterprise.

_setRegisteredEnterpriseUser

```
function _setRegisteredEnterpriseUser(address user_, bytes4 enterprise_)
internal
```

Internal function to set the registration status of an enterprise user.

This function is used internally to manage the registration status of enterprise users.

Parameters

Name	Type	Description
user_	address	The address of the user whose enterprise registration will be set.
enterprise_	bytes4	The identifier of the enterprise associated with the user.

_setRegisteredEnterpriseDapp

```
function _setRegisteredEnterpriseDapp(address dapp_, bytes4 enterprise_)
internal
```

Internal function to set the registration status of an enterprise Dapp.

This function is used internally to manage the registration status of enterprise Dapps.

Parameters

Name	Type	Description
dapp_	address	The address of the Dapp whose enterprise registration will be set.
enterprise_	bytes4	The identifier of the enterprise associated with the Dapp.

isProviderRegistered

```
function isProviderRegistered(address provider_) public view returns (bool)
```

Checks if a provider is registered.

Parameters

Name	Type	Description
provider_	address	The address of the provider to check.

Return Values

Name	Type	Description
[0]	bool	A boolean indicating whether the provider is registered.

isProviderAgentRegistered

```
function isProviderAgentRegistered(address agent_) public view returns
(address)
```

Checks if a provider agent is registered and returns the associated provider's address.

Parameters

Name	Type	Description
agent_	address	The address of the provider agent to check.

Return Values

Name	Type	Description
[0]	address	The address of the associated registered provider.

isEnterpriseRegistered

```
function isEnterpriseRegistered(bytes4 enterprise_) public view returns
(address)
```

Checks if an enterprise is registered and returns the associated provider's address.

Parameters

Name	Type	Description
enterprise_	bytes4	The identifier of the enterprise to check.

Return Values

Name	Type	Description
[0]	address	The address of the associated registered provider.

isEnterpriseUserRegistered

```
function isEnterpriseUserRegistered(address user_) public view returns
(bytes4)
```

Checks if an enterprise user is registered and returns the associated enterprise identifier.

Parameters

Name	Type	Description
user_	address	The address of the user to check.

Return Values

Name	Type	Description
[0]	bytes4	The identifier of the associated registered enterprise.

isEnterpriseDappRegistered

```
function isEnterpriseDappRegistered(address dapp_) public view returns
(bytes4)
```

Checks if an enterprise Dapp is registered and returns the associated enterprise identifier.

Parameters

Name	Type	Description
dapp_	address	The address of the Dapp to check.

Return Values

Name	Type	Description
[0]	bytes4	The identifier of the associated registered enterprise.

registerProvider

```
function registerProvider(address provider_) public
```

Registers a provider.

_Relation between provider, agent, enterprise, users and dapps is as follows:

zbyte
____(1) <--------> (n) provider
____________________________(1) <--------> (n) agent
____________________________(1) <--------> (n) enterprise
____(1) <--------> (n) user
____(1) <--------> (n) dapp

For an enterprise usecase, an enterprise can allow users to invoke registered dapps.
Users can invoke the contract functions without any need to hold crypto assets.
L1 needed for the call is given by the authorized workers and providers compensate them in vERC20.

For opensource usecase,
Users can invoke the contract functions without any need to hold L1 assets.
L1 needed for the call is given by the authorized workers and the users compensate them in vERC20

NOTE: When one of the components (provider, enterprise, agent, user or dapp) is deregistered,
all the other components registered under it remain registered.
So, if the component is registered again, the entire subtree becomes active again_

Parameters

Name	Type	Description
provider_	address	The address of the provider to register.

deregisterProvider

```
function deregisterProvider(address provider_) public
```

Deregisters a provider.

Parameters

Name	Type	Description
provider_	address	The address of the provider to deregister.

registerProviderAgent

```
function registerProviderAgent(address agent_) public
```

Registers a provider agent.

Parameters

Name	Type	Description
agent_	address	The address of the provider agent to register.

deRegisterProviderAgent

```
function deRegisterProviderAgent(address agent_) public
```

Deregisters a provider agent.

Parameters

Name	Type	Description
agent_	address	The address of the provider agent to deregister.

registerEnterprise

```
function registerEnterprise(bytes4 enterprise_) public
```

Registers an enterprise.

Parameters

Name	Type	Description
enterprise_	bytes4	The bytes4 identifier of the enterprise to register.

deregisterEnterprise

```
function deregisterEnterprise(bytes4 enterprise_) public
```

Deregisters an enterprise.

Parameters

Name	Type	Description
enterprise_	bytes4	The bytes4 identifier of the enterprise to deregister.

registerEnterpriseUser

```
function registerEnterpriseUser(address user_, bytes4 enterprise_) public
```

Registers an enterprise user.

Parameters

Name	Type	Description
user_	address	The address of the user to register.
enterprise_	bytes4	The bytes4 identifier of the enterprise.

deregisterEnterpriseUser

```
function deregisterEnterpriseUser(address user_) public
```

Deregisters an enterprise user.

Parameters

Name	Type	Description
user_	address	The address of the user to deregister.

registerDapp

```
function registerDapp(address dapp_, bytes4 enterprise_) public
```

Registers a Dapp for an enterprise.

Parameters

Name	Type	Description
dapp_	address	The address of the Dapp to register.
enterprise_	bytes4	The bytes4 identifier of the enterprise.

deregisterDapp

```
function deregisterDapp(address dapp_) public
```

Deregisters a Dapp for an enterprise.

Parameters

Name	Type	Description
dapp_	address	The address of the Dapp to deregister.

setEnterpriseLimit

```
function setEnterpriseLimit(bytes4 enterprise_, uint256 amount_) public
```

Sets the enterprise limit for a specific enterprise.

Parameters

Name	Type	Description
enterprise_	bytes4	The bytes4 identifier of the enterprise.
amount_	uint256	The new limit amount.

ZbyteForwarderDPlat

ForwarderDplatSet

```
event ForwarderDplatSet(address)
```

events event (0xeae099e1): Forwarder address is set.

ForwarderDplatMinimumProcessingGasSet

```
event ForwarderDplatMinimumProcessingGasSet(uint256)
```

event (0x6342abcf): Forwarder minimum processing gas is set.

ForwarderDplatWorkerRegistered

```
event ForwarderDplatWorkerRegistered(address, bool)
```

event (0xe1554bda): Forwarder worker is registered.

RefundEth

```
event RefundEth(address, uint256)
```

event (0xe5cac075): Refund Eth to payer.

ZbyteForwarderDPlatExecute

```
event ZbyteForwarderDPlatExecute(bool, bytes)
```

event (0x5c3206c6): Execute result and return data

ZeroAddress

```
error ZeroAddress()
```

errors error (0xd92e233d): Address is zero.

ArraySizeMismatch

```
error ArraySizeMismatch(uint256, uint256)
```

error (0xfb3dd446): Array sizes don't match.

NotEnoughEtherSent

```
error NotEnoughEtherSent(uint256, uint256)
```

error (0xf9309a09): Not enough ether sent the function.

FailedToSendEther

```
error FailedToSendEther(address, uint256, bytes)
```

error (0xb7da4a55): Failed to send ether.

NotAWorker

```
error NotAWorker(address)
```

error (0x9059e055): Not a worker.

minProcessingGas

```
uint256 minProcessingGas
```

Minimum amount of gas needed for a call via the forwarder

zbyteDPlat

```
address zbyteDPlat
```

Address of the Zbyte DPlat contract

registeredWorkers

```
mapping(address => bool) registeredWorkers
```

Mapping of registered workers

onlyWorker

```
modifier onlyWorker()
```

Modifier to restrict a function to only be callable by registered workers.

*The function using this modifier will only execute if the sender's address is a registered worker
It will revert with a 'NotAWorker' error if the sender is not a registered worker.*

setMinProcessingGas

```
function setMinProcessingGas(uint256 minProcessingGas_) public
```

Sets the minimum processing gas

Parameters

Name	Type	Description
minProcessingGas_	uint256	The new minimum processing gas value

setZbyteDPlat

```
function setZbyteDPlat(address zbyteDPlat_) public
```

Sets the address of the Zbyte DPlat contract

Parameters

Name	Type	Description
zbyteDPlat_	address	The address of the Zbyte DPlat contract

registerWorkers

```
function registerWorkers(address[] workers_, bool[] register_) public
```

Registers workers with the contract

Parameters

Name	Type	Description
workers_	address[]	An array of worker addresses
register_	bool[]	An array of boolean values indicating registration status

zbyteExecute

```
function zbyteExecute(struct MinimalForwarder.ForwardRequest req_, bytes
signature_) public payable returns (bool, bytes)
```

Executes a forward request, ensuring that it is called by a registered worker and handling gas fees.

_This function facilitates call to a target contract while allowing the user to pay in DPLAT tokens
The user would have received vERC20 necessary for the call execution. An equivalent amount is charged in vERC20 from the user
If the target contract accepts msg.value, equivalent of that is charged from the user during preExecute
If preExecute collects more vERC20 than that is needed for the call, an event is emitted with the refund amount
If the target contract sends any refund to the *msgSender(), the caller receives the refund directly If the target contract call reverts, msg.value is not sent to the target and an event is emitted with the refund amount*

Parameters

Name	Type	Description
req_	struct MinimalForwarder.ForwardRequest	The forward request data containing the recipient, value, data, and other information.
signature_	bytes	The signature for the forward request (if required).

Return Values

Name	Type	Description
[0]	bool	success A boolean indicating whether the execution was successful.
[1]	bytes	returndata The return data from the executed contract.

withdrawEth

```
function withdrawEth(address receiver_) public
```

Allows the owner of the contract to withdraw the contract's Ether balance.

Parameters

Name	Type	Description
receiver_	address	The address to which the Ether balance will be sent.

ZbyteVToken

The ZBYT vERC20 contract

ZeroAddress

```
error ZeroAddress()
```

error (0xd92e233d): Address is address(0)

CannotSendEther

```
error CannotSendEther()
```

error (0xbf064619): Contract cannot receive ether

InvalidDestroyAddress

```
error InvalidDestroyAddress(address, address, address)
```

error (b034fa06): The address sent for destroy is not valid

PaymasterAddressSet

```
event PaymasterAddressSet(address)
```

event (0xa16990bf) Paymaster address is set

ZbyteDPlatAddressSet

```
event ZbyteDPlatAddressSet(address)
```

event (0xcdb1d336) ZbyteDPlat address is set

constructor

```
constructor(address burner_) public
```

ZBYT ERC20 constructor

Parameters

Name	Type	Description

Name	Type	Description
burner_	address	Burn account address (Tokens are locked here, not destroyed)

pause

```
function pause() external
```

Pauses the contract (mint, transfer and burn operations are paused)

unpause

```
function unpause() external
```

Unpauses the paused contract

setPaymasterAddress

```
function setPaymasterAddress(address paymaster_) public
```

Set the paymaster (forwarder) address

Parameters

Name	Type	Description
paymaster_	address	Paymaster contract address

setZbyteDPlatAddress

```
function setZbyteDPlatAddress(address dplat_) public
```

Set the DPlat address

Parameters

Name	Type	Description
dplat_	address	DPlat contract address

transfer

```
function transfer(address to_, uint256 value_) public returns (bool)
```

Transfer vERC20 from caller's account to receiver's account

requiresAuth ensures that this call can be complely disabled, or only specific accounts can call

Parameters

Name	Type	Description
to_	address	Receiver account address
value_	uint256	Amount of tokens to be transferred

transferFrom

```
function transferFrom(address from_, address to_, uint256 value_) public
returns (bool)
```

Transfers tokens from a specified address to another address.

requiresAuth ensures that this call can be complely disabled, or only specific accounts can call Allowing only specific accounts to perform transferFrom allows controlled transfer of vERC20 in future

Parameters

Name	Type	Description
from_	address	The address to transfer tokens from
to_	address	The address to transfer tokens to
value_	uint256	The amount of tokens to transfer

mint

```
function mint(address to_, uint256 amount_) external returns (uint256)
```

mint vZBYT ERC20

The forwarder charges user in this ERC20 token for the contract call. Approve the tokens to dplat at mint itself.

Parameters

Name	Type	Description

Name	Type	Description
to_	address	Receiver address
amount_	uint256	Amount to mint to the address(to_) and approve to dplat

burn

```
function burn(address from_, uint256 amount_) external returns (uint256)
```

Transfer vERC20 to 'burner' address

requiresAuth ensures that this call can be complely disabled, or only specific accounts can call

Parameters

Name	Type	Description
from_	address	Sender address to burn tokens from
amount_	uint256	Amount to burn

destroy

```
function destroy(address from_) external returns (uint256)
```

Destroy vERC20

This is called during withdraw / reconciliation only. Withdraw is allowed only from the paymaster or burner address

Parameters

Name	Type	Description
from_	address	Paymaster/burner address from which tokens are destroyed

receive

```
receive() external payable
```

receive function (reverts)

IEscrowERC20

InvalidRelay

```
error InvalidRelay(address)
```

Caller is not a valid relay

InvalidCallbackMessage

```
error InvalidCallbackMessage(uint256, uint256, uint256, uint256)
```

event (0xd6facdff): The callback received was invalid

InvalidCallbackAck

```
error InvalidCallbackAck(uint256, bytes32, bool, uint256)
```

event (0xcd9d7bb0): The ack in callback received was not found

vERC20AddressSet

```
event vERC20AddressSet(address, uint256)
```

event (0x1a40ce6d): vERC20 contract address is set

RelayWrapperAddressSet

```
event RelayWrapperAddressSet(address)
```

event (0x95290bcc): Core relay wrapper contract address is set

ERC20Deposited

```
event ERC20Deposited(address, address, uint256, uint256, bytes32)
```

event (0xcae09af7): ERC20 tokens deposited

ERC20DepositFailed

```
event ERC20DepositFailed(address, address, uint256, uint256, bytes32)
```

event (0x0583eefc): ERC20 tokens deposit failed

ERC20DepositConfirmed

```
event ERC20DepositConfirmed(bytes32, bool, uint256)
```

event (0xf64578a8): ERC20 tokens deposit confirmed

ERC20Withdrawn

```
event ERC20Withdrawn(address, address, address, uint256, bytes32)
```

event (0x8b923c21): ERC20 tokens withdrawn

ERC20WithdrawFailed

```
event ERC20WithdrawFailed(address, address, address, uint256, bytes32)
```

event (0x2b4d7cea): ERC20 tokens withdraw failed

ERC20WithdrawConfirmed

```
event ERC20WithdrawConfirmed(bytes32, bool, uint256)
```

event (0xf5a60bd1): ERC20 tokens withdraw confirmed

TreasuryAddressSet

```
event TreasuryAddressSet(address, address)
```

event (0x1db696c9): The Treasury address is set

getNonce

```
function getNonce() external view returns (uint256)
```

totalSupplyAllChains

```
function totalSupplyAllChains() external view returns (uint256)
```

totalSupply

```
function totalSupply(uint256 chain_) external view returns (uint256)
```

asset

```
function asset() external view returns (address)
```

callbackHandler

```
function callbackHandler(uint256 chain_, bytes32 ack_, bool success_,
uint256 retval_) external returns (uint256)
```

IEnterprisePaymentPolicy

isUserOrDappEligibleForPayment

```
function isUserOrDappEligibleForPayment(address user_, address dapp_,
bytes4 functionSig_, uint256 amount_) external returns (bool)
```

IZbyteDPlat

preExecute

```
function preExecute(address user_, address dapp_, bytes4 functionSig_,
uint256 chargeEth_) external returns (address)
```

postExecute

```
function postExecute(address payer_, bool executeResult_, uint256
reqValue_, uint256 gasConsumedEth_, uint256 preChargeEth_) external
```

IvERC20

Interface for a contract representing a variation of the ERC20 token.

burn

```
function burn(address to, uint256 amount) external returns (uint256)
```

Burns a specified amount of tokens by transferring them to the specified address.

Parameters

Name	Type	Description
to	address	The address to which the tokens will be burned.
amount	uint256	The amount of tokens to be burned.

mint

```
function mint(address to, uint256 amount) external returns (uint256)
```

Mints a specified amount of tokens and transfers them to the specified address.

Parameters

Name	Type	Description
to	address	The address to which the tokens will be minted and transferred.
amount	uint256	The amount of tokens to be minted.

IRelayWrapper

performCrossChainCall

```
function performCrossChainCall(uint256 relay_, uint256 srcChain_, uint256
destChain_, address destContract_, bytes destCallData_, bytes32 ack_,
address callbackContract_, bytes relayParams_) external payable returns
(bool)
```

isValidRelay

```
function isValidRelay(uint256 chainId, address relay_) external returns
(bool)
```

updatePayload

```
function updatePayload(uint256 destChain_, address destContract_, bytes32
ack_, address callbackContract_, bytes data_) external pure returns (bytes)
```

RelayWrapper

The Relay wrapper to facilitate ZBYT deposit/mint

InvalidCallBackContract

```
error InvalidCallBackContract()
```

error (0xeed987a0): The callback contract address is 0 but ack is set

RelayContractNotSet

```
error RelayContractNotSet(uint256, address, address)
```

error (0x089c2a3e): The relay contract address is not set for the given relay id

CallerNotEscrow

```
error CallerNotEscrow(address, address)
```

error (0x5c87504d): Caller is not the registered escrow

EscrowAddressSet

```
event EscrowAddressSet(address)
```

error (0x14229a64): Address of escrow contract is set

RelayAddressSet

```
event RelayAddressSet(uint256, uint256, address)
```

error (0xbe32fe92): Address of Relay is set for given chain id and relay id

relayContract

```
mapping(uint256 => mapping(uint256 => address)) relayContract
```

mapping of chain id => relay id => relay address

relay id is an identifier for relay (e.g., 0 -> zbyte relay, 1 -> axelar, etc)

chainRelays

```
mapping(uint256 => uint256[]) chainRelays
```

mapping of chain id => array of valid relay ids

escrow

```
address escrow
```

Registered escrow contract address

constructor

```
constructor(address forwarder_) public
```

Relay Wrapper constructor

Parameters

Name	Type	Description
forwarder_	address	Forwarder contact address

onlyEscrow

```
modifier onlyEscrow()
```

Modifier to check if the caller is the registered escrow

setEscrowAddress

```
function setEscrowAddress(address escrow_) public
```

Set the address of Escrow contract

Parameters

Name	Type	Description
escrow_	address	Escrow contract address

setRelayAddress

```
function setRelayAddress(uint256 chain_, uint256 relayid_, address relay_)
public
```

Set the address of Relay contract

set the relay address to 0 to disable the relay

Parameters

Name	Type	Description
chain_	uint256	Chain id for which the relay address is set
relayid_	uint256	Relay id for which relay address is set
relay_	address	Relay contract Address

isValidRelay

```
function isValidRelay(uint256 chain_, address relay_) external view returns
(bool)
```

Verify if given relay is a valid one for the given chain id

Parameters

Name	Type	Description
chain_	uint256	Chain id for which the relay address is set
relay_	address	Relay contract Address

performCrossChainCall

```
function performCrossChainCall(uint256 relayid_, uint256 srcChain_, uint256
destChain_, address destContract_, bytes destCallData_, bytes32 ack_,
address callbackContract_, bytes relayParams_) external payable returns
(bool)
```

Initiate the cross chain call for deposit/mint

This function can be called only the the registered escrow contract

Parameters

Name	Type	Description
relayid_	uint256	Relay id that should be used for this call
srcChain_	uint256	Chain id of source chain
destChain_	uint256	Chain id of destination chain
destContract_	address	Address of contract to be called on destination chain
destCallData_	bytes	Calldata for the call on destination chain
ack_	bytes32	Unique hash of the cross chain deposit/mint call
callbackContract_	address	Address of contract on source chain to handle callback
relayParams_	bytes	Additional data that can be sent to the relay

updatePayload

```
function updatePayload(uint256 destChain_, address destContract_, bytes32
ack_, address callbackContract_, bytes data_) public pure returns (bytes)
```

Update the payload to include additional information

Parameters

Name	Type	Description
destChain_	uint256	Chain id of destination chain
destContract_	address	Address of contract to be called on destination chain
ack_	bytes32	Unique hash of the cross chain deposit/mint call
callbackContract_	address	Address of contract on source chain to handle callback
data_	bytes	original payload

_msgSender

```
function _msgSender() internal view returns (address sender)
```

ERC2771 _msgSender override

_msgData

```
function _msgData() internal view returns (bytes)
```

ERC2771 _msgData override

ZbyteRelay

The Zbyte Relay contract

NotApproved

```
error NotApproved(address)
```

error (0x0ca968d8): Caller is not an approved caller

NotRelayWrapperOrSelf

```
error NotRelayWrapperOrSelf(address, address)
```

error (0x26fb3778): Caller is not the RelayWrapper or this contract

InvalidChain

```
error InvalidChain(uint256, uint256)
```

error (0xc16b00ce): Current chain id does not match with the one sent in payload

RelayCallRemoteReceived

```
event RelayCallRemoteReceived(uint256, address, uint256, address, bytes)
```

event (0x9a3d7ba1): Received the request to perform a remote call

RelayReceiveCallExecuted

```
event RelayReceiveCallExecuted(bytes, bool, uint256)
```

event (0xceeaa702): Executed the call request from a source chain

RelayWrapperSet

```
event RelayWrapperSet(address)
```

event (0x2658b600): Relay Wrapper address is set

RelayApproveeAdded

```
event RelayApproveeAdded(address)
```

event (0xe89d9bcd): Approvee address is set

approved

```
mapping(address => bool) approved
```

mapping of approved addresses. Only these addresses can invoke the 'receiveCall'

relayWrapper

```
contract IRelayWrapper relayWrapper
```

Address of the RelayWrapper (on core)

constructor

```
constructor(address forwarder_) public
```

Zbyte Relay constructor

Parameters

Name	Type	Description
forwarder_	address	Forwarder contact address

onlyApprovedOrSelf

```
modifier onlyApprovedOrSelf()
```

Modifier to check if the caller is approved or this contract

onlyRelayWrapperOrSelf

```
modifier onlyRelayWrapperOrSelf()
```

Modifier to check if the caller is RelayWrapper or this contract

setRelayWrapper

```
function setRelayWrapper(address wrapper_) external
```

Set the RelayWrapper contract address

Parameters

Name	Type	Description
wrapper_	address	RelayWrapper contact address

addRelayApprovee

```
function addRelayApprovee(address approvee_) external
```

Set the approvee address

Parameters

Name	Type	Description
approvee_	address	Address of the approvee

callRemote

```
function callRemote(uint256 destChain_, address destRelay_, bytes payload_)
public payable returns (bool)
```

Initiate the remote chain call

Parameters

Name	Type	Description
destChain_	uint256	Chain id of destination chain
destRelay_	address	Address of the trusted relay on destination chain
payload_	bytes	Payload to be used for the destination call

receiveCall

```
function receiveCall(uint256 srcChain_, address srcRelay_, bytes payload_)
external returns (bool)
```

Handle the call received from source chain

Call can be made only by approved accounts or self

Parameters

Name	Type	Description
srcChain_	uint256	Chain id of source chain
srcRelay_	address	Address of the trusted relay on source chain
payload_	bytes	Payload to be used for the call on this chain

updatePayload

```
function updatePayload(uint256 destChain_, address destContract_, bytes32
ack_, address callbackContract_, bytes data_) public pure returns (bytes)
```

Update the payload to include additional information

Parameters

Name	Type	Description
destChain_	uint256	Chain id of destination chain
destContract_	address	Address of contract to be called on destination chain
ack_	bytes32	Unique hash of the cross chain deposit/mint call
callbackContract_	address	Address of contract on source chain to handle callback
data_	bytes	original payload

_msgSender

```
function _msgSender() internal view returns (address sender)
```

ERC2771 _msgSender override

_msgData

```
function _msgData() internal view returns (bytes)
```

ERC2771 _msgData override

SampleDstoreDapp

DStoreSet

```
event DStoreSet(address, uint256)
```

storedValue

```
uint8 storedValue
```

storedBy

```
address storedBy
```

constructor

```
constructor(address forwarder_) public
```

storeValue

```
function storeValue(uint8 _value) public
```

Auth

UserRoleUpdated

```
event UserRoleUpdated(address user, uint8 role, bool enabled)
```

PublicCapabilityUpdated

```
event PublicCapabilityUpdated(bytes4 functionSig, bool enabled)
```

RoleCapabilityUpdated

```
event RoleCapabilityUpdated(uint8 role, bytes4 functionSig, bool enabled)
```

DiamondStorage

```
struct DiamondStorage {
  mapping(address => bytes32) getUserRoles;
  mapping(bytes4 => bool) isCapabilityPublic;
  mapping(bytes4 => bytes32) getRolesWithCapability;
}
```

diamondStorage

```
function diamondStorage() internal pure returns (struct Auth.DiamondStorage
ds)
```

getOwner

```
function getOwner() public virtual returns (address)
```

doesUserHaveRole

```
function doesUserHaveRole(address user, uint8 role) public view returns
(bool)
```

doesRoleHaveCapability

```
function doesRoleHaveCapability(uint8 role, bytes4 functionSig) public view
returns (bool)
```

canCall

```
function canCall(address user, bytes4 functionSig) public view returns
(bool)
```

isAuthorized

```
function isAuthorized(address user, bytes4 functionSig) internal view
returns (bool)
```

isAuthorizedOrOwner

```
function isAuthorizedOrOwner(address user, bytes4 functionSig) internal
returns (bool)
```

requiresAuth

```
modifier requiresAuth()
```

requiresAuthOrOwner

```
modifier requiresAuthOrOwner()
```

setPublicCapability

```
function setPublicCapability(bytes4 functionSig, bool enabled) public
```

setRoleCapability

```
function setRoleCapability(uint8 role, bytes4 functionSig, bool enabled)
public
```

setUserRole

```
function setUserRole(address user, uint8 role, bool enabled) public
```

AuthDiamond

getOwner

```
function getOwner() public virtual returns (address)
```

AuthSimple

getOwner

```
function getOwner() public virtual returns (address)
```

LibCommonErrors

ZeroAddress

```
error ZeroAddress()
```

NotOwner

```
error NotOwner()
```

Unauthorized

```
error Unauthorized()
```

ArraySizeMismatched

```
error ArraySizeMismatched(uint256, uint256)
```

LibZbyteForwarderFacet

The Zbyte Forwarder Facet

The Zbyte Forwarder Facet

DiamondStorage

```
struct DiamondStorage {
  address trustedForwarder;
}
```

diamondStorage

```
function diamondStorage() internal pure returns (struct
LibZbyteForwarderFacet.DiamondStorage ds)
```

Retrieves the DiamondStorage struct for the library.

trustedForwarder: Address of the trusted forwarder

_setTrustedForwarder

```
function _setTrustedForwarder(address forwarder_) internal
```

Sets the address of trusted forwarder

Parameters

Name	Type	Description
forwarder_	address	

_getTrustedForwarder

```
function _getTrustedForwarder() internal view returns (address)
```

Gets the address of trusted forwarder

isTrustedForwarder

```
function isTrustedForwarder(address forwarder_) internal view returns
(bool)
```

Checks if the given forwarder is the trusted forwarder

Parameters

Name	Type	Description
forwarder_	address	

ZbyteContext

ERC2771Context with a function to set forwarder

CannotSendEther

```
error CannotSendEther()
```

error (0xbf064619): Contract cannot receive ether

ZeroAddress

```
error ZeroAddress()
```

error (0xd92e233d): Address is address(0)

ZeroValue

```
error ZeroValue()
```

error(): Value sent is 0

ForwarderSet

```
event ForwarderSet(address, address)
```

event (0x94aed472): Forwarder address is changed

isTrustedForwarder

```
function isTrustedForwarder(address forwarder_) public view virtual returns
(bool)
```

Check if the given address is the trusted forwarder

Parameters

Name	Type	Description
forwarder_	address	Address to check

Return Values

Name	Type	Description
[0]	bool	true if forwarder_ is trusted forwarder

_setTrustedForwarder

```
function _setTrustedForwarder(address forwarder_) internal
```

Set a trusted forwarder address

emits ForwarderSet on success

Parameters

Name	Type	Description
forwarder_	address	Trusted forwarder address

setTrustedForwarder

```
function setTrustedForwarder(address forwarder_) public
```

Set the forwarder contract address

onlyOwner can call

Parameters

Name	Type	Description
forwarder_	address	Frwarder conract address

_msgSender

```
function _msgSender() internal view virtual returns (address sender)
```

Extract true caller if called via trusted forwarder

_msgData

```
function _msgData() internal view virtual returns (bytes)
```

Extract data if called via trusted forwarder

ZbyteContextDiamond

NotAForwarder

```
error NotAForwarder()
```

error (0x5ac85bab): Caller is not a forwarder

onlyOwner

```
modifier onlyOwner()
```

modifier to enforce that the caller is the owner

onlyForwarder

```
modifier onlyForwarder()
```

modifier to enforce that the caller is the forwarder

_msgSender

```
function _msgSender() internal view returns (address ret)
```

Extract true caller if called via trusted forwarder

_msgData

```
function _msgData() internal view returns (bytes ret)
```

Extract data if called via trusted forwarder

ZbyteForwarderFacet

ForwarderSet

```
event ForwarderSet(address)
```

event (0x94aed472): Forwarder address is changed

setForwarder

```
function setForwarder(address forwarder_) public
```

Set the address of trusted forwarder

Parameters

Name	Type	Description
forwarder_	address	Address of the trusted forwarder

getTrustedForwarder

```
function getTrustedForwarder() public view returns (address)
```

Get the address of trusted forwarder