Smart Contract SDK Reference

V2.0.5

Smart Contract SDK Reference
1 Document Overview
2 sdk
2.1 functions
2.1 func Require()
2.2 func RequireNotError()
2.3 func RequireOwner()
2.4 func RequireAddress()
2.5 func Array()
2.2 interfaces
2.2.1 interface ISmartContract
2.2.1 interface IBlock
2.2.3 interface ITx
2.2.4 interface IMessage
2.2.5 interface IAccount
2.2.2 interface IContract
2.2.6 interface IToken
2.2.7 interface IHelper
2.2.8 interface IAccountHelper
2.2.9 interface IBlockChainHelper
2.2.10 interface IBuildHelper
2.2.11 interface IContractHelper
2.2.12 interface IReceiptHelper
2.2.13 interface IGenesisHelper
2.2.14 interface IStateHelper
2.2.15 interface ITokenHelper
3 sdk/types
3.1 struct Error
3.1.1 definition
3.1.2 func Error()
3.1.3 code
3.2 type HexBytes
3.2.1 definition
3.2.2 func Marshal()
3.2.3 func Unmarshal()
3.2.4 func MarshalJSON()
3.2.5 func UnmarshalJSON()
3.2.6 func Bytes()
3.2.7 func String()
3.3 type Address
3.3.1 definition
3.4 type Hash
3.4.1 definition
3.5 type PubKey
3.5.1 definition
3.6 type KVPair
3.6.1 definition
4 sdk/bn
4.1 functions
4.1.1 func N()
4.1.2 func N1()
4.1.3 func N2()

4.1.4 func NB()
4.1.5 func NBS()
4.1.6 func NSBS()
4.1.7 func NBytes()
4.1.8 func NSBytes()
4.1.9 func NString()
4.1.10 func NStringHex()
4.1.11 func NewNumber()
4.1.12 func NewNumberStringBase()
4.1.13 func NewNumberBigInt()
4.1.14 func NewNumberLong()
4.1.15 func NewNumberLongLong()
4.2 class Number
4.2.1 func String()
4.2.2 func Value()
4.2.3 func CmpI()
4.2.4 func Cmp()
4.2.5 func IsZero()
4.2.6 func IsPositive()
4.2.7 func IsNegative()
4.2.8 func IsEqualI()
4.2.9 func IsEqual()
4.2.10 func IsGreaterThanI()
4.2.11 func IsGreaterThan()
4.2.12 func IsLessThanI()
4.2.13 func IsLessThan()
4.2.14 func IsGEI()
4.2.15 func IsGE()
4.2.16 func IsLEI()
4.2.17 func IsLE()
4.2.18 func AddI()
4.2.19 func Add()
4.2.20 func SubI()
4.2.21 func Sub()
4.2.22 func MulI()
4.2.23 func Mul()
4.2.24 func DivI()
4.2.25 func Div()
4.2.26 func ModI()
4.2.27 func Mod()
4.2.28 func Sq()
4.2.29 func Sqrt()
4.2.30 func Exp()
4.2.31 func Lsh()
4.2.32 func Rsh()
4.2.33 func And()
4.2.34 func Or()
4.2.35 func Xor()
4.2.36 func Not()
4.2.37 func Bytes()
4.2.37 func SetBytes()
4.2.38 func MarshalJSON()
4.2.39 func UnmarshalJSON()

5 sdk/crypto/ed25519
5.1 functions
5.1.1 func VerifySign()
6 sdk/crypto/sha3
6.1 functions
6.1.1 func Sum224()
6.1.2 func Sum256()
6.1.3 func Sum384()
6.1.4 func Sum512()
7 sdk/jsoniter
7.1 functions
7.1.1 func Marshal()
7.1.2 func Unmarshal()
8 sdk/forx
8.1 functions
8.1.1 func Range()
9 sdk/rlp
10 sdk/ISmartContract
10.1 func Block()
10.2 func Tx()
10.3 func Message()
10.4 func Helper()
11 sdk/IBlock
11.1 func ChainID()
11.2 func BlockHash()
11.3 func Height()
11.4 func Time()
11.5 func Now()
11.6 func NumTxs()
11.7 func DataHash()
11.8 func ProposerAddress()
11.9 func RewardAddress()
11.10 func RandomNumber()
11.11 func Version()
11.12 func LastBlockHash()
11.13 func LastCommitHash()
11.14 func LastAppHash()
11.15 func LastFee()
12 sdk/ITx
12.1 func Note()
12.2 func GasLimit()
12.3 func GasLeft()
12.4 func Signer()
13 sdk/IMessage
13.1 func Contract()
13.2 func MethodID()
13.3 func Items()
13.4 func GasPrice()
13.5 func Sender()
13.6 func Payer()
13.7 func Origins()
13.8 func InputReceipts()
13.9 func GetTransferToMe()

14 sdk/IAccount
- 14.1 func Address()
- 14.2 func PubKey()
- 14.3 func Balance()
- 14.4 func BalanceOfToken()
- 14.5 func BalanceOfName()
- 14.6 func BalanceOfSymbol()
- 14.7 func Transfer()
- 14.8 func TransferByToken()
- 14.9 func TransferByName()
- 14.10 func TransferBySymbol()

15 sdk/IContract
- 15.1 func Address()
- 15.2 func Account()
- 15.3 func Owner()
- 15.4 func Name()
- 15.5 func Version()
- 15.6 func CodeHash()
- 15.7 func EffectHeigt()
- 15.8 func LoseHeight()
- 15.9 func KeyPrefix()
- 15.10 func Methods()
- 15.11 func Interfaces()
- 15.12 func Mine()
- 15.13 func Token()
- 15.14 func OrgID()
- 15.15 func SetOwner()

16 sdk/IToken
- 16.1 func Address()
- 16.2 func Owner()
- 16.3 func Name()
- 16.4 func Symbol()
- 16.5 func TotalSupply()
- 16.6 func AddSupplyEnabled()
- 16.7 func BurnEnabled()
- 16.8 func GasPrice()
- 16.9 func SetTotalSupply()
- 16.10 func SetGasPrice()

17 sdk/IHelper
- 17.1 func AccountHelper()
- 17.2 func BlockchainHelper()
- 17.3 func BuildHelper()
- 17.4 func ContractHelper()
- 17.5 func GenesisHelper()
- 17.6 func ReceiptHelper()
- 17.7 func TokenHelper()
- 17.8 func StateHelper()

18 sdk/IAccountHelper
- 18.1 func AccountOf()
- 18.2 func AccountOfPubKey()

19 sdk/IBlockChainHelper
- 19.1 func CalcAccountFromPubkey()
- 19.2 func CalcAccountFromName()

19.3 func CalcContractAddress()
19.4 func CalcOrgID()
19.5 func CheckAddress()
19.6 func GetBlock()
20 sdk/IBuildHelper
20.1 func Build()
21 sdk/IContractHelper
21.1 func ContractOfAddress()
21.2 func ContractOfToken()
21.3 func ContractOfName()
22 sdk/IReceiptHelper
22.1 func Emit()
23 sdk/IGenesisHelper
23.1 func ChainID()
23.2 func OrgID()
23.3 func Contracts()
23.4 func Token()
24 sdk/ITokenHelper
24.1 func RegisterToken()
24.2 func Token()
24.3 func TokenOfAddress()
24.4 func TokenOfName()
24.5 func TokenOfSymbol()
24.6 func TokenOfContract()
24.7 func BaseGasPrice()
25 sdk/IStateHelper
25.1 func Check()
25.2 func Get()
25.3 func Set()
25.4 func Delete()
25.5 func Flush()
25.6 func McCheck()
25.7 func McGet()
25.8 func McSet()
25.9 func McClear()
25.10 func McDelete()

1 Document Overview

The BCBChain SDK is a programming interface designed specifically for programmers to develop smart contracts running on BCBChain. This document describes in detail the interfaces and types provided by the BCBChain SDK. The SDK package path is `blockchain/smcsdk/sdk`.

2 sdk

The code package `blockchain/smcsdk/sdk` encapsulates some of the helper functions needed to develop smart contracts, and the various support interfaces defined by the smart contract context.

2.1 functions

2.1 func Require()

Function Prototype:

```
func Require(expr bool, errCode uint32, errInfo string)
```

Asserts `expr` is true. If `expr` is false, the smart contract panic will be triggered with a type `types.Error` object having the error code as the specified `errCode` and the error message as the specified `errInfo`. If `expr` is true, the smart contract will be allowed to continue to execute.

2.2 func RequireNotError()

Function Prototype:

```
func RequireNotError(err error, errCode uint32)
```

Asserts `err` is not for error, that the `err` object is empty. If the `err` object is not empty, the smart contract panic will be triggered with a type `types.Error` object having the error code as the specified `errCode` and the error message is the error description in the `err` object information. If `err` is empty, the smart contract will be allowed to continue to execute.

2.3 func RequireOwner()

Function Prototype:

```
func RequireOwner()
```

Asserts that the caller must be the owner of the contract. If the requirement is not met, the smart contract panic will be triggered with a type `types.Error` object having error code `types.ErrNoAuthorization` and the specific error reason as the error message. If the requirements are met, the smart contract will be allowed to continue to execute.

2.4 func RequireAddress()

Function Prototype:

```
func RequireAddress(addr types.Address)
```

Asserts `addr` is a valid address. If the address format is incorrect, the smart contract panic will be triggered with a type `types.Error` object having error code as `types.ErrInvalidAddress`, and the error message as the specific error reason. If the address is formatted correctly, the smart contract will be allowed to continue to execute.

2.5 func Array()

Function Prototype:

```
func Array(items ...interface{}) []interface{}
```

Creates an array to store multiple objects.

2.2 interfaces

2.2.1 interface ISmartContract

See the `sdk/ISmartContract` section of this document.

2.2.1 interface IBlock

See the `sdk/IBlock` section of this document.

2.2.3 interface ITx

See the `sdk/ITx` section of this document.

2.2.4 interface IMessage

See the `sdk/IMessage` section of this document.

2.2.5 interface IAccount

See the `sdk/IAccount` section of this document.

2.2.2 interface IContract

See the `sdk/IContract` section of this document.

2.2.6 interface IToken

See the `sdk/IToken` section of this document.

2.2.7 interface IHelper

See the `sdk/IHelper` section of this document.

2.2.8 interface IAccountHelper

See the `sdk/IAccountHelper` section of this document.

2.2.9 interface IBlockChainHelper

See the `sdk/IBlockChainHelper` section of this document.

2.2.10 interface IBuildHelper

See the `sdk/IBuildHelper` section of this document.

2.2.11 interface IContractHelper

See the `sdk/IContractHelper` section of this document.

2.2.12 interface IReceiptHelper

See the `sdk/IReceiptHelper` section of this document.

2.2.13 interface IGenesisHelper

See the `sdk/IGenesisHelper` section of this document.

2.2.14 interface IStateHelper

See the `sdk/IStateHelper` section of this document.

2.2.15 interface ITokenHelper

See the `sdk/ITokenHelper` section of this document.

3 sdk/types

The code package `blockchain/smcsdk/sdk/types` encapsulates some of the basic data types needed to develop smart contracts.

3.1 struct Error

The standard type provided by the SDK that describes the error.

3.1.1 definition

```
//Error define type for error of sdk
type Error struct {
    ErrorCode uint32 // Error code
    ErrorDesc string // Error description
}
```

3.1.2 func Error()

Function Prototype:

```
func (err *Error) Error() string
```

Returns the error message text.

3.1.3 code

```
CodeOK = 200                        ErrorDesc = ""

//for stub
ErrStubDefined = 51000              ErrorDesc = "Error stub defined"
ErrGasNotEnough = 51001             ErrorDesc = "Gas limit is not enough"
ErrFeeNotEnough = 51002             ErrorDesc = "Insufficient balance to pay fee"

//for sdk
ErrAddSupplyNotEnabled = 52000      ErrorDesc = "Add supply is not enabled"
ErrBurnNotEnabled = 52001           ErrorDesc = "Burn supply is not enabled"
ErrInvalidAddress = 52002           ErrorDesc = "Invalid address"

//for contract
ErrNoAuthorization = 53000          ErrorDesc = "No authorization to execute contract"
ErrInvalidParameter = 53001         ErrorDesc = "Invalid parameter"
ErrInsufficientBalance = 53002      ErrorDesc = "Insufficient balance"
```

```
//for user defined
ErrUserDefined = 55000            ErrorDesc = "Error user defined"
```

3.2 type HexBytes

The byte array type provided by the SDK encapsulates some common serialization interfaces.

3.2.1 definition

```
type HexBytes []byte
```

3.2.2 func Marshal()

Function Prototype:

```
func (bz HexBytes) Marshal() ([]byte, error)
```

Implements a standard binary serialization interface.

3.2.3 func Unmarshal()

Function Prototype:

```
func (bz *HexBytes) Unmarshal(data []byte) error
```

Implement a standard binary deserialization interface that sets the value of `bz` to the value of `data`.

3.2.4 func MarshalJSON()

Function Prototype:

```
func (bz HexBytes) MarshalJSON() ([]byte, error)
```

Implements the standard JSON serialization interface.

3.2.5 func UnmarshalJSON()

Function Prototype:

```
func (bz *HexBytes) UnmarshalJSON(data []byte) error
```

Implements the standard JSON deserialization interface that sets the value of `bz` to the value of the JSON string `data`.

3.2.6 func Bytes()

Function Prototype:

```
func (bz HexBytes) Bytes() []byte
```

Implement a standard get byte array interface.

3.2.7 func String()

Function Prototype:

```
func (bz HexBytes) String() string
```

Implements the standard get string interface, converting `` `bz `` to all uppercase hex strings.

3.3 type Address

The standard type provided by the SDK that describes the account address.

3.3.1 definition

```
type Address = string
```

3.4 type Hash

The standard type that the SDK provides for hash data.

3.4.1 definition

```
type Hash = HexBytes
```

3.5 type PubKey

The standard type that the SDK provides to describe public key data.

3.5.1 definition

```
type PubKey = HexBytes
```

3.6 type KVPair

The standard type that the SDK provides for describing key/value pairs.

3.6.1 definition

```
type KVPair struct {
    Key   []byte `protobuf:"bytes,1,opt,name=key,proto3" json:"key,omitempty"`
    Value []byte `protobuf:"bytes,2,opt,name=value,proto3" json:"value,omitempty"`
}
```

4 sdk/bn

The package `blockchain/smcsdk/sdk/bn` encapsulates a class that handles large numbers without having to worry about overflow when performing adding, subtracting, multiplying, and dividing operations.

4.1 functions

This section describes the simple constructor for the class `Number` provided by the package `sdk/bn`.

4.1.1 func N()

Function Prototype:

```
func N(x int64) Number
```

Converts `x` of type `int64` to a `Number` type object and returns it.

4.1.2 func N1()

Function Prototype:

```
func N1(b int64, d int64) Number
```

Calculates and returns the result of `b * d` as a `Number` object

4.1.3 func N2()

Function Prototype:

```
func N2(b int64, d1, d2 int64) Number
```

Calculates and returns the result of `b * d1 * d2` as a `Number` type object.

4.1.4 func NB()

Function Prototype:

```
func NB(x *big.Int) Number
```

Converts `x` of type `big.Int` to a `Number` type object and returns it.

4.1.5 func NBS()

Function Prototype:

```
func NBS(x []byte) Number
```

Converts a byte array `x` that represents an large unsigned integer to a `Number` type object and returns it.

4.1.6 func NSBS()

Function Prototype:

```
func NSBS(x []byte) Number
```

Converts a byte array `x` that represents an large signed integer to a `Number` type object and returns it.

4.1.7 func NBytes()

Function Prototype:

```
func NBytes(x []byte) Number
```

Converts a byte array `x` that represents an unsigned large integer to a `Number` type object and returns it.

4.1.8 func NSBytes()

Function Prototype:

```
func NSBytes(x []byte) Number
```

Converts a byte array `x` representing a large signed integer to a `Number` type object and returns it.

4.1.9 func NString()

Function Prototype:

```
func NS(s string) Number
```

Converts a large integer `s` represented by a decimal string to a `Number` type object and returns it. If the parse fails, it will return `0` instead.

4.1.10 func NStringHex()

Function Prototype:

```
func NS(s string) Number
```

Converts an unsigned large integer `s` represented by a hexadecimal string beginning with `0x` or `0X` to a `Number` type object and returns it. If the parse fails, it will return `0` instead.

4.1.11 func NewNumber()

Function Prototype:

```
func NewNumber(x int64) Number
```

Converts a large integer `x` of type `int64` to an object of type `Number` and returns it .

4.1.12 func NewNumberStringBase()

Function Prototype:

```
func NewNumberStringBase(s string, base int) Number
```

Converts the large integer in string data type s into a Number type object and returns it. The string is parsed according to the given number base, and returns 0 if the parsing fails.

The number `base` must be an integer `0` or between `2` to `MaxBase`. If the `base` argument is `0`, the function will derive the actual conversion base from the prefix of the string argument `s`: the prefix is `"0x"`, `"0X"` for hexadecimal; the prefix `"0b"`, `"0B"` for binary; the prefix `"0"` for octal; The default base is decimal.

For the number bases of `<= 36`, uppercase and lowercase letters express the same number, and the letters `'a'` through `'z'` and `'A'` to `'Z'` all express values 10 through 35.

For the number base of `> 36`, the uppercase letters `'A'` through `'Z'` express values `36` to `61`.

4.1.13 func NewNumberBigInt()

Function Prototype:

```
func NewNumberBigInt(x *big.Int) Number
```

Converts a large integer `x` of type `big.Int` to a `Number` type object and returns it.

4.1.14 func NewNumberLong()

Function Prototype:

```
func NewNumberLong(b int64, d int64) Number
```

Calculates and returns the result of the arguments `b * d` as a `Number` object.

4.1.15 func NewNumberLongLong()

Function Prototype:

```
func NewNumberLongLong(b int64, d1, d2 int64) Number
```

Calculates and returns the result of the arguments `b * d1 * d2` as a `Number` object

4.2 class Number

This section describes the member functions of the `Number` class.

4.2.1 func String()

Function Prototype:

```
func (x Number) String() string
```

Implements a standard get string() interface that converts `x` to a string in decimal number base. Returns `nil` If the initial large value is not set.

4.2.2 func Value()

Function Prototype:

```
func (x Number) Value() *big.Int
```

Returns the value of `x` in `big.Int`. Returns `nil` if the initial large value is not set.

4.2.3 func CmpI()

Function Prototype:

```
func (x Number) CmpI(y int64) int
```

Compares `x` with `y`, return `-1` for `x < y`, `0` for `x == y`, and `+1` for `x > y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.4 func Cmp()

Function Prototype:

```
func (x Number) Cmp(y Number) int
```

Compares `x` with `y`, return `-1` for `x < y`, `0` for `x == y`, and `+1` for `x > y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.5 func IsZero()

Function Prototype:

```
func (x Number) IsZero() bool
```

Determines if `x` equals `0`. Returns true if `x` equals `0`. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.6 func IsPositive()

Function Prototype:

```
func (x Number) IsPositive() bool
```

Determines if `x` is a positive integer (not including `0`). Returns true if `x` is positive. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.7 func IsNegative()

Function Prototype:

```
func (x Number) IsNegative() bool
```

Determines if `x` is a negative integer. Returns true if x is negative. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.8 func IsEqualI()

Function Prototype:

```
func (x Number) IsEqualI(y int64) bool
```

Compares `x` with `y`, returns true for `x == y`, and false for `x != y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.9 func IsEqual()

Function Prototype:

```
func (x Number) IsEqual(y Number) bool
```

Compares `x` with `y`, returns true for `x == y`, and false for `x != y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.10 func IsGreaterThanI()

Function Prototype:

```
func (x Number) IsGreaterThanI(y int64) bool
```

Compares `x` with `y`, returns true for `x > y`, and false for `x <= y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.11 func IsGreaterThan()

Function Prototype:

```
func (x Number) IsGreaterThan(y Number) bool
```

Compares `x` with `y`, returns true for `x > y`, and false for `x <= y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.12 func IsLessThanI()

Function Prototype:

```
func (x Number) IsLessThanI(y int64) bool
```

Compares `x` with `y`, returns true for `x < y`, and false for `x >= y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.13 func IsLessThan()

Function Prototype:

```
func (x Number) IsLessThan(y Number) bool
```

Compares `x` with `y`, returns true for `x < y`, and false for `x >= y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.14 func IsGEI()

Function Prototype:

```
func (x Number) IsGEI()(y int64) bool
```

Compares `x` with `y`, returns true for `x >= y`, and false for `x < y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.15 func IsGE()

Function Prototype:

```
func (x Number) IsGE(y Number) bool
```

Compares `x` with `y`, returns true for `x >= y`, and false for `x < y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.16 func IsLEI()

Function Prototype:

```
func (x Number) IsLEI(y int64) bool
```

Compares `x` with `y`, returns true for `x <= y`, and false for `x > y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.17 func IsLE()

Function Prototype:

```
func (x Number) IsLE(y Number) bool
```

Compares `x` with `y`, returns true for `x <= y`, and false for `x > y`. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.18 func AddI()

Function Prototype:

```
func (x Number) AddI(y int64) Number
```

Calculates `x + y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.19 func Add()

Function Prototype:

```
func (x Number) Add(y Number) Number
```

Calculates `x + y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.20 func SubI()

Function Prototype:

```
func (x Number) SubI(y int64) Number
```

Calculates `x - y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.21 func Sub()

Function Prototype:

```
func (x Number) Sub(y Number) Number
```

Calculates `x - y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.22 func MulI()

Function Prototype:

```
func (x Number) MulI(y int64) Number
```

Calculates `x * y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.23 func Mul()

Function Prototype:

```
func (x Number) Mul(y Number) Number
```

Calculates `x * y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.24 func DivI()

Function Prototype:

```
func (x Number) DivI(y int64) Number
```

Calculates `x / y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.25 func Div()

Function Prototype:

```
func (x Number) Div(y Number) Number
```

Calculates `x / y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.26 func ModI()

Function Prototype:

```
func (x Number) ModI(y int64) Number
```

Calculates `x % y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.27 func Mod()

Function Prototype:

```
func (x Number) Mod(y Number) Number
```

Calculates `x % y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.28 func Sq()

Function Prototype:

```
func (x Number) Sq() Number
```

Calculates `x ** 2` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.29 func Sqrt()

Function Prototype:

```
func (x Number) Sqrt() Number
```

Calculates the square root of `x` and returns the result of the calculation. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.30 func Exp()

Function Prototype:

```
func (x Number) Exp(y Number) Number
```

Calculates `x ** y` and returns the result of the calculation. If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.31 func Lsh()

Function Prototype:

```
func (x Number) Lsh(n uint) Number
```

Shifts `x` to the left by `n` bits and returns the result. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.32 func Rsh()

Function Prototype:

```
func (x Number) Rsh(n uint) Number
```

Shifts `x` to the right by `n` bits and returns the result. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.33 func And()

Function Prototype:

```
func (x Number) And(y Number) Number
```

Calculates `x & y` by bits and returns the result of the calculation (Note that `x` and `y` may be negative numbers, and bitwise and binary complements are used). If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.34 func Or()

Function Prototype:

```
func (x Number) Or(y Number) Number
```

Calculates `x | y` by bits and returns the result of the calculation (Note that `x` and `y` may be negative numbers, and bitwise and binary complements are used). If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.35 func Xor()

Function Prototype:

```
func (x Number) Xor(y Number) Number
```

Calculates `x ^ y` by bits and returns the result of the calculation (Note that `x` and `y` may be negative numbers, and bitwise and binary complements are used). If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.36 func Not()

Function Prototype:

```
func (x Number) Not() Number
```

Calculates `^ x` by bits and returns the result of the calculation (Note that `x` may be negative numbers, and bitwise and binary complements are used). If the large value of `x` or `y` is not set, the function will trigger a `panic`.

4.2.37 func Bytes()

Function Prototype:

```
func (x Number) Bytes() []byte
```

Implements the standard get byte array interface. Converts `x` to a big endian byte array (the first byte is interpreted as a representation symbol, the negative number is `0xFF`, the non-negative number is `0x00`), and the conversion result is returned. For example: `380` will be converted to `0x00017C`; `-380` will be converted to `0xFF017C`. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.37 func SetBytes()

Function Prototype:

```
func (x *Number) SetBytes(buf []byte) Number
```

Sets the value of `x` to a big endian byte array (the first byte is `0xFF` to indicate a negative number, the absolute value is encoded from the second byte, otherwise the entire byte slice represents a non-negative integer), and returns the value of `x`. For example: `0x00017C` and `0x017C` will all be converted to `380`; `0xFF017C` will be converted to `-380`.

4.2.38 func MarshalJSON()

Function Prototype:

```
func (x Number) MarshalJSON() (data []byte, err error)
```

Implements the standard JSON serialization interface. Converts `x` to a simplified version of a JSON string, such as string `380`. If the large value of `x` is not set, the function will trigger a `panic`.

4.2.39 func UnmarshalJSON()

Function Prototype:

```
func (x *Number) UnmarshalJSON(data []byte) error
```

Implements the standard JSON deserialization interface. Sets the value of `x` to the large number corresponding to the input JSON string. Support for a simplified version of a JSON string (for example: string `380`) and a structured version of a JSON string (for example: the string `{"v":380}`).

5 sdk/crypto/ed25519

The package `blockchain/smcsdk/sdk/crypto/ed25519` encapsulates a simple application interface to the elliptic curve `ed25519`.

5.1 functions

5.1.1 func VerifySign()

Function Prototype:

```
func VerifySign(pubkey, data, sign []byte) bool
```

Verifies the signature and returns true if successful. `pubkey` is the public key, `data` is the signed data, and `sign` is the signature data.

6 sdk/crypto/sha3

The package `blockchain/smcsdk/sdk/crypto/sha3` encapsulates a simple application interface to the hashing algorithm `SHA-3`.

6.1 functions

6.1.1 func Sum224()

Function Prototype:

```
func Sum224(datas ...[]byte) []byte
```

Uses the `SHA3-224` algorithm to calculate the 224-bits hash value of the input data table (multiple input parameters are calculated in the input order) and returns the result of the calculation.

6.1.2 func Sum256()

Function Prototype:

```
func Sum256(datas ...[]byte) []byte
```

Uses the `SHA3-256` algorithm to calculate the 256-bits hash value of the input data table (multiple input parameters are calculated in the order of input) and returns the result of the calculation.

6.1.3 func Sum384()

Function Prototype:

```
func Sum384(datas ...[]byte) []byte
```

Uses the `SHA3-384` algorithm to calculate the 384-bits hash value of the input data table (multiple input parameters are calculated in the input order) and returns the result of the calculation

6.1.4 func Sum512()

Function Prototype:

```
func Sum512(datas ...[]byte) []byte
```

Uses the `SHA3-512` algorithm to calculate the 512-bits hash value of the input data table (multiple input parameters are calculated in the order of input) and returns the result of the calculation.

7 sdk/jsoniter

The package `blockchain/smcsdk/sdk/jsoniter` encapsulates a simple application interface to `jsoniter`, a third-party package that quickly processes JSON data.

7.1 functions

7.1.1 func Marshal()

Function Prototype:

```
func Marshal(v interface{}) ([]byte, error)
```

Serializes the input object `v` into a JSON-formatted string and returns it.

7.1.2 func Unmarshal()

Function Prototype:

```
func Unmarshal(bz []byte, v interface{}) error
```

Deserializes the input JSON string to the object pointed to by `v`.

8 sdk/forx

The package `blockchain/smcsdk/sdk/mapx` encapsulates the application interface that optimizes the `for` loop.

8.1 functions

8.1.1 func Range()

Function Prototype:

```
func Range(args... interface{})
```

The keyword `for` is not allowed to be used in any part of the smart contract code.

To perform loops, use the `Range()` function instead.

There are six ways to use the `Range()` function, examples are as follows:

> Model one:
>
> ```
> func Range( m map[keyType]valType, f func(key keyType, val valType) )
> func Range( m map[keyType]valType, f func(key keyType, val valType) bool )
> ```
>
> ```
> The traversal operation is performed on the mapping table object m, and the order of the
> traversal is based on the mapping table key. The operation function is f. If the return
> value of f is defined as void, the loop can be ended after the traversal is finished. If
> the return value of f is a boolean, false value will terminate the execution of the
> traversal operation and exit the loop, while true value will continue the traversal
> operation. The input parameters are required to meet the following conditions. If any
> one is not satisfied, an exception occurs. When an exception occurs, the SDK will
> automatically trigger a panic:
> ```
>
> ```
> 1. The type of the parameter m must be a mapping table;
> 2. The parameter f must be a function, and the return value of the function must be
> between void and boolean type;
> 3. Function f must have two arguments
> 4. The first parameter type of the function f must be the type corresponding to the
> key of the mapping table m;
> 5. The second parameter type of the function f must be the type corresponding to the
> value of the mapping table m.
> ```
>
> The sample code is as follows:

```
m := make(map[int]string)
m[93] = "23231"
m[13] = "23423423234324"
m[54] = "3432432423"
m[23] = "3434545345345"

forx.Range(m, func(k int, v string) {
    printf("key=%v value=%v\n", k, v)
})
```

Model two:

```
func Range( s []valType, f func(i int, val valType) )
func Range( s []valType, f func(i int, val valType) bool )
```

```
The traversal operation is performed on the object array s. The traversal is performed
in order of the array order from front to end. The operation function is f. If the
return value of f is defined as void, the loop can be ended after the traversal is
finished.  If the return value of f is a boolean, false value will terminate the
execution of the traversal operation and exit the loop, while true value will continue
the traversal operation. The input parameters are required to meet the following
conditions. If any one is not satisfied, an exception occurs. When an exception occurs,
the SDK will automatically trigger a panic:
```

```
1. The type of the parameter m must be an array;
2. The parameter f must be a function, and the return value of the function must be
between void and boolean type;
3. Function f must have two arguments
4. The first argument type of function f must be of type int, indicating the index
number of the element, starting at 0;
5. The second parameter type of function f must be the type corresponding to the
element value of array s.
```

The sample code is as follows:

```
s := make([]string)
s = append(s, "23231")
s = append(s, "423792234")
s = append(s, "23232")
s = append(s, "3243455454")

forx.Range(s, func(i int, v string) {
    printf("i=%v value=%v\n", i, v)
})
```

Model three:

```
func Range( n intType, f func(i intType) )
func Range( n intType, f func(i intType) bool )
```

```
The loop executes for the specified number of times. The operation function is f. If the return value of f is defined as void, the loop can be ended after the specified number of times. If the return value of f is a boolean, false value will exit the loop, while true value will continue the loop. The input parameters are required to meet the following conditions. If any one is not satisfied, an exception occurs. When an exception occurs, the SDK will automatically trigger a panic:
```

```
1. The type of the parameter n must be a type that expresses an integer, such as: int, uint, int32, etc., and the value must be greater than or equal to 0;
2. The parameter f must be a function, and the return value of the function must be between void and boolean type;
3. Function f must have exactly one argument. The type must be the same type of argument n, indicating the index number of the execution loop, starting at 0.
```

The sample code is as follows:

```
forx.Range(10, func(i int) {
    printf("i=%v\n", i)
})
```

Model four:

```
func Range( m,n intType, f func(k intType) )
func Range( m,n intType, f func(k intType) bool )
```

```
According to the input parameters, all integers between m and n (including m and n) are traversed, starting from m and ending at n, the operation function is f. If the return value of f is defined as void, the loop can be ended after the traversal is over. If the return value of f is a boolean, false value will terminate the execution of the traversal operation and exit the loop, while true value will continue the traversal operation. The input parameters are required to meet the following conditions. If any one is not satisfied, an exception occurs. When an exception occurs, the SDK will automatically trigger a panic:
```

```
1. The parameters m and n must be of the same type that express an integer, for example: int, uint, int32, etc. If m is less than n, then the loop will be in ascending order. If m is greater than n, the order will be descending;
2. The parameter f must be a function, and the return value of the function must be between void and boolean type;
3. Function f must have exactly one argument. The type must be the same type as the arguments m and n, representing the integer value traversed, starting at m.
```

The sample code is as follows:

```
forx.Range(1,100, func(k int) {
    printf("k=%v\n", k)
})
```

Model five:

```
func Range( c func() bool, f func(i int) )
func Range( c func() bool, f func(i int) bool )
```

```
The loop executes the function f. When the control function c returns true, the loop is terminated. This controls when the loop ends. If the return value of f is a boolean, false value will exit the loop, while true value will continue the loop. The input parameters are required to meet the following conditions. If any one is not satisfied, an exception occurs. When an exception occurs, the SDK will automatically trigger a panic:
```

```
1. The parameter c must be a function, and the return value of the function must be a boolean type;
2. The parameter f must be a function, and the return value of the function must be between void and boolean type;
3. Function f must have exactly one argument. The type must be int, indicating the index of the number of execution loops, starting at 0.
```

The sample code is as follows:

```
toVoter := ballot._voters(to)
forx.range( func() bool {
              return toVoter.delegate != ""
         },
         func(i int) {
              to = toVoter.delegate
            toVoter = ballot._voters(to)
            })
```

Model six:

```
func Range( f func(i int) bool )
```

```
The loop will execute function f in each iteration. Since the return value of f is a boolean, false value will exit the loop, while true value will continue the loop. The input parameters are required to meet the following conditions. If any one is not satisfied, an exception occurs. When an exception occurs, the SDK will automatically trigger a panic:
```

```
1. The argument f must be a function, and the return value of the function must be a boolean type;
2. Function f must have exactly one argument. The type must be int, which represents the index of the number of execution loops, starting at 0.
```

The sample code is as follows:

```
toVoter := ballot._voters(to)
forx.Range( func(i int) bool {
  to = toVoter.delegate
 toVoter = ballot._voters(to)

 if toVoter.delegate != "" {
    return forx.Break
  } else {
    return forx.Continue
     }
   })
```

9 sdk/rlp

The package `blockchain/smcsdk/sdk/rlp` encapsulates the BCBChain RLP encoding of the transaction. The source code is based on the open source version of `geth`. Due to BCBChain requirements, codec support for `bn.Number` large numbers is added, as well as support for mapping tables.

10 sdk/ISmartContract

The interface `sdk/ISmartContract` encapsulates the path to smart contract context acquisition.

10.1 func Block()

Function Prototype:

```
func (ISmartContract) Block() IBlock
```

Returns the block data when the blockchain of BCBChain is called.

10.2 func Tx()

Function Prototype:

```
func (ISmartContract) Tx() ITx
```

Returns the transaction data called on the blockchain of BCBChain.

10.3 func Message()

Function Prototype:

```
func (ISmartContract) Message() IMessage
```

Returns the message for this smart contract call.

10.4 func Helper()

Function Prototype:

```
func (ISmartContract) Helper() IHelper
```

Returns a helper object that encapsulates all the support functions of the SDK.

11 sdk/IBlock

The interface `sdk/IBlock` encapsulates the block information obtained by the smart contract.

When a smart contract is invoked, the `IBlock` interface will have different block information depending on two different phases.

> In the checkTx phase, the current smart contract call has yet to reach a consensus, ISmartContract::Block() is called to obtain an simulated block (may be an empty block. In short, this block has nothing to do with the currently executed smart contract call) generated above the last block on the blockchain (block height plus 1).

> In the deliverTx phase, the transaction that invoked the smart contract has reached a consensus on the blockchain and is written to the latest block. At this point, ISmartContract::Block() is called to get the latest real-time information on the blockchain. (The block now cannot be an empty block, it contains at least one transaction information called for this smart contract).

11.1 func ChainID()

Function Prototype:

```
func (IBlock) ChainID() string
```

Returns the chain ID of the current blockchain.

11.2 func BlockHash()

Function Prototype:

```
func (IBlock) BlockHash() types.Hash
```

Returns the block hash of the current block.

11.3 func Height()

Function Prototype:

```
func (IBlock) Height() int64
```

Returns the height of the current block.

11.4 func Time()

Function Prototype:

```
func (IBlock) Time() int64
```

Returns the timestamp when the current block was created. The result is the number of seconds past the `1970-1-1 00:00:00`.

11.5 func Now()

Function Prototype:

```
func (IBlock) Now() bn.Number
```

An alias for the `Time()` function. Returns the timestamp when the current block was created. The result is the number of seconds past the `1970-1-1 00:00:00`.

11.6 func NumTxs()

Function Prototype:

```
func (IBlock) NumTxs() int32
```

Returns the number of transactions in the current block.

11.7 func DataHash()

Function Prototype:

```
func (IBlock) DataHash() types.Hash
```

Returns the data hash of the current block.

11.8 func ProposerAddress()

Function Prototype:

```
func (IBlock) ProposerAddress() types.Address
```

Returns the proposer address of the current block.

11.9 func RewardAddress()

Function Prototype:

```
func (IBlock) RewardAddress() types.Address
```

Returns the reward address of the current block.

11.10 func RandomNumber()

Function Prototype:

```
func (IBlock) RandomNumber() types.HexBytes
```

Returns the block random number of the current block.

11.11 func Version()

Function Prototype:

```
func (IBlock) Version() string
```

Returns the software version when the current block proposer created the block.

11.12 func LastBlockHash()

Function Prototype:

```
func (IBlock) LastBlockHash() types.Hash
```

Returns the block hash of the previous block.

11.13 func LastCommitHash()

Function Prototype:

```
func (IBlock) LastCommitHash() types.Hash
```

Returns the commit hash of the previous block.

11.14 func LastAppHash()

Function Prototype:

```
func (IBlock) LastAppHash() types.Hash
```

Returns the application hash of the previous block.

11.15 func LastFee()

Function Prototype:

```
func (IBlock) LastFee() int64
```

Returns the handling fee (unit: `cong`) of the previous block.

12 sdk/ITx

The interface `sdk/ITx` encapsulates the transaction information that is called on the BCBChain blockchain.

12.1 func Note()

Function Prototype:

```
func (ITx) Note() string
```

Returns the comment information of the current transaction.

12.2 func GasLimit()

Function Prototype:

```
func (ITx) GasLimit() int64
```

Returns the max gas limit of the current transaction.

12.3 func GasLeft()

Function Prototype:

```
func (ITx) GasLeft() int64
```

Returns the amount of gas remaining in the current transaction (the amount of gas required for the current smart contract call has been pre-deducted).

12.4 func Signer()

Function Prototype:

```
func (ITx) Signer() IAccount
```

Returns the account object of the signer (initiator) for the current transaction.

13 sdk/IMessage

The interface `sdk/IMessage` encapsulates all the information related to a single call to a smart contract method. In the same transaction, you can cascade multiple messages that are called for different smart contract methods.

13.1 func Contract()

Function Prototype:

```
func (IMessage) Contract() types.Address
```

Returns the smart contract address called by the current message。

13.2 func MethodID()

Function Prototype:

```
func (IMessage) MethodID() string
```

Returns the smart contract method ID (represented by a hex string) called by the current message.

13.3 func Items()

Function Prototype:

```
func (IMessage) Items() []types.HexBytes
```

Returns the raw data of each parameter field called by the current message. When a cross-contract call occurs, the data obtained by `Items()` is `nil` inside the called contract.

13.4 func GasPrice()

Function Prototype:

```
func (IMessage) GasPrice() int64
```

Returns the gas price (unit: `cong`) called by the current message.

13.5 func Sender()

Function Prototype:

```
func (IMessage) Sender() IAccount
```

Returns the account object of the caller. If the call is made on the first level, the caller is the signer of the transaction. If the call is cross-contract, within the called contract, `Sender()` returns the contract account object that initiated the call.

13.6 func Payer()

Function Prototype:

```
func (IMessage) Payer() IAccount
```

Returns the account object that pays the current message handling fee.

13.7 func Origins()

Function Prototype:

```
func (IMessage) Origins() []types.Address
```

Returns the complete call chain of the message. If it is not a cross-contract call, `Origins()` returns an array containing the account address of the originator of the transaction. If it is a cross-contract call, `Origins()` returns the contract chain that represents the call. Within the called contract, the last address in the list of addresses returned by `Origins()` is the contract address of the contract that initiated the call.

13.8 func InputReceipts()

Function Prototype:

```
func (IMessage) InputReceipts() []types.KVPair
```

When the cascading and cross-contract message is called, the receipt of the previous message call output will be used as the input of the next message call. This function returns the array of receipts entered into the current message call.

13.9 func GetTransferToMe()

Function Prototype:

```
func (IMessage) GetTransferToMe() []*std.Transfer
```

Queries and returns all receipts that are transferred to the current smart contract in the receipt of the previous message call output, or `nil` if not found. The transfer receipt is defined as follows:

```
package std

type Transfer struct {
    Token types.Address `json:"token"` // token address
    From  types.Address `json:"from"`  // account address of fund transferor
    To    types.Address `json:"to"`    // account address of receiver of funds
    Value bn.Number     `json:"value"` // transfer amount (unit: ```cong```)
}
```

14 sdk/IAccount

The interface `sdk/IAccount` encapsulates the access interface to an account address.

14.1 func Address()

Function Prototype:

```
func (IAccount) Address() types.Address
```

Returns the account address of the account object.

14.2 func PubKey()

Function Prototype:

```
func (IAccount) PubKey() types.PubKey
```

Returns the public key data of the account object (may be empty).

14.3 func Balance()

Function Prototype:

```
func (IAccount) Balance() bn.Number
```

In a token-issuing contract, the fund balance of the token-issuing sub-account issued by the object of the account is returned (unit: `cong`). In a non-token-issuing contract, it is return `0`.

14.4 func BalanceOfToken()

Function Prototype:

```
func (IAccount) BalanceOfToken(token types.Address) bn.Number
```

Given the token address, return the fund balance (unit: `cong`) of the account object in the specified token sub-account. If the specified token address is not a token, the return balance is `0`.

14.5 func BalanceOfName()

Function Prototype:

```
func (IAccount) BalanceOfName(name string) bn.Number
```

Given the token name, return the fund balance (unit: `cong`) of the account object in the specified token sub-account. If the specified token name is not a token, the return balance is `0`.

14.6 func BalanceOfSymbol()

Function Prototype:

```
func (IAccount) BalanceOfSymbol(symbol string) bn.Number
```

Given the token symbol, return the fund balance (unit: `cong`) of the account object in the specified token sub-account. If the specified token symbol is not a token, the return balance is `0`.

14.7 func Transfer()

Function Prototype:

```
func (IAccount) Transfer(to types.Address, value bn.Number)
```

In a token-issuing contract, transfers the funds of account object from token sub-account to the account designated in the parameter `to` (unit: `cong`), whereby the account object includes the account that initiated the message or the current contract account.

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically triggered a `panic`:

```
1. The parameter to is not an address;
2. The parameter to is the same as the account address;
3. The parameter value is less than or equal to 0;
4. The balance of the account is insufficient to pay the transfer;
5. The current contract has not issued tokens (because the parameters of the function do
   not specify which token to transfer);
6. The current contract has issued tokens, but the account is neither the one that
   initiated the message call nor the contract account of the current smart contract.
```

14.8 func TransferByToken()

Function Prototype:

```
func (IAccount) TransferByToken(token types.Address, to types.Address, value bn.Number)
```

If the specified token address is the token issued by the contract, transfers the funds of account object from token sub-account to the account designated in the parameter `to` (unit: `cong`), whereby the account object includes the account that initiated the message or the current contract account.

If the specified token address is not the token issued by this contract, transfers the funds of account object from the specified token address to the account designated as `to` (unit: `cong`), whereby the account object can only be the current contract account.

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically triggered a `panic`:

```
1. The parameter token is not an address;
2. The parameter token is not a token;
3. The parameter to is not an address;
4. The parameter to is the same as the account address;
5. The parameter value is less than or equal to 0;
6. The balance of the account is insufficient to pay the transfer;
7. If the current contract does not issue tokens, and the account is not the account of
   the current smart contract;
8. The current contract has issued tokens, but the account is neither the one that
   initiated the message call nor the contract account of the current smart contract.
```

14.9 func TransferByName()

Function Prototype:

```
func (IAccount) TransferByName(name string, to types.Address, value bn.Number)
```

If the designated token name is the token issued by the contract, transfers the funds of account object from token sub-account to the account designated in the parameter `to` (unit: `cong`), whereby the account object includes the account that initiated the message or the current contract account.

If the specified token address is not the token issued by this contract, transfers the funds of account object from the specified token address to the account designated as to (unit: `cong`), whereby the account object can only be the current contract account. .

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically triggered a panic:

```
1. The parameter name is not the name of a token;
2. The parameter to is not an address;
3. The parameter to is the same as the account address;
4. The parameter value is less than or equal to 0;
5. The balance of the account is insufficient to pay the transfer;
6. If the current contract does not issue tokens, and the account is not the account of
   current smart contract;
7. The current contract has issued tokens, but the account is neither the one that
   initiated the message call nor the contract account of the current smart contract.
```

14.10 func TransferBySymbol()

Function Prototype:

```
func (IAccount) TransferBySymbol(symbol string, to types.Address, value bn.Number)
```

If the designated token symbol is the token issued by the contract, transfers the funds of account object from token sub-account to the account designated in the parameter `to` (unit: `cong`), whereby the account object includes the account that initiated the message or the current contract account.

If the specified token symbol is not the token issued by this contract, transfers the funds of account object from the specified token address to the account designated as `to` (unit: `cong`), whereby the account object can only be the current contract account. .

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically triggered a panic:

```
1. The parameter symbol is not the symbol of a token;
2. The parameter to is not an address;
3. The parameter to is the same as the account address;
4. The parameter value is less than or equal to 0;
5. The balance of the account is insufficient to pay the transfer;
6. If the current contract does not issue tokens, and the account is not the account of
   current smart contract;
7. The current contract has issued tokens, but the account is neither the one that
   initiated the message call nor the contract account of the current smart contract.
```

15 sdk/IContract

The interface `sdk/IContract` encapsulates all the information about a smart contract and the interface to operate it.

15.1 func Address()

Function Prototype:

```
func (IContract) Address() types.Address
```

Returns the address of the smart contract.

15.2 func Account()

Function Prototype:

```
func (IContract) Account() IAccount
```

Returns the account object of the smart contract.

The account address of the smart contract can be used to receive funds transferred to the contract. The calculation of the smart contract account address only requires the contract name and organization ID. The change in the address after the contract upgrades will not affect the address of the account. The contract account address does not contain a corresponding private key, so the funds in the contract account can only be changed through the contract.

15.3 func Owner()

Function Prototype:

```
func (IContract) Owner() IAccount
```

Returns the external account object of the owner of the smart contract (the external account address has a corresponding private key).

15.4 func Name()

Function Prototype:

```
func (IContract) Name() string
```

Returns the smart contract name.

15.5 func Version()

Function Prototype:

```
func (IContract) Version() string
```

Returns the smart contract version number.

15.6 func CodeHash()

Function Prototype:

```
func (IContract) CodeHash() types.Hash
```

Returns the hash corresponding to the smart contract code.

15.7 func EffectHeigt()

Function Prototype:

```
func (IContract) EffectHeight() int64
```

Returns the block height at which the smart contract begins to take effect.

15.8 func LoseHeight()

Function Prototype:

```
func (IContract) LoseHeight() int64
```

Returns the block height at which the smart contract begins to fail, with `0` indicating no failure.

15.9 func KeyPrefix()

Function Prototype:

```
func (IContract) KeyPrefix() string
```

Returns the prefix of the state database KEY value (in the format `/xxx/yyy`) that can be accessed in the smart contract, which can be used to separate the data between smart contracts.

15.10 func Methods()

Function Prototype:

```
func (IContract) Methods() []std.Method
```

Returns the details of all public methods of the smart contract (can be used to call the contract for cascading messages). The `std.Method` structure is defined as follows:

```
package std

type Method struct {
    MethodID    string    // method id
    Gas         int64     // gas required to call the method
    ProtoType   string    // method prototype
}
```

15.11 func Interfaces()

Function Prototype:

```
func (IContract) Interfaces() []std.Method
```

Returns the details of all public interfaces of the smart contract that can be used to make cross-contract calls.

15.12 func Mine()

Function Prototype:

```
func (IContract) Mine() []std.Method
```

Returns the details of the mining interface exposed by the smart contract.

15.13 func Token()

Function Prototype:

```
func (IContract) Token() types.Address
```

Returns the token address registered by the smart contract. If the contract has not registered a token, it returns an empty address.

15.14 func OrgID()

Function Prototype:

```
func (IContract) OrgID() string
```

Returns the organization ID of the smart contract.

15.15 func SetOwner()

Function Prototype:

```
func (IContract) SetOwner( newOwner types.Address)
```

Sets the new owner of the smart contract. `newOwner` is the external account address of the new owner of the smart contract. The function `SetOwner()` can only be called by the original owner of the smart contract. If the smart contract issues a token, the operation will also set the token owner to the new owner.

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically triggered a `panic`:

```
1. The message caller is not the original owner of the contract;
2. The parameter newOwner is not an address;
3. The parameter newOwner is the original owner of the contract;
4. The parameter newOwner is the address of the contract;
5. The parameter newOwner is the account address of the contract.
```

16 sdk/IToken

The interface `sdk/IToken` encapsulates all the information of a token and the interface for operating the token.

16.1 func Address()

Function Prototype:

```
func (IToken) Address() types.Address
```

Returns the address of the token.

16.2 func Owner()

Function Prototype:

```
func (IToken) Owner() IAccount
```

Returns the external account object of the token owner.

16.3 func Name()

Function Prototype:

```
func (IToken) Name() string
```

Returns the name of the token.

16.4 func Symbol()

Function Prototype:

```
func (IToken) Symbol() string
```

Returns the symbol of the token.

16.5 func TotalSupply()

Function Prototype:

```
func (IToken) TotalSupply() Number
```

Returns the total supply of the token (unit: `cong`).

16.6 func AddSupplyEnabled()

Function Prototype:

```
func (IToken) AddSupplyEnabled() bool
```

Return whether additional issuance of tokens is allowed.

16.7 func BurnEnabled()

Function Prototype:

```
func (IToken) BurnEnabled() bool
```

Return whether burned of tokens is allowed.

16.8 func GasPrice()

Function Prototype:

```
func (IToken) GasPrice() int64
```

Returns the gas price of the token (unit: `cong`).

16.9 func SetTotalSupply()

Function Prototype:

```
func (IToken) SetTotalSupply(newTotalSupply bn.Number)
```

Sets the new total supply of tokens. `newTotalSupply` is the new total supply of tokens (unit: `cong`), and updates the balance of the contract owner on the token sub-account.

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically trigger a `panic`:

```
1. The message caller is not the owner of the contract;
2. The parameter newTotalSupply is less than 1000000000(cong);
3. The parameter newTotalSupply is greater than the original supply, but additional
   issuance of tokens is not allowed;
4. The parameter newTotalSupply is less than the original supply, but the tokens are not
   allowed to burn;
5. The balance of the contract owner on the token sub-account will be less than 0 after
   the update.
```

16.10 func SetGasPrice()

Function Prototype:

```
func (IToken) SetGasPrice(newGasPrice int64)
```

Sets the gas price of the token. `newGasPrice` is the new gas price for the token (unit: `cong`).

An exception occurs when any of the following is satisfied. When an exception occurs, the SDK will automatically trigger a panic:

```
1. The message caller is not the owner of the contract;
2. The parameter newGasPrice is greater than 1000000000(cong);
3. The parameter newGasPrice is less than the base gas price.
```

17 sdk/IHelper

The interface `sdk/IHelper` encapsulates all the help objects of the SDK.

17.1 func AccountHelper()

Function Prototype:

```
func (IHelper) AccountHelper() IAccountHelper
```

Returns the account helper object of the account.

17.2 func BlockchainHelper()

Function Prototype:

```
func (IHelper) BlockChainHelper() IBlockChainHelper
```

Returns the helper object related to the blockchain.

17.3 func BuildHelper()

Function Prototype:

```
func (IHelper) BuildHelper() IBuildHelper
```

Returns the helper object related to the contract building.

17.4 func ContractHelper()

Function Prototype:

```
func (IHelper) ContractHelper() IContractHelper
```

Returns the helper object related to the smart contract.

17.5 func GenesisHelper()

Function Prototype:

```
func (IHelper) GenesisHelper() IGenesisHelper
```

Return to the helper object related to genesis of blockchain.

17.6 func ReceiptHelper()

Function Prototype:

```
func (IHelper) ReceiptHelper() IReceiptHelper
```

Returns the helper object related to the receipt.

17.7 func TokenHelper()

Function Prototype:

```
func (IHelper) TokenHelper() ITokenHelper
```

Returns the helper object related to the token.

17.8 func StateHelper()

Function Prototype:

```
func (IHelper) StateHelper() IStateHelper
```

Returns the helper object related to the state storage.

18 sdk/IAccountHelper

The interface `sdk/IAccountHelper` encapsulates help objects for accessing accounts on blockchain .

18.1 func AccountOf()

Function Prototype:

```
func (IAccountHelper) AccountOf(addr types.Address) IAccount
```

Uses the address of the account to generate an account object, for the purpose of performing some operations on the account. `Addr` is the account address. Returns the created account object.

An error occurs when any of the following is satisfied, and `nil` will be returned:

```
1. The parameter addr is not an address.
```

18.2 func AccountOfPubKey()

Function Prototype:

```
func (IAccountHelper) AccountOfPubKey(pubkey types.PubKey) IAccount
```

Uses the public key of the account to generate an account object, for the purpose of performing some operations on the account. `pubkey` is the account's public key. Returns the created account object.

An error occurs when any of the following is satisfied, and `nil` will returned:

```
1. The parameter pubkey is not a public key (the length is not equal to 32 bytes).
```

19 sdk/IBlockChainHelper

The interface `sdk/IBlockChainHelper` encapsulates help objects for accessing the blockchain.

19.1 func CalcAccountFromPubkey()

Function Prototype:

```
func (IBlockchainHelper) CalcAccountFromPubkey(pubkey types.PubKey) types.Address
```

Calculates and returns the account address using the public key. `pubkey` is the public key.

An error occurs if any of the following is satisfied, and an empty address will be returned:

```
1. The parameter pubkey is not a public key (the length is not equal to 32 bytes).
```

19.2 func CalcAccountFromName()

Function Prototype:

```
func (IBlockchainHelper) CalcAccountFromName(name string, orgID string) types.Address
```

Calculates and returns account address of the contract using the contract name and its organization ID. `name` is the contract name and `orgID` is the organization ID.

19.3 func CalcContractAddress()

Function Prototype:

```
func (IBlockchainHelper) CalcContractAddress(
                                name string,
                                version string,
                                orgID string) types.Address
```

Calculates and returns the contract address based on the given contract parameters. `name` is the contract name. `version` is the contract version. `orgID` is the organization ID to which the contract belongs.

19.4 func CalcOrgID()

Function Prototype:

```
func CalcOrgID(name string) string
```

Calculates and returns the organization ID is calculated based on the organization name. `name` is the organization name.

19.5 func CheckAddress()

Function Prototype:

```
func (IBlockchainHelper) CheckAddress(addr types.Address) error
```

Validates the address format. `addr` is the address. Returns `nil` if the validation is successful.

19.6 func GetBlock()

Function Prototype:

```
func (IBlockChainHelper) GetBlock(height int64) IBlock
```

Returns the block information of the height specified. `height` is the specified block height. If the input height is less than or equal to 0, `nil` is returned. If there is no block information for the specified height, or the block information is abnormal, a generic block information object with all other parameters as null is returned.

20 sdk/IBuildHelper

The interface `sdk/IBuildHelper` encapsulates help objects for building smart contracts.

20.1 func Build()

Function Prototype:

```
func (IBuildHelper) Build(meta std.ContractMeta) std.BuildResult
```

Build a contract.

`meta` is the contract metadata entered, and the `std.ContractMeta` structure is defined as follows:

```
package std

type ContractMeta struct {
    Name         string        //Contract name
    ContractAddr types.Address //Contract address
    OrgID        string        //The organization ID to which the contract belongs
    Version      string        //Contract version
    EffectHeight int64         //Block height when the contract begins to take effect
    LoseHeight   int64         //Height at which the block fails (Fixed at 0)
    CodeData     []byte        //Contract code compression package data
    CodeHash     []byte        //Contract code hash
    CodeDevSig   []byte        //Contract developer's signature data for the contract
                               //code compression package
    CodeOrgSig   []byte        //The signature data of the organization's signature on
                               //the contract developer
}
```

Returns the details of the build results. The `std.BuildResult` structure is defined as follows:

```
package std

type Method struct {
    MethodID     string        //Method ID, method ID calculation is based on method
                               //prototype
    Gas          int64         //Gas to be burnt when calling the method
    ProtoType    string        //Method prototype
}

type BuildResult struct {
    Code         uint32
    Error        string
    Methods      []Method      //Public method list
    Interfaces   []Method      //Public interface list
    Mine         []Method      //Public mining interface
```

```
    OrgCodeHash  []byte        //All valid contract codes in the organization (program
                               //names after compilation)
}
```

Any of the below will cause a failure and return `BuildResult.Code != types.CodeOK (200)`:

```
1. The current contract is not a contract management contract;
2. The organization to which the current contract belongs is not a genesis organization;
3. The contract compilation process failed.
```

21 sdk/IContractHelper

The interface `sdk/IContractHelper` encapsulates help objects for accessing smart contracts.

21.1 func ContractOfAddress()

Function Prototype:

```
func (IContractHelper) ContractOfAddress(addr types.Address) IContract
```

Constructs a contract object based on the contract address and reads in the contract information, with the purpose of performing some operations on the contract. `addr` is the contract address. Returns the contract object.

An error occurs when any of the following is satisfied, and `nil` will be returned:

```
1. The parameter addr is not an address;
2. The parameter addr is not a contract.
```

21.2 func ContractOfToken()

Function Prototype:

```
func (IContractHelper) ContractOfToken(tokenAddr types.Address) IContract
```

Constructs a contract object based on the token address and reads in the current effective version of the contract information, with the purpose of performing some operations on the contract. `tokenAddr` is the token address. Returns the contract object.

An error occurs when any of the following is satisfied, and `nil` will be returned:

```
1. The parameter tokenAddr is not an address;
2. The parameter tokenAddr is not a token.
```

21.3 func ContractOfName()

Function Prototype:

```
func (IContractHelper) ContractOfName(name string) IContract
```

Constructs a contract object based on the contract name and reads in the contract information with the purpose of performing some operations on the contract. `name` is the contract name. Returns the contract object.

An error occurs when any of the following is satisfied, and `nil` will be returned:

```
1. The parameter name is an empty string;
2. The parameter name is not a contract.
```

22 sdk/IReceiptHelper

The interface `sdk/IReceiptHelper` encapsulates help objects for processing receipts.

22.1 func Emit()

Function Prototype:

```
func (IReceiptHelper) Emit(interface Interface{})
```

Sends a receipt. The underlying implementation of the SDK will automatically serialize the incoming receipt object as a member of the output receipt that is set for this call contract. `interface` is the receipt object.

The SDK helper will automatically call the `Emit()` function with the relevant code generated by the smart contract definition, so it need not be called manually. An example is as follows:

```
//@:public:receipt
type receipt interface {
    emitTransferMyCoin(token, from, to types.Address, value bn.Number)
}

func (mc *Mycoin) Transfer(to types.Address, value bn.Number) {
    .
    .
    .
    mc.emitTransferMyCoin(
      mc.sdk.Message().Contract().Address(),
      sender,
      receiver,
      value,
    )
}
```

23 sdk/IGenesisHelper

The interface `sdk/IGenesisHelper` encapsulates helper objects for accessing information at the time of creation.

23.1 func ChainID()

Function Prototype:

```
func (IGenesisHelper) ChainID() string
```

Return to the block chain ID specified at the time of genesis。

23.2 func OrgID()

Function Prototype:

```
func (IGenesisHelper) OrgID() string
```

Returns the genesis organization ID specified at the time of genesis.

23.3 func Contracts()

Function Prototype:

```
func (IGenesisHelper) Contracts() []IContract
```

Return to the list of smart contract objects deployed at the time of genesis.

23.4 func Token()

Function Prototype:

```
func (IGenesisHelper) Token() IToken
```

Returns the genesis Token object specified at the time of genesis.

24 sdk/ITokenHelper

The interface `sdk/ITokenHelper` encapsulates helper objects for accessing BRC20 standard tokens.

24.1 func RegisterToken()

Function Prototype:

```
func (ITokenHelper) RegisterToken(name string,
                                  symbol string,
                                  totalSupply Number,
                                  addSupplyEnabled bool,
                                  burnEnabled bool) IToken
```

Registers a BRC20 standard token on the BCBChain blockchain. `name` is the token name, `symbol` is the token symbol, `totalSupply` is the total supply (unit: `cong`), `addSupplyEnabled` is whether the additional supply is allowed, and `burnEnabled` is whether the burn is allowed. The registration successfully returns the corresponding token object.

If any of the following is satisfied, the registration fails and returns `nil`:

```
1. The initiator of the message call is not the owner of the contract;
2. The length of the parameter name is not in the range of [3,40];
3. The token corresponding to the parameter name already exists;
4. The length of the parameter symbol is not in the range of [3,20];
5. The token corresponding to the parameter symbol already exists;
6. The parameter totalSupply is less than 1000000000(cong);
7. The contract has previously registered a BRC20 standard token;
8. The current contract does not define a standard transfer method or standard transfer
   interface.
```

24.2 func Token()

Function Prototype:

```
func (ITokenHelper) Token() IToken
```

Returns the token information object that is registered in the current contract, for the purpose of performing some operations on the token.

An error occurs when any of the following is satisfied and `nil` will be returned:

```
1. There is no registered token in the current contract.
```

24.3 func TokenOfAddress()

Function Prototype:

```
func (ITokenHelper) TokenOfAddress(tokenAddr types.Address) IToken
```

Returns the token object that is fetched based on the token address provided, for the purpose of performing some operations on the token. `tokenAddr` is the token address.

An error occurs when any of the following is satisfied and `nil` will be returned:

```
1. The parameter tokenAddr is not an address;
2. The parameter tokenAddr is not a token.
```

24.4 func TokenOfName()

Function Prototype:

```
func (ITokenHelper) TokenOfName(name string) IToken
```

Returns the token object that is fetched based on the token name provided, for the purpose of performing some operations on the token. `name` is the name of the token.

An error occurs when any of the following is satisfied and `nil` will be returned:

```
1. The parameter name is not the name of a token.
```

24.5 func TokenOfSymbol()

Function Prototype:

```
func (ITokenHelper) TokenOfSymbol(symbol string) IToken
```

Returns the token object that is fetched based on the token symbol provided, for the purpose of performing some operations on the token. `symbol` is the token symbol.

An error occurs when any of the following is satisfied and `nil` will be returned:

```
1. The parameter symbol is not a token of a token.
```

24.6 func TokenOfContract()

Function Prototype:

```
func (ITokenHelper) TokenOfContract(contractAddr types.Address) IToken
```

Returns the token object that is fetched based on the contract address provided, for the purpose of performing some operations on the token. `contractAddr` is the contract address.

An error occurs when any of the following is satisfied and nil will be returned:

```
1. The parameter contractAddr is not an address;
2. The parameter contractAddr is not a contract;
3. The contract corresponding to the contractAddr has no registered tokens.
```

24.7 func BaseGasPrice()

Function Prototype:

```
func (ITokenHelper) BaseGasPrice() int64
```

Returns the base gas price (unit: `cong`).

25 sdk/IStateHelper

The interface `sdk/IStateHelper` encapsulates help objects that access the state storage.

25.1 func Check()

Function Prototype:

```
func (IStateHelper) Check(key string) bool
```

Validates if the provided KEY value exists within the permissible scope of the smart contract. `key` is the KEY value.

25.2 func Get()

Function Prototype:

```
func (IStateHelper) Get(key string, defaultData Interface{}) Interface{}
func (IStateHelper) GetEx(key string, defaultData Interface{}) Interface{}
```

Fetches data from the permissible scope of the smart contract using the provided KEY value and returns it. `key` is the KEY value, and `defaultData` is a template or default for the data type of the storage object. If the data does not exist, `Get()` returns nil, and `GetEx()` returns the default value.

Here are some simple read functions for the underlying type package:

```
func (IStateHelper) GetInt(key string) int
func (IStateHelper) GetInt8(key string) int8
func (IStateHelper) GetInt16(key string) int16
func (IStateHelper) GetInt32(key string) int32
func (IStateHelper) GetInt64(key string) int64
func (IStateHelper) GetUint(key string) uint
func (IStateHelper) GetUint8(key string) uint8
func (IStateHelper) GetUint16(key string) uint16
func (IStateHelper) GetUint32(key string) uint32
func (IStateHelper) GetUint64(key string) uint64
func (IStateHelper) GetByte(key string) byte
func (IStateHelper) GetBool(key string) bool
func (IStateHelper) GetString(key string) string

func (IStateHelper) GetInts(key string) []int
func (IStateHelper) GetInt8s(key string) []int8
func (IStateHelper) GetInt16s(key string) []int16
func (IStateHelper) GetInt32s(key string) []int32
func (IStateHelper) GetInt64s(key string) []int64
```

```
func (IStateHelper) GetUints(key string) []uint
func (IStateHelper) GetUint8s(key string) []uint8
func (IStateHelper) GetUint16s(key string) []uint16
func (IStateHelper) GetUint32s(key string) []uint32
func (IStateHelper) GetUint64s(key string) []uint64
func (IStateHelper) GetBytes(key string) []byte
func (IStateHelper) GetBools(key string) []bool
func (IStateHelper) GetStrings(key string) []string
```

If the above function does not read the data from the state database, it directly returns the default value of the corresponding type.

25.3 func Set()

Function Prototype:

```
func (IStateHelper) Set(key string, data Interface{})
```

Sets the data object based on the KEY value allowed by the state database in the smart contract. `key` is the KEY value and data is the data object to be saved.

Here are some easy setup functions for the underlying type package:

```
func (IStateHelper) SetInt(key string, v int)
func (IStateHelper) SetInt8(key string, v int8)
func (IStateHelper) SetInt16(key string, v int16)
func (IStateHelper) SetInt32(key string, v int32)
func (IStateHelper) SetInt64(key string, v int64)
func (IStateHelper) SetUint(key string, v uint)
func (IStateHelper) SetUint8(key string, v uint8)
func (IStateHelper) SetUint16(key string, v uint16)
func (IStateHelper) SetUint32(key string, v uint32)
func (IStateHelper) SetUint64(key string, v uint64)
func (IStateHelper) SetByte(key string, v byte)
func (IStateHelper) SetBool(key string, v bool)
func (IStateHelper) SetString(key string, v string)

func (IStateHelper) SetInts( key string, v []int )
func (IStateHelper) SetInt8s( key string, v []int8 )
func (IStateHelper) SetInt16s( key string, v []int16 )
func (IStateHelper) SetInt32s( key string, v []int32 )
func (IStateHelper) SetInt64s( key string, v []int64 )
func (IStateHelper) SetUints( key string, v []uint )
func (IStateHelper) SetUint8s( key string, v []uint8 )
func (IStateHelper) SetUint16s( key string, v []uint16 )
func (IStateHelper) SetUint32s( key string, v []uint32 )
func (IStateHelper) SetUint64s( key string, v []uint64 )
func (IStateHelper) SetBytes(key string, v []byte)
func (IStateHelper) SetBools( key string, v []bool )
func (IStateHelper) SetStrings( key string, v []string )
```

Sets the data object based on the KEY value allowed by the state database in the smart contract. `key` is the KEY value and `v` is the base type data value.

25.4 func Delete()

Function Prototype:

```
func (IStateHelper) Delete(key string)
```

Delete the data corresponding to the `key` value in the state database.

25.5 func Flush()

Function Prototype:

```
func (IStateHelper) Flush()
```

The update to the state database is flushed to the database layer without committing the transaction.

25.6 func McCheck()

Function Prototype:

```
func (sh *StateHelper) McCheck(key string) bool
```

Checks if there is data specified by the KEY value within the permissible scope of the smart contract, including the cache and the database. `key` is the KEY, and value will be automatically loaded into the cache if it exists.

25.7 func McGet()

Function Prototype:

```
func (IStateHelper) McGet(key string, defaultData Interface{}) Interface{}
func (IStateHelper) McGetEx(key string, defaultData Interface{}) Interface{}
```

Given the KEY value, the data is read within the permissible scope of the smart contract, and cached in the memory. This makes it possible to directly read from the memory in the call message of the subsequent smart contract without accessing the database. `key` is the KEY value, and `defaultData` is a template or default for the data type of the storage object. If the data does not exist, `McGet()` returns `nil`, and `McGetEx()` returns the default value.

Here are some simple read functions for the underlying type package:

```
func (IStateHelper) McGetInt(key string) int
func (IStateHelper) McGetInt8(key string) int8
func (IStateHelper) McGetInt16(key string) int16
func (IStateHelper) McGetInt32(key string) int32
func (IStateHelper) McGetInt64(key string) int64
func (IStateHelper) McGetUint(key string) uint
func (IStateHelper) McGetUint8(key string) uint8
func (IStateHelper) McGetUint16(key string) uint16
func (IStateHelper) McGetUint32(key string) uint32
func (IStateHelper) McGetUint64(key string) uint64
func (IStateHelper) McGetByte(key string) byte
func (IStateHelper) McGetBool(key string) bool
func (IStateHelper) McGetString(key string) string

func (IStateHelper) McGetInts(key string) []int
func (IStateHelper) McGetInt8s(key string) []int8
func (IStateHelper) McGetInt16s(key string) []int16
func (IStateHelper) McGetInt32s(key string) []int32
func (IStateHelper) McGetInt64s(key string) []int64
func (IStateHelper) McGetUints(key string) []uint
func (IStateHelper) McGetUint8s(key string) []uint8
func (IStateHelper) McGetUint16s(key string) []uint16
func (IStateHelper) McGetUint32s(key string) []uint32
func (IStateHelper) McGetUint64s(key string) []uint64
func (IStateHelper) McGetBytes(key string) []byte
func (IStateHelper) McGetBools(key string) []bool
func (IStateHelper) McGetStrings(key string) []string
```

If the above function is unable to read the data from the state database, it will direct return the default value of the corresponding base type.

25.8 func McSet()

Function Prototype:

```
func (IStateHelper) McSet(key string, data Interface{})
```

Sets the data object based on the KEY value allowed by the state database smart contract, and updates it in the memory cache. This makes it possible to read directly from the memory in the call message of the subsequent smart contract without having to access the database. `key` is the KEY value and `data` is the data object to be saved.

Here are some easy setup functions for the underlying type package:

```
func (IStateHelper) McSetInt(key string, v int)
func (IStateHelper) McSetInt8(key string, v int8)
func (IStateHelper) McSetInt16(key string, v int16)
```

```
func (IStateHelper) McSetInt32(key string, v int32)
func (IStateHelper) McSetInt64(key string, v int64)
func (IStateHelper) McSetUint(key string, v uint)
func (IStateHelper) McSetUint8(key string, v uint8)
func (IStateHelper) McSetUint16(key string, v uint16)
func (IStateHelper) McSetUint32(key string, v uint32)
func (IStateHelper) McSetUint64(key string, v uint64)
func (IStateHelper) McSetByte(key string, v byte)
func (IStateHelper) McSetBool(key string, v bool)
func (IStateHelper) McSetString(key string, v string)

func (IStateHelper) McSetInts( key string, v []int )
func (IStateHelper) McSetInt8s( key string, v []int8 )
func (IStateHelper) McSetInt16s( key string, v []int16 )
func (IStateHelper) McSetInt32s( key string, v []int32 )
func (IStateHelper) McSetInt64s( key string, v []int64 )
func (IStateHelper) McSetUints( key string, v []uint )
func (IStateHelper) McSetUint8s( key string, v []uint8 )
func (IStateHelper) McSetUint16s( key string, v []uint16 )
func (IStateHelper) McSetUint32s( key string, v []uint32 )
func (IStateHelper) McSetUint64s( key string, v []uint64 )
func (IStateHelper) McSetBytes(key string, v []byte)
func (IStateHelper) McSetBools( key string, v []bool )
func (IStateHelper) McSetStrings( key string, v []string )
```

Sets the data object based on the KEY value allowed by the state database smart contract, and updates it in the memory cache. This makes it possible to read directly from the memory in the call message of the subsequent smart contract without having to access the database. `key` is the KEY value and `v` is the base type data value.

25.9 func McClear()

Function Prototype:

```
func (IStateHelper) McClear(key string)
```

Clears the data specified in the in-memory cache (retain the data in the state database). `key` is the KEY value.

25.10 func McDelete()

Function Prototype:

```
func (IStateHelper) McDelete(key string)
```

The data corresponding to the `key` value is deleted in the state database, and the data corresponding to the `key` value is cleared from the cache.