NULS Yellow Paper V1.1

Make Trust Simpler

NULS, A Highly Customizable Blockchain Infrastructure

hi@NULS.io

ABSTRACT

NULS is a global block chain open source community project, which provides a customizable block chain infrastructure. It consists of a micro-kernel and functional modules. With innovative ideas on reducing the burden of main block chain and decoupling events and services, modularization on all NULS underlying operational components is architected and implemented. Thus, NULS is fully capable of providing functional mechanisms to conduct smart contracts, to integrate main chain and sub-chains, to yield cross-chain consensus and to reduce costs in development and operation in business applications.

Null is nothing,

NULS is everything you want in Blockchain world!

Content

1 Preface

NULS is a global blockchain open source community project, which provides a customizable blockchain infrastructure. It consists of a micro-kernel and functional modules. With innovative ideas on reducing the burden of main blockchain and decoupling events and services, modularization on all NULS underlying operational components is architected and implemented. Thus, NULS is fully capable of providing functional mechanisms to:

- conduct smart contracts
- integrate main chain and sub-chains
- yield cross-chain consensus and to reduce costs in development and operation in business applications

This paper introduces the technical design and implementation of NULS's basic blockchain intended for engineers with a technical background. If one only wishes to understand NULS technology in general, please read the NULS white paper via the link below:

https://NULS.io/pdf/NULS_whitepaper_zh_V1.0.pdf (Chinese)

https://nuls.io/pdf/NulsWhitepaper1.3.pdf (English)

2 The Vision of NULS Technical Design

2.1 Define a set of development standards of blockchain

NULS defines a set of development standards of the blockchain, including but not limited to: modular standards, service standards, data/protocol standards, process/thread standards and complete event-driven model standards.

2.2 Simplify blockchain technology

The blockchain is an integration of various advanced technologies, such as encryption technology and p2p network, which leads to a high technical barrier. NULS re-architects the underlying implementation of the blockchain by decomposing blockchain into different modules. Detailed and direct interactions among modules are encapsulated since connections among modules are realized through the interactions between modules and event bus as well as the interactions between modules and service bus. Developers only need to focus on simple abstracted interfaces called service terminals and event data structure of each module. To enable developers from all fields to easily participate in the development of NULS ecosystem, the NULS community provides and maintains the complete development documents of all service interfaces and event data structures.

2.3 Highly customizable and highly scalable

NULS' is highly customizable and scalable by use of three subsystems: Logic Layer, Module Layer, and Cross-Chain Layer.

2.4 Ability of continuous improvement

NULS modular architecture design makes the internal implementation of each module, such as ledge module, account module, consensus module as well as smart contract module, only visible within its own module. Hence each module can be upgraded independently without interfering with other modules' implementations.

3 Overview of NULS Technology

3.1 Complete development standards

3.1.1 Unified data format

Among NULS system nodes and modules, events are employed as carriers for data transmissions. Each event has its own specific identification number which contains the identification numbers for modules and the module internal events.

3.1.2 The Standard of the module definition

The NULS system provides a complete set of module definitions. Both the built-in and extension modules should obey the NULS module definitions. The principle of decoupling definition and implementation should be followed as well during module development.

3.1.3 The standard of process and thread

All processes and threads in NULS system need to be subject to kernel management.

3.2 Modularization

3.2.1 Separation of definition and implementation

According to NULS module design, a module is classified into two parts: definition and implementation. Only the definition is published to other modules, whereas the implementation of each module remains as private. A module can be upgraded or replaced as technology advancements and new business demands require.

3.2.2 Interaction among modules

In order to achieve interactions among modules in NULS ecosystem, either actively or passively, the module event notifications and service calls are performed in event bus and service bus.

3.3 Event Driven

The NULS built-in event module defines the publish/subscribe models, handler chains, and filter chains. To process events, a developer only needs to subscribe to those events interested (that are published by other modules) and to define handler and filter in the developer's own module (for each of interested event).

3.4 Multi-encryption algorithm support

NULS supports multi-encryption algorithms in Provider mode. Encryption algorithms can be self-defined in sub-chains and expansion modules. NULS uses ECC / Hash / AES as default algorithms.

3.5 Multi-chains Parallel Operation

Multiple-chain is operated in parallel to realize functional support for various tasks and chains. Meanwhile, it would improve the system; and cross-chain consensus would enable data auditing and values flow.

4 POC consensus mechanism

4.1 Role

Name	Responsibility	Benefit
Agent	Provide hardware	Rewarded from producing block bonus
	Create agent nodes Lock deposit	Rewarded from agent commission
	Receive commission from clients	Rewarded from trading transaction fee

Client	Lock the deposit Entrust agent to produce block Pay commission to agent for producing block	Rewarded from producing block bonus Rewarded from trading transaction fee

4.2 Types of Bonus

Name	Source	Rewarded person
The reward for producing a block	5,000,000 NULS will be mined annually.	Agent, Client
Agent commission	Fixed percentage of rewards from producing block	Agent
Transaction fee	Fees of transactions in NULS chain	Agent, Client

4.3 Instruction Of Rules

All agent nodes take turns producing a block. The order of producing block in each round is randomly computed.

Redeeming the deposit when a client meets consensus in agent node. After redeeming, if the deposit in the agent node is greater than or equal to b*d%, then the agent node continues to be in consensus meeting; if the deposit in the agent node is less than b*d%, the agent node will be out of consensus meeting automatically and will adopt a waiting status for future commission.

If an agent node initializes to withdraw from the consensus meeting, then all deposit from that agent will be locked for 24 hours.

When one agent creates an agent node, he may only accept the commission from a designed account or commissions from all accounts.

Red card. If an agent node attempts to perform hostile attacking, such as double spending or forking, the red card will be issued. Consequently, the agent's deposit will be locked for 30 days; all the tokens in the agent nodes will be locked for 3 days; that guilty account will not be allowed to create any agent node in future.

Yellow card. If an agent node does not produce block within a predetermined time frame, one yellow card will be issued.

4.4 Parameters

Name	Name of Variables	Description	Default Value
Lowest limit of agent deposit	a	The minimum number, **a**, of NULS required to be deposited when agent initially creates the agent node.	a=20,000
The lowest limit of agent commissions from client	b	Only when an agent node receives client commission which is greater than **b**, can it join the consensus meeting. NULS deposited from agent himself cannot be accounted as part of it.	b=200,000
The highest limit of agent commissions from client	c	When commissions from a client in agent node in total are higher than **c**, then the agent node is no longer allowed to accept more commissions from the client. NULS deposited from agent himself is not accounted as part of it.	c=500,000
The lowest commission ratio needed for agent node to be in consensus meeting	d	When an agent node is in consensus meeting, the commission from a client may decrease. If the total commission reaches the amount which is lower than **b*d%**, then the agent node has to withdraw from consensus meeting.	d=80
The lowest limit of client deposit	e	When a client entrusts agent to participate in the consensus meeting, the lowest locked deposit from a client is **e**.	e=2,000
The ratio of agent commissions.	f%-g%	The agent sets up the highest commission ratio **g** and the lowest commission ratio **f**.	f>=0 g=20
The transaction fee in the block	fee	The total amount of transaction **fee** in the consensus data block	

The fee per transaction	tx_fee	The fee per transaction	0.01NULS
The number of agent nodes in consensus meeting in current around	Rnc	The number of agent nodes in consensus meeting in current around	
The producing block time interval between two adjacent blocks	bti	The producing block time interval between two adjacent blocks	
Total number of seconds in a year	spy	Total number of seconds in a year	
Total deposit in one agent node	cmc	The total deposit, which includes the agent deposit and client commissions, in one agent node	
Credit Ratio	cr	credit ratio	

4.5 Calculation of Consensus Reward

4.5.1 Total rewards in all nodes per round

$$coinbase = fee + 5{,}000{,}000 * \frac{rnc * bti}{spy} * \frac{cmc * \max(0, cr)}{\sum_{1}^{rnc} cmc * \max(0, cr)}$$

4.5.2 Agent node rewards (excluding commissions)

$$member\,\mathrm{Re}\,ward = (coinbase)\frac{guaranteedeposit}{cmc}$$

4.5.3 Agent received rewards

$$reward = member\,\mathrm{Re}\,ward(self) + coinbase * \frac{(cmc - guaranteedeposit)}{cmc} * Commission\ rate$$

4.5.4 Client received rewards

$$reward = member\,\mathrm{Re}\,ward(self) * (1 - Commission\ \ rate)$$

4.6 Credit Ratio

4.6.1 Parameters

Name	Name of Parameters	Description	Default Value
The number of Round to compute Credit Ratio	**R**	Only data from latest **R** round are statistically evaluated when computing the credit ratio. The calculation of credit ratio may be postponed in two rounds	R=100
The number of blocks produced at nodes in recent R round	**N**	The number of blocks produced at node in recent R round	
The number of Yellow cards	**Y**	The number of Yellow cards	
The turns when a yellow card issued in the node.	**yr**	Given RC as the current round, node receives a yellow card at **Z** round, then **yr = R – min (RC - Z, R)**	
The yellow card co-efficient	**mn**	The yellow card coefficient	Mn=4

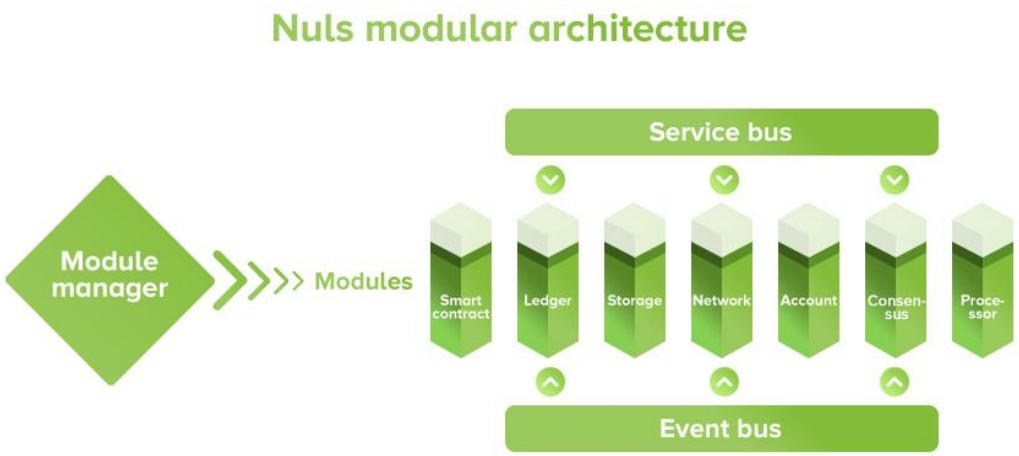
4.7 The formula to compute the credit ratio

$$cr = \frac{N}{X} - \frac{mn\sum_{1}^{Y} yr}{R * R}$$

5 Modulization Architecture

5.1 Introduction

Recognizing the unpredictable nature of ever-changing business applications and rapidly developing technology, NULS team has developed the kernel by following the Linux kernel's modular design. Based on the principle, "Everything is a Module", NULS supports new technology development and new business application with modularization (of each functional component).



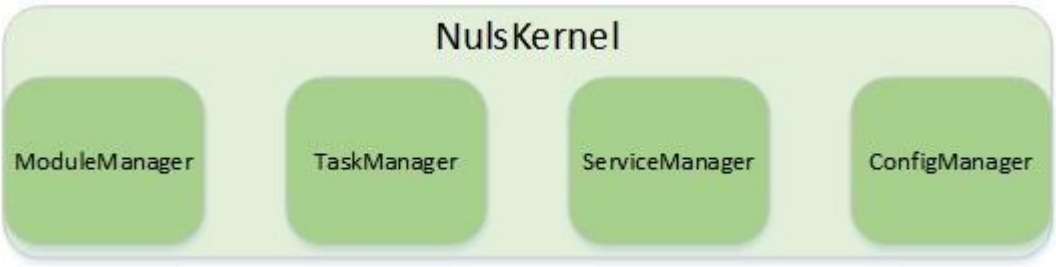
Architecture of NULS Modularization

NULS architecture consists of two major components, namely, micro-kernel and functional module.

The traditional blockchain is decomposed into multiple modules, such as smart contract, ledger, account, storage, networking, consensus, event buses, caching, etc. Each module is capable of being upgraded or replaced as future technology development or individual customers require.

Module Manager is responsible for managing modules. Interactions among modules are carried out through the service bus and event bus. Service bus is responsible for managing the service handler of each module, whereas event bus is responsible for handling the distribution and subscription of module events.

5.2 NULS Micro-Kernel

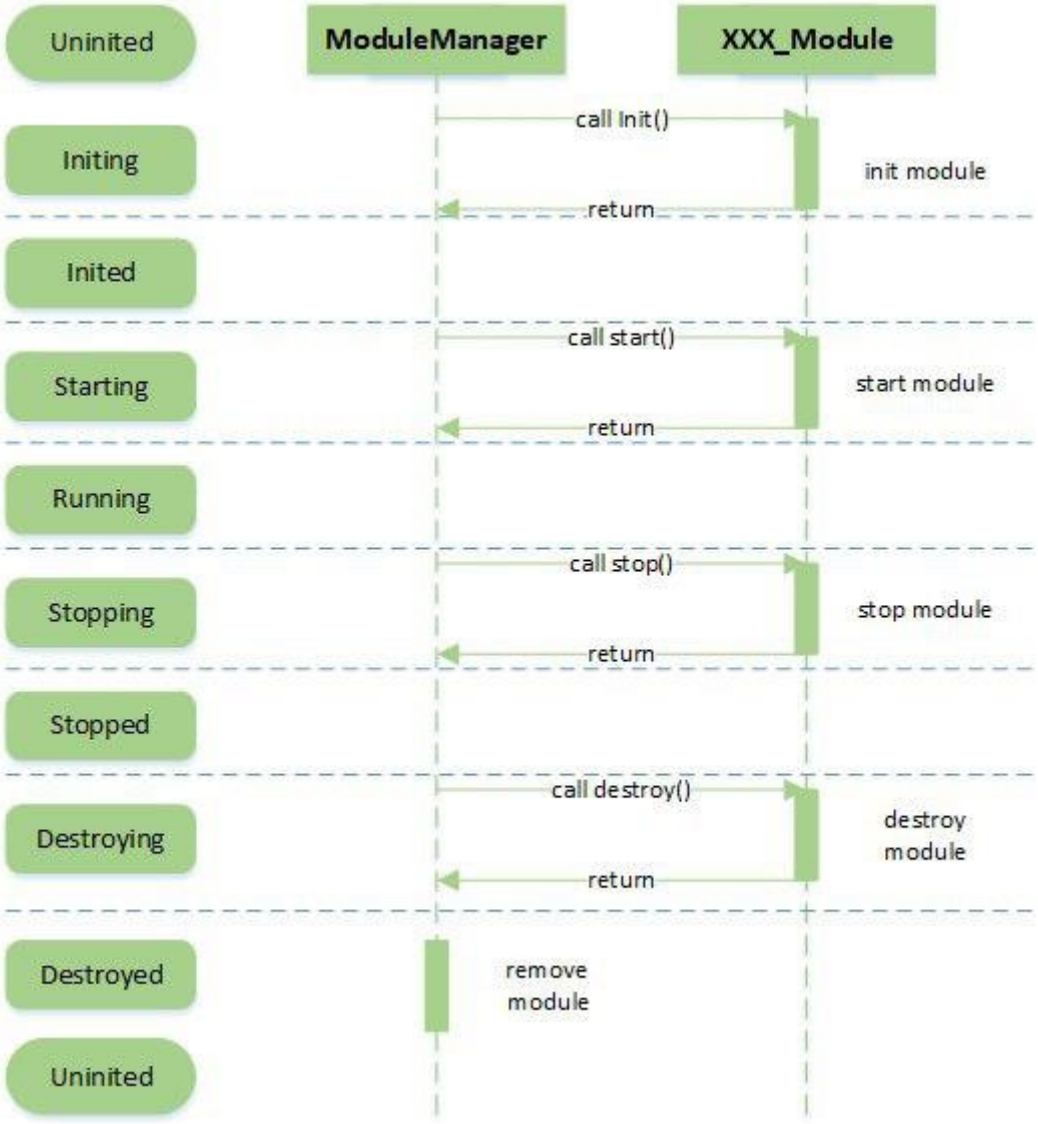


The NULS system consists of a set of decentralized nodes. Within each node, all modules of NULS kernel coordinate to operate together. The function of the NULSKernel can be categorized into four aspects: module management, task management, service management and configuration management.

5.3 ModuleManager

The Module Manager is responsible for managing the life cycle of all modules in NULS nodes.

5.3.1 The life cycle of module



The life cycle of a module includes the stages of Uninited, Initing, Inited, Starting, Running, Stopping, Stopped, Destroying, Destroyed. The name of each stage represents its functionality.

5.4 TaskManager

NULS regulates two types of tasks, NULS Process and NULS Thread. NULS kernel uses Task Manager to manage all the tasks.

5.4.1 NULS Process

NULS defines the NULS Process as Module Process, and each module has its own main process, which is automatically created when the module is created by the Module Manager. Extension module developers will only need to focus on business logic based on the module specification and definitions, thus it is unnecessary to pay attention to other details.

5.4.2 NULSThread

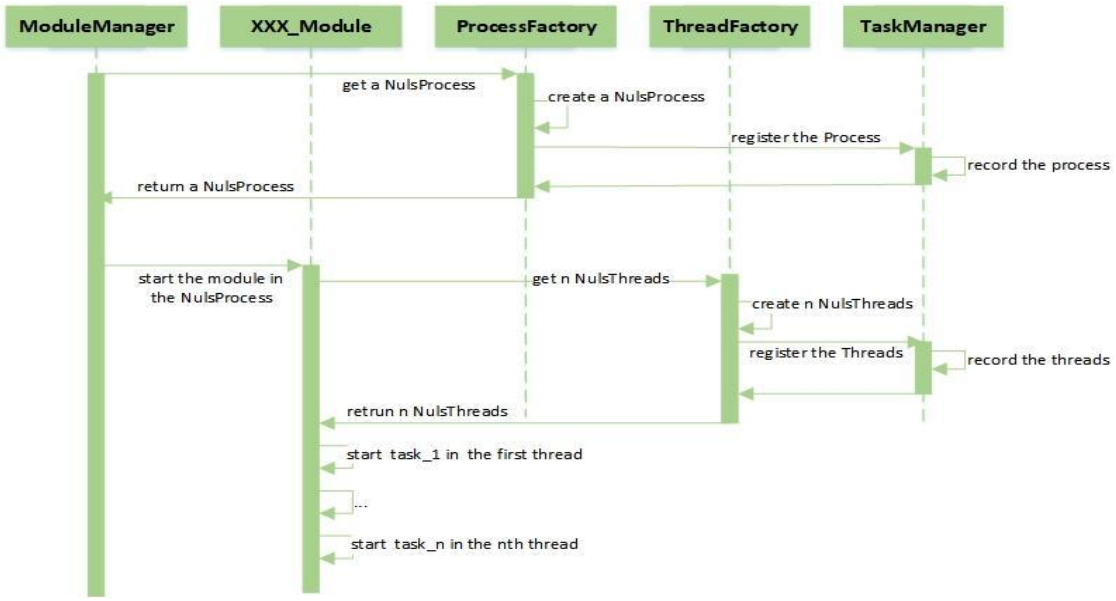
NULS defines the thread as NULS Thread. A module may perform several different tasks simultaneously. Each module will create one NULS Thread for every task. Each NULS Thread has a module identification number, thread name, and an executing handler. The NULS development regulation does not allow developers to create anonymous threads (This will be further instructed in the future development regulation. NULS community will evaluate all codes). NULS module developers must create threads as follows:

```
/**
 * @param moduleId
 * @param threadName
 * @param runnable
 */

create Single Thread and Run (short module ID, String thread Name, Runnable)
```

5.4.3 The Launch of the Process and the Thread

The following chart illustrates the creation of process and thread when one module is launched.



Module Manager creates a process using Process Factory.

Process Factory creates a process, and registers the process information into the Task Manager;

Task Manager records process information;

Process Factory returns the process back to Module Manager;

Module Manager calls the module start method to start the module;

The module needs to start *n jobs*, so that *n threads are created using* Thread Factory.

Thread Factory creates *n threads* and registers thread information to the Task Manager;

Task Manager records thread information;

Thread Factory returns *n threads* back to the module;

The module starts one job in each thread

Module starts a task in every process;

Status of module: Running.

5.5 ServiceManager

NULS system consists of various modules, and each module provides external services through its interfaces. NULS uses Service Manager (Service Manager) to manage all services. (For those developers familiar with Window OS, one can simply understand the Service Manager as a Windows DLL Registration Table. Once the DLL is registered in the registry, the developer will obtain the DLL

service handler from the registry, then, call service by referring to the DLL provider document). In the NULS system, all expansions module developers are service providers; therefore, module developers not only need to provide desired modules body but also, at the same time, need to provide detailed documentation of service interface.

Service Manager provides service registration introduction (Register Service) for module developers. When loading each module, one needs to call all service objects of the module to be registered using Register Service (Service Manager is used for the internal implementation, while the module developers actually call the registered Service interface of those Base NULS Module).

The NULS community will document all service interfaces/protocol for developers use.

5.6 NULS Module Layer

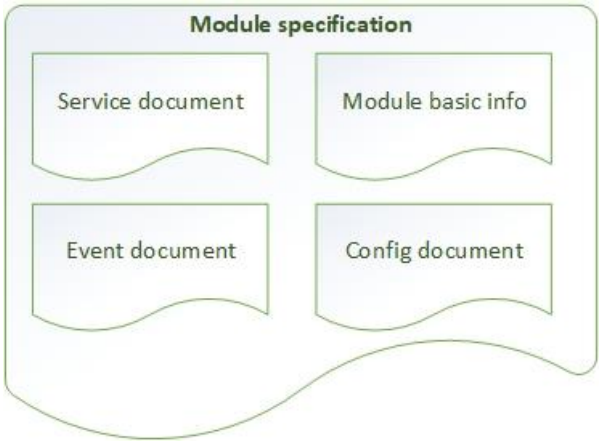
5.6.1 Definition of Module

NULS provides a standard definition of a module. A complete module definition includes four parts:

- basic information
- module services
- module events
- module configuration

5.6.2 Module specification

Module developers need to provide a module specification to define each module. The module specification must declare in details on published events, service interfaces, data structure and module configuration. In the process of modular design and abstraction, one does not need to know the underlying technology inside another module. Rather, one can make use of another module's features by simply using the service bus and event bus. Therefore, the specification of each new module is in particular important. In NULS ecosystem, the module specification serves as a community testing benchmark and technical guide for other developers who wish to employ the module. It is one of the important factors to evaluate for deciding whether the community is to approve the module to be released.



5.6.3 Basic Information

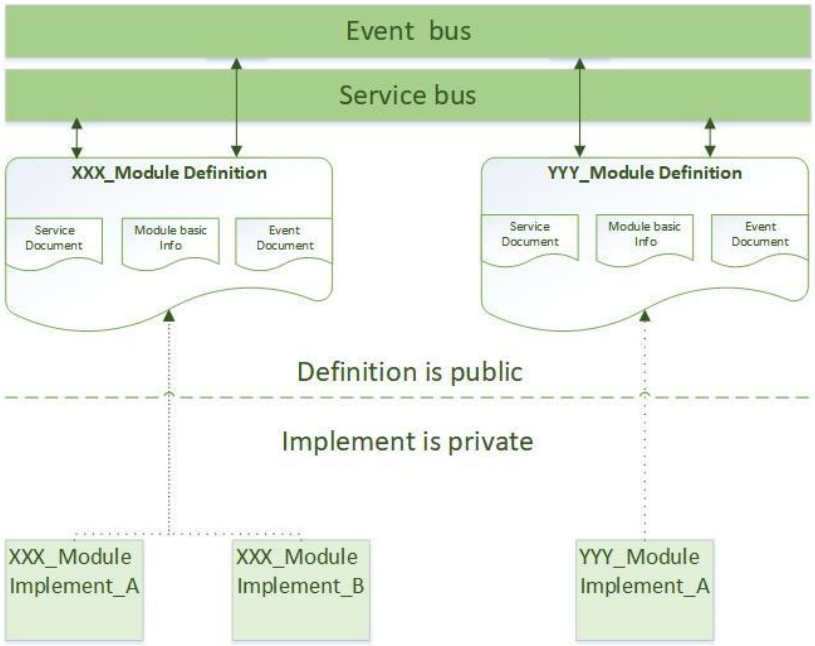
Each NULS module has its own basic information including an identification number, name, version, and description. Each module has a unique number and name in the module library and it is used in process management/thread management, event publication/event subscription, and service registration/service query.

5.6.4 Module Service

Each NULS module can define its own service as an interface, and it can also invoke services of other modules. The Service Manager manages all service objects (service handlers) and all the services registered in the module when the module starts. Therefore, another module can call the referenced module service document to perform under the guidance of business logic within its own module.

5.7 Module Implementation

By separation of module implementation and definition, the same functional module may have various ways to implement and an individual module can be upgraded and replaced independently.



5.7.1 Bootstrap

Based on the configuration file modules, properties, the default startup module is automatically loaded when each node is initialized. When in operation, the NULS node can also call Module Manager's load Module method to dynamically load the module. The loading process of each module is carried out by the module bootstrap, which is provided by NULS as a unified base class, Module Bootstrap. All module bootstraps should be developed based on this base class and managed by the kernel. Module Bootstrap is defined as follows:

5.7.1.1 Init method to initialize module

Load configuration, create a cache, create this module message queue, etc.

5.7.1.2 Start Method to initiate task

The method starts module internal tasks, registers module services in the service manager and subscribes to those events interested by this module. This process includes registration of processors and filters for each event.

5.7.1.3 Stop Method to terminate task

This method cancels the module’s subscription to events, stops the module’s internal services and stops the module’s all internal tasks.

5.7.1.4 Destroy Module

Destroy cache and message queue created within this module.

5.8 Module Configuration

In the NULS module configuration file, Each config Section represents one module, while bootstrap configures the module boot class. During module initialization, Config Loader is called to obtain the configuration information of the module.

```
[Network]

bootstrap=Network Module bootstrap

network. server. port=8632

network. external. port=8003

network. magic=936152748


[Consensus]

.....
```

5.9 NULS expansion module development process

NULS specifies the development process for expansion module, including module definition, implementation, code review and QA release. This detailed regulation will be published on the developer's official website later.

6 Event Driven

High scalability, achieved by implementing NULS modulization architecture, is capable of supporting various blockchain applications. Relying on NULS framework, developers from various industries around the world may develop their own application modules or underlying modules (For example, private blockchain developers may use direct network rather than P2P network, or suppress the block producing time by increasing unit block size in an attempt to improve performance. Whereas in terms of the financial industry, developers may replace the account modules with industry standard CA for authentication). Different modules will be developed by different developers (different industry sectors may request different specific expertise). For instance, application module developers may not be familiar with block chain’s principles and mechanism; underlying module developers may not be alien with other modules’ design and implementation (i.e. P2P network expert does not need to concern how the crystallization is achieved). Therefore, it is necessary to establish a set of simple and effective communications among modules. Event-driven communication is one of such simple and effective methods.

NULS is an event-driven system (namely, receiving a transaction of a trade or receiving an instant messaging via DApp. Different modules perform their own business logics to process different events). The NULS event bus employs the disruptor framework to publish and/or to subscribe event models. Each module may publish its own events as well as subscribe to events published by other modules.

On the basis of specific scenarios, NULS standard classifies events into two types, namely, Local Event and Network event.

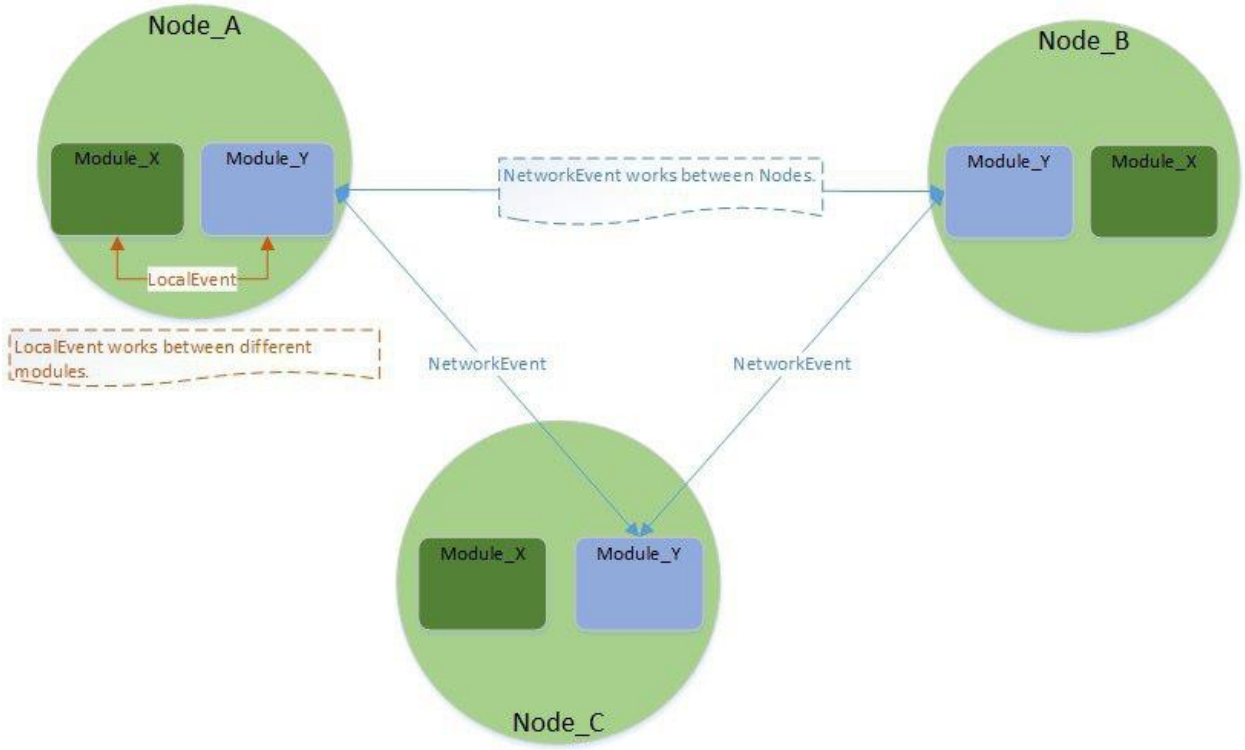
6.1 Local Event

Logical events, also known as local events are used for asynchronous communication among different modules within a node. Such events are not propagated through the network and are sent directly to the local event distributor/dispatcher to inform other modules to process their business logic.

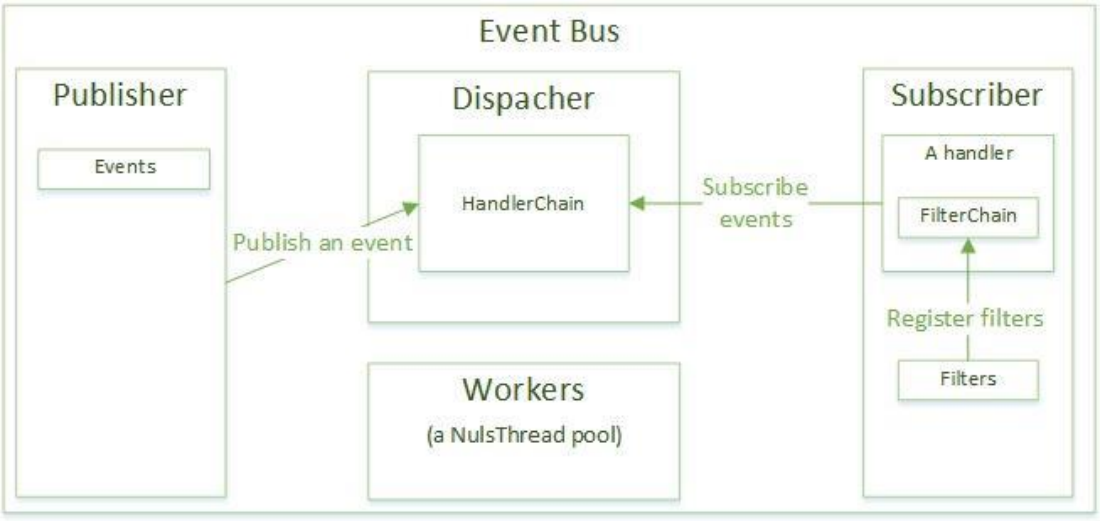
6.2 Network Event

Network events are used for asynchronous communication among different nodes. Such events are sent to other nodes in the network. The data structure is interpreted internally by the module. Each module does not need to know data structure of other modules.

The below chart demonstrates the difference between a logical event and an underlying event:



6.3 Event Bus

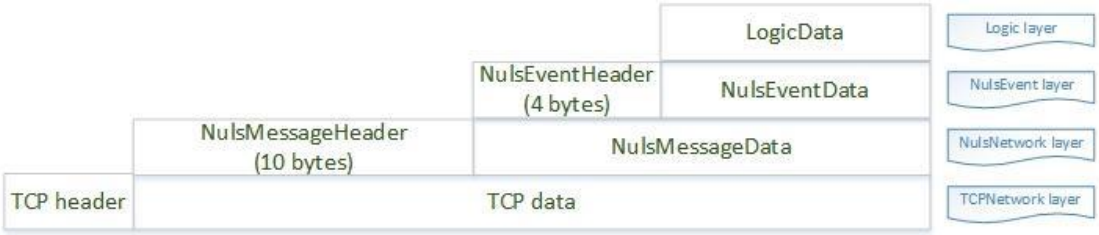


Module developers define the module by publishing module instruction manual. The instruction manual describes in detail all possible events in this module. If other module developers plan to subscribe to those events, they need to create a handler to handle that event and register the handler to the event bus as well. To improve the flexibility and scalability, we have designed a filter chain (Filter Chain). If a handler is designed only for processing partial information of an event, it can be achieved by applying the filter (Filter).

The dispatcher (Dispatcher) of NULS event bus is conducted by using disruptors. A disruptor itself works in a single-threaded mode. To improve efficiency, a disruptor is only responsible for dispatching but not for processing business logic. When an event is published, the dispatcher will dispatch each subscriber's handler into an independent thread to operate.

6.4 Data Structure

NULS events can be published and/or subscribed among nodes or among modules within one node. The data protocol stack is set up on top of the TCP protocol, which is followed from bottom to top as NULS Network Layer, NULS Event Layer and NULS Logic Layer.



6.4.1 Logic Layer

Logic Layer defines the business logic of each upper module. Module developers who extend modules only have to focus on the business logic and data of their own modules.

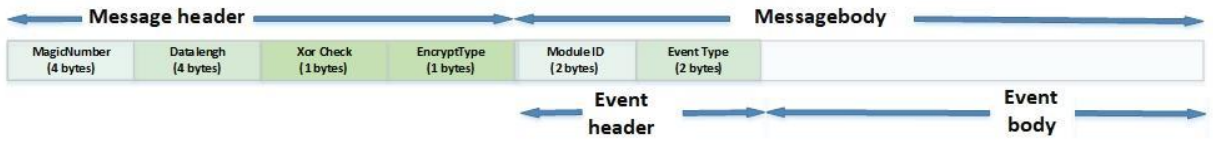
6.4.2 Event Layer

Four digits' bytes are added prior to the business data as the event header (NULS Event Header). An event header contains a module identification number and an event identification number. Each event owns a unique event header. Modules will publish and/ or subscribe an event based on the event header.

6.4.3 Network Layer

Ten-digit bytes added prior to the event layer data from the network message header (NULS Message Header). Such a header includes magic numbers, data length, an odd-even parity bit and data encryption type.

Data Structure:



	Name	Length	Description
Message Header	Magic Number	4 bytes	It is used to differentiate cross-chain network segments. Data Packets in different network segments will be filtered by network layer filters.
	Data Length	4 bytes	It stands for the data length of Message Body (from the Little Endian)
	Xor	1 byte	The odd-even parity bit for data in Message Body. Failed data will be filtered out by network layer filter
	Encrypt Type	1 byte	Encryption method to support network layer data encryption and expansion.

Event Header	Module ID	2 bytes	Module identification number
	Event Type	2 bytes	Module internal event identification number

7 Multi-chains

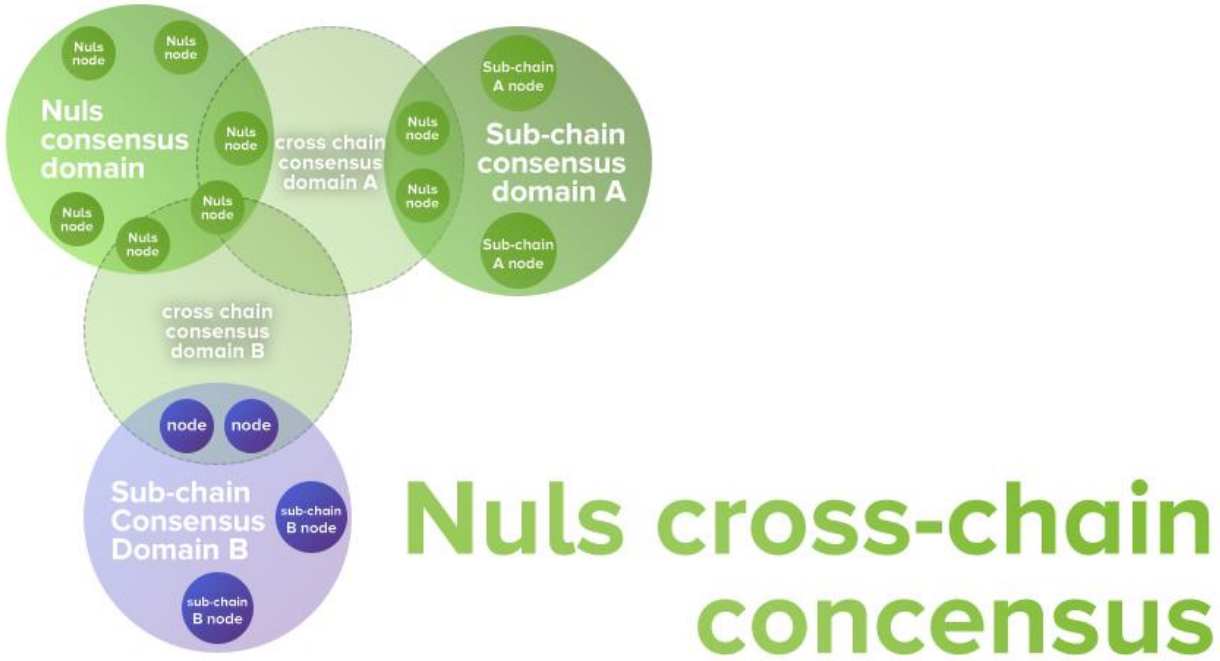
7.1 Introduction

NULS supports sub-link registration access. The main chain and sub-chain verify blocks and transactions through cross-chain consensus; NULS tokens and other tokens of the sub-chains can circulate between the NULS main chain and the sub-chain. The sub chain reports the block header to the main chain, and in return, the main chain audits the block of sub chain.

7.2 Consensus of Cross-chain

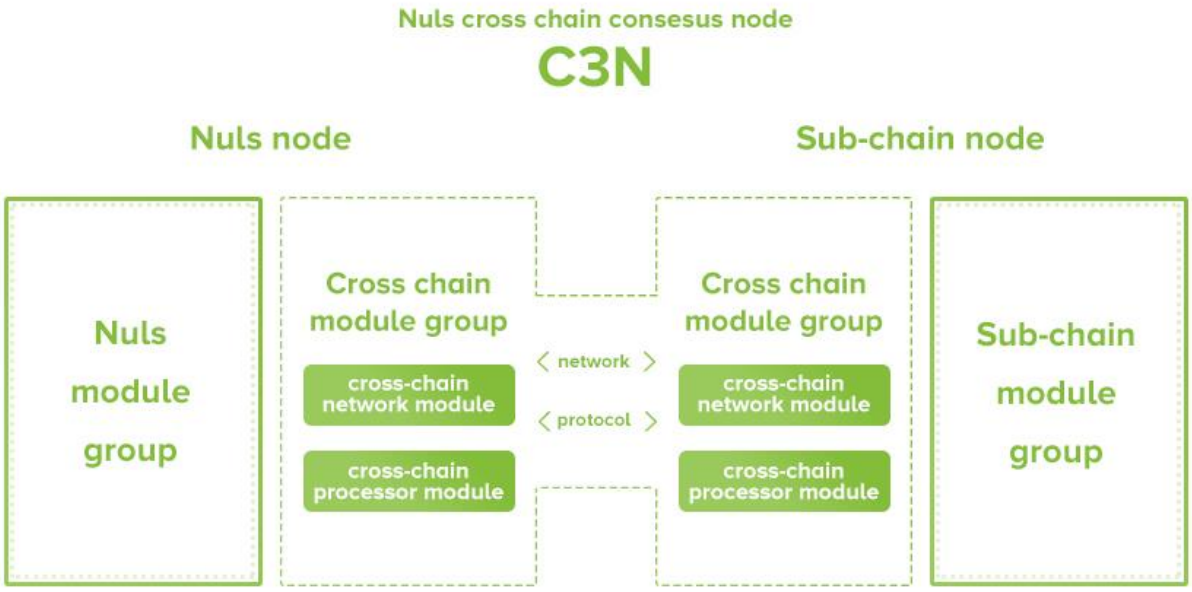
7.2.1 C3D：Cross chain consensus domain

The part nodes of the main chain and sub-chains compose the cross chain consensus domain, the node within the cross chain consensus domain agrees on the trans-chain transaction, and the trans-chain data protocol is transformed to share the cross-link data to other peer nodes.



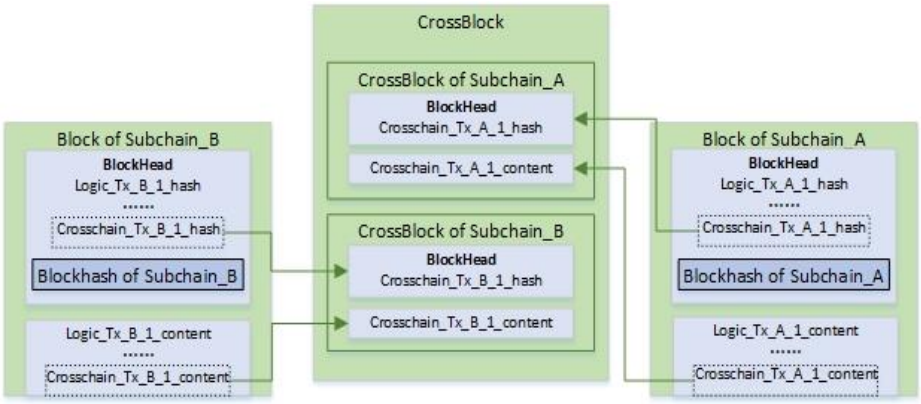
7.2.2 C3N：Cross chain consensus node

The nodes that join the cross chain consensus domain add additional load to the modules required for the cross chain consensus, such as network modules, cross-link protocol processor modules, etc.

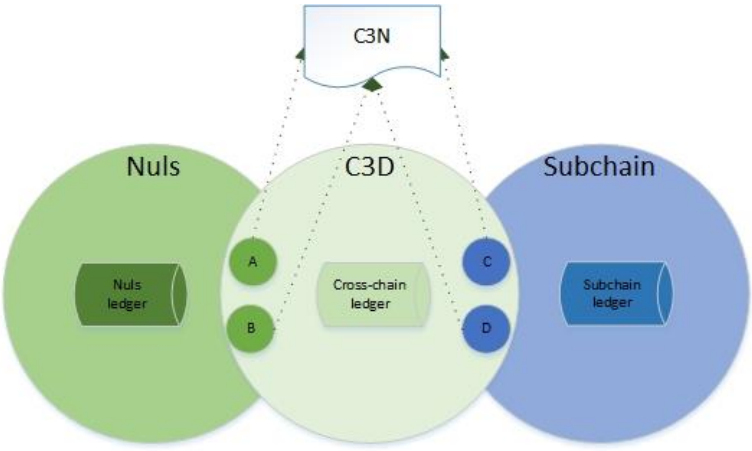


7.2.3 Cross-chain Trading ledger

The cross-chain trading ledger is run by a trans-chain consensus node that records all chain transactions across all chains. The trans-chain consensus module of the sub chain needs to implement the Protocol converter, which is responsible for the transformation of the sub chain transaction data and the trans-chain transaction data format. Cross-chain transactions are divided into two types: for the technical audit, private chain/alliance chain that requires NULS to provide data integrity, the transaction is the sub-chain blockhead; for asset transactions, the trans-chain transaction ledger stores the entire contents of the transaction.



7.2.4 Transaction Confirmation



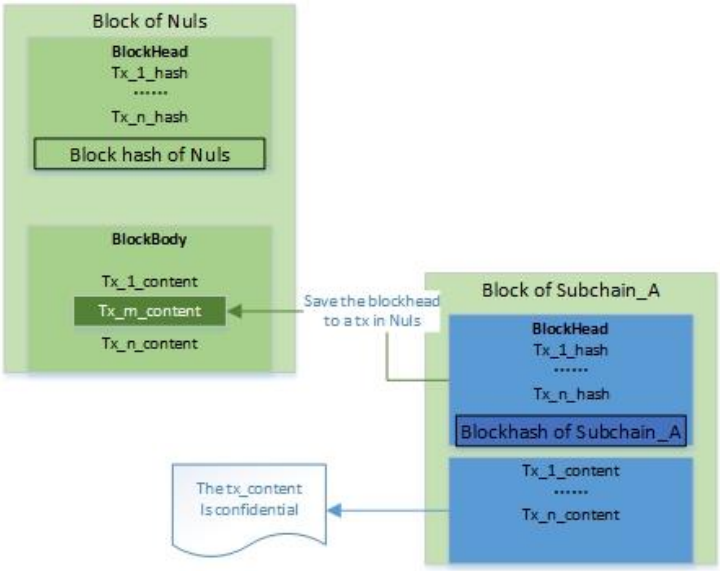
The main chain nodes A and B, and sub chain nodes C and D are composed of cross chain C3N. When a new block is generated in the chain, C and D collects the cross-chain transaction in the new block to make the cross-link block broadcast to C3N, node C, and node D to authenticate the integrity of all the data in the block. When node A and node B receive a cross-linked block, the node is validated to the nodes other than the broadcaster (for example, when a node receives a broadcast of node D, validate data to node C.)

7.3 Private chain data audit

For organizations, data confidentiality and security are extremely important, so the openness and transparency of the blockchain is a concern for the organization. Through data isolation and trans-chain audit, NULS ensures the business data confidentiality and security of the sub chain, and solves the balance between data transparency and business confidentiality.

NULS provides a set of programs to audit the integrity of the private chain data where the sub-chain encapsulates each block to a transaction in the main chain, and the main chain uses the multi-level digest technology to audit the transaction data integrity of the sub chain, which not only ensures the confidentiality of the enterprise, but also provides the enterprise with the technical means to prove the integrity of the data to the public.

Through the chain of consensus, the private chain reports to the main chain of the Blockhead. By retaining a complete data digest for the sub chain, the main chain provides the sub-main with the service of integrity self-proving. Therefore, records of each sub-chain block header need to charge a certain number of tokens as a service fee.



8 NULS Built-In Modules

Name	Variables	Identification number	bootstrap
database	db	2	MybatisDBModuleImpl
cache	cache	3	EhCacheModuleImpl
network	network	4	NetworkModuleImpl
account	account	5	AccountModuleImpl
event bus	event-bus	6	EventBusModuleImpl
consensus	consensus	7	PocConsensusModuleImpl
ledger	ledger	8	UtxoLedgerModuleImpl
RPC Services	rpc	9	RpcServerModuleImpl
Smart contract	Smart-contract	10	NULSVMImpl

8.1 Database Module

8.1.1 Basic information

Name	db	Identification number	2
Description	The data persistence is provided and maintained in the db module through service interfaces for all business modules.		

8.1.2 Services

Name of Service	Data Service
Function	Basic data operations: create, read, update and delete (CRUD)
interface name	**function**
save	Save data
update	Update data
get	Query data according to (the data) logo or mark
delete	Delete data
getList	Query the filtered data list according to conditions
getCount	Get the count number of queried data according to conditions

8.2 Cache module

8.2.1 Basic information

Name	cache	**Identification number**	3
Description	Provide fundamental caching functionality for the entire NULS system, including functions to create a cache and to manipulate cached data by CRUD. Each module can define its cache based on its own need.		

8.2.2 Service

Service name	Cache Service
functionality	Create cache and manipulate cached data by CRUD
interface	**function**
createCache	Create cache
removeCache	Remove cache
clearCache	Clear cache
getCacheTitleList	Obtain the list of all cache names

putElement	Save data in cache
getElement	Obtain data from cache
getElementList	Obtain the list of cached data
removeElement	Remove data from cache
containsKey	Whether or not contain a cache key
keySet	Obtain all the keys in one cache

8.3 Network Module

8.3.1 Basic information

name	network	**Identification number**	4
description	Provide network communication services for NULS block chains, including P2P nodes management, send the network data to the event bus by unpacking it into an event, and encapsulate the event into a network message and send it to the network.		

8.3.2 Service

service name	NetworkService
function	Node management, node group management and message sending.
interface name	**function**
addNode	Add designated nodes
removeNode	Remove designated nodes
addNodeToGroup	Add node into the designated node group
removeNodeFromGroup	Remove node from the node group
addGroup	Add node group
removeGroup	Remove node group
sendToAllNode	Send message to all nodes
sendToNode	Send message to designated node

sendToGroup	Send message to designated node group

8.3.3 Event

event	Identification number	description of functionality
GetVersionEvent	0x0100	Query version information from peer node
VersionEvent	0x0200	Return the version information of the node
PingEvent	0x0300	Ping event
PongEvent	0x0400	Pong event
ByeEvent	0x0500	Proactively disconnect notification
GetNodeEvent	0x0600	Ask the connected peer node for additional nodes.
NodeEvent	0x0700	Return a list of known peer nodes.

8.4 Account Module

8.4.1 Basic information

name	account	**Identification number**	5
description	NULS Block Chain Account System. NULS' account can be shared by the main chain and sub-chains. Both main chain and sub-chains can create accounts. Account module provides the basic operations of an account, creating, exporting, importing, password setting, etc.		

8.4.2 Service

Service name	AccountService
function	Create accounts, export, import, set passwords, set aliases, signature transaction, etc.
interface name	**functions**
createAccount	Create accounts (in batches)
getAccount	Obtain local designated account
getAccountList	Obtain the list of local designated accounts
getAddress	Obtain the address based on public key

getPrivateKey	Obtain the private key based on address
getDefaultAccount	Obtains the local default account
setDefaultAccount	Set up the default (account) address
encryptAccount	Encrypt the account.
changePassword	Change password (for the account)
isEncrypted	whether an account is encrypted.
unlockAccounts	Unlock (locked) accounts.
signData	Signature
verifySign	Verify signature.
setAlias	Sets alias.
exportAccount	Export an account.
importAccount	Import an account.

8.5 Event Bus Module

8.5.1 Basic information

Name	event-bus	**Identification number**	6
Description	NULS is driven by events. All data transmission among nodes as well as communication data among all modules in a node are encapsulated within the events. The event-bus module is responsible for sending, receiving, filtering, distributing, publishing/subscribing events.		

8.5.2 Service

Service NAME	EventBusService
Function	Responsible for receiving, filtering, publishing/subscribing events, etc.
Interface name	Function
subscribeEvent	Subscribe to events, register handlers.
unsubscribeEvent	Cancel event subscriptions, unregister handlers
publishEvent	Release events

8.5.3 Event

Event	Identification number	Description of functionality
CommonDigestEvent	0x0100	Common Digest Event: Employed to broadcast event digest. After event digests being received in nodes, the decision will be made on whether to pull the complete event.
GetEventBodyEvent	0x0200	Event-data-obtaining event: obtain the full data of the entire event.

8.6 Consensus Module

8.6.1 Basic information

Name	consensus	Identification number	7
Description	The consensus module is responsible for maintaining the operation of consensus for nodes, including block packing, block verification, consensus rewards calculation, consensus rewards distribution, consensus status maintenance, transaction fee calculations, and ill-behaved nodes punishment		

8.6.2 Service

Service name	ConsensusService
Function	Provide consensus related operations, be responsible for node's consensus status maintenance.
Interface name	**Function**
getTxFee	Calculate transaction fees.
startConsensus	Start consensus.
stopConsensus	Stop consensus.
getConsensusAccountList	Get the list of consensus accounts.
getConsensusInfo	Get consensus information.

Service name	BlockService

Function	All operations related to a block.
Interface name	**Function**
getGenesisBlock	Get the genesis block (the fist block)
getLocalHeight	Get the height of the local block.
getLocalBestBlock	Get the best (or newest) local block
getBlockHeader	Get the block header
getBlock	Get a complete block.
saveBlock	Save a block
rollbackBlock	Rollback a block.

8.6.3 Event

Event	Identification number	Description of Functionality
GetSmallBlockEvent	0x0100	Get-small-block event: Get the small block from the peer node (including all the transaction digests).
SmallBlockEvent	0x0200	Small block event: the bearer event when a small block is sent.
GetBlockEvent	0x0300	Get-the-block event: get a block from the peer node.
BlockEvent	0x0400	Complete-block event: the bearer event used to send entire block.
GetBlockHeaderEvent	0x0500	Get-block-header event: get a block header from peer node.
BlockHeaderEvent	0x0600	Block header event: the bearer event used to send the block header
GetTxGroupEvent	0x0700	Get-transaction-group event: get multiple transactions from peer nodes.
TxGroupEvent	0x0800	Transaction group events: the bearer events used to send multiple transactions.

8.7 Ledger Module

8.7.1 Basic information

Name	ledger	**Identification number**	8
Description	Ledger module is the accounting book for the transactions of all blocks. It not only provides the operation vehicles for all the transactions but also defines all the basic operations (such as CRUD) related to NULS tokens. NULS token transactions in other modules are processed through the ledger module.		

8.7.2 Service

Service Name	Ledger Service
function	Transaction cache, transaction query, transaction verification, balance inquiry, etc.
Interface	**Function**
verifyTx	Verify transaction
getTx	Query transaction
Get Balance	Get the balance of relevant address.
Query TxList by Account	Query the transactions listed by designed account.
Query TxList by Hashs	Query the transaction list according to hash list.

8.7.3 Event

Event	**Identification number**	**Description of functionality**
Transaction Event	0x0100	Transaction Event: Broadcasts all transactions.