# Plugins

warning This article requires a revision. Nethermind plugins is a powerful way of extending your local node capabilities.

(see also an article here: https://medium.com/nethermind-eth/writing-your-first-nethermind-plugin-a9e04d81cf59 )

Plugins that you can write:

Plugin Type What can it be used for? RPC Adding additional RPC modules to the client that have full access to the internal Nethermind APIs and can extend capabilities of the node when integrating with your infrastructure / systems. Block Tree Visitors Code allowing you to analyze entire block tree from genesis to the head block and execute aggregated calculations and checks. Devp2p Allows you to create additional devp2p network protocol for your nodes to communicate over TCP/IP. You can also build custom products that will run attached to Nethermind nodes. State Visitors Allow you to run aggregated analysis on the entire raw format state (or just some accounts storages). Config You can add additional configuration categories to our config files and then use them in env variables, json files or command line to configure behaviour of your plugins. TxPool TxPool behaviours and listeners. Tracers Custom, powerful EVM tracers capable of extracting elements of EVM execution in real time. CLI Additional modules for Nethermind CLI that can allow you build some quick scratchpad style JavaScript based behaviors. Note: When writing a plugin be carefull about exceptions you throw. Especially if you are hooking up event handlers on some core objects like BlockProcessor or BlockTree. Those exceptions can bring the node down. This is by design. Its responsibility of plugin writer to correctly handle those exceptions.

How to build a plugin? We included an example inside the Nethermind.Analytics plugin:

## RPC Plugin example:

[ RpcModule ( ModuleType . Clique ) ] public

interface

IAnalyticsModule

:

IModule { [ JsonRpcMethod ( Description =

"Retrieves ETH supply counted from state." , IsImplemented =

true ) ] ResultWrapper < UInt256

analytics_verifySupply ( ) ;

[ JsonRpcMethod ( Description =

"Retrieves ETH supply counted from rewards." , IsImplemented =

true ) ] ResultWrapper < UInt256

analytics_verifyRewards ( ) ; }

## CLI Plugin example:

[ CliModule ( "analytics" ) ] public

class

AnalyticsCliModule

:

CliModuleBase { [ CliFunction ( "analytics" ,

"verifySupply" ) ] public

string

VerifySupply ( ) { return NodeManager . Post < string

( "analytics_verifySupply" ) . Result ; }

[ CliFunction ( "analytics" ,

"verifyRewards" ) ] public

string

VerifyRewards ( ) { return NodeManager . Post < string

( "analytics_verifyRewards" ) . Result ; }

public

AnalyticsCliModule ( ICliEngine cliEngine ,

INodeManager nodeManager ) :

base ( cliEngine , nodeManager )

{

} }

## Block Tree Visitor Plugin example:

public

class

RewardsVerifier

:

IBlockTreeVisitor { private

ILogger _logger ; public

bool PreventsAcceptingNewBlocks =>

true ; public

long StartLevelInclusive =>

0 ; public

long EndLevelExclusive {

get ;

}

private

UInt256 _genesisAllocations = UInt256 . Parse ( "720099904994800000000000000" ) ; private

UInt256 _uncles ;

public

UInt256 BlockRewards {

get ;

private

set ;

}

public

RewardsVerifier ( ILogManager logManager ,

long endLevelExclusive ) { _logger = logManager . GetClassLogger ( ) ; EndLevelExclusive = endLevelExclusive ; BlockRewards = _genesisAllocations ; }

private

RewardCalculator _rewardCalculator =

new

RewardCalculator ( MainnetSpecProvider . Instance ) ;

public

Task < BlockVisitOutcome

VisitBlock ( Block block ,

CancellationToken cancellationToken ) { BlockReward [ ] rewards = _rewardCalculator . CalculateRewards ( block ) ; for

( int i =

0 ; i < rewards . Length ; i ++ ) { if

( rewards [ i ] . RewardType == BlockRewardType . Uncle ) { _uncles += rewards [ i ] . Value ; } else { BlockRewards += rewards [ i ] . Value ; } }

_logger . Info ( "Visiting block { block . Number } , total supply is (genesis + miner rewards + uncle rewards) | { _genesisAllocations } + { BlockRewards } + { _uncles } " ) ; return Task . FromResult ( BlockVisitOutcome . None ) ; }

public

Task < LevelVisitOutcome

VisitLevelStart ( ChainLevelInfo chainLevelInfo ,

CancellationToken cancellationToken ) => Task . FromResult ( LevelVisitOutcome . None ) ;

public

Task < bool

VisitMissing ( Keccak hash ,

CancellationToken cancellationToken ) => Task . FromResult ( true ) ;

public

Task < HeaderVisitOutcome

VisitHeader ( BlockHeader header ,

CancellationToken cancellationToken ) => Task . FromResult ( HeaderVisitOutcome . None ) ;

public

Task < LevelVisitOutcome

VisitLevelEnd ( CancellationToken cancellationToken ) => Task . FromResult ( LevelVisitOutcome . None ) ; }

## Config plugin example:

public

class

AnalyticsConfig

:

IAnalyticsConfig { public

bool PluginsEnabled {

get ;

set ;

} public

bool StreamTransactions {

get ;

set ;

} public

bool StreamBlocks {

get ;

set ;

} public

bool LogPublishedData {

get ;

set ;

} }

## State Tree Visitor example:

public

class

SupplyVerifier

:

ITreeVisitor { private

readonly

ILogger _logger ; private

HashSet < Keccak

# _ignoreThisOne

new

HashSet < Keccak

( ) ; private

int _accountsVisited ; private

int _nodesVisited ;

public

SupplyVerifier ( ILogger logger ) { _logger = logger ; }

public

UInt256 Balance {

get ;

set ;

}

= UInt256 . Zero ;

public

bool

ShouldVisit ( Keccak nextNode ) { if

( _ignoreThisOne . Count

16 ) { _logger . Warn ( "Ignore count leak -> { _ignoreThisOne . Count } " ) ; }

if

( _ignoreThisOne . Contains ( nextNode ) ) { _ignoreThisOne . Remove ( nextNode ) ; return

false ; }

return

true ; }

public

void

VisitTree ( Keccak rootHash ,

TrieVisitContext trieVisitContext ) { }

public

void

VisitMissingNode ( Keccak nodeHash ,

TrieVisitContext trieVisitContext ) { _logger . Warn ( "Missing node { nodeHash } " ) ; }

public

void

VisitBranch ( TrieNode node ,

TrieVisitContext trieVisitContext ) { _logger . Info ( "Balance after visiting { _accountsVisited } accounts and { _nodesVisited } nodes: { Balance } " ) ; _nodesVisited ++ ;

if

( trieVisitContext . IsStorage ) { for

( int i =

0 ; i <

16 ; i ++ ) { Keccak childHash = node . GetChildHash ( i ) ; if

( childHash !=

null ) { _ignoreThisOne . Add ( childHash ) ; } } } }

public

void

VisitExtension ( TrieNode node ,

TrieVisitContext trieVisitContext ) { _nodesVisited ++ ; if

( trieVisitContext . IsStorage ) { _ignoreThisOne . Add ( node . GetChildHash ( 0 ) ) ; } }

public

void

VisitLeaf ( TrieNode node ,

TrieVisitContext trieVisitContext ,

byte [ ]

value

=

null ) { _nodesVisited ++ ;

if

( trieVisitContext . IsStorage ) { return ; }

AccountDecoder accountDecoder =

new

AccountDecoder ( ) ; Account account = accountDecoder . Decode ( node . Value . AsRlpStream ( ) ) ; Balance += account . Balance ; _accountsVisited ++ ;

_logger . Info ( "Balance after visiting { _accountsVisited } accounts and { _nodesVisited } nodes: { Balance } " ) ; }

public

void

VisitCode ( Keccak codeHash ,

TrieVisitContext trieVisitContext ) { _nodesVisited ++ ; } }Edit this page Last updatedonFeb 17, 2024 Previous Building from source Next Custom analytic tools