Technical Mechanics — System Overview

The following section describes the technical architecture and operational mechanics of the GENESIScoin ecosystem. It provides a structured overview of the smart contract modules and automation infrastructure that together form the protocol.

GENESIScoin is built as a modular on-chain system composed of multiple specialized smart contracts responsible for token economics, token distribution, reward mechanisms, treasury management, and operational allocations. Each module operates independently but interacts with others through clearly defined on-chain events and function calls.

The protocol architecture is divided into several core layers:

• **Token Core** — defines the fundamental economic mechanics of the GENc token, including taxation, liquidity support, burn mechanics, and reward funding.
• **PreSale Module** — manages the initial token distribution process, stage progression, whitelist management, and the transition to the post-sale ecosystem.
• **Public Sale Module** — operates the post-sale reward infrastructure, including dividend cycles, bonus programs, and holder participation mechanisms.
• **Asset Manager Contracts (AM v1 and AM v2)** — control treasury vesting, operational allocations, and ecosystem funding through deterministic release logic and multisignature governance.
• **Automation Infrastructure** — external services that monitor on-chain conditions and trigger predefined contract functions without holding protocol state or controlling token balances.

All critical protocol logic and state remain stored directly on-chain within the smart contracts. External automation services act only as execution agents that trigger transactions when predefined on-chain conditions are satisfied.

This architecture ensures deterministic behavior, transparent execution, and full traceability of protocol operations through on-chain events and state verification.

Module 1 — Token Core (GENESIScoin)

The Token Core defines the base economic mechanics of GENESIScoin.
It contains the internal mechanisms responsible for transaction taxation, swap execution, supply reduction, liquidity support, and reward funding.

Core Mechanisms

Burn Mechanism
Permanent removal of GENc tokens from circulation to reduce the total supply.

Automated Tax Routing
A transaction tax is applied to transfers and automatically routed to the contract’s internal tax pool.

Tax Accumulation
Collected tax tokens accumulate inside the contract until the swap threshold is reached.

Threshold Swap
When the accumulated tax exceeds the defined threshold, the contract triggers a swap.

Swap Execution
Accumulated GENc tokens are swapped through the DEX router into BNB.

Liquidity Injection
Part of the swap proceeds is used to add liquidity to the GENc–BNB pair.

BNB Rewards Source
BNB generated from the swap funds the holder reward system.

Token Core Components
• Burn Mechanism
• Automated Tax Routing
• Tax Accumulation
• Threshold Swap
• Swap Execution
• Liquidity Injection
• BNB Rewards Source

The Token Core forms the economic base layer of GENESIScoin.

PreSale Module

The PreSale Module manages the controlled distribution of GENESIScoin during the initial sale phase.
It defines stage progression, pricing, bonus allocation, whitelist registration, and the transition to post-sale reward mechanisms.

Core Mechanisms

Stage-Based Token Distribution
The pre-sale is divided into sequential stages.

Each stage defines:
• token allocation
• token price
• bonus percentage
• stage duration

Stages progress automatically based on token sales or time limits.

Dynamic Stage Control
Each stage contains parameters that control availability and progression:
• tokens available
• tokens sold
• stage start timestamp
• stage duration
• stage active state

Stage transitions occur when:
• token allocation is sold out
• stage duration expires

Oracle Price Update
The module receives external price updates used to determine the BNB equivalent of token pricing. The Oracle periodically updates the BNB price used in the sale calculation.

Whitelist Registration
During the final stage of the pre-sale, a whitelist window opens allowing users to register for participation in the B100D reward program.

Whitelist lifecycle:

1. whitelist open
2. registration period
3. whitelist close

B100D Program Initialization
After the pre-sale ends:

1. the whitelist closes
2. the B100D program is activated
3. daily reward distribution begins

The system triggers the start of the B100D distribution schedule automatically through the Oracle layer.

Daily B100D Distribution
The module manages the execution of daily reward distributions for B100D participants.

Each cycle:
• increments the program day counter
• triggers reward distribution
• verifies distribution completion

PreSale Funds Distribution
During the pre-sale period, the contract accumulates BNB from purchases.
Funds are distributed periodically according to the internal allocation rules.

Public Sale Transition
After the pre-sale process is completed:
• the whitelist closes
• the B100D program starts
• the system triggers the start of the Public Sale phase

PreSale Module Components
• Stage-Based Token Distribution
• Dynamic Stage Control
• Oracle Price Update
• Whitelist Registration
• B100D Program Initialization
• Daily B100D Distribution

• PreSale Funds Distribution
• Public Sale Transition

Public Sale (GENc)

The Public Sale module manages the post-sale reward infrastructure of GENESIScoin.
The contract does not sell tokens.
Its role is to manage GENc reward pools and automated distribution cycles for token holders.

Core Mechanisms

Public Sale Activation
The post-sale phase begins when the system triggers:
startPublicSale()
The start timestamp is recorded on-chain and used to determine reward phases and daily reward cycles.

Dividend Pool (GENc)
The contract maintains an internal GENc dividend pool.
dividendPoolGENc()
This pool is the source of all dividend payouts to eligible holders.

Snapshot-Based Holder Verification
Holder eligibility is determined through snapshot cycles.

During a cycle:

1. a snapshot of holders is created
2. each holder’s balance becomes the reference balance
3. balances are continuously monitored

If a holder reduces their balance below the recorded amount, they are removed from the cycle.

Dividend Cycle System
Dividend distribution operates in repeating cycles.

Each cycle:

1. creates a holder snapshot
2. monitors balances during the cycle
3. calculates payouts
4. distributes rewards from the GENc pool

Cycle timing is randomized within a predefined window to prevent predictable execution patterns.

Progressive Reward Model
Rewards increase with continuous holding.
For each completed cycle, the holder’s reward percentage increases.

Example progression:
2% → 3% → 4% → ... → max level

The system tracks:
• entry cycle
• holding progress
• reward level progression

Batch Distribution
Rewards are distributed in batch transactions to reduce gas consumption.
distributeDividendsAdjusted(address[], amount[])
Each transaction distributes rewards to multiple holders at once.

Daily Purchase Bonus System
In addition to dividend cycles, the system includes daily purchase rewards.

When tokens are purchased from the liquidity pool:
• the transaction is detected
• minimum purchase size is verified
• a daily bonus reward can be issued

processDailyBonus(address, day)
Bonuses are limited per day and depend on the current reward phase.

Holder Tracking System
The system continuously tracks token holders by monitoring Transfer events.

New holders are added only when:
• tokens are purchased from the LP
• minimum balance requirements are met

Existing holders are updated or removed when balances change.

Public Sale Module Components
• Public Sale Activation
• GENc Dividend Pool
• Snapshot Holder Verification
• Dividend Cycle System
• Progressive Reward Model
• Batch Distribution Engine
• Daily Purchase Bonus System
• Holder Tracking System

Module 4 — Asset Manager v1
(Treasury Vesting Controller)

Asset Manager v1 manages long-term ecosystem treasury allocations.
The contract enforces time-based vesting schedules and releases GENc tokens in deterministic stages.
All releases are executed directly on-chain and triggered by the automation infrastructure.

Core Mechanisms

Vesting Pools
AM v1 manages multiple treasury pools identified by pool IDs.

Each pool represents a dedicated ecosystem allocation such as:
• Development reserve
• Marketing reserve
• Team allocation
• Events / ecosystem promotion
• Future project development
• SBA ecosystem reserve

Each pool follows its own release schedule and release amount configuration.

Waste Start (Vesting Activation)
Treasury vesting begins when the system records the waste start timestamp.
setWasteStart(uint64 ts)
This timestamp becomes the reference point for all vesting schedules.
The timestamp is triggered automatically when the PreSale whitelist closes.

Release Advisory System
Before a release can occur, the system evaluates whether the pool is eligible for distribution.
adviseRelease(uint8 id)

The function returns:
• due → whether the release is currently allowed
• index → vesting step number
• dueTs → release timestamp
• amount → GENc amount scheduled for release

This function is used by automation services to verify when tokens can be released.

Release Execution
When a pool becomes eligible, the system executes:
release(uint8 id)

This transfers the scheduled GENc amount from the treasury pool to the destination wallet.
Each call releases only one vesting step.

Release Model
AM v1 operates as a step-based vesting system.

For each pool:

1. a vesting start timestamp is defined
2. a release schedule is configured
3. each release step becomes eligible at a defined timestamp
4. automation triggers the release call

Only one step is executed per release transaction.

Automation Layer
The vesting process is executed by the generalAM automation bot.

The bot performs the following logic:

1. waits for the WhitelistClosed event from PreSale
2. calls setWasteStart() if vesting has not been initialized
3. continuously polls vesting pools
4. calls adviseRelease(id) to check eligibility
5. triggers release(id) when a step becomes due

This architecture ensures that treasury releases occur only when on-chain conditions are satisfied.

Execution Model
The AM v1 system operates with the following structure:
PreSale Event
↓
WhitelistClosed
↓
setWasteStart()
↓
Vesting timer begins
↓
adviseRelease(poolID)
↓
release(poolID)

Key Properties
• fully deterministic vesting logic
• step-based release execution
• automation-triggered releases
• pool-based treasury allocation control
• on-chain verification of release eligibility

Module 5 — Asset Manager v2
(Operational Allocation Controller)

Asset Manager v2 manages conditional GENc distributions across four dedicated operational pools. The contract separates bot-executed stage releases from multisig-governed milestone, partner, and contest allocations.
All balances, release state, and execution conditions are stored on-chain.

Core Roles

Owner
The owner funds pools and proposes multisig operations.

GENERAL Bot
The GENERAL address can execute stage pool releases only.

MSIG Committee
A 2-of-4 multisignature committee approves milestone, partner, and contest operations before execution.

Managed Pools

Asset Manager v2 manages four pools:

Pool 1 — Stage Airdrop Pool
Bot-controlled release to the airdrop wallet.

Pool 2 — Milestone Airdrop Pool
Multisig-approved release to the airdrop wallet.

Pool 3 — Partners Pool
Multisig-approved release to the partners wallet with a 240-hour cliff.

Pool 4 — Contest Airdrop Pool
Multisig-approved release to the airdrop wallet with a 72-hour cliff.

Pool Caps

Two pools have hard funding caps:

Stage Pool Cap
1.5B GENc

Milestone Pool Cap
1.0B GENc

Partners and Contest pools do not use fixed hardcoded cap constants in the same way, but they are still tracked through on-chain pool balances.

On-Chain Accounting

The contract tracks pool state through:
• funded[poolId]
• released[poolId]
• poolBal[poolId]

It also tracks one-time execution state through:
• releasedStage[stageId]
• releasedMilestone[milestoneId]

This prevents duplicate releases.

Stage Pool Logic

The Stage Airdrop Pool is controlled only by the GENERAL bot.

Stage Release
The bot executes:
releaseStage(stageId, metaHash)

Allowed stage range:
• Stage 1
• Stage 2
• Stage 3
• Stage 4
• Stage 5

Release amounts are fixed:
• Stage 1 → 100M GENc
• Stage 2 → 200M GENc
• Stage 3 → 300M GENc
• Stage 4 → 400M GENc
• Stage 5 → 500M GENc

A successful stage release sends GENc to the AIRDROP_WALLET.

Failed Stage Release
For stages 2, 3, and 4 only, the bot may execute:
releaseStageFail(stageId, metaHash)

The stage amount is split as follows:
• 50% → BURN_WALLET
• 50% → DIVIDEND_WALLET

This path is only available for failed stages 2–4.

Milestone Pool Logic

The Milestone Airdrop Pool is multisig-controlled.

Valid Milestones
Allowed milestone IDs:
• 100
• 250
• 500
• 1000
• 1500

Fixed Milestone Amounts
• 100 → 20M GENc
• 250 → 60M GENc
• 500 → 140M GENc
• 1000 → 300M GENc
• 1500 → 480M GENc

Execution Flow
Milestone release follows this sequence:

1. owner calls proposeMilestone(milestoneId, metaHash)
2. two distinct MSIG signers approve
3. anyone may call execute(opId)
4. GENc is sent to the AIRDROP_WALLET

Each milestone can be released only once.

Partners Pool Logic

The Partners Pool is multisig-controlled and protected by a 240-hour cliff.

Execution Flow

1. owner funds pool 3
2. owner proposes arm operation through:
   proposePartnersArm(amount, metaHash)
3. two MSIG signers approve
4. execute(opId) arms the release
5. the amount is moved out of active pool balance and stored as partnersArmed
6. the contract starts a 240-hour cliff
7. after cliff expiry, owner proposes execution through:
   proposePartnersExecute(metaHash)
8. two MSIG signers approve again
9. execute(opId) transfers GENc to the PARTNERS_WALLET

Cancel Path
Before execution, the owner may propose cancellation through:
proposePartnersCancel(metaHash)

After approvals and execution, the armed amount returns to the pool balance.

Contest Pool Logic

The Contest Airdrop Pool is multisig-controlled and protected by a 72-hour cliff.

Execution Flow

1. owner funds pool 4
2. owner proposes arm operation through:
   proposeContestArm(amount, metaHash)
3. two MSIG signers approve
4. execute(opId) arms the release
5. the amount is stored as contestArmed
6. the contract starts a 72-hour cliff
7. after cliff expiry, owner proposes execution through:
   proposeContestExecute(metaHash)
8. two MSIG signers approve again
9. execute(opId) transfers GENc to the AIRDROP_WALLET

Cancel Path
Before execution, the owner may propose cancellation through:
proposeContestCancel(metaHash)

After approvals and execution, the armed amount is restored to the pool balance.

Multisig Operation Model

All multisig-controlled flows use the same structure:

1. owner proposes operation
2. two distinct MSIG addresses approve
3. anyone may execute once the approval count reaches 2

The contract tracks approvals with:
• approvalsCount
• approvalsMask

This prevents duplicate approvals by the same signer.

Cliff Model

AM v2 contains two hardcoded cliff durations:

Partners Cliff
240 hours

Contest Cliff
72 hours

These cliffs begin only after the corresponding ARM operation has been executed.
Execution before cliff expiry is blocked.

Event Proof Layer

The contract emits a full proof trail for funding, arming, release, cancellation, and multisig governance.

Key events include:
• PoolFunded
• PoolReleased
• StageReleased
• StageFailedSplit
• MilestoneReleased
• Armed
• Executed
• Cancelled
• Proposed
• Approved
• OpExecuted

This provides on-chain traceability for every AM v2 action.

Key Properties
• four isolated operational pools
• bot-only stage release path
• multisig-controlled milestone releases
• multisig + cliff-controlled partner releases
• multisig + cliff-controlled contest releases

• fixed stage and milestone amounts
• one-time release protection for stages and milestones
• full on-chain accounting and event traceability

Automation Layer

AM v2 is operated by the v2GENERAL automation service.

The automation bot performs the following cycle:

1. reads presale stage state
2. checks whether a stage allocation has already been released
3. evaluates stage success or expiration conditions
4. calls the appropriate release function

The system runs continuously in scheduled intervals and executes one stage operation per cycle.

Execution Flow
PreSale Stage Progress
↓
Stage Result Evaluation
↓
releaseStage(stageId)
OR
releaseStageFail(stageId)
↓
releasedStage(stageId) marked true

Key Properties
• conditional release logic tied to presale outcomes
• stage-level allocation control
• on-chain verification preventing duplicate execution
• fail-safe stage expiration handling
• automation-driven execution infrastructure

Automation Infrastructure
(Bot Execution Layer)

The GENESIS ecosystem uses an external automation layer responsible for triggering predefined smart-contract functions and monitoring on-chain conditions.
Automation services do not store protocol state and do not control token balances.
All critical protocol state remains inside the smart contracts.
Bots act exclusively as transaction executors when predefined on-chain conditions are satisfied.

Oracle Bot

The Oracle Bot manages scheduled protocol actions during and after the PreSale phase.

Core responsibilities
• updates the BNB price used in the PreSale module via updateBNBPrice()
• closes expired presale stages via closeStageByOracle(stageId)

• opens the B100D whitelist via triggerWhitelistOpen()
• closes the whitelist via triggerWhitelistClose()
• starts the B100D program via triggerB100DStart()
• triggers daily B100D rewards via triggerDailyB100DPayout()
• distributes PreSale funds via distributeFunds()
• activates the Public Sale phase via startPublicSale()

The Oracle bot ensures that time-dependent protocol operations execute automatically without manual intervention.

Asset Manager v1 Bot — generalAM.js

The AM v1 bot manages treasury vesting execution.

Core responsibilities
• listens for the WhitelistClosed event emitted by the PreSale module
• initializes treasury vesting via setWasteStart(ts)
• checks pool eligibility using adviseRelease(id)
• executes vesting releases using release(id)

Each execution releases a single vesting tranche from the configured treasury pool.

Asset Manager v2 Bot — v2GENERAL.js

The AM v2 bot manages stage-based operational allocations tied to PreSale outcomes.

Core responsibilities
• reads stage parameters from stages(stageId)
• verifies stage release status via releasedStage(stageId)
• executes releaseStage(stageId, metaHash) for successful stages
• executes releaseStageFail(stageId, metaHash) when a stage expires below threshold
• performs post-presale sweep logic to resolve remaining stages

Decision model
A stage is considered successful when:
tokensSold > 50% of stage supply

If the stage duration expires below this threshold, the fail path is executed.

Dividend Tracker — divT.js

The Dividend Tracker maintains the dataset of eligible dividend holders.

Core responsibilities
• monitors GENc Transfer events
• tracks holders above the minimum balance threshold
• records holders during PreSale from TokensPurchased events
• switches to post-sale tracking after WhitelistClosed
• adds new holders only when tokens are purchased from the liquidity pool
• removes holders whose balance drops below the required threshold

The resulting dataset is used by the dividend verification system.

Dividend Verifier — divV.js

The Dividend Verifier manages dividend cycle execution and payout verification.

Core responsibilities
• starts dividend cycles after PreSale completion
• applies an initial cooldown before the first cycle
• creates holder snapshots at cycle start
• monitors balances during the cycle
• ejects holders who reduce balances below initialHold
• updates initialHold when balances increase by ≥151%
• calculates progressive reward percentages
• distributes GENc rewards via distributeDividendsAdjusted()
• finalizes completed cycles and opens new cycles automatically

Reward model
The system implements:
• snapshot-based participation
• hold-proof enforcement
• progressive reward scaling
• +151% balance recalibration rule
• batched reward distribution

Eligible Transaction Bot — eliB.js

The Eligible Transaction Bot controls the daily purchase bonus system during the Public Sale reward phase.

Core responsibilities
• listens for GENc transfers originating from the liquidity pool
• determines the current Public Sale phase and program day
• verifies minimum purchase thresholds
• enforces phase-based daily wallet caps
• verifies bonus pool availability
• executes processDailyBonus(address, day) for eligible purchases
• manages phase transitions and unused bonus rollover logic

Airdrop Tracker — airT.js

The Airdrop Tracker maintains the holder dataset used for ecosystem airdrops.

Core responsibilities
• tracks holders above the airdrop minimum balance
• processes historical transfers from the last stored block
• records holders from TokensPurchased events during PreSale
• switches to post-sale tracking after WhitelistClosed
• after PreSale, adds new holders only from LP-originated buys
• removes holders when balances fall below the eligibility threshold
• persists the holder dataset in trackedHolders.json

Airdrop Execution Tool — AirdropCLI.js

The Airdrop CLI executes token distribution transactions.

Core responsibilities
• loads holder data from trackedHolders.json
• verifies balances directly from the token contract
• supports proportional distribution mode
• supports fixed distribution mode
• verifies available balance via airdropPoolGENc()
• executes batched transfers using airdropByList(recipients, amounts)
• records execution history in airdropLog.json

B100D Loyalty Bot — b100d.js

The B100D bot manages monitoring of loyalty program participants.

Core responsibilities
• listens for WhitelistOpened, WhitelistClosed, and B100DRegistered events
• records participant addresses and initialHold balances
• synchronizes participant state via b100dParticipants(address)
• waits for B100D program activation
• monitors transfer events after B100D start
• verifies balance ≥ initialHold
• calls ejectFromWhitelist(address) when a participant breaks the hold requirement

This bot enforces continuous holding conditions for the B100D program.

Final Bonus Bot — finB.js

The Final Bonus Bot controls the final distribution stage of the B100D program.

Core responsibilities
• listens for B100DDayCompleted(day)
• detects when the configured B100D duration is reached
• waits a configurable delay period
• executes triggerFinalBonusDistribution()
• monitors FinalBonusBatchProcessed events and retriggers distribution until all batches are processed
• stops when FinalBonusCompleted is emitted

This mechanism distributes the remaining B100D pool proportionally to eligible participants.

Automation Layer Properties

The automation layer follows several safety constraints:
• bots trigger functions but do not define protocol state
• smart contracts remain the single source of truth
• holder balances are always verified on-chain
• release eligibility is verified on-chain before execution
• bot restarts do not affect protocol state
• all executions appear as standard blockchain transactions

Complete Bot Set

The GENESIS automation infrastructure consists of:
• Oracle Bot — protocol timing and phase transitions
• generalAM.js — Asset Manager v1 vesting execution
• v2GENERAL.js — Asset Manager v2 stage release execution
• divT.js — dividend holder tracking
• divV.js — dividend cycle verification and payouts
• eliB.js — Public Sale daily bonus execution
• airT.js — airdrop holder tracking
• AirdropCLI.js — airdrop distribution execution
• b100d.js — B100D loyalty monitoring
• finB.js — final B100D bonus distribution