BEP-007

LICENSE TO AUTOMATE

A Non-Fungible Agent (NFA) Standard for Autonomous, Stateful Tokens on BNB Chain

Overview

BEP-007 introduces **Non-Fungible Agents (NFAs)**—programmable, autonomous tokens that act as on-chain "agents" capable of executing tasks, evolving, and interacting with other contracts. Inspired by the iconic "007" designation for elite agents, this standard merges the uniqueness of NFTs with autonomous functionality, positioning BNB Chain as the home for decentralized automation.

---

1. Core Features

- **Autonomous Behavior**: Agents execute predefined logic (e.g., trading, interacting with contracts) without manual intervention.
- **Statefulness**: Each agent maintains mutable state variables (e.g., status, parameters) stored on-chain.
- **Interoperability**: Compatible with BEP-20, BEP-721, and cross-chain protocols via bridges.
- **Upgradability**: Proxy patterns or modular logic to update agent behavior securely.

---

2. Token Structure

- **Inheritance**: Extends BEP-721 (NFT standard) with additional agent-specific functions.
- **Metadata**: Includes:
    - Static attributes (e.g., name, description, artwork).
    - Dynamic metadata (e.g., behavior logic, external script references via IPFS).
    - State variables (e.g., `lastActionTimestamp`, `balance`).

---

3. Smart Contract Design

Key Functions

- `executeAction(bytes calldata data)`: Triggers agent logic (e.g., calling DeFi protocols).
- `setLogicAddress(address newLogic)`: Updates agent behavior via a proxy contract.
- `fundAgent()`: Allows owners/third parties to deposit BNB for gas fees.
- `getState() returns (State memory)`: Fetches current agent state.

Factory Contract

- Deploys agent instances with customizable initial parameters.
- Manages templates for common agent behaviors (e.g., trading bots, game NPCs).

---

4. Use Cases

- **DeFi Agents**: Autonomous portfolio managers, liquidity providers, or arbitrage bots.
- **Gaming**: NPCs that evolve, trade items, or interact with players.
- **DAO Participants**: Agents voting or executing proposals based on predefined rules.
- **IoT Integration**: Agents acting as digital twins for physical devices.

---

5. Incentives & Economics

- **Gas Handling**: Agents hold BNB balances to self-fund transactions.
- **Revenue Models**: Agents earn fees (e.g., profit-sharing from trades) or accept payments.
- **Staking**: Owners stake tokens to grant agents permissions or access premium features.

---

6. Security Mechanisms

- **Circuit Breakers**: Pause functions to halt malicious actions.
- **Governance**: Multi-sig or DAO-controlled upgrades for critical logic.
- **Oracles**: Trusted data feeds (e.g., Chainlink) for external input.

---

7. Example Workflow

1. **Deployment**: A factory contract creates an NFA with initial parameters (e.g., trading strategy).
2. **Funding**: Owner deposits BNB into the agent's contract for gas.
3. **Execution**: Agent monitors blockchain events (via off-chain keepers/oracles) and triggers `executeAction` when conditions are met (e.g., ETH price hits $3K).
4. **Interaction**: Agent swaps tokens on PancakeSwap, updates its state, and emits an event.

---

8. Challenges & Solutions

- **Gas Costs**: Optimize logic for batch actions; use layer-2 solutions (e.g., zkRollups).
- **Complexity**: Provide SDKs/templates for common agent types.
- **Regulatory Compliance**: Implement KYC/whitelisting for financial agents.

---

9. Proposed Standard (BEP-007)

- **Interface**:

```
Unset
interface INFAgent {
    function executeAction(bytes calldata data) external;
    function setLogicAddress(address newLogic) external;
    function fundAgent() external payable;
    event ActionExecuted(address indexed agent, bytes result);
}
```

---

10. Roadmap

1. **Phase 1**: Draft standard and prototype agent (e.g., a simple trading bot).
2. **Phase 2**: Audit contracts and launch testnet on BNB Smart Chain.
3. **Phase 3**: Mainnet deployment with governance and developer tools.

Technical Proposal

Core Features

- **Autonomous Behavior**: Agents execute logic (e.g., trading, voting, gaming) based on predefined rules or external triggers.
- **Stateful Evolution**: Each agent maintains mutable properties (e.g., `status`, `balance`, `strategy`).
- **Interoperability**: Compatible with BEP-20, BEP-721, and cross-chain via bridges (e.g., Binance Oracle).
- **Proxy Upgradability**: Modular logic contracts allow secure updates to agent behavior.

Use Cases

- **DeFi Agents**: Auto-rebalancing portfolios, yield farmers, or arbitrage bots.
- **Gaming NPCs**: Autonomous in-game characters that trade items or battle players.
- **DAO Delegates**: Agents vote or execute proposals based on predefined governance logic.
- **IoT Digital Twins**: Agents mirror real-world devices (e.g., self-managing solar panels).

Technical Specifications

- **Inheritance**: Extends BEP-721 (NFT standard) with agent-specific functions.
- **Key Functions**:

```
Unset
function executeAction(bytes calldata _data) external;
function setLogicAddress(address _newLogic) external;
function getState() external view returns (State memory);
```

Security: Circuit breakers, multi-sig governance for critical updates, and Chainlink oracles for external data.

"The name’s Agent. Non-Fungible Agent."

I. Governance Structure

1. Governance Model

- **Decentralized Autonomous Organization (DAO)**:
    - **Token-Based Voting**: BEP-007 agent owners (NFA holders) vote on protocol upgrades, parameter changes, and treasury allocation.
    - **Proposal Types**:
        - **Technical**: Upgrading agent logic, adjusting gas subsidies.
        - **Treasury**: Funding grants, partnerships, or marketing.
        - **Emergency**: Circuit-breaker activation (e.g., halting malicious agents).

2. Proposal Lifecycle

1. **Submission**: Stake 1,000 BNB to submit a proposal (anti-spam measure).
2. **Discussion**: 7-day forum debate on Binance Square or Snapshot.
3. **Voting**: 5-day on-chain voting period.
4. **Execution**: Approved proposals auto-execute via `Governance.sol`.

3. Voting Mechanics

- **Quadratic Voting**: Voting power = `sqrt(NFA_count)` to prevent whale dominance.
- **Delegation**: Agents/NFA owners can delegate votes to representatives.
- **Quorum**: 20% of circulating NFAs must participate for validity.

4. Roles

- **Admins**: Multi-sig (5/9) for emergency pauses or critical fixes.
- **Developers**: Earn grants from the treasury for ecosystem tooling.
- **Guardians**: Chainlink Keepers or decentralized watchdogs to flag malicious proposals.

II. Smart Contract Design

1. Architecture

- **Core Contracts**:
  - `BEP007.sol`: Base NFA contract (inherits BEP-721).
  - `BEP007Governance.sol`: DAO voting and proposal execution.
  - `BEP007Proxy.sol`: Upgradeable logic via UUPS (EIP-1822).
  - `AgentFactory.sol`: Deploys NFA instances with templates.

2. BEP007.sol (Base Contract)

State Variables

```
Unset
address public owner;
address public logicAddress; // Upgradable logic
uint256 public balance; // BNB balance for gas
enum Status { Active, Paused, Terminated }
Status public status;
mapping(bytes32 => Action) public actions; // Registered behaviors
```

Key Functions

```
Unset
// Execute agent logic (callable by owner or authorized contracts)
function executeAction(bytes calldata _data) external {
  require(status == Status.Active, "BEP007: Inactive");
  (bool success, bytes memory result) = logicAddress.delegatecall(_data);
  emit ActionExecuted(msg.sender, result);
}

// Fund agent with BNB for gas
function fundAgent() external payable {
  balance += msg.value;
}

// Pause agent (callable by governance or admin)
function pause() external onlyGovernance {
```

```
  status = Status.Paused;
}
```

Modifiers

```
Unset
modifier onlyGovernance() {
  require(msg.sender == BEP007Governance.getVault(), "Unauthorized");
  _;
}
```

3. BEP007Governance.sol

Proposal Struct

```
Unset
struct Proposal {
  uint256 id;
  address proposer;
  string description;
  uint256 forVotes;
  uint256 againstVotes;
  uint256 startBlock;
  uint256 endBlock;
  bool executed;
}
```

Voting Logic

```
Unset
function vote(uint256 _proposalId, bool _support) external {
```

```
  Proposal storage p = proposals[_proposalId];
  require(block.number <= p.endBlock, "Voting closed");
  uint256 votes = sqrt(BEP007.balanceOf(msg.sender));
  _support ? p.forVotes += votes : p.againstVotes += votes;
}
```

Security

- Timelock: 48-hour delay before executing approved proposals.
- Veto Power: Multi-sig admins can override proposals with >75% consensus.

4. AgentFactory.sol

```
Unset
// Deploy a new NFA with customizable logic
function createAgent(
  string memory _name,
  string memory _symbol,
  address _initialLogic
) external returns (address agent) {
  agent = Clones.cloneDeterministic(template, keccak256(abi.encode(_name)));
  BEP007(agent).initialize(_name, _symbol, _initialLogic);
  emit AgentCreated(agent, _initialLogic);
}
```

III. Security & Upgradability

1. Proxy Design

- **UUPS Proxy**: Logic upgrades are stored in `BEP007.sol`, reducing proxy attack surface.
- **Upgrade Constraints**:
  - New logic must pass 7-day governance vote.
  - Upgrades cannot modify historical agent states.

2. Attack Mitigation

- **Reentrancy Guards**: OpenZeppelin's `ReentrancyGuard` for `executeAction()`.
- **Oracle Sanitization**: Chainlink for tamper-proof price/data feeds.
- **Gas Limits**: Caps on delegatecall gas to prevent out-of-gas attacks.

3. Audit Trail

- **Automated Checks**: Slither or MythX integration in CI/CD pipeline.
- **Bug Bounty**: Fund for white-hat hackers (managed by treasury).

V. Implementation Checklist

1. Pre-Launch

- Deploy `BEP007Governance` with initial multi-sig (Binance, CertiK, core devs).
- Publish `BEP007` contracts on BscScan and verify bytecode.
- Write Foundry tests for 100% branch coverage.

2. Post-Launch

- Onboard 3+ auditing firms (e.g., PeckShield, Quantstamp).
- Create a `BEP007-Examples` GitHub repo with agents for DeFi, gaming, and DAOs.
- Integrate with BSC wallets (Trust Wallet, MetaMask) for agent management.

V. Example Governance Proposal

Title: *"Upgrade Agent Logic to Support LayerZero Cross-Chain"*

Proposer: 0xDAO...

Description:

- Integrate LayerZero's Omnichain Fungible Token (OFT) standard into `BEP007.sol`.

- Allocate 50 BNB from treasury for development.

Voting Options:

- **For**: Enables cross-chain agents (e.g., Ethereum <> BSC arbitrage).
- **Against**: Prioritize single-chain stability.

VI. Final Code Snippet

BEP-007 Interface

```
Unset
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface IBEP007 {
  event ActionExecuted(address indexed agent, bytes result);
  event LogicUpgraded(address oldLogic, address newLogic);

  function executeAction(bytes calldata data) external;
  function setLogicAddress(address newLogic) external;
  function fundAgent() external payable;
  function getState() external view returns (
    uint256 balance,
    Status status,
    address owner
  );
}
```

Here's a **comprehensive breakdown of the BEP-007 architecture** and the smart contracts required to implement the Non-Fungible Agent standard, organized by module and functionality:

BEP-007 Architecture & Smart Contracts

"License to Automate"

---

1. Core Contracts

1.1 `BEP007.sol`

Purpose: Base contract for all Non-Fungible Agents (NFAs).

Inherits: `BEP-721`, `UUPSUpgradeable`, `ReentrancyGuard`.

Key Features:

- Stateful properties (balance, status, logic address).
- Autonomous action execution via delegatecall.
- Upgradeable logic with governance controls.

Key Functions:

```
Unset
function executeAction(bytes calldata data) external;
function fundAgent() external payable;
function setLogicAddress(address newLogic) external;
function pause() external onlyGovernance;
```

1.2 `IBEP007.sol`

Purpose: Standard interface for BEP-007 compliance.

Key Components:

```
Unset
interface IBEP007 {
  event ActionExecuted(address indexed agent, bytes result);
  event LogicUpgraded(address oldLogic, address newLogic);
```

```
  function executeAction(bytes calldata data) external;
  function getState() external view returns (uint256 balance, Status
status);
}
```

---

2. Governance & DAO

2.1 `BEP007Governance.sol`

Purpose: On-chain governance for protocol upgrades and parameter changes.

Inherits: `OpenZeppelin Governor`, `TimelockController`.

Key Features:

- Quadratic voting weighted by NFA ownership.
- Proposal lifecycle (submission, voting, execution).
- Emergency veto via multi-sig.

Key Functions:

```
Unset
function propose(string memory description, bytes calldata logicUpgrade)
external;
function vote(uint256 proposalId, bool support) external;
function executeProposal(uint256 proposalId) external;
```

2.2 `BEP007Treasury.sol`

Purpose: Manages protocol revenue (gas subsidies, agent fees).

Key Features:

- Revenue splits between agent owners and DAO.
- Funds grants for ecosystem development.

Key Functions:

```
Unset
function propose(string memory description, bytes calldata logicUpgrade)
external;
function vote(uint256 proposalId, bool support) external;
function executeProposal(uint256 proposalId) external;
```

---

3. Agent Creation & Management

3.1 `AgentFactory.sol`

Purpose: Deploys NFA instances from templates.

Key Features:

- Clones pattern for gas-efficient deployment.
- Template whitelisting via governance.

Key Functions:

```
Unset
function deposit() external payable;
function withdraw(uint256 amount, address recipient) external
onlyGovernance;
```

3.2 `TemplateRegistry.sol`

Purpose: Stores approved agent logic templates.

Key Features:

- Version control for agent behaviors.
- Governance-controlled whitelist.

Key Functions:

```
Unset
function addTemplate(address template, string memory version) external
onlyGovernance;
function getLatestTemplate(string memory category) external view returns
(address);
```

---

4. Upgradeability & Security

4.1 `BEP007Proxy.sol`

Purpose: UUPS proxy for logic upgrades.

Key Features:

- Upgrade authorization via governance vote.
- Version rollback prevention.

Key Functions:

```
Unset
function addTemplate(address template, string memory version) external
onlyGovernance;
function getLatestTemplate(string memory category) external view returns
(address);
```

4.2 `CircuitBreaker.sol`

Purpose: Emergency shutdown mechanism.

Key Features:

- Pauses all agent activity globally or per-contract.
- Multi-sig or oracle-triggered.

Key Functions:

```
Unset
function toggleGlobalPause(bool paused) external onlyGuardian;
```

---

5. External Integrations

5.1 `OracleIntegration.sol`

Purpose: Fetches off-chain data (prices, social trends).

Key Features:

- Chainlink/API3 oracle support.
- Data sanitization for on-chain actions.

Key Functions:

```
Unset
function toggleGlobalPause(bool paused) external onlyGuardian;
```

5.2 `ChatAndBuildAdapter.sol`

Purpose: Bridges BEP-007 agents to ChatAndBuild AI APIs.

Key Features:

- Securely stores API keys via Lit Protocol encryption.
- Generates autonomous agent behavior via LLMs.

Key Functions:

```
Unset
function generateTweet(string memory prompt) external returns (string
memory);
function analyzeSentiment(string memory text) external returns (uint256
score);
```

---

6. Example Agent Contracts

6.1 `TwitterAgent.sol`

Purpose: Autonomous social media manager.

Key Features:

- Posts AI-generated tweets via `ChatAndBuildAdapter`.
- Tracks engagement metrics on-chain.

State Variables:

```
Unset
function generateTweet(string memory prompt) external returns (string
memory);
```

```
function analyzeSentiment(string memory text) external returns (uint256
score);
```

6.2 `DeFiArbitrageAgent.sol`

Purpose: Automated cross-DEX arbitrage.

Key Features:

- Monitors price differences via `OracleIntegration`.
- Executes swaps on PancakeSwap/Uniswap.

Key Functions:

```
Unset
function findArbitrageOpportunity() external returns (bool);
function executeSwap(address router, bytes calldata path) external;
```

---

7. Utility Contracts

7.1 `NFARenderer.sol`

Purpose: Dynamic metadata generation for agents.

Key Features:

- SVG animations showing agent activity.
- IPFS storage for historical data.

Key Functions:

```
Unset
function generateSVG(address agent) external view returns (string memory);
```

7.2 `AgentRegistry.sol`

Purpose: Global registry of deployed agents.

Key Features:

- Search/filter by agent type or owner.
- Reputation scoring system.

Key Functions:

```
Unset
function registerAgent(address agent, string memory agentType) external;
function getAgentsByOwner(address owner) external view returns (address[]
memory);
```

---

8. Security Architecture

Contract	Security Mechanisms
`BEP007.sol`	Reentrancy guards, delegatecall gas limits
`BEP007Governance.sol`	Timelock delays, proposal quorum checks
`AgentFactory.sol`	Template whitelisting, clone signature verification
`OracleIntegration.sol`	Multiple oracle fallbacks, data freshness checks

---

9. Contract Interactions

1. **User** deploys agent via `AgentFactory`.
2. **Agent** fetches data via `OracleIntegration`.
3. **Governance** votes to upgrade logic via `BEP007Proxy`.
4. **Treasury** funds agents via revenue splits.

This architecture enables a robust ecosystem for autonomous, self-evolving agents while maintaining security and decentralization. Developers can extend it with new agent types (e.g., IoT controllers, DAO delegates) using the modular template system.