GNX ECIGEM · OPERATING PRINCIPLES BLUE BOOK

GNX ECIGEM 청서

The Blue Book of GNX ECIGEM

작동·운영 원리와 방식에 관한 공식 설명서·매뉴얼·참고서 — 일곱 단계 처리 평면의 구조와 구현에 관한 의견서.

발행 · **주식회사 지엔엑스 (GNX Co., Ltd.)**
대표이사 · Kim Chul
판본 · v2.1.0 / rustc 1.96.0 / 2026
대상 · 구현·운영·연동 엔지니어 및 검증 실무자

본 청서의 모든 구조·규칙은 공개 엔진 소스(origin·compress·motion·evaluate·graph·binding·audit)와 청구항 도면에 근거합니다.

의견서 · POSITION STATEMENT

여는 의견서

본 청서는 GNX ECIGEM 엔진이 ‘어떻게 작동하는가’를 구현 수준에서 정확히 기술하는 공식 설명서이자, 동시에 그 작동 방식이 기존 접근과 구조적으로 다르다는 점을 객관적 근거로 정중하게 주장하는 의견서입니다.

백서(The White Book)가 검증·라이선스의 관점에서 ‘무엇이 보장되는가’를 다룬다면, 청서는 운영·구현의 관점에서 ‘무엇이 어떤 순서로 일어나는가’를 다룹니다. 본 청서의 모든 단계 설명은 공개 모듈 소스와 청구항 도면에 1:1로 정박되어 있으며, 추상적 비유는 구현과의 대응이 명시될 때에만 사용됩니다.

엔진은 일곱 단계의 순방향 평면으로 구성됩니다 — 출생점, 점 신체, 점 운동, 점 상태, 별자리, 자성 결속, 감사. 한 요청은 이 평면을 한 방향으로 통과하며 효력 상태를 확정하고, 그 전 과정은 위변조 검출이 가능한 영수증 체인에 봉인됩니다.

본 청서는 엔지니어와 검증 실무자를 1차 독자로 하며, 각 장은 해당 단계의 구조·규칙·관찰 가능한 산출을 제시한 뒤 요약문으로 마무리됩니다.

GNX ECIGEM 발행 책임 · 주식회사 지엔엑스
본 의견서는 제10장 「닫는 의견서」에서 동일한 어조로 결론지어집니다.

목차 · Contents

CHAPTER I

시스템 개관 — 일곱 단계 처리 평면

1.1 설계 철학과 전체 구성(도 1)

GNX ECIGEM은 '기능효력 제어 시스템(100)'으로서, 기존 기능 체계(200)와 대상 입력(300) 사이에 위치하여 효력 부여 이전에 판단을 수행합니다(도 1). 대상 입력의 예시로는 API 요청(301), 프롬프트 입력(302), AI 에이전트 도구 호출(303), 세션 지속(304), 전화번호 표시 신뢰(305), 금융 승인(306), 데이터 반출(307), 계정 상태전이(308), 계약상 효력(309), 관리자 예외(310)가 있습니다.

엔진은 입력 수신부(110)에서 시작하여 출생점 부여부(120), 점 신체 압축부(130), 점 운동 수집부(140), 점 상태 평가부(150), 별자리 평가부(160), 자성 결속부(170), 기능효력 제어부(180), 감사 영수증부(190)로 이어집니다. 중앙에서는 출생점 기록(400)·점 신체 정보(500)·점 운동 정보(600)·점 상태 평가 결과(700)·기능효력 결정(800)이 차례로 생성되고, 하단에서는 감사 영수증(900)이 체인(906)에 저장됩니다.

1.2 처리 평면 7단계 개요

실제 구현(ecigem_core)은 이 도면 구조를 일곱 개의 Rust 모듈로 실현합니다.

단계	모듈	핵심 산출
출생점	`origin.rs`	OriginProof(HMAC-SHA256)
점 신체	`compress.rs`	CompressedIdentity(dot_id, BLAKE3)
점 운동	`motion.rs`	MotionPulse[]
점 상태	`evaluate.rs`	EffectDecision[]
별자리	`graph.rs`	odd_state
자성 결속	`binding.rs`	BindingStatus, clearance
감사	`audit.rs`	AuditReceipt(체인)

한 요청은 이 일곱 단계를 한 방향으로 통과하며, 각 단계의 산출은 다음 단계의 입력이 됩니다. 평면은 분기 없이 전진하므로, 결정 경로는 항상 추적 가능합니다.

1.3 실행 환경과 시간 윈도우

엔진은 Rust(rustc 1.96.0)와 tokio 비동기 런타임 위에서 동작하며, 점 운동 수집은 100ms 단위의 마이크로 배치 윈도우로 처리됩니다. 평가는 rayon을 통해 병렬화되어, 한 윈도우 내 다수 펄스를 동시에 산정합니다.

이 시간 윈도우는 단순한 성능 장치가 아니라 판단의 단위입니다. 별자리 단계의 이상 판정은 '단기간 내' 관계 급증을 본질로 하므로, 윈도우의 존재 자체가 odd-state 판정의 의미를 규정합니다.

> 요약문
>
> 엔진은 기존 기능 체계와 대상 입력 사이에서 효력 부여 이전에 판단하는 기능효력 제어 시스템(100)이다. 도면 구조(110–190, 400–906)는 일곱 개 Rust 모듈로 실현되며, 한 요청은 출생점→감사까지 분기 없이 전진한다. 실행은 tokio·rayon 위에서 100ms 윈도우 단위로 이루어진다.

CHAPTER II

출생점 — origin.rs

2.1 입력 수신과 정규화(도 2)

출생점 단계는 대상 입력 또는 참조값을 수신하고, 기능효력 요청 유형을 식별한 뒤, 기원·표면·세션·목적·범위·어댑터·정책·라이선스 정보를 추출합니다(도 2의 1~3단계). 이어 문자·유니코드·구분자 정규화와 순서 고정, 시간 버킷 부여가 수행됩니다(4단계).

여기서 핵심은 도 2의 주석입니다 — 원문 개인정보·원문 프롬프트·원문 키값·원문 세션 식별자는 기능효력 판단을 위한 원문 보관 객체로 저장되지 않습니다. 엔진이 다루는 것은 정규화된 특질값(facets)과 표면 유형이며, 원문 자체가 아닙니다.

2.2 HMAC-SHA256 출생 증명

정규화된 입력은 테넌트 비밀키로 봉인됩니다. 출생 증명은 다음과 같이 산출됩니다.

```
hmac_signature = HMAC_SHA256( tenant_id ‖ surface_type ‖ time_bucket(BE u64) ‖ facets )
```

시간 버킷은 time_bucket = unix_secs / 10으로 정의되어 10초 단위로 고정됩니다. 따라서 동일 입력은 동일 버킷 안에서 동일 서명을 산출하여 결정성과 재현성을 동시에 확보하고, 버킷 경계를 가로지르면 의도적으로 새로운 서명이 생성됩니다(회전).

산출물 OriginProof는 tenant_id·surface_type·time_bucket·hmac_signature를 담습니다. 이 서명은 서버의 ring 라이브러리와 브라우저 WebCrypto에서 동일 알고리즘으로 계산되며, 키 없이는 위조할 수 없습니다.

> 요약문
>
> 출생점은 대상 입력을 수신·정규화하되 원문을 보관하지 않고, 테넌트 비밀키 기반 HMAC-SHA256으로 요청 기원을 봉인한다. 서명 메시지는 tenant·surface·time_bucket(BE)·facets의 결합이며, time_bucket=unix/10으로 10초 단위 결정성과 회전을 동시에 제공한다.

CHAPTER III

점 신체 — compress.rs

3.1 dot_id와 출생 컨텍스트 해시

점 신체 단계는 출생점 증명을 하나의 '점'으로 압축합니다. 이 단계는 고유 식별자 dot_id(UUID v4)를 발급하고, 출생 컨텍스트를 비가역 해시로 응축합니다.

```
birth_context_hash = BLAKE3( hmac_signature ‖ tenant_id ‖ surface_type ‖ time_bucket(BE) )
```

산출물 CompressedIdentity는 dot_id·birth_context_hash·policy_version(v2.1.0)을 담습니다. dot_id는 이후 점 운동·별자리·감사 단계에서 이 요청을 가리키는 단일 키로 사용됩니다.

3.2 비가역성과 원문 비보관

birth_context_hash는 BLAKE3 해시이므로 원문 컨텍스트로 복원되지 않습니다. 즉 점 신체는 '무엇이었는가'의 원문이 아니라 '무엇으로부터 태어났는가'의 봉인입니다.

이 비가역성은 비삭제 원칙과 짝을 이룹니다. 엔진은 원문을 저장하지 않으면서도, 동일 입력이 동일 점 신체로 압축됨을 보장하여 결정성과 프라이버시를 동시에 만족합니다.

> 요약문
>
> 점 신체는 출생 증명을 dot_id(UUID v4)와 birth_context_hash(BLAKE3)로 압축한 CompressedIdentity를 산출한다. dot_id는 이후 단계의 단일 키가 되며, 해시는 비가역적이어서 원문을 복원하지 않는다. 비가역성과 원문 비보관이 결정성·프라이버시를 동시에 보장한다.

CHAPTER Ⅳ

점 운동 — motion.rs

4.1 표면 이벤트와 MotionPulse(도 3)

점 운동 단계는 각 어댑터에서 발생하는 표면 이벤트를 점 운동 이벤트(601)로 변환합니다(도 3). 어댑터에는 API 게이트웨이(201), AI 에이전트 런타임(206), 통신 세션(207), 금융 승인(208), 데이터 반출(209), 관리자 행위(210)가 병렬로 존재합니다.

각 이벤트는 MotionPulse 구조로 표현되며, dot_id·context_hash(=birth_context_hash)·event_type·timestamp_ms를 담습니다. 즉 모든 펄스는 자신이 어느 점 신체에 속하는지를 context_hash로 명시합니다.

4.2 100ms 마이크로 배치 윈도우

변환된 펄스는 pulse queue(142)로 유입되어 시간 윈도우 처리 모듈(143)에서 0.1초 단위로 수집됩니다(도 3). 이어 마이크로 배치 요약 모듈(144)과 운동 변화값 산출 모듈(146)을 거쳐 점 상태 평가부(150)로 전달됩니다.

100ms 윈도우는 '짧은 시간 안의 움직임'을 하나의 판단 단위로 묶습니다. 이 묶음이 있어야 비로소 다음 단계에서 '급격한 수렴'을 의미 있게 판정할 수 있습니다.

요약문

점 운동은 어댑터(201–210)의 표면 이벤트를 MotionPulse(dot_id·context_hash·event_type·timestamp_ms)로 변환한다. 펄스는 pulse queue를 거쳐 100ms 마이크로 배치 윈도우로 수집되며, 이 시간 단위가 이후 odd-state 판정의 전제가 된다.

CHAPTER V

점 상태 — evaluate.rs

5.1 병렬 평가와 결정 규칙

점 상태 단계는 수집된 펄스 묶음을 rayon 병렬 반복(par_iter)으로 동시에 평가합니다. 참조 구현의 결정 규칙은 결정론적이며 다음과 같습니다.

- 펄스의 context_hash가 비어 있으면 → `Drop` (출생 컨텍스트 결손)
- event_type이 `TRANSITION_6`을 포함하면 → `Watch` (주시)
- 그 외에는 → `Allow`

산출물 EffectDecision은 Allow·Watch·Drop의 세 값을 가지며, 평가는 펄스별로 독립적으로 이루어집니다. 병렬 평가의 결과 순서는 입력 순서와 일치하도록 수집됩니다.

5.2 개념 모델(도 4)과 참조 구현의 대응

도 4는 점 상태 평가부의 개념 모델로서 점도값 산출부(151), 향기 벡터 산출부(152), 후각마비 상태 판정부(153), drift index 산출부(154), rupture flag 판정부(155)를 제시하며, 출력으로 점도값(701)·향기 벡터(702)·후각마비 상태(703)를 둡니다.

본 청서는 개념 모델과 구현을 혼동하지 않습니다. 위 개념 구성요소는 '기원의 결, 운반 표면의 결, 의미의 결' 같은 연속성·결의 평가를 추상화한 모델이며, 현재 참조 구현(evaluate.rs)은 그 모델이 지향하는 판정을 context_hash 유효성·전이 신호·이상 수렴이라는 결정론적 규칙으로 실현합니다. 개념과 구현의 이 대응 관계를 명시하는 것이 본 청서의 표현 원칙입니다.

> 요약문
>
> 점 상태는 rayon 병렬로 펄스를 평가하여 Allow·Watch·Drop을 산출한다. 규칙은 결정론적이다 — context_hash 결손→Drop, TRANSITION_6 포함→Watch, 그 외→Allow. 도 4의 점도값·향기벡터·후각마비는 개념 모델이며, 참조 구현은 그 지향을 결정론적 규칙으로 실현한다는 대응 관계를 명시한다.

CHAPTER VI

별자리 — graph.rs

6.1 관계 그래프와 edge_count

별자리 단계는 점들을 관계 그래프로 연결합니다(도 5). 각 노드는 출생점 기록 또는 dot_id를 의미하고, 각 엣지는 시간·목적·표면·세션·기원·어댑터 관계를 의미합니다. 구현은 petgraph 기반 방향 그래프로, 한 펄스가 들어올 때마다 해당 dot에 엣지를 추가합니다.

이상 판정은 단순하고 명확합니다 — 동일 dot의 edge_count가 5를 초과하면 odd_state가 참이 됩니다. 즉 단기간(100ms 윈도우) 안에 한 점의 컨텍스트 전이가 6회 이상 급증하면 별자리가 비정상으로 수렴한 것으로 판정합니다.

6.2 NORMAL·WATCH·ODD 상태(도 5)

도 5는 세 가지 별자리 상태를 제시합니다 — 정상 관계의 NORMAL(706), 추가 관찰이 필요한 WATCH(707), 급격히 수렴하는 ODD(708). 별자리 평가부(160)는 관계 그래프 생성·edge 산출·시간/목적/표면/세션 관계 평가·상태 판정부(161~167)를 통해 관계 별자리 상태(704)를 산출합니다.

odd_state가 참이면, 앞 단계의 펄스 판정이 무엇이었든 최종 판정은 안전측 Drop으로 격상됩니다. 이 격상 규칙이 별자리 단계의 핵심 효용입니다 — 개별 펄스로는 정상으로 보일 수 있는 흐름도, 관계의 모양이 비정상이면 효력을 잃습니다.

요약문

별자리는 점들을 petgraph 관계 그래프로 연결하고, 동일 dot의 edge_count>5이면 odd_state를 참으로 판정한다. 상태는 NORMAL(706)·WATCH(707)·ODD(708)로 구분되며, ODD인 경우 펄스 판정과 무관하게 최종 판정이 Drop으로 격상된다. 관계의 '모양'을 보는 것이 정적 규칙과의 차별점이다.

CHAPTER VII

자성 결속 — binding.rs

7.1 BindingStatus와 effect scope(도 6)

자성 결속 단계는 점 상태·별자리 결과에 따라 기능효력 범위를 결속합니다(도 6). 구현은 DashMap 기반 결속 레지스트리로, dot_id를 키로 효력 상태를 관리합니다.

결정	결속(BindingStatus)	scope
`Allow`	Active(scope)	부여
`Watch`	Watched(scope)	주시 하 부여
`Drop`	FailInert	없음(null)

effect scope의 하위 항목은 실행·승인·데이터 반출·신뢰표시·세션 지속·계약상 효력·상태전이를 포함합니다(도 6). 즉 효력은 '전부 아니면 전무'가 아니라, 특정 범위에 한정하여 부여되거나 주시 상태로 전환됩니다. 대상 입력·출생점·점 군집은 삭제·파괴되지 않습니다(도 6 주석).

7.2 clearance 조회

후단 시스템은 실행 직전 결속 상태를 조회합니다. check_clearance는 Active에 대해 CLEARANCE_GRANTED, Watched에 대해 CLEARANCE_WATCHED를 반환하고, FailInert에 대해서는 "FAIL_INERT: 원문은 보존되나 기능적 효력 획득이 영구 차단됨"이라는 거부를 반환합니다.

이 조회 패턴이 효력 제어의 실천적 접점입니다. 후단은 clearance 응답에 따라 실행·제한실행·미실행을 분기하며, FAIL_INERT 대상에 대해서는 실행 가능한 핸들·승인값을 생성·사용하지 않습니다.

요약문

자성 결속은 DashMap 레지스트리에서 결정에 따라 Active·Watched·FailInert를 부여한다. effect scope는 실행·승인·데이터반출·신뢰표시·세션지속·계약상효력·상태전이로 세분되어, 효력이 범위 한정으로 제어된다. check_clearance는 후단 실행 직전의 접점으로, FAIL_INERT에 대해 거부를 반환한다.

CHAPTER Ⅷ

감사 영수증 — audit.rs

8.1 체인 구성과 제네시스(도 10)

감사 단계는 모든 결정을 append-only 영수증 체인에 봉인합니다. 체인은 고정 제네시스 GNX_GENESIS_BLOCK_HASH_00000000에서 출발하며, 각 영수증의 현재 해시는 다음과 같이 계산됩니다.

```
current_hash = BLAKE3( prev_hash ‖ dot_id ‖ final_decision ‖ timestamp(BE) )
```

산출물 AuditReceipt는 receipt_id·dot_id·final_decision·prev_hash·current_hash·timestamp를 담습니다. 도 10에 따르면 각 이벤트에는 event_hash(901)가 부여되고, previous_hash 연결부(192)를 통해 감사 영수증 체인(906)이 구성됩니다.

8.2 검증자 역추적

검증자(907)는 receipt_hash(903) 또는 verifier record(904)를 이용하여, 특정 기능효력 결정이 어떤 출생점·점 운동·점 상태·자성 결속에 기초했는지를 역추적합니다(도 10).

체인의 핵심 성질은 변조 검출입니다. 임의 위치의 단일 영수증을 변조하면 그 이후 모든 current_hash가 불일치하므로, 외부 신뢰 없이 체인의 산술만으로 위변조가 드러납니다. 이로써 '결정의 정당성'이 사후에도 독립적으로 재구성됩니다.

요약문

감사는 결정을 제네시스에서 시작하는 append-only 체인에 봉인한다. current_hash=BLAKE3(prev ∥ dot ∥ decision ∥ ts)이며, AuditReceipt가 결정을 기록한다. 검증자는 receipt_hash로 결정의 근거를 역추적하고, 단일 변조가 후속 전체를 어긋나게 하므로 외부 신뢰 없이 변조를 검출한다.

CHAPTER IX

연동·운영

9.1 결정 API와 응답 계약

운영 표면에서 엔진은 결정 API로 노출됩니다. 후단 또는 검증자는 대상 입력을 제공하고, 엔진은 일곱 단계를 실행하여 결정과 증거를 반환합니다.

```
POST /api/decide { tenant_id, surface_type, facets, scope, events[] }
```

응답은 dot_id·birth_context_hash·hmac_signature·decisions[]·edge_count·odd_state·final_decision·binding_status·effect_scope·check_clearance·receipt(를 포함합니다. 최종 판정은 odd_state면 Drop, 아니면 윈도우 내 보수적 최댓값(Drop>Watch>Allow)으로 집계되며, 원시 펄스 판정은 decisions[]로 함께 반환되어 검증자가 직접 대조할 수 있습니다.

9.2 배포 표면과 후단 연동

정적 검증 캔버스와 문서는 nginx 웹 루트에서 서비스되고, 결정 API는 127.0.0.1의 내부 포트에 바인드되어 nginx의 /api/ 프록시를 통해서만 외부로 노출됩니다. 서비스는 systemd로 관리되며, 퍼블릭 가용성은 /health/ready로 점검됩니다.

후단 연동의 표준 패턴은 '실행 직전 결정 조회'입니다. 후단은 final_decision에 따라 분기하고(Allow→실행, Watch→제한실행/추가검증, Drop→미실행), 결속 레지스트리에 대한 clearance 조회로 최종 확인합니다. 응답은 식별자·해시·결정만 반환하며 원문을 저장·반환하지 않습니다(도 7·8·9의 연동 예시와 일치).

> 요약문
>
> 운영 표면에서 엔진은 POST /api/decide로 노출되어 결정과 증거(해시·체인 포함)를 반환한다. 최종 판정은 odd_state면 Drop, 아니면 윈도우 내 보수적 최댓값이며, decisions[]로 원시 판정을 함께 제공한다. 정적 표면은 nginx, 결정 API는 내부 포트+프록시, 서비스는 systemd, 가용성은 /health/ready로 관리된다.

CHAPTER X

닫는 의견서 — 결론

본 청서는 엔진의 작동을 일곱 단계로 정확히 기술하면서, 그 방식이 기존 접근과 구조적으로 다르다는 점을 단계마다 구현 증거로 뒷받침했습니다.

출생점의 결정론적 봉인, 점 신체의 비가역 압축, 점 운동의 시간 윈도우, 점 상태의 병렬·결정론 평가, 별자리의 관계 그래프 odd-state, 자성 결속의 범위 한정 효력, 감사의 자기검출 체인 — 이 일곱은 분리된 기능이 아니라 하나의 순방향 평면을 이룹니다. 그리고 그 평면의 모든 단계는 /api/decide 응답과 공개 표면에서 관찰됩니다.

우리는 개념 모델과 참조 구현을 구분하여 기술했고, 시연과 운영의 경계를 명시했습니다. 이 정직한 구분이 본 청서를 '설명서'이자 동시에 신뢰할 수 있는 '의견서'로 만듭니다.

엔지니어와 검증 실무자에게 본 청서가 드리는 결론은 간명합니다 — 엔진의 작동은 추적 가능하고, 결정은 재현 가능하며, 무결성은 자체 검출 가능합니다. 이 세 성질이 GNX ECIGEM 작동 원리의 본질입니다.

이상으로 의견서를 닫습니다.
주식회사 지엔엑스 (GNX Co., Ltd.) · 대표이사 Kim Chul

요약문

결론적으로 일곱 단계는 분리된 기능이 아니라 하나의 순방향 평면이며, 모든 단계가 /api/decide와 공개 표면에서 관찰된다. 개념 모델과 구현, 시연과 운영을 구분한 정직함 위에서 — 작동은 추적 가능, 결정은 재현 가능, 무결성은 자체 검출 가능하다는 것이 본 청서의 최종 입장이다.

APPENDIX

부록 · 용어와 참조

A. 검증 명령 예시

아래 명령으로 6펄스 금융 승인 시나리오를 직접 실행하여 FAIL_INERT 결과를 재현할 수 있습니다.

```
curl -s -XPOST https://ecigem.com/api/decide -H 'content-type: application/json' -d '{"surface_type":"FINANCIAL_APPROVAL","facets":"type:transfer|amount:high","scope":"FINANCIAL_API_EXECUTION","events":["STATE_TRANSITION_1",...,"STATE_TRANSITION_6"]}'
```

기대 응답 — odd_state: true, final_decision: "Drop", binding_status: "FailInert".

B. 모듈·구조체 색인

모듈	핵심 구조체/함수
origin.rs	OriginGenerator, generate_proof, OriginProof
compress.rs	compress_origin, CompressedIdentity
motion.rs	MotionPulse
evaluate.rs	PulseEvaluator, evaluate_batch, EffectDecision
graph.rs	ConstellationGraph, evaluate_odd_state
binding.rs	EffectBinder, bind_effect, check_clearance
audit.rs	AuditChain, record_receipt, AuditReceipt

본 청서는 「The White Book of GNX ECIGEM」(검증·라이선스)과 상호 보완적이며, 두 문서는 동일한 구현 사실 위에서 작성되었습니다.