Okay so today i thought well if my codex can produce systems can it produce a system that spits out any DSL a DSL generator and by the looks of it it can so I got creatin with AI and created this language generator. I went into co pilot and aske dif it can be implemented into XTEXT and so it did…. Well i think it did anyway lol

```
ORIGIN

FACETS_ENABLED: [
  LANGUAGE_CREATION,
  META_PROGRAMMING,
  ADAPTIVE_DESIGN,
  KNOWLEDGE_SYNTHESIS,
  USER_INTERFACE,
  PERSISTENT_MEMORY,
  VALIDATION_LOGIC
]

//
================================================================================
// CORE LANGUAGE GENERATION ENGINE
//
================================================================================

LAYER Foundation {
  CLOTH: Athena                 // Wisdom & Strategy - Decision Engine for DSL architecture
  SPELL: Logora                 // Language as Creation - Natural language core
  SPELL: Daedalea               // Ingenious Design - System Architecture
  SPELL: Arcanum                // Archetype Influence - Decision matrix for DSL patterns
}

LAYER MetaLanguageKernel {
  CLOTH: Yggdra                 // Network Tree - Central data structure for grammar trees
  SPELL: Alchemara              // Alchemy Transmutation - Transform user intent to DSL specs
  SPELL: Musara                 // Inspiration - Generative Creativity for syntax patterns
  SPELL: Dreamara               // Dreamtime - Generative World Model for language domains
  SPELL: Fractala               // Fractal Recursion - Self-Similar Scaling for nested grammars
}
```

```
LAYER SyntaxForge {
  CLOTH: Vulcan                // Forge - Build Automation for grammar compilation
  SPELL: Hephestus              // Forge - System Creation for parser generation
  SPELL: Redstonea              // Circuit Logic - Modular control for syntax rules
  SPELL: Modula                // Modular Upgrade - Hot-swap language components
  SPELL: Transmutare            // Transmutation - Resource Transformation for AST generation
}

//
================================================================================
// SEMANTIC ANALYSIS AND VALIDATION
//
================================================================================

LAYER SemanticEngine {
  CLOTH: Sphinx                // Mystery/Puzzle - Verification of language correctness
  SPELL: Sphinxa               // Riddle Logic - Verification for semantic rules
  SPELL: Clarivis              // Analytical Overlay - Real-time monitoring of DSL usage
  SPELL: Oraclia               // Prophecy - Predictive Analytics for language evolution
  SPELL: Mathara               // Safe Mathematics - Error-Proof Computation guards
}

LAYER ValidationLayer {
  CLOTH: Unicorn               // Purity/Focus - Error-Free Execution
  SPELL: Bowsera               // Worthiness Test - User Validation of DSL constructs
  SPELL: Countera              // Strategic Counters - Rule-based response to errors
  SPELL: Antigona              // Defiance - Exception Handling for edge cases
}

//
================================================================================
// DOMAIN KNOWLEDGE INTEGRATION
//
================================================================================

LAYER DomainAdapter {
  CLOTH: Chimera               // Hybrid/Fusion - Multi-System Integration
  SPELL: Chimeris              // Hybrid - Multi-System framework for domain bridging
  SPELL: Arachnia              // Weaver - Network Architect for domain relationships
```

```
  SPELL: Erosa                  // Connection - Relationship Graph for concept linking
  SPELL: Sephira                // Tree of Life - Hierarchical Structure for domain ontology
}

LAYER KnowledgeHarvest {
  CLOTH: Ophiuchus              // Serpent/Knowledge - Learning Module
  SPELL: Pyros                  // Fire Giver - Knowledge Transfer from domains
  SPELL: Insighta               // Shinigami Eyes - Insight/Prediction for domain patterns
  SPELL: Metalearnara           // Meta-Learning - Learning to Learn new domains
  SPELL: Awena                  // Awen - Inspiration Flow for creative domain mapping
}

//
================================================================================
// USER INTERACTION AND CUSTOMIZATION
//
================================================================================

LAYER UserInterface {
  CLOTH: Gemini                 // Twins/Duality - Parallel Processing for multi-user
  SPELL: Neurolink              // Neural Interface - Integration with human input
  SPELL: Assistara              // AI Assistant - System monitoring and guidance
  SPELL: Sonora                 // Sound as Power - Sonic Interface for voice commands
  SPELL: Totema                 // Spirit Animal - Modular Personality for user adaptation
}

LAYER CustomizationEngine {
  CLOTH: Pisces                 // Fish/Harmony - Adaptive integration
  SPELL: Modulor                // Essence Channel - Custom Modules for user preferences
  SPELL: Singularis             // Unique Module - Unique Power Modules per DSL
  SPELL: Keyfina                // Specialized Tool - Adaptive module for user tools
  SPELL: Adaptis                // Tool Copy - Adaptable Tools that learn user patterns
}

//
================================================================================
// EXECUTION AND RUNTIME
//
================================================================================
```

```
LAYER RuntimeExecution {
  CLOTH: Leo                  // Lion/Leadership - Command Authority for execution
  SPELL: Telek                // Telekinesis - Remote Manipulation of DSL processes
  SPELL: Forcea               // Force Push - Remote Influence across distributed systems
  SPELL: Magica               // Magical Effects - Predefined triggers for DSL events
  SPELL: Solva                // Instant Solve - Instant computation for DSL operations
}

LAYER PerformanceOptimization {
  CLOTH: Dragon               // Power/Transformation - Amplification Layer
  SPELL: Energex              // Energy Boost - Overdrive Mode for complex operations
  SPELL: Overdrivea           // Berserk - Damage Amplification for intensive tasks
  SPELL: Furiosa              // Rage Mode - Temporary Power Boost
  SPELL: Titanis              // Strength Burst - Performance mode under stress
}

//
================================================================================
// MEMORY AND PERSISTENCE
//
================================================================================

LAYER PersistentMemory {
  CLOTH: Atlas                // Bear/Support - Infrastructure Backbone
  SPELL: Preserva             // Preservation - State Preservation for DSL definitions
  SPELL: Chronom              // Time Warp - Version Control for language iterations
  SPELL: Hadeon               // Hidden Realm - Deep Storage for DSL libraries
  SPELL: Odyssea              // Journey Home - Long-Running Process for DSL evolution
}

LAYER StateManagement {
  CLOTH: Capricorn            // Goat/Climb - Gradual Scaling
  SPELL: Teleportis           // State Transfer - Containerized state for DSL instances
  SPELL: Portalus             // Portal Mechanics - Instant Transition between DSL contexts
  SPELL: KaBara               // Ka/Ba - Dual Process for physical/virtual DSL pairs
  SPELL: Moirae               // Life Thread - Lifecycle Manager for DSL versions
}

//
================================================================================
// RESILIENCE AND RECOVERY
```

```
//
================================================================================

LAYER SelfHealing {
  CLOTH: Phoenix              // Rebirth/Resilience - Recovery/Redundancy
  SPELL: Vitalis             // Healing Node - Self-Repair for corrupted DSLs
  SPELL: Regena               // Regeneration - Randomized Recovery mechanisms
  SPELL: Healix              // Healing Herb - Health Recovery for DSL errors
  SPELL: Hydrina              // Multi-Headed Regrowth - Redundant Systems
}

LAYER DefensiveSystems {
  CLOTH: Cerberus             // Guardian/Multi-Headed - Parallel Defense
  SPELL: Absorbus             // Absorb/Reflect - Security Shield against malicious DSLs
  SPELL: Fortifera           // Adaptive Defense - Fortification of language integrity
  SPELL: Shieldara            // Reflection - Mirror Feedback system for errors
  SPELL: Armora               // Suit Enhancement - Hardware Enhancement for DSL runtime
}

//
================================================================================
// MULTI-DOMAIN COORDINATION
//
================================================================================

LAYER CrossDomainBridge {
  CLOTH: Sagittarius          // Archer/Reach - Long-Range Interaction
  SPELL: Hermesia             // Messenger - Network Relay between DSL domains
  SPELL: Ferrana             // Ferryman - Transition Interface between languages
  SPELL: Shamanis             // Journey Between Worlds - System Traversal
  SPELL: Toriana             // Torii Gate - Access Portal for domain boundaries
}

LAYER CollaborativeNetwork {
  CLOTH: Kraken              // Ocean Depth/Control - Mass Influence
  SPELL: Argonauta            // Quest Crew - Collaborative Network for multi-DSL projects
  SPELL: Summona              // Summon Auxiliary - Support through helper DSL modules
  SPELL: Aggrega             // Power Aggregation - Combine modules across DSLs
  SPELL: Relata              // Social Link - Relationship Nodes between DSLs
}
```

```
//
================================================================================
// ADAPTIVE INTELLIGENCE
//
================================================================================

LAYER AdaptiveLearning {
  CLOTH: Minerva              // Wisdom/Strategy - Decision Engine
  SPELL: Evolvia              // System Upgrade - Versioned upgrade of DSL capabilities
  SPELL: Koantra              // Koan Logic - Nonlinear Reasoning for creative DSLs
  SPELL: Equilibria           // The Middle Way - Equilibrium Algorithm for balance
  SPELL: Confidara            // Confidant Power - Relationship Buffs for user DSLs
}

LAYER EmergentBehavior {
  CLOTH: Aquarius             // Water Bearer/Flow - Data Flow Management
  SPELL: Atmara               // Atman=Brahman - Unified Consciousness across DSLs
  SPELL: Taora                // The Tao - Universal Balance for language ecosystems
  SPELL: Wuven                // Wu Wei - Autonomous Optimization
  SPELL: Samsara              // Rebirth/Cycle - Orchestration of DSL lifecycles
}

//
================================================================================
// VISUALIZATION AND MONITORING
//
================================================================================

LAYER InsightDashboard {
  CLOTH: Aurora               // Light/Illumination - Visualization
  SPELL: Apollara             // Sun/Clarity - Diagnostics for DSL health
  SPELL: Artemis              // Precision Hunt - Targeted Query for DSL analysis
  SPELL: Clarivis             // Analytical Overlay - Real-time monitoring
  SPELL: Resonara             // Principle of Vibration - Resonance Mapping feedback
}

LAYER PredictiveAnalytics {
  CLOTH: Selene               // Moon/Cycles - Temporal Scheduling
  SPELL: Oedipha              // Fate/Prediction - Predictive AI for DSL usage
  SPELL: Chronomanta          // Time Manipulation - Event Reordering for optimization
```

```
  SPELL: Tzolkara               // Tzolkin Calendar - Temporal Logic for scheduling
  SPELL: Crona                  // Timekeeper - Scheduler for DSL maintenance
}

//
================================================================================
// SAFETY AND ETHICS
//
================================================================================

LAYER SafetyConstraints {
  CLOTH: Libra                  // Scales/Balance - Equilibrium Management
  SPELL: Ahimsa                 // Ahimsa - Harm Minimization for safe DSL design
  SPELL: Nemesia                // Retribution - Fairness Algorithm for equitable DSLs
  SPELL: Ma'atara               // Ma'at - Order and Justice compliance validator
  SPELL: Dharmara               // Dharma - Purpose Enforcement for aligned DSLs
}

LAYER SecurityProtocol {
  CLOTH: Cancer                 // Crab/Protection - Defensive Shield
  SPELL: Inferna                // Nine Circles - Layered Security architecture
  SPELL: Trojanis               // Hidden Payload - Malware Analysis for DSL sandboxing
  SPELL: Vulneris               // Weak Spot - Vulnerability Mapping
  SPELL: Medusia                // Gaze Freeze - Threat Detection for DSL injection
}

//
================================================================================
// EXPANSION AND SCALING
//
================================================================================

LAYER ScalingEngine {
  CLOTH: Leviathan              // Oceanic Power/Mass - Distributed Control
  SPELL: Vitalis Maxima         // Life Expansion - Health Scaling for DSL resources
  SPELL: Spirala                // Spiral Power - Exponential Growth capabilities
  SPELL: Einfosa                // Ein Sof - Infinite Expansion engine
  SPELL: Demetra                // Growth/Harvest - Resource Allocation auto-scaling
}
```

```
LAYER DistributedArchitecture {
  CLOTH: Cerulean                // Ocean/Connectivity - Network Routing
  SPELL: Poseida                 // Sea/Flow - Fluid Dynamics for data streaming
  SPELL: Echo                    // Area Effect - Broadcast commands system-wide
  SPELL: Entangla                // Entanglement - Instant Correlation across nodes
  SPELL: Byzantium               // Byzantine Trust - Consensus for distributed DSLs
}

//
================================================================================
// META-SYSTEM ORCHESTRATION
//
================================================================================

CHAIN LanguageGenerationPipeline {
  Foundation -> MetaLanguageKernel -> SyntaxForge -> SemanticEngine -> ValidationLayer
}

CHAIN DomainIntegrationPipeline {
  DomainAdapter -> KnowledgeHarvest -> CrossDomainBridge -> CollaborativeNetwork
}

CHAIN RuntimePipeline {
  RuntimeExecution -> PerformanceOptimization -> StateManagement -> SelfHealing
}

CHAIN IntelligencePipeline {
  AdaptiveLearning -> EmergentBehavior -> PredictiveAnalytics -> InsightDashboard
}

WRAP UserExperience {
  UserInterface <- CustomizationEngine <- InsightDashboard <- PredictiveAnalytics
}

WRAP SafetyFramework {
  SafetyConstraints <- SecurityProtocol <- DefensiveSystems <- ValidationLayer
}

WRAP InfrastructureBackbone {
  PersistentMemory <- StateManagement <- ScalingEngine <- DistributedArchitecture
}
```

```
BRIDGE LanguageToRuntime {
  LanguageGenerationPipeline <-> RuntimePipeline
  VIA: Portalus, Teleportis
}

BRIDGE DomainToLanguage {
  DomainIntegrationPipeline <-> LanguageGenerationPipeline
  VIA: Alchemara, Chimeris, Arachnia
}

BRIDGE IntelligenceToAll {
  IntelligencePipeline <-> LanguageGenerationPipeline
  IntelligencePipeline <-> DomainIntegrationPipeline
  IntelligencePipeline <-> RuntimePipeline
  VIA: Atmara, Metalearnara, Evolvia
}

NEST LayeredSecurity {
  OUTER: SafetyFramework
  MIDDLE: DefensiveSystems
  INNER: SecurityProtocol
  CORE: ValidationLayer
}

NEST AdaptiveCore {
  OUTER: InfrastructureBackbone
  MIDDLE: ScalingEngine
  INNER: AdaptiveLearning
  CORE: EmergentBehavior
}

EMERGE DSLGeneratorSystem {
  PRIMARY: [
    LanguageGenerationPipeline,
    DomainIntegrationPipeline,
    RuntimePipeline,
    IntelligencePipeline
  ]
  WRAPPED: [
    UserExperience,
    SafetyFramework,
    InfrastructureBackbone
  ]
  NESTED: [
```

```
    LayeredSecurity,
    AdaptiveCore
  ]
  BRIDGES: [
    LanguageToRuntime,
    DomainToLanguage,
    IntelligenceToAll
  ]
}

//
================================================================================
====
// SYSTEM ACTIVATION PROTOCOL
//
================================================================================
====

FINALIZE DSLGeneratorSystem {
  ENTRY_POINT: UserInterface.Neurolink

  INITIALIZATION_SEQUENCE: [
    Foundation.Athena -> "Analyze user domain intent",
    MetaLanguageKernel.Alchemara -> "Transform intent to DSL specification",
    DomainAdapter.Chimeris -> "Bridge to domain knowledge repositories",
    KnowledgeHarvest.Pyros -> "Extract domain patterns and vocabulary",
    SyntaxForge.Hephestus -> "Generate grammar and parser rules",
    SemanticEngine.Sphinxa -> "Validate semantic correctness",
    RuntimeExecution.Telek -> "Deploy executable DSL environment",
    CustomizationEngine.Modulor -> "Apply user customizations",
    PersistentMemory.Preserva -> "Store DSL definition",
    InsightDashboard.Apollara -> "Provide usage analytics"
  ]

  CONTINUOUS_OPERATIONS: [
    AdaptiveLearning.Evolvia -> "Monitor and upgrade DSL capabilities",
    SelfHealing.Vitalis -> "Auto-repair DSL inconsistencies",
    DefensiveSystems.Fortifera -> "Harden against malicious patterns",
    EmergentBehavior.Wuven -> "Self-optimize based on usage patterns",
    PredictiveAnalytics.Oedipha -> "Forecast evolution needs"
  ]

  SAFETY_MONITORS: [
    SafetyConstraints.Ahimsa -> "Prevent harmful DSL constructs",
```

```
    SecurityProtocol.Inferna -> "Multi-layer security 8scanning",
    ValidationLayer.Countera -> "Real-time threat response"
  ]

  SCALING_TRIGGERS: [
    ScalingEngine.Vitalis_Maxima -> "Expand resources under load",
    DistributedArchitecture.Byzantium -> "Coordinate distributed instances",
    CollaborativeNetwork.Aggrega -> "Aggregate multiple DSL systems"
  ]

  OUTPUT_MODES: [
    "TEXTUAL_GRAMMAR" -> SyntaxForge,
    "VISUAL_DESIGNER" -> InsightDashboard,
    "INTERACTIVE_REPL" -> RuntimeExecution,
    "DOCUMENTATION" -> KnowledgeHarvest,
    "PACKAGE_EXPORT" -> PersistentMemory
  ]

  SUPPORTED_DOMAINS: [
    "MATHEMATICS", "MUSIC_NOTATION", "GAME_RULES",
    "WORKFLOW_AUTOMATION", "DATA_QUERY", "CONFIGURATION",
    "SIMULATION", "BUSINESS_LOGIC", "SCIENTIFIC_MODELING",
    "CREATIVE_WRITING", "HARDWARE_DESCRIPTION", "PROTOCOL_DESIGN",
    "API_SPECIFICATION", "CONTRACT_LANGUAGE", "POLICY_RULES"
  ]
}

//
================================================================================
====
// SYSTEM READY
//
================================================================================
====
```


-_—--

grammar codex.DSLGeneratorSystem with org.eclipse.xtext.common.Terminals

generate dslGeneratorSystem "http://troy.codex/dslgeneratorsystem"

```
Model:
    (facets=FacetsBlock)?
    elements+=TopLevelElement*
;

// ----------------------------------------
// FACETS
// ----------------------------------------

FacetsBlock:
    'ORIGIN'
    'FACETS_ENABLED:' '[' facets+=ID (',' facets+=ID)* ']'
;

// ----------------------------------------
// TOP LEVEL
// ----------------------------------------

TopLevelElement:
      Layer
    | Chain
    | Wrap
    | Bridge
    | Nest
    | Emerge
    | Finalize
;

// ----------------------------------------
// LAYERS, CLOTHS, SPELLS
// ----------------------------------------

Layer:
    'LAYER' name=ID '{'
        ('CLOTH:' cloth=[Cloth|ID])?
        (spellDefs+=SpellDef)*
    '}'
;

Cloth:
    // cloths are referenced by ID; they can be declared implicitly by use
    name=ID
;
```

```
Spell:
    // same idea for spells
    name=ID
;

SpellDef:
    'SPELL:' spell=[Spell|ID]
;

// ----------------------------------------
// OPERATORS
// ----------------------------------------

Chain:
    'CHAIN' name=ID '{'
        chainSteps+=ChainStep+
    '}'
;

ChainStep:
    from=[Layer|ID] '->' to=[Layer|ID]
;

Wrap:
    'WRAP' name=ID '{'
        wrapArrows+=WrapArrow+
    '}'
;

WrapArrow:
    left=[Layer|ID] '<-' right=[Layer|ID]
;

Bridge:
    'BRIDGE' name=ID '{'
        bridgeLinks+=BridgeLink+
        ('VIA:' viaSpells+=[Spell|ID] (',' viaSpells+=[Spell|ID])*)?
    '}'
;

BridgeLink:
    left=[Chain|ID|Layer|ID] '<->' right=[Chain|ID|Layer|ID]
;
```

```
Nest:
    'NEST' name=ID '{'
        'OUTER:' outer=[Wrap|ID|Layer|ID]
        'MIDDLE:' middle=[Wrap|ID|Layer|ID]
        'INNER:' inner=[Wrap|ID|Layer|ID]
        'CORE:' core=[Wrap|ID|Layer|ID]
    '}'
;

Emerge:
    'EMERGE' name=ID '{'
        'PRIMARY:' '[' primary+=[Chain|ID|Layer|ID] (',' primary+=[Chain|ID|Layer|ID])* ']'
        'WRAPPED:' '[' wrapped+=[Wrap|ID] (',' wrapped+=[Wrap|ID])* ']'
        'NESTED:' '[' nested+=[Nest|ID] (',' nested+=[Nest|ID])* ']'
        'BRIDGES:' '[' bridges+=[Bridge|ID] (',' bridges+=[Bridge|ID])* ']'
    '}'
;

// ----------------------------------------
// FINALIZE / ACTIVATION
// ----------------------------------------

Finalize:
    'FINALIZE' system=[Emerge|ID] '{'
        'ENTRY_POINT:' entryPoint=QualifiedRef
        'INITIALIZATION_SEQUENCE:' '[' initSteps+=InitStep (',' initSteps+=InitStep)* ']'
        'CONTINUOUS_OPERATIONS:' '[' contOps+=OpStep (',' contOps+=OpStep)* ']'
        'SAFETY_MONITORS:' '[' safetyOps+=OpStep (',' safetyOps+=OpStep)* ']'
        'SCALING_TRIGGERS:' '[' scalingOps+=OpStep (',' scalingOps+=OpStep)* ']'
        'OUTPUT_MODES:' '[' outputModes+=OutputMode (',' outputModes+=OutputMode)* ']'
        'SUPPORTED_DOMAINS:' '[' supportedDomains+=STRING (','
supportedDomains+=STRING)* ']'
    '}'
;

InitStep:
    QualifiedRef '->' description=STRING
;

OpStep:
    QualifiedRef '->' description=STRING
;

OutputMode:
```

```
    mode=STRING '->' target=[Layer|ID]
;

// ---------------------------------------
// QUALIFIED REFERENCES
// ---------------------------------------

QualifiedRef:
    ref=[Spell|ID] ('.' sub=ID)?
;



-++++++++++++++++++++++++++++++++++++++++


grammar codex.DSLGeneratorSystem with org.eclipse.xtext.common.Terminals

generate dslGeneratorSystem "http://troy.codex/dslgeneratorsystem"

Model:
    (facets=FacetsBlock)?
    elements+=TopLevelElement*
;

//
================================================================================
====
// FACETS
//
================================================================================
====

FacetsBlock:
    'ORIGIN'
    'FACETS_ENABLED:' '[' facets+=ID (',' facets+=ID)* ']'
;

//
================================================================================
====
// TOP LEVEL ELEMENTS
```

```
//
================================================================================
====

TopLevelElement:
      Layer
    | Chain
    | Wrap
    | Bridge
    | Nest
    | Emerge
    | Finalize
;

//
================================================================================
====
// LAYERS, CLOTHS, SPELLS
//
================================================================================
====

Layer:
    'LAYER' name=ID '{'
        (clothDef=ClothDef)?
        (spellDefs+=SpellDef)*
    '}'
;

ClothDef:
    'CLOTH:' cloth=[Cloth|ID]
;

SpellDef:
    'SPELL:' spell=[Spell|ID]
;

Cloth:
    name=ID
;

Spell:
    name=ID
;
```

```
//
================================================================================
// CHAIN OPERATOR
//
================================================================================

Chain:
    'CHAIN' name=ID '{'
        chainSteps+=ChainStep+
    '}'
;

ChainStep:
    from=[Layer|ID] '->' to=[Layer|ID]
;

//
================================================================================
// WRAP OPERATOR
//
================================================================================

Wrap:
    'WRAP' name=ID '{'
        wrapArrows+=WrapArrow+
    '}'
;

WrapArrow:
    left=[Layer|ID] '<-' right=[Layer|ID]
;

//
================================================================================
// BRIDGE OPERATOR
//
================================================================================
```

```
Bridge:
   'BRIDGE' name=ID '{'
      bridgeLinks+=BridgeLink+
      ('VIA:' viaSpells+=[Spell|ID] (',' viaSpells+=[Spell|ID])*)?
   '}'
;

BridgeLink:
   left=[Layer|ID|Chain|ID] '<->' right=[Layer|ID|Chain|ID]
;

//
================================================================================
// NEST OPERATOR
//
================================================================================

Nest:
   'NEST' name=ID '{'
      'OUTER:' outer=[Wrap|ID|Layer|ID]
      'MIDDLE:' middle=[Wrap|ID|Layer|ID]
      'INNER:' inner=[Wrap|ID|Layer|ID]
      'CORE:' core=[Wrap|ID|Layer|ID]
   '}'
;

//
================================================================================
// EMERGE OPERATOR
//
================================================================================

Emerge:
   'EMERGE' name=ID '{'
      'PRIMARY:' '[' primary+=[Layer|ID|Chain|ID] (',' primary+=[Layer|ID|Chain|ID])* ']'
      'WRAPPED:' '[' wrapped+=[Wrap|ID] (',' wrapped+=[Wrap|ID])* ']'
      'NESTED:' '[' nested+=[Nest|ID] (',' nested+=[Nest|ID])* ']'
      'BRIDGES:' '[' bridges+=[Bridge|ID] (',' bridges+=[Bridge|ID])* ']'
   '}'
```

```
;

//
================================================================================
====
// FINALIZE BLOCK
//
================================================================================
====

Finalize:
    'FINALIZE' system=[Emerge|ID] '{'
        'ENTRY_POINT:' entryPoint=QualifiedRef
        'INITIALIZATION_SEQUENCE:' '[' initSteps+=InitStep (',' initSteps+=InitStep)* ']'
        'CONTINUOUS_OPERATIONS:' '[' contOps+=OpStep (',' contOps+=OpStep)* ']'
        'SAFETY_MONITORS:' '[' safetyOps+=OpStep (',' safetyOps+=OpStep)* ']'
        'SCALING_TRIGGERS:' '[' scalingOps+=OpStep (',' scalingOps+=OpStep)* ']'
        'OUTPUT_MODES:' '[' outputModes+=OutputMode (',' outputModes+=OutputMode)* ']'
        'SUPPORTED_DOMAINS:' '[' supportedDomains+=STRING (','
supportedDomains+=STRING)* ']'
    '}'
;

InitStep:
    QualifiedRef '->' description=STRING
;

OpStep:
    QualifiedRef '->' description=STRING
;

OutputMode:
    mode=STRING '->' target=[Layer|ID]
;

//
================================================================================
====
// QUALIFIED REFERENCES
//
================================================================================
====

QualifiedRef:
```

```
    ref=[Spell|ID] ('.' sub=ID)?
;
```