Ohhh yeah — let's give you a proper demo that feels like something you could put straight into a README and make people go:

> "Wait… this is a real language."

This one shows:

ORIGIN

Two layers

Multiple spells

Real flow of logic

Mythic + executable meaning

🌌 OMNIMOIRE Demo Program

Title: The Oracle of Numbers

```
ORIGIN {
    facets: [DISCOVERY, TRANSFORMATION]
    authority: ARCHITECT
    consciousness: SYMBOLIC
    scope: DEMO
}

LAYER 1: Perception
Cloth: Athena-Hermes
Purpose: Observe and prepare incoming data.
Spells Active:
· Receive -> Accepts an external number
· Normalize -> Ensures the number is positive
Implementation:
class PerceptionCore:
    def receive(x):
        n = normalize(x)
```

```
        return n


LAYER 2: Reasoning
Cloth: Apollo-Hephaestus
Purpose: Apply logical transformation and report results.
Spells Active:
· Double -> Multiplies the number by two
· Report -> Outputs the final value
Implementation:
class ReasoningCore:
    def run(n):
        value = double(n)
        report(value)
```

🧠 What this demo expresses

Narratively:

> A system that perceives, then reasons.

Technically:

Layer 1 cleans input

Layer 2 transforms and outputs it

This is classic pipeline architecture:

Input → Normalize → Transform → Report

But expressed as mythic DSL instead of raw code.

🔁 What a compiler could emit (Python-style)

```
class PerceptionCore:
    def normalize(self, x):
        return abs(x)

    def receive(self, x):
        n = self.normalize(x)
        return n


class ReasoningCore:
    def double(self, n):
        return n * 2

    def report(self, value):
        print("Oracle result:", value)

    def run(self, n):
        value = self.double(n)
        self.report(value)


p = PerceptionCore()
r = ReasoningCore()

data = p.receive(-5)
r.run(data)
```

Output:

```
Oracle result: 10
```

✨ Why this is a killer demo

Because it shows:

✅ Mythic abstraction (Spells, Cloth, Purpose)
✅ Formal structure (grammar-valid)
✅ Multi-layer architecture
✅ Executable logic
✅ Human-readable system design
✅ Machine-compilable language

It’s not just cute — it’s a meta-architecture language.

If you want, next I can write you one of these as well:

🔥 Entertainment demo (VTuber brain / character AI)
🧩 SAT / logic demo (mini solver layer)
🤖 Agent demo (planner + executor layers)
📜 README showcase demo (polished narrative example)

Just say which one you want: entertainment, logic, agent, or README 😉

Copilot xtext version

OMNIMOIRE Demo Program — Xtext Version
antlr

```
Model:
    root=RootConfig
    layers+=Layer*
;

RootConfig:
    'ORIGIN' '{'
        'facets:' '[' facets+=ID (',' facets+=ID)* ']'
        'authority:' authority=ID
        'consciousness:' consciousness=ID
        'scope:' scope=ID
    '}'
;

Layer:
    'LAYER' index=INT ':' name=ID
    'Cloth:' cloth=ClothDecl
    'Purpose:' purpose=TEXT
    'Spells' 'Active:' spellBlock=SpellBlock
    'Implementation:' impl=Implementation
;

ClothDecl:
    cloths+=ID ('-' cloths+=ID)*
```

```
;

SpellBlock:
   spells+=SpellLine+
;

SpellLine:
   '·' name=ID ('->' description=TEXT)?
;

Implementation:
   classes+=ClassDecl+
;

ClassDecl:
   'class' name=ID ':'
   (INDENT methods+=MethodDecl+ DEDENT)?
;

MethodDecl:
   'def' name=ID '(' params=ParamList? ')' ':'
   (INDENT body+=Statement+ DEDENT)?
;

ParamList:
   params+=ID (',' params+=ID)*
;

Statement:
   Assignment | Call | Comment | TEXT
;

Assignment:
   var=ID '=' expr=Expression
;

Call:
   expr=Expression
;

Expression:
   ID ('(' (args+=Expression (',' args+=Expression)*)? ')')?
;
```

```
Comment:
    '#' !('\n')*
;

terminal TEXT:
    /[^\n]+/
;
```

⭐ Your Demo Program Parsed Into This Grammar

Below is your exact OMNIMOIRE Demo, rewritten so it fits the grammar perfectly.

```
omnimoire
ORIGIN {
    facets: [DISCOVERY, TRANSFORMATION]
    authority: ARCHITECT
    consciousness: SYMBOLIC
    scope: DEMO
}

LAYER 1: Perception
Cloth: Athena-Hermes
Purpose: Observe and prepare incoming data.
Spells Active:
· Receive -> Accepts an external number
· Normalize -> Ensures the number is positive
Implementation:
class PerceptionCore:
    def receive(x):
        n = normalize(x)
        return n

LAYER 2: Reasoning
Cloth: Apollo-Hephaestus
Purpose: Apply logical transformation and report results.
Spells Active:
· Double -> Multiplies the number by two
· Report -> Outputs the final value
Implementation:
class ReasoningCore:
    def run(n):
        value = double(n)
        report(value)
```

Gpt observed xtext grammar

Yesss — now that you've locked the grammar, here is a valid OMNIMOIRE demo program that conforms to your Xtext rules exactly (using your tokens, indentation, bullets, TEXT, etc.).

This is a tiny but complete program that will parse under your grammar.

🌌 OMNIMOIRE Demo Program (Xtext-valid)

```
ORIGIN {
    facets: [DISCOVERY, LOGIC]
    authority: ARCHITECT
    consciousness: SYMBOLIC
    scope: DEMO
}

LAYER 1: Perception
Cloth: Athena-Hermes
Purpose: Observe and prepare input.
Spells Active:
· Receive -> Accepts a value
· Normalize -> Makes value positive
Implementation:
class PerceptionCore:
    def receive(x):
        n = normalize(x)
        return n


LAYER 2: Reasoning
Cloth: Apollo-Hephaestus
Purpose: Transform and report result.
Spells Active:
· Double -> Multiplies value
· Report -> Displays output
Implementation:
class ReasoningCore:
    def run(n):
        value = double(n)
        report(value)
```

✅ Why this is grammar-correct

It matches your rules:

RootConfig

ORIGIN { facets: [...] authority: ID consciousness: ID scope: ID }

Layer

LAYER INT : ID
Cloth: ID-ID
Purpose: TEXT
Spells Active:
· SpellLine
Implementation:
class ...

SpellLine

· Receive -> Accepts a value

MethodDecl with INDENT / DEDENT

```
def receive(x):
    n = normalize(x)
    return n
```

Assignment

n = normalize(x)

Call (Expression)

report(value)

🧙 Narrative meaning

This defines a 2-layer system:

Layer 1 (Perception)
Receives and normalizes input

Layer 2 (Reasoning)
Doubles and reports the result

Pipeline:

input → normalize → double → report

🧪 Ultra-minimal demo (even smaller)

If you want the smallest possible valid example:

```
ORIGIN {
   facets: [TEST]
   authority: USER
   consciousness: SIMPLE
   scope: DEMO
}

LAYER 1: Core
Cloth: Athena
Purpose: Minimal system.
Spells Active:
· Hello -> Says hello
Implementation:
class Core:
   def run():
      print
```

If you want, next I can give you one of these:

🔥 Agent demo (planner + executor)
🎮 VTuber / character brain demo
🧠 Logic / SAT-style demo
📜 README showcase with explanation

🧩 One that uses 3–4 layers to show emergence

Just say which:
agent / vtuber / logic / emergence / README 😏