

```
"""
Grimoire Codex — Global Adaptive Infrastructure System
Python translation of the original JavaScript docx builder.
Requires: pip install python-docx
"""
```

```
from docx import Document
from docx.shared import Pt, RGBColor, Inches, Twips
from docx.enum.text import WD_ALIGN_PARAGRAPH
from docx.enum.table import WD_ALIGN_VERTICAL, WD_TABLE_ALIGNMENT
from docx.oxml.ns import qn
from docx.oxml import OxmlElement
from docx.enum.style import WD_STYLE_TYPE
import copy
```

```
# — Color palette
```

```
—
```

```
class C:
```

```
    MIDNIGHT = (0x0D, 0x11, 0x17)
    DEEP     = (0x16, 0x1B, 0x22)
    ACCENT   = (0x1F, 0x4E, 0x79)
    GOLD    = (0xC9, 0xA8, 0x4C)
    SILVER  = (0x8B, 0x9D, 0xC3)
    EMBER   = (0xC0, 0x39, 0x2B)
    TEAL    = (0x1A, 0x6B, 0x5E)
    LIGHT   = (0xE8, 0xF0, 0xFE)
    WHITE   = (0xFF, 0xFF, 0xFF)
    DIVIDER = (0x2E, 0x40, 0x57)
    PALE    = (0xF0, 0xF4, 0xFF)
    PALE2   = (0xEA, 0xF4, 0xF0)
    PALE3   = (0xFF, 0xF8, 0xE7)
```

```
def rgb(t): return RGBColor(*t)
def hex6(t): return "{:02X}{:02X}{:02X}".format(*t)
```

```
# — Low-level XML helpers
```

```
def set_cell_bg(cell, color_tuple):
    """Set cell background shading via XML."""
    tc = cell._tc
```

```

tcPr = tc.get_or_add_tcPr()
shd = OxmlElement("w:shd")
shd.set(qn("w:val"), "clear")
shd.set(qn("w:color"), "auto")
shd.set(qn("w:fill"), hex6(color_tuple))
# Remove any existing shd
for s in tcPr.findall(qn("w:shd")):
    tcPr.remove(s)
tcPr.append(shd)

def set_cell_border(cell, color_tuple=(0x2E, 0x40, 0x57), size=4):
    """Set all four borders on a cell."""
    tc = cell._tc
    tcPr = tc.get_or_add_tcPr()
    for existing in tcPr.findall(qn("w:tcBorders")):
        tcPr.remove(existing)
    borders = OxmlElement("w:tcBorders")
    for side in ("top", "left", "bottom", "right"):
        el = OxmlElement(f"w:{side}")
        el.set(qn("w:val"), "single")
        el.set(qn("w:sz"), str(size))
        el.set(qn("w:space"), "0")
        el.set(qn("w:color"), hex6(color_tuple))
        borders.append(el)
    tcPr.append(borders)

def set_cell_no_border(cell):
    """Remove all cell borders."""
    tc = cell._tc
    tcPr = tc.get_or_add_tcPr()
    for existing in tcPr.findall(qn("w:tcBorders")):
        tcPr.remove(existing)
    borders = OxmlElement("w:tcBorders")
    for side in ("top", "left", "bottom", "right"):
        el = OxmlElement(f"w:{side}")
        el.set(qn("w:val"), "none")
        el.set(qn("w:sz"), "0")
        el.set(qn("w:space"), "0")
        el.set(qn("w:color"), "FFFFFF")
        borders.append(el)
    tcPr.append(borders)

```

```

def set_cell_margins(cell, top=80, bottom=80, left=120, right=120):
    """Set cell internal padding in twips."""
    tc = cell._tc
    tcPr = tc.get_or_add_tcPr()
    for existing in tcPr.findall(qn("w:tcMar")):
        tcPr.remove(existing)
    mar = OxmlElement("w:tcMar")
    for side, val in [("top", top), ("left", left), ("bottom", bottom), ("right", right)]:
        el = OxmlElement(f"w:{side}")
        el.set(qn("w:w"), str(val))
        el.set(qn("w:type"), "dxa")
        mar.append(el)
    tcPr.append(mar)

def set_cell_width(cell, width_dxa):
    """Set explicit cell width in DXA twips."""
    tc = cell._tc
    tcPr = tc.get_or_add_tcPr()
    for existing in tcPr.findall(qn("w:tcW")):
        tcPr.remove(existing)
    tcW = OxmlElement("w:tcW")
    tcW.set(qn("w:w"), str(width_dxa))
    tcW.set(qn("w:type"), "dxa")
    tcPr.insert(0, tcW)

def set_para_spacing(para, before=80, after=80):
    pPr = para._p.get_or_add_pPr()
    spacing = OxmlElement("w:spacing")
    spacing.set(qn("w:before"), str(before))
    spacing.set(qn("w:after"), str(after))
    for s in pPr.findall(qn("w:spacing")):
        pPr.remove(s)
    pPr.append(spacing)

def set_para_indent(para, left=720):
    pPr = para._p.get_or_add_pPr()
    ind = OxmlElement("w:ind")
    ind.set(qn("w:left"), str(left))
    for i in pPr.findall(qn("w:ind")):
        pPr.remove(i)

```

```

pPr.append(ind)

def set_para_bottom_border(para, color_tuple, size=4):
    pPr = para._p.get_or_add_pPr()
    for existing in pPr.findall(qn("w:pBdr")):
        pPr.remove(existing)
    pBdr = OxmLElement("w:pBdr")
    bot = OxmLElement("w:bottom")
    bot.set(qn("w:val"), "single")
    bot.set(qn("w:sz"), str(size))
    bot.set(qn("w:space"), "1")
    bot.set(qn("w:color"), hex6(color_tuple))
    pBdr.append(bot)
    pPr.append(pBdr)

def set_table_column_widths(table, widths_dxa):
    """Set column widths on a table via tblGrid."""
    tbl = table._tbl
    # Remove existing tblGrid
    for g in tbl.findall(qn("w:tblGrid")):
        tbl.remove(g)
    grid = OxmLElement("w:tblGrid")
    for w in widths_dxa:
        col = OxmLElement("w:gridCol")
        col.set(qn("w:w"), str(w))
        grid.append(col)
    # Insert after tblPr if it exists, else at start
    tblPr = tbl.find(qn("w:tblPr"))
    if tblPr is not None:
        tblPr.addnext(grid)
    else:
        tbl.insert(0, grid)

def set_table_width(table, width_dxa=9360):
    """Set total table width."""
    tbl = table._tbl
    tblPr = tbl.find(qn("w:tblPr"))
    if tblPr is None:
        tblPr = OxmLElement("w:tblPr")
        tbl.insert(0, tblPr)
    for existing in tblPr.findall(qn("w:tblW")):

```

```
tblPr.remove(existing)
tblW = OxmlElement("w:tblW")
tblW.set(qn("w:w"), str(width_dxa))
tblW.set(qn("w:type"), "dxa")
tblPr.append(tblW)
```

FULL = 9360 # content width in DXA (US Letter, 1" margins each side)

— Document-level helpers

```
def add_heading(doc, text, level=1, color=None, bottom_border_color=None):
    """Add a heading paragraph."""
    style_name = f"Heading {level}"
    para = doc.add_paragraph(style=style_name)
    set_para_spacing(para, before=400 if level == 1 else 300, after=160 if level == 1 else 120)
    run = para.add_run(text)
    run.bold = True
    run.font.name = "Arial"
    run.font.size = Pt(18 if level == 1 else 14 if level == 2 else 12)
    if color:
        run.font.color.rgb = rgb(color)
    else:
        default_colors = {1: C.MIDNIGHT, 2: C.ACCEPT, 3: C.TEAL}
        run.font.color.rgb = rgb(default_colors.get(level, C.MIDNIGHT))
    if bottom_border_color:
        set_para_bottom_border(para, bottom_border_color, size=8)
    return para
```

```
def add_body(doc, text, italic=False, bold=False, color=None):
    """Add a normal body paragraph."""
    para = doc.add_paragraph()
    set_para_spacing(para, 80, 80)
    run = para.add_run(text)
    run.font.name = "Arial"
    run.font.size = Pt(10)
    run.italic = italic
    run.bold = bold
    if color:
        run.font.color.rgb = rgb(color)
    return para
```

```

def add_mono(doc, text):
    """Add a monospaced code line, indented."""
    para = doc.add_paragraph()
    set_para_spacing(para, 60, 60)
    set_para_indent(para, 720)
    run = para.add_run(text)
    run.font.name = "Courier New"
    run.font.size = Pt(9)
    run.font.color.rgb = rgb(C.ACCEPT)
    return para

def add_code_block(doc, lines):
    """Add a list of monospaced lines as a code block."""
    for line in lines:
        add_mono(doc, line)

def add_space(doc, n=1):
    """Add n blank spacer paragraphs."""
    for _ in range(n):
        para = doc.add_paragraph()
        set_para_spacing(para, 40, 40)

def add_ruler(doc):
    """Add a horizontal rule paragraph."""
    para = doc.add_paragraph()
    set_para_spacing(para, 80, 80)
    set_para_bottom_border(para, C.SILVER, size=4)

# — Banner helpers

```

```

def add_title_banner(doc):
    """Full-width dark title banner."""
    table = doc.add_table(rows=1, cols=1)
    set_table_width(table, FULL)
    set_table_column_widths(table, [FULL])
    cell = table.cell(0, 0)
    set_cell_bg(cell, C.MIDNIGHT)

```

```

set_cell_no_border(cell)
set_cell_margins(cell, 200, 200, 300, 300)
set_cell_width(cell, FULL)

# Star line
p1 = cell.paragraphs[0]
p1.alignment = WD_ALIGN_PARAGRAPH.CENTER
set_para_spacing(p1, 0, 0)
r1 = p1.add_run("★ GRIMOIRE CODEX ★")
r1.bold = True; r1.font.name = "Arial"; r1.font.size = Pt(14)
r1.font.color.rgb = rgb(C.GOLD)

# Main title
p2 = OxmLElement("w:p")
cell._tc.append(p2)
pPr2 = OxmLElement("w:pPr")
jc = OxmLElement("w:jc"); jc.set(qn("w:val"), "center"); pPr2.append(jc)
spacing = OxmLElement("w:spacing"); spacing.set(qn("w:before"), "80")
spacing.set(qn("w:after"), "0"); pPr2.append(spacing)
p2.append(pPr2)
r2_el = OxmLElement("w:r")
rPr2 = OxmLElement("w:rPr")
bold2 = OxmLElement("w:b"); rPr2.append(bold2)
color2 = OxmLElement("w:color"); color2.set(qn("w:val"), hex6(C.WHITE));
rPr2.append(color2)
sz2 = OxmLElement("w:sz"); sz2.set(qn("w:val"), "36"); rPr2.append(sz2)
font2 = OxmLElement("w:rFonts"); font2.set(qn("w:ascii"), "Arial"); rPr2.append(font2)
r2_el.append(rPr2)
t2 = OxmLElement("w:t"); t2.text = "GLOBAL ADAPTIVE INFRASTRUCTURE SYSTEM";
r2_el.append(t2)
p2.append(r2_el)

# Subtitle
p3 = OxmLElement("w:p")
cell._tc.append(p3)
pPr3 = OxmLElement("w:pPr")
jc3 = OxmLElement("w:jc"); jc3.set(qn("w:val"), "center"); pPr3.append(jc3)
sp3 = OxmLElement("w:spacing"); sp3.set(qn("w:before"), "80"); sp3.set(qn("w:after"), "0");
pPr3.append(sp3)
p3.append(pPr3)
r3_el = OxmLElement("w:r")
rPr3 = OxmLElement("w:rPr")
i3 = OxmLElement("w:i"); rPr3.append(i3)
col3 = OxmLElement("w:color"); col3.set(qn("w:val"), hex6(C.SILVER)); rPr3.append(col3)

```

```

sz3 = OxmLElement("w:sz"); sz3.set(qn("w:val"), "20"); rPr3.append(sz3)
f3 = OxmLElement("w:rFonts"); f3.set(qn("w:ascii"), "Arial"); rPr3.append(f3)
r3_el.append(rPr3)
t3 = OxmLElement("w:t")
t3.text = "Root Rune: ORIGIN · Full Codex Composition · Chronicle Audit Trail"
r3_el.append(t3)
p3.append(r3_el)

```

```

def add_section_banner(doc, title, subtitle=None, fill=None):
    """Colored section header banner."""
    if fill is None:
        fill = C.ACCEPT
    table = doc.add_table(rows=1, cols=1)
    set_table_width(table, FULL)
    set_table_column_widths(table, [FULL])
    cell = table.cell(0, 0)
    set_cell_bg(cell, fill)
    set_cell_no_border(cell)
    set_cell_margins(cell, 120, 120, 200, 200)
    set_cell_width(cell, FULL)

    p1 = cell.paragraphs[0]
    set_para_spacing(p1, 0, 0)
    r1 = p1.add_run(title)
    r1.bold = True; r1.font.name = "Arial"; r1.font.size = Pt(13)
    r1.font.color.rgb = rgb(C.WHITE)

    if subtitle:
        p2 = OxmLElement("w:p")
        cell._tc.append(p2)
        pPr2 = OxmLElement("w:pPr")
        sp2 = OxmLElement("w:spacing"); sp2.set(qn("w:before"), "0"); sp2.set(qn("w:after"), "0");
        pPr2.append(sp2)
        p2.append(pPr2)
        r2 = OxmLElement("w:r")
        rPr2 = OxmLElement("w:rPr")
        i2 = OxmLElement("w:i"); rPr2.append(i2)
        c2 = OxmLElement("w:color"); c2.set(qn("w:val"), hex6(C.LIGHT)); rPr2.append(c2)
        sz2 = OxmLElement("w:sz"); sz2.set(qn("w:val"), "18"); rPr2.append(sz2)
        f2 = OxmLElement("w:rFonts"); f2.set(qn("w:ascii"), "Arial"); rPr2.append(f2)
        r2.append(rPr2)
        t2 = OxmLElement("w:t"); t2.text = subtitle; r2.append(t2)
        p2.append(r2)

```

— Table builder

```
def add_table(doc, headers, rows, widths):
    """Build a styled data table with header row."""
    n_cols = len(headers)
    table = doc.add_table(rows=1 + len(rows), cols=n_cols)
    set_table_width(table, FULL)
    set_table_column_widths(table, widths)

    # Header row
    hrow = table.rows[0]
    for i, (text, w) in enumerate(zip(headers, widths)):
        cell = hrow.cells[i]
        set_cell_bg(cell, C.ACCEPT)
        set_cell_border(cell, C.WHITE, size=4)
        set_cell_margins(cell, 100, 100, 140, 140)
        set_cell_width(cell, w)
        cell.vertical_alignment = WD_ALIGN_VERTICAL.CENTER
        p = cell.paragraphs[0]
        p.alignment = WD_ALIGN_PARAGRAPH.CENTER
        set_para_spacing(p, 0, 0)
        run = p.add_run(text)
        run.bold = True; run.font.name = "Arial"; run.font.size = Pt(9)
        run.font.color.rgb = rgb(C.WHITE)

    # Data rows
    for ri, row_data in enumerate(rows):
        bg = C.PALE if ri % 2 == 0 else C.WHITE
        trow = table.rows[ri + 1]
        for i, (text, w) in enumerate(zip(row_data, widths)):
            cell = trow.cells[i]
            set_cell_bg(cell, bg)
            set_cell_border(cell, C.DIVIDER, size=1)
            set_cell_margins(cell, 80, 80, 120, 120)
            set_cell_width(cell, w)
            p = cell.paragraphs[0]
            set_para_spacing(p, 0, 0)
            run = p.add_run(text)
            run.font.name = "Arial"; run.font.size = Pt(9)
            run.font.color.rgb = rgb(C.MIDNIGHT)
```

```

return table

def add_divider(doc):
    """Silver horizontal rule via paragraph border."""
    para = doc.add_paragraph()
    set_para_spacing(para, 80, 80)
    set_para_bottom_border(para, C.SILVER, size=4)

# — Page setup


---




---




---




---


def setup_page(doc):
    section = doc.sections[0]
    section.page_width = Twips(12240)
    section.page_height = Twips(15840)
    section.left_margin = Inches(0.75)
    section.right_margin = Inches(0.75)
    section.top_margin = Inches(0.75)
    section.bottom_margin = Inches(0.75)

#


---




---




---




---


# CONTENT SECTIONS
#


---




---




---




---


def build_section_0(doc):
    """ROOT RUNE INVOCATION"""
    add_section_banner(doc, "SECTION 0 — ROOT RUNE INVOCATION: ORIGIN",
                      "The primordial gate. All composition begins here.", C.MIDNIGHT)
    add_space(doc)
    add_body(doc, ("ORIGIN is the Root Rune. Its invocation opens the full combinatorial space
of "
                  "the Grimoire Codex. No spell, cloth, operator, or module may be instantiated "
                  "prior to this declaration. ORIGIN does not itself produce behavior — it is the "
                  "invariant precondition for all that follows."), italic=True)
    add_space(doc)

```

```

add_code_block(doc, [
    "ORIGIN {",
        " version: GRIMOIRE_CODEX_v1",
        " scope: GLOBAL",
        " facets: [SELF_HEALING, SECURITY, RESOURCE_FLOW, TEMPORARY_BOOST,",
            " CUSTOM_MODULES, PERSISTENCE, OVERDRIVE, ADAPTIVE_TOOLS,",
            " MODE_SWITCHING, HARDWARE_ADAPTION, STATE_TRANSFER,",
            " RESILIENCE_SCALING, ADAPTIVE_RECOVERY, UNIQUE_MODULE,",
            " SURVEILLANCE_INSIGHT, COUNTERMEASURE, TIME_MANAGEMENT,",
            " REMOTE_CONTROL, TRANSFORMATION, ENERGY_MANAGEMENT,",
            " STRATEGIC_PLANNING, DEFENSIVE_RECOVERY, SHAPE SHIFTING,",
            " PERFORMANCE_BOOST, FUNCTION_TRIGGER, MODULAR_SCALING,",
            " AREA_EFFECT, HUB_RECOVERY, HIGH_IMPACT, NETWORK_MAPPING,",
            " ADAPTIVE_DEFENSE, HIGH_PRIORITY, RESOURCE_MANIPULATION,",
            " LAYERED_ABSTRACTION, SYSTEM_SHOCK, SUMMONING,
        TOOL_MODULE,",
            " AGGREGATED_POWER, TIME_CONTROL, CONDITIONAL_BUFF,",
            " PREDICTIVE_INSIGHT, ASSISTANT_AI, NEURAL_INTERFACE,",
            " BURST_MODE, INSTANT_SOLVE, LOGIC_MODULE, UPGRADE_SYSTEM,",
            " EXPONENTIAL_SCALING, ENHANCEMENTS, ARCHETYPE_MAPPING,",
            " LAYERED_DEFENSE, PERSISTENCE_LOOP, LABYRINTH_LOGIC,",
            " VISUAL_SHIELD, DIVINE_MODULES, SONIC_INPUT, WEAK_POINT,",
            " TASK_ORCHESTRATION, COLLECTIVE_INTELLIGENCE, FOCUS_FILTER,",
            " THREAT_CONTAINMENT, INNOVATION_NODE, INFRASTRUCTURE,",
            " CYCLICAL_STATE, DATA_FLOW, DATA_TRANSFER, CLARITY_ENGINE,",
            " PRECISION_QUERY, CREATION_NODE, STRATEGIC_CORE,
        STRESS_TEST,",
            " FLOW_SYSTEM, STABILITY_CORE, RESOURCE_GROWTH, ROOT_NODE,",
            " ORDER_MANAGEMENT, CHAOS_ENGINE, ENFORCEMENT_CYCLE,",
            " PREDICTION_SYSTEM, EXCEPTION_HANDLER, PROPHETIC_NODE,",
            " FAIL_SAFE, TRANSFER_NODE, LIFECYCLE_CONTROL,
        ADAPTIVE_LEARNING,",
            " RECURSIVE_DEPTH, ENTANGLED_SYNC, VOID_PRUNING,
        ETERNAL_LOOP",
            " COMPASSION_NODE, SAFETY_BOUND, CONNECTION_TREE,",
            " DIVINE_MAPPING, DIVINE_NETWORK, TRUST_CHAIN, ANGELIC_ORDER,",
            " EMANATION_STACK, HIERARCHY_NODE, ENERGY_FLOW,
        CIRCULATION_PATH",
            " ENERGY_CORE, ENERGY_LAYER, TOTEM_MODULE, DUAL_NODE,",
            " SPIRIT_NODE, RESONANCE_ENGINE, SPIN_RESET, TRUTH_KERNEL,",
            " REVELATION_NODE, WORD_ENGINE, MIRROR_LOGIC,
        INSPIRATION_CORE",
            " CREATION_GRID, INSPIRATION_ENGINE, ALCHEMY_NODE,",
            " CONTINUITY_NODE, BALANCE_LAW, TEMPORAL_NODE, DUAL_FLOW",
    }
]
)

```

```

        "      FRACTAL_MIRROR, BRIDGE_NODE, INFINITY_KERNEL, TRIAD_LOGIC",
        "      GATEWAY_NODE, NUMERICAL_SAFETY",
        "      CLOTH_STANDARD, CLOTH_MAX, CLOTH_ULTRA, CLOTH_FUSED,
CLOTH_META]",
    " audit: CHRONICLE_MODE_ON",
    "}",
]
)

```

```

def build_section_1(doc):
    """GLOBAL ARCHITECTURE OVERVIEW"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 1 — GLOBAL ARCHITECTURE: OPERATIONAL
OVERVIEW",
                       "Scope, domain hierarchy, consensus topology, and facet activation map",
C.ACCENT)
    add_space(doc)
    add_heading(doc, "1.1 — System Domains and Scope", level=2)
    add_body(doc, ("This system integrates four primary operational domains — Climate
Monitoring, "
                  "Real-Time Logistics, Distributed Finance, and Autonomous Energy Distribution — "
                  "across a three-tier node hierarchy operating in geopolitically diverse trust "
                  "zones. All composition flows exclusively from spells and cloths named within "
                  "the Grimoire Codex, governed by enabled facets declared under ORIGIN."))
    add_space(doc)
    add_table(doc,
              ["Domain", "Primary Spell", "Cloth Binding", "Recovery Spell", "Pattern Tag"],
              [
                  ["Climate Monitoring",     "Insighta",   "Aurora Ultra",    "Vitalis",
"PREDICTIVE_INSIGHT"],
                  ["Real-Time Logistics",   "Fluxa",       "Aquarius",       "Regena",
"RESOURCE_FLOW"],
                  ["Distributed Finance",   "Byzantium",   "Cerberus Ultra",  "Absorbus",
"TRUST_CHAIN"],
                  ["Autonomous Energy Dist.", "Energex",     "Helios Ultra",    "Healix",
"ENERGY_MANAGEMENT"],
              ],
              [2400, 1560, 1720, 1560, 2120],
)
    add_space(doc)
    add_heading(doc, "1.2 — Three-Tier Node Hierarchy", level=2)
    add_table(doc,
              ["Tier", "Role", "Spells Assigned", "Cloth", "Recovery Policy", "Escalation"],
              [

```

```

        ["PRIMARY", "Strategic Control", "Athena · Zephyrus · Decisus · Ultima", "Minerva Ultra", "Preserva → Chronom", "DIRECT → HUMAN"],  

        ["SECONDARY", "Regional Optimization", "Fluxa · Morphis · Counteria · Clarivis",  

        "Griffin Max", "Vitalis → Regena", "→ PRIMARY x3"],  

        ["TERTIARY", "Local Execution", "Telek · Magica · Summona · Samsara",  

        "Pegasus Std", "Healix → Samsara", "→ SECONDARY x1"],  

        ],  

        [1100, 1480, 2520, 1480, 1680, 1100],  

    )  

    add_space(doc)  

    add_heading(doc, "1.3 — Geopolitical Trust Zone Map", level=2)  

    add_body(doc, ("The Byzantine consensus layer (Byzantium spell) partitions nodes into trust domains."  

        "Cross-domain calls require consensus quorum  $\geq 2/3$  of active nodes per zone."  

        "Ashara provides integrity verification at every zone boundary."))  

    add_table(doc,  

        ["Zone ID", "Region", "Trust Level", "Consensus Spell", "Boundary Guard"],  

        [  

            ["Z-1", "North America / EU", "HIGH", "Byzantium", "Ashara + Sphinxa"],  

            ["Z-2", "Asia-Pacific", "HIGH", "Byzantium", "Ashara + Medusia"],  

            ["Z-3", "Latin America / Africa", "MEDIUM", "Covenara", "Ashara + Fortifera"],  

            ["Z-4", "Middle East / Central Asia", "MEDIUM", "Covenara", "Ashara + Armora"],  

            ["Z-5", "Isolated / Contested", "LOW", "Confidara", "Sphinx + Icarion"],  

            ["Z-SIM", "Simulation / Alien Domain", "VIRTUAL", "Koantra", "Pandora + Dionysia"],  

        ],  

        [900, 1860, 1000, 1560, 2040],  

    )
)

```

```

def build_section_2(doc):  

    """BYZANTINE CONSENSUS"""  

    add_divider(doc); add_space(doc)  

    add_section_banner(doc, "SECTION 2 — BYZANTINE-RESILIENT DISTRIBUTED  

    CONSENSUS",  

        "Operator Composition: WRAP · CHAIN · LAYER · BRIDGE", C.TEAL)  

    add_space(doc)  

    add_heading(doc, "2.1 — Consensus Core Module", level=2)  

    add_body(doc, ("WRAP(Byzantium) encapsulates the consensus engine. CHAIN binds  

    Covenara and Ashara "  

        "for mutual trust handshaking. BRIDGE connects inter-zone routing. LAYER applies  

    "  

        "Sphinx challenge-response as the outermost verification shell."))  

    add_space(doc)  

    add_code_block(doc, [

```

```

"MODULE: ConsensusCore",
"",
"WRAP(Byzantium) {",
"  consensus_algo: PAXOS_BFT",
"  quorum:      0.667",
"  zones:       [Z-1, Z-2, Z-3, Z-4, Z-5]",
"  tick_ms:     500",
"} AS consensus_engine",
"",
"CHAIN(Covenara → Ashara) {",
"  handshake_spell: Covenara",
"  integrity_spell: Ashara",
"  chain_depth:   per_zone_boundary",
"} AS trust_handshake",
"",
"BRIDGE(Z-1 ↔ Z-2 ↔ Z-3 ↔ Z-4 ↔ Z-5) {",
"  relay_spell:   Hermesia",
"  protocol:     mTLS + Ashara signature",
"} AS cross_zone_bridge",
"",
"LAYER(Sphinx) OVER consensus_engine {",
"  auth_rounds: 3",
"  adaptive:   true",
"} AS verified_consensus",
"",
"FINALIZE(verified_consensus) {",
"  invariant_checks: [quorum_met, zone_integrity, no_byzantine_majority]",
"  failure_mode:   ISOLATE_ZONE → RE ELECT → ESCALATE_TO_PRIMARY",
"}",
])
add_space(doc)
add_heading(doc, "2.2 — Trust Escalation and Recovery Path", level=2)
add_table(doc,
  ["Failure Condition", "Trigger Spell", "Recovery Operator", "Escalation Spell"],
  [
    ["Zone quorum < 2/3", "Icarion", "REFLECT", "Counter → Zephyrus"],
    ["Integrity breach", "Medusia", "WRAP(Absorbus)", "Fortifera → Trojanis"],
    ["Byzantine node detected", "Counter", "LAYER", "Inferna + Ashara"],
    ["Cross-zone link failure", "Kinetis", "BRIDGE", "Teleportis → Hermesia"],
  ],
  [2200, 1700, 1700, 1760],
)

```

```

def build_section_3(doc):
    """CLIMATE MONITORING"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 3 — CLIMATE MONITORING LAYER",
                       "Operator Composition: NEST · CYCLE · EMERGE · REFLECT", C.ACCENT)
    add_space(doc)
    add_heading(doc, "3.1 — Real-Time Climate Sensor Pipeline", level=2)
    add_body(doc, ("Climate data is ingested every 500ms via Fluxa (resource flow) and passed through "
                  "Insighta (predictive analytics) to detect anomalies. Clarivis provides real-time "
                  "visualization overlays. NEST embeds Dreama for layered environment
simulation."))
    add_code_block(doc, [
        "MODULE: ClimatePipeline",
        "",
        "NEST(Dreama) {",
        "  layers: [live_sensor, historical_model, simulation_zone]",
        "  inner_spell: Fluxa",
        "  outer_spell: Dreamara",
        "} AS climate_environment",
        "",
        "CYCLE(Insighta, interval_ms=500) {",
        "  input: climate_environment.live_sensor",
        "  model: Oedipha",
        "  threshold_engine: dynamic",
        "  anomaly_spell: Medusia",
        "} AS climate_predictor",
        "",
        "REFLECT(Clarivis) ONTO climate_predictor {",
        "  viz_spell: Apollara",
        "  alert_spell: Echo",
        "} AS climate_monitor",
        "",
        "EMERGE(climate_environment, climate_predictor, climate_monitor) {",
        "  negotiation: Relata",
        "  delegation: Telek",
        "  combine_spell: Aggrega",
        "} AS climate_emergent_layer",
        "",
        "FINALIZE(climate_emergent_layer) {",
        "  invariants: [latency_lt_500ms, no_data_gap, alert_delivered]",
        "  failure_mode: Regena",
        "}",
        ],
    ])

```

```

add_space(doc)
add_heading(doc, "3.2 — Anomaly Countermeasure Binding", level=2)
add_table(doc,
    ["Anomaly Event", "Trigger Threshold", "Countermeasure Spell", "Escalation"],
    [
        ["Temperature spike",      "> 2σ dynamic",  "Counter", "Ultima → PRIMARY tier"],
        ["CO2 flux deviation", "> 1.5σ",       "Pandora", "Oracula predictive lock"],
        ["Storm pattern emergence", "> 3σ",       "Echo",   "Forcea → regional nodes"],
        ["Sea-level anomaly",      "static +5cm",   "Insighta", "Oedipha causal trace"],
        ["Sensor network partition", "loss > 20%",  "Kinetis", "Samsara restart cluster"],
    ],
    [2200, 1560, 1700, 1900],
)

```

```

def build_section_4(doc):
    """LOGISTICS ENGINE"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 4 — REAL-TIME LOGISTICS ENGINE",
                       "Operator Composition: CHAIN · EVOLVE · LAYER · BRIDGE", C.TEAL)
    add_space(doc)
    add_heading(doc, "4.1 — Logistics Routing Core", level=2)
    add_body(doc, ("Logistics routes recalculate every 500ms. Labyrintha resolves recursive
routing "
                  "problems. Chronomanta reorders task queues on delay events. Shiftara switches "
                  "between routing modes dynamically."))
    add_code_block(doc, [
        "MODULE: LogisticsEngine",
        "",
        "CHAIN(Fluxa → Labyrintha → Shiftara) {",
        " flow_spell: Fluxa",
        " solver_spell: Labyrintha",
        " switch_spell: Shiftara",
        " recalc_ms: 500",
        "} AS route_solver",
        "",
        "LAYER(Chronomanta) OVER route_solver {",
        " scheduler: Crona",
        " reorder_spell: Chronomanta",
        " priority_spell: Angelica",
        "} AS timed_route_engine",
        "",
        "EVOLVE(timed_route_engine) WITH Metalearnara {",
        " learning_spell: Metalearnara",
    ])

```

```

    " feedback_spell: Karmalis",
    " version_spell: Evolia",
    "} AS adaptive_logistics",
    "",
    "BRIDGE(adaptive_logistics ↔ climate_emergent_layer) {",
    " bridge_spell: Poseida",
    " handoff_spell: Ferrana",
    "} AS climate_logistics_bridge",
    "",
    "FINALIZE(adaptive_logistics) {",
    " invariants: [route_valid, no_loop, max_hops_lt_12, latency_lt_500ms]",
    " failure_mode: Teleportis → Portalus",
    "}",
    ]
)
add_space(doc)
add_heading(doc, "4.2 — Mode Switching Decision Table", level=2)
add_table(doc,
  ["Condition", "Active Spell", "Mode Switch (Shiftara)", "Recovery"],
  [
    ["Climate event detected", "Fluxa", "→ ground-safe route", "Regena"],
    ["Port closure", "Counter", "→ air reroute", "Teleportis"],
    ["Geopolitical zone closed", "Confidara", "→ neutral corridor", "Morphis"],
    ["Supply surplus (node)", "Bioflux", "→ push to deficit zone", "Energos"],
    ["Demand spike > 40%", "Fortis", "→ overdrive mode", "Energex"],
  ],
  [2000, 1700, 2060, 1600],
)

```

```

def build_section_5(doc):
    """DISTRIBUTED FINANCE"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 5 — DISTRIBUTED FINANCE MODULE",
                      "Operator Composition: WRAP · NEST · LAYER · CHAIN · REFLECT",
C.ACCEPT)
    add_space(doc)
    add_heading(doc, "5.1 — Finance Consensus and Settlement", level=2)
    add_body(doc, ("Distributed finance relies on Byzantium for multi-party settlement
consensus."
                  "Revela handles encrypted ledger states. Nemesia enforces fairness and bias "
                  "correction. Pyroxis automates compliance audit cycles."))
    add_code_block(doc, [
        "MODULE: DistributedFinance",
        ""],

```

```

"WRAP(Byzantium) {" ,
" domain: FINANCE",
" ledger_spell: Revela",
" audit_spell: Pyroxis",
"} AS finance_consensus",
"",
"NEST(Inferna) {" ,
" tiers: 9",
" inner: finance_consensus",
" outer_spell: Fortifera",
"} AS finance_vault",
"",
"LAYER(Nemnesia) OVER finance_vault {" ,
" equity_spell: Nemnesia",
" ethics_spell: Ahimsa",
" fairness_spell: Ma'atara",
"} AS ethical_finance",
"",
"CHAIN(Transmutare → Netheris → Hermesia) {" ,
" convert_spell: Transmutare",
" archive_spell: Netheris",
" relay_spell: Hermesia",
"} AS settlement_pipeline",
"",
"REFLECT(Apollara) ONTO ethical_finance {" ,
" viz_spell: Apollara",
" integrity_check: Ashara",
"} AS finance_monitor",
"",
"FINALIZE(ethical_finance) {" ,
" invariants: [equity_preserved, audit_logged, no_double_spend, consent_valid]",
" failure_mode: Pandoria",
"}",
})
add_space(doc)
add_heading(doc, "5.2 — Cross-Chain Dependencies", level=2)
add_table(doc,
  ["Finance Event", "Upstream Dependency", "Downstream Effect", "Integrity Spell"],
  [
    ["Supply chain payment", "LogisticsEngine", "Triggers Energex", "Ashara"],
    ["Climate levy trigger", "ClimatePipeline", "Equity redistribution", "Nemnesia"],
    ["Energy credit trade", "EnergyDistribution", "Settlement finalize", "Byzantium"],
    ["Node failure (finance)", "ConsensusCore", "Re-elect settlement", "Pandoria"],
    ["Geopolitical freeze", "Zone Z-5 isolation", "Escrow hold", "Icarion"],
  ]
)

```

```

        ],
        [1900, 1900, 2060, 1500],
    )
}

def build_section_6(doc):
    """ENERGY DISTRIBUTION"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 6 — AUTONOMOUS ENERGY DISTRIBUTION",
        "Operator Composition: CYCLE · EVOLVE · EMERGE · LAYER · WRAP",
        C.MIDNIGHT)
    add_space(doc)
    add_heading(doc, "6.1 — Energy Core Module", level=2)
    add_body(doc, ("Energex drives overdrive mode for high-demand bursts. Energos manages
CPU/GPU "
        "compute allocation. Qiflow routes distributed power. Gaiana enforces sustainable "
        "computing constraints. Icarion acts as safety limiter against overload."))
    add_code_block(doc, [
        "MODULE: EnergyDistribution",
        "",
        "WRAP(Energos) {",
        "    allocation_spell: Energos",
        "    routing_spell: Qiflow",
        "    circulation: Qiara",
        "    safety_spell: Icarion",
        "} AS energy_pool",
        "",
        "CYCLE(Energex, interval_ms=500) {",
        "    trigger: demand_gt_threshold",
        "    overdrive_spell: Energex",
        "    cooldown_spell: Defendora",
        "    balancer_spell: Bioflux",
        "} AS energy_cycle",
        "",
        "LAYER(Gaiana) OVER energy_cycle {",
        "    eco_spell: Gaiana",
        "    target: carbon_neutral_per_zone",
        "    override_rule: HUMAN_APPROVAL_REQUIRED if carbon_overshoot",
        "} AS green_energy",
        "",
        "EVOLVE(green_energy) WITH Spirala {",
        "    scale_spell: Spirala",
        "    cloud_spell: Demetra",
        "    version_spell: Evovlia",
    ])

```

```

"} AS adaptive_energy",
"",
"EMERGE(adaptive_energy, energy_pool, energy_cycle) {",
" negotiation: Relata",
" delegation: Forcea",
" aggregate: Aggrega",
"} AS energy_emergent",
"",
"FINALIZE(energy_emergent) {",
" invariants: [no_overload, carbon_limit_met, local_authority_preserved]",
" failure_mode: Hydrina → Samsara",
"}",
])
add_space(doc)
add_heading(doc, "6.2 — Energy State Transition Table", level=2)
add_table(doc,
  ["State", "Trigger Spell", "Action", "Recovery"],
  [
    ["NORMAL", "Energos", "Standard allocation via Qiflow", "—"],
    ["SURGE", "Fortis", "Activate Energex overdrive", "Defendora cooldown"],
    ["OVERDRIVE", "Energex", "Overdrive burst amplification", "Icarion limiter"],
    ["SCARCITY", "Vitalis Max.", "Scale buffers; Gaiana conservation mode", "Wuven self-adjust"],
    ["BLACKOUT", "Kinetis", "Samsara restart; Hydrina spawn backup", "Heartha restore hub"],
    ["ALIEN_ZONE", "Koantra", "Pandora risk containment; Dreamara sim", "Pandoria fail-safe"],
  ],
  [1400, 1500, 2860, 1600],
)

```

```

def build_section_7(doc):
    """ETHICS LAYER"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 7 — ETHICS, POLICY & HUMAN OVERRIDE LAYER",
                      "Operator Composition: LAYER · WRAP · REFLECT · CHAIN", C.EMBER)
    add_space(doc)
    add_heading(doc, "7.1 — Ethics Enforcement Module", level=2)
    add_body(doc, ("Ethics constraints are woven into every decision tier via Ahimsa (harm minimization), "
                  "Dharmara (purpose enforcement), Compassa (compassion-driven logic), Nemnesia (fairness), "

```

"and Ma'atara (compliance audit). Human override triggers are implemented via Antigona "

```
        "(exception handler) with Heraia (role-based governance).")  
add_code_block(doc, [  
    "MODULE: EthicsLayer",  
    "",  
    "LAYER(Ahimsa) UNIVERSAL {",  
    "  harm_spell: Ahimsa",  
    "  purpose_spell: Dharmara",  
    "  compassion_spell: Compassa",  
    "} AS harm_guard",  
    "",  
    "WRAP(Heraia) {",  
    "  rbac_spell: Heraia",  
    "  root_spell: Zephyrus",  
    "  order_spell: Ma'atara",  
    "} AS governance_shell",  
    "",  
    "CHAIN(Antigona → Decisus → Athena) {",  
    "  exception_spell: Antigona",  
    "  buffer_spell: Decisus",  
    "  strategy_spell: Athena",  
    "} AS human_override_chain",  
    "",  
    "REFLECT(Karmalis) ONTO governance_shell {",  
    "  karma_spell: Karmalis",  
    "  audit_spell: Pyroxis",  
    "} AS ethics_audit",  
    "",  
    "FINALIZE(harm_guard) {",  
    "  invariants: [no_harm_action_without_override,",  
    "    equity_constraints_met,",  
    "    environmental_targets_logged,",  
    "    human_override_auditible]",  
    "  failure_mode: Pandoria → Heraia",  
    "}",  
])  
add_space(doc)  
add_heading(doc, "7.2 — Human Override Decision Flow", level=2)  
add_table(doc,  
    ["Override Type", "Initiating Spell", "Evaluation Spell", "Resolution Spell", "Audit"],  
    [  
        ["Ethical breach", "Ahimsa veto", "Athena", "Antigona", "Karmalis + Pyroxis"],  
        ["Carbon overshoot", "Gaiana alert", "Decisus", "Heraia", "Ma'atara"],  
    ]
```

```

        ["Equity violation", "Nemesia flag", "Athena", "Compassa", "Karmalis"],
        ["Emergency stop", "Ultima trigger", "Zephyrus", "Antigona", "Chronom + Ashara"],
        ["Zone authority claim", "Dharmara check", "Arcanum", "Heraia", "Ashara"],
    ],
    [1700, 1560, 1360, 1460, 1880],
)

```

```

def build_section_8(doc):
    """ANOMALY ENGINE"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 8 — PREDICTIVE ANOMALY DETECTION &
COUNTERMEASURE SYSTEM",
        "Operator Composition: CYCLE · NEST · EMERGE · CHAIN", C.TEAL)
    add_space(doc)
    add_heading(doc, "8.1 — Anomaly Detection Core", level=2)
    add_code_block(doc, [
        "MODULE: AnomalyEngine",
        "",
        "NEST(Insighta) {",
        "    inner_spell: Insighta",
        "    outer_spell: Clarivis",
        "    historical_spell: Chronom",
        "} AS anomaly_detector",
        "",
        "CYCLE(anomaly_detector, interval_ms=500) {",
        "    threshold_spell: Vitalis Maxima",
        "    learning_spell: Metalearnara",
        "    stream_spell: Poseida",
        "} AS live_anomaly_cycle",
        "",
        "CHAIN(live_anomaly_cycle → Counter → Fortifera → Echo) {",
        "    detect_spell: live_anomaly_cycle",
        "    counter_spell: Counter",
        "    harden_spell: Fortifera",
        "    broadcast_spell: Echo",
        "} AS countermeasure_chain",
        "",
        "EMERGE(anomaly_detector, countermeasure_chain) {",
        "    negotiation: Relata",
        "    containment: Trojanis",
        "    isolation: Medusia",
        "} AS emergentContainment",
        ""
    ])

```

```

    "FINALIZE(emergentContainment) {",
    "  invariants: [alert_within_500ms, no_false_positive_cascade,",
    "    countermeasure_audited, threshold_recalculated]",
    "  failure_mode: Regena → Pandora",
    "}",
  ],
)
add_space(doc)
add_heading(doc, "8.2 — Dynamic Threshold Table", level=2)
add_table(doc,
  ["Signal Domain", "Base Threshold", "Dynamic Expansion Spell", "Trigger Countermeasure"],
  [
    ["Climate temperature", "2σ from 30d mean", "Vitalis Maxima", "Counter + Echo"],
    ["Logistics latency", "> 450ms per hop", "Fluxa burst", "Shiftara reroute"],
    ["Finance settlement", "deviation > 0.5%", "Insighta", "Nemesia + Ashara"],
    ["Energy load", "95% capacity", "Icarion", "Energex + Defendora"],
    ["Consensus failure", "quorum < 2/3", "Counter", "Kinetis + Byzantium"],
  ],
  [1900, 1900, 1760, 1800],
)

```

```

def build_section_9(doc):
    """EMERGENT BEHAVIOR"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 9 — EMERGENT BEHAVIOR: NODE NEGOTIATION & DELEGATION",
        "Operator Composition: EMERGE · EVOLVE · CYCLE · BRIDGE", C.GOLD)
    add_space(doc)
    add_heading(doc, "9.1 — Node Negotiation Protocol", level=2)
    add_body(doc, ("Emergent behavior is governed by EMERGE operators binding Relata (relationship graph),",
        "Forcea (distributed command), Aggrega (power aggregation), and Adaptis (adaptive",
        "capabilities). Nodes may negotiate supply/demand balancing, delegate tasks, and combine",
        "capabilities without violating local authority constraints (Dharmara, Heraia)."))
    add_code_block(doc, [
        "MODULE: EmergentBehavior",
        "",
        "EMERGE(ConsensusCore, EnergyDistribution, LogisticsEngine, ClimatePipeline) {",
        "  graph_spell: Relata",
        "  command_spell: Forcea",
        "  combine_spell: Aggrega",
    ])

```

```

    " copy_spell: Adaptis",
    " authority_guard: Dharmara",
    "} AS global_negotiator",
    "",
    "EVOLVE(global_negotiator) WITH Arcanum {",
    " archetype_spell: Arcanum",
    " balance_spell: Equilibria",
    " flow_spell: Wuven",
    "} AS adaptive_negotiator",
    "",
    "CYCLE(adaptive_negotiator, interval_ms=500) {",
    " supply_spell: Fluxa",
    " demand_spell: Bioflux",
    " routing_spell: Poseida",
    " balance_spell: Taora",
    "} AS live_negotiation",
    "",
    "BRIDGE(live_negotiation ↔ EthicsLayer) {",
    " gate_spell: Ahimsa",
    " audit_spell: Karmalis",
    "} AS ethical_emergence",
    "",
    "FINALIZE(ethical_emergence) {",
    " invariants: [local_authority_not_overridden, ethics_cleared,
    "           recalc_within_500ms, no_infinite_delegation_loop]",
    " failure_mode: Icarion → Sisyphea",
    "}",
    ])
}

def build_section_10(doc):
    """SIMULATION LAYER"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 10 — SIMULATION LAYER: ALIEN & POST-HUMAN
    DOMAIN TESTING",
        "Operator Composition: NEST · WRAP · EMERGE · LAYER · EVOLVE",
    C.MIDNIGHT)
    add_space(doc)
    add_heading(doc, "10.1 — Simulation Module", level=2)
    add_body(doc, ("The Z-SIM zone activates a nested Dreamara (generative world model)
    inside Dreama "
        "(virtual container hierarchy). Koantra provides non-linear paradox-based reasoning
    "
        "for alien-domain scenarios. Dionyssa injects procedural chaos. Fractala enables "

```

```

        "recursive self-similar scaling for post-human computational depth."))

add_code_block(doc, [
    "MODULE: SimulationLayer",
    "",
    "NEST(Dreamara) INSIDE Dreama {",
    " outer:      Dreama",
    " inner:      Dreamara",
    " chaos_spell:  Dionyssa",
    " risk_spell:   Pandora",
    "} AS simulation_world",
    "",
    "WRAP(Koantra) {",
    " logic_spell:  Koantra",
    " solver_spell: Solva",
    " evolve_spell: Metalearnara",
    "} AS alien_reasoner",
    "",
    "LAYER(Fractala) OVER simulation_world {",
    " fractal_spell: Fractala",
    " mirror_spell: Mirorra",
    " growth_spell: Spirala",
    "} AS recursive_sim",
    "",
    "EMERGE(simulation_world, alien_reasoner, recursive_sim) {",
    " containment: Trojanis",
    " risk_control: Pandora",
    " recovery_spell: Pandoria",
    "} AS alien_domain",
    "",
    "EVOLVE(alien_domain) WITH Eternara {",
    " eternal_spell: Eternara",
    " cycle_spell: Samsara",
    "} AS simulation_evolution",
    "",
    "FINALIZE(simulation_evolution) {",
    " invariants: [sim_isolated_from_production, risk_logged,
        " no_alien_logic_bleeds_to_live_system]",
    " failure_mode: Nirvara → production isolation",
    "}",
    ],
)
add_space(doc)
add_heading(doc, "10.2 — Alien Domain Scenario Coverage", level=2)
add_table(doc,
    ["Scenario", "Simulation Spell", "Reasoning Spell", "Containment"],

```

```

[  

    ["Post-human compute topology", "Dreamara", "Koantra", "Trojanis + Pandora"],  

    ["Post-biological sensor arrays", "Dreama", "Fractala", "Pandoria"],  

    ["Disconnected zone (dark net)", "Dionyssa", "Labyrinthha", "Medusia + Icarion"],  

    ["Infinite scaling edge case", "Spirala", "Eternara", "Nirvara (final state)"],  

    ["Zero-trust Byzantine collapse", "Koantra", "Byzantium", "Ashara + Counter"],  

],  

[2200, 1560, 1560, 2040],  

)

```

```

def build_section_11(doc):  

    """CLOTH BINDINGS"""  

    add_divider(doc); add_space(doc)  

    add_section_banner(doc, "SECTION 11 — CLOTH SYSTEM: META-COMPOSITION  

BINDINGS",  

        "Standard · Max · Ultra · Fused · Tri-Fused · Meta Cloth Activations", C.ACCENT)  

    add_space(doc)  

    add_heading(doc, "11.1 — Active Cloth Assignments by Module", level=2)  

    add_table(doc,  

        ["Module", "Cloth", "Tier", "Composite Motif", "Binding Operator"],  

        [  

            ["ConsensusCore", "Cerberus Ultra", "Ultra", "Multi-Layer Security",  

"WRAP"],  

            ["ClimatePipeline", "Aurora Ultra", "Ultra", "Insight + Diagnostics",  

"CYCLE + EMERGE"],  

            ["LogisticsEngine", "Chimera-Hydra (Fused)", "Fused", "Fusion + Redundancy",  

"CHAIN + EVOLVE"],  

            ["DistributedFinance", "Phoenix-Cerberus (Fused)", "Fused", "Self-repairing  

security", "NEST + LAYER"],  

            ["EnergyDistribution", "Helios Ultra", "Ultra", "High-Capacity Distribution",  

"WRAP + CYCLE"],  

            ["EthicsLayer", "Nemesis Ultra", "Ultra", "Balance / Policy Enforcement",  

"AYER"],  

            ["AnomalyEngine", "Griffin Ultra", "Ultra", "Vigilance / Monitoring",  

"NEST + CHAIN"],  

            ["EmergentBehavior", "Chimera-Argonauta-Hydra", "Tri-Fused", "Collaborative  

Self-Healing", "EMERGE + BRIDGE"],  

            ["SimulationLayer", "Janus-Valkyrie-Pandora", "Tri-Fused", "Duality + Rescue +  

Risk", "NEST + EVOLVE"],  

            ["Global Orchestrator", "Pegasus-Phoenix-Hydra-Aurora", "Meta", "Hyper-resilient  

Auto-Healing", "FINALIZE"],  

        ],  

        [1760, 2100, 900, 2000, 1600],
    )

```

```

)
add_space(doc)
add_heading(doc, "11.2 — Global Orchestrator Meta Cloth", level=2)
add_body(doc, ("The entire system is bound under a single Meta cloth:  

Pegasus-Phoenix-Hydra-Aurora "
"(Speed + Rebirth + Regeneration + Insight). This Meta cloth governs the root  

FINALIZE "
"operator, ensuring the system recovers dynamically, predicts load surges, and  

maintains "
"real-time operational awareness across all domains."))

add_code_block(doc, [
    "CLOTH[META]: Pegasus-Phoenix-Hydra-Aurora",
    " motif: Speed + Rebirth + Regeneration + Insight",
    " function: Hyper-resilient, predictive auto-healing",
    " real_world: Self-optimizing distributed microservices",
    " pattern_tag: Dimensional Resilience",
    " use_case: Systems recover dynamically while predicting load surges",
    " binding: FINALIZE(ALL_MODULES)",
])
)
```

```

def build_section_12(doc):
    """API & DSL"""
    add_divider(doc); add_space(doc)
    add_section_banner(doc, "SECTION 12 — API SURFACE, DSL, AND CROSS-LANGUAGE  

BINDINGS",
        "Codex-derived module interfaces for Python · Rust · Go · React · Kubernetes",
        C.TEAL)
    add_space(doc)
    add_heading(doc, "12.1 — Codex DSL Grammar (Xtext-compatible)", level=2)
    add_code_block(doc, [
        "// GrimoireDSL.xtext — Root Grammar",
        "grammar org.grimoire.codex.GrimoireDSL",
        "",
        "ORIGIN ::= 'ORIGIN' '{' FacetList AuditMode '}'",
        "FacetList ::= 'facets:' '[' ID (',' ID)* ']'",
        "AuditMode ::= 'audit:' ('CHRONICLE_MODE_ON' | 'CHRONICLE_MODE_OFF'),
        "",
        "Module ::= 'MODULE:' ID '{' Operator+ 'FINALIZE(' ID ')' '}'",
        "",
        "Operator ::= WrapOp | ChainOp | LayerOp | NestOp | BridgeOp",
        "           | ReflectOp | EvolveOp | CycleOp | EmergeOp",
        "",
        "WrapOp ::= 'WRAP(' SpellRef ')' '{' Properties '}'",
    ])
)
```

```

"ChainOp ::= 'CHAIN(' SpellRef ('→' SpellRef)+ ')' '{' Properties '}'",
"LayerOp ::= 'LAYER(' SpellRef ')' ('OVER' | 'UNIVERSAL') ID? '{' Properties '}'",
"NestOp ::= 'NEST(' SpellRef ')' ('INSIDE' SpellRef)? '{' Properties '}'",
"BridgeOp ::= 'BRIDGE(' BridgeTarget ')' '{' Properties '}'",
"ReflectOp ::= 'REFLECT(' SpellRef ')' 'ONTO' ID '{' Properties '}'",
"EvolveOp ::= 'EVOLVE(' ID ')' 'WITH' SpellRef '{' Properties '}'",
"CycleOp ::= 'CYCLE(' ID ',' interval_ms=' INT ')' '{' Properties '}'",
"EmergeOp ::= 'EMERGE(' ID (',' ID)* ')' '{' Properties '}'",
"",
"SpellRef ::= ID // Must match Codex spell name exactly",
"BridgeTarget ::= ID ('↔' ID)+",
])
add_space(doc)
add_heading(doc, "12.2 — Python SDK Interface", level=2)
add_code_block(doc, [
    "# grimoire_sdk.py — Auto-generated from Codex composition",
    "from dataclasses import dataclass, field",
    "from typing import List, Optional",
    "",
    "@dataclass",
    "class GrimoireModule:",
    "    name: str",
    "    cloth: str",
    "    spells: List[str]",
    "    _audit: List[dict] = field(default_factory=list)",
    "",
    "    def wrap(self, spell: str, **props) -> 'GrimoireModule': ...",
    "    def chain(self, *spells: str, **props) -> 'GrimoireModule': ...",
    "    def layer(self, spell: str, target=None, **props) -> 'GrimoireModule': ...",
    "    def nest(self, spell: str, inside=None, **props) -> 'GrimoireModule': ...",
    "    def bridge(self, *targets: str, **props) -> 'GrimoireModule': ...",
    "    def reflect(self, spell: str, onto: str, **props) -> 'GrimoireModule': ...",
    "    def evolve(self, with_spell: str, **props) -> 'GrimoireModule': ...",
    "    def cycle(self, interval_ms: int, **props) -> 'GrimoireModule': ...",
    "    def emerge(self, *modules: str, **props) -> 'GrimoireModule': ...",
    "    def finalize(self, invariants: List[str], failure_mode: str): ...",
    "",
    "# Example instantiation — ConsensusCore",
    "consensus = (",
    "    GrimoireModule('ConsensusCore', 'Cerberus Ultra',",
    "        ['Byzantium','Covenara','Ashara','Sphinx']),
    "    .wrap('Byzantium', consensus_algo='PAXOS_BFT', quorum=0.667, tick_ms=500)",
    "    .chain('Covenara', 'Ashara')",
    "    .bridge('Z-1','Z-2','Z-3','Z-4','Z-5', relay_spell='Hermesia')",
]

```

```

    " .layer('Sphinxa', auth_rounds=3, adaptive=True)",
    " .finalize(",
    "     invariants=['quorum_met','zone_integrity','no_byzantine_majority'],",
    "     failure_mode='ISOLATE_ZONE → RE ELECT → ESCALATE_TO_PRIMARY'",
    " ),
    ")",
])
add_space(doc)
add_heading(doc, "12.3 — Rust Module Skeleton", level=2)
add_code_block(doc, [
    "// consensus_core.rs",
    "use grimoire::{Spell, Cloth, Chronicle};",
    "",
    "pub struct ConsensusCore {",
    "    cloth: Cloth,      // Cerberus Ultra",
    "    spells: Vec<Spell>, // [Byzantium, Covenara, Ashara, Sphinxa]",
    "    quorum: f64,        // 0.667",
    "    tick_ms: u64,       // 500",
    "    audit: Chronicle,",
    "}",
    "",
    "impl ConsensusCore {",
    "    pub fn new() -> Self      { /* WRAP(Byzantium) */ ... }",
    "    pub fn tick(&mut self)      { /* CYCLE at 500ms */ ... }",
    "    pub fn verify_zone(&self) -> bool { /* LAYER(Sphinxa) */ ... }",
    "    pub fn bridge(&self, zones: &[&str]) { /* BRIDGE(Hermesia) */ ... }",
    "    pub fn finalize(&self) -> Result<(), GrimoireError> { ... }",
    "}",
],
])
add_space(doc)
add_heading(doc, "12.4 — Kubernetes Manifest (Codex-Annotated)", level=2)
add_code_block(doc, [
    "# k8s/consensus-core.yaml",
    "apiVersion: apps/v1",
    "kind: StatefulSet",
    "metadata:",
    "  name: consensus-core",
    "  annotations:",
    "    grimoire.codex/cloth: 'Cerberus Ultra'",
    "    grimoire.codex/spells: 'Byzantium,Covenara,Ashara,Sphinxa'",
    "    grimoire.codex/operator: 'WRAP+CHAIN+LAYER+BRIDGE'",
    "spec:",
    "  replicas: 7 # Samsara: container rebirth",
    "  template:",

```

```

    " spec:",
    " containers:",
    " - name: consensus",
    "   image: grimoire/consensus-core:latest",
    "   env:",
    "     - { name: QUORUM,   value: '0.667' }",
    "     - { name: TICK_MS,  value: '500'  }",
    "     - { name: SPELL_WRAP, value: 'Byzantium' }",
    "     - { name: SPELL_LAYER, value: 'Sphinxa'  }",
    "   livenessProbe: # Vitalis: self-repair",
    "     httpGet: { path: /health, port: 8080 }",
    "     periodSeconds: 1",
    "   - name: ethics-sidecar",
    "     image: grimoire/ethics-layer:latest",
    "     env:",
    "       - { name: SPELL_LAYER, value: 'Ahimsa,Dharmara' }",
    "       - { name: OVERRIDE,   value: 'Antigona'      }",
  ])
add_space(doc)
add_heading(doc, "12.5 — React Dashboard Component", level=2)
add_code_block(doc, [
  "// GrimoireDashboard.jsx — Clarivis + Apollara",
  "import { useGrimoireCycle } from './hooks/useGrimoireCycle';",
  "",
  "export default function GrimoireDashboard() {",
  " // CYCLE(Insighta, interval_ms=500) — live anomaly feed",
  " const { anomalies, energyState, consensusHealth,",
  "   logisticsRoutes, climateAlerts } = useGrimoireCycle(500);",
  " return (",
  "   <div className='grimoire-dashboard'>",
  "     <ConsensusPanel data={consensusHealth} cloth='Cerberus Ultra' />",
  "     <ClimateFeed alerts={climateAlerts} spell='Insighta' />",
  "     <LogisticsMap routes={logisticsRoutes} spell='Labyrinthia' />",
  "     <EnergyGauge state={energyState} spell='Energex' />",
  "     <EthicsIndicator          spell='Ahimsa' />",
  "     <AnomalyLog   items={anomalies}   spell='Clarivis' />",
  "   </div>",
  " );",
  "}",
])

```

```

def build_section_13(doc):
    """CHRONICLE AUDIT TRAIL"""

```

```

add_divider(doc); add_space(doc)
add_section_banner(doc, "SECTION 13 — CHRONICLE AUDIT TRAIL",
    "Full lineage, reasoning, and predicted failure modes for every module",
C.MIDNIGHT)
add_space(doc)
add_heading(doc, "13.1 — Chronicle Schema", level=2)
add_code_block(doc, [
    "CHRONICLE_ENTRY {",
    " timestamp: ISO-8601",
    " module: STRING",
    " operator:",
    " spell_invoked: STRING",
    " cloth_active: STRING",
    " facet_context: LIST[STRING]",
    " reasoning: STRING",
    " invariants: LIST[STRING]",
    " outcome: ENUM[OK, WARN, FAIL, ESCALATE]",
    " failure_mode: STRING",
    " lineage: LIST[ENTRY_ID]",
    "}",
])
add_space(doc)
add_heading(doc, "13.2 — Sample Chronicle Entries", level=2)
add_table(doc,
    ["Seq", "Module", "Operator", "Spell", "Outcome", "Failure Mode Predicted"],
    [
        ["001", "ORIGIN",      "—",     "ORIGIN",      "OK", "—"],
        ["002", "ConsensusCore", "WRAP",   "Byzantium",   "OK", "Zone isolation → RE_ELECT"],
        ["003", "ConsensusCore", "CHAIN",  "Covenara",    "OK", "Handshake timeout → Kinetis"],
        ["004", "ConsensusCore", "LAYER",  "Sphinxa",     "OK", "Auth fail → Counteria"],
        ["005", "ClimatePipeline", "NEST",   "Dreama",     "OK", "Sensor loss → Regena"],
        ["006", "ClimatePipeline", "CYCLE",  "Insighta",    "OK", "Threshold breach → Echo"],
        ["007", "LogisticsEngine", "CHAIN",  "Fluxa",      "OK", "Latency spike → Shiftara"],
        ["008", "LogisticsEngine", "EVOLVE", "Metalearnara", "OK", "Model stale → Evolvia"]
    ]
)

```

```

        ["009", "DistributedFinance", "WRAP",   "Byzantium",
Pandoria"],                                "OK", "Ledger split →
        ["010", "DistributedFinance", "LAYER",  "Nemesis",
Antigona"],                                "OK", "Equity fail →
        ["011", "EnergyDistribution", "CYCLE",   "Energex",
Icarion"],                                 "OK", "Overload →
        ["012", "EthicsLayer",       "LAYER",    "Ahimsa",
Athena veto"],                             "OK", "Harm detect →
        ["013", "AnomalyEngine",    "CHAIN",    "Counteria",
Pandora"],                                "OK", "Cascade →
        ["014", "EmergentBehavior", "EMERGE",   "Aggrega",
loop → Icarion"],                         "OK", "Delegation
        ["015", "SimulationLayer",  "NEST",     "Dreamara",
Nirvara"],                                "OK", "Bleed to prod →
        ["016", "ALL",             "FINALIZE", "Pegasus-Phoenix-Hydra-Aurora","OK", "See
per-module failure modes"],
        ],
        [500, 1500, 960, 1700, 800, 2900],
)
add_space(doc)

```

add_heading(doc, "13.3 — Operator Invocation Lineage Map", level=2)
add_body(doc, ("Every FINALIZE carries a dependency graph (lineage) referencing all prior entries "

"in the Chronicle. The system is fully reproducible: any operator following the Codex
"

"and this Chronicle can re-derive the identical system from ORIGIN forward."))

```

add_code_block(doc, [
"FINALIZE(ALL_MODULES) {",
" cloth:      Pegasus-Phoenix-Hydra-Aurora [META]",
" lineage:    [001→002→003→004, // ConsensusCore",
"             001→005→006,      // ClimatePipeline",
"             001→007→008,      // LogisticsEngine",
"             001→009→010,      // DistributedFinance",
"             001→011,          // EnergyDistribution",
"             001→012,          // EthicsLayer",
"             001→013,          // AnomalyEngine",
"             001→014,          // EmergentBehavior",
"             001→015]          // SimulationLayer",
" global_invariants: [",
"   all_modules_finalized",
"   ethics_cleared_at_every_layer",
"   chronicle_complete",
"   no_spell_outside_codex",
"   no_cloth_outside_codex",

```

```

    " operator_law_respected",
    " root_rune_ORIGIN_invoked_first",
    " 500ms_recalc_maintained_globally",
    " human_override_accessible",
    " simulation_isolated_from_production",
    "]",
    " system_state: NIRVARA",
    "}",
    ])
)

def add_closing_seal(doc):
    """Full-width closing banner."""
    add_divider(doc); add_space(doc)
    table = doc.add_table(rows=1, cols=1)
    set_table_width(table, FULL)
    set_table_column_widths(table, [FULL])
    cell = table.cell(0, 0)
    set_cell_bg(cell, C.MIDNIGHT)
    set_cell_no_border(cell)
    set_cell_margins(cell, 160, 160, 300, 300)
    set_cell_width(cell, FULL)

    p1 = cell.paragraphs[0]
    p1.alignment = WD_ALIGN_PARAGRAPH.CENTER
    set_para_spacing(p1, 0, 80)
    r1 = p1.add_run("◆ SYSTEM COMPOSITION COMPLETE ◆")
    r1.bold = True; r1.font.name = "Arial"; r1.font.size = Pt(13)
    r1.font.color.rgb = rgb(C.GOLD)

    for text, color in [
        ("Root Rune: ORIGIN · Meta Cloth: Pegasus-Phoenix-Hydra-Aurora · Final State: NIRVARA", C.SILVER),
        ("Chronicle entries: 016 · Modules: 9 · Spells active: 84 · Cloths bound: 10", C.LIGHT),
    ]:
        p = Oxmlelement("w:p")
        cell._tc.append(p)
        pPr = Oxmlelement("w:pPr")
        jc = Oxmlelement("w:jc"); jc.set(qn("w:val"), "center"); pPr.append(jc)
        sp = Oxmlelement("w:spacing"); sp.set(qn("w:before"), "60"); sp.set(qn("w:after"), "0");
        pPr.append(sp)
        p.append(pPr)
        r = Oxmlelement("w:r")
        rPr = Oxmlelement("w:rPr")

```

```
c = OxmLElement("w:color"); c.set(qn("w:val"), hex6(color)); rPr.append(c)
sz = OxmLElement("w:sz"); sz.set(qn("w:val"), "18"); rPr.append(sz)
f = OxmLElement("w:rFonts"); f.set(qn("w:ascii"), "Arial"); rPr.append(f)
i = OxmLElement("w:i"); rPr.append(i)
r.append(rPr)
t = OxmLElement("w:t"); t.text = text; r.append(t)
p.append(r)
```

```
#
```

```
# MAIN
```

```
def main():
    doc = Document()
    setup_page(doc)
```

```
    # Remove default empty paragraph
    for para in doc.paragraphs:
        p = para._element
        p.getparent().remove(p)
```

```
    # Cover
    add_title_banner(doc)
    add_space(doc, 2)
```

```
    # All sections
    build_section_0(doc)
    build_section_1(doc)
    build_section_2(doc)
    build_section_3(doc)
    build_section_4(doc)
    build_section_5(doc)
    build_section_6(doc)
    build_section_7(doc)
    build_section_8(doc)
    build_section_9(doc)
    build_section_10(doc)
    build_section_11(doc)
    build_section_12(doc)
```

```
build_section_13(doc)
add_closing_seal(doc)

out = "/mnt/user-data/outputs/Grimoire_Codex_System_Python.docx"
doc.save(out)
print(f"Saved: {out}")

if __name__ == "__main__":
    main()
```