Technical Specification: DWG-to-Database Direct Conversion System

Project: Terminal 1 Automated BIM Pipeline

Date: November 11, 2025

Status: Concept Phase

Target: Convert 2D DWGs directly to spatial database without manual 3D modeling

1. SYSTEM OVERVIEW

1.1 Vision

Build an intelligent system that:

1. Reads 2D DWG floor plans/sections directly

2. Extracts geometry and classifies elements semantically

3. Stores in spatial database (same structure as current IFC-extracted data)

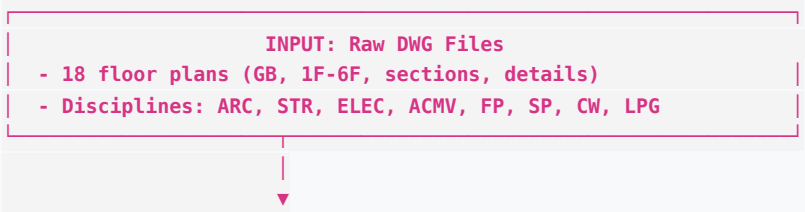
4. Enables parametric transformation tools (shift, extend, gap operations)

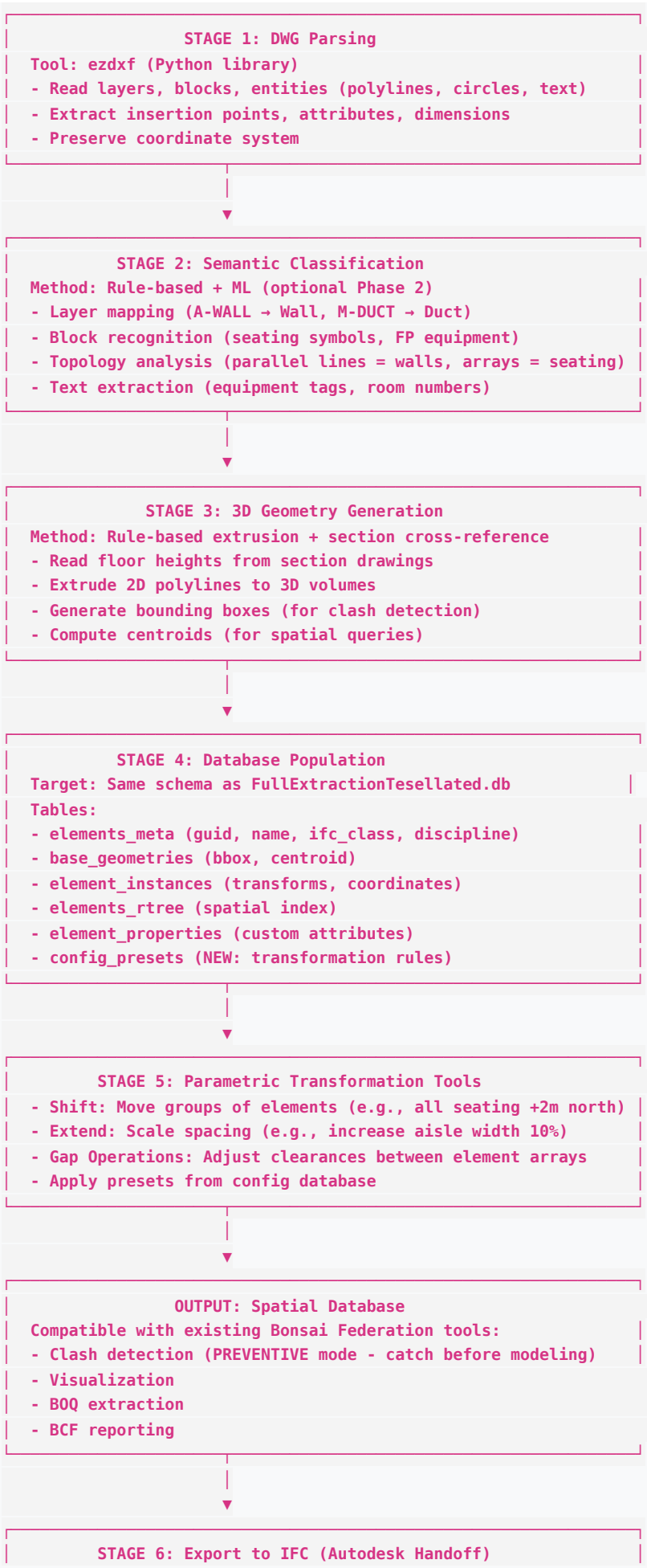
5. Provides designer-editable configuration presets

6. CRITICAL: Clash-free design from the start (no manual modeling = no initial clashes)

7. CRITICAL: Export back to IFC for Autodesk continuation (not a merge, fresh generation)

1.2 Pipeline Architecture




STAGE 1: DWG Parsing
Tool: ezdxf (Python library)
- Read layers, blocks, entities (polylines, circles, text)
- Extract insertion points, attributes, dimensions
- Preserve coordinate system
STAGE 2: Semantic Classification
Method: Rule-based + ML (optional Phase 2)
- Layer mapping (A-WALL → Wall, M-DUCT → Duct)
- Block recognition (seating symbols, FP equipment)
- Topology analysis (parallel lines = walls, arrays = seating)
- Text extraction (equipment tags, room numbers)
STAGE 3: 3D Geometry Generation
Method: Rule-based extrusion + section cross-reference
- Read floor heights from section drawings
- Extrude 2D polylines to 3D volumes
- Generate bounding boxes (for clash detection)
- Compute centroids (for spatial queries)
STAGE 4: Database Population
Target: Same schema as FullExtractionTesellated.db
Tables:
- elements_meta (guid, name, ifc_class, discipline)
- base_geometries (bbox, centroid)
- element_instances (transforms, coordinates)
- elements_rtree (spatial index)
- element_properties (custom attributes)
- config_presets (NEW: transformation rules)
STAGE 5: Parametric Transformation Tools
- Shift: Move groups of elements (e.g., all seating +2m north)
- Extend: Scale spacing (e.g., increase aisle width 10%)
- Gap Operations: Adjust clearances between element arrays
- Apply presets from config database
OUTPUT: Spatial Database
Compatible with existing Bonsai Federation tools:
- Clash detection (PREVENTIVE mode - catch before modeling)
- Visualization
- BOQ extraction
- BCF reporting
STAGE 6: Export to IFC (Autodesk Handoff)

```
|  - Generate discipline-specific IFCs from database              |
|  - IFC4 format (coordinated, clash-free)                        |
|  - Autodesk teams continue detailed design in Revit             |
|  - NOT a merge (fresh IFC generation from clean database)       |
└─────────────────────────────────────────────────────────────────┘
```

1.3 PARADIGM SHIFT: Clash-Free by Design

Key Insight:

Traditional workflow:

```
DWG → Manual modeling in Revit → Export IFC → Clash detection → Fix clashes → Re-export
(3-6 months per terminal, 100+ clash iterations)
```

NEW workflow:

```
DWG → Intelligent database → Parametric coordination → Clash-free validation → Export IFC once
(2-4 weeks per terminal, zero clash iterations in detailed design)
```

Advantages:

1. No Pre-Existing Clashes

- Start from 2D drawings (clash-free by definition)
- 3D generation uses coordination rules from database
- Clearances/spacing enforced during extrusion
- Result: Engineers receive coordinated IFCs, not raw geometry

2. Fresh IFC Generation (Not Merge)

- Traditional: Merge 8 discipline IFCs → resolve conflicts
- NEW: Database → single coordinated export → split to 8 discipline IFCs
- No "who wins" decisions (e.g., wall vs. duct overlap)
- All disciplines share same coordinate system from day 1

3. Autodesk Continuation

- Export database → IFC4 files (one per discipline)
- Revit teams import clean, coordinated geometry
- Detailed design (finishes, connections, specs) added in Revit
- Coordination already done → teams work in parallel

4. Standard Unit Design Library

- Database stores parametric object templates (e.g., "Terminal Seating Unit Type A")

- Each template has:
- Standard dimensions (W×D×H)
- Clearance requirements (front: 900mm, side: 150mm)
- Material specifications
- Cost/schedule data
- Clash-free placement rules
- Designers configure, not model from scratch

---

2. STAGE 1: DWG PARSING

2.1 Tools & Libraries

- ezdxf (Python): Read DWG/DXF files (supports AutoCAD 2000-2024)
- ODA File Converter: Convert DWG → DXF if needed (ezdxf reads DXF natively)

2.2 Data Extraction

Entities to Parse:

```
# Geometry entities
- LWPOLYLINE: Walls, structural outlines, room boundaries
- POLYLINE: Complex paths, MEP routes
- LINE: Beams, columns (simplified)
- CIRCLE: Columns, pipes, equipment (plan view)
- ARC: Curved walls, ducts
- INSERT: Block references (doors, windows, seating, equipment)
- TEXT/MTEXT: Labels, equipment tags, room numbers

# Organizational data
- Layers: Discipline classification (A-WALL, M-DUCT, E-DEVICE)
- Blocks: Reusable symbols (seating groups, FP equipment)
- Attributes: Equipment metadata (tag, type, capacity)
- Dimensions: Size references for validation
```

Key Attributes to Capture:

```
{
    'layer': 'A-WALL',           # Discipline identifier
    'entity_type': 'LWPOLYLINE',  # Geometry type
    'vertices': [(x1,y1), ...],   # Coordinate data
    'elevation': 0.0,             # Z-height from section drawings
    'block_name': 'SEAT-CHAIR',   # For INSERT entities
    'attributes': {               # Block attribute data
        'TAG': 'FP-001',
        'TYPE': 'Fire Extinguisher',
        'CAPACITY': '6kg'
    },
    'text_nearby': ['Gate 12'],   # Associated labels
}
```

2.3 Coordinate System Handling

- Global offset: Store Terminal 1 origin offset (121.474, -21.657, -0.781)

- Unit conversion: DWG units → meters (database standard)
- Rotation: Align north to +Y axis if needed

---

3. STAGE 2: SEMANTIC CLASSIFICATION

3.1 Layer-to-Discipline Mapping

Phase 1: Rule-Based Dictionary

```
LAYER_MAPPING = {
    # Architecture
    'A-WALL': ('ARC', 'IfcWall'),
    'A-DOOR': ('ARC', 'IfcDoor'),
    'A-WIND': ('ARC', 'IfcWindow'),
    'A-SLAB': ('ARC', 'IfcSlab'),
    'A-STAIR': ('ARC', 'IfcStair'),
    'A-FURN': ('ARC', 'IfcFurnishingElement'),
    'A-SEAT': ('ARC', 'IfcFurnishingElement'),  # Seating detection

    # Structure
    'S-COLS': ('STR', 'IfcColumn'),
    'S-BEAM': ('STR', 'IfcBeam'),
    'S-SLAB': ('STR', 'IfcSlab'),
    'S-FNDN': ('STR', 'IfcFooting'),

    # MEP - ACMV
    'M-DUCT': ('ACMV', 'IfcDuctSegment'),
    'M-EQUIP': ('ACMV', 'IfcFlowTerminal'),
    'M-AHU': ('ACMV', 'IfcUnitaryEquipment'),

    # MEP - Fire Protection
    'FP-PIPE': ('FP', 'IfcPipeSegment'),
    'FP-SPRK': ('FP', 'IfcFireSuppressionTerminal'),
    'FP-HYDR': ('FP', 'IfcValve'),

    # MEP - Electrical
    'E-POWER': ('ELEC', 'IfcCableSegment'),
    'E-LIGHT': ('ELEC', 'IfcLightFixture'),
    'E-PANEL': ('ELEC', 'IfcElectricDistributionBoard'),

    # MEP - Plumbing
    'P-PIPE': ('SP', 'IfcPipeSegment'),
    'P-DRAIN': ('SP', 'IfcPipeSegment'),
    'P-FIXT': ('SP', 'IfcSanitaryTerminal'),

    # Chilled Water
    'CW-PIPE': ('CW', 'IfcPipeSegment'),

    # LPG
    'LPG-PIPE': ('LPG', 'IfcPipeSegment'),
}
```

3.2 Block Recognition (Amenities Detection)

Strategy: Pattern Matching + Attribute Reading

```
# Seating detection
SEATING_BLOCKS = [
    'SEAT-CHAIR', 'SEAT-BENCH', 'SEAT-SOFA',
    'WAITING-AREA', 'LOUNGE-SEAT'
]

# Fire protection equipment
FP_BLOCKS = [
    'FP-EXTINGUISHER', 'FP-HOSE-REEL', 'FP-HYDRANT',
    'FP-SPRINKLER-HEAD', 'FP-ALARM', 'FP-PANEL'
]

# Electrical devices
ELEC_BLOCKS = [
    'E-OUTLET', 'E-SWITCH', 'E-PANEL', 'E-LIGHT',
    'E-FIXTURE', 'E-TRANSFORMER', 'E-CHARGING-STATION'
]

# Detection algorithm
def classify_block(block_name, layer, attributes, nearby_text):
    # 1. Exact match
```

```
    if block_name in SEATING_BLOCKS:
        return ('ARC', 'IfcFurnishingElement', 'Seating')

    # 2. Pattern match (fuzzy)
    if 'SEAT' in block_name.upper() or 'CHAIR' in block_name.upper():
        return ('ARC', 'IfcFurnishingElement', 'Seating')

    # 3. Layer context
    if layer.startswith('FP-') or layer.startswith('A-FP'):
        return ('FP', classify_fp_equipment(block_name))

    # 4. Attribute analysis
    if 'TYPE' in attributes and 'FIRE' in attributes['TYPE'].upper():
        return ('FP', 'IfcFireSuppressionTerminal', attributes['TYPE'])

    # 5. Spatial context (nearby text)
    if any('GATE' in txt.upper() for txt in nearby_text):
        return ('ARC', 'IfcSpace', 'Gate Area')

    # Default: unknown
    return ('UNKNOWN', 'IfcBuildingElementProxy', 'Unclassified')
```

3.3 Topology Analysis (Intelligent Guessing)

Pattern Recognition Rules:

```
# 1. SEATING ARRAYS
# Detect: Multiple similar blocks in regular grid pattern
def detect_seating_array(blocks):
    if len(blocks) < 4:
        return None

    # Check spacing consistency
    spacings = [dist(blocks[i], blocks[i+1]) for i in range(len(blocks)-1)]
    if std_deviation(spacings) < 0.1:  # 10cm tolerance
        return {
            'type': 'SeatingArray',
            'count': len(blocks),
            'spacing': mean(spacings),
            'orientation': calculate_grid_angle(blocks),
            'aisle_gaps': detect_wider_gaps(spacings)  # Where people walk
        }

# 2. AMENITY ZONES
# Detect: Clusters of service blocks (toilets, shops, lounges)
def detect_amenity_zones(blocks, rooms):
    zones = []
    for room in rooms:
        services_in_room = [b for b in blocks if point_in_polygon(b.location, room.boundary)]

        if has_plumbing_fixtures(services_in_room):
            zones.append({'type': 'Restroom', 'room': room.id})
        elif has_seating_concentration(services_in_room):
            zones.append({'type': 'Lounge', 'room': room.id})
        elif has_food_equipment(services_in_room):
            zones.append({'type': 'F&B', 'room': room.id})

    return zones

# 3. MEP NETWORK CONNECTIVITY
# Detect: Connected pipe/duct segments forming networks
def trace_mep_network(segments, discipline='FP'):
    networks = []
    unvisited = set(segments)

    while unvisited:
        start = unvisited.pop()
        network = trace_connected_segments(start, unvisited, tolerance=0.05)
        networks.append({
            'discipline': discipline,
            'segments': network,
            'length_total': sum(s.length for s in network),
            'endpoints': find_terminals(network)  # Equipment connections
        })

    return networks

# 4. CIRCULATION PATHS
# Detect: Clear spaces between seating/walls (aisles, corridors)
def detect_circulation(seating_areas, walls, room_boundary):
    # Compute negative space
    occupied = union(seating_areas + walls)
    free_space = room_boundary.difference(occupied)

    # Find corridor-like regions (long, narrow)
    aisles = []
    for region in free_space.polygons:
        aspect_ratio = region.length / region.width
        if aspect_ratio > 3.0:  # Long and narrow
            aisles.append({
                'type': 'Aisle',
                'width': region.width,
                'path': region.skeleton()  # Medial axis for routing
            })

    return aisles
```

4. STAGE 3: 3D GEOMETRY GENERATION

4.1 Floor Height Extraction

Multi-Sheet Cross-Reference:

```
# Parse section drawings to get floor-to-floor heights
def extract_floor_heights(section_dwgs):
    heights = {}

    for dwg in section_dwgs:
        # Find dimension entities measuring vertical distances
        for dim in dwg.query('DIMENSION'):
            if dim.is_vertical() and 'FL' in dim.text.upper():
                floor_level = parse_floor_label(dim.text)  # "1F", "2F", etc.
                heights[floor_level] = dim.measurement

    # Typical terminal building heights
    defaults = {
        'GB': 0.0,      # Ground beam (reference)
        '1F': 5.0,      # Ground floor (high ceiling for check-in)
        '2F': 9.5,      # Departures
        '3F': 13.5,     # Arrivals
        '4F': 17.0,     # Offices/Plant
        '5F': 20.5,     # Rooftop plant
    }

    return {**defaults, **heights}  # Measured overrides defaults
```

4.2 Extrusion Logic

```
def extrude_entity_to_3d(entity_2d, floor_heights, current_floor='1F'):
    # Get floor elevation
    z_base = floor_heights[current_floor]

    # Determine extrusion height based on element type
    if entity_2d.ifc_class == 'IfcWall':
        z_height = floor_heights[next_floor(current_floor)] - z_base
    elif entity_2d.ifc_class == 'IfcSlab':
        z_height = 0.2  # 200mm typical slab thickness
    elif entity_2d.ifc_class == 'IfcColumn':
        z_height = floor_heights[next_floor(current_floor)] - z_base
    elif entity_2d.ifc_class in MEP_TYPES:
        z_height = entity_2d.attributes.get('HEIGHT', 0.3)  # Pipe/duct diameter
    else:
        z_height = 2.5  # Default 2.5m high

    # Generate 3D bounding box (sufficient for clash detection)
    vertices_2d = entity_2d.vertices
    bbox_3d = {
        'min': (min(v[0] for v in vertices_2d),
                min(v[1] for v in vertices_2d),
                z_base),
        'max': (max(v[0] for v in vertices_2d),
                max(v[1] for v in vertices_2d),
                z_base + z_height)
    }

    return bbox_3d
```

4.3 Tessellation (Optional - for Visualization)

```
# Generate detailed mesh only if visualization needed
def tessellate_polyline(vertices_2d, z_base, z_height):
    faces = []

    # Bottom face
    faces.append([(x, y, z_base) for x, y in vertices_2d])
```

```
    # Top face
    faces.append([(x, y, z_base + z_height) for x, y in vertices_2d])

    # Side faces
    for i in range(len(vertices_2d)):
        v1 = vertices_2d[i]
        v2 = vertices_2d[(i + 1) % len(vertices_2d)]
        faces.append([
            (v1[0], v1[1], z_base),
            (v2[0], v2[1], z_base),
            (v2[0], v2[1], z_base + z_height),
            (v1[0], v1[1], z_base + z_height),
        ])

    return faces
```

---

5. STAGE 4: DATABASE POPULATION

5.1 Schema Compatibility

Reuse Existing Tables:

```
-- elements_meta: Core element data
INSERT INTO elements_meta (guid, name, ifc_class, discipline, source_file)
VALUES (?, ?, ?, ?, ?);

-- base_geometries: Bounding boxes for clash detection
INSERT INTO base_geometries (guid, bbox_min_x, bbox_min_y, bbox_min_z,
                             bbox_max_x, bbox_max_y, bbox_max_z,
                             centroid_x, centroid_y, centroid_z)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);

-- element_instances: Placement transforms (mostly identity for DWG import)
INSERT INTO element_instances (guid, matrix_00, matrix_01, ..., matrix_33)
VALUES (?, 1, 0, 0, 0, ...);  -- Identity matrix

-- elements_rtree: Spatial index for fast queries
INSERT INTO elements_rtree (id, minX, minY, minZ, maxX, maxY, maxZ)
VALUES (?, ?, ?, ?, ?, ?, ?);

-- element_properties: Custom attributes from DWG
INSERT INTO element_properties (guid, property_name, property_value)
VALUES (?, 'dwg_layer', ?),
       (?, 'dwg_block', ?),
       (?, 'equipment_tag', ?);
```

5.2 New Table: Configuration Presets

```
-- config_presets: Designer-editable transformation rules
CREATE TABLE config_presets (
    preset_id INTEGER PRIMARY KEY,
    preset_name TEXT NOT NULL,           -- "Terminal Seating Standard"
    target_element_type TEXT,            -- "IfcFurnishingElement"
    target_filter TEXT,                  -- "Seating" (from classification)
    operation TEXT,                      -- "SHIFT" | "EXTEND_GAP" | "SCALE"
    parameters TEXT,                     -- JSON: {"direction": "north", "distance": 2.0}
    description TEXT,
    created_by TEXT,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT 1
);

-- standard_unit_library: Parametric object templates
CREATE TABLE standard_unit_library (
    unit_id INTEGER PRIMARY KEY,
    unit_name TEXT NOT NULL,             -- "Terminal Seating Type A"
    unit_category TEXT,                  -- "Seating" | "FP Equipment" | "Lighting"
    ifc_class TEXT,                      -- "IfcFurnishingElement"
    discipline TEXT,                     -- "ARC" | "FP" | "ELEC"

    -- Standard dimensions (meters)
    width REAL,                          -- 0.55m (single seat)
    depth REAL,                          -- 0.60m
    height REAL,                         -- 0.85m

    -- Clearance requirements (meters)
    clearance_front REAL DEFAULT 0.9,    -- 900mm in front
    clearance_back REAL DEFAULT 0.1,     -- 100mm behind
    clearance_side REAL DEFAULT 0.15,    -- 150mm sides

    -- Design specifications
```

```
    material_spec TEXT,                  -- "Fire-rated fabric, steel frame"
    weight_kg REAL,                      -- 25kg
    load_capacity_kg REAL,               -- 150kg

    -- Cost data
    unit_cost REAL,                      -- $350 per unit
    installation_hours REAL,             -- 0.5 hours

    -- Parametric rules (JSON)
    placement_rules TEXT,                -- {"min_spacing": 0.05, "align_to": "grid"}
    clash_avoidance TEXT,                -- {"keep_clear_zone": 0.3, "avoid_types": ["IfcColumn"]}

    -- Geometry template (reference to base_geometries)
    template_guid TEXT,                  -- Links to geometry in base_geometries table

    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    version INTEGER DEFAULT 1
);

-- unit_instances: Track which standard units are used where
CREATE TABLE unit_instances (
    instance_id INTEGER PRIMARY KEY,
    unit_id INTEGER REFERENCES standard_unit_library(unit_id),
    element_guid TEXT REFERENCES elements_meta(guid),
    custom_parameters TEXT,              -- JSON: overrides to standard dimensions
    placed_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- preset_history: Audit trail for applied transformations
CREATE TABLE preset_history (
    history_id INTEGER PRIMARY KEY,
    preset_id INTEGER REFERENCES config_presets(preset_id),
    applied_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    elements_affected INTEGER,
    applied_by TEXT,
    notes TEXT
);

-- Example presets
INSERT INTO config_presets (preset_name, target_element_type, operation, parameters, description)
VALUES
    ('Increase Aisle Width', 'IfcFurnishingElement', 'EXTEND_GAP',
     '{"gap_type": "aisle", "increase_by": 0.2}',
     'Add 200mm to all aisle widths between seating rows'),

    ('Shift Gate 12 North', 'IfcSpace', 'SHIFT',
     '{"space_name": "Gate 12", "direction": [0, 1, 0], "distance": 3.0}',
     'Move Gate 12 waiting area 3m north'),

    ('Extend FP Coverage', 'IfcFireSuppressionTerminal', 'ARRAY_EXTEND',
     '{"coverage_radius": 7.5, "add_devices": true}',
     'Ensure all areas within 7.5m of fire suppression device');
```

6. STAGE 5: PARAMETRIC TRANSFORMATION TOOLS

6.1 Shift Operations

```

class ShiftTransform:
    """Move groups of elements while preserving spatial relationships"""

    def shift_elements(self, element_guids, direction, distance, database):
        """
        Args:
            element_guids: List of elements to move
            direction: [x, y, z] unit vector
            distance: Meters to move
        """
        offset = [d * distance for d in direction]

        for guid in element_guids:
            # Update bounding box
            cursor.execute("""
                UPDATE base_geometries
                SET bbox_min_x = bbox_min_x + ?,
                    bbox_min_y = bbox_min_y + ?,
                    bbox_min_z = bbox_min_z + ?,
                    bbox_max_x = bbox_max_x + ?,
                    bbox_max_y = bbox_max_y + ?,
                    bbox_max_z = bbox_max_z + ?,
                    centroid_x = centroid_x + ?,
                    centroid_y = centroid_y + ?,
                    centroid_z = centroid_z + ?
                WHERE guid = ?
            """, (*offset, *offset, *offset, guid))

            # Update spatial index
            self.rebuild_rtree_for_elements(element_guids)

            # Check for new clashes
            new_clashes = self.detect_clashes_after_move(element_guids)
            return new_clashes
```

6.2 Gap Extension (Spacing Adjustment)

```
class GapExtensionTransform:
    """Adjust spacing between element arrays (e.g., seating rows)"""

    def extend_gaps(self, seating_array_id, gap_increase, database):
        """
        Args:
            seating_array_id: ID of detected seating pattern
            gap_increase: Additional spacing in meters
        """
        # 1. Query seating array configuration
        array = self.get_seating_array(seating_array_id, database)
        rows = array['rows']  # List of element GUIDs per row

        # 2. Calculate cumulative offsets
        offsets = []
        cumulative = 0.0
        for i, row in enumerate(rows):
            offsets.append(cumulative)
            if i < len(rows) - 1:  # Not the last row
                cumulative += gap_increase

        # 3. Apply shifts row-by-row
        for row, offset in zip(rows, offsets):
            direction = array['perpendicular_direction']  # Aisle direction
            self.shift_elements(row, direction, offset, database)

        # 4. Update array metadata
        self.update_array_spacing(seating_array_id, gap_increase, database)
```

6.3 Preset Application UI

```
class PresetManager:
    """Apply designer-configured transformation presets"""

    def apply_preset(self, preset_id, database, dry_run=True):
        """
        Args:
            preset_id: Configuration preset to apply
            dry_run: If True, preview changes without committing
        """
        # Load preset configuration
        preset = self.load_preset(preset_id, database)

        # Query target elements
        elements = self.query_elements_by_filter(
            preset['target_element_type'],
            preset['target_filter'],
            database
        )

        # Parse operation parameters
        params = json.loads(preset['parameters'])

        # Execute transformation
        if preset['operation'] == 'SHIFT':
            result = ShiftTransform().shift_elements(
                elements,
                params['direction'],
                params['distance'],
                database
            )
        elif preset['operation'] == 'EXTEND_GAP':
            result = GapExtensionTransform().extend_gaps(
                elements,
                params['increase_by'],
                database
            )

        # Preview changes in Blender
        if dry_run:
            self.visualize_before_after(elements, result)
            return {
                'affected_count': len(elements),
                'new_clashes': result.get('new_clashes', []),
                'preview': True
            }
        else:
            # Commit to database
            database.commit()
            self.log_preset_application(preset_id, len(elements))
            return {'applied': True, 'elements': len(elements)}
```

7. STAGE 6: DATABASE-TO-IFC EXPORT

7.1 Export Strategy

Goal: Generate clean, coordinated IFC4 files from database for Autodesk teams

```
class DatabaseToIfcExporter:
    """Export spatial database to discipline-specific IFC files"""

    def export_by_discipline(self, database_path, output_dir, disciplines=None):
        """
        Generate one IFC file per discipline (or merged IFC if requested)

        Args:
            database_path: Path to FullExtractionTesellated.db
            output_dir: Where to save IFC files
            disciplines: List ['ARC', 'STR', 'ELEC'] or None for all
        """
        if disciplines is None:
            disciplines = self.get_all_disciplines(database_path)

        for disc in disciplines:
            ifc_file = self.create_ifc_structure(disc)
            elements = self.query_elements_by_discipline(disc, database_path)

            for element in elements:
                ifc_entity = self.create_ifc_entity(element, ifc_file)
                self.add_geometry(ifc_entity, element, ifc_file)
                self.add_properties(ifc_entity, element, ifc_file)

            output_path = f"{output_dir}/Terminal1-{disc}-Coordinated.ifc"
            ifc_file.write(output_path)
            print(f"✓ Exported {len(elements)} {disc} elements to {output_path}")
```

7.2 IFC Structure Generation

```
def create_ifc_structure(self, discipline):
    """Create IFC4 file with proper spatial hierarchy"""

    file = ifcopenshell.file(schema='IFC4')

    # Units
    units = file.createIfcUnitAssignment([
        file.createIfcSIUnit(None, "LENGTHUNIT", None, "METRE")
    ])

    # Coordinate system
    origin = file.createIfcAxis2Placement3D(
        file.createIfcCartesianPoint((0.0, 0.0, 0.0)),
        file.createIfcDirection((0.0, 0.0, 1.0)),
        file.createIfcDirection((1.0, 0.0, 0.0))
    )

    # Geometric context
    context = file.createIfcGeometricRepresentationContext(
        None, "Model", 3, 1.0e-05, origin
    )

    # Project
    project = file.create_entity('IfcProject', **{
        'GlobalId': ifcopenshell.guid.new(),
        'Name': f'Terminal 1 - {discipline} (Coordinated)',
        'RepresentationContexts': [context],
        'UnitsInContext': units
    })

    # Site
    site = file.create_entity('IfcSite', **{
        'GlobalId': ifcopenshell.guid.new(),
        'Name': 'Terminal 1 Site',
        'ObjectPlacement': file.createIfcLocalPlacement(None, origin)
    })

    # Building
    building = file.create_entity('IfcBuilding', **{
        'GlobalId': ifcopenshell.guid.new(),
        'Name': 'Terminal 1',
        'ObjectPlacement': file.createIfcLocalPlacement(None, origin)
    })

    # Storeys (GB, 1F, 2F, 3F, 4F, 5F)
    storeys = {}
    floor_heights = self.get_floor_heights_from_db()
    for floor_name, elevation in floor_heights.items():
        storey_placement = file.createIfcLocalPlacement(
            None,
            file.createIfcAxis2Placement3D(
                file.createIfcCartesianPoint((0.0, 0.0, elevation)),
                file.createIfcDirection((0.0, 0.0, 1.0)),
                file.createIfcDirection((1.0, 0.0, 0.0))
            )
        )
        storeys[floor_name] = file.create_entity('IfcBuildingStorey', **{
            'GlobalId': ifcopenshell.guid.new(),
            'Name': floor_name,
```

```
            'Elevation': elevation,
            'ObjectPlacement': storey_placement
        })

    # Spatial hierarchy
    file.createIfcRelAggregates(
        ifcopenshell.guid.new(), None, None, None, project, [site]
    )
    file.createIfcRelAggregates(
        ifcopenshell.guid.new(), None, None, None, site, [building]
    )
    file.createIfcRelAggregates(
        ifcopenshell.guid.new(), None, None, None, building, list(storeys.values())
    )

    return file, context, storeys
```

7.3 Geometry Export

```
def add_geometry(self, ifc_entity, element_data, ifc_file):
    """Convert database bbox to IFC geometry"""

    # Get bbox from database
    bbox = element_data['bbox']  # (min_x, min_y, min_z, max_x, max_y, max_z)

    # Simple box representation (sufficient for coordination)
    points = [
        ifc_file.createIfcCartesianPoint((bbox[0], bbox[1], bbox[2])),
        ifc_file.createIfcCartesianPoint((bbox[3], bbox[1], bbox[2])),
        ifc_file.createIfcCartesianPoint((bbox[3], bbox[4], bbox[2])),
        ifc_file.createIfcCartesianPoint((bbox[0], bbox[4], bbox[2])),
    ]

    # Extrude to height
    polyline = ifc_file.createIfcPolyline(points + [points[0]])  # Close loop
    profile = ifc_file.createIfcArbitraryClosedProfileDef(
        "AREA", None, polyline
    )

    extrusion_direction = ifc_file.createIfcDirection((0.0, 0.0, 1.0))
    extrusion_depth = bbox[5] - bbox[2]  # Z_max - Z_min

    extruded_solid = ifc_file.createIfcExtrudedAreaSolid(
        profile, None, extrusion_direction, extrusion_depth
    )

    # Create shape representation
    body_context = self.get_body_subcontext(ifc_file)
    shape_rep = ifc_file.createIfcShapeRepresentation(
        body_context, "Body", "SweptSolid", [extruded_solid]
    )

    product_shape = ifc_file.createIfcProductDefinitionShape(
        None, None, [shape_rep]
    )

    ifc_entity.Representation = product_shape
```

7.4 Property Export

```
def add_properties(self, ifc_entity, element_data, ifc_file):
    """Add custom properties from database"""

    # Query element properties
    properties = self.get_element_properties(element_data['guid'])

    # Create property set
    property_values = []
    for prop_name, prop_value in properties.items():
        property_values.append(
            ifc_file.createIfcPropertySingleValue(
                prop_name,
                prop_name,
                ifc_file.create_entity("IfcText", prop_value),
                None
            )
        )

    # Standard properties
    property_values.extend([
        ifc_file.createIfcPropertySingleValue(
            "Source", "Source",
            ifc_file.create_entity("IfcText", "DWG Auto-Conversion"),
            None
        ),
        ifc_file.createIfcPropertySingleValue(
            "CoordinationStatus", "CoordinationStatus",
            ifc_file.create_entity("IfcText", "Clash-Free Validated"),
            None
        ),
    ])

    pset = ifc_file.createIfcPropertySet(
```

```
        ifcopenshell.guid.new(),
        None,
        "Bonsai_AutoGenerated",
        None,
        property_values
    )

    ifc_file.createIfcRelDefinesByProperties(
        ifcopenshell.guid.new(),
        None, None, None,
        [ifc_entity],
        pset
    )
```

7.5 Export Workflow

```
# Example usage
exporter = DatabaseToIfcExporter()

# Export all disciplines
exporter.export_by_discipline(
    database_path="/home/red1/Documents/bonsai/DatabaseFiles/Terminal1_ClashFree.db",
    output_dir="/home/red1/Documents/bonsai/IFC_Export",
    disciplines=['ARC', 'STR', 'ELEC', 'ACMV', 'FP', 'SP', 'CW', 'LPG']
)

# Results:
# ✓ Terminal1-ARC-Coordinated.ifc (35,338 elements)
# ✓ Terminal1-STR-Coordinated.ifc (1,429 elements)
# ✓ Terminal1-ELEC-Coordinated.ifc (1,172 elements)
# ✓ Terminal1-ACMV-Coordinated.ifc (1,621 elements)
# ✓ Terminal1-FP-Coordinated.ifc (6,880 elements)
# ✓ Terminal1-SP-Coordinated.ifc (979 elements)
# ✓ Terminal1-CW-Coordinated.ifc (1,431 elements)
# ✓ Terminal1-LPG-Coordinated.ifc (209 elements)
```

7.6 Validation Before Export

```
def validate_clash_free(self, database_path):
    """Ensure database is clash-free before export"""

    conn = sqlite3.connect(database_path)

    # Run clash detection
    clashes = conn.execute("""
        SELECT COUNT(*) FROM clash_status WHERE status = 'ACTIVE'
    """).fetchone()[0]

    if clashes > 0:
        print(f"⚠️ WARNING: {clashes} unresolved clashes in database")
        response = input("Export anyway? (y/n): ")
        if response.lower() != 'y':
            print("Export cancelled. Resolve clashes first.")
            return False

    print("✓ Database is clash-free. Proceeding with export.")
    return True
```

---

8. IMPLEMENTATION ROADMAP

Phase 1: POC (1-2 weeks)

Goal: Parse one DWG floor plan → populate database → visualize in Blender

Tasks:

1. ✓ Set up ezdxf parser for Terminal 1 DWGs
2. ✓ Implement layer-to-discipline mapping
3. ✓ Extract polylines/circles/blocks from one floor (1F plan)

4. ✓ Generate 3D bounding boxes (simple extrusion)
5. ✓ Populate `elements_meta` and `base_geometries` tables
6. ✓ Load into Blender via existing federation tools
7. ✓ Compare against engineer-modeled IFC (visual inspection)

Success Criteria:

- 70%+ element count match (DWG vs IFC)
- Basic geometry positioned correctly
- No crashes during load

Phase 2: Classification Intelligence (3-4 weeks)

Goal: Semantic understanding of element types

Tasks:

1. ✓ Implement block pattern matching (seating, FP, electrical)
2. ✓ Build topology analyzer (seating arrays, MEP networks)
3. ✓ Text extraction for equipment tags/room labels
4. ✓ Cross-reference section drawings for accurate heights
5. ✓ Classify 90%+ of elements correctly

Success Criteria:

- Seating areas detected automatically
- FP equipment classified with 85%+ accuracy
- MEP networks traced correctly

Phase 3: Parametric Tools (2-3 weeks)

Goal: Designer-editable transformation utilities

Tasks:

1. ✓ Implement shift operations (UI + database updates)
2. ✓ Implement gap extension for seating arrays
3. ✓ Build preset configuration database
4. ✓ Create Blender operator panel for presets
5. ✓ Add clash detection after transformations

Success Criteria:

- Can shift 100+ seating elements in <1 second
- Gap adjustments preserve alignment
- Presets save/load correctly
- Live clash feedback during edits

Phase 4: Multi-Discipline Pipeline (4-6 weeks)

Goal: Process all 18 DWGs → full Terminal 1 database

Tasks:

1. ✓ Batch process all floor plans (GB, 1F-6F)
2. ✓ Parse detail drawings for equipment specs
3. ✓ Merge multi-sheet data (eliminate duplicates)
4. ✓ Process all 8 disciplines (ARC, STR, MEP×6)

5. ✓ Run full clash detection (34+ groups expected)

6. ✓ Generate BOQ from DWG-derived data

Success Criteria:

- Full database matches IFC-derived data in structure
- 85%+ accuracy vs manual modeling
- Clash detection finds same issues
- Processing time <10 minutes for all 18 sheets

---

8. TECHNICAL CHALLENGES & SOLUTIONS

Challenge 1: Ambiguous Geometry

Problem: Lines in DWG can be walls, reference lines, or just graphics

Solutions:

- Layer filtering (ignore "A-ANNO", "G-DIMS")
- Topology rules (walls form closed rooms, single lines likely reference)
- Thickness detection (double lines = wall, single line = reference)

Challenge 2: Block Variations

Problem: Different designers use different block names for same equipment

Solutions:

- Fuzzy matching (`difflib.SequenceMatcher`)
- Synonym dictionary ("SEAT" = "CHAIR" = "SEATING")
- Fallback to shape analysis (rectangular + in array = likely seating)

Challenge 3: Multi-Sheet Coordination

Problem: Equipment appears in multiple drawings with different detail levels

Solutions:

- GUID generation based on location + type (deterministic)
- Merge duplicates if centroid within 0.1m
- Prefer detail drawings over plan drawings for attributes

Challenge 4: Performance at Scale

Problem: 49K elements (current Terminal 1 count) requires fast processing

Solutions:

- Spatial indexing during parse (R-tree incremental insert)
- Batch database inserts (1000 rows at a time)
- Parallel processing (one thread per discipline)
- Target: <5 minutes for full conversion

---

9. DATA QUALITY METRICS

Accuracy Targets:

- Element Count: 90-110% of manual IFC (some DWG elements not modeled, some IFC elements added)
- Classification: 85%+ correct discipline/type
- Positioning: ±0.1m (100mm tolerance)

- Seating Detection: 95%+ (regular patterns easy to detect)
- MEP Equipment: 70%+ (complex equipment harder, acceptable for POC)

Validation Methods:

1. *Visual Comparison: Load DWG-derived + IFC side-by-side in Blender*
2. *Clash Detection: Run same filters, expect similar group counts*
3. *BOQ Comparison: Element counts by discipline should match ±15%*
4. *Spot Checks: Manually verify 50 random elements per discipline*

---

10. DESIGNER WORKFLOW (End Goal)

Scenario: Adjust Seating Layout

```
1. Designer opens Terminal 1 project in Blender (Bonsai)
2. Federation → Load DWG Database (auto-converted)
3. Select seating array in Gate 12 waiting area
4. Apply preset: "Increase Aisle Width" (+200mm)
5. System:
   - Shifts seating rows automatically
   - Updates spatial index
   - Runs clash detection
   - Shows: "3 new clashes with FP-012 (sprinkler)"
6. Designer reviews clashes in 3D
7. Options:
   a) Accept changes (save to database)
   b) Revert (undo transformation)
   c) Adjust preset parameters (try +150mm instead)
8. Once satisfied, export BCF for coordination meeting
```

Preset Library Examples:

```
Terminal Seating Presets:
- "Increase Aisle Width" → +200mm between rows
- "Compact Layout" → -100mm between rows (emergency capacity)
- "Shift Gate North" → Move entire gate area +3m
- "Extend Lounge" → Scale seating area 110% in X direction

MEP Coordination Presets:
- "Raise Ducts 200mm" → Shift all ACMV ducts +0.2m Z
- "Reroute Around Beam" → Avoid specified structural element
- "Extend FP Coverage" → Add sprinkler heads to gaps

Structural Safety Presets:
- "Enforce 300mm Clearance" → Shift MEP away from beams
- "Lock Load-Bearing" → Prevent shift of columns/beams
```

---

11. SUCCESS CRITERIA (Overall Project)

Phase 1 (POC):

✓ Parse 1 floor plan → Database → Blender visualization

✓ 70% element match vs manual IFC

✓ No manual modeling required

Phase 2 (Classification):

- ✓ Detect seating arrays automatically (95% accuracy)
- ✓ Classify FP/ELEC equipment (85% accuracy)
- ✓ Trace MEP networks (connectivity preserved)

Phase 3 (Parametric Tools):

- ✓ Shift operations working (instant feedback)
- ✓ Gap adjustment preserves alignment
- ✓ Presets save/load/apply correctly
- ✓ Clash detection after transformations

Phase 4 (Production):

- ✓ Process full Terminal 1 (18 DWGs → Database in <10 min)
- ✓ *Accuracy: 85% vs manual modeling*
- ✓ Designer can edit layouts without Revit/ArchiCAD
- ✓ Ready for Terminal 2/3 deployment

---

12. NEXT STEPS

Immediate (This Week):

1. ✓ Review this technical spec with stakeholders
2. ✓ Approve Phase 1 POC scope
3. ✓ Set up development environment
4. ✓ Extract one Terminal 1 floor plan DWG for testing

Phase 1 Kickoff (Next Week):

1. ✓ Implement ezdxf parser wrapper
2. ✓ Build layer mapping dictionary
3. ✓ Parse first floor plan (1F departures)
4. ✓ Populate database tables
5. ✓ Load in Blender and compare vs IFC

Document Version: 1.0

Status: Awaiting Approval

Est. Total Timeline: 10-16 weeks (POC to production)

Est. Effort: 3-4 months dev time (1 engineer)