GIS Script Generator
User Guide

PyQGIS | ArcPy | QGIS Project | ArcGIS Toolbox
Folium | Kepler.gl | pydeck | Web UI
Catalogue-driven script generation from PostGIS

Version 0.1.0 | 2026-02-28

This guide covers all eight template types generated by gis-codegen and gis-catalogue, the Web UI, the full operations reference, the symbology dispatch table, and complete CLI documentation.

1. Project Overview

gis-codegen is a command-line tool that connects to a PostGIS database, extracts spatial layer metadata (geometry type, SRID, columns, primary keys, row counts), and generates ready-to-run scripts or project files for eight GIS platforms.

gis-catalogue reads a map catalogue Excel file and writes one script per map entry, auto-selecting the correct renderer based on the symbology_type column.

gis-ui launches a browser-based form that connects, extracts, and returns the generated file as a download -- no command line required.

Template types

CLI command	Template type	Output file
gis-codegen --platform pyqgis	PyQGIS standalone script	*.py
gis-codegen --platform arcpy	ArcPy / ArcGIS Pro script	*.py
gis-codegen --platform folium	Folium / Leaflet HTML map	*.py
gis-codegen --platform kepler	Kepler.gl HTML map	*.py
gis-codegen --platform deck	pydeck / deck.gl HTML map	*.py
gis-codegen --platform export	PostGIS -> GeoPackage script	*.py
gis-codegen --platform qgs	QGIS project file	*.qgs
gis-codegen --platform pyt	ArcGIS Python Toolbox	*.pyt
gis-catalogue --platform pyqgis	Catalogue PyQGIS (x N maps)	M##_name.py
gis-catalogue --platform arcpy	Catalogue ArcPy (x N maps)	M##_name.py

Package layout

```
src/gis_codegen/
    __init__.py         Public API: connect, extract_schema, generate_*
    extractor.py        PostGIS metadata queries
    generator.py        Code generation (all 8 platforms)
    catalogue.py        Excel-driven per-map code generation
    cli.py              gis-codegen CLI entry point
    app.py              Flask web UI (gis-ui entry point)
tests/
    conftest.py         Shared fixtures
    test_generator.py   test_catalogue.py   test_extractor.py
    test_app.py         test_integration.py
.github/workflows/
    ci.yml              Unit + integration CI jobs
maps/                   Generated output (git-ignored)
make_pdf.py             This PDF generator
```

2. Installation & Setup

Install

```
# From the project root:
pip install -e .

# With web mapping extras (folium, kepler, pydeck):
pip install -e ".[web]"

# With Flask web UI:
pip install -e ".[server]"

# With dev tools (pytest, coverage, openpyxl):
pip install -e ".[dev]"

# With PostGIS integration tests (requires Docker):
pip install -e ".[integration]"

# Entry points after install:
gis-codegen   --help
gis-catalogue --help
gis-ui                  # launches web UI at http://localhost:5000
```

Required environment variable

PGPASSWORD must always be set before running any command. It is never stored in config files or embedded in generated scripts.

```
# Windows
set PGPASSWORD=your_password

# Linux / macOS
export PGPASSWORD=your_password
```

Optional environment variables

Variable	Default	Description
PGHOST	localhost	Database host
PGPORT	5432	Database port
PGDATABASE	my_gis_db	Database name
PGUSER	postgres	Database user
PGPASSWORD	(required)	Database password -- no default

Config file (gis_codegen.toml)

Place in the project root, or pass with --config. CLI flags override it.

```
[database]
host   = "localhost"
port   = 5432
dbname = "my_gis_db"
user   = "postgres"
# password NOT stored here -- use PGPASSWORD env var
```

```
[defaults]
platform      = "pyqgis"
schema_filter = "public"
no_row_counts = false
output        = "output.py"
save_schema   = "schema.json"
```

3. Configuration & Connection

Connection value priority

When a command runs, connection values are resolved in this order (highest wins):

```
1. CLI flags       --host, --port, --dbname, --user, --password
2. Config file     gis_codegen.toml  [database] section
3. Environment     PGHOST, PGPORT, PGDATABASE, PGUSER, PGPASSWORD
4. Built-in        localhost / 5432 / my_gis_db / postgres
```

Example: gis-codegen --host prod.example.com --platform pyqgis will use prod.example.com for the host, but PGPORT, PGDATABASE, and PGUSER from env vars or config file.

Config file search order

gis-codegen looks for a TOML config file in this order (first found wins):

```
1. --config FILE                          (explicit CLI path)
2. $GIS_CODEGEN_CONFIG env var            (if set)
3. ./gis_codegen.toml                     (project root)
4. ~/.config/gis_codegen/config.toml      (user home)
```

Tip: Use project-level gis_codegen.toml for workspace defaults, and user-level config.toml for persistent personal settings.

Complete config file reference

```
# [database] section -- connection credentials
[database]
host     = "localhost"      # PGHOST override
port     = 5432             # PGPORT override
dbname   = "my_gis_db"      # PGDATABASE override
user     = "postgres"       # PGUSER override
# password = "pass"          # NOT recommended; use PGPASSWORD env var

# [defaults] section -- pre-set generation options
[defaults]
platform      = "pyqgis"    # default platform (pyqgis or arcpy)
schema_filter = "public"    # only extract from this PG schema
no_row_counts = false       # set to true to speed up on large DBs
output        = "output.py" # output file (overridable with -o)
save_schema   = "schema.json"  # auto-save schema for offline use
```

Password security best practices

PGPASSWORD should never be stored in committed config files or scripts.

```
# Good: set as environment variable at shell level
export PGPASSWORD=secret
gis-codegen --list-layers

# Good: use shell script or CI secret management
# Avoid: hardcoding in gis_codegen.toml
# Avoid: embedding in generated Python scripts
```

Windows users: use set instead of export. The password expires when the shell closes, so this is safe for interactive use.

4. Schema Extraction

The extractor queries geometry_columns, information_schema.columns, information_schema.table_constraints, and pg_class to build a full metadata snapshot of all spatial layers in the database.

Preview layers (no files written)

```
gis-codegen --list-layers

# Example output:
#  schema  table    geom_type        srid   rows
#  ------  -----    ---------        ----   ----
#  public  parcels  MULTIPOLYGON     4326   ~1 000
#  public  roads    MULTILINESTRING  4326   ~500
```

Save schema JSON for offline use

```
gis-codegen --save-schema schema.json
# Then generate without a live DB connection:
gis-codegen --platform pyqgis -i schema.json -o my_map.py
```

Schema JSON structure

```
{
  "database": "my_gis_db",
  "host": "localhost",
  "layer_count": 1,
  "layers": [
    {
      "schema": "public",
      "table": "parcels",
      "qualified_name": "public.parcels",
      "geometry": {"column":"geom","type":"MULTIPOLYGON","srid":4326},
      "columns": [
        {"name":"parcel_id","data_type":"integer","nullable":false},
        {"name":"address",  "data_type":"character varying","nullable":true}
      ],
      "primary_keys": ["parcel_id"],
      "row_count_estimate": 1000
    }
  ]
}
```

5. PyQGIS Template

PyQGIS scripts connect to PostGIS via QgsDataSourceUri and load layers into a QGIS project. They can run as standalone scripts (outside QGIS) or be pasted into the QGIS Python console.

Generate

```
# From a live database:
gis-codegen --platform pyqgis -o my_map.py

# From a saved schema:
gis-codegen --platform pyqgis -i schema.json -o my_map.py

# Filter to one layer:
gis-codegen --platform pyqgis --layer public.parcels -o parcels.py

# Add operation blocks:
gis-codegen --platform pyqgis --op buffer --op dissolve -o parcels.py
```

Template anatomy

```
"""
Auto-generated PyQGIS script
Database : my_gis_db @ localhost:5432
Generated: 2026-02-23 14:00   Layers: 2
"""

import os, sys
from qgis.core import (
    QgsApplication, QgsDataSourceUri, QgsVectorLayer, QgsProject,
    QgsCoordinateReferenceSystem,
)

# Remove the next 2 lines if running inside the QGIS console:
qgs = QgsApplication([], False)
qgs.initQgis()

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = "..."         # value at generation time

# ================================================================
# Layer : public.parcels   Geom: MULTIPOLYGON   SRID: 4326
# ================================================================
uri_parcels = QgsDataSourceUri()
uri_parcels.setConnection(DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD)
uri_parcels.setDataSource("public", "parcels", "geom", "", "parcel_id")

lyr_parcels = QgsVectorLayer(uri_parcels.uri(False), "parcels", "postgres")
if not lyr_parcels.isValid():
    print("[ERROR] Layer 'parcels' failed to load.")
else:
    QgsProject.instance().addMapLayer(lyr_parcels)
    print(f"[OK] parcels: {lyr_parcels.featureCount()} features")

qgs.exitQgis()   # remove if running inside QGIS console
```

How to run

- Standalone: set PGPASSWORD=... then python my_map.py
- (requires the QGIS Python environment on PATH)
- QGIS console: paste the script body; remove QgsApplication init/exit lines
- QGIS Processing Toolbox: Plugins -> Python Console -> Open Script

6. ArcPy Template

ArcPy scripts create a temporary .sde connection file, access PostGIS feature classes through that connection, and optionally run analysis tools. They require ArcGIS Pro with the PostgreSQL client libraries installed.

Generate

```
gis-codegen --platform arcpy -o my_map.py

# With 3D massing ops:
gis-codegen --platform arcpy --op extrude --op scene_layer -o massing.py
```

Template anatomy

```
import arcpy, os, tempfile

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = "..."

SDE_FOLDER = tempfile.gettempdir()
SDE_FILE   = os.path.join(SDE_FOLDER, f"{DB_NAME}.sde")
if not os.path.exists(SDE_FILE):
    arcpy.management.CreateDatabaseConnection(
        database_platform="POSTGRESQL",
        instance=f"{DB_HOST},{DB_PORT}",
        account_authentication="DATABASE_AUTH",
        username=DB_USER, password=DB_PASSWORD,
        save_user_pass="SAVE_USERNAME", database=DB_NAME, ...
    )

fc_parcels = os.path.join(SDE_FILE, "public.parcels")
if arcpy.Exists(fc_parcels):
    count = int(arcpy.management.GetCount(fc_parcels)[0])
    print(f"[OK] parcels: {count} rows")
```

How to run

- ArcGIS Pro Python console: paste and run directly
- Standalone: run from the ArcGIS Pro Python env (arcgispro-py3)
- Script Tool: add as a Python Script Tool in a Toolbox (.atbx)

NOTE: The SDE file persists between runs. Delete it manually if you change the host, database name, or credentials.

7. QGIS Project File (--platform qgs)

Instead of a Python script, this platform emits a .qgs XML file that opens directly in QGIS 3.x. All PostGIS layers appear pre-connected in the Layers panel -- no scripting required. QGIS prompts for the database password on open; it is never embedded in the file.

Generate

```
gis-codegen --platform qgs -o project.qgs

# Then open in QGIS:
#   File -> Open Project -> select project.qgs
#   QGIS prompts for the PostGIS password
#   All layers appear in the Layers panel
```

Key design decisions

- Layer IDs are deterministic: {table}_{md5(qualified_name)[:8]} -- safe to version-control, same ID on every regeneration
- Geometry type mapping: POINT/MULTIPOINT -> Point (code 0), LINESTRING/MULTILINESTRING -> Line (code 1), POLYGON/MULTIPOLYGON -> Polygon (code 2)
- Project CRS and map canvas default to EPSG:4326 (world bounds)
- Per-layer <srs> uses minimal <authid>EPSG:{srid}</authid> -- QGIS resolves the full WKT from its internal EPSG registry
- sslmode=disable in the datasource string (change to require for production)

File structure

```
<!DOCTYPE qgis PUBLIC 'http://mrcc.com/qgis.dtd' 'SYSTEM'>
<qgis projectname="my_gis_db" version="3.28.0-Firenze">
  <projectCrs>
    <spatialrefsys><authid>EPSG:4326</authid></spatialrefsys>
  </projectCrs>
  <mapcanvas name="theMapCanvas">
    <units>degrees</units>
    <extent>-180 -90 180 90</extent>
    <destinationsrs><spatialrefsys><authid>EPSG:4326</authid></spatialrefsys></destinationsrs>
  </mapcanvas>
  <projectlayers>
    <maplayer type="vector" geometry="Polygon">
      <id>parcels_a1b2c3d4</id>
      <datasource>dbname='my_gis_db' host=localhost port=5432
        sslmode=disable key='parcel_id' srid=4326 type=Polygon
        table="public"."parcels" (geom) sql=</datasource>
      <layername>parcels</layername>
      <provider encoding="UTF-8">postgres</provider>
      <srs><spatialrefsys><authid>EPSG:4326</authid></spatialrefsys></srs>
      <layerGeometryType>2</layerGeometryType>
    </maplayer>
  </projectlayers>
  <legend>...</legend>
</qgis>
```

NOTE: --op flags are silently ignored for the qgs platform. Operations are only supported on pyqgis and arcpy.

8. ArcGIS Python Toolbox (--platform pyt)

This platform generates a .pyt Python Toolbox file. When opened in ArcGIS Pro via Insert -> Toolbox -> Add Python Toolbox, it presents a GUI dialog with connection parameters pre-filled. Running the tool loads all PostGIS layers into the active map. The password is never hardcoded -- the dialog prompts for it at runtime.

Generate

```
gis-codegen --platform pyt -o loader.pyt

# Then in ArcGIS Pro:
#   Insert -> Toolbox -> Add Python Toolbox
#   Navigate to loader.pyt
#   Double-click 'Load PostGIS Layers'
#   Fill in password, click Run
```

Toolbox structure

```
class Toolbox:
    def __init__(self):
        self.label = "PostGIS Loader"
        self.tools = [LoadPostGISLayers]

class LoadPostGISLayers:
    def getParameterInfo(self):   # 6 parameters:
        # host     (GPString, Required, default from db_config)
        # port     (GPString, Required, default from db_config)
        # dbname   (GPString, Required, default from db_config)
        # user     (GPString, Required, default from db_config)
        # password (GPStringHidden, Required, NO default)
        # schema_filter (GPString, Optional)

    def isLicensed(self):  return True
    def updateParameters(self, parameters):  pass
    def updateMessages(self, parameters):    pass

    def execute(self, parameters, messages):
        # CreateDatabaseConnection -> postgis_conn.sde in scratchFolder
        # ArcGISProject('CURRENT').activeMap.addDataFromPath per layer
        _tables = [
            ('public', 'parcels'),
            ('public', 'roads'),
            ...
        ]
```

NOTE: --op flags are silently ignored for the pyt platform. Operations are only supported on pyqgis and arcpy.

9. Web UI (gis-ui)

The web UI is a minimal Flask application that exposes the full gis-codegen pipeline as a browser form. Fill in the connection details, choose a platform, and click Generate -- the script or project file is returned as a file download. No command line or Python knowledge needed.

Start the server

```
pip install -e ".[server]"
gis-ui
# -> Serving on http://0.0.0.0:5000
# Open http://localhost:5000 in a browser
```

Routes

Method	Path	Description
GET	/	Render the connection + platform form
POST	/generate	Connect, extract, generate, return file download

File download extensions

Platform	Downloaded as
qgs	*.qgs (QGIS project file)
pyt	*.pyt (ArcGIS Python Toolbox)
pyqgis, arcpy, folium	*.py (Python script)

Security notes

- Password is submitted via POST body only -- never in the URL or query string
- Password is never stored, logged, or reflected in the response
- Jinja2 autoescaping is active (Flask default) -- XSS safe
- Connection errors re-render the form with an error message (HTTP 400); the error text is from the psycopg2 exception and does not include the password
- No authentication layer is included -- run behind a reverse proxy or firewall if exposing beyond localhost

NOTE: The web UI supports all eight platforms. --op flags are not available in the web form; use the CLI directly for operation blocks.

10. Web Mapping Templates (folium / kepler / deck)

Web templates read PostGIS via geopandas + SQLAlchemy and produce a standalone HTML file. Install extras first: pip install -e ".[web]"

NOTE: All web templates use DB_PASSWORD = os.environ["PGPASSWORD"] -- the password is NEVER embedded in generated scripts.

Folium / Leaflet (--platform folium)

```
gis-codegen --platform folium -o map.py
python map.py        # -> map.html
```

Produces a Leaflet map with GeoJson layers and tooltips from the first 5 columns per layer. Polygons, lines, and points each get a distinct colour from a 6-colour cycling palette. A LayerControl widget is added automatically. All geometries are reprojected to EPSG:4326.

Kepler.gl (--platform kepler)

```
gis-codegen --platform kepler -o kepler_map.py
python kepler_map.py     # -> kepler_map.html
                         # (or render inline in Jupyter)
```

Loads all layers into a KeplerGl map object. If a height column is detected (height, floors, elevation, z, roof_height, ...) a 3D tip comment is added. Enable it in the Kepler UI: Layers -> type -> 3D buildings, height field = detected column.

pydeck / deck.gl (--platform deck)

```
gis-codegen --platform deck -o deck_map.py
python deck_map.py   # -> deck_map.html
```

Polygon/line layers use GeoJsonLayer; point layers use ScatterplotLayer. When a height column is detected, commented extrusion lines are added -- uncomment extruded=True and set pitch=45 for a 3D view.

GeoPackage export (--platform export)

```
gis-codegen --platform export -o export.py
python export.py     # -> my_gis_db_export.gpkg
```

Dumps every spatial layer from PostGIS to a single GeoPackage file using geopandas. Each layer is written in a try/except block so one failure does not abort the rest. The script exits with code 1 if any layer failed.

Colour palette (shared by Folium and pydeck)

Layer index	Hex colour	RGBA
0 (first)	#ff8c00	[255, 140, 0, 160] orange
1	#0080ff	[0, 128, 255, 160] blue
2	#00c864	[0, 200, 100, 160] green
3	#ff3232	[255, 50, 50, 160] red
4	#b400ff	[180, 0, 255, 160] purple

5+	#00c8c8	[0, 200, 200, 160] teal (cycles)

11. Operations Reference (--op flag)

Operations inject additional code blocks into PyQGIS or ArcPy scripts. Repeat --op for multiple operations in one script:

```
gis-codegen --platform pyqgis --op buffer --op dissolve -o out.py
gis-codegen --platform arcpy  --op extrude --op scene_layer -o 3d.py
```

NOTE: Operations are not supported on web platforms (folium, kepler, deck, export) or on qgs and pyt. The --op flag is silently ignored for those platforms.

General operations (10)

--op value	PyQGIS	ArcPy
reproject	processing: native:reprojectlayer	management.Project
export	QgsVectorFileWriter -> GeoJSON	conversion.FeatureClassToShapefile
buffer	processing: native:buffer	analysis.Buffer
clip	processing: native:clip (commented)	analysis.Clip (commented)
select	selectByExpression	SelectLayerByAttribute
dissolve	processing: native:dissolve	management.Dissolve
centroid	processing: native:centroids	management.FeatureToPoint
field_calc	processing: native:fieldcalculator	management.CalculateField
spatial_join	joinattributesbylocation (commented)	analysis.SpatialJoin (commented)
intersect	native:intersection (commented)	analysis.Intersect (commented)

Operations marked (commented) produce a ready-to-uncomment template that requires you to define an input boundary or overlay layer first.

3D massing operations (5)

--op value	Description	Notes
extrude	Data-driven height extrusion renderer	PyQGIS: QgsPolygon3DSymbol ArcPy: ddd.ExtrudePolyg
z_stats	Min / max / mean Z vertex statistics	PyQGIS: hasZ + vertex loop ArcPy: ddd.AddZInformation
floor_ceiling	Extrude from base_height to roof_height field	Two-field floor-to-roof extrusion
volume	Approx. volume = footprint area x height	Fast estimate; exact: ST_Volume() in PostGIS
scene_layer	Export 3D layer package	PyQGIS: native:convert3dtiles ArcPy: CreateSceneLayerP

12. Catalogue-Driven Generation

gis-catalogue reads a map catalogue Excel file and writes one script per map entry. Only rows where status is 'have' or 'partial' AND spatial_layer_type contains 'Vector' are included.

Inclusion rules

status	spatial_layer_type	Included?
have	Vector	YES
partial	Vector	YES
have	Raster/Vector	YES (+ raster TODO note)
todo	Vector	NO -- skipped
have	Raster	NO -- skipped
todo	Raster	NO -- skipped

Key catalogue columns

Column	Role in generated script
map_id	Section header comment and layout name (e.g. M07_layout)
short_name	Python variable names and PostGIS table name (public.short_name)
spatial_layer_type	Triggers raster TODO note when 'Raster' is in the value
symbology_type	Drives automatic renderer selection (see Chapter 12)
validation_checks	Written as # [ ] item checklist
deliverable_format	Written into the export stub comment
classification	Passed to categorized renderer block as scheme comment

CLI usage

```
# Generate PyQGIS scripts (default):
gis-catalogue --input catalogue.xlsx --output-dir ./maps/

# Generate ArcPy scripts:
gis-catalogue --input catalogue.xlsx --platform arcpy --output-dir ./maps_arcpy/

# Preview what would be generated (no files written):
gis-catalogue --input catalogue.xlsx --list

# Use a saved schema JSON instead of a live DB connection:
gis-catalogue --input catalogue.xlsx --schema schema.json

# Add operation blocks to every script:
gis-catalogue --input catalogue.xlsx --op buffer --op reproject

# Override DB connection:
gis-catalogue --input catalogue.xlsx --host myserver --dbname prod_db
```

Output naming

```
# One file per included map:
```

```
maps/
    M03_occupation_sol_2026.py
    M07_hauteurs_etages_degrade.py
    M27_commerces_typologie_concentrations.py
    ...
```

13. Symbology Dispatch Table

The symbology_type cell in the catalogue drives automatic renderer selection. Matching is case-insensitive on the full cell value. The first matching rule wins.

Keyword(s) in symbology_type	PyQGIS renderer	ArcPy renderer
heatmap OR densité	QgsHeatmapRenderer	HeatMapRenderer (Pro 3.x)
réseau OR network	QgsCategorizedSymbolRenderer (QgsLineSymbol)	UniqueValueRenderer (NET_FIELD placeholde
catégoriel / catégorie (without choroplèthe)	QgsCategorizedSymbolRenderer	UniqueValueRenderer
choroplèthe + catégoriel	QgsCategorizedSymbolRenderer	UniqueValueRenderer
choroplèthe / dégradé / gradué	QgsGraduatedSymbolRenderer	GraduatedColorsRenderer
points OR polygones	QgsSingleSymbolRenderer	SimpleRenderer
(anything else)	# TODO: configure renderer	# TODO: configure renderer

NOTE: All renderer blocks use a TODO comment to mark the field name (GRAD_FIELD, CAT_FIELD, NET_FIELD) that you must verify against the actual PostGIS table schema.

14. Catalogue PyQGIS Template Anatomy

Each file generated by gis-catalogue --platform pyqgis has six sections:

1 Docstring header (16 fields)

```
"""
Map ID    : M07
Title     : Hauteurs / nombre d'etages (degrade)
Theme     : Forme urbaine > Gabarits
Objective : Representer gabarits et gradient de hauteur
Symbology : choroplethe (degrade)
Sources   : OSM building:levels  [2024-2026]
Status    : have  |  Owner: Liam
Generated : 2026-02-23 14:48
"""
```

2 QGIS imports & init

```
import os
from qgis.core import (
    QgsApplication, QgsDataSourceUri, QgsVectorLayer, QgsProject,
)
qgs = QgsApplication([], False)
qgs.initQgis()
```

3 DB credentials

```
DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = os.environ["PGPASSWORD"]   # never hardcoded
```

4 Layer load block

```
uri_hauteurs_etages_degrade = QgsDataSourceUri()
uri_hauteurs_etages_degrade.setConnection(
    DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD)
uri_hauteurs_etages_degrade.setDataSource(
    "public", "hauteurs_etages_degrade", "geom", "", "")

lyr_hauteurs_etages_degrade = QgsVectorLayer(
    uri_hauteurs_etages_degrade.uri(False), "hauteurs_etages_degrade", "postgres")

if not lyr_hauteurs_etages_degrade.isValid():
    print("[ERROR] 'hauteurs_etages_degrade' failed to load.")
else:
    QgsProject.instance().addMapLayer(lyr_hauteurs_etages_degrade)
```

5 Auto-dispatched symbology block

```
    # --- Symbology: choroplethe (degrade) ---
    from qgis.core import (
        QgsGraduatedSymbolRenderer, QgsClassificationQuantile,
        QgsColorBrewerColorRamp,
    )
```

```
    GRAD_FIELD_... = "value"   # TODO: verify field name
    _rend = QgsGraduatedSymbolRenderer(GRAD_FIELD_...)
    _rend.setClassificationMethod(QgsClassificationQuantile())
    _rend.updateClasses(lyr_..., 5)
    _rend.updateColorRamp(QgsColorBrewerColorRamp("YlOrRd", 5))
    lyr_....setRenderer(_rend)
    lyr_....triggerRepaint()
```

6 Validation checklist & export stub

```
    # --- Validation checks ---
    # [ ] valeurs nulles
    # [ ] plausibilite niveaux
    # [ ] palette lisible

    # --- Export: Layout PDF + couche ---
    # TODO: configure a QGIS print layout named "M07_layout" then:
    # from qgis.core import QgsLayoutExporter
    # ...

qgs.exitQgis()
```

15. Catalogue ArcPy Template Anatomy

Each file generated by gis-catalogue --platform arcpy has the same six sections as the PyQGIS version but uses ArcPy idioms throughout.

1-3 Docstring, imports & credentials

```
# Docstring header identical to PyQGIS version.

import arcpy
import os
import tempfile

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = os.environ["PGPASSWORD"]
```

4 SDE connection file

```
SDE_FOLDER = tempfile.gettempdir()
SDE_FILE   = os.path.join(SDE_FOLDER, f"{DB_NAME}.sde")

if not os.path.exists(SDE_FILE):
    arcpy.management.CreateDatabaseConnection(
        database_platform="POSTGRESQL",
        instance=f"{DB_HOST},{DB_PORT}",
        account_authentication="DATABASE_AUTH",
        username=DB_USER, password=DB_PASSWORD,
        save_user_pass="SAVE_USERNAME", database=DB_NAME, ...
    )
```

5 Feature class check and project setup

```
fc_... = os.path.join(SDE_FILE, "public.hauteurs_etages_degrade")

if arcpy.Exists(fc_...):
    count = int(arcpy.management.GetCount(fc_...)[0])
    print(f"[OK] {count} rows")

    aprx   = arcpy.mp.ArcGISProject("CURRENT")
    mp_map = aprx.listMaps()[0]
    lyr_... = mp_map.addDataFromPath(fc_...)
```

6 Auto-dispatched ArcPy symbology block

```
    # --- Symbology: choroplethe (degrade) ---
    GRAD_FIELD_... = "value"   # TODO: verify field name
    sym = lyr_....symbology
    sym.updateRenderer("GraduatedColorsRenderer")
    sym.renderer.classificationField = GRAD_FIELD_...
    sym.renderer.breakCount = 5
    lyr_....symbology = sym
    aprx.save()
```

NOTE: Use APRX_PATH = "CURRENT" when running inside the ArcGIS Pro console. For standalone scripts, set

APRX_PATH to the path of your .aprx file.

16. CLI Quick Reference

gis-codegen -- all flags

```
gis-codegen [connection] [generation] [schema]

Connection flags (override config file and env vars):
  --host HOST         Database host
  --port PORT         Database port
  --dbname DBNAME     Database name
  --user USER         Database user
  --config FILE       TOML config file path

Generation flags:
  --platform          pyqgis | arcpy | folium | kepler | deck |
                      export | qgs | pyt
  --op OPERATION      Add an operation block (repeatable)
                      See Chapter 10 for all 15 valid values
                      (ignored for folium/kepler/deck/export/qgs/pyt)
  --layer SCHEMA.TABLE  Restrict to this layer (repeatable)
  -o / --output FILE  Write to file (default: stdout)

Schema flags:
  --list-layers       Print layer table and exit
  --save-schema FILE  Save schema JSON and exit
  --no-row-counts     Skip row count queries (faster)
  --schema-filter S   Only include layers in schema S

Priority: CLI flags > config file > env vars > built-in defaults
```

gis-catalogue -- all flags

```
gis-catalogue [options]

Required:
  -i / --input FILE        Path to catalogue .xlsx file

Optional:
  -o / --output-dir DIR    Output directory  (default: ./maps/)
  -p / --platform          pyqgis | arcpy    (default: pyqgis)
  --host HOST              (default: localhost / PGHOST)
  --port PORT              (default: 5432 / PGPORT)
  --dbname DBNAME          (default: my_gis_db / PGDATABASE)
  --user USER              (default: postgres / PGUSER)
  --schema FILE            Schema JSON from gis-codegen --save-schema
                           (offline mode -- no DB connection needed)
  --op OPERATION           Add operation block to every script (repeatable)
  --list                   Print filtered maps and exit (no files written)

Filter applied automatically:
  status IN {"have","partial"}  AND  "Vector" IN spatial_layer_type
```

gis-ui -- web UI

```
gis-ui
# Starts Flask server at http://0.0.0.0:5000
# Open http://localhost:5000 in a browser
```

```
# Requires:  pip install -e ".[server]"
```

End-to-end workflow examples

```
# 1. Preview what is in the database:
set PGPASSWORD=secret
gis-codegen --list-layers

# 2. Save the schema for offline generation:
gis-codegen --save-schema schema.json

# 3. Generate a PyQGIS script with buffer + dissolve:
gis-codegen --platform pyqgis --op buffer --op dissolve -o analysis.py

# 4. Open all layers directly in QGIS:
gis-codegen --platform qgs -o project.qgs

# 5. Load all layers into ArcGIS Pro via a toolbox dialog:
gis-codegen --platform pyt -o loader.pyt

# 6. Generate all catalogue maps as PyQGIS scripts:
gis-catalogue --input catalogue.xlsx --output-dir ./maps/

# 7. Generate a Kepler.gl web map:
gis-codegen --platform kepler -o kepler_map.py
python kepler_map.py

# 8. Use the browser-based web UI:
gis-ui
```

17. Testing

Unit tests (no database or Docker required)

```
# Install dev + server extras:
pip install -e ".[dev,server]"

# Run all unit tests (excludes integration):
python -m pytest tests/ -m "not integration" -v

# With coverage report (must reach 80%):
python -m pytest tests/ -m "not integration" \
    --cov=gis_codegen --cov-report=term-missing

# Run one file:
python -m pytest tests/test_generator.py -v

# Filter by keyword:
python -m pytest tests/ -k "qgs" -v
```

Integration tests (requires Docker)

```
# Install integration extras:
pip install -e ".[dev,integration]"

# Run integration tests (spins up postgis/postgis:15-3.3 container):
python -m pytest tests/test_integration.py -v -m integration

# The tests skip gracefully if testcontainers is not installed.
```

Test suite summary

File	Tests	What is covered
test_generator.py	173	safe_var, pg_type_to_*, _qgs_geom_type, 15 op blocks x 2 platforms, 8 generators (
test_catalogue.py	108	load_catalogue filtering, 5 PyQGIS + 5 ArcPy renderer blocks, symbology dispatch x
test_extractor.py	34	fetch_columns, fetch_primary_keys, fetch_row_count_estimate, extract_schema (mo
test_app.py	11	GET / form rendering, POST /generate happy paths (pyqgis, qgs, pyt), error paths (ba
test_integration.py	19	Live PostGIS container via testcontainers: extract_schema, generate_pyqgis, genera
TOTAL	345	

Unit tests (all except test_integration.py) run in under 2 seconds because the generators are pure string-building functions -- no database connection or GIS library is needed.

CI pipeline (.github/workflows/ci.yml)

- unit job: runs on ubuntu-latest, installs .[dev,server], runs pytest -m 'not integration' with coverage
- integration job: runs on ubuntu-latest (Docker available), installs .[dev,integration], runs pytest test_integration.py -m integration
- Both jobs trigger on push to main/master and on pull requests

18. Practical Workflows

Workflow 1: Load all layers in QGIS with 3D visualization

Goal: Open all spatial layers in QGIS and extrude buildings by height.

```
# 1. Connect to PostGIS and generate a PyQGIS script with 3D extrusion:
set PGPASSWORD=secret
gis-codegen --platform pyqgis --op extrude --op z_stats \
             -o load_3d.py

# 2. Open QGIS
# 3. Open Python console (Plugins > Python Console)
# 4. Run the script:
exec(open('load_3d.py').read())

# 5. In the Layers panel, select a layer and use the 3D view button
```

The generated script includes height calculations and is ready for 3D rendering. For full 3D visualization, use the --op scene_layer operation to export a 3D layer package.

Workflow 2: Create web maps from PostGIS data

Goal: Generate interactive Leaflet and Kepler.gl maps from the database.

```
# 1. Generate a Folium map (Leaflet):
gis-codegen --platform folium -o my_map.py
python my_map.py  # -> my_map.html

# 2. Generate a Kepler.gl map (large datasets):
gis-codegen --platform kepler -o kepler_map.py
python kepler_map.py  # -> kepler_map.html

# 3. Generate a pydeck map (deck.gl with 3D support):
gis-codegen --platform deck -o deck_map.py
python deck_map.py  # -> deck_map.html

# Open the .html files in any browser
```

Each generator automatically selects an appropriate layer color scheme based on geometry type and data distribution.

Workflow 3: Filter layers and apply operations

Goal: Generate scripts for only polygon layers, with buffer and dissolve.

```
# List layers first to see geometry types:
gis-codegen --list-layers

# Generate PyQGIS for specific layers with operations:
gis-codegen --platform pyqgis \
             --layer public.buildings \
             --layer public.parcels \
             --op buffer --op dissolve \
             -o analysis.py
```

Tip: --layer accepts schema.table format. Repeat --layer to select multiple layers. Operations are applied in order.

Workflow 4: Batch generate scripts from an Excel catalogue

Goal: Create one script per map in an Excel catalogue, auto-configured.

```
# 1. Prepare catalogue.xlsx with columns:
#    map_id, theme, symbology_type, status, spatial_layer_type, etc.
# 2. Run gis-catalogue:
gis-catalogue --input catalogue.xlsx \
              --platform pyqgis \
              --output-dir ./generated_scripts/

# 3. Check what was filtered (dry-run):
gis-catalogue --input catalogue.xlsx --list
```

Included maps: status IN {have, partial} AND spatial_layer_type contains Vector. The symbology_type column determines which renderer is used (choropleth, categorized, etc.).

Workflow 5: Offline generation with saved schema

Goal: Generate scripts when database is offline or unavailable.

```
# 1. While database is online, save the schema:
gis-codegen --save-schema schema.json

# 2. Later, generate scripts offline using the saved schema:
gis-codegen --schema schema.json --platform arcpy -o output.py

# 3. No database connection is needed for offline generation:
gis-catalogue --input catalogue.xlsx \
              --schema schema.json \
              --platform arcpy
```

Schema files are regular JSON and can be committed to version control. Use this for reproducible, offline-friendly CI/CD pipelines.

19. Troubleshooting

Error: No database password supplied

This error occurs when PGPASSWORD is not set and no password was provided via CLI flags or config file.

```
# Solution 1: Set PGPASSWORD environment variable
set PGPASSWORD=your_password  # Windows
gis-codegen --list-layers

# Solution 2: Pass password on command line
gis-codegen --password your_password --list-layers

# Solution 3: Add to gis_codegen.toml [database] section
# (Not recommended for security)
```

Error: No spatial layers found

The database connected successfully, but no geometry columns were found.

```
# Check which tables have geometry:
SELECT table_name, column_name, geometry_type, srid
FROM geometry_columns;
```

Common causes: (1) Geometry columns were added manually without AddGeometryColumn function; (2) table is in a non-public schema and --schema-filter excludes it; (3) the layer is actually a view and the geometry wasn't registered.

Error: Operation ignored (warning)

This is a warning, not an error. Operations (--op) are only supported on pyqgis and arcpy platforms.

```
# Good: operation is applied
gis-codegen --platform pyqgis --op buffer -o out.py

# Warning: operation is silently ignored (generates file anyway)
gis-codegen --platform qgs --op buffer -o out.qgs
[warn] --op buffer ignored (qgs does not support operations)
```

This is intentional design. The script still generates successfully.

Error: Connection refused (ECONNREFUSED)

The CLI tried to connect to the database but the server is unreachable.

```
# Check connection parameters:
gis-codegen --host localhost --port 5432 --dbname mydb \
            --user postgres --list-layers

# Verify PostgreSQL is running:
psql --host localhost -U postgres -c 'SELECT 1'  # If psql is installed

# Check firewall (if remote):
telnet prod.example.com 5432
```

Error: Missing module (openpyxl, fpdf2, etc.)

A required optional dependency is not installed.

```
# For gis-catalogue (uses openpyxl for Excel):
pip install -e ".[dev]"

# For PDF generation:
pip install fpdf2

# For web mapping:
pip install -e ".[web]"

# For everything:
pip install -e ".[dev,server,web,integration]"
```

Generated script fails with ImportError

The generated script imports a module that is not installed in the target environment.

```
# For PyQGIS scripts, run inside QGIS Python console
# For ArcPy scripts, run in ArcGIS Pro Python environment
# For Folium/Kepler/pydeck, install web dependencies:
pip install -e ".[web]"
```

Generated scripts use minimal imports: qgis.core (PyQGIS), arcpy (ArcPy), folium (Folium), etc. Ensure the target platform has its SDK installed.

Generated script missing columns

The script ran but some attribute columns are missing from the output layer.

```
# This usually means the column was dropped in a dissolve or spatial join.
# Check the source table:
SELECT * FROM public.my_table LIMIT 1;

# And verify the generated script includes all columns in the operation
```

The generators include all columns by default unless an operation (e.g., dissolve) drops them. Review the operation parameters in the generated script.

20. Architecture & Design

System overview

gis-codegen is a code generation pipeline with three main stages:

```
PostGIS Database
      |
      v
  Extractor (extract_schema)
      |
      | Returns: Schema dict (JSON structure)
      |
      v
  Generator (generate_* functions)
      |
      | Returns: Python code string (or XML for QGS)
      |
      v
  Output (stdout or file)
```

Schema dict structure

extract_schema() returns a normalized dictionary representing all spatial layers in the database. This structure is the lingua franca between extraction and generation.

```
{
  "database": "my_gis_db",
  "host": "localhost",
  "layer_count": 3,
  "layers": [
    {
      "schema": "public",
      "table": "buildings",
      "qualified_name": "public.buildings",
      "geometry": {
        "column": "geom",
        "type": "MULTIPOLYGON",
        "srid": 4326
      },
      "columns": [
        {"name": "gid", "data_type": "integer", "nullable": false},
        {"name": "height", "data_type": "double precision", "nullable": true}
      ],
      "primary_keys": ["gid"],
      "row_count_estimate": 42000,
      "comment": "Optional table description"
    }
  ]
}
```

Generator functions

Each platform has a generator function that takes a schema dict and returns a string of code ready to run.

```
from gis_codegen import generate_pyqgis, generate_arcpy

schema = extract_schema(conn)
```

```
pyqgis_code = generate_pyqgis(schema, operations=['buffer'])
arcpy_code = generate_arcpy(schema, operations=['buffer'])

# Write to file or stdout
with open('load.py', 'w') as f:
    f.write(pyqgis_code)
```

Operation blocks

Operations are Python code blocks inserted into the generated script. Each operation (buffer, clip, dissolve, etc.) is implemented as a template that receives the layer name and geometry type as context.

```
Valid operations: reproject, export, buffer, clip, select, dissolve,
                  centroid, field_calc, spatial_join, intersect,
                  extrude, z_stats, floor_ceiling, volume, scene_layer

PyQGIS operations use QGIS Processing algorithms (native:buffer, etc.).
ArcPy operations use ArcGIS Spatial Analyst and Management tools.
```

Type mapping

PostgreSQL data types are mapped to platform-specific types.

```
PostgreSQL          PyQGIS              ArcPy
-----------------------------------------------
integer             QVariant.Int        -  (not used)
text                QVariant.String     TEXT
double precision    QVariant.Double     DOUBLE
boolean             QVariant.Bool       -
timestamp           QVariant.DateTime   DATE
```

Type mappings are defined in generator.py as pg_type_to_pyqgis() and pg_type_to_arcpy() functions. Custom types fall back to STRING.

Catalogue (map-driven generation)

gis-catalogue reads an Excel file where each row represents a "map" (a visualization of one or more layers with a specific symbology).

```
Columns: map_id, theme, symbology_type, status, spatial_layer_type,
         data sources, description, owner, effort, dependencies, ...

Filter: status IN {have, partial} AND spatial_layer_type contains Vector

Symbology dispatch table maps symbology_type to renderer logic:
  "choroplèthe (dégradé)" -> continuous color scale (PyQGIS GraduatedSymbol)
  "choroplèthe catégoriel" -> categorical colors (PyQGIS CategorizedSymbol)
  "points" -> single color (PyQGIS SimpleMarker)
```

One script is generated per included map, with per-map layer selection and symbology applied automatically.

Extension points

Developers can extend gis-codegen by:

- Adding custom generators for new platforms (implement signature: generate_myplatform(schema: dict, operations: list = None) -> str)
- Adding custom operations (add to VALID_OPERATIONS set and implement

_op_MYOP_pyqgis/arcpy functions)
- Adding custom symbology renderers (extend SymbologyRenderer class in catalogue.py)
- Using the public API (connect, extract_schema, generate_*) in custom workflows
- Saving and loading schemas offline for reproducible batch generation

Performance notes

- Unit tests (no database): ~2 seconds
- Extraction with row counts: depends on table size, usually 1-5 seconds
- Extraction with --no-row-counts: <1 second (recommended for large DBs)
- Generation: instant (string building)
- Catalogue: ~1 second per 100 maps (depends on layer count and operations)