Expertises Topologie - Claude et Qgis

Contexte PLU

Pour une couche de zones PLU topologiquement correcte, trois règles sont non-négociables :

1. **Pas de gaps** — tout le territoire communal doit être couvert, sans trou entre zones
2. **Pas d'overlaps** — une parcelle n'appartient qu'à une seule zone
3. **Géométries valides** — pas de self-intersection, pas de géométrie nulle

Ces trois vérifications + corrections sont pilotables intégralement depuis Claude Code via le MCP QGIS, sans manipulation manuelle.

Bloc 1 — Configuration du snapping avant saisie

À lancer en début de session d'édition sur une couche PLU. Configure l'environnement QGIS pour que la saisie soit topologiquement propre dès le départ.

```
from qgis.core import QgsSnappingConfig, QgsTolerance, QgsProject

proj = QgsProject.instance()
snap_cfg = proj.snappingConfig()

snap_cfg.setEnabled(True)
snap_cfg.setMode(QgsSnappingConfig.SnappingMode.AllLayers)
snap_cfg.setType(QgsSnappingConfig.SnappingType.VertexAndSegment)
snap_cfg.setTolerance(10.0)
snap_cfg.setUnits(QgsTolerance.UnitType.Pixels)
```

```
snap_cfg.setIntersectionSnapping(True)   # accrochage sur les intersections

proj.setSnappingConfig(snap_cfg)
proj.setTopologicalEditing(True)          # partage de noeuds entre couches
proj.setAvoidIntersectionsMode(
    QgsProject.AvoidIntersectionsMode.AvoidIntersectionsLayers
)  # évite automatiquement les overlaps lors de la saisie
```

Paramètres clés :

Paramètre	Valeur	Effet
`AllLayers`	mode	snap sur toutes les couches visibles
`VertexAndSegment`	type	accroche vertex ET bord
`10 px`	tolérance	adapté à la saisie courante PLU
`TopologicalEditing`	True	noeuds partagés entre polygones adjacents
`AvoidIntersectionsLayers`	mode	saisie d'un polygone qui "découpe" automatiquement ses voisins

Piège : `setType()` attend `QgsSnappingConfig.SnappingType` — pas `Qgis.SnappingType` ni `Qgis.SnappingTypes`.

Bloc 2 — Audit topologique

Routine complète : validité + overlaps + gaps sur n'importe quelle couche polygone.

```
import processing
from qgis.core import QgsProject

def audit_topo(layer, unique_id='fid', gap_threshold=0, min_overlap_area=0):
    """
    Audit topologique d'une couche polygone PLU.
    Retourne un dict avec les couches d'erreurs et un résumé.
    """
    rapport = {}

    # 1. Validité géométrique (GEOS)
    r_val = processing.run("native:checkvalidity", {
```

```
    'INPUT_LAYER': layer,   # NOTE : INPUT_LAYER, pas INPUT
    'METHOD': 2,            # GEOS
    'IGNORE_RING_SELF_INTERSECTION': False,
    'VALID_OUTPUT': 'memory:',
    'INVALID_OUTPUT': 'memory:',
    'ERROR_OUTPUT': 'memory:'
})
rapport['validite'] = {
    'valides': r_val['VALID_COUNT'],
    'invalides': r_val['INVALID_COUNT'],
    'erreurs': r_val['ERROR_COUNT'],
    'layer_invalides': r_val['INVALID_OUTPUT'],
    'layer_erreurs': r_val['ERROR_OUTPUT']
}

# 2. Overlaps
r_ov = processing.run("native:checkgeometryoverlap", {
    'INPUT': layer,
    'UNIQUE_ID': unique_id,
    'MIN_OVERLAP_AREA': min_overlap_area,
    'TOLERANCE': 8,
    'ERRORS': 'memory:',
    'OUTPUT': 'memory:'
})
rapport['overlaps'] = {
    'count': r_ov['ERRORS'].featureCount(),
    'layer_errors': r_ov['ERRORS'],
    'layer_features': r_ov['OUTPUT']
}

# 3. Gaps
r_gap = processing.run("native:checkgeometrygap", {
    'INPUT': layer,
    'UNIQUE_ID': unique_id,
    'GAP_THRESHOLD': gap_threshold,
    'TOLERANCE': 8,
    'NEIGHBORS': 'memory:',
    'ERRORS': 'memory:',
    'OUTPUT': 'memory:'
})
```

```
    rapport['gaps'] = {
        'count': r_gap['OUTPUT'].featureCount(),
        'layer_gaps': r_gap['OUTPUT'],
        'layer_neighbors': r_gap['NEIGHBORS'],  # requis pour fixgeometrygap
        'layer_errors': r_gap['ERRORS']
    }

    # Résumé lisible
    print(f"=== Audit topo : {layer.name()} ({layer.featureCount()} features) ===")
    print(f"Validité  : {rapport['validite']['valides']} OK / "
          f"{rapport['validite']['invalides']} invalides")
    print(f"Overlaps  : {rapport['overlaps']['count']} erreur(s)")
    print(f"Gaps      : {rapport['gaps']['count']} trou(s)")
    ok = (rapport['validite']['invalides'] == 0 and
          rapport['overlaps']['count'] == 0 and
          rapport['gaps']['count'] == 0)
    print(f"Résultat  : {'PROPRE' if ok else 'ERREURS DETECTEES'}")

    return rapport
```

Usage :

```
from qgis.core import QgsProject
layer = QgsProject.instance().mapLayersByName("zones_plu")[0]
rapport = audit_topo(layer, unique_id='fid')
```

Piège critique : `native:checkvalidity` → paramètre `INPUT_LAYER` (pas `INPUT`). Erreur silencieuse sinon.

Bloc 3 — Correction automatique

S'appuie sur les couches d'erreurs produites par le Bloc 2.

3a. Corriger les overlaps

```
r_fix_ov = processing.run("native:fixgeometryoverlap", {
    'INPUT': layer,
```

```
    'ERRORS': rapport['overlaps']['layer_errors'],
    'UNIQUE_ID': 'fid',
    'OVERLAP_FEATURE_UNIQUE_IDX': 'gc_overlap_fid',  # champ produit par checkgeometryoverlap
    'ERROR_VALUE_ID': 'gc_error',
    'OUTPUT': 'memory:',
    'REPORT': 'memory:',
    'TOLERANCE': 8
})
layer_corrige = r_fix_ov['OUTPUT']
print(f"Overlaps corrigés : {r_fix_ov['REPORT'].featureCount()} opérations")
```

3b. Combler les gaps

3 méthodes disponibles :

Valeur	Méthode	Usage recommandé
`0`	Ajouter à la plus longue bordure en commun	PLU — zone voisine qui partage le plus de frontière
`1`	Créer une nouvelle entité	si le gap est une zone à part entière
`2`	Ajouter à la plus grande zone voisine	quand la surface prime

```
r_fix_gap = processing.run("native:fixgeometrygap", {
    'INPUT': layer,
    'NEIGHBORS': rapport['gaps']['layer_neighbors'],  # issu de checkgeometrygap
    'GAPS': rapport['gaps']['layer_gaps'],
    'METHOD': 0,    # 0 = plus longue bordure (recommandé PLU)
    'UNIQUE_ID': 'fid',
    'ERROR_ID_IDX': 'gc_errorid',
    'OUTPUT': 'memory:',
    'REPORT': 'memory:',
    'TOLERANCE': 8
})
layer_sans_gap = r_fix_gap['OUTPUT']
print(f"Gaps comblés : {r_fix_gap['REPORT'].featureCount()} opérations")
```

3c. Corriger les géométries invalides (self-intersections, anneaux)

```
r_fix_val = processing.run("native:fixgeometries", {
    'INPUT': rapport['validite']['layer_invalides'],
    'METHOD': 1,    # 1 = structure linéaire (robuste)
    'OUTPUT': 'memory:'
})
layer_valide = r_fix_val['OUTPUT']
```

Bloc 4 — Accrochage post-saisie (snapgeometries)

Quand une couche a été saisie sans snapping actif, ou importée avec de micro-décalages entre polygones adjacents. Accroche les géométries d'une couche sur une couche de référence (ex : limites communales, autre zonage).

8 comportements disponibles :

Valeur	Comportement	Usage
`0`	Aligner les nœuds, ajoute sommets si besoin	**défaut recommandé**
`1`	Point le plus proche, ajoute sommets	si nœuds mal alignés
`2`	Aligner les nœuds, sans ajout de sommet	quand on veut conserver la densité
`4`	Déplacer uniquement les extrémités (nœuds)	micro-corrections de bord
`6`	Extrémités sur extrémités uniquement	lignes/filaires

```
r_snap = processing.run("native:snapgeometries", {
    'INPUT': layer_a_corriger,
    'REFERENCE_LAYER': layer_reference,
    'TOLERANCE': 0.5,    # en unités de la couche (mètres si Lambert 93)
    'BEHAVIOR': 0,
    'OUTPUT': 'memory:'
})
layer_snapped = r_snap['OUTPUT']
```

Recommandation PLU : tolérance 0.1–0.5 m en EPSG:2154 (Lambert 93). Trop grande → déformation des géométries.

Bloc 5 — Reconstruction automatique depuis le cadastre

Quand une zone (ex : périmètre PLU, emprise de projet) a été saisie grossièrement et doit être recalée exactement sur les limites parcellaires cadastrales. Claude reconstruit le polygone par dissolution des parcelles qui chevauchent la zone rough ; l'opérateur complète ensuite manuellement les parties sans référence cadastrale (domaine public non-parcellaire : voiries, cours d'eau).

Validé le 2026-06-21 sur `06079-Minelle.gpkg` (Mandelieu-la-Napoule) : 11 parcelles intersectantes → 2 retenues (overlap ≥ 50%) → 137 004 m², 31 nœuds, géométrie valide.

Principe

1. Fixer les géométries du cadastre (WFS souvent invalide)
2. Extraire les parcelles qui intersectent la zone rough
3. Calculer le % de chevauchement pour chaque parcelle → garder celles à ≥ 50%
4. Dissoudre → multiparttosingleparts → garder la plus grande part
5. Écrire dans le GeoPackage en retirant d'abord les couches QGIS (file locking Windows)

Code complet

```
import processing
import urllib.parse
from qgis.core import (
    QgsProject, QgsVectorFileWriter, QgsWkbTypes,
    QgsFields, QgsMemoryProviderUtils
)

# Pré-requis : layer_cad (cadastre WFS reprojeté EPSG:3857)
#              layer_zone (zone rough à recaler)
#              gpkg_path (chemin du GeoPackage à écrire)
```

```
# 1. Fixer les géométries cadastrales (WFS Géoplateforme souvent invalide)
cad_fixed = processing.run("native:fixgeometries", {
    'INPUT': layer_cad,
    'METHOD': 1,    # structure linéaire, robuste
    'OUTPUT': 'memory:'
})['OUTPUT']

# 2. Extraire les parcelles qui intersectent la zone rough
parcelles_in = processing.run("native:extractbylocation", {
    'INPUT': cad_fixed,
    'PREDICATE': [0],   # intersects
    'INTERSECT': layer_zone,
    'OUTPUT': 'memory:'
})['OUTPUT']

# 3. Calculer le % de chevauchement et retenir celles ≥ 50%
geom_zone = next(layer_zone.getFeatures()).geometry()
a_inclure = []
for f in parcelles_in.getFeatures():
    inter = f.geometry().intersection(geom_zone)
    pct = inter.area() / f.geometry().area() * 100
    if pct >= 50:
        a_inclure.append(f['gid'])

print(f"{parcelles_in.featureCount()} parcelles intersectantes → "
      f"{len(a_inclure)} retenues (overlap ≥ 50%)")
print(f"GIDs retenus : {a_inclure}")

# 4. Sélectionner + dissoudre
ids_str = ', '.join(str(i) for i in a_inclure)
parcelles_sel = processing.run("native:extractbyexpression", {
    'INPUT': cad_fixed,
    'EXPRESSION': f'"gid" IN ({ids_str})',
    'OUTPUT': 'memory:'
})['OUTPUT']

dissolved = processing.run("native:dissolve", {
    'INPUT': parcelles_sel,
    'FIELD': [],
```

```
    'SEPARATE_DISJOINT': False,
    'OUTPUT': 'memory:'
})['OUTPUT']

# 5. Single part → garder la plus grande
single = processing.run("native:multiparttosingleparts", {
    'INPUT': dissolved,
    'OUTPUT': 'memory:'
})['OUTPUT']

largest = max(single.getFeatures(), key=lambda f: f.geometry().area())
print(f"Résultat : {largest.geometry().area():.0f} m², "
      f"{largest.geometry().constGet().nCoordinates()} nœuds")

layer_main = QgsMemoryProviderUtils.createMemoryLayer(
    "main", QgsFields(), QgsWkbTypes.Type.Polygon, single.crs()
)
layer_main.dataProvider().addFeature(largest)

# 6. Écrire dans le GeoPackage
# CRITIQUE : retirer toutes les couches QGIS référençant le fichier avant écriture
for name in ["06079-Minelle"]:
    for lyr in QgsProject.instance().mapLayersByName(name):
        QgsProject.instance().removeMapLayer(lyr)

options = QgsVectorFileWriter.SaveVectorOptions()
options.driverName = "GPKG"
options.layerName = "06079-Minelle"
options.actionOnExistingFile = QgsVectorFileWriter.ActionOnExistingFile.CreateOrOverwriteLayer

# writeAsVectorFormatV3 retourne un tuple de 4 valeurs
res = QgsVectorFileWriter.writeAsVectorFormatV3(
    layer_main, gpkg_path, QgsCoordinateTransformContext(), options
)
# res = (error_code, error_message, filename, layername)
if res[0] == 0:
    print(f"Écrit : {gpkg_path} → couche '{options.layerName}'")
else:
    print(f"ERREUR {res[0]} : {res[1]}")
```

```
# 7. Recharger dans QGIS pour vérification
layer_reload = QgsVectorLayer(
    f"{gpkg_path}|layername={options.layerName}",
    options.layerName, "ogr"
)
QgsProject.instance().addMapLayer(layer_reload, False)
root = QgsProject.instance().layerTreeRoot()
root.insertLayer(0, layer_reload)
iface.mapCanvas().setExtent(layer_reload.extent())
iface.mapCanvas().refresh()
print("Couche rechargée et zoomée.")
```

Pièges spécifiques

Sujet	Constat
`writeAsVectorFormatV3` **retourne 4 valeurs**	`(error_code, error_message, filename, layername)` — pas 2. Déstructurer en conséquence.
File locking Windows	QGIS garde un handle sur le fichier GeoPackage. Retirer toutes les couches référençant le fichier avec `removeMapLayer()` avant d'écrire.
`CreateOrOverwriteLayer` **pas** `CreateOrOverwriteFile`	`CreateOrOverwriteFile` supprime tout le fichier (détruisant les autres couches). `CreateOrOverwriteLayer` ne touche qu'à la couche cible.
`native:savefeatures` **échoue si fichier existe**	"A file system object already exists" — utiliser `writeAsVectorFormatV3` à la place.
MultiPolygon → `asPolygon()` **fail**	Si les parcelles retenues ne sont pas adjacentes, le dissolve produit un MultiPolygon. Appliquer `multiparttosingleparts` + `max(..., key=lambda f: f.geometry().area())` pour garder la plus grande part.
Cadastre WFS invalide	La Géoplateforme retourne parfois des géométries invalides (ex : feature 15026). Toujours fixer avec `native:fixgeometries METHOD=1` avant usage comme couche de référence.

Workflow collaboratif

Claude reconstruit le périmètre sur les limites parcellaires exactes. L'opérateur complète manuellement les parties manquantes non-parcellaires (domaine public : voiries, cours d'eau, espaces naturels non cadastrés). Les deux résultats sont complémentaires : la précision automatique sur le cadastre + le jugement humain sur le hors-cadastre.

Routine principale — Audit + Rapport

Fonction complète prête à l'emploi. Appeler par Claude en une seule commande.

```
import processing
from qgis.core import QgsProject, QgsSnappingConfig, QgsTolerance

def configurer_snapping_plu():
    proj = QgsProject.instance()
    snap_cfg = proj.snappingConfig()
    snap_cfg.setEnabled(True)
    snap_cfg.setMode(QgsSnappingConfig.SnappingMode.AllLayers)
    snap_cfg.setType(QgsSnappingConfig.SnappingType.VertexAndSegment)
    snap_cfg.setTolerance(10.0)
    snap_cfg.setUnits(QgsTolerance.UnitType.Pixels)
    snap_cfg.setIntersectionSnapping(True)
    proj.setSnappingConfig(snap_cfg)
    proj.setTopologicalEditing(True)
    proj.setAvoidIntersectionsMode(
        QgsProject.AvoidIntersectionsMode.AvoidIntersectionsLayers)
    print("Snapping PLU configuré")

def audit_topo(layer, unique_id='fid', gap_threshold=0, min_overlap_area=0):
    r_val = processing.run("native:checkvalidity", {
        'INPUT_LAYER': layer, 'METHOD': 2,
        'IGNORE_RING_SELF_INTERSECTION': False,
        'VALID_OUTPUT': 'memory:', 'INVALID_OUTPUT': 'memory:', 'ERROR_OUTPUT': 'memory:'
    })
    r_ov = processing.run("native:checkgeometryoverlap", {
        'INPUT': layer, 'UNIQUE_ID': unique_id,
        'MIN_OVERLAP_AREA': min_overlap_area, 'TOLERANCE': 8,
        'ERRORS': 'memory:', 'OUTPUT': 'memory:'
    })
    r_gap = processing.run("native:checkgeometrygap", {
        'INPUT': layer, 'UNIQUE_ID': unique_id,
```

```
        'GAP_THRESHOLD': gap_threshold, 'TOLERANCE': 8,
        'NEIGHBORS': 'memory:', 'ERRORS': 'memory:', 'OUTPUT': 'memory:'
    })
    rapport = {
        'layer': layer,
        'unique_id': unique_id,
        'validite': {
            'valides': r_val['VALID_COUNT'],
            'invalides': r_val['INVALID_COUNT'],
            'layer_invalides': r_val['INVALID_OUTPUT']
        },
        'overlaps': {
            'count': r_ov['ERRORS'].featureCount(),
            'layer_errors': r_ov['ERRORS'],
            'layer_features': r_ov['OUTPUT']
        },
        'gaps': {
            'count': r_gap['OUTPUT'].featureCount(),
            'layer_gaps': r_gap['OUTPUT'],
            'layer_neighbors': r_gap['NEIGHBORS'],
            'layer_errors': r_gap['ERRORS']
        }
    }
    ok = (r_val['INVALID_COUNT'] == 0 and
          r_ov['ERRORS'].featureCount() == 0 and
          r_gap['OUTPUT'].featureCount() == 0)
    print(f"\n=== Audit topo : {layer.name()} ===")
    print(f"  Géométries invalides : {r_val['INVALID_COUNT']}")
    print(f"  Overlaps             : {r_ov['ERRORS'].featureCount()}")
    print(f"  Gaps                 : {r_gap['OUTPUT'].featureCount()}")
    print(f"  => {'PROPRE' if ok else 'ERREURS — voir rapport'}")
    return rapport

# Usage type
# layer = QgsProject.instance().mapLayersByName("zones_plu")[0]
# configurer_snapping_plu()
# rapport = audit_topo(layer)
```

Pièges à retenir

Sujet	Constat
`INPUT_LAYER` **vs** `INPUT`	`native:checkvalidity` exige `INPUT_LAYER`, tous les autres utilisent `INPUT`. Erreur silencieuse sinon.
`QgsSnappingConfig.SnappingType`	`setType()` attend ce type précis — pas `Qgis.SnappingType` ni `Qgis.SnappingTypes`.
NEIGHBORS obligatoire pour fixgeometrygap	La couche `NEIGHBORS` produite par `checkgeometrygap` doit être conservée et passée à `fixgeometrygap`.
Tolérance snapgeometries en unités couche	Si la couche est en Lambert 93 (mètres), 0.5 = 50 cm. Ne pas confondre avec la tolérance en pixels du snapping canvas.
METHOD 0 pour PLU gaps	"Plus longue bordure en commun" est la méthode la plus cohérente pour les zonages PLU (le gap va à la zone avec laquelle il partage le plus de frontière).
AvoidIntersectionsLayers	Mode le plus adapté PLU — évite les overlaps lors de la saisie mais uniquement sur les couches cochées dans les paramètres du projet.
`writeAsVectorFormatV3` **retourne 4 valeurs**	`(error_code, error_message, filename, layername)` — pas 2.
File locking GeoPackage Windows	Retirer toutes les couches QGIS référençant le fichier avant écriture, utiliser `CreateOrOverwriteLayer`.

Statut

- **Validé le 2026-06-21** : Blocs 1–5 testés sur QGIS 4.0.3 via MCP Claude Code.
  - Bloc 1 : snapping PLU
  - Bloc 2 : `checkvalidity`, `checkgeometryoverlap`, `checkgeometrygap`
  - Bloc 5 : reconstruction `06079-Minelle.gpkg` depuis cadastre WFS → 137 004 m², 31 nœuds, valide
- **À tester** : `fixgeometryoverlap`, `fixgeometrygap`, `snapgeometries` sur une vraie couche PLU avec erreurs.

Revision #3
Created 2026-06-20 22:21:47 UTC by Julien
Updated 2026-06-20 23:04:17 UTC by Julien