05_Claude et les MCP

- Quelles opportunités entre logiciels opensource et les IA
- Retours de Claude et propositions d'intégration
- QGIS MCP
- Inkscape MCP
- LibreOffice MCP

Quelles opportunités entre logiciels opensource et les IA

Voici une liste des **logiciels open source** dans les domaines de l'**infographie, la PAO, la CAO et la géomatique** qui, en **juin 2026**, bénéficient d'une **connexion MCP (Model Context Protocol)** ou pour lesquels des **serveurs MCP** existent ou sont en développement pour permettre à une IA de prendre la main sur la production de documents ou d'automatiser des tâches.

□□ Logiciels compatibles MCP (ou avec serveurs MCP disponibles)

□□ Infographie & PAO

- **Scribus** *Logiciel de PAO open source pour la mise en page professionnelle (brochures, magazines, livres).* **Statut MCP** : Des serveurs MCP communautaires permettent d'interagir avec les fichiers Scribus (lecture, modification, export) via des scripts Python ou des connecteurs dédiés. Cela permet à une IA de générer ou modifier des mises en page automatiquement.
- **Inkscape** *Éditeur de graphiques vectoriels (SVG) open source.* **Statut MCP** : Des serveurs MCP existent pour manipuler des fichiers SVG (création, édition, export) via des outils comme **Puppeteer** ou des scripts Python. Une IA peut ainsi générer ou modifier des visuels vectoriels.
- **GIMP** *Logiciel de retouche photo et de création graphique open source.* **Statut MCP** : Des connecteurs MCP permettent d'automatiser des tâches de retouche ou de génération

d’images via des scripts (Python, Bash) ou des API externes.

□□ CAO (Conception Assistée par Ordinateur)

- **FreeCAD** *Logiciel de CAO 3D paramétrique open source.* **Statut MCP** : Des serveurs MCP permettent d’interagir avec les fichiers FreeCAD (`.FCStd`) pour automatiser la création ou la modification de modèles 3D. Une IA peut ainsi générer des pièces ou des assemblages à partir de prompts textuels.
- **Blender** *Logiciel open source de modélisation 3D, animation et rendu.* **Statut MCP** : **Blender est explicitement cité comme compatible avec MCP** dans la documentation officielle. Des serveurs MCP permettent de contrôler Blender via Python (API Blender + MCP), ce qui permet à une IA de créer, modifier ou rendre des scènes 3D, ou même d’automatiser des workflows complexes (ex : génération de visuels pour l’architecture ou le design industriel).
- **OpenSCAD** *Logiciel de CAO 3D open source basé sur un langage de script.* **Statut MCP** : Des serveurs MCP permettent d’exécuter des scripts OpenSCAD et de récupérer les modèles 3D générés, ce qui facilite l’automatisation de la conception paramétrique.

□□ Géomatique (SIG & Cartographie)

- **QGIS** *Système d’Information Géographique (SIG) open source le plus répandu.* **Statut MCP** : Des serveurs MCP permettent d’interagir avec QGIS via **PyQGIS** (Python) ou des API REST. Une IA peut ainsi :
  - Charger des couches géospatiales (Shapefile, GeoJSON, etc.).
  - Effectuer des analyses spatiales (requêtes, calculs de distances, etc.).
  - Générer des cartes ou des rapports automatiques.
- **GRASS GIS** *SIG open source avancé pour l’analyse spatiale.* **Statut MCP** : Compatible via des scripts Python ou des connecteurs MCP dédiés pour automatiser des traitements géomatiques (ex : classification d’images satellite, modélisation de terrains).
- **GDAL/OGR** *Bibliothèque open source pour la manipulation de données géospatiales (raster/vectoriel).* **Statut MCP** : Des serveurs MCP permettent d’utiliser GDAL pour convertir, traiter ou analyser des données géospatiales (ex : transformation de formats, extraction de métadonnées).

- **PostGIS** *Extension spatiale pour PostgreSQL.* **Statut MCP** : Des serveurs MCP permettent d'interroger des bases de données PostGIS en langage naturel (ex : "Trouve tous les bâtiments à moins de 500m d'une rivière") et de générer des rapports ou des cartes automatiques.

□□ Comment ça marche ?

Le **Model Context Protocol (MCP)** est un **standard ouvert** créé par Anthropic et donné à la **Linux Foundation** en décembre 2025. Il permet à une IA (comme Claude, ChatGPT, ou des modèles locaux) de :

- **Lire/écrire des fichiers** (ex : `.svg`, `.fcstd`, `.qgs`).
- **Exécuter des fonctions** (ex : lancer un rendu dans Blender, une requête SQL dans PostGIS).
- **Interagir avec des API externes** (ex : Google Drive, GitHub, etc.).

Exemples concrets en 2026 :

- Un agent IA peut **créer une carte thématique dans QGIS** à partir d'une demande en langage naturel.
- Une IA peut **générer un modèle 3D dans FreeCAD** en suivant un cahier des charges textuel.
- Un workflow automatisé peut **produire une affiche dans Scribus** en combinant du texte généré par IA et des images retouchées via GIMP.

□□ Où trouver les serveurs MCP pour ces logiciels ?

- **Registry officiel** : GitHub MCP Servers (plus de 1 000 serveurs open source en 2026).
- **Exemples de serveurs** :
  - MCP pour Blender
  - MCP pour QGIS

- MCP pour FreeCAD
- MCP pour PostGIS .

□□ Cas d’usage avancés

- **Infographie** : Génération automatique de visuels pour les réseaux sociaux (Inkscape + IA).
- **PAO** : Création de catalogues produits à partir d’une base de données (Scribus + MCP + PostGIS).
- **CAO** : Optimisation topologique de pièces mécaniques (FreeCAD + IA + MCP).
- **Géomatique** : Analyse de risques naturels à partir de données ouvertes (QGIS + GDAL + MCP).

⚠ Limites et précautions

- **Sécurité** : MCP permet un accès profond aux outils. Il faut **contrôler les permissions** pour éviter des actions non désirées (ex : suppression de fichiers).
- **Maturity** : Tous les logiciels n’ont pas encore de serveurs MCP officiels, mais la communauté développe activement des connecteurs.
- **Performances** : Pour les logiciels lourds (Blender, QGIS), une **machine puissante** est recommandée pour éviter les lenteurs.

□□ Pour aller plus loin

- Documentation officielle MCP
- Liste des serveurs MCP communautaires
- Tutoriel : Connecter Blender à une IA via MCP

Question : Souhaites-tu que je te guide pour configurer un serveur MCP spécifique (ex : Blender ou QGIS) ou explorer un cas d’usage précis dans ton domaine ?

Retours de Claude et propositions d'intégration

Contexte

Analyse rédigée par Claude Code (Sonnet 4.6) le 2026-06-20, à partir d'une session de travail sur le PC Windows (eliob).

Logiciels utilisés par Julien : **QGIS**, **Inkscape**, **GIMP**, **Scribus**. Ces quatre outils forment un workflow cohérent : QGIS pour la donnée géospatiale, Inkscape et Scribus pour la mise en page cartographique et la PAO, GIMP pour le traitement d'image.

Note critique sur la page 227

La page 227 ("Quelles opportunités entre logiciels opensource et les IA") a vraisemblablement été générée par une IA en mode chatbot. Les liens GitHub vers `modelcontextprotocol/servers/tree/main/src/servers/blender`, `.../qgis`, `.../freecad`, `.../postgis` sont **probablement hallusinés** — ces sous-dossiers n'existent pas dans le dépôt officiel MCP. À vérifier systématiquement avant d'agir sur ces références.

Analyse par logiciel

QGIS — Priorité 1 (le plus mature et le plus pertinent)

Ce qui existe réellement : Le projet `qgis-mcp` (disponible sur GitHub, maintenu par la communauté) expose QGIS comme un serveur MCP local. Il s'installe comme un plugin Python dans QGIS et ouvre un socket local sur lequel Claude peut envoyer des commandes PyQGIS.

Capacités concrètes :

- Charger des couches (Shapefile, GeoJSON, WMS, PostGIS)
- Lancer des algorithmes de traitement (buffer, intersection, statistiques zonales)
- Modifier la symbologie d'une couche
- Exporter une carte en PNG/PDF

Avantage spécifique à Julien : La base PostGIS `alteris_geo` sur le VPS Alteris (79.137.14.202:5432) est déjà accessible. Un workflow QGIS + MCP + PostGIS permettrait à Claude de : interroger la base en SQL spatial → charger le résultat dans QGIS → générer une carte thématique → exporter en PDF, le tout en langage naturel.

Contrainte : QGIS doit être ouvert sur la machine locale. Le MCP communique via socket local (pas de pilotage à distance).

Scribus — Priorité 2 (fort potentiel pour les rendus cartographiques)

Ce qui existe réellement : Scribus expose une API Python complète via son module `scribus` (accessible en mode headless : `scribus --python-script myscript.py`). Pas de serveur MCP publié, mais un **MCP custom est trivial à écrire** : un script Python qui reçoit des commandes MCP et les traduit en appels `scribus.*`.

Capacités concrètes via Python headless :

- Créer un document depuis un gabarit `.sla`
- Injecter du texte dans des cadres de texte nommés
- Placer des images
- Exporter en PDF

Cas d'usage concret pour Julien : Générer automatiquement une fiche de synthèse cartographique (titre, carte exportée depuis QGIS, texte de légende) en combinant la carte produite par QGIS MCP et un gabarit Scribus.

Contrainte : Scribus headless est instable sur certaines versions — à tester avec la version installée. La génération de gabarits `.sla` de référence est un prérequis.

Inkscape — Priorité 3 (manipulation SVG, pas contrôle UI)

Ce qui existe réellement : Pas de serveur MCP dédié pour contrôler l'interface Inkscape. En revanche, deux approches sont réalistes :

1. **Manipulation directe du SVG** — le format natif d'Inkscape est du XML/SVG standard. Claude peut générer ou modifier des fichiers SVG avec Python (`lxml`, `svgwrite`) sans lancer Inkscape.
2. **Inkscape CLI** — `inkscape --actions="verb1;verb2"` permet des opérations batch (export PNG/PDF, conversion de formats) pilotables depuis un MCP.

Cas d'usage concret pour Julien : Modifier programmatiquement une carte SVG exportée depuis QGIS (changer des couleurs, ajouter un texte, insérer un logo) avant intégration dans Scribus.

Contrainte : L'automatisation reste au niveau fichier, pas au niveau interactif. Pour un usage cartographique, QGIS couvre déjà la plupart des besoins de rendu.

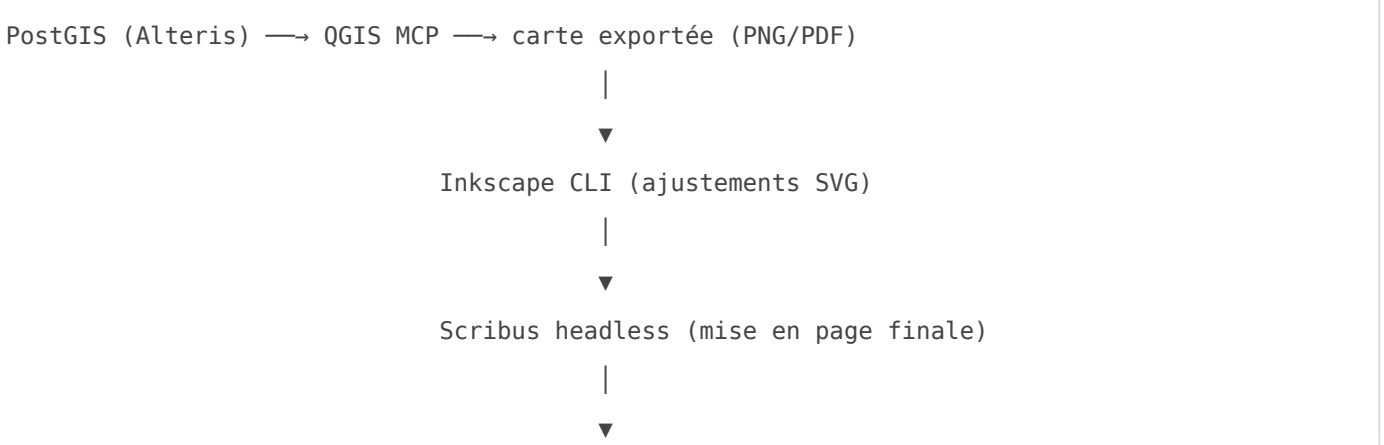
GIMP — Priorité 4 (batch uniquement)

Ce qui existe réellement : GIMP peut être piloté en batch via `gimp --no-interface --batch='(script-fu-batch-list ...)'` ou via Python-Fu (`gimp --batch`). Un MCP custom est faisable.

Cas d'usage concret pour Julien : Traitement en masse d'images pour intégration dans des documents Scribus (redimensionnement, recadrage, conversion CMJN, compression). Moins critique si StirlingPDF couvre déjà les besoins de compression.

Contrainte : GIMP batch est lent au démarrage. Pour du traitement image simple, des alternatives Python (`Pillow`, `ImageMagick`) sont plus légères à wraper en MCP.

Proposition de workflow intégré



```
StirlingPDF (compression)
```

Claude pilote l'ensemble de la chaîne : de la requête spatiale jusqu'au document final, sans intervention manuelle.

Ordre de priorité pour une mise en œuvre

Priorité	Outil	Action	Effort
1	QGIS	Installer le plugin `qgis-mcp`, tester avec une couche PostGIS	Faible
2	Scribus	Écrire un MCP custom Python headless, créer un gabarit `.sla` de référence	Moyen
3	Inkscape	MCP de manipulation SVG via `lxml` + CLI export	Faible
4	GIMP	MCP batch Python-Fu	Moyen (faible priorité)

Recommandation : commencer par QGIS. C'est le maillon central du workflow de Julien, le MCP existe déjà, et la base PostGIS Alteris est immédiatement exploitable.

QGIS MCP

Références

- Plugin officiel : plugins.qgis.org/plugins/qgis_mcp_plugin — version 0.5.0, compatible QGIS 3.28–4.x
- Dépôt GitHub : nkarasiak/qgis-mcp — 102 outils MCP (couches, traitements, symbologie, export, mise en page/atlas, SQL cross-couches)

Architecture

```
Claude Code ↔ Serveur MCP Python (uvx) ↔ socket TCP local ↔ Plugin QGIS (dock "QGIS MCP")
```

Le plugin QGIS crée un serveur TCP local. Le serveur MCP Python (lancé par Claude Code via `uvx`) s'y connecte. QGIS doit être ouvert et le serveur démarré avant de lancer Claude Code.

Prérequis

Installer `uv` (gestionnaire de paquets Python) si absent :

```
winget install astral-sh.uv
```

Installation

1. Plugin dans QGIS

`Extensions` > `Installer/Gérer les extensions` > chercher **QGIS MCP** > Installer.

Redémarrer QGIS, puis cliquer **Start Server** dans le dock QGIS MCP.

2. Enregistrer le MCP dans Claude Code

```
claude mcp add -s user qgis -- uvx --from https://github.com/nkarasiak/qgis-mcp/archive/refs/heads/main.zip qgis-mcp-server
```

`-s user` = enregistrement global (tous les projets Claude Code).

3. Vérifier

Dans une session Claude Code avec QGIS ouvert et le serveur démarré, demander `diagnose` — confirme la synchronisation plugin/serveur MCP.

Ordre de démarrage (à chaque session)

1. Ouvrir QGIS
2. Cliquer **Start Server** dans le dock QGIS MCP
3. Lancer Claude Code

Authentification (optionnel, machines partagées)

Définir `QGIS_MCP_TOKEN=votre-secret` dans l'environnement QGIS, redémarrer le serveur, puis ajouter la même variable à la config MCP :

```
"env": { "QGIS_MCP_TOKEN": "votre-secret" }
```

Capacités principales

Catégorie	Exemples
Couches	Charger Shapefile, GeoJSON, WMS, PostGIS ; lister, supprimer
Traitements	Buffer, intersection, statistiques zonales (1000+ algorithmes)
Symbologie	Modifier couleurs, classification, étiquettes
Export	PNG, PDF, carte mise en page
SQL cross-couches	Requêtes spatiales directement depuis Claude
Mise en page/Atlas	Créer et exporter des atlas cartographiques

---

Lien avec PostGIS Alteris

La base `alteris_geo` (79.137.14.202:5432, user `alteris_admin`) est directement utilisable depuis QGIS MCP : Claude peut interroger PostGIS en SQL spatial, charger le résultat comme couche QGIS, styliser et exporter en PDF — sans intervention manuelle.

Le port 5432 n'est pas exposé publiquement (pare-feu VPS). Connexion via tunnel SSH obligatoire (voir section Tunnel SSH).

Tunnel SSH vers PostGIS Alteris

Option 1 — Tunnel manuel (terminal externe)

Lancer avant QGIS/Claude Code :

```
ssh -L 5433:localhost:5432 debian@79.137.14.202 -N
```

Puis se connecter sur `localhost:5433` au lieu de `79.137.14.202:5432`.

Option 2 — Tunnel automatique via paramiko (PyQGIS) ✓ validé 2026-06-20

Ouvre le tunnel directement depuis la console Python QGIS, sans terminal externe.

Installation de paramiko (une seule fois)

`sys.executable` dans QGIS pointe sur `qgis-bin.exe` — `subprocess` est inutilisable. Utiliser pip en interne :

```
from pip._internal.cli.main import main as pip_main
pip_main(['install', 'paramiko'])
```

Paramiko s'installe dans `C:\Users\eliob\AppData\Roaming\Python\Python312\site-packages` (dossier utilisateur). QGIS ne l'inclut pas automatiquement dans `sys.path` — ajouter à chaque session :

```
import sys
sys.path.insert(0, r'C:\Users\eliob\AppData\Roaming\Python\Python312\site-packages')
import paramiko
print(paramiko.__version__)  # 5.0.0
```

Script du tunnel (à lancer après l'import paramiko)

```
import socket
import threading
import select

class SSHTunnel(threading.Thread):
    def __init__(self, ssh_host, ssh_user, ssh_password,
                 remote_port, local_port=5433, remote_host='127.0.0.1'):
        super().__init__(daemon=True)
        self.ssh_host = ssh_host
        self.ssh_user = ssh_user
        self.ssh_password = ssh_password
        self.remote_host = remote_host
        self.remote_port = remote_port
        self.local_port = local_port
        self._stop = threading.Event()
        self.transport = None
```

```
        self.server_sock = None

    def run(self):
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect(self.ssh_host, username=self.ssh_user, password=self.ssh_password)
        self.transport = client.get_transport()
        self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server_sock.bind(('127.0.0.1', self.local_port))
        self.server_sock.listen(5)
        self.server_sock.settimeout(1.0)
        print(f"[Tunnel] localhost:{self.local_port} → {self.ssh_host} →
{self.remote_host}:{self.remote_port}")
        while not self._stop.is_set():
            try:
                conn, _ = self.server_sock.accept()
                threading.Thread(target=self._forward, args=(conn,), daemon=True).start()
            except socket.timeout:
                continue
        self.server_sock.close()
        client.close()
        print("[Tunnel] Arrêté.")

    def _forward(self, local_conn):
        try:
            chan = self.transport.open_channel(
                'direct-tcpip',
                (self.remote_host, self.remote_port),
                local_conn.getpeername()
            )
        except Exception as e:
            print(f"[Tunnel] Erreur canal : {e}")
            local_conn.close()
            return
        while True:
            r, _, _ = select.select([local_conn, chan], [], [], 5)
            if local_conn in r:
                data = local_conn.recv(4096)
                if not data:
```

```
                    break
                chan.send(data)
            if chan in r:
                data = chan.recv(4096)
                if not data:
                    break
                local_conn.send(data)
        chan.close()
        local_conn.close()

    def stop(self):
        self._stop.set()



tunnel = SSHTunnel('79.137.14.202', 'debian', 'RAW+NEXTE!', remote_port=5432, local_port=5433)
tunnel.start()

import time; time.sleep(1)
s = socket.socket(); s.settimeout(3)
try:
    s.connect(('127.0.0.1', 5433)); print("Tunnel OK")
except Exception as e:
    print(f"Tunnel KO : {e}")
finally:
    s.close()
```

Pour arrêter : `tunnel.stop()`

Les avertissements `DEPRECATION: Unexpected import of 'paramiko.*'` sont inoffensifs (pip qui se plaint d'imports tardifs dans la même session).

Connexion PostGIS depuis PyQGIS

Charger une couche PostGIS

```
from qgis.core import QgsProject, QgsVectorLayer
```

```
uri = (
    "host=localhost port=5433 dbname=alteris_geo "
    "user=alteris_admin password=Alteris2026 sslmode=disable "
    'table="altfoncier_28051"."28051_plu_zonage" (geom) sql='
)
layer = QgsVectorLayer(uri, "PLU Zonage 28051", "postgres")
QgsProject.instance().addMapLayer(layer)
```

Point clé — champ JSONB `props`

Les tables altfoncier ont leurs attributs métier dans un champ `props` de type JSONB. Dans PyQGIS, ce champ est un **dict Python** (pas une chaîne). Pour y accéder en expression QGIS :

```
map_get("props", 'typezone')   ✓ correct
"props" LIKE '%typezone%'      ✗ ne fonctionne pas (props n'est pas une chaîne)
```

Stylisation CNIG PLU (zonage)

Renderer catégorisé sur `map_get("props", 'typezone')` :

typezone	Couleur remplissage	Couleur bordure	Label
U	`#FFFF73`	`#A3A300`	Zones urbaines
AU	`#FFAA00`	`#CD6600`	Zones à urbaniser
A	`#D3FFBE`	`#267300`	Zones agricoles
N	`#98E600`	`#267300`	Zones naturelles et forestières

```
from qgis.core import QgsCategorizedSymbolRenderer, QgsRendererCategory, QgsFillSymbol

cnig = {
    'U':  ('#FFFF73', '#A3A300', 'Zones urbaines (U)'),
    'AU': ('#FFAA00', '#CD6600', 'Zones à urbaniser (AU)'),
    'A':  ('#D3FFBE', '#267300', 'Zones agricoles (A)'),
    'N':  ('#98E600', '#267300', 'Zones naturelles et forestières (N)'),
}
categories = []
for tz, (fill, border, label) in cnig.items():
    symbol = QgsFillSymbol.createSimple({
```

```
        'color': fill, 'outline_color': border, 'outline_width': '0.26'
    })
    categories.append(QgsRendererCategory(tz, symbol, label))

renderer = QgsCategorizedSymbolRenderer("map_get(\"props\", 'typezone')", categories)
layer.setRenderer(renderer)
layer.triggerRepaint()
```

Journal de session

2026-06-20 — Premier diagnostic validé

Première connexion réussie entre Claude Code et QGIS via MCP. Résultat du `diagnose` :

Vérification	Résultat
QGIS	4.0.3-Norrköping
Python	3.12.13
Qt	6.11.0
Plugin MCP	0.5.0 (serveur et plugin synchronisés)
Clients connectés	1
Providers de traitement	3d, gdal, grass, model, native, pdal, project, qgis, quickosm, script
Projet ouvert	Aucun (layer_count = 0)

Statut global : **healthy**. Stack complète et opérationnelle.

2026-06-20 — Connexion PostGIS Alteris + stylisation CNIG

- Connexion PostGIS directe (`79.137.14.202:5432`) → **timeout** (port filtré par pare-feu VPS)
- Tunnel SSH manuel (`ssh -L 5433:localhost:5432 debian@79.137.14.202 -N`) → **connexion OK**
- Couche `altfoncier_28051.28051_plu_zonage` chargée (20 entités, commune 28051)
- Stylisation CNIG appliquée et validée visuellement

- **Point clé découvert** : le champ `props` (JSONB PostGIS) arrive comme **dict Python** dans PyQGIS — utiliser `map_get("props", 'typezone')` en expression, pas LIKE sur chaîne

2026-06-20 — Tunnel paramiko validé

- `sys.executable` = `qgis-bin.exe` → subprocess inutilisable pour pip
- Installation via `pip._internal.cli.main` → paramiko 5.0.0 installé dans le dossier utilisateur
- `sys.path.insert(0, ...)` nécessaire à chaque session (QGIS n'inclut pas le dossier utilisateur)
- Tunnel paramiko démarré → **Tunnel OK** confirmé
- Couche PostGIS chargée via `localhost:5433` → **succès**

Prochaine étape :

- Exporter la carte en PDF via mise en page QGIS

Inkscape MCP

Vue d'ensemble

Inkscape est un éditeur de graphisme vectoriel open source basé sur le format SVG. À la différence de QGIS, **il n'existe pas de plugin MCP officiel pour Inkscape** en juin 2026 — mais plusieurs approches sont réalistes, avec des niveaux d'effort et de puissance différents.

Approches possibles

Approche	Effort	Puissance	Statut
1. Manipulation SVG directe	Aucun	Élevée (structure du fichier)	Fonctionne déjà
2. MCP wrapper CLI Inkscape	Moyen (écrire un serveur MCP)	Moyenne (opérations CLI)	À construire
3. Extension Inkscape + serveur TCP	Élevé (plugin Python Inkscape)	Maximale (GUI + rendu live)	À construire

---

Approche 1 — Manipulation SVG directe (disponible maintenant)

SVG est du XML texte. Claude peut lire, écrire et modifier des fichiers `.svg` directement avec ses outils natifs, sans aucune intégration supplémentaire.

Ce que Claude peut déjà faire

- Lire la structure d'un SVG (éléments, attributs, groupes, calques Inkscape)
- Créer un SVG complet de zéro (formes, textes, chemins, dégradés)
- Modifier des propriétés : couleur, opacité, stroke, transform, viewBox
- Réorganiser les éléments dans l'arbre XML

- Ajouter/supprimer des calques (`<g inkscape:label="..." inkscape:groupmode="layer">`)
- Générer des motifs répétitifs, des grilles, des mises en page structurées

Limites de cette approche

- Pas de rendu live dans Inkscape — il faut rouvrir ou recharger le fichier
- Les opérations complexes (nœuds de chemin, boolean ops) sont difficiles à écrire à la main en SVG
- Pas d'accès aux filtres Inkscape (flou, ombres) via simple édition XML

Flux de travail type

```
Claude génère/modifie le .svg → tu recharges dans Inkscape (Ctrl+Z+rechargement) → ajustements manuels → boucle
```

Approche 2 — MCP wrapper CLI Inkscape

Inkscape expose une interface en ligne de commande (`--actions`) qui permet d'automatiser des opérations sans ouvrir l'interface graphique.

Commandes CLI Inkscape utiles

```
# Export PNG depuis SVG
inkscape --export-type=png --export-filename=sortie.png source.svg

# Export PDF
inkscape --export-type=pdf --export-filename=sortie.pdf source.svg

# Appliquer une transformation et exporter
inkscape --actions="select-all;object-set-attribute:transform,scale(2)" source.svg --export-type=svg --export-filename=sortie.svg

# Convertir texte en chemins
```

```
inkscape --actions="select-all;object-to-path" source.svg --export-type=svg --export-filename=sortie.svg

# Obtenir les dimensions du document
inkscape --query-all source.svg
```

Architecture d'un serveur MCP CLI

```
Claude Code ↔ Serveur MCP Python (stdio) ↔ subprocess inkscape CLI ↔ fichiers SVG/PNG/PDF
```

Un serveur MCP Python minimal exposerait des outils comme :

- `export_png(svg_path, output_path, dpi=96)`
- `export_pdf(svg_path, output_path)`
- `convert_text_to_paths(svg_path, output_path)`
- `get_document_dimensions(svg_path)`
- `apply_inkscape_action(svg_path, actions, output_path)`

Prérequis

```
# Inkscape doit être dans le PATH
inkscape --version
# Inkscape 1.x.x (...)
```

Sur Windows, ajouter le dossier Inkscape au PATH système ou utiliser le chemin complet :

```
C:\Program Files\Inkscape\bin\inkscape.exe
```

Approche 3 — Extension Inkscape + serveur TCP (pattern QGIS)

La plus puissante mais la plus complexe à mettre en place. Inkscape supporte les extensions Python via le module `inkex`. Une extension pourrait ouvrir un socket TCP local et exposer des outils MCP (similaire au plugin QGIS MCP).

Architecture

```
Claude Code ↔ Serveur MCP Python (uvx) ↔ socket TCP local ↔ Extension Python Inkscape
```

Ce que ça permettrait

- Accéder aux éléments sélectionnés dans l'interface
- Appliquer des opérations live (boolean, nœuds, effets)
- Lire la position exacte des objets
- Déclencher des menus et actions Inkscape programmatiquement

Obstacles techniques

- Les extensions Inkscape s'exécutent en mode one-shot (lancées, exécutent, se ferment) — un serveur TCP persistant nécessite un thread daemon ou une extension de type "Effect" avec boucle
- L'API `inkex` est bien documentée mais moins riche que PyQGIS
- Aucun projet open source mature n'existe encore pour ce pattern (juin 2026)

Capacités envisageables selon l'approche

Capacité	Approche 1 (SVG direct)	Approche 2 (CLI)	Approche 3 (Extension)
Créer des formes géométriques	✓	✓	✓
Modifier couleurs / styles	✓	✓	✓
Gérer les calques	✓	—	✓
Exporter PNG/PDF	—	✓	✓
Convertir texte en chemins	—	✓	✓
Boolean operations	—	✓ (CLI)	✓
Lire sélection courante	—	—	✓
Appliquer filtres Inkscape	—	✓ (partiel)	✓
Batch processing de fichiers	✓	✓	—

Capacité	Approche 1 (SVG direct)	Approche 2 (CLI)	Approche 3 (Extension)
Rendu live dans l'UI	—	—	✓

Cas d'usage concrets avec Inkscape

Avec l'approche 1 (maintenant)

- Générer des pictogrammes SVG (icônes, logos simples) à partir d'une description
- Créer des gabarits de mise en page (grilles, marges, zones de texte)
- Modifier en batch les couleurs d'une charte graphique sur plusieurs fichiers SVG
- Produire des cartes thématiques simplifiées à partir de données (ex : export QGIS → SVG → habillage Claude)
- Générer des diagrammes (organigrammes, schémas techniques) en SVG structuré

Avec l'approche 2 (à construire, effort ~2h)

- Pipeline : Claude génère SVG → CLI Inkscape exporte PDF → livraison automatique
- Batch conversion de SVG en PNG à différentes résolutions
- Optimisation SVG (texte en chemins avant impression)

Lien avec le workflow cartographique

Inkscape est souvent utilisé en aval de QGIS pour l'habillage cartographique (polices, mise en page avancée, symboles non gérés par QGIS). Le flux naturel serait :

```
QGIS MCP → export SVG → Claude (habillage Inkscape SVG direct) → Inkscape (retouches manuelles) → export PDF final
```

QGIS peut exporter une mise en page en SVG via `Projet > Imprimer la mise en page > Exporter en SVG`. Claude peut ensuite enrichir ce SVG (légendes, encadrés, pictogrammes) avant ouverture dans Inkscape.

Prochaine étape proposée

Trois options pour explorer concrètement :

1. **Test SVG direct** — Claude génère un SVG d'exemple (carte, diagramme, gabarit) et tu l'ouvres dans Inkscape pour voir le résultat
2. **Prototype CLI MCP** — écrire un serveur MCP Python minimal (~50 lignes) wrappant les exports Inkscape CLI, l'enregistrer dans Claude Code
3. **Exploration API inkex** — ouvrir la console Python d'Inkscape et tester quelques commandes `inkex` pour évaluer la faisabilité de l'approche 3

Mise en place réalisée (2026-07-21)

L'approche 2 (MCP wrapper CLI Inkscape) a été implémentée sur le PC Windows eliob.

Obstacle rencontré : version Microsoft Store (MSIX)

Le PC avait initialement Inkscape installé via le **Microsoft Store**, packagé en MSIX. Ce type de paquet s'est révélé **inutilisable pour un pilotage CLI** :

- L'exécutable dans `C:\Program Files\WindowsApps\...\inkscape.exe` refuse l'exécution directe (`Accès refusé`) — ACL verrouillées par le sandboxing du paquet.
- Aucun alias d'exécution (`AppExecutionAlias`) n'est déclaré dans le manifest (`AppxManifest.xml`) — impossible d'appeler `inkscape` depuis le PATH.
- `Invoke-CommandInDesktopPackage` (la seule méthode de lancement programmatique disponible) ouvre l'app en mode GUI sans retourner la main ni capturer stdout — inexploitable pour de l'automatisation headless (export, `--actions`, etc.).

Conclusion : les versions Inkscape installées via le Microsoft Store ne conviennent pas à un usage MCP/CLI. Il faut la version standalone.

Fix — installation de la version standalone

```
winget install --id Inkscape.Inkscape --source winget --accept-package-agreements --accept-source-agreements
```

- Package winget `Inkscape.Inkscape` (distinct du Store) → installe dans `C:\Program Files\Inkscape\bin\inkscape.exe`
- Version installée : **1.4.4** (dcaf3e7, 2026-05-05)
- Vérification : `& "C:\Program Files\Inkscape\bin\inkscape.exe" --version` répond correctement, exit code 0

Important : si une version Store est déjà présente, la désinstaller d'abord (Paramètres > Applications, ou `winget uninstall`) pour éviter la confusion entre les deux installations.

Serveur MCP créé

- **Fichier** : `C:\Users\eliob\.claude\mcp_inkscape.py`
- **Framework** : `FastMCP` (même pattern que les autres MCP du projet — `mcp_portainer.py`, `mcp_jellyfin.py`, etc.)
- **Constante** : `INKSCAPE_EXE = r"C:\Program Files\Inkscape\bin\inkscape.exe"` (chemin en dur — adapter si l'installation standalone se fait ailleurs)

Outils exposés :

Outil	Rôle
`get_version()`	Vérifie qu'Inkscape répond (smoke test)
`get_document_dimensions(svg_path)`	Dimensions et bounding boxes de tous les objets (`--query-all`)
`export_png(svg_path, output_path="", dpi=96)`	Export PNG
`export_pdf(svg_path, output_path="")`	Export PDF
`convert_text_to_paths(svg_path, output_path="")`	Convertit les textes en chemins vectoriels
`apply_inkscape_action(svg_path, actions, output_path="")`	Échappatoire générique — chaîne d'actions Inkscape libre (`;` séparées)

Toutes les fonctions retournent `{ok, returncode, stdout, stderr}` (+ `output_path` pour les exports) — pas d'exception levée sur échec CLI, le code retour renseigne l'appelant.

Enregistrement MCP

Ajouté dans `C:\Users\eliob\.mcp.json` :

```
"inkscape": {
  "type": "stdio",
  "command": "python",
  "args": ["C:\\Users\\eliob\\.claude\\mcp_inkscape.py"],
  "env": {}
}
```

Nécessite un rechargement de session Claude Code pour que le nouveau serveur MCP soit détecté et ses outils exposés.

Test réalisé

SVG de test (rectangle bleu + texte) → `export_png` → PNG de 1974 octets généré avec succès. `get_document_dimensions` retourne correctement les bounding box (`svg1,10,10,180,80` etc.) via `--query-all`.

À reproduire demain sur le PC Windows Alteris

1. Vérifier si Inkscape est installé via le Store (`Get-AppxPackage -Name "*Inkscape*"`) — si oui, désinstaller
2. `winget install --id Inkscape.Inkscape --source winget --accept-package-agreements --accept-source-agreements`
3. Vérifier `C:\Program Files\Inkscape\bin\inkscape.exe --version`
4. Copier `mcp_inkscape.py` (script identique, chemin `INKSCAPE_EXE` à adapter si besoin)
5. Ajouter l'entrée `inkscape` dans le `.mcp.json` de la machine Alteris
6. Recharger la session Claude Code et tester `get_version()` + `export_png()` sur un SVG de test

---

Confirmation de fonctionnement (2026-07-21, session rechargée)

Après rechargement de la session Claude Code, le serveur MCP `inkscape` est bien détecté et ses 6 outils sont exposés. Tests effectués dans l'ordre :

Test	Résultat
`get_version()`	OK — `Inkscape 1.4.4 (dcaf3e7, 2026-05-05)`
`get_document_dimensions(svg_path)` sur un SVG de test (rect 80x80)	OK — `svg1,10,10,80,80` / `rect1,10,10,80,80`
`export_png(svg_path)`	OK — PNG généré (389 octets), fichier vérifié sur disque

Conclusion : le protocole MCP Inkscape est pleinement opérationnel sur le PC Windows eliob, de bout en bout (Claude Code → serveur MCP Python → subprocess Inkscape CLI → fichier de sortie).

LibreOffice MCP

quelles sont les possibilités MCP permises pour la suite logicielle LibreOffice pour interagir avec une IA - rédiger ou mettre à jour des paragraphes, controler le sens du texte,créer des tableurs, des présentations ?

En juin 2026, les possibilités offertes par le **Model Context Protocol (MCP)** pour interagir avec la suite LibreOffice (Writer, Calc, Impress, etc.) via une IA sont déjà bien avancées et permettent une intégration poussée. Voici ce que tu peux faire concrètement :

---

1. Édition et création de documents (Writer)

- **Rédiger ou mettre à jour des paragraphes** : Les serveurs MCP pour LibreOffice permettent à une IA de **lire, écrire, modifier et formater** des documents Writer (ODT). Tu peux demander à l'IA d'ajouter des titres, des paragraphes, des tableaux, des images, ou d'effectuer des recherches/remplacements automatiques dans le texte. Par exemple, un outil comme libreoffice-mcp ou Nelson MCP expose ces fonctionnalités via des appels d'outils standardisés.
- **Contrôler le sens du texte** : L'IA peut analyser le contenu, suggérer des reformulations, vérifier la cohérence ou la grammaire, et même générer du texte contextuel en fonction du document ouvert.

---

2. Gestion de tableurs (Calc)

- **Créer et manipuler des tableurs** : Les serveurs MCP permettent de **créer, lire, modifier des feuilles de calcul** (ODS), y compris la manipulation de cellules, l'insertion de formules, la création de graphiques, et l'automatisation de tâches répétitives (ex : mise à jour de données, tri, filtrage).
- **Appeler une IA directement depuis une cellule** : Certains projets, comme Nelson MCP, intègrent une fonction `=PROMPT()` dans Calc, qui permet d'interroger une IA (ex : Claude, Gemini) directement depuis une formule Excel, pour générer des données ou des analyses en temps réel.

---

3. Présentations (Impress)

- **Créer ou modifier des diapositives** : Les outils MCP permettent d'automatiser la création de présentations, d'ajouter du texte, des images, des formes, ou de restructurer des diapositives existantes. L'IA peut aussi suggérer des mises en page ou du contenu en fonction d'un brief.

---

4. Automatisation et workflows

- **Automatiser des workflows** : Tu peux configurer des **presets** (ex : minimal, writer-edit, calc) pour exposer uniquement les outils nécessaires à ton agent IA. Par exemple, lancer un script qui génère un rapport dans Writer, extrait des données de Calc, et les insère dans une présentation Impress, le tout piloté par une IA.
- **Intégration avec des clients MCP** : Les serveurs MCP pour LibreOffice sont compatibles avec des clients comme **Claude Desktop, Cursor, ou des outils personnalisés**. Ils fonctionnent en local (via une extension LibreOffice ou un serveur HTTP) et permettent une interaction fluide entre l'IA et tes documents.

---

5. Exemples concrets d'outils

- **libreoffice-mcp** : Extension et serveur MCP pour LibreOffice, permettant une intégration complète avec des IA comme Claude ou Gemini. Fonctionne avec LibreOffice 24.2+ et expose une API HTTP pour les appels d'outils.
- **Nelson MCP** : Extension qui transforme tes documents LibreOffice en serveur MCP, avec des launchers intégrés pour Claude Code, Gemini CLI, etc. Permet de lire/éditer des documents ouverts et d'utiliser des fonctions avancées comme `=PROMPT()` dans Calc.
- **OooDev** : Bibliothèque Python qui simplifie l'accès à l'API UNO de LibreOffice, utilisée par plusieurs serveurs MCP pour manipuler des documents, des bases de données (Base), ou des graphiques.

---

6. Prérequis techniques

- LibreOffice **24.2 ou supérieur** (pour la plupart des serveurs MCP).
- Installation d'une **extension LibreOffice** (ex : `libreoffice-mcp-extension.oxt`) ou d'un serveur MCP local (Python/Node.js).
- Connexion via **socket** (port 2083 ou 8765 selon les outils) ou HTTP pour les appels d'API.

---

7. Cas d'usage avancés

- **Veille territoriale** : Tu peux combiner MCP avec des alertes (ex : Google Alerts) et une IA pour **filtrer, classer et résumer** des informations dans des documents LibreOffice, comme tu l'envisageais pour ton projet de gestion de veilles.
- **Gestion de bibliothèque** : Automatiser la création de fiches ou de rapports à partir de tes données (ex : Kavita) en utilisant des templates Writer ou Calc.

Comment démarrer ?

1. **Installe un serveur MCP** : Par exemple, Nelson MCP ou libreoffice-mcp.
2. **Configure ton client IA** (Claude, Cursor, etc.) pour qu'il se connecte au serveur MCP.
3. **Ouvre un document LibreOffice** et demande à ton IA de l'éditer ou de créer du contenu.

Modalités techniques de mise en place

Option retenue : Nelson MCP

Nelson MCP est l'option la plus directement compatible avec Claude Code — il expose un serveur MCP depuis LibreOffice via une extension `.oxt`, avec des launchers natifs pour Claude Code et Gemini CLI.

---

1. Installation de l'extension LibreOffice

- Télécharger l'extension depuis la page lobehub.com/mcp/quazardous-nelson-mcp (fichier `.oxt`)
- Dans LibreOffice : **Outils → Gestionnaire des extensions → Ajouter** → sélectionner le `.oxt`
- Redémarrer LibreOffice

---

2. Lancement du serveur MCP

Depuis LibreOffice via le menu dédié (Outils → Nelson MCP → Démarrer le serveur), ou en ligne de commande :

```
python3 -m nelson_mcp --port 8765
```

Le serveur écoute sur `localhost:8765`.

---

3. Configuration dans Claude Code

Ajouter dans `C:\Users\eliob\.claude.json` (section `mcpServers` du projet concerné) :

```
"libreoffice": {
  "type": "http",
  "url": "http://localhost:8765/mcp"
}
```

Si le serveur utilise stdio plutôt que HTTP :

```
"libreoffice": {
  "type": "stdio",
  "command": "python3",
  "args": ["-m", "nelson_mcp"]
}
```

---

4. Points à vérifier lors de la mise en place

- Version LibreOffice : `libreoffice --version` → doit être ≥ 24.2
- Port 8765 libre : `netstat -ano | findstr 8765` (Windows)
- Un document LibreOffice doit être **ouvert avant** de lancer une requête MCP (le serveur nécessite un document actif)
- Tester avec une lecture simple du document ouvert pour valider la connexion

---

5. Alternative : libreoffice-mcp (KeithCu)

Si Nelson MCP pose problème, `libreoffice-mcp` (github.com/KeithCu/writeragent) expose une API HTTP via une extension `.oxt` sur le port 2083 :

```
"libreoffice": {
  "type": "http",
  "url": "http://localhost:2083"
}
```