# 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).

---

Revision #4
Created 2026-06-21 18:19:17 UTC by Julien
Updated 2026-07-21 14:53:27 UTC by Julien