07

Desktop Extensions: Claude Desktop 向けの 1 クリック MCP サーバーインストール

— Desktop Extensions: One-click MCP server installation for Claude Desktop —

公開日	2025-06-26
原題	Desktop Extensions: One-click MCP server installation for Claude Desktop
著者	Anthropic Engineering Team
原文	https://www.anthropic.com/engineering/desktop-extensions
翻訳	Claude（機械翻訳／Anthropic）
編集	2026-04-09

Desktop Extensions: Claude Desktop 向けの 1 クリック MCP サーバーインストール

- ファイル拡張子の更新

2025 年 9 月 11 日

Claude Desktop Extensions は `.dxt` の代わりに `.mcpb`（MCP Bundle）ファイル拡張子を使うようになりました。既存の `.dxt` 拡張機能は引き続き動作しますが、今後新規作成される拡張機能では `.mcpb` の使用を推奨します。機能は同じで、これは純粋に命名規則の更新です。

昨年 Model Context Protocol（MCP）をリリースした際、私たちは開発者がファイルシステムからデータベースに至るあらゆるものに Claude からアクセスできる素晴らしいローカルサーバーを作っていくのを目の当たりにしました。しかし、同じフィードバックを何度も聞くことになりました——インストールが複雑すぎる、と。ユーザーは開発者ツールが必要で、手動で設定ファイルを編集しなければならず、依存関係で詰まることも多かったのです。

本日、MCP サーバーのインストールをボタンをクリックするだけにする新しいパッケージフォーマット、**Desktop Extensions** を紹介します。

MCP のインストール問題への対処

ローカル MCP サーバーは Claude Desktop ユーザーに強力な能力を解き放ちます。ローカルアプリケーションと対話し、プライベートデータにアクセスし、開発ツールと連携でき、しかもデータはユーザーのマシンに留まります。しかし、現在のインストールプロセスには大きな障壁があります。

- **開発者ツールが必要**: ユーザーは Node.js、Python、その他のランタイムをインストールする必要がある
- **手動設定**: 各サーバーごとに JSON 設定ファイルの編集が必要
- **依存関係管理**: パッケージ衝突やバージョン不一致を解決しなければならない
- **発見の仕組みがない**: 有用な MCP サーバーを見つけるには GitHub を探し回る必要がある
- **更新の複雑さ**: 最新に保つには手動で再インストール

こうした摩擦のせいで、MCP サーバーはその力にもかかわらず、非技術者にはほぼ手の届かないものになっていました。

Desktop Extensions の紹介

Desktop Extensions（`.mcpb` ファイル）は、MCP サーバー全体——すべての依存関係を含めて——を 1 つのインストール可能パッケージにバンドルすることで、これらの問題を解決します。ユーザーにとっての変化はこうです。

Before:

```
# まず Node.js をインストール
npm install -g @example/mcp-server
# ~/.claude/claude_desktop_config.json を手動編集
# Claude Desktop を再起動
# 動くことを祈る
```

After:

1. `.mcpb` ファイルをダウンロード
2. ダブルクリックして Claude Desktop で開く
3. 「Install」をクリック

以上です。ターミナルなし、設定ファイルなし、依存衝突なし。

アーキテクチャ概要

Desktop Extension は、ローカル MCP サーバーと `manifest.json` を含む ZIP アーカイブです。`manifest.json` には、Claude Desktop や Desktop Extensions をサポートする他のアプリが必要とするすべての情報が記述されます。

```
extension.mcpb (ZIP archive)
├── manifest.json          # 拡張機能のメタデータと設定
├── server/                # MCP サーバー実装
│   └── [server files]
├── dependencies/          # すべての必要なパッケージ/ライブラリ
└── icon.png              # オプション: 拡張機能アイコン

# 例: Node.js 拡張機能
extension.mcpb
├── manifest.json          # 必須: 拡張機能のメタデータと設定
├── server/                # サーバーファイル
│   └── index.js           # メインエントリポイント
├── node_modules/          # バンドルされた依存関係
├── package.json           # オプション: NPM パッケージ定義
└── icon.png               # オプション: 拡張機能アイコン

# 例: Python 拡張機能
extension.mcpb (ZIP file)
├── manifest.json          # 必須: 拡張機能のメタデータと設定
├── server/                # サーバーファイル
│   ├── main.py            # メインエントリポイント
│   └── utils.py           # 追加モジュール
├── lib/                   # バンドルされた Python パッケージ
├── requirements.txt       # オプション: Python 依存関係リスト
└── icon.png               # オプション: 拡張機能アイコン
```

Desktop Extension で唯一必須のファイルは `manifest.json` です。Claude Desktop が複雑さをすべて引き受けます。

- **組み込みランタイム**: Claude Desktop に Node.js を同梱し、外部依存をなくす
- **自動更新**: 新バージョンが利用可能になったら拡張機能は自動更新
- **安全なシークレット**: API キーのようなセンシティブな設定は OS のキーチェーンに保存

マニフェストには、人間が読める情報（名前、説明、著者など）、機能の宣言（ツール、プロンプト）、ユーザー設定、ランタイム要件が含まれます。ほとんどのフィールドはオプションなので、最小バージョンは非常に短くなります。ただし、実運用では、サポートされる 3 タイプの拡張機能（Node.js、Python、クラシックバイナリ／実行可能ファイル）すべてが何らかのファイルを含むことを想定しています。

```
{
  "mcpb_version": "0.1",                    // このマニフェストが準拠する MCPB 仕様のバージョン
  "name": "my-extension",                   // マシン可読名（CLI や API で使用）
  "version": "1.0.0",                       // 拡張機能のセマンティックバージョン
  "description": "A simple MCP extension",  // 拡張機能の簡単な説明
  "author": {                               // 著者情報（必須）
    "name": "Extension Author"              // 著者名（必須フィールド）
  },
  "server": {                               // サーバー設定（必須）
    "type": "node",                         // サーバータイプ: "node"、"python"、"binary"
    "entry_point": "server/index.js",       // メインサーバーファイルのパス
    "mcp_config": {                         // MCP サーバー設定
      "command": "node",                    // サーバーを実行するコマンド
      "args": [                             // コマンドに渡す引数
        "${__dirname}/server/index.js"      // ${__dirname} は拡張機能のディレクトリに置換される
      ]
    }
  }
}
```

マニフェスト仕様 には、ローカル MCP サーバーのインストールと設定を簡単にするための便利なオプションが多数あります。サーバー設定オブジェクトは、テンプレートリテラル形式のユーザー定義設定と、プラットフォーム固有のオーバーライドの両方を受け入れるように定義できます。拡張機能の開発者は、ユーザーからどのような設定を収集したいかを詳細に定義できます。

マニフェストが設定をどのように助けるか、具体例を見てみましょう。以下のマニフェストでは、開発者がユーザーに `api_key` の供給を要求すると宣言しています。Claude は、ユーザーがその値を供給するまで拡張機能を有効化せず、自動的に OS のシークレットボールトに保管し、サーバー起動時に `${user_config.api_key}` をユーザーの値で透過的に置き換えます。同様に、`${__dirname}` は展開されたディレクトリのフルパスに置換されます。

```
{
  "mcpb_version": "0.1",
  "name": "my-extension",
  "version": "1.0.0",
  "description": "A simple MCP extension",
  "author": {
    "name": "Extension Author"
  },
  "server": {
    "type": "node",
    "entry_point": "server/index.js",
    "mcp_config": {
      "command": "node",
      "args": ["${__dirname}/server/index.js"],
      "env": {
        "API_KEY": "${user_config.api_key}"
      }
    }
  },
  "user_config": {
    "api_key": {
      "type": "string",
      "title": "API Key",
      "description": "Your API key for authentication",
      "sensitive": true,
      "required": true
    }
  }
}
```

オプションフィールドの多くを含んだ完全な `manifest.json` はこんな感じになります。

```
{
  "mcpb_version": "0.1",
  "name": "My MCP Extension",
  "display_name": "My Awesome MCP Extension",
  "version": "1.0.0",
  "description": "A brief description of what this extension does",
  "long_description": "A detailed description that can include multiple paragraphs e
xplaining the extension's functionality, use cases, and features. It supports basic
markdown.",
  "author": {
    "name": "Your Name",
    "email": "yourname@example.com",
    "url": "https://your-website.com"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/your-username/my-mcp-extension"
  },
  "homepage": "https://example.com/my-extension",
  "documentation": "https://docs.example.com/my-extension",
  "support": "https://github.com/your-username/my-extension/issues",
  "icon": "icon.png",
  "screenshots": [
    "assets/screenshots/screenshot1.png",
    "assets/screenshots/screenshot2.png"
  ],
  "server": {
    "type": "node",
    "entry_point": "server/index.js",
    "mcp_config": {
      "command": "node",
      "args": ["${__dirname}/server/index.js"],
      "env": {
        "ALLOWED_DIRECTORIES": "${user_config.allowed_directories}"
      }
    }
  },
  "tools": [
    {
      "name": "search_files",
      "description": "Search for files in a directory"
    }
  ],
  "prompts": [
    {
      "name": "poetry",
      "description": "Have the LLM write poetry",
      "arguments": ["topic"],
      "text": "Write a creative poem about the following topic: ${arguments.topic}"
    }
```

```
  ],
  "tools_generated": true,
  "keywords": ["api", "automation", "productivity"],
  "license": "MIT",
  "compatibility": {
    "claude_desktop": ">=1.0.0",
    "platforms": ["darwin", "win32", "linux"],
    "runtimes": {
      "node": ">=16.0.0"
    }
  },
  "user_config": {
    "allowed_directories": {
      "type": "directory",
      "title": "Allowed Directories",
      "description": "Directories the server can access",
      "multiple": true,
      "required": true,
      "default": ["${HOME}/Desktop"]
    },
    "api_key": {
      "type": "string",
      "title": "API Key",
      "description": "Your API key for authentication",
      "sensitive": true,
      "required": false
    },
    "max_file_size": {
      "type": "number",
      "title": "Maximum File Size (MB)",
      "description": "Maximum file size to process",
      "default": 10,
      "min": 1,
      "max": 100
    }
  }
}
```

拡張機能とマニフェストの実例は、MCPB リポジトリの examples を参照してください。

`manifest.json` の全必須／オプションフィールドの完全仕様は、オープンソースのツールチェーン にあります。

最初の拡張機能を作る

既存の MCP サーバーを Desktop Extension としてパッケージ化する流れを追ってみましょう。例としてシンプルなファイルシステムサーバーを使います。

ステップ 1: マニフェストを作る

まず、サーバー用のマニフェストを初期化します。

```
npx @anthropic-ai/mcpb init
```

この対話型ツールがサーバーについて質問し、完全な manifest.json を生成します。最小限の manifest.json に一気に進みたいなら、`--yes` パラメータ付きで実行できます。

ステップ 2: ユーザー設定を扱う

サーバーがユーザー入力（API キーや許可ディレクトリなど）を必要とするなら、マニフェストで宣言します。

```
"user_config": {
  "allowed_directories": {
    "type": "directory",
    "title": "Allowed Directories",
    "description": "Directories the server can access",
    "multiple": true,
    "required": true,
    "default": ["${HOME}/Documents"]
  }
}
```

Claude Desktop は以下を行います。

- ユーザーフレンドリーな設定 UI を表示
- 拡張機能を有効化する前に入力をバリデーション
- センシティブな値を安全に保存
- 設定を引数または環境変数としてサーバーに渡す（開発者の設定に応じて）

以下の例ではユーザー設定を環境変数として渡していますが、引数として渡すこともできます。

```
"server": {
   "type": "node",
   "entry_point": "server/index.js",
   "mcp_config": {
   "command": "node",
   "args": ["${__dirname}/server/index.js"],
   "env": {
      "ALLOWED_DIRECTORIES": "${user_config.allowed_directories}"
   }
   }
}
```

ステップ 3: 拡張機能をパッケージする

すべてを `.mcpb` ファイルにバンドルします。

```
npx @anthropic-ai/mcpb pack
```

このコマンドは以下を行います。

1. マニフェストをバリデート
2. `.mcpb` アーカイブを生成

ステップ 4: ローカルでテスト

`.mcpb` ファイルを Claude Desktop の設定ウィンドウにドラッグ&ドロップします。以下が表示されます。

- 拡張機能についての人間可読な情報
- 必要な権限と設定
- シンプルな「Install」ボタン

高度な機能

クロスプラットフォームサポート

拡張機能は異なる OS に適応できます。

```
"server": {
  "type": "node",
  "entry_point": "server/index.js",
  "mcp_config": {
    "command": "node",
    "args": ["${__dirname}/server/index.js"],
    "platforms": {
      "win32": {
        "command": "node.exe",
        "env": {
          "TEMP_DIR": "${TEMP}"
        }
      },
      "darwin": {
        "env": {
          "TEMP_DIR": "${TMPDIR}"
        }
      }
    }
  }
}
```

動的設定

実行時値にはテンプレートリテラルを使います。

- `${__dirname}`: 拡張機能のインストールディレクトリ
- `${user_config.key}`: ユーザーが提供した設定
- `${HOME}`, `${TEMP}`: システム環境変数

機能の宣言

ユーザーに能力を事前に理解させましょう。

```
"tools": [
  {
    "name": "read_file",
    "description": "Read contents of a file"
  }
],
"prompts": [
  {
    "name": "code_review",
    "description": "Review code for best practices",
    "arguments": ["file_path"]
  }
]
```

拡張機能ディレクトリ

Claude Desktop にキュレート済みの拡張機能ディレクトリを組み込んだ状態でローンチします。ユーザーはブラウズ、検索、1 クリックでインストールできます——GitHub を探し回ったり、コードを精査したりする必要はありません。

Desktop Extension の仕様と macOS / Windows 版 Claude での実装の両方が今後進化していくと期待しつつ、拡張機能が Claude の能力を創造的に広げるさまざまな活用法を見られるのを楽しみにしています。

拡張機能を提出するには:

1. 提出フォームにあるガイドラインに従っていることを確認
2. Windows と macOS の両方でテスト
3. 拡張機能を提出
4. 私たちのチームが品質とセキュリティをレビュー

オープンなエコシステムを築く

私たちは MCP サーバーを取り巻くオープンエコシステムにコミットしており、複数のアプリやサービスで普遍的に採用される能力がコミュニティに利益をもたらしてきたと信じています。このコミットメントに沿って、Desktop Extension の仕様、ツールチェーン、そして macOS / Windows 版 Claude が Desktop Extensions のサポートを実装するのに使うスキーマと主要機能をオープンソース化します。MCPB フォーマットが Claude だけでなく、他の AI デスクトップアプリケーションにとってもローカル MCP サーバーをよりポータブルなものにすることを期待しています。

オープンソース化するもの:

- 完全な MCPB 仕様
- パッケージングとバリデーションのツール
- リファレンス実装コード
- TypeScript の型とスキーマ

これが意味するのは:

- **MCP サーバー開発者**: 一度パッケージ化すれば、MCPB 対応のあらゆる場所で動く
- **アプリ開発者**: ゼロから作らずに拡張機能サポートを追加できる
- **ユーザー**: MCP 対応の全アプリで一貫した体験

仕様とツールチェーンは意図的に 0.1 とバージョニングしています。コミュニティと一緒にフォーマットを進化・変更していくことを楽しみにしています。皆さんの声をお聞かせください。

セキュリティとエンタープライズの考慮

拡張機能が新しいセキュリティ上の考慮——特に企業にとって——を生むことを理解しています。Desktop Extensions のプレビューリリースにはいくつかの安全策を組み込みました。

ユーザー向け

- センシティブなデータは OS のキーチェーンに留まる
- 自動更新
- インストール済み拡張機能の監査が可能

エンタープライズ向け

- Windows のグループポリシーと macOS の MDM をサポート
- 承認済み拡張機能のプリインストールが可能
- 特定の拡張機能や公開者をブロックリスト化
- 拡張機能ディレクトリを完全に無効化
- プライベートな拡張機能ディレクトリの展開

組織内で拡張機能を管理する方法の詳細は、ドキュメント を参照してください。

始め方

自分の拡張機能を作る準備はできましたか？ ここから始めましょう。

MCP サーバー開発者向け: 開発者ドキュメント を確認するか、ローカル MCP サーバーのディレクトリで以下のコマンドを実行:

```
npm install -g @anthropic-ai/mcpb
mcpb init
mcpb pack
```

Claude Desktop ユーザー向け: 最新版にアップデートし、設定の Extensions セクションを探してみてください。

エンタープライズ向け: デプロイオプションについてはエンタープライズドキュメントを確認してください。

Claude Code で作る

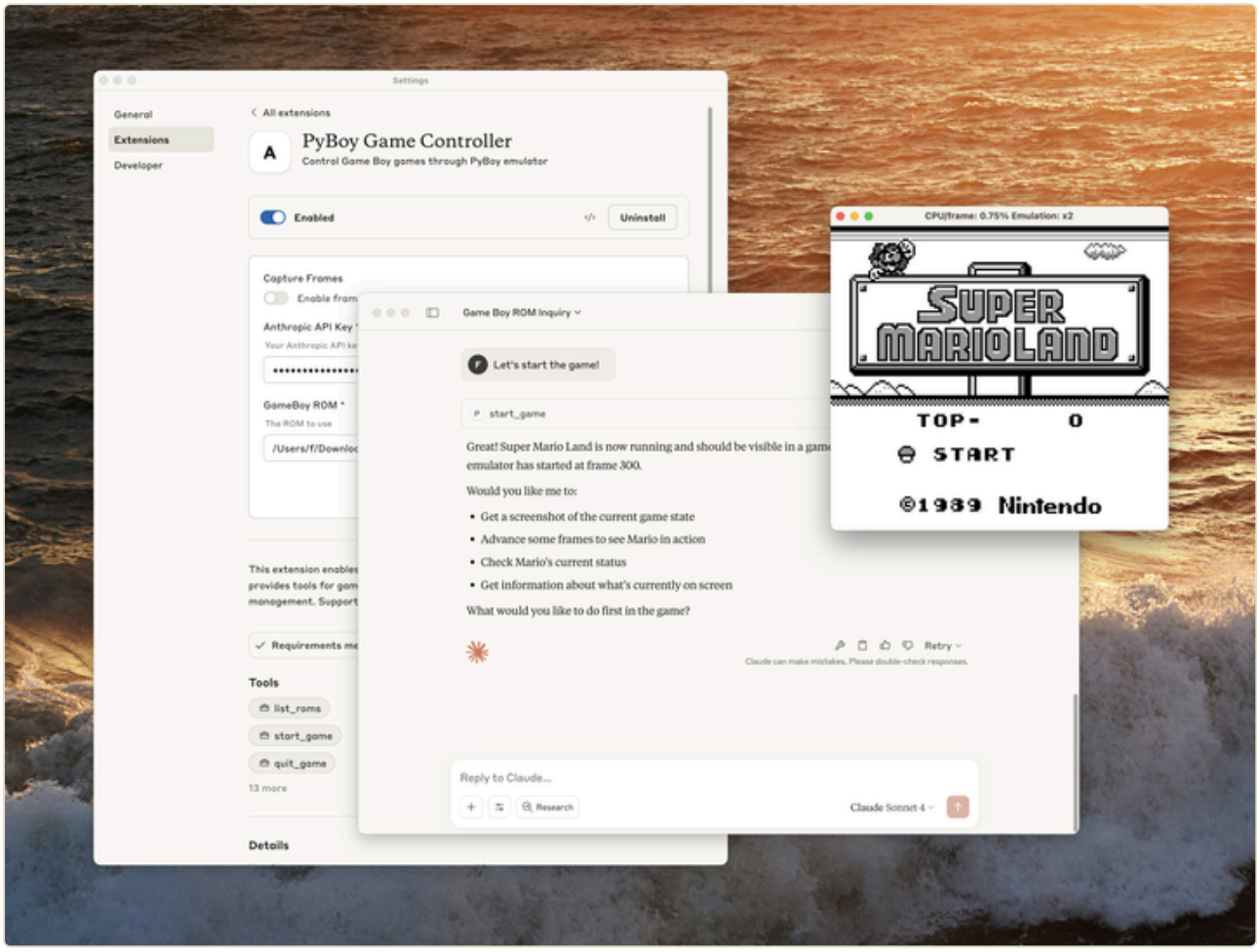
Anthropic 社内では、Claude が最小限の介入で拡張機能を上手に作ることを発見しました。Claude Code を使いたい場合は、まず拡張機能で何をしたいかを簡単に説明し、以下の文脈をプロンプトに追加することを勧めます。

```
I want to build this as a Desktop Extension, abbreviated as "MCPB". Please follow th
ese steps:

1. **Read the specifications thoroughly:**
   - https://github.com/anthropics/mcpb/blob/main/README.md - MCPB architecture over
view, capabilities, and integration patterns
   - https://github.com/anthropics/mcpb/blob/main/MANIFEST.md - Complete extension m
anifest structure and field definitions
   - https://github.com/anthropics/mcpb/tree/main/examples - Reference implementatio
ns including a "Hello World" example

2. **Create a proper extension structure:**
   - Generate a valid manifest.json following the MANIFEST.md spec
   - Implement an MCP server using @modelcontextprotocol/sdk with proper tool defini
tions
   - Include proper error handling and timeout management

3. **Follow best development practices:**
   - Implement proper MCP protocol communication via stdio transport
   - Structure tools with clear schemas, validation, and consistent JSON responses
   - Make use of the fact that this extension will be running locally
   - Add appropriate logging and debugging capabilities
   - Include proper documentation and setup instructions

4. **Test considerations:**
   - Validate that all tool calls return properly structured responses
   - Verify manifest loads correctly and host integration works

Generate complete, production-ready code that can be immediately tested. Focus on de
fensive programming, clear error messages, and following the exact
MCPB specifications to ensure compatibility with the ecosystem.
```

結論

Desktop Extensions は、ユーザーがローカルの AI ツールとどう対話するかに根本的な変化をもたらします。インストールの摩擦を取り除くことで、強力な MCP サーバーを誰もが——開発者だけでなく——使えるようにしています。

社内では、非常に実験的な MCP サーバーを共有するために Desktop Extensions を使っています——一部は楽しげに、一部は有用に。あるチームは、「Claude plays Pokémon」の研究 のように、モデルを GameBoy に直接接続したらどこまで行けるか試していました。人気の PyBoy GameBoy エミュレータを起動して Claude に操作させる拡張機能 1 つをパッケージ化するのに Desktop Extensions を使いました。モデルの能力を、ユーザーがローカルマシンで既に持っているツール・データ・アプリケーションに接続する機会は無数にあると私たちは信じています。



皆さんが何を作るのか、待ちきれません。数千の MCP サーバーをもたらしたのと同じ創造性が、今では 1 クリックで数百万のユーザーに届きます。自分の MCP サーバーを共有する準備はできましたか？ 拡張機能をレビューに提出 してください。