ChatGPT

GWexpyドキュメント再構成提案

以下では、GWexpyドキュメントの情報アーキテクチャ再構成と移行戦略について提案します。現状の問題点（単純なトップページ構成、多言語ディレクトリ構造の非対称性、`.rst`/`.md`混在など）を踏まえ、他パッケージの事例を参考に最適な構成を提示し、移行時のURL維持戦略やフォーマット統一策、さらに自動化ツール（Codex）への具体的な指示内容について述べます。

1. 他パッケージに見る望ましいドキュメント構成

他の拡張系Pythonパッケージ（例：Seaborn, GeoPandas, TorchMetrics, scikit-imageなど）のドキュメント構成には共通したパターンがあります。それらを参考に、GWexpyでも次のような情報分類とページ構成を採用することを提案します：

- **イントロダクション/概要**（Introduction/Overview）: パッケージの概要や目的を説明し、インストール方法・簡単な使用例を示す「はじめに」のセクションです（Seabornでは「Installing」「Introduction」、GeoPandasでは「Getting started」に相当）。
- **チュートリアル/ユーザーガイド**（Tutorials/User Guide）: 基本的な使い方から応用まで段階的に学べるガイド集です。SeabornやTorchMetricsでは「Tutorial」や「User Guide」として様々なトピック別ページを提供しています [1]。GeoPandasも「User Guide」で基本機能を解説し、さらに「Advanced Guide」で上級トピックを分けています [2]。ここにはGWexpyにおける**チュートリアルノートブック**（Jupyter Notebookの例）も含めると良いでしょう。例えば「Getting Started」「時系列解析の基礎」「高度な解析チュートリアル」等のページ群を配置します。
- **レシピ集/How-To**（How-to guides）: 具体的な課題解決手順を示す短いハウツー集です（必要に応じて）。Divio方式のドキュメント分類ではチュートリアルと別にHow-Toガイドを設けますが、規模によってはユーザーガイドに統合しても構いません。
- **APIリファレンス**（API Reference）: 関数・クラス等の詳細な自動生成リファレンスです。他パッケージ同様、パブリックAPIを網羅して項目別に整理します。GeoPandasでは「API reference」で全クラス・関数を詳細に列挙しています [3]。GWexpyでも`gwexpy`名前空間以下の型や関数をモジュール別に整理する予定です。
- **実例集/ギャラリー**（Examples/Gallery）: 代表的な使用例やプロット集を画像付きで示すギャラリーページです。scikit-imageやSeabornでは「Examples」「Gallery」として豊富なサンプルが掲載されています [4]。GWexpyでも、既存のJupyter Notebookチュートリアルから静的なサンプル図やコード例を抽出し、ギャラリー形式で閲覧できるページを設けると学習効果が高まります。
- **FAQ/トラブルシューティング**: ユーザーがハマりがちなポイントや質問をまとめたFAQや問題解決ガイドです（必要に応じて追加）。
- **リリースノート/変更履歴**（Release Notes/Changelog）: バージョンごとの新機能や変更点を記載します。他パッケージでも「Releases」「Changelog」等のページで変更履歴を管理しています [5] [6]。
- **引用情報**（Citing）や**貢献方法**（Contributing/Development）: 研究用途での引用方法や、開発者向けの貢献ガイドラインを含めます（Seabornでは「Citing」 [7]、scikit-imageでは「Development」「About」 [4] に相当）。GWexpyでは開発者向け文書は`docs/developers/`以下に分離する計画があるため、ユーザー向けサイトには最低限の「Contributing」リンクだけ載せ、詳細はGitHubリポジトリの貢献ガイドに誘導すると良いでしょう。

以上を踏まえ、推奨される新しい構成の一例をディレクトリツリー形式で示します（英語版を基準とし、日本語版は対訳となる構成です）：

```
docs/
└── web/                    # ユーザー向け公開ドキュメント
    ├── en/                 # 英語版ドキュメント
    │   ├── index.rst (.md)        # トップページ（概要・目次）
    │   ├── introduction.rst       # 導入・インストールガイド
    │   ├── user_guide/            # ユーザーガイド（基本〜応用）
    │   │   ├── quickstart.md      # クイックスタートチュートリアル
    │   │   ├── data_handling.md   # データ読み込みなどトピック別ガイド
    │   │   ├── ...（他のガイド）...
    │   │   └── tutorials/         # Jupyterチュートリアルノートブック集
    │   │       ├── tutorial1.ipynb (or .md)  # 例: ARIMA解析チュートリアル
    │   │       └── ...
    │   ├── examples/              # 実例ギャラリー
    │   │   ├── example1.rst (.md)  # 例: 典型的なプロットの例
    │   │   └── ...
    │   ├── reference/             # APIリファレンス
    │   │   ├── index.rst          # リファレンス目次（モジュール一覧）
    │   │   ├── module1.rst        # 例: gwexpy.core モジュール
    │   │   └── ...
    │   ├── faq.rst                # FAQ・トラブルシューティング
    │   ├── changelog.rst          # リリースノート
    │   └── contribute.rst         # 貢献方法（必要に応じて）
    └── ja/                 # 日本語版ドキュメント（構成は英語版と対称）
        ├── index.rst
        ├── introduction.rst
        ├── user_guide/
        │   ├── quickstart.md
        │   ├── ...
        │   └── tutorials/
        ├── examples/
        ├── reference/
        │   └── index.rst 等
        ├── faq.rst
        ├── changelog.rst
        └── contribute.rst
```

上記のように**言語ごとに対称なディレクトリ構成**を取り、各言語内で「概要（Index）→ガイド/チュートリアル→リファレンス→付録」という流れが分かる構造にします [8] [9]。これにより利用者は目的の情報（使い方解説かAPI仕様か）を言語を問わず容易に見つけられるようになります。また、大規模パッケージの例にならい**ガイド（ユーザー向け解説）とリファレンス（自動生成APIドキュメント）を明確に分離**することで、それぞれに適した記述スタイル・深度で情報提供できます [2]。

なお、TorchMetricsのように分野別ガイドを設けるケース [1] もありますが、GWexpyは主に重力波データ解析という1つのドメイン内の機能集なので、ガイドは用途別トピック（時系列解析、スペクトル解析、GUIツールの使い方など）に分けると良いでしょう。

2. 現行構成との比較: 維持すべきURLと変更可能な箇所

現行のディレクトリ構成（`docs/web/`以下）では、言語別フォルダの構造に不整合がある可能性があります。例えば現在は`docs/guide/`（英語ガイド）と`docs/ja/guide/`（日本語ガイド）が別々に存在し、さらにリファレンス内に言語別サブフォルダ（`reference/en/`や`reference/ja/`）が二重にネストしているといった冗長構造が報告されています[10]。提案構成へ移行する際には、こうした不要な入れ子を解消してフラットに整理します（既存の`reference/en/en/`といったパスは廃止し、一段階で言語を識別する）[11][12]。

移行にあたり維持すべき既存URLと、**変更（破棄）しても許容できるURL**を分類します。

- **維持すべきURL（互換性を保つべきページ）**:
- **主要な入り口ページ**: 各言語トップページ（例: 現行の英語トップ`docs/web/en/index.html`、日本語トップ`docs/web/ja/index.html`）はブックマークや検索エンジンから直接訪問される可能性が高いため、できる限り同じURLで提供するかリダイレクトを設定します。
- **基本ガイドページ**: ユーザーガイドの中でも特に参照頻度が高いページ（例：「はじめに」「クイックスタート」「GWpyユーザー向けガイド」など）はURL維持または転送を検討します。例えば、現在`docs/web/ja/guide/gwexpy_for_gwpy_users_ja.html`のようなページがあるなら、新構成でも同等のページ（内容更新を伴うとしても）を持ち、旧URLから新URLへの誘導を行うべきです。
- **APIリファレンスの主要エントリ**: 主要なクラスや関数のリファレンスページ（例:`FrequencySeriesList`等）は、外部から直接リンクされている可能性があります。完全自動生成の場合URLは規則的ですが、クラス名ページが移動・改名される場合は互換性を考慮します。

- **リリースノート等**: 以前の特定バージョンのリリースノートに直接リンクしているユーザーは少ないかもしれませんが、プロジェクトの信頼性のため変更しないか、少なくとも最新版の変更履歴ページはURLを維持します。

- **変更してもよいURL（移行時に再編成する箇所）**:

- **内部の細分化ページ**: 今回の再構成で新設・統合する予定のページ群について、旧URLとの一対一対応が難しいものは新構成に合わせます。例えば「Advanced Guide」に相当する詳説ページが英語では存在せず日本語でのみ存在していた場合（あるいはその逆）、新体系では英語版を新規作成するか、日本語版を英語版に統合するなどして**新しいURL**を設定します。こうしたページ（元々利用者が限定的だったページ）は旧URLを積極的に維持せず、必要なら後述のリダイレクトでカバーします。
- **自動生成リファレンスの細部**: APIリファレンス内で大量のページ（関数・メソッド個別ページ）が生成されている場合、ディレクトリ構造整理に伴いパスが変わる可能性があります。例として`docs/web/en/reference/en/gwexpy.module.ClassName.html`→`docs/web/en/reference/gwexpy.module.ClassName.html`のように**ネストが一段減る**変更が起きるでしょう[12]。これら個々のURLは利用者が直接知っているケースは少ないため、一括して構造変更することを優先し、互換性維持は必須ではありません（必要なら後述の自動リダイレクト設定を検討）。
- **開発者向け文書**: `docs/developers/`以下の開発者向けドキュメントはウェブ公開から除外する予定[13]ですので、GitHub上のMarkdown閲覧に委ね、ウェブ上のURL互換は気にしなくてよいでしょう。この部分はユーザー向けサイトから切り離すため、現行で誤って公開されていたとしても問題ありません。

以上の分類により、**ユーザーが直接アクセスする可能性の高いURL**（トップページや主要ガイド）は極力保全し、それ以外（内部構造の再編成で変わる部分）は大幅な改名・移動を許容すると定めます。こうすることで、新構成への移行自由度を確保しつつ、ユーザー体験を損なわないバランスを取ります。

3. URL互換性維持の方法と利点・欠点

移行によって生じる旧URLと新URLの不一致に対して、以下のような**URL互換性維持策**を検討します。それぞれの手法のメリット・デメリットを比較し、適切な方法を選択します。

- **リダイレクトページ（自動転送）を設置**: 古いパスに対し、新ページへ即座に転送する仕組みです。静的サイトの場合、転送用の薄いHTMLページを旧URL位置に配置し、`<meta refresh>`やJavaScriptで新URLへジャンプさせる方法や、Sphinx拡張を使って転送設定を埋め込む方法があります。例えば、Sphinx拡張の**sphinxext-rediraffe**や**sphinxcontrib-redirects**を利用すると、削除・リネームしたページへのリダイレクトを生成できます [14]。メリットはユーザーが**ブックマークや検索経由で古いURLにアクセスしても自動的に新ページに案内される**ためスムーズであることです。また一度設定すれば多数のページも包括的に処理でき、サイト全体のリンク切れを防げます。デメリットは、静的環境では完全なHTTPステータスコード(301等)の転送にならずクローラによっては認識が遅れる可能性があること、また転送用のページ生成や拡張設定に多少手間がかかる点です [14] [15]。とはいえユーザー影響を最小にするためには有力な方法です。

- **移転告知ページ（手動誘導）を設置**: 旧URLにアクセスした際に「このページは移動しました」の旨を表示し、新URLへのリンクを掲載する方法です。具体的には旧パスと同名の`.rst`ファイルを残し、内容に「本ページは**新しい場所**に移転しました。数秒後に自動でジャンプしない場合は[こちら]をクリックしてください」などと書いておくイメージです。メリットは**ユーザーに移行情報を伝えられる**ため、ブックマーク更新を促すことができ、また自動転送が技術的に難しい場合でも実現が容易な点です。デメリットはユーザーが一手間クリックする必要がある（自動ジャンプを入れない場合）ことと、一度に大量のページを移転した場合は旧ページそれぞれに告知を用意する必要がありメンテナンス負荷が高まることです。加えて見た目上は「ページが移動した」という一時的な不便さをユーザーに感じさせてしまいます。

- **URLエイリアス/マッピング設定**: サーバ側で旧→新URLマッピングを設定できる場合の方法です。GitHub Pagesではサーバサイドのリダイレクト設定はできませんが、Read the Docs等では`redirects`設定が使えるケースもあります。この方法は運用環境に依存するため、GWexpyの公開形式（おそらくGitHub Pages）では現実的でないかもしれません。

以上を踏まえると、GitHub Pagesによる静的ホスティングでは**旧ページに薄い転送ページを置くアプローチ**が適切と考えられます。具体的には、移行で消える各ページについて、対応する`.rst`ファイルを残し以下のような内容を記述します：

```
.. redirect from old_page_name:

旧コンテンツは新しいドキュメント構成に移動されました。
自動的にページを転送しています... [新しいページ](/gwexpy/docs/web/en/new_section/new_page.html) を開く。
```

（上記は擬似コードで、実際にはsphinxext-rediraffe等を用いて自動生成させることも検討します。）

利点: ユーザーはブックマーク変更の猶予期間が得られ、リンク切れによる混乱を避けられます。**欠点**: 完全な転送ではないためユーザーが一瞬旧ページに到達する遅れがあります。また運用上いつまで旧ページを残すか判断が必要です。

最終的には、**主要ページはリダイレクトで維持**し、細部のページはサイト内検索や目次から辿れるので404でも許容、といったハイブリッド運用も考えられます。例えば「GWpyユーザー向けガイド（旧URL）」は転送を仕込み、細かいAPI項目ページは新サイトで検索できればよしとする、といった具合です。

4. `.rst`と`.md`混在への対応方針（MyST拡張の利用有無別）

現行ドキュメントではReStructuredText（.rst）とMarkdown（.md）が混在していますが、Sphinxコンフィギュレーションで MyST Parserを導入済みであり、Markdownもビルドに含められる設定になっています [16] [17]。MyST拡張はMarkdown記法のままSphinxの機能（ディレクティブやロール）を利用できる強力なツールです。これを活用するかどうかで、ドキュメントのフォーマット統一戦略が変わります。

MyST拡張を今後も使用する場合（推奨）： MySTを有効化している利点を活かし、**フォーマットをMarkdownに統一**することを検討します。具体的には、現在`.rst`で書かれているユーザーガイド部分をMarkdownに書き換えるか、あるいは新規ページはMarkdownで書く方針とします。一方、APIリファレンス部分はautodocの自動生成テンプレートがあり`.rst`のままでも問題ありませんが、MyST上でも同様の記法が可能です。**メリット**: Markdownはより直観的で執筆しやすく、GitHub上でプレビューもしやすい点があります。開発チーム内にMarkdown志向のメンバーが多い場合、貢献ドキュメントの敷居が下がります。また Jupyter NotebookからMarkdown（MyST対応の`myst-nb`など）に変換して取り込むフローもスムーズです。**留意点**: MyST記法ではSphinx特有の構文（例えば`.. toctree::`や各種ディレクティブ）をMarkdown内で書く際に若干の記法ルールがあります。例えば見出しレベルや空行などRSTとの違いに注意が必要です。今後もMySTを使い続けるのであれば、プロジェクト内でMarkdown記法＋必要な箇所でMySTディレクティブを使うスタイルをドキュメント化しておくとよいでしょう。

MyST拡張を使用しない（RSTに統一する）場合: 仮にMarkdownパーサを無効にする方向であれば、**すべてのドキュメントをRST形式に揃える**必要があります。その場合、現状Markdownで書かれているページ（例: 日本語ガイドの一部など）はRSTに書き直すか、Pandoc等のツールで変換します。RST統一のメリットは、Sphinx標準構文に完全準拠する安心感と、一貫したフォーマット管理です。ただしデメリットとしては、**技術記事執筆者にRST記法の習熟を求める**ことになり、Markdownに比べて扱いづらいと感じる場合がある点です。またJupyter Notebookとの連携においても、直接NotebookをSphinxに組み込む際はnbsphinxを使うのでフォーマット差は吸収できますが、Notebook由来のMarkdownセルをRSTに直す手間が発生することも考えられます。

現状、conf.pyを見る限りMySTは既に導入されています [16]。したがって**混在を許容しつつ徐々にMarkdownに寄せる**のが得策と思われます。移行に際しては以下のガイドラインで整理します：

- **基本方針**: ユーザー向け説明文書（ガイド、チュートリアル等）は執筆効率を重視してMarkdownベース（MyST記法）で統一する。自動生成のAPIリファレンスや複雑な目次ファイル（index.rstなど）はRSTのままでもよいが、可能ならMySTに置き換えても構わない。
- **手順**: まず既存のMarkdownページとRSTページを洗い出し、内容が重複していないか確認します。次に、それぞれのページで用いているSphinx構文（TOCツリーやクロスリファレンス等）をチェックし、Markdown移行時に同等表現が可能か確認します（MySTドキュメントを参照）。例えば`.. code-block::`はMarkdownでは``` ``囲み+言語指定で書けますし、`.. note::`等のアドモンもMySTでは`:::{note}`ブロックで記述できます。
- **運用**: ドキュメント執筆ルールを整備し、新規追加は原則MySTに沿ったMarkdownで行う、とチームに展開します。一方で古いRSTファイルも無理に全部書き換えず、更新時に順次Markdown化するぐらいの移行でもよいでしょう。Sphinxは`.rst`と`.md`を混在ビルドできるため、過渡期の混在自体は問題ありません [17]。

このように、**MySTを活用する前提でMarkdown主体に移行する**のが現実的かつ効率的と考えられます。最終的には、寄稿者がストレスなく書ける形式を尊重しつつプロジェクトの統一感を持たせることが重要です。

5. Codex等による再構成依頼時の作業項目と指示内容

ドキュメント再構成をAI（例えばOpenAI CodexやGPT系モデル）に支援させる場合、明確なタスク分割と出力指針を与えることが成功の鍵となります。以下に、Codexへ依頼する際に列挙すべき作業項目と、期待するアウトプット形式の指示をまとめます。

- **作業項目1: 現行ドキュメント構造の調査とバックアップ**
  指示:　「まず `docs/` 以下の現行ファイル構成をリストアップしてください。特に `docs/web/en/` と `docs/web/ja/` 内のファイル・フォルダ構造を把握し、一覧を表示してください。」
  内容: このステップではAIに現在のファイルツリーを取得・提示させ、どのファイルがどこにあるか確認します（誤って必要ファイルを消さないように）。Git上でのバックアップは人間が担保するとして、AIには読み取り専用でも現状構造を認識させます。

- **作業項目2: 新ディレクトリ構造の作成**
  指示: 「提案されたターゲット構造に基づき、`docs/web/en/` および `docs/web/ja/` ディレクトリを整備してください。必要に応じて新規フォルダ（例: `user_guide/`, `examples/`, `reference/` など）を作成し、既存ファイルを移動します。」
  内容:　AIには具体的な構造（前述のツリー図を参照）を伝え、例えば以下のような操作をさせます[18]:
  - `docs/web/en/guide/` → `docs/web/en/user_guide/` のフォルダ名変更（もし「guide」を「user_guide」に改名するなら）。
  - `docs/web/en/reference/en/` 内のファイルを一段上の `docs/web/en/reference/` に移動[19]（二重になっている言語フォルダの解消）。
  - 日本語側も同様に `docs/web/ja/guide/` を必要なら改名 or 移動、`docs/web/ja/reference/ja/` 内のファイルを上位へ移動[20]。
  - 存在しなかったフォルダ（例えばexamplesディレクトリ等）の新規作成。 Codexには一度に大量の移動を誤りなく行わせる必要があるため、「英語版から順に処理し、その後日本語版を処理」と段階的に命令します。また**ファイル移動後の構造をツリー表示**させ、中間確認するよう指示します（例:「以下のようなツリーになるはずです。実際のディレクトリ構成を表示して検証してください。」と伝える）。

- **作業項目3: コンテンツファイルの移動と名称変更**
  指示:　「各言語フォルダ内の `.rst` / `.md` ファイルを新構成に対応する場所へ移してください。例えば、`docs/web/en/index.rst` はそのまま（英語トップとして）使用し、`docs/index.rst` があればそれはルートリダイレクト用に変更します。`docs/ja/index.rst` は `docs/web/ja/index.rst` に移します[21]。ガイドやリファレンス内のファイルも同様に移動してください。」
  内容:　ここでは実ファイルの移動操作です。AIにはファイルパスを正確に指定させ、Gitの差分追跡も考慮して `git mv` コマンド相当の操作をさせます。**名称変更**（リネーム）が必要な場合（例えばファイル名に言語コードが含まれている場合や、セクション名変更によるファイル名変更）はこの段階で対処します。AIへの指示としては「ファイルXをYにリネームして移動」と具体的に一覧を渡すと安全です。例えば:
  - 「`docs/web/ja/guide/oldname_ja.md` を `docs/web/ja/user_guide/newname.md` にリネーム&移動」など一覧化。 移動後は、**各 `index.rst` のtoctreeを更新**する必要もあります（次項で対応）。

- **作業項目4: TOCツリーとリンクの更新**
  指示: 「目次ページ（index.rstや各セクションのindexファイル）の `toctree` ディレクティブ内のパスを新構成に合わせて修正してください[22]。また、各ページ中で他ページへの相対リンクがある場合、そのパスも見直してください（特に言語間リンク: 英→日のリンクは新パスに変更[23]）。例：英

語トップから日本語トップへのリンクは以前 `../../ja/index.html` だったのを、新構成でも有効か確認。」
内容: ファイル移動によって、Sphinxの `toctree` で参照しているパスや、RST/Markdown内部の `:ref:` やリンクURLが切れていないか点検・修正する工程です。AIには各主要indexファイルを開いて編集させます。例えば `docs/web/en/index.rst` を開かせ、旧構成の節リストを新しい順序・名前に書き換えます。また、言語切替リンク（おそらく各ページに「[日本語] [English]」のようなリンクが埋め込まれている）は、新ディレクトリに合わせパスを書き換えます。**注意**: このリンク更新作業は自動一括置換でミスが出やすいので、AIに慎重にパターンを確認・置換させます。計画書にも「toctreeパスの更新と言語リンクの修正」が明記されている通りです [22] 。

- **作業項目5: URL互換性維持のための処置**
指示: 「旧URLから新URLへのリダイレクトもしくは案内ページを実装してください。具体的には、移動または名称変更により**消滅する各ページ**について、旧パスと同じ場所に新規RSTファイルを作り、そこから新ページへのリンクを張る内容にしてください。または、sphinxext-rediraffeを導入する場合は、`conf.py` にリダイレクト設定を追記し、`redirects.txt` ファイルに旧→新パスのマッピングを記述してください。」
内容: これはAIによる自動化が難しい部分ではありますが、定型的な対応は可能です。例えばAIに対して「以下のリストの旧→新パスについて、順次プレースホルダページを作成せよ」と指示します。先述の通り、重要ページのみ対処する方針なら対象も限定的です。AIにはプレースホルダのテンプレートを与えるとよいでしょう（「`.. meta:: refresh` を入れる」など具体的なマークアップまで指示する）。Rediraffe拡張を使うなら、その設定ファイル生成をAIに任せることも可能です。「以下の内容で `redirects.txt` を作成してください…」といった具合です [24] [25] 。

- **作業項目6: フォーマット統一と整合性チェック**
指示: 「`.rst` と `.md` の混在状況を整理してください。MySTを使う前提で、可能な限りMarkdownに変換できるものはしてください。たとえば、簡単なガイドページでRSTのものはMarkdown記法に書き換えて `.md` 拡張子にリネームしてください（目次とリンクの整合性に注意）。逆に、どうしてもRSTでないと書きづらいAPIリファレンスの目次などはそのままで構いません。最終的に、各ディレクトリ内のファイル拡張子の使われ方を統一します。」
内容: 具体的にはAIに対し、Markdown化するファイルの一覧と変換ルールを与えて処理させます。CodexであればRST→MDの簡易変換（見出し記法やコードブロック記法の置換など）もある程度可能でしょう。ここでも一気に変換させると誤りが出る可能性があるため、1ファイルずつ変換させてレビューする手順を踏みます。**MySTディレクティブ**への対応（例: `.. note::` → `:::note` ブロックへ）もAIにパターン学習させ、統一フォーマットに沿わせます。

- **作業項目7: ビルドと検証**
指示: 「ドキュメントをビルドし、エラーや警告がないことを確認してください [26] 。`sphinx-build -nW -b html docs docs/_build/html` を実行し、エラー時はログを提示してください。問題がなければ、生成されたHTMLの主要なページパスを列挙して構造が意図通りか確認してください。」
内容: AIにSphinxビルドを実行させ、その出力ログをチェックします。エラーが出ればどのファイルで発生したか報告・修正させます（例えば「toctreeに含まれないページがある」等の警告は見逃さず対応）。`-nW` オプションでWarningをErrorとして扱わせているなら、完全に警告がなくなるまで修正です。さらに `linkcheck` ビルダーで内部リンク切れをチェックさせる手もあります [27] 。ビルド成果物については、AIにファイル一覧を出力させ、たとえば `docs/_build/html/web/en/` 以下にindex.htmlや各ページが存在することを確認します。ここで**最終的なサイトマップ**が完成するので、人間とAI双方で認識を揃えます。

- **作業項目8: 新旧構成の比較出力（ファイル構成ツリーの提示）**
指示: 「最終的な `docs/web/` 以下のディレクトリツリーをテキストで出力してください（`tree` コマンド形式で構いません）。**英語版と日本語版の構成が並行して対称になっていること**を確認しま

す。」
内容: これは成果物の提示指示です。AIに対して、Unix系の `tree` ユーティリティの出力形式で構造をリストさせるか、Pythonでディレクトリを再帰列挙するコードを書いて実行させ、その結果テキストを取得します。重要なのは、人間がレビューしやすいフォーマットで提示させることです。例えば以下のような出力が得られるでしょう： [9]

```
docs/web/
├── en/
│   ├── index.html
│   ├── introduction.html
│   ├── user_guide/
│   │   ├── quickstart.html
│   │   ├── ...
│   │   └── tutorials/
│   │       ├── tutorial1.html
│   │       └── ...
│   ├── examples/
│   │   └── example1.html
│   ├── reference/
│   │   ├── index.html
│   │   └── gwexpy.module.ClassName.html
│   ├── faq.html
│   ├── changelog.html
│   └── contribute.html
└── ja/
    ├── index.html
    ├── introduction.html
    ├── user_guide/
    │   ├── quickstart.html
    │   └── ...
    └── （以下略）
```

AIには、上記のような**HTML生成物ベース**でも、もしくはソースRST/MDファイルベースでも構いませんが、統一したツリーを出力させます。この情報は、最終レビューやチーム共有用の再構成結果報告にも利用できます。

- **作業項目9: （必要なら）Git差分の説明とコミットメッセージ提案**
  指示: 「以上の変更をまとめるためのコミットメッセージ案を出力してください。どのファイルを新設・移動・変更したかを箇条書きにしてください。」
  内容: ドキュメント構造変更はプルリクエストでも大きな変更になるため、変更点を整理したメッセージが重要です。AIに変更概要をリストアップさせることで、人間の書き忘れ防止になります。例えば:
  - "Reorganized docs structure: moved English content to docs/web/en, Japanese to docs/web/ja [18] ; flattened reference folders; updated toctrees and links; added redirect pages for moved URLs; converted guide pages to MyST markdown." 等、計画に沿った記述を作成させます。

以上のステップをCodexに指示する際には、**逐一確認とフィードバックを行いながら進める対話型のアプローチ**が望ましいです。特にファイル操作系のコマンドは取り返しがつかない場合もあるため、AIからの応答を注意深くレビューし、安全と判断して次に進めます。幸い、本提案では各フェーズが明確に分かれており、

計画段階から段取りが決まっています [28] [29]。この計画を共有しつつ、Codexには一つ一つのタスクを遂行させれば、大きな齟齬なく移行が完了できるでしょう。

最後に、以上のドキュメント再構成後は、ユーザーに向けて**「ドキュメント構成を刷新しました」**というアナウンスを行い、場合によっては旧構成からの主な変更点（ページ場所の変化、リンク切れ対策など）を伝えることも検討してください。新しい情報アーキテクチャのもとで、GWexpyユーザーがより使いやすいドキュメント体験を得られることを期待します。

参考文献・出典: 他プロジェクトのドキュメント構成例 [2] [4]、GWexpy開発リポジトリ内のドキュメント再編計画 [8] [9]、Sphinxリダイレクト拡張のドキュメント [14] など。これらを総合して本提案を作成しました。今後の実施にあたっては、本提案をたたき台に関係者で詳細を詰めてください。

---

[1] torchmetrics.utilities — PyTorch-Metrics 1.8.2 documentation
https://lightning.ai/docs/torchmetrics/stable/references/utilities.html

[2] [3] [6] Documentation — GeoPandas 1.1.2+0.g81214bf.dirty documentation
https://geopandas.org/en/stable/docs.html

[4] [5] scikit-image's documentation — skimage 0.26.0 documentation
https://scikit-image.org/docs/stable/

[7] seaborn: statistical data visualization — seaborn 0.13.2 documentation
https://seaborn.pydata.org/

[8] [9] [10] [11] [12] [13] [18] [19] [20] [21] [22] [23] [26] [28] [29] plan_docs_restructure.md
https://github.com/tatsuki-washimi/gwexpy/blob/710fd8a87b96c0bcb1530b49d0b3b5380d1a1db2/docs/developers/archive/plans/plan_docs_restructure.md

[14] [24] [25] sphinxext-rediraffe 0.3.0 documentation
https://sphinxext-rediraffe.readthedocs.io/en/latest/

[15] sphinxcontrib-redirects - PyPI
https://pypi.org/project/sphinxcontrib-redirects/

[16] [17] conf.py
https://github.com/tatsuki-washimi/gwexpy/blob/710fd8a87b96c0bcb1530b49d0b3b5380d1a1db2/docs/conf.py

[27] GWExPy_Documentation_Fixes_and_Quality_Improvement_Plan_20260128_1415.md
https://github.com/tatsuki-washimi/gwexpy/blob/710fd8a87b96c0bcb1530b49d0b3b5380d1a1db2/docs/developers/archive/plans/GWExPy_Documentation_Fixes_and_Quality_Improvement_Plan_20260128_1415.md