

Guide d'intégration du manuel utilisateur dans une application JavaFX

Ce guide explique comment doter une nouvelle application JavaFX d'un manuel utilisateur intégré avec `mgt-manuel-core` : production du manuel, affichage dans l'application, écran d'administration optionnel et personnalisation de la charte graphique.

Application de référence : **Sirius-FT** intègre le viewer (menu A propos → « Manuel utilisateur ») exactement comme décrit ici ; son manuel est produit avec **Manuel Studio** et déposé dans le répertoire configuré (clé `REP_MANUEL`).

1. Vue d'ensemble

Le manuel est un **bundle de fichiers autonome** posé dans un répertoire :

```
<repertoire-manuel>/
├─ index.html           page d'accueil (sommaire cliquable)
├─ 01_<document>.html  une page HTML par document source (préfixe = ordre)
├─ images/             images extraites (Word) ou diapositives rendues (PowerPoint)
├─ styles.css           feuille de style des pages (rendue par le thème)
├─ toc.json            manifeste de navigation lu par le viewer
└─ sources/            copies des documents importés + sources.json (régénération)
```

Deux briques l'exploitent :

Brique	Rôle
<code>ManuelGenerator</code>	produit le bundle à partir de documents Word/PowerPoint
<code>HelpViewer</code>	affiche le bundle (sommaire arborescent, recherche, zoom, navigation)

La charte (couleurs, logo, CSS) des deux côtés est portée par un `ManuelTheme` .

2. Prérequis et dépendance

- **JDK 17+** et **JavaFX 17+** avec les modules `javafx-controls` **et** `javafx-web` (le viewer repose sur `WebView`). La lib les déclare en scope `provided` : c'est votre application qui choisit et fournit sa version de JavaFX.
- La lib doit être installée dans le dépôt Maven (local ou d'entreprise) : `cd mgt-manuel-module && ./build.sh` (fait un `mvn -o install`).

```

<dependency>
  <groupId>com.megatim.manuel</groupId>
  <artifactId>mgt-manuel-core</artifactId>
  <version>1.0</version>
</dependency>

<!-- Fournis par VOTRE application (versions à votre main, >= 17) -->
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>17.0.7</version>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-web</artifactId>
  <version>17.0.7</version>
</dependency>

<!-- Logging : la lib n'embarque que slf4j-api. Apache POI logge via log4j2-api :
      ajoutez le pont vers votre backend (exemple avec Logback). -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.5.6</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-to-slf4j</artifactId>
  <version>2.20.0</version>
</dependency>

```

⚠ Si vos modules JavaFX viennent de dépendances transitives, vérifiez avec `mvn dependency:tree -Dincludes=org.openjfx` qu'une **seule** version est résolue : un runtime mélangé (ex. base 11 + web 17) fait planter WebView au démarrage (PlatformLogger ... not implemented).

3. Choisir le répertoire du manuel

Décidez où vit le bundle dans votre application. Recommandation : un répertoire relatif au dossier d'installation, rendu configurable par une clé de configuration (pattern ManuelConfig de Sirius-FT) :

```
// Exemple minimal : <dossier-courant>/manuel, surchargé par une propriété
public final class MonManuelConfig {
    public static Path repertoireManuel() {
        String rep = System.getProperty("app.manuel.dir",
            Paths.get("").toAbsolutePath() + File.separator + "manuel");
        return Path.of(rep);
    }
}
```

4. Produire le manuel

Option A — avec Manuel Studio (sans écrire de code)

1. Lancer l'application autonome : `java -jar mgt-manuel-studio-1.0.jar`.
2. Ajouter les documents Word/PowerPoint, saisir titre du projet / version / titre de l'aide.
3. Choisir comme dossier de sortie le répertoire du manuel de votre application.
4. (Optionnel) Charger votre thème `.properties` pour que le `styles.css` généré porte vos couleurs.
5. « Générer le manuel », vérifier l'aperçu, livrer le dossier avec l'application.

Option B — depuis votre application (écran d'administration intégré)

```
ManuelGenerator generator = new ManuelGenerator(monTheme); // ou new ManuelGenerator()
Path manuel = generator.generate(
    MonManuelConfig.repertoireManuel(), // répertoire cible (remplacé atomiquement)
    List.of(Path.of("Manuel utilisateur.docx")), // sources ordonnées
    "Mon Application", // titre du projet (en-tête du viewer)
    "1.0.0", // version
    "Manuel utilisateur", // titre de l'aide
    ligne -> Platform.runLater(() -> journal.appendText(ligne + "\n")));
```

Points importants : - **Jamais sur le thread JavaFX** : lancez `generate(...)` dans un `javafx.concurrent.Task` (les convertisseurs PowerPoint rendent les diapositives en PNG via AWT et la conversion peut durer plusieurs secondes). - La génération est **atomique** : elle se fait dans `<cible>.tmp` puis remplace le répertoire ; un échec ne détruit jamais le manuel existant. - Les documents importés sont copiés dans `<cible>/sources/` avec `sources.json` (`SourcesManifest.load(dir)`) : votre écran d'administration peut recharger la liste des sources et les métadonnées entre deux sessions, et régénérer sans les originaux. - Formats acceptés : `.docx`, `.doc`, `.pptx`, `.ppt` (filtrez vos `FileChooser` avec `ConverterFactory.isSupported(nomFichier)`).

5. Afficher le manuel dans l'application

Intégration minimale

```
import com.megatim.manuel.core.viewer.HelpViewer;

HelpViewer viewer = new HelpViewer(MonManuelConfig.repertoireManuel());
Parent contenu = viewer.build();      // IOException si toc.json absent/corrompu
monConteneur.setCenter(contenu);      // s'intègre dans n'importe quel layout
maFenetre.setTitle(viewer.title());   // "Mon Application – v1.0.0"
```

`build()` retourne un `Parent` classique : onglet, fenêtre dédiée, panneau latéral... Le viewer fournit sommaire arborescent + recherche instantanée, navigation Précédent/Suivant (grisée selon l'historique), bouton Accueil et zoom.

Entrée de menu avec garde-fou (pattern Sirius-FT)

```
@FXML
public void ouvrirManuel(ActionEvent e) {
    Path toc = MonManuelConfig.repertoireManuel().resolve(HelpExporter.TOC_JSON);
    if (!Files.exists(toc)) {
        afficherAvertissement("Aucun manuel disponible. Contactez votre administrateur.");
        return;
    }
    Stage stage = new Stage();
    HelpViewer viewer = new HelpViewer(MonManuelConfig.repertoireManuel());
    stage.setScene(new Scene((Parent) viewer.build(), 1120, 700));
    stage.setTitle(viewer.title());
    stage.show();
}
```

6. Personnaliser la charte graphique

Le thème par défaut (`ManuelTheme.defaultTheme()`) reproduit la charte Sirius-FT. Pour votre application, construisez un thème et passez-le **aux deux briques** (le générateur pour le `styles.css` des pages HTML, le viewer pour l'interface JavaFX) :

```

ManuelTheme theme = ManuelTheme.builder()
    // Interface JavaFX du viewer
    .accentColor("#ffe3d0")           // bouton Accueil + hover
    .buttonColor("#faf6f2")           // boutons de la barre d'outils
    .buttonBorderColor("#d9b89c")
    .panelBorderColor("#e0d5ca")      // encadrés (sommaire)
    .selectionColor("#ffd9c2")        // sélection dans le sommaire
    .hoverColor("#fff3ea")
    .textColor("#3a2c20")
    .backgroundColor("#ffffff")
    // Identité
    .appTitle("Mon Application")
    .logo(getClass().getResource("/images/logo.png"))
    // Pages HTML générées
    .htmlVar("html.h1", "#8a3e00")
    .htmlVar("html.link", "#c2571a")
    .htmlVar("html.navbar.bg", "#fdf1e7")
    .build();

new ManuelGenerator(theme).generate(...); // bundle aux couleurs du thème
new HelpViewer(dossier, theme).build();  // viewer aux couleurs du thème

```

Trois niveaux de personnalisation du viewer

1. **Couleurs seulement** (le plus courant) : les setters du builder ci-dessus. Aucune CSS à écrire — les valeurs sont injectées en *looked-up colors*.
2. **Surcharge CSS** : `.extraViewerStylesheet(uri)` ajoute votre feuille APRÈS la CSS par défaut. Contrat stable de `styleClass` : `help-viewer` (racine), `button-primary` (bouton Accueil), `help-left-panel`, `help-panel-title`, `help-project-title`, `help-project-version`, `help-status`.
3. **Remplacement complet** : `.viewerStylesheet(uri)` remplace la CSS embarquée (repartez de `mgt-manuel-core/src/main/resources/com/megatim/manuel/core/viewer/help-viewer.css`).

Variables du template HTML

Surchargeables une à une via `htmlVar(cle, valeur)` — défauts entre parenthèses : `html.font`, `html.text` (`#1a1a1a`), `html.bg` (`#ffffff`), `html.h1` (`#1f3a5f`), `html.h2` (`#22507a`), `html.h3` (`#2b6098`), `html.link` (`#1565c0`), `html.caption`, `html.muted`, `html.rule`, `html.navbar.bg` (`#eef2f7`), `html.navbar.border`, `html.navbar.disabled`, `html.table.border`, `html.img.border`. Pour un contrôle total : `.htmlStyleTemplate(monTemplate)` (placeholders `${...}`).

Thème en fichier `.properties`

Le studio (et votre application, via la même classe `ThemeLoader` du module studio ou une copie) peut charger un thème depuis un fichier :

```
accentColor=#ffe3d0
selectionColor=#ffd9c2
appTitle=Mon Application
logo=logo.png
htmlVars.html.h1=#8a3e00
extraViewerStylesheets=file:/opt/monapp/styles/mon-viewer.css
```

7. Rédiger le document Word source

Le convertisseur exploite la structure du document — respectez ces conventions :

Élément du manuel	Convention Word
Page de garde	tout le contenu avant le premier titre
Préambule (onglet dédié)	titres nommés Résumé, Mots clés, Historique, Diffusion
Sommaire numéroté (1, 1.1...)	styles Titre 1 / Titre 2 / Titre 3 (Heading 1-3)
Onglet Illustrations	légendes d'images en style Légende (Caption), ex. « Figure 12 : Page de connexion »
Images	insérées normalement (elles sont extraites dans <code>images/</code>)

Le **nom du fichier** devient le titre du document dans le sommaire — nommez-le proprement (ex. `Manuel utilisateur MonApp v1.0.0.docx`).

8. Packaging et déploiement

- **Livraison du manuel** : livrez le répertoire du bundle avec l'application (installateur) ou générez-le sur site via l'écran d'administration / le studio.
- **Fat-jar (shade)** : la lib est du classpath pur, aucun `module-info` requis. Excluez `META-INF/*.SF|*.DSA|*.RSA` (jars POI signés).
- **Poids** : POI et ses transitives ≈ 15-25 Mo ; `javafx-web` (WebKit) ≈ 45 Mo de natives par plateforme.

9. Dépannage

Symptôme	Cause / correctif
<code>IOException: toc.json introuvable</code>	Aucun manuel généré dans le répertoire — affichez votre message « Aucun manuel disponible » (voir §5).
<code>NoClassDefFoundError: javafx/scene/web/WebView</code>	Le module <code>javafx-web</code> n'est pas fourni par l'hôte — ajoutez la dépendance (la lib est en <code>provided</code>).
Crash <code>WebView PlatformLogger ... not implemented</code>	Versions JavaFX mélangées sur le classpath — alignez tout sur une seule version (<code>dependency:tree</code>).
Fenêtre UNDECORATED impossible à déplacer sur la zone du manuel	La <code>WebView</code> consomme les événements souris : prévoir le drag sur votre barre de titre uniquement.
Génération qui gèle l'interface	<code>generate()</code> appelé sur le thread JavaFX — passez par un <code>Task</code> (voir §4).
Warning <code>multiple Log4j providers</code>	<code>log4j-core</code> présent ailleurs dans l'hôte en plus de <code>log4j-to-slf4j</code> — cosmétique, le pont gagne ; sinon excluez <code>log4j-core</code> .
Caractères  en carrés dans vos surcharges	Police système sans ces glyphes — préférez des libellés texte (le viewer utilise « Précédent / Suivant »).

10. Checklist d'intégration

- [] Dépendance `mgt-manuel-core` + modules JavaFX `controls / web` fournis par l'hôte
- [] Pont de logging (`log4j-to-slf4j` + backend)
- [] Répertoire du manuel choisi et configurable
- [] Entrée de menu « Manuel utilisateur » avec garde-fou `toc.json`
- [] (Optionnel) Écran d'administration ou usage du studio pour générer
- [] (Optionnel) `ManuelTheme` aux couleurs de l'application, passé au générateur ET au viewer
- [] Document Word conforme aux conventions (§7)
- [] Test : générer, ouvrir le viewer, recherche/zoom/navigation, régénérer par-dessus