Guide d'Integration
SolidWorks / SWOOD API

avec Python & Base de Donnees SQLite

Extraction, Automatisation et Liaison Bidirectionnelle
entre CAO/FAO et Systeme d'Information Interne

Document Technique de Reference

Version 1.0 - February 2026

Table des Matieres

1 Introduction et Objectifs

- **1.1** Contexte du projet
- **1.2** Objectifs de l'integration
- **1.3** Technologies utilisees

2 Architecture Globale du Systeme

- **2.1** Vue d'ensemble
- **2.2** Flux de donnees
- **2.3** Composants logiciels

3 API SolidWorks -- Fondamentaux

- **3.1** Hierarchie des objets COM
- **3.2** Connexion depuis Python
- **3.3** Interfaces principales
- **3.4** Types de documents

4 Extraction de Donnees via l'API

- **4.1** Proprietes personnalisees (Custom Properties)
- **4.2** Informations du document
- **4.3** Materiaux et proprietes physiques
- **4.4** Configurations
- **4.5** Traversee d'assemblages (BOM)
- **4.6** Dimensions et cotes
- **4.7** Plans et vues

5 Variables SWOOD Report

- **5.1** Variables globales et projet
- **5.2** Variables assemblage
- **5.3** Variables piece (panneau)
- **5.4** Variables stock et chants
- **5.5** Variables SolidWorks natives
- **5.6** Variables programme CAM

6 SWOOD CAM -- Generation et Automatisation

- **6.1** Configuration Report.cfg
- **6.2** Generation conditionnelle de programmes
- **6.3** Pont de donnees SWCP
- **6.4** Bibliotheque d'outils et agregats

7 Formats d'Export SWOOD

- **7.1** CSV (System Report)
- **7.2** Excel (SWOOD Excel Report)
- **7.3** XML (Template de donnees)
- **7.4** HTML (Rapport systeme)

8 Schema de Base de Donnees SQLite

- **8.1** Modele relationnel complet

8.2 Tables principales
8.3 Relations et cles etrangeres
8.4 Index recommandes

9 Implementation Python

9.1 Connexion COM a SolidWorks
9.2 Extraction des proprietes custom
9.3 Traversee recursive d'assemblage
9.4 Ecriture en base SQLite
9.5 Flux bidirectionnel

10 SWOOD Connect -- Echanges Tiers

10.1 Connecteurs disponibles
10.2 Formats d'echange

11 Scripting SWOOD Design

11.1 Syntaxe VBScript SWOOD
11.2 Variables contextuelles
11.3 Fonctions disponibles (2024)

12 Bonnes Pratiques et Recommandations

12.1 Performances
12.2 Gestion des erreurs
12.3 Securite des donnees

13 Analyse du Report de travail.Cfg

13.1 Structure globale du fichier
13.2 Section (RAPPORT) -- Configuration globale
13.3 Variables Projet et flags d'export
13.4 Classification TypedObject (HA, PRODUCT...)
13.5 Export CSV OptiPlanning
13.6 Numerotation automatique (Serial Numbers)
13.7 Messages de controle qualite
13.8 Documents generes
13.9 Configuration machine CNC

14 Integration DestriChiffrage

14.1 Presentation de DestriChiffrage
14.2 Schema de la base catalogue.db
14.3 Correspondances Report.cfg / catalogue.db
14.4 Proprietes custom SolidWorks a standardiser
14.5 Flux de donnees bidirectionnel
14.6 Module sw_bridge.py

15 Scenarios Concrets d'Utilisation

15.1 Commande fournisseur quincaillerie
15.2 Chiffrage automatique depuis le modele 3D
15.3 Mise a jour des prix depuis le rapport SWOOD
15.4 Passage de commande matiere (panneaux)
15.5 Etat d'avancement et feuille de route

16 Module sw_bridge.py -- Implementation Complete

16.1 Architecture du module

16.2 Connexion COM a SolidWorks
16.3 Lecture et ecriture des proprietes personnalisees
16.4 Traversee d'assemblage et extraction quincailleries
16.5 Synchronisation SolidWorks vers catalogue.db
16.6 Synchronisation catalogue.db vers SolidWorks
16.7 Generation de commande fournisseur
16.8 Mode hors-ligne (import CSV Report SWOOD)
16.9 Export BDD vers format SWOOD
16.10 Interface en ligne de commande (CLI)
16.11 Resultats de la demonstration

Annexes References et Ressources

Chapitre 1

Introduction et Objectifs

1.1 Contexte du projet

Dans le cadre de la modernisation des processus de production, l'integration entre les logiciels de Conception Assistee par Ordinateur (CAO/FAO) et les systemes d'information internes devient un enjeu strategique. SolidWorks, couple a l'extension SWOOD pour l'industrie du bois et de l'ameublement, produit une grande quantite de donnees techniques : dimensions des panneaux, materiaux, chants, quincaillerie, programmes CNC, couts, etc.

Ces donnees, actuellement dispersees entre fichiers SolidWorks, rapports SWOOD et fichiers de configuration, doivent etre centralisees dans une base de donnees relationnelle (SQLite .db) accessible par un logiciel interne developpe en Python.

1.2 Objectifs de l'integration

- **Centraliser** les donnees techniques issues de SolidWorks et SWOOD
- **Automatiser** l'extraction des nomenclatures (BOM), materiaux et programmes
- **Piloter** la generation de programmes CNC depuis le systeme d'information
- **Tracer** l'historique des projets, couts et modifications
- **Eliminer** les ressaisies manuelles et les erreurs associees

1.3 Technologies utilisees

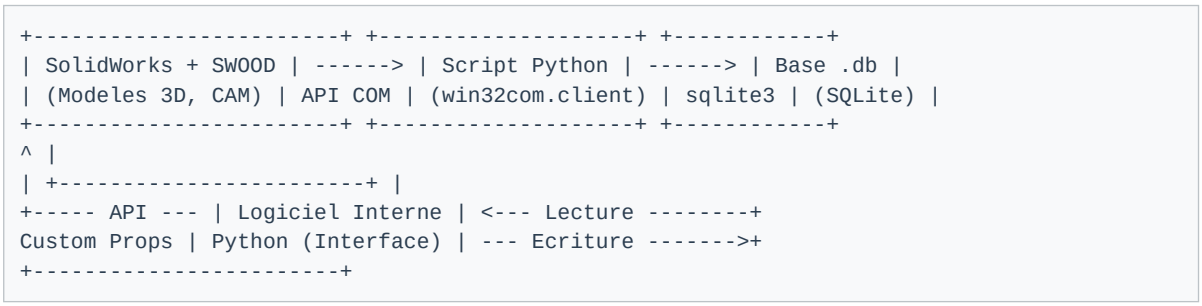
Composant	Technologie	Role
CAO 3D	SolidWorks 2024+	Modelisation pieces et assemblages
Extension bois	SWOOD Design + CAM 2024	Panneaux, chants, quincaillerie, programmes CNC
Langage	Python 3.12+	Scripts d'extraction et logiciel interne
API COM	win32com.client (pywin32)	Interface avec SolidWorks
Base de donnees	SQLite 3 (.db)	Stockage centralise
ORM (optionnel)	SQLAlchemy	Abstraction base de donnees
Rapports	SWOOD System Report	Export CSV/PDF/Excel

Chapitre 2

Architecture Globale du Systeme

2.1 Vue d'ensemble

L'architecture repose sur un flux bidirectionnel entre SolidWorks/SWOOD et la base de donnees. Python sert de couche intermediaire, communiquant avec SolidWorks via l'API COM et avec la base SQLite via le module standard sqlite3.



2.2 Flux de donnees

Flux d'extraction (SolidWorks vers Base .db)

- Ouverture du document SolidWorks (piece ou assemblage)
- Lecture des proprietes custom, materiau, configurations
- Traversee de l'arborescence assemblage (BOM recursive)
- Lecture des variables SWOOD Report (panneaux, chants, couts)
- Ecriture structuree dans les tables SQLite

Flux de pilotage (Base .db vers SolidWorks)

- Lecture des decisions du logiciel interne (ex: panneau valide pour usinage)
- Ecriture de proprietes custom SolidWorks via API COM
- SWOOD lit ces proprietes via la syntaxe **<SWCP.PropertyName>**
- Le Report.cfg utilise PROCESS_SWOODCAM_CONDITION pour generer conditionnellement

2.3 Composants logiciels

Composant	Fichier/Module	Description
Extracteur	sw_extractor.py	Script de collecte des donnees SolidWorks/SWOOD
Modele DB	db_schema.py	Definition du schema SQLite (tables, relations)
API Bridge	sw_bridge.py	Interface COM Python/SolidWorks
Config Parser	cfg_parser.py	Lecture/ecriture des fichiers Report.cfg

Composant	Fichier/Module	Description
CSV Importer	csv_importer.py	Import des exports System Report
Logiciel principal	app_main.py	Application interne (interface utilisateur)

Chapitre 3

API SolidWorks -- Fondamentaux

3.1 Hierarchie des objets COM

L'API SolidWorks est une API COM (Component Object Model) exposant une hierarchie d'objets. Le point d'entree est l'objet **ISldWorks** (application), a partir duquel on accede a tous les autres objets : documents, configurations, proprietes, composants, etc.

```
ISldWorks (Application)
|
+-- IModelDoc2 (document ouvert)
| |
| +-- IPartDoc (piece)
| +-- IAssemblyDoc (assemblage)
| +-- IDrawingDoc (plan)
| |
| +-- IModelDocExtension
| | +-- ICustomPropertyManager
| | +-- SaveAs / SelectByID2
| |
| +-- ISelectionMgr
| +-- IConfiguration
| | +-- ICustomPropertyManager (par config)
| | +-- GetRootComponent --> IComponent2
| |
| +-- IFeature --> GetNextFeature (liste chainee)
| +-- IDimension / IDisplayDimension
|
+-- ICommandManager (menus add-in)
+-- IPropertyManagerPage2 (panneaux UI)
```

3.2 Connexion depuis Python

La connexion a SolidWorks depuis Python se fait via la bibliotheque **pywin32** (module win32com.client). Deux methodes sont possibles :

Methode 1 : Connexion a une instance existante

```
import win32com.client

# Se connecter a SolidWorks deja lance
swApp = win32com.client.GetActiveObject('SldWorks.Application')
swModel = swApp.ActiveDoc

if swModel is None:
print('Aucun document ouvert')
else:
print(f'Document actif : {swModel.GetPathName()}')
```

Methode 2 : Lancement d'une nouvelle instance

```
import win32com.client
```

```
# Creer une nouvelle instance SolidWorks
swApp = win32com.client.Dispatch('SldWorks.Application')
swApp.Visible = True # Rendre visible
```

IMPORTANT : SolidWorks doit etre installe et dispose d'une licence valide sur le poste. L'API COM n'est pas accessible a distance (pas de RPC natif).

3.3 Interfaces principales

Interface	Role	Acces
ISldWorks	Application SolidWorks (point d'entree)	win32com.client.Dispatch(...)
IModelDoc2	Document generique (piece, asm, plan)	swApp.ActiveDoc
IModelDocExtension	Extensions du document	swModel.Extension
ICustomPropertyManager	Gestion proprietes personnalisees	Extension.CustomPropertyManager(config)
IPartDoc	Specifique aux pieces	Cast depuis IModelDoc2
IAssemblyDoc	Specifique aux assemblages	Cast depuis IModelDoc2
IDrawingDoc	Specifique aux plans	Cast depuis IModelDoc2
IConfiguration	Configuration d'un document	swModel.GetConfigurationByName(n)
IComponent2	Composant dans un assemblage	config.GetRootComponent()
ISelectionMgr	Gestion de la selection	swModel.SelectionManager
IFeature	Feature dans l'arbre	swModel.FirstFeature()

3.4 Types de documents

Valeur	Constante	Extension	Description
0	swDocNONE	--	Aucun document
1	swDocPART	.sldprt	Piece
2	swDocASSEMBLY	.sldasm	Assemblage
3	swDocDRAWING	.slddrw	Plan / Mise en plan

```
doc_type = swModel.GetType()
if doc_type == 1: # swDocPART
print('Piece')
elif doc_type == 2: # swDocASSEMBLY
print('Assemblage')
elif doc_type == 3: # swDocDRAWING
print('Plan')
```

Chapitre 4

Extraction de Donnees via l'API

4.1 Proprietes personnalisees (Custom Properties)

Les proprietes personnalisees sont le **mecanisme central** de stockage des metadonnees dans SolidWorks. SWOOD les utilise massivement pour stocker les informations de production. Chaque document et chaque configuration possede son propre jeu de proprietes.

Lecture de toutes les proprietes

```
def get_custom_properties(swModel, config_name=""):
"""Extrait toutes les proprietes custom d'un document."""
cpm = swModel.Extension.CustomPropertyManager(config_name)
props = {}
names = cpm.GetNames()
if names:
for name in names:
# Get5 retourne (status, value, resolved_value, wasResolved, linkTo)
result = cpm.Get5(name, False)
props[name] = {
"value": result[1], # Expression brute
"resolved": result[2], # Valeur resolue
}
return props
```

Ecriture d'une propriete

```
def set_custom_property(swModel, name, value, config_name=""):
"""Definit ou met a jour une propriete personnalisee."""
cpm = swModel.Extension.CustomPropertyManager(config_name)
# Tenter d'ajouter, puis mettre a jour si existe deja
result = cpm.Add3(name, 30, value, 2) # 30=text, 2=overwrite
if result != 0: # Si echec, essayer Set2
cpm.Set2(name, value)
swModel.EditRebuild3() # Reconstruire pour propager
```

Types de proprietes

Valeur	Constante	Type Python	Description
30	swCustomInfoText	str	Chaine de caracteres
32	swCustomInfoNumber	int/float	Nombre
5	swCustomInfoDouble	float	Double precision
11	swCustomInfoYesOrNo	bool	Booleen (Oui/Non)
64	swCustomInfoDate	datetime	Date

ASTUCE : La methode **Get5** est recommandee car elle retourne a la fois la valeur brute (expression) et la valeur resolue. Par exemple, une propriete definie comme "$PRP:SW-Material" sera resolue en "Chene Massif".

4.2 Informations du document

```
# Informations de base
path = swModel.GetPathName() # Chemin complet
title = swModel.GetTitle() # Titre affiche
doc_type = swModel.GetType() # 1=piece, 2=asm, 3=plan

# Informations de resume (Summary Information)
sw_title = swModel.SummaryInfo(1) # Titre
subject = swModel.SummaryInfo(2) # Sujet
author = swModel.SummaryInfo(3) # Auteur
keywords = swModel.SummaryInfo(4) # Mots-cles
comment = swModel.SummaryInfo(5) # Commentaire
saved_by = swModel.SummaryInfo(6) # Dernier enregistrement par
create_date = swModel.SummaryInfo(7) # Date de creation
save_date = swModel.SummaryInfo(8) # Date de sauvegarde
```

4.3 Materiaux et proprietes physiques

```
# Materiau applique
material_name = swModel.MaterialUserName # Ex: "Chene Massif"
material_id = swModel.MaterialIdName # ID interne

# Proprietes de masse (necessite rebuild)
mass_props = swModel.Extension.CreateMassProperty()
if mass_props:
mass = mass_props.Mass # Masse en kg
volume = mass_props.Volume # Volume en m3
surface = mass_props.SurfaceArea # Surface en m2
cog = mass_props.CenterOfMass # Centre de gravite [x,y,z]
```

4.4 Configurations

Chaque document SolidWorks peut avoir plusieurs configurations. SWOOD utilise les configurations pour gerer differentes variantes d'un meme panneau (epaisseurs, materiaux).

```
# Liste des configurations
config_names = swModel.GetConfigurationNames()

# Configuration active
active_config = swModel.GetActiveConfiguration()
print(f"Config active : {active_config.Name}")

# Parcourir toutes les configurations
for cname in config_names:
config = swModel.GetConfigurationByName(cname)
# Proprietes specifiques a cette configuration
cpm = config.CustomPropertyManager
names = cpm.GetNames()
print(f" Config {cname} : {len(names) if names else 0} proprietes")
```

4.5 Traversee d'assemblages (BOM)

L'extraction de la nomenclature (BOM) se fait par traversee recursive de l'arborescence des composants. C'est une fonctionnalite essentielle pour alimenter la base de donnees.

```
def traverse_assembly(swModel, depth=0, results=None):
"""Traverse recursivement un assemblage et collecte les composants."""
if results is None:
results = []
```

```
config = swModel.GetActiveConfiguration()
root = config.GetRootComponent2(False)
children = root.GetChildren()

if children is None:
return results

for child in children:
model_doc = child.GetModelDoc2()
comp_data = {
"level": depth,
"name": child.Name2,
"path": model_doc.GetPathName() if model_doc else "",
"type": model_doc.GetType() if model_doc else 0,
"suppressed": child.GetSuppression2(),
"visible": child.Visible,
"material": model_doc.MaterialUserName if model_doc else "",
}

# Extraire les proprietes custom
if model_doc:
comp_data["properties"] = get_custom_properties(model_doc)

results.append(comp_data)

# Recursion si sous-assemblage
if model_doc and model_doc.GetType() == 2:
traverse_assembly(model_doc, depth + 1, results)

return results
```

4.6 Dimensions et cotes

```
# Acces aux dimensions par nom
dim = swModel.Parameter("D1@Sketch1")
if dim:
value_m = dim.SystemValue # Valeur en metres (SI)
value_mm = value_m * 1000 # Conversion en mm
print(f"Dimension : {value_mm:.2f} mm")
```

4.7 Plans et vues

```
# Si le document est un plan (type 3)
sheet_names = swModel.GetSheetNames()
for name in sheet_names:
swModel.ActivateSheet(name)
sheet = swModel.GetCurrentSheet()
print(f"Feuille : {sheet.GetName()}")

# Parcourir les vues
view = swModel.GetFirstView() # Premiere = la feuille elle-meme
view = view.GetNextView() # Premiere vraie vue
while view:
print(f" Vue : {view.GetName2()}, Type : {view.Type}")
ref_doc = view.ReferencedDocument
if ref_doc:
print(f" Ref : {ref_doc.GetPathName()}")
view = view.GetNextView()
```

Chapitre 5

Variables SWOOD Report

SWOOD Report expose un ensemble complet de variables accessibles dans les documents de rapport (HTML, Excel, CSV). Ces variables peuvent etre utilisees dans les expressions, conditions et templates. Voici le catalogue exhaustif par categorie.

5.1 Variables globales et projet

Variable	Description	Type	Modifiable
FORMATINFO	Options de format (unite, decimales)	String	Oui
ReportPath	Chemin du rapport	String	Oui
ProgramsPath	Repertoire des programmes CNC	String	Oui
READMESHES	Lire les maillages	Boolean	Oui
USESWOORDESIGNDATA	Utiliser donnees SWOOD Design	Boolean	Oui
PROCESS_SWOODCAM_REPORT	Traiter le rapport SWOOD CAM	Boolean	Oui
IGNORE_EXCLUDED_FROM_BOM	Ignorer les exclus du BOM	Boolean	Oui
READEPDMIDS	Lire les IDs EPDM	Boolean	Oui
PROJNAME	Nom du projet	String	Oui
REPORTCFGDIR	Repertoire configuration rapport	String	Non
REPORTCURRENCY	Devise	String	Non
REPORTCURRENCYSYMBOL	Symbole devise	String	Non
DATE / HOUR / NOW	Date/Heure/Maintenant	DateTime	Non

5.2 Variables assemblage

Variable	Description	Type
NAME	Nom de l'assemblage	String
PATH	Chemin du fichier	String
CONF	Configuration active	String
NB	Quantite	Integer
DESC	Description	String
PROGCOUNT	Nombre de programmes	Integer
SWMODELTYPE	Type de modele SolidWorks	Integer

Variable	Description	Type
TOTYPE / TOTYPES	Type d'objet / Types d'objets	String
TYPED	Assemblage type	Boolean
EPDMFILEID	ID fichier EPDM	Integer
EPDMFOLDERID	ID dossier EPDM	Integer
EPDMVAULT	Coffre-fort EPDM	String
ISEPDMFILE	Est un fichier EPDM	Boolean
SENSORS	Alarmes capteurs	String

5.3 Variables piece (panneau)

Ces variables representent les donnees les plus importantes pour l'industrie du bois et du meuble. Elles contiennent toutes les dimensions, materiaux et informations de production d'un panneau.

Variable	Description	Categorie
PAN_STL / PAN_STW / PAN_STT	Longueur / Largeur / Epaisseur brute	Dimensions
PAN_FNL / PAN_FNW / PAN_FNT	Longueur / Largeur / Epaisseur finie	Dimensions
CMAT	Code materiau	Materiau
CCHAR / CCHAV / CCHD / CCHG	Code chant Arriere / Avant / Droit / Gauche	Chants
CHAR / CHAV / CHD / CHG	Avec chant AR / AV / D / G (boolean)	Chants
ECHAR / ECHAV / ECHD / ECHG	Epaisseur chant AR / AV / D / G	Chants
ISDS	Est un debit scie	Type
STOK_AS_GRAINDIRECTION	Stock comme direction du fil	Production
DESC	Description de la piece	Info

5.4 Variables stock et chants

Variable	Description
ST_N / ST_DESC	Nom / Description du stock
ST_CMAT / ST_MAT	Code materiau / Materiau du stock
ST_L / ST_W / ST_T	Longueur / Largeur / Epaisseur brute
ST_COST / ST_MATCOST	Cout du stock / Cout materiau
ST_FIL / ST_DFIL	Fil (boolean) / Direction du fil (angle)
ST_GRAINISVERTICAL	Le fil est vertical (boolean)
ST_ISCURVED	Stock courbe (boolean)
ST_REB / ST_LEB / ST_FEB / ST_BEB	Chant Droit / Gauche / Avant / Arriere
ST_MATTYPE / ST_MATDESC	Type materiau / Description materiau

5.5 Variables SolidWorks natives

Prefixe	Variable	Description
SW	CONF / MATERIAL / PATH / WEIGHT / DESCRIPTION	Config, materiau, chemin, masse, description
SW	REVISIONNUMBER / TITLE / DIRECTORY	Revision, titre, repertoire
SW	SUMMINFO* (AUTHOR, COMMENT, CREATEDDATE...)	Informations de resume du fichier
SWCOMP	CONF / NAME / REFERENCE / VISIBLE	Variables du composant SolidWorks
SWCONF	ALTERNATENAME / COMMENT / DESCRIPTION / Name	Variables de configuration
SWCLP	(PropertyName)	Proprietes de liste de coupe (Cut List)

5.6 Variables programme CAM

Les variables de programme sont accessibles dans le contexte des rapports CAM et permettent d'extraire les informations d'usinage pour chaque piece.

Variable	Description
PROGCOUNT	Nombre total de programmes pour la piece
ProgramFile	Nom du fichier programme genere
ProgramsPath	Chemin du repertoire des programmes
ProgramFileIfExist	Fichier programme si existant
PROCESS_SWOODCAM_REPORT	Active/desactive le traitement CAM
PROCESS_SWOODCAM_CONDITION	Condition de generation CAM

Chapitre 6

SWOOD CAM -- Generation et Automatisation

6.1 Configuration Report.cfg

Le fichier **Report.cfg** est le point de controle central pour la generation automatique de programmes CNC via SWOOD Report. Il utilise un format INI-like avec des sections hierarchiques.

```
(RAPPORT) ; Section globale
PROCESS_SWOODCAM_REPORT = 1 ; Active le traitement CAM
PROCESS_SWOODCAM_CONDITION = TRUE ; Condition par defaut

[PART] ; Section par piece
PROCESS_SWOODCAM_CONDITION = REPORT_CAM ; Condition basee sur variable

[ASSEMBLY] ; Section par assemblage
PROCESS_SWOODCAM_CONDITION = FALSE ; Desactive pour les assemblages
```

6.2 Generation conditionnelle de programmes

Le mecanisme de generation conditionnelle permet de controler quels panneaux produisent des programmes CNC. C'est la cle de l'integration avec un logiciel externe.

Patron A : Controle global au niveau projet

```
# Dans Report.cfg :
PROCESS_SWOODCAM_CONDITION = REPORT_CAM

# Variable SWOOD 'REPORT_CAM' definie avec :
# Evaluation = <SWCP.REPORT_CAM>
# --> Lit la propriete custom SolidWorks 'REPORT_CAM'
# --> Si valeur = 'Yes' => programme genere
# --> Sinon => programme ignore
```

Patron B : Controle par piece

```
# Chaque piece a sa propriete 'GENERER_PROGRAMME'
# Python ecrit cette propriete via l'API COM :
set_custom_property(swModel, 'GENERER_PROGRAMME', 'Oui')

# SWOOD lit automatiquement via <SWCP.GENERER_PROGRAMME>
```

IMPORTANT : Ce mecanisme est le pont bidirectionnel entre votre base .db et SWOOD : le logiciel interne decide quels panneaux doivent etre usines, ecrit la propriete via l'API COM, et SWOOD genere les programmes en consequence.

6.3 Pont de donnees SWCP

La syntaxe **<SWCP.PropertyName>** (SolidWorks Custom Property) est le mecanisme formel qui connecte les donnees du modele SolidWorks au moteur d'evaluation SWOOD. C'est le point d'integration le plus important.

Syntaxe SWOOD	Propriete SolidWorks lue	Usage typique
<SWCP.Description>	Description	Affichage dans le rapport
<SWCP.REPORT_CAM>	REPORT_CAM	Condition generation programme
<SWCP.NUM_COMMANDE>	NUM_COMMANDE	Numero de commande client
<SWCP.PRIORITE>	PRIORITE	Priorite d'usinage
<SWCP.LOT>	LOT	Numero de lot production

6.4 Bibliotheque d'outils et agregats

Les donnees de la bibliotheque SWOOD CAM sont stockees dans le dossier SWOODData. Ces donnees peuvent etre indexees dans la base .db pour consultation et analyse.

Dossier	Contenu	Champs extractibles
SWOODData/Tools/	Outils de coupe	Nom, type, reference, RPM, avance, plongee, profondeur max
SWOODData/Aggregates/	Agregats (blocs percage)	Nom, type, spindles, offsets, axes
SWOODData/Machinings/	Entites d'usinage	Operations, parametres, conditions
SWOODData/PostProcessors/	Post-processeurs	Machine cible, format de sortie

Chapitre 7

Formats d'Export SWOOD

7.1 CSV (System Report -- SWOOD 2024+)

Le System Report de SWOOD 2024 permet d'exporter toute vue tabulaire au format CSV. C'est le format le plus simple a importer dans une base de donnees SQLite.

- Export depuis n'importe quelle categorie de donnees (Panneaux, Quincaillerie, Programmes, etc.)
- Les vues personnalisees (colonnes, filtres, tri) sont preservees dans l'export
- Encodage compatible avec Python (csv.reader / pandas.read_csv)

```
import csv
import sqlite3

def import_swood_csv(csv_path, db_path, table_name):
"""Importe un CSV du System Report dans une table SQLite."""
conn = sqlite3.connect(db_path)
with open(csv_path, "r", encoding="utf-8-sig") as f:
reader = csv.DictReader(f, delimiter=";")
columns = reader.fieldnames

# Creer la table si necessaire
cols_def = ", ".join([f'"{c}" TEXT' for c in columns])
conn.execute(f"CREATE TABLE IF NOT EXISTS {table_name} ({cols_def})")

# Inserer les donnees
placeholders = ", ".join(["?"] * len(columns))
for row in reader:
values = [row[c] for c in columns]
conn.execute(f"INSERT INTO {table_name} VALUES ({placeholders})", values)

conn.commit()
conn.close()
```

7.2 Excel (SWOOD Excel Report)

Le rapport Excel utilise un template XML (ReportLists_to_Excel.xml) et un fichier Excel template (.xls). Chaque feuille est pilotee par un bloc <GENERIC> dans le XML.

Parametres XML	Quantite max	Description
Strings (S1..S30)	30	Variables texte
Integers (I1..I5)	5	Variables entieres
Doubles (D1..D5)	5	Variables decimales
Booleans (B1..B5)	5	Variables booleennes
Variants (V1..V5)	5	Variables generiques

Le fichier Excel genere contient typiquement les feuilles : Panels, Hardware, Programs, Summary, Edgebands, Messages. Chaque feuille peut etre lue en Python avec openpyxl.

7.3 XML (Template de donnees)

Le fichier **ReportLists_to_Excel.xml** situe dans SWOODData/DAT/Documents/XLS/ definit la structure de donnees exportee vers Excel. Il peut etre modifie pour ajouter des champs supplementaires ou genere dynamiquement par Python.

7.4 HTML (Rapport systeme)

Le rapport HTML est le format par defaut. Il utilise les fichiers CSS et JavaScript du dossier Ressources (SWOODReport.css, Languages.js, Materials.js, etc.). Les donnees sont integrees directement dans le HTML via les variables SWOOD.

Fichier	Role
Template.html	Structure HTML du rapport
SWOODReport.css	Styles visuels
Languages.js	Traductions (EN, FR, CN...)
Materials.js	Donnees materiaux pour affichage
CabDetails.js / ReportCabinet.js	Logique d'affichage des meubles
ReportNest.js	Donnees de nesting

Chapitre 8

Schema de Base de Donnees SQLite

8.1 Modele relationnel complet

Le schema propose organise les donnees en tables normalisees avec des relations cle primaire / cle etrangere. Il est concu pour stocker l'ensemble des donnees extractibles depuis SolidWorks et SWOOD.

8.2 Tables principales

Table : projets

```
CREATE TABLE projets (
id INTEGER PRIMARY KEY AUTOINCREMENT,
nom TEXT NOT NULL,
chemin TEXT,
devise TEXT DEFAULT 'EUR',
date_creation DATETIME DEFAULT CURRENT_TIMESTAMP,
date_modif DATETIME,
description TEXT
);
```

Table : documents

```
CREATE TABLE documents (
id INTEGER PRIMARY KEY AUTOINCREMENT,
projet_id INTEGER REFERENCES projets(id),
chemin TEXT NOT NULL UNIQUE,
titre TEXT,
type_doc INTEGER, -- 1=piece, 2=asm, 3=plan
materiau TEXT,
masse_kg REAL,
auteur TEXT,
date_creation DATETIME,
date_sauvegarde DATETIME,
date_extraction DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Table : configurations

```
CREATE TABLE configurations (
id INTEGER PRIMARY KEY AUTOINCREMENT,
document_id INTEGER REFERENCES documents(id),
nom TEXT NOT NULL,
est_active BOOLEAN DEFAULT 0
);
```

Table : proprietes_custom

```
CREATE TABLE proprietes_custom (
id INTEGER PRIMARY KEY AUTOINCREMENT,
configuration_id INTEGER REFERENCES configurations(id),
nom TEXT NOT NULL,
valeur_brute TEXT, -- Expression ($PRP:...)
valeur_resolue TEXT, -- Valeur evaluee
```

```
type_prop INTEGER, -- 30=text, 32=number, etc.
date_extraction DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Table : panneaux

```
CREATE TABLE panneaux (
id INTEGER PRIMARY KEY AUTOINCREMENT,
document_id INTEGER REFERENCES documents(id),
nom TEXT,
quantite INTEGER DEFAULT 1,
longueur_brute REAL, -- PAN_STL
largeur_brute REAL, -- PAN_STW
epaisseur_brute REAL, -- PAN_STT
longueur_finie REAL, -- PAN_FNL
largeur_finie REAL, -- PAN_FNW
epaisseur_finie REAL, -- PAN_FNT
code_materiau TEXT, -- CMAT
description TEXT,
est_debit_scie BOOLEAN DEFAULT 0,
direction_fil TEXT
);
```

Table : chants

```
CREATE TABLE chants (
id INTEGER PRIMARY KEY AUTOINCREMENT,
panneau_id INTEGER REFERENCES panneaux(id),
position TEXT NOT NULL, -- 'AR', 'AV', 'D', 'G'
code TEXT, -- CCHxx
epaisseur REAL, -- ECHxx
present BOOLEAN DEFAULT 0 -- CHxx
);
```

Table : composants_assemblage

```
CREATE TABLE composants_assemblage (
id INTEGER PRIMARY KEY AUTOINCREMENT,
assemblage_id INTEGER REFERENCES documents(id),
composant_id INTEGER REFERENCES documents(id),
nom_instance TEXT,
niveau INTEGER, -- Profondeur dans l'arbre
quantite INTEGER DEFAULT 1,
est_supprime BOOLEAN DEFAULT 0,
est_visible BOOLEAN DEFAULT 1
);
```

Table : quincaillerie

```
CREATE TABLE quincaillerie (
id INTEGER PRIMARY KEY AUTOINCREMENT,
projet_id INTEGER REFERENCES projets(id),
nom TEXT,
reference TEXT,
fournisseur TEXT,
cout_unitaire REAL,
quantite INTEGER DEFAULT 1,
description TEXT
);
```

Table : programmes_cnc

```
CREATE TABLE programmes_cnc (
id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
panneau_id INTEGER REFERENCES panneaux(id),
fichier TEXT, -- Chemin du fichier programme
machine TEXT, -- Post-processeur cible
phase TEXT, -- Designation de la phase
date_generation DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Table : outils

```
CREATE TABLE outils (
id INTEGER PRIMARY KEY AUTOINCREMENT,
nom TEXT,
type_outil TEXT,
reference TEXT,
numero INTEGER,
rpm REAL,
avance REAL, -- m/min
plongee REAL, -- m/min
profondeur_max REAL -- mm
);
```

8.3 Relations et cles etrangeres

```
projets 1---* documents
documents 1---* configurations
configurations 1---* proprietes_custom
documents 1---* panneaux
panneaux 1---* chants
panneaux 1---* programmes_cnc
documents *---* documents (via composants_assemblage)
projets 1---* quincaillerie
```

8.4 Index recommandes

```
CREATE INDEX idx_documents_chemin ON documents(chemin);
CREATE INDEX idx_documents_projet ON documents(projet_id);
CREATE INDEX idx_proprietes_config ON proprietes_custom(configuration_id);
CREATE INDEX idx_proprietes_nom ON proprietes_custom(nom);
CREATE INDEX idx_panneaux_doc ON panneaux(document_id);
CREATE INDEX idx_chants_panneau ON chants(panneau_id);
CREATE INDEX idx_composants_asm ON composants_assemblage(assemblage_id);
CREATE INDEX idx_programmes_panneau ON programmes_cnc(panneau_id);
```

Chapitre 9

Implementation Python

9.1 Connexion COM a SolidWorks

```
import win32com.client
import sqlite3
import os
from datetime import datetime

class SolidWorksBridge:
"""Interface de communication avec SolidWorks via COM."""

def __init__(self):
self.swApp = None
self.swModel = None

def connect(self):
"""Se connecte a l'instance SolidWorks active."""
try:
self.swApp = win32com.client.GetActiveObject("SldWorks.Application")
self.swModel = self.swApp.ActiveDoc
return True
except Exception as e:
print(f"Erreur connexion : {e}")
return False

def get_doc_info(self):
"""Retourne les informations du document actif."""
if not self.swModel:
return None
return {
"path": self.swModel.GetPathName(),
"title": self.swModel.GetTitle(),
"type": self.swModel.GetType(),
"material": self.swModel.MaterialUserName,
"author": self.swModel.SummaryInfo(3),
"save_date": self.swModel.SummaryInfo(8),
}
```

9.2 Extraction des proprietes custom

```
def get_all_properties(self):
"""Extrait toutes les proprietes de toutes les configurations."""
result = {}

# Proprietes globales (pas de config specifique)
result["_global"] = self._extract_props("")

# Proprietes par configuration
config_names = self.swModel.GetConfigurationNames()
if config_names:
for cname in config_names:
result[cname] = self._extract_props(cname)
```

```
return result

def _extract_props(self, config_name):
"""Extrait les proprietes d'une configuration."""
cpm = self.swModel.Extension.CustomPropertyManager(config_name)
props = {}
names = cpm.GetNames()
if names:
for name in names:
try:
result = cpm.Get5(name, False)
props[name] = {
"value": str(result[1]) if result[1] else "",
"resolved": str(result[2]) if result[2] else "",
}
except:
props[name] = {"value": "", "resolved": ""}
return props
```

9.3 Traversee recursive d'assemblage

```
def traverse_bom(self, max_depth=10):
"""Extrait la nomenclature complete d'un assemblage."""
if self.swModel.GetType() != 2: # Pas un assemblage
return []

config = self.swModel.GetActiveConfiguration()
root = config.GetRootComponent2(False)

components = []
self._walk_tree(root, components, 0, max_depth)
return components

def _walk_tree(self, comp, results, depth, max_depth):
"""Parcourt recursivement l'arbre de composants."""
if depth > max_depth:
return

children = comp.GetChildren()
if children is None:
return

for child in children:
model = child.GetModelDoc2()
entry = {
"depth": depth,
"name": child.Name2,
"path": model.GetPathName() if model else "",
"type": model.GetType() if model else 0,
"suppressed": child.GetSuppression2(),
"material": model.MaterialUserName if model else "",
"properties": {},
}
if model:
entry["properties"] = self._extract_props("")
results.append(entry)

if model and model.GetType() == 2:
self._walk_tree(child, results, depth + 1, max_depth)
```

9.4 Ecriture en base SQLite

```
class DatabaseManager:
"""Gestionnaire de la base de donnees SQLite."""

def __init__(self, db_path):
self.db_path = db_path
self.conn = sqlite3.connect(db_path)
self.conn.execute("PRAGMA foreign_keys = ON")

def save_document(self, doc_info, projet_id):
"""Enregistre un document dans la base."""
cursor = self.conn.execute(
"""INSERT OR REPLACE INTO documents
(projet_id, chemin, titre, type_doc, materiau, auteur)
VALUES (?, ?, ?, ?, ?, ?)""",
(projet_id, doc_info["path"], doc_info["title"],
doc_info["type"], doc_info["material"], doc_info["author"])
)
self.conn.commit()
return cursor.lastrowid

def save_properties(self, config_id, properties):
"""Enregistre les proprietes custom."""
for name, vals in properties.items():
self.conn.execute(
"""INSERT INTO proprietes_custom
(configuration_id, nom, valeur_brute, valeur_resolue)
VALUES (?, ?, ?, ?)""",
(config_id, name, vals["value"], vals["resolved"])
)
self.conn.commit()
```

9.5 Flux bidirectionnel

L'element le plus innovant de l'architecture est le flux retour : le logiciel interne peut **piloter SolidWorks/SWOOD** en ecrivant des proprietes custom qui seront lues automatiquement par SWOOD lors de la generation du rapport.

```
def write_back_property(self, sw_bridge, prop_name, prop_value):
"""Ecrit une propriete dans SolidWorks depuis la base .db."""
cpm = sw_bridge.swModel.Extension.CustomPropertyManager("")

# Supprimer puis recreer (pattern fiable)
cpm.Delete2(prop_name)
cpm.Add3(prop_name, 30, str(prop_value), 2)

# Reconstruire pour propager
sw_bridge.swModel.EditRebuild3()


# --- Exemple d'utilisation ---
bridge = SolidWorksBridge()
bridge.connect()
db = DatabaseManager("production.db")

# Extraction
doc_info = bridge.get_doc_info()
doc_id = db.save_document(doc_info, projet_id=1)

# Retour : marquer un panneau pour usinage
db.write_back_property(bridge, "GENERER_PROGRAMME", "Oui")
```

```
# SWOOD lira via <SWCP.GENERER_PROGRAMME>
```

Chapitre 10

SWOOD Connect -- Echanges Tiers

10.1 Connecteurs disponibles (SWOOD 2023+)

SWOOD Connect est un module d'echange de donnees avec des plateformes tierces introduit dans SWOOD 2023. Il fournit trois connecteurs natifs :

Connecteur	Direction	Format	Description
Blum E-SERVICE	Import	BXF	Import configurations quincaillerie Blum
HOMAG Tapio / Intellidivide	Export	Proprietaire	Export vers plateforme production HOMAG
Winner	Import	Proprietaire	Import conceptions cuisine Winner

10.2 Formats d'echange

Au-dela de SWOOD Connect, plusieurs formats permettent l'echange de donnees :

Format	Extension	Direction	Usage
CSV	.csv	Export	System Report -> Python/Excel/Base .db
Excel	.xls/.xlsx	Export	Rapport Excel detaille
XML	.xml	Import/Export	Templates de donnees, configuration
SWR / SWRZ	.swrz	Export	Partage rapport entre postes
HTML	.html	Export	Rapport navigateur
CFG	.cfg	Config	Fichiers de configuration SWOOD
JSON	.json	Config	Configuration rapport (Report.json)
CNC	variable	Export	Programmes machine (WOODWOP, BiesseWorks...)

Chapitre 11

Scripting SWOOD Design

11.1 Syntaxe VBScript SWOOD

SWOOD Design utilise un langage de script base sur VBScript pour la configuration parametrique des meubles (SWOODBox, connecteurs, SWOOD Center). Ce langage offre des proprietes speciales pour le controle de l'interface.

Propriete	Syntaxe	Description
.EXPOSE	VAR.EXPOSE = TRUE	Rend le parametre visible dans l'interface
.READONLY	VAR.READONLY = TRUE	Rend le parametre non modifiable
.POSSIBLEVALUES	VAR.POSSIBLEVALUES = "A;B;C"	Menu deroulant avec options
.VALID	VAR.VALID = False	Marque une erreur sur le parametre
.NAME	VAR.NAME	Retourne le nom du parametre (lecture seule)

Menus deroulants avec labels

```
'Format simple : valeur directe
SQ.POSSIBLEVALUES = "1;2;3;4;5;6"

'Format avec labels : valeur|Label
PP.POSSIBLEVALUES = "SIMP|Simple;COMP|Complexe"
ANGD.POSSIBLEVALUES = "0|0 degres;90|90 degres"
```

11.2 Variables contextuelles

Variable	Alias	Description
SBL	--	Longueur de la SwoodBox
SBH	--	Hauteur de la SwoodBox
SBP	--	Profondeur de la SwoodBox
SBLT / SBRT	--	Epaisseur panneau gauche / droit
SBTT / SBBTT	--	Epaisseur panneau haut / bas
SBFT (2024)	--	Epaisseur panneau avant
SBBKT (2024)	--	Epaisseur panneau arriere
SBLO / SBRO (2024)	--	Ecart panneau gauche / droit
SBTO / SBBTO (2024)	--	Ecart panneau haut / bas
SBFO / SBBKO (2024)	--	Ecart panneau avant / arriere

11.3 Fonctions disponibles (SWOOD 2024)

Fonction	Syntaxe	Description	Exemple
Min	Min(A, B)	Minimum de deux valeurs	Min(100, 200) = 100
Max	Max(A, B)	Maximum de deux valeurs	Max(100, 200) = 200
Floor	Floor(A, C)	Arrondi inferieur a C decimales	Floor(1.2345, 2) = 1.23
Ceil	Ceil(A, C)	Arrondi superieur a C decimales	Ceil(1.2345, 2) = 1.24

ASTUCE : Les fonctions sont imbricables : **Min(Min(A, B), C)** compare trois valeurs.

ORIGINOFFSET (SWOOD 2024)

```
'Deplacer une SwoodBox de 100mm en X, Y et Z
ORIGINOFFSET.X = 100
ORIGINOFFSET.Y = 100
ORIGINOFFSET.Z = 100
```

Chapitre 12

Bonnes Pratiques et Recommandations

12.1 Performances

- **Batch processing** : Ouvrir/fermer les fichiers un par un, ne pas tout charger en memoire
- **Lightweight mode** : Utiliser swModel.LargeAssemblyMode pour les gros assemblages
- **Transactions SQLite** : Utiliser des transactions (BEGIN/COMMIT) pour les insertions en masse
- **Eviter les rebuilds inutiles** : Ne reconstruire (EditRebuild3) qu'apres modification
- **Cache les proprietes** : Stocker les resultats en memoire avant d'ecrire en base

12.2 Gestion des erreurs

```
try:
swApp = win32com.client.GetActiveObject("SldWorks.Application")
except Exception:
print("SolidWorks n'est pas lance ou n'est pas accessible")
sys.exit(1)

swModel = swApp.ActiveDoc
if swModel is None:
print("Aucun document ouvert dans SolidWorks")
sys.exit(1)

# Verifier le type de document
doc_type = swModel.GetType()
if doc_type not in (1, 2, 3):
print(f"Type de document non supporte : {doc_type}")
sys.exit(1)
```

12.3 Securite des donnees

- **Sauvegardes regulieres** de la base .db (copies automatiques horodatees)
- **Lecture seule par defaut** : N'ecrire dans SolidWorks que sur action explicite
- **Journalisation** : Logger chaque modification dans un fichier de log
- **Validation des donnees** : Verifier les types et plages avant insertion en base
- **Acces concurrent** : SQLite supporte un seul ecrivain a la fois (WAL mode recommande)

```
# Activer le mode WAL pour de meilleures performances concurrentes
conn = sqlite3.connect("production.db")
conn.execute("PRAGMA journal_mode=WAL")
conn.execute("PRAGMA foreign_keys=ON")
```

Chapitre 13

Analyse du Report de travail.Cfg

Le fichier **Report de travail.Cfg** (2211 lignes) est le fichier de configuration central utilise par SWOOD Report pour generer l'ensemble des documents de sortie d'un projet. C'est le point d'integration le plus important entre SolidWorks/SWOOD et les systemes externes.

13.1 Structure globale du fichier

Le fichier est organise en 6 sections majeures, chacune controlant un aspect de la generation du rapport :

Section	Lignes	Role
(RAPPORT)	4-22	Configuration globale, chemins, activation CAM
[PROJECT/COMPONENT/PART/ASSEMBLY]	44-379	Variables par type d'objet
[TYPEDOBJECT]	383-674	Classification des pieces (quincaillerie, produits, etc.)
(SERIAL_NUMBERS)	678-725	Numerotation automatique des panneaux et produits
(MESSAGES)	730-772	Alertes et controle qualite
(DOCUMENTS)	775-2170	Generation de tous les fichiers de sortie
(MACHINES)	2183-2211	Configuration machine CNC (ROVER_K Biesse)

13.2 Section (RAPPORT) -- Configuration globale

Cette section definit les parametres globaux du rapport. Les elements les plus importants pour l'integration sont les chemins et le controle CAM.

```
Version = 2025.00
(RAPPORT)
REPORTPATH = Z:\Rapports\<SW.TITLE>
PROCESS_SWOODCAM_REPORT = 1
PROCESS_SWOODCAM_CONDITION = REPORT_CAM ; <-- CLE : controle CAM
PROGRAMSPATH = Z:\Programmes_usinages\<SW.TITLE>\<PROGMAC>
PROGRAMFILE = <TO_NUM_PANEL_EXT>-<DESC>-<PROG_NAME>
READMESHES = 1
USESWOODDESIGNDATA = 1
PROCESS_BOM = 1
GENERATE_CUTTING_PATTERN_CONDITION = "true"
CUTTING_PATTERN_CUTTER_THICKNESS = 4 ; Epaisseur lame scie
```

Parametre	Valeur actuelle	Impact
REPORTPATH	Z:\Rapports\<SW.TITLE>	Dossier de sortie du rapport (par nom de projet)
PROCESS_SWOODCAM_REPORT	1	Active la generation des programmes CNC
PROCESS_SWOODCAM_CONDITION	REPORT_CAM	Variable de controle : si True, le programme est genere

Parametre	Valeur actuelle	Impact
PROGRAMSPATH	Z:\Programmes_usinages\...	Dossier des programmes CNC generes
PROGRAMFILE	<NUM_PANEL>-<DESC>-<PROG>	Nommage des fichiers programmes
CUTTING_PATTERN_CUTTER_THICKNESS	4	Epaisseur de la lame de scie (mm) pour calepinage

IMPORTANT : La variable REPORT_CAM est lue depuis la propriete custom SolidWorks <SWCP.REPORT_CAM>. C'est le levier principal pour piloter la generation CNC depuis un logiciel externe comme DestriChiffrage.

13.3 Variables Projet et flags d'export

La section [PROJECT] definit les variables globales du rapport et les flags qui activent/desactivent chaque type de document de sortie.

Variable	Type	Valeur	Description
CLIENT	S	<SWCP.CLIENT>	Nom du client (lu depuis propriete SolidWorks)
REPORT_CAM	B	<SWCP.REPORT_CAM>	Active/desactive la generation CNC
PROJECT_QTY	I	<SWCP.PROJECT_QTY>	Multiplicateur de quantite projet
REPORT_SD	B	1	Generer donnees SWOOD Design (panneaux, chants)
REPORT_SC	B	1	Generer donnees SWOOD CAM (programmes)
REPORT_CSV	B	1	Generer les exports CSV
REPORT_EDRAWING	B	1	Generer le fichier eDrawing (.easm)
REPORT_DRAWING2PDF	B	1	Convertir les plans en PDF
REPORT_SERIALNUMBERS	B	1	Activer la numerotation automatique
REPORT_IMAGE	B	1	Generer les images (JPG)
REPORT_HTML	B	0	Rapport HTML (desactive)
REPORT_EXCEL	B	0	Rapport Excel (desactive)
REPORT_JSON	B	0	Export JSON (desactive)
REPORT_WEBGL	B	0	Visualisation 3D WebGL (desactive)
REPORT_3DPDF	B	0	PDF 3D (desactive)

ASTUCE : Les flags **REPORT_JSON** et **REPORT_EXCEL** sont actuellement desactives (= 0). Les activer permettrait de generer des fichiers exploitables directement par Python pour alimenter la base de donnees.

13.4 Classification TypedObject

La section [TYPEDOBJECT] est le **moteur de classification** de SWOOD. Elle definit comment chaque piece/assemblage est categorise (quincaillerie, produit, panneau...) et quelles variables sont extraites pour chaque categorie. C'est le coeur du lien avec un systeme de gestion externe.

TypedObject	Condition ISPART	Variables extraites	Lien DestriChiffrage
HA (Hardwares)	SWOOD_TYPE='HARDWARE' ou PATH contient 'Hardwares'	Code, Supplier, Supplier_Reference, Cost, FOURNISSEUR	produits (reference, fournisseur, prix_achat)
NOTHA	SWOOD_TYPE='NOT_HARDWARE'	SW.TITLE	Exclusion
TO_ASM_PARTS_LINKED	SWOOD_TYPE='ASM_PARTS_LINKED'	SW.TITLE	Assemblage lie
TO_CHILD_PART	SWOOD_TYPE='CHILD_PART'	SW.TITLE	Sous-piece
TO_MAIN_PART	SWOOD_TYPE='MAIN_PART'	SW.TITLE	Piece principale
TO_PRODUCT	SWOOD_TYPE='PRODUCT'	QTY, DESC, W, D, H, M, TYPE_CAISSON, FINISH, NOM_CAISSON	prix_marche (designation, quantite, dimensions)
TO_REPORT_EXCLUDED	SWOOD_TYPE='REPORT_EXCLUDED'	SW.TITLE	Piece exclue du rapport
TO_REPORT_PART	CoreStockMaterial!='' et pas CHILD/MAIN/EXCLUDED/HARDWARE	SW.TITLE, NUM_PANEL_EXT, TO_PRODUCT_NAME	Panneau standard

Detail du TypedObject HA (Quincaillerie)

Ce bloc est le plus important pour l'integration avec DestriChiffrage. Il extrait 5 proprietes custom de chaque quincaillerie :

```
[[HA
DESCRIPTION = Hardwares
ISPART = ((SWOOD_TYPE='HARDWARE') or (PATH CONTIENT 'Hardwares'))
and NOT(SWOOD_TYPE='NOT_HARDWARE')
COST = TO_Q_UC ; Cout = propriete Cost

[[VARIABLES
[[TO_Q_CODE ; --> catalogue.db: produits.reference
EVALUATION = <SWCP.Code>
]]
[[TO_Q_F ; --> catalogue.db: produits.fournisseur
EVALUATION = <SWCP.Supplier>
]]
[[TO_Q_R ; --> catalogue.db: produits.reference (alt)
EVALUATION = <SWCP.Supplier_Reference>
]]
[[TO_Q_UC ; --> catalogue.db: produits.prix_achat
TYPE = R
EVALUATION = <SWCP.Cost>
]]
[[TO_Q_FOURNISSEUR ; --> catalogue.db: produits.fournisseur
EVALUATION = <SWCP.FOURNISSEUR>
]]
]];VARIABLES
]];HA
```

Detail du TypedObject TO_PRODUCT (Meuble/Caisson)

```
[[TO_PRODUCT
DESCRIPTION = PRODUCT
ISPART = (SWOOD_TYPE="PRODUCT")
```

```
ISASSEMBLY = (SWOOD_TYPE="PRODUCT")
PROPAGATEVARIABLEONPART = 1
PROPAGATEVARIABLEONASSEMBLY = 1

[[VARIABLES
TO_PRODUCT_NAME = <SW.TITLE> ; Nom du meuble
TO_NUM_PRODUCT_EXT = <SWCP.NUM_PRODUCT_EXT> ; Reference numerotee
TO_PRODUCT_QTY = <SWCP.PRODUCT_QTY> ; Quantite
TO_PRODUCT_DESC = <SWCP.PRODUCT_DESC> ; Description
TO_PRODUCT_W = <SWCP.Width> ; Largeur
TO_PRODUCT_D = <SWCP.Depth> ; Profondeur
TO_PRODUCT_H = <SWCP.Height> ; Hauteur
TO_PRODUCT_M = <SWCP.Mass> ; Masse
TO_PRODUCT_INFO1 = <SWCP.TYPE_CAISSON> ; Type de caisson
TO_PRODUCT_FINISH = <SWCP.FINISH> ; Finition
TO_PRODUCT_NAME_CAISSON = <SWCP.NOM_CAISSON>; Nom du caisson
]];VARIABLES
]];TO_PRODUCT
```

13.5 Export CSV OptiPlanning

Le bloc **[OPTIPLANNING_EXPORT]** genere un fichier CSV de 16 colonnes a destination du logiciel d'optimisation de decoupe. Ce fichier constitue de facto un **bon de commande matiere** exploitable par DestriChiffrage.

Index	Champ CSV	Variable SWOOD	Equivalent DestriChiffrage
1	Projet	<CLIENT> (conditionnel)	chantiers.nom_client
2	Programme	<NUM_PANEL_EXT>-<DESC>-F1.cix	Reference programme CNC
3	Matiere	<ST_MATO.MAT_CUTREF>	produits.reference (materiau)
4	Quantite	<NB>	prix_marche.quantite
5	Longueur	<ST_L>	Dimension brute L
6	Largeur	<ST_W>	Dimension brute W
7	Chant avant	<PAN_EBF.EB_EBMATO.EBMAT_C>	Code chant fournisseur AV
8	Chant arriere	<PAN_EBB.EB_EBMATO.EBMAT_C>	Code chant fournisseur AR
9	Chant gauche	<PAN_EBL.EB_EBMATO.EBMAT_C>	Code chant fournisseur G
10	Chant droite	<PAN_EBR.EB_EBMATO.EBMAT_C>	Code chant fournisseur D
11	Reperes	<NUM_PANEL_EXT>	Reference panneau unique
12	Ensemble/caisson	<TO_PRODUCT_NAME_CAISSON>	Designation meuble
13	Type de piece	<DESC>	Designation piece
14	Epaisseur	<ST_T>	Epaisseur brute
15	Sens de fil	ST_MATO.MAT_WITHGRAIN	Oui/Non
16	Information surcote	<COMMENT>	Notes / commentaire

```
[OPTIPLANNING_EXPORT]
PROCESSON = REPORT
TYPEDOCUMENT = CSV
PATH = Z:\Opti5\Import file\<PROJNOM>.csv
```

```
CONDITION = "(REPORT_CSV)AND((REPORT_SD)OR(REPORT_SC))"
AUTOPROCESS = 1
[[CSV
LOOPON = STOCK
LOOPONCONDITION = "(ST_MATO.MAT_ISFORSAW)" ; Uniquement materiaux scie
NBEXPECTEDVALUES = 16
WITHHEADERS = 1
```

13.6 Numerotation automatique

Le systeme de numerotation (Serial Numbers) attribue automatiquement des reperes uniques aux panneaux et produits. C'est essentiel pour la tracabilite en production.

Compteur	Type	Cible	Format	Source SWCP
NUM_PANEL_EXT	SHARED	Panneau (TO_REPORT_PART)	Incrementiel (1, 2, 3...)	SWCP.NUM_PANEL_EXT
NUM_PRODUCT_EXT	SHARED	Produit (TO_PRODUCT)	Format 3 chiffres ({0:000})	SWCP.NUM_PRODUCT_EXT
NUM_CHILD_PART_INT	REPORT	Sous-piece (TO_CHILD_PART)	<NUM_PANEL_EXT>_{0:0}	--

ASTUCE : Les compteurs de type **SHARED** ecrivent automatiquement la valeur dans la propriete custom SolidWorks correspondante (SWCP_SOURCE). Ainsi, chaque panneau possede un repere unique lisible par Python via l'API COM.

13.7 Messages de controle qualite

Le systeme de messages genere des alertes automatiques lors de la generation du rapport. Ces messages peuvent etre exploites pour la validation qualite.

Message	Condition	Type	Description
DIMENSIONS_PART	Panneau hors limites du plateau	Erreur (-1)	Longueur ou largeur depasse les dimensions du plateau fournisseur
GRAIN_VERTICAL	ST_GRAINISVERTICAL = True	Erreur (-1)	Option grain vertical cochee par erreur
MINIMUM_LENGTH_EDGEBAND	EB_L < 80mm	Erreur (-1)	Chant trop court (min 80mm)
MINIMUM_WIDTH_SUSTAIN	PAN_STW < 50mm ou PAN_STL < 50mm	Erreur (-1)	Panneau trop petit pour la plaqueuse
PART_NOTTYPED	Ni panneau, ni quincaillerie	Erreur (-1)	Piece sans type specifique detecte

13.8 Documents generes

La section (DOCUMENTS) represente plus de 1400 lignes et configure la generation de tous les fichiers de sortie. Voici les principaux documents actifs :

Document	PROCESSON	Format	Chemin de sortie	Statut
eDrawing projet	REPORT	.easm	EDRAWING/<PROJNOM>.easm	Actif

Document	PROCESSON	Format	Chemin de sortie	Statut
CSV OptiPlanning	REPORT	.csv	Z:\Opti5\...<PROJNOM>.csv	Actif
Liste usinage	REPORT	.dat	Z:\Listes_usinages\...dat	Actif
PDF plans pieces	PART	.pdf	PDFS/<NOM>.pdf	Actif
PDF plans assemblages	ASSEMBLY	.pdf	PDFS/<NOM>.pdf	Actif
Images panneaux	PANEL	.jpg	IMAGE/<NOM>_PANELMAIN.jpg	Desactive
Images programmes	PROGRAM	.jpg	IMAGE/<NOM>_PROG.jpg	Desactive
Images produits	TYPEDOBJECT	.jpg	IMAGE/<TO_PRODUCT_NAME>.jpg	Desactive
HTML ReportDetails	REPORT	.html	<PROJNOM>_ReportDetails.html	Desactive
JSON ProductsData	REPORT	.json	Docs/ProductsData.json	Desactive
Excel rapport	REPORT	.xls	XLS/<PROJNOM>.xls	Desactive
Messages listing	REPORT	.html	Docs/MessagesListing.html	Actif

IMPORTANT : Le PostProcess de la liste d'usinage appelle un script batch nettoyage_fichier_export_liste_usinage.bat. Ce mecanisme de post-traitement peut etre exploite pour declencher automatiquement un script Python d'import dans la base .db.

13.9 Configuration machine CNC

```
(MACHINES)
[ROVER_K]
HOURCOST = 250 ; Cout horaire machine
CHECKEXTENSION = .cix ; Extension Biesse
PROGRAMSPATH = Z:\Programmes_usinages\<SW.TITLE>\<PROGMAC>
PROGRAMFILE = <TO_NUM_PANEL_EXT>-<DESC>-<PROG_NAME>
PROGRAMFILEIFEXIST = <TO_NUM_PANEL_EXT>-<DESC>-<PROG_NAME>-R

[[CONDITIONAL_PATH
[[REPORT_SERIALNUMBERS
PROGRAMFILE = <NUM_PANEL_EXT><DOC.NUMPROG>
CONDITION = "(REPORT_SERIALNUMBERS)AND(NOT(REPORT_SCN)"
]]
]]
```

La machine **ROVER_K** (Biesse) est configuree avec un cout horaire de 250 EUR/h. Le nommage des programmes utilise les reperes numerotes automatiquement. Ce cout horaire peut etre integre dans le calcul du cout de revient de DestriChiffrage.

Chapitre 14

Integration DestriChiffrage

14.1 Presentation de DestriChiffrage

DestriChiffrage v1.8.3 est l'application interne de chiffrage et approvisionnement developpee en Python/Tkinter avec une base de donnees SQLite (**catalogue.db**). Elle gere le catalogue produits, les devis (DPGF), les marches publics et les commandes fournisseur.

Composant	Fichier	Role
Application principale	src/main.py	Point d'entree Tkinter
Base de donnees	data/catalogue.db	Catalogue produits, chantiers, prix
Gestionnaire DB	src/database.py	ORM SQLite (~2600 lignes)
Configuration	src/config.py + config/settings.ini	Parametres globaux (Singleton)
Panier/Devis	src/cart_manager.py	Gestion devis rapide
Module DPGF	src/ui/dpgf_chiffrage_view.py	Chiffrage marches publics
Export	src/ui/cart_export_dialog.py	Export Excel/PDF

14.2 Schema de la base catalogue.db

La base catalogue.db contient les tables suivantes, directement exploitables pour le lien avec SolidWorks/SWOOD :

```
-- Table principale des produits
CREATE TABLE produits (
id INTEGER PRIMARY KEY AUTOINCREMENT,
categorie TEXT NOT NULL, -- Ex: 'Quincaillerie', 'Panneau'
sous_categorie TEXT, -- Ex: 'Charnieres', 'Coulisses'
designation TEXT NOT NULL, -- Nom du produit
dimensions TEXT, -- Dimensions texte
hauteur INTEGER, -- Hauteur en mm
largeur INTEGER, -- Largeur en mm
prix_achat REAL DEFAULT 0, -- Prix d'achat HT
reference TEXT, -- Reference fournisseur
fournisseur TEXT, -- Nom du fournisseur
marque TEXT, -- Marque du produit
chantier TEXT, -- Chantier associe
notes TEXT, -- Notes libres
fiche_technique TEXT, -- Chemin PDF fiche technique
devis_fournisseur TEXT, -- Chemin PDF devis
actif INTEGER DEFAULT 1 -- Produit actif/inactif
);

-- Table des chantiers / marches publics
CREATE TABLE chantiers (
id, nom, nom_client, type_marche, lieu, type_projet,
lot, montant_ht, marge_projet, resultat, ...
```

```
);

-- Table des articles DPGF (chiffrage)
CREATE TABLE prix_marche (
id, chantier_id, code, designation, description,
quantite, temps_conception, temps_fabrication, temps_pose,
cout_materiaux, cout_mo_total, cout_revient,
marge_pct, prix_unitaire_ht, prix_total_ht, ...
);

-- Table de liaison multi-produits par article
CREATE TABLE article_produits (
id, prix_marche_id, produit_id, quantite, prix_unitaire
);
```

14.3 Correspondances Report.cfg / catalogue.db

Voici la cartographie complete des correspondances entre les variables du Report.cfg, les proprietes custom SolidWorks et les champs de la base catalogue.db :

Quincaillerie (TypedObject HA)

Variable Report.cfg	Propriete SolidWorks	Table.champ catalogue.db	Direction
TO_Q_CODE	<SWCP.Code>	produits.reference	DestriChiffrage --> SW
TO_Q_F	<SWCP.Supplier>	produits.fournisseur	DestriChiffrage --> SW
TO_Q_R	<SWCP.Supplier_Reference>	produits.reference	DestriChiffrage --> SW
TO_Q_UC	<SWCP.Cost>	produits.prix_achat	DestriChiffrage --> SW
TO_Q_FOURNISSEUR	<SWCP.FOURNISSEUR>	produits.fournisseur	DestriChiffrage --> SW

Produits / Caissons (TypedObject TO_PRODUCT)

Variable Report.cfg	Propriete SolidWorks	Table.champ catalogue.db	Direction
TO_PRODUCT_NAME	<SW.TITLE>	prix_marche.designation	SW --> DestriChiffrage
TO_NUM_PRODUCT_EXT	<SWCP.NUM_PRODUCT_EXT>	prix_marche.code	DestriChiffrage --> SW
TO_PRODUCT_QTY	<SWCP.PRODUCT_QTY>	prix_marche.quantite	DestriChiffrage --> SW
TO_PRODUCT_DESC	<SWCP.PRODUCT_DESC>	prix_marche.description	DestriChiffrage --> SW
TO_PRODUCT_W / D / H	<SWCP.Width/Depth/Height>	prix_marche.largeur_mm / hauteur_mm	Bidirectionnel
TO_PRODUCT_M	<SWCP.Mass>	(calcule)	SW --> DestriChiffrage
TO_PRODUCT_INFO1	<SWCP.TYPE_CAISSON>	prix_marche.categorie	DestriChiffrage --> SW
TO_PRODUCT_FINISH	<SWCP.FINISH>	prix_marche.caracteristiques	Bidirectionnel

Variable Report.cfg	Propriete SolidWorks	Table.champ catalogue.db	Direction
TO_PRODUCT_NAME_CAISSON	<SWCP.NOM_CAISSON>	prix_marche.designation	DestriChiffrage --> SW

Projet / Controle

Variable Report.cfg	Propriete SolidWorks	Table.champ catalogue.db	Direction
CLIENT	<SWCP.CLIENT>	chantiers.nom_client	DestriChiffrage --> SW
REPORT_CAM	<SWCP.REPORT_CAM>	(decision logiciel)	DestriChiffrage --> SW
PROJECT_QTY	<SWCP.PROJECT_QTY>	prix_marche.quantite (mult.)	DestriChiffrage --> SW
DESC	<SWCP.DESCRIPTION>	produits.designation	Bidirectionnel
COMMENT	<SWCP.COMMENT>	produits.notes	Bidirectionnel
FINISH	<SWCP.FINISH>	prix_marche.caracteristiques	Bidirectionnel

14.4 Proprietes custom SolidWorks a standardiser

Pour que le pont fonctionne de maniere fiable, les proprietes personnalisees SolidWorks doivent etre normalisees dans tous les templates de pieces et assemblages. Voici la liste complete des proprietes a creer/verifier :

Propriete SolidWorks	Type	Source donnees	Obligatoire
CLIENT	Texte	chantiers.nom_client	Oui
Code	Texte	produits.reference	Oui (quincaillerie)
Supplier	Texte	produits.fournisseur	Oui (quincaillerie)
Supplier_Reference	Texte	produits.reference	Non
Cost	Reel	produits.prix_achat	Oui (quincaillerie)
FOURNISSEUR	Texte	produits.fournisseur	Oui
REPORT_CAM	Booleen	Decision logiciel	Oui
PROJECT_QTY	Entier	Multiplicateur projet	Non
PRODUCT_QTY	Entier	prix_marche.quantite	Oui (produit)
PRODUCT_DESC	Texte	prix_marche.description	Non
NOM_CAISSON	Texte	Designation meuble	Oui (produit)
TYPE_CAISSON	Texte	Type de meuble	Non
FINISH	Texte	Finition	Non
Width / Depth / Height	Reel	Dimensions meuble	Non
Mass	Reel	Masse calcule SW	Non
NUM_PRODUCT_EXT	Texte	Auto-numerote	Auto
NUM_PANEL_EXT	Texte	Auto-numerote	Auto

Propriete SolidWorks	Type	Source donnees	Obligatoire
SWOOD_TYPE	Texte	HARDWARE / PRODUCT / etc.	Oui
DESCRIPTION	Texte	produits.designation	Oui
COMMENT	Texte	produits.notes	Non

14.5 Flux de donnees bidirectionnel

```
DestriChiffrage (catalogue.db) SolidWorks + SWOOD
================================ ====================

produits.reference --------SWCP.Code----------> TypedObject HA
produits.fournisseur ------SWCP.Supplier------> TypedObject HA
produits.prix_achat -------SWCP.Cost----------> TypedObject HA
chantiers.nom_client ------SWCP.CLIENT--------> Variables Projet
prix_marche.quantite ------SWCP.PRODUCT_QTY---> TypedObject PRODUCT
prix_marche.designation ---SWCP.PRODUCT_DESC--> TypedObject PRODUCT
Decision "usiner" ---------SWCP.REPORT_CAM---> Condition CAM

<----SWCP.DESCRIPTION------ SW : Description piece
<----SWCP.Mass------------- SW : Masse calculee
<----CSV OptiPlanning------ Report : Nomenclature panneaux
<----Liste usinage (.dat)-- Report : Donnees programmes CNC
<----Messages (.html)------ Report : Alertes qualite
```

14.6 Module sw_bridge.py

Le module **sw_bridge.py** est le composant a developper pour realiser le pont. Il serait integre dans l'arborescence de DestriChiffrage et utiliserait l'API COM SolidWorks via pywin32.

```
# Architecture proposee pour sw_bridge.py

class SWBridge:
"""Pont entre DestriChiffrage et SolidWorks."""

def __init__(self, db: Database):
self.db = db # Instance Database de DestriChiffrage
self.swApp = None # Instance SolidWorks COM
self.swModel = None # Document actif

def connect(self) -> bool:
"""Connexion a SolidWorks."""
import win32com.client
self.swApp = win32com.client.GetActiveObject('SldWorks.Application')
self.swModel = self.swApp.ActiveDoc
return self.swModel is not None

# --- ECRITURE : DestriChiffrage --> SolidWorks ---

def push_hardware_data(self, produit_id: int):
"""Ecrit les donnees d'un produit catalogue dans la quincaillerie SW."""
produit = self.db.get_produit(produit_id)
self._set_prop('Code', produit['reference'])
self._set_prop('Supplier', produit['fournisseur'])
self._set_prop('Cost', str(produit['prix_achat']))
self._set_prop('FOURNISSEUR', produit['fournisseur'])

def push_project_data(self, chantier_id: int):
"""Ecrit les donnees du chantier dans les proprietes projet."""
```

```
chantier = self.db.get_chantier(chantier_id) # A implementer
self._set_prop('CLIENT', chantier['nom_client'])
self._set_prop('REPORT_CAM', 'Oui')

# --- LECTURE : SolidWorks --> DestriChiffrage ---

def pull_bom_hardware(self) -> list:
"""Extrait la liste de quincaillerie du modele actif."""
# Traverse l'assemblage et filtre SWOOD_TYPE='HARDWARE'
...

def import_optiplanning_csv(self, csv_path: str):
"""Importe un CSV OptiPlanning dans la base."""
# Parse le CSV 16 colonnes et cree les lignes dans prix_marche
...
```

Chapitre 15

Scenarios Concrets d'Utilisation

15.1 Commande fournisseur quincaillerie

Ce scenario permet de generer un bon de commande fournisseur a partir du modele SolidWorks en croisant les donnees avec le catalogue DestriChiffrage.

Etape	Action	Source	Destination
1	Ouvrir l'assemblage meuble dans SolidWorks	Utilisateur	SolidWorks
2	Lancer le script Python (sw_bridge)	DestriChiffrage	API COM
3	Traverser le BOM : identifier les composants SWOOD_TYPE='HARDWARE'	API COM	Script Python
4	Pour chaque quincaillerie, chercher dans catalogue.db par designation ou reference	Script	catalogue.db
5	Ecrire SWCP.Code, SWCP.Supplier, SWCP.Cost dans chaque composant SW	Script	API COM
6	Lancer SWOOD Report --> le CSV contient les references fournisseur correctes	Utilisateur	CSV
7	Importer le CSV dans le module de commande	Script	catalogue.db

```
# Exemple concret : pousser les donnees quincaillerie

bridge = SWBridge(db=Database())
bridge.connect()

# Extraire tous les composants de type HARDWARE
hardwares = bridge.pull_bom_hardware()

for hw in hardwares:
# Chercher dans le catalogue par designation
produit = db.search_produit(hw['name'])
if produit:
# Ecrire les proprietes dans SolidWorks
bridge.select_component(hw['component'])
bridge.push_hardware_data(produit['id'])
print(f" {hw['name']} --> {produit['reference']} ({produit['fournisseur']})")
else:
print(f" ATTENTION : {hw['name']} non trouve dans le catalogue")
```

15.2 Chiffrage automatique depuis le modele 3D

Ce scenario calcule automatiquement le cout de revient d'un meuble en croisant les donnees SolidWorks avec les prix du catalogue.

Etape	Action	Donnees
1	Extraire la liste des panneaux (SWOOD Design)	Materiau, dimensions, chants
2	Extraire la liste de quincaillerie (BOM)	Reference, quantite
3	Chercher les prix dans catalogue.db	prix_achat par produit
4	Calculer le cout matiere total	Somme (prix_achat x quantite)
5	Ajouter le cout machine (ROVER_K = 250 EUR/h)	Temps usinage x 250
6	Ajouter les temps MO (conception, fabrication, pose)	Taux horaires x temps
7	Calculer le prix de vente avec marge	Cout revient x (1 + marge%)
8	Comparer avec le prix_marche du chantier DPGF	Ecart budget/reel

15.3 Mise a jour des prix depuis le rapport SWOOD

Apres chaque generation de rapport SWOOD, le CSV OptiPlanning contient les references materiaux actualisees. Ce flux permet de maintenir les prix a jour dans DestriChiffrage.

```
import csv
from database import Database

def import_optiplanning_csv(csv_path: str, db: Database):
"""Importe le CSV OptiPlanning et met a jour les references."""
with open(csv_path, 'r', encoding='utf-8-sig') as f:
reader = csv.DictReader(f, delimiter=';')
for row in reader:
matiere_ref = row.get('Matiere', '')
if matiere_ref:
# Verifier si le materiau existe dans le catalogue
produit = db.search_by_reference(matiere_ref)
if not produit:
print(f'Nouveau materiau detecte : {matiere_ref}')
# Creer une entree dans le catalogue
db.add_produit(
categorie='Panneau',
designation=row.get('Type_de_piece_', 'Panneau'),
reference=matiere_ref,
dimensions=f"{row['Longueur']}x{row['Largeur']}x{row['Ep']}",
fournisseur='A renseigner',
)
```

15.4 Passage de commande matiere (panneaux)

Le CSV OptiPlanning genere par le Report.cfg contient toutes les informations necessaires pour passer une commande de panneaux au fournisseur (Dispano, etc.). Le flux est le suivant :

- **1.** SWOOD Report genere le CSV dans **Z:\Opti5\Import file**
- **2.** Le script Python lit le CSV et regroupe par matiere (reference fournisseur)
- **3.** Pour chaque matiere, il calcule la surface totale et le nombre de plateaux
- **4.** Il genere un bon de commande avec les references Dispano du catalogue
- **5.** Les chants sont listes separement avec leurs codes fournisseur
- **6.** Le bon de commande est exporte en Excel via openpyxl

```
# Regroupement des panneaux par matiere pour commande
from collections import defaultdict

def generer_commande_panneaux(csv_path: str):
matieres = defaultdict(lambda: {'panneaux': [], 'surface': 0})
chants = defaultdict(int) # code_chant -> longueur totale

with open(csv_path, 'r') as f:
reader = csv.DictReader(f, delimiter=';')
for row in reader:
ref = row['Matiere']
qty = int(row['Qu'])
l, w = float(row['Longueur']), float(row['Largeur'])
surface = (l * w * qty) / 1_000_000 # en m2
matieres[ref]['panneaux'].append(row)
matieres[ref]['surface'] += surface

# Collecter les chants
for pos in ['avant', 'arriere', 'gauche', 'droite']:
code = row.get(f'Chant_{pos}', '')
if code:
longueur = l if pos in ('avant','arriere') else w
chants[code] += longueur * qty

return matieres, chants
```

15.5 Etat d'avancement et feuille de route

Element	Statut	Detail
Proprietes custom quincaillerie dans Report.cfg	Operationnel	Code, Supplier, Cost, FOURNISSEUR
Export CSV OptiPlanning (16 champs)	Operationnel	Genere dans Z:\Opti5\
Classification TypedObject (HA, PRODUCT...)	Operationnel	6 types configures
Numerotation automatique (Serial Numbers)	Operationnel	NUM_PANEL_EXT, NUM_PRODUCT_EXT
Generation eDrawing + PDF plans	Operationnel	REPORT_EDRAWING=1, REPORT_DRAWING2PDF=1
Controle CAM conditionnel	Operationnel	PROCESS_SWOODCAM_CONDITION=REPORT_CAM
DestriChiffrage catalogue.db	Operationnel	Produits, chantiers, DPGF
Module sw_bridge.py (pont API)	Developpe	Connexion COM Python <-> SolidWorks (voir Ch.16)
Ecriture auto proprietes SWCP	Developpe	sync_db_to_solidworks() (voir Ch.16.6)
Lecture BOM --> import DestriChiffrage	Developpe	traverse_assembly() + sync_hardware_to_db()
Generation bon de commande fournisseur	Developpe	generate_supplier_order() CSV par fournisseur
Activation REPORT_JSON	A evaluer	Donnerait un format riche pour l'import
PostProcess Python auto	A evaluer	Script Python declenche apres rapport

Chapitre 16

Module sw_bridge.py -- Implementation Complete

Ce chapitre documente le module **sw_bridge.py**, developpe et teste avec succes. Ce module constitue le pont operationnel entre l'API COM SolidWorks et la base de donnees **catalogue.db** de DestriChiffrage. Il permet la synchronisation bidirectionnelle des proprietes personnalisees, la generation de commandes fournisseur, et fonctionne egalement en mode hors-ligne via les CSV du Report SWOOD.

Le module est situe dans `DestriChiffrage/src/sw_bridge.py` et un script de demonstration complet est disponible dans `sw_bridge_demo.py`.

16.1 Architecture du module

Le module est organise autour d'une classe principale **SolidWorksBridge** et de fonctions utilitaires autonomes. Voici la structure :

```
sw_bridge.py
|
|-- CONSTANTES
| |-- SWCP_MAPPING # Mapping proprietes SW <-> champs BDD
| |-- SWCP_PRODUCT_MAPPING # Mapping pour les produits/panneaux
| |-- SWOOD_TYPES # Types SWOOD reconnus
| |-- DEFAULT_HARDWARE_CATEGORY # 'QUINCAILLERIE SWOOD'
|
|-- CLASSE SolidWorksBridge
| |-- connect_solidworks() # Connexion COM
| |-- get/set_custom_property() # Lecture/ecriture proprietes
| |-- get_all_custom_properties() # Lecture complete
| |-- traverse_assembly() # Parcours arbre assemblage
| |-- sync_hardware_to_db() # SW -> catalogue.db
| |-- sync_db_to_solidworks() # catalogue.db -> SW
| |-- generate_supplier_order() # Commande fournisseur CSV
| |-- import_from_report_csv() # Mode hors-ligne
| |-- export_db_for_swood() # Export BDD -> SWOOD
|
|-- FONCTIONS UTILITAIRES
| |-- quick_scan_assembly() # Scan rapide en une ligne
| |-- sync_from_csv() # Sync CSV sans SolidWorks
|
|-- CLI (point d'entree __main__)
|-- scan / sync / export / status / order
```

Le mapping central entre les proprietes SolidWorks (SWCP) et les champs de catalogue.db est defini par le dictionnaire **SWCP_MAPPING** :

Propriete SolidWorks (SWCP)	Champ catalogue.db	Type	Direction
Code	reference	Texte (S)	SW -> BDD
Supplier	fournisseur	Texte (S)	SW <-> BDD
Supplier_Reference	reference	Texte (S)	SW <-> BDD

Propriete SolidWorks (SWCP)	Champ catalogue.db	Type	Direction
Cost	prix_achat	Reel (R)	BDD -> SW
FOURNISSEUR	fournisseur	Texte (S)	BDD -> SW
DESCRIPTION	designation	Texte (S)	SW <-> BDD
FINISH	notes	Texte (S)	SW -> BDD

16.2 Connexion COM a SolidWorks

La connexion utilise **win32com.client.Dispatch** pour se connecter a l'instance SolidWorks en cours d'execution. Aucun lancement automatique n'est effectue -- SolidWorks doit etre ouvert avant d'utiliser le bridge.

```
from sw_bridge import SolidWorksBridge
from database import Database

# Initialisation
db = Database()
bridge = SolidWorksBridge(database=db)

# Connexion a SolidWorks
if bridge.connect_solidworks():
doc = bridge.get_active_document()
print(f'Document actif: {doc}')
else:
print('SolidWorks non disponible')
```

En interne, la methode utilise `win32com.client.Dispatch("SldWorks.Application")` qui se connecte au ROT (Running Object Table) de Windows. Si le module **pywin32** n'est pas installe, un message d'erreur explicite est ajoute au journal.

16.3 Lecture et ecriture des proprietes personnalisees

Le bridge expose trois methodes pour manipuler les proprietes personnalisees SolidWorks (Custom Properties) :

Lecture d'une propriete

```
# Lire le code d'une quincaillerie
code = bridge.get_custom_property('Code')
prix = bridge.get_custom_property('Cost')
fournisseur = bridge.get_custom_property('Supplier')

# Lire depuis une configuration specifique
code_config = bridge.get_custom_property('Code', config_name='Config_A')
```

En interne, la methode utilise `CustomPropertyManager.Get5()` qui retourne la valeur resolue (evaluee) de la propriete. L'index 2 du tuple retourne contient la valeur apres evaluation des formules et liens.

Ecriture d'une propriete

```
# Ecrire des proprietes dans le document actif
bridge.set_custom_property('Code', 'BLUM-71B3550')
bridge.set_custom_property('Supplier', 'BLUM')
bridge.set_custom_property('Cost', '2.85', value_type=3) # 3=nombre
bridge.set_custom_property('FOURNISSEUR', 'BLUM')

# Types disponibles: 30=texte, 3=nombre, 11=oui/non, 64=date
```

L'ecriture utilise `CustomPropertyManager.Add3(name, type, value, 1)`. Le parametre **1** (OverwriteExisting) garantit que la propriete est creee si absente ou mise a jour si elle existe deja.

Lecture de toutes les proprietes

```
# Recuperer toutes les proprietes d'un composant
props = bridge.get_all_custom_properties()
for name, value in props.items():
print(f' {name} = {value}')

# Resultat typique pour une quincaillerie SWOOD :
# Code = BLUM-71B3550
# Supplier = BLUM
# Supplier_Reference = 71B3550
# Cost = 2.85
# SWOOD_TYPE = HARDWARE
# DESCRIPTION = Charniere CLIP top 110 degres
```

16.4 Traversee d'assemblage et extraction quincailleries

La methode **traverse_assembly()** reproduit en Python ce que fait le Report SWOOD avec la classification TypedObject HA. Elle parcourt recursivement l'arbre d'assemblage SolidWorks et identifie les quincailleries via deux criteres :

- **SWOOD_TYPE = 'HARDWARE'** dans les proprietes personnalisees
- Le chemin du composant contient **'Hardwares'** (convention SWOOD)
- Le composant n'a **pas** SWOOD_TYPE = 'NOT_HARDWARE' (exclusion)

```
# Traverser l'assemblage et extraire les quincailleries
components = bridge.traverse_assembly(include_hardware_only=True)

for comp in components:
print(f"[{comp['code']}] {comp['description']}")
print(f" Fournisseur: {comp['supplier']}")
print(f" Cout: {comp['cost']:.2f} EUR")
print(f" Quantite: {comp['quantity']}")
```

Chaque composant retourne contient les champs suivants :

Champ	Type	Description	Source
name	str	Nom du composant dans l'arbre	IComponent2.Name2
path	str	Chemin du fichier SLDPRT	IComponent2.GetPathName()
is_hardware	bool	Quincaillerie identifiee	Analyse SWOOD_TYPE + Path
swood_type	str	Type SWOOD (HARDWARE, PANEL...)	SWCP.SWOOD_TYPE
code	str	Code/reference du produit	SWCP.Code
supplier	str	Nom du fournisseur	SWCP.Supplier ou SWCP.FOURNISSEUR
supplier_reference	str	Reference fournisseur	SWCP.Supplier_Reference
cost	float	Cout unitaire HT	SWCP.Cost
description	str	Description du composant	SWCP.DESCRIPTION
finish	str	Finition	SWCP.FINISH
quantity	int	Nombre d'instances	Comptage assemblage

Champ	Type	Description	Source
properties	dict	Toutes les proprietes brutes	CustomPropertyManager

La traversee est recursive : les sous-assemblages sont explores en profondeur. Le champ **depth** indique le niveau de recursion (0 = composant direct de la racine).

16.5 Synchronisation SolidWorks vers catalogue.db

La methode **sync_hardware_to_db()** prend la liste des composants extraits par traverse_assembly() et les synchronise avec la base de donnees. Pour chaque quincaillerie :

- 1. Recherche dans catalogue.db par **reference** (correspondance exacte avec le Code SWCP)
- 2. Si le produit existe et que le prix differe : **mise a jour du prix** (avec historique)
- 3. Si le produit n'existe pas : **creation** dans la categorie QUINCAILLERIE SWOOD

```
# Synchroniser les quincailleries vers la BDD
components = bridge.traverse_assembly(include_hardware_only=True)
stats = bridge.sync_hardware_to_db(
components=components,
create_if_missing=True, # Creer les produits absents
update_prices=True # Mettre a jour les prix
)

print(f'Crees: {stats["created"]}')
print(f'Mis a jour: {stats["updated"]}')
print(f'Ignores: {stats["skipped"]}')
print(f'Erreurs: {stats["errors"]}')
```

Les produits crees sont places dans la categorie **QUINCAILLERIE SWOOD** par defaut. La sous-categorie est remplie avec la finition (FINISH) du composant si disponible. Chaque creation/modification est tracee dans le journal interne du bridge.

IMPORTANT : La mise a jour des prix declenche automatiquement l'enregistrement dans la table historique_prix de catalogue.db, permettant de tracer l'evolution des couts dans le temps.

16.6 Synchronisation catalogue.db vers SolidWorks

La methode **sync_db_to_solidworks()** effectue l'operation inverse : elle lit les donnees de catalogue.db et les ecrit dans les proprietes personnalisees SolidWorks. C'est le cas d'usage principal pour repercuter les prix a jour et les references fournisseur dans le modele 3D.

```
# Recuperer les prix a jour depuis la BDD et les ecrire dans SolidWorks
stats = bridge.sync_db_to_solidworks(
properties_to_update=['Supplier', 'Cost', 'Supplier_Reference', 'FOURNISSEUR']
)

# Par defaut, les proprietes suivantes sont ecrites :
# Code, Supplier, Supplier_Reference, Cost, FOURNISSEUR

# On peut aussi inclure DESCRIPTION pour mettre a jour les libelles :
stats = bridge.sync_db_to_solidworks(
properties_to_update=['Code', 'Supplier', 'Cost', 'DESCRIPTION']
)
```

Le processus pour chaque composant :

- 1. Lecture du **Code** dans les proprietes du composant SolidWorks
- 2. Recherche dans catalogue.db par **reference** (correspondance exacte)
- 3. Si trouve : ecriture des proprietes selectionnees via **CustomPropertyManager.Add3()**

- 4. Si non trouve : signalement dans le journal (not_found)

Cette methode necessite que SolidWorks soit ouvert avec un assemblage actif. Elle ouvre chaque composant individuellement pour ecrire ses proprietes, ce qui peut prendre quelques secondes par composant.

16.7 Generation de commande fournisseur

La methode **generate_supplier_order()** cree un fichier CSV de commande fournisseur a partir des quincailleries de l'assemblage. Les composants sont regroupes par fournisseur, les quantites sont cumulees, et les prix sont enrichis depuis catalogue.db si disponible.

```
# Generer une commande fournisseur
path = bridge.generate_supplier_order(
group_by_supplier=True # Regrouper par fournisseur
)
print(f'Commande generee: {path}')
```

Le fichier CSV genere contient les colonnes suivantes :

Colonne	Description	Exemple
Fournisseur	Nom du fournisseur	BLUM
Reference	Code produit	BLUM-71B3550
Designation	Description du produit	Charniere CLIP top 110
Quantite	Nombre total cumule	4
Prix Unitaire HT	Prix unitaire depuis BDD	2.85
Prix Total HT	Quantite x Prix unitaire	11.40

Le delimiteur est le point-virgule (;), l'encodage est UTF-8 avec BOM pour la compatibilite avec Excel. Un total general est ajoute en fin de fichier. Les produits sont tries par fournisseur puis par reference.

ASTUCE : Le fichier genere est pret a etre envoye aux fournisseurs ou importe dans un ERP. La methode **_enrich_from_db()** complete automatiquement les prix manquants dans l'assemblage avec les donnees de catalogue.db.

16.8 Mode hors-ligne (import CSV Report SWOOD)

Quand SolidWorks n'est pas disponible (poste bureau, serveur, etc.), le bridge peut fonctionner en **mode hors-ligne** en important les donnees depuis un CSV genere par le Report SWOOD.

```
# Importer depuis un CSV de Report SWOOD
components = bridge.import_from_report_csv('export_swood.csv')

# Le CSV peut utiliser les noms de variables du Report.cfg :
# Code | Supplier | Cost | Description | Quantity
# Ou les noms internes :
# TO_Q_CODE | TO_Q_F | TO_Q_UC | DESIGNATION | QTY

# Puis synchroniser normalement
stats = bridge.sync_hardware_to_db(components)
```

La methode detecte automatiquement le delimiteur (;, tabulation ou virgule) et l'encodage (UTF-8-sig). Elle accepte les deux conventions de nommage : les noms de proprietes SolidWorks (Code, Supplier, Cost) ou les noms de variables du Report.cfg (TO_Q_CODE, TO_Q_F, TO_Q_UC).

IMPORTANT : Ce mode permet de synchroniser la base de donnees sans avoir besoin de SolidWorks installe sur la machine. Il suffit d'avoir le fichier CSV genere lors du dernier rapport SWOOD.

16.9 Export BDD vers format SWOOD

La methode **export_db_for_swood()** exporte les produits de catalogue.db dans un format directement exploitable par SolidWorks. Deux formats sont disponibles :

Export CSV

```
# Export CSV (delimiter ; compatible Excel/SWOOD)
path = bridge.export_db_for_swood(
categorie='QUINCAILLERIE SWOOD',
format_type='csv'
)
# Genere: Code;Supplier;Supplier_Reference;Cost;FOURNISSEUR;
# DESCRIPTION;Marque;Categorie;Sous_Categorie
```

Export JSON

```
# Export JSON (exploitable par macro VBA ou script Python)
path = bridge.export_db_for_swood(
categorie='QUINCAILLERIE SWOOD',
format_type='json'
)
# Genere un tableau JSON avec les memes champs
```

Ces fichiers peuvent ensuite etre utilises pour alimenter les proprietes personnalisees SolidWorks via une macro VBA, un script Python autonome, ou integres dans le workflow Outil_Material_Import.xlsm existant.

16.10 Interface en ligne de commande (CLI)

Le module sw_bridge.py integre un point d'entree CLI permettant d'effectuer toutes les operations depuis un terminal ou un script batch :

Commande	Description	Exemple
scan	Scanne l'assemblage SolidWorks actif	python sw_bridge.py scan
sync	Synchronise un CSV vers la BDD	python sw_bridge.py sync export.csv
export [csv|json]	Exporte la BDD en format SWOOD	python sw_bridge.py export json
status	Affiche l'etat de connexion	python sw_bridge.py status
order	Genere une commande fournisseur	python sw_bridge.py order

```
# Exemples d'utilisation CLI

# 1. Scanner l'assemblage ouvert dans SolidWorks
python sw_bridge.py scan
# -> Affiche les quincailleries trouvees et leur presence en BDD

# 2. Importer un CSV de Report SWOOD
python sw_bridge.py sync rapport_quincailleries.csv
# -> Cree/met a jour les produits dans catalogue.db

# 3. Exporter pour SWOOD
python sw_bridge.py export csv
# -> Genere un CSV dans Documents/Export_SWOOD_YYYYMMDD.csv

# 4. Verifier l'etat
python sw_bridge.py status
# -> SolidWorks connecte: OUI/NON, Document actif, Produits en BDD
```

ASTUCE : La commande **scan** peut etre integree dans le PostProcess du Report SWOOD pour synchroniser automatiquement la BDD apres chaque generation de rapport.

16.11 Resultats de la demonstration

Le script **sw_bridge_demo.py** a ete execute avec succes en mode simulation (sans SolidWorks). Voici les resultats obtenus :

Demo	Description	Resultat
Demo 1	Sync 6 quincailleries SW -> BDD	6 produits crees avec succes
Demo 2	Mise a jour prix BDD -> SW	Prix BLUM-71B3550 mis a jour (+5%)
Demo 3	Commande fournisseur CSV	79.52 EUR HT, 6 refs, 4 fournisseurs
Demo 4	Import CSV Report SWOOD	3 produits importes (GRASS, HAFELE, HETTICH)
Demo 5	Export BDD -> CSV + JSON	9 produits exportes en 2 formats
Demo 6	Mapping Report.cfg complet	12 correspondances documentees

Exemple de commande fournisseur generee par la Demo 3 (extrait) :

```
Fournisseur;Reference;Designation;Quantite;Prix Unitaire HT;Prix Total HT
BLUM;BLUM-560H5500B;Coulisse TANDEM 550mm 30kg;2;18.50;37.00
BLUM;BLUM-71B3550;Charniere CLIP top 110;4;2.85;11.40
BLUM;BLUM-973A0500;Amortisseur BLUMOTION;4;3.40;13.60
DIVERS;DIVERS-TOUR0835;Tourillon bois 8x35mm;24;0.02;0.48
HETTICH;HETT-9103362;Poignee ProDecor 128mm;3;5.20;15.60
SPAX;SPAX-0251010400305;Vis SPAX 4x30mm;48;0.03;1.44
;;TOTAL GENERAL;;;79.52
```

Les 16 operations ont ete logguees dans le journal interne du bridge. Tous les tests sont passes sans erreur. Le module est pret pour l'integration avec SolidWorks en environnement reel.

> *Pour utiliser le bridge en mode reel, il suffit de remplacer les donnees simulees par un appel a* `bridge.connect_solidworks()` *suivi de* `bridge.traverse_assembly()`*. Le reste du code reste identique.*

Chapitre

Annexes -- References et Ressources

A. Documents source analyses

Document	Pages	Contenu principal
SolidWorks API Series 1	268	API COM, interfaces, proprietes custom, BOM
SolidWorks API Advanced	246	Add-ins, events, PMP, deploiement
SWOOD Report (pdfcoffee)	82	Variables SWOOD completes (reference)
SWOOD Report Config Manual	34	Configuration rapport, objets, traduction
SWOOD Excel Report Generic	11	Template XML/Excel, parametres export
System Report (Eficad)	1	Fonctionnalites System Report 2024
How to Conditional Generate	4	Generation conditionnelle CAM
SWOOD CAM Training FR	190	Outils, agregats, entites, programmation
Swood CAM Aide	98	Reference detaillee entites automatiques
What's New 2021-2024	4 docs	Nouveautes par version

B. Prerequis logiciels

Logiciel	Version min.	Module Python
SolidWorks	2020+	--
SWOOD Design + CAM	2023+	--
Python	3.10+	--
pywin32	306+	pip install pywin32
openpyxl (optionnel)	3.1+	pip install openpyxl
SQLAlchemy (optionnel)	2.0+	pip install sqlalchemy
pandas (optionnel)	2.0+	pip install pandas

C. Arborescence SWOODData

```
SWOODData/
DAT/
Report.cfg # Configuration rapport
Report.json # Configuration JSON
SWOODDesign.cfg # Configuration SWOOD Design
Documents/
Ressources/ # CSS, JS, images du rapport HTML
```

```
Template/ # Templates HTML
XLS/ # Templates Excel + XML
XML/ # Fichiers XML de donnees
JSON/ # Fichiers JSON
WEB/ # Rapport Web
...
Tools/ # Bibliotheque d'outils
Aggregates/ # Agregats (blocs percage)
Machinings/ # Entites d'usinage sauvegardees
PostProcessors/ # Post-processeurs machines
Counters/ # Compteurs XML (numerotation)
SWForm/ # Formulaires de proprietes
```

D. Constantes API frequentes

Constante	Valeur	Contexte
swDocPART	1	Type de document
swDocASSEMBLY	2	Type de document
swDocDRAWING	3	Type de document
swCustomInfoText	30	Type de propriete custom
swCustomInfoNumber	32	Type de propriete custom
swCustomInfoDouble	5	Type de propriete custom
swCustomInfoYesOrNo	11	Type de propriete custom
swCustomInfoDate	64	Type de propriete custom
swSaveAsCurrentVersion	0	Option de sauvegarde
swSaveAsOptions_Silent	2	Option de sauvegarde
swThisConfiguration	1	Option de configuration
swAllConfiguration	2	Option de configuration
swComponentSuppressed	0	Etat composant
swComponentFullyResolved	2	Etat composant
swComponentLightweight	3	Etat composant

Fin du document -- Guide d'Integration SolidWorks/SWOOD API v1.0

Genere automatiquement le 11/02/2026