Project Online to Smartsheet Migration Guide

Migration Tool Documentation

December 2024

Contents

1 Migrating from Project Online to Smartsheet

2 Migrating from Project Online to Smartsheet

Last Updated: 2024-12-08

2.1 Overview

If you're using Microsoft Project Online and evaluating migration options, this guide explains how to move your project data to Smartsheet. The migration process preserves your project structure, tasks, resources, and assignments while transitioning to a modern work management platform.

2.1.1 What This Tool Does

The migration tool: - Connects to your existing Project Online environment - Extracts all your project data including tasks, resources, and assignments - Converts the data to work in Smartsheet's structure - Creates organized workspaces in Smartsheet with your projects - Maintains all relationships between tasks, resources, and assignments - Handles errors and can resume if interrupted

2.1.2 Migration Structure

Each of your Project Online projects becomes a dedicated Smartsheet workspace:

```
Your Project Online Project "Website Redesign Q1"
    ↓ MIGRATION ↓
Smartsheet Workspace "Website Redesign Q1"
├── Sheet: Website Redesign Q1 - Summary (project overview)
├── Sheet: Website Redesign Q1 - Tasks (your task list with hierarchy)
└── Sheet: Website Redesign Q1 - Resources (your team members and resou
```

2.1.3 Key Features

1. **One-to-One Project Mapping**: Each Project Online project becomes its own Smartsheet workspace for clear organization
2. **Name Preservation**: Your workspace names match your Project Online project names
3. **Embedded Assignments**: Team member assignments appear directly in your task list
4. **Centralized Standards**: Status and priority values are managed centrally across all your projects

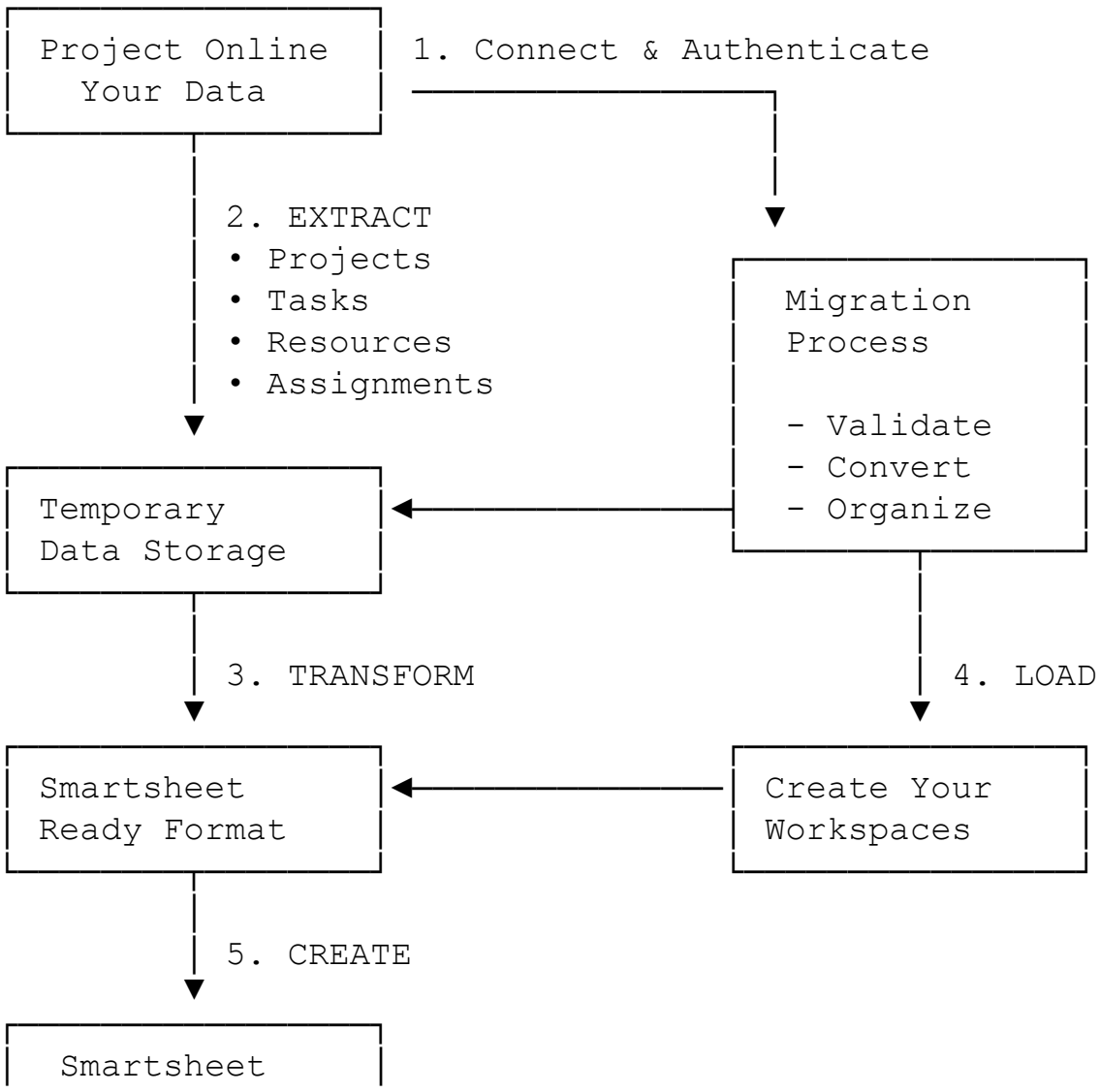
2.2 How Your Data Transforms

Your Project Online Data	Becomes in Smartsheet	How It Works
Project	Workspace + Summary Sheet	Each project gets its own dedicated workspace

Your Project Online Data	Becomes in Smartsheet	How It Works
Task	Row in Tasks Sheet	Your task hierarchy is preserved with parent-child relationships
Resource	Row in Resources Sheet	Your team members and resources with their information
Assignment	Column in Tasks Sheet	Who's assigned to each task appears right in the task list

2.2.1 Data Flow

The migration follows these steps:



```
|   - Workspaces    |
|   - Sheets        |
|   - Tasks         |
|   - Relationships|
└──────────────────┘
```

2.3 Technical Foundation

The migration tool uses:

For Connecting to Project Online: - Microsoft's authentication system (OAuth 2.0) - Secure token-based access - Automatic pagination for large datasets

For Creating in Smartsheet: - Smartsheet's official software development kit - Batch operations for efficiency - Automatic retry logic for reliability

2.3.1 What You'll Need

To run the migration, you'll need:

1. **Project Online Access**:
   - Azure Active Directory tenant information
   - Application credentials for secure access
   - Your Project Online site URL
2. **Smartsheet Access**:
   - A Smartsheet account with workspace creation permissions
   - API access token
3. **Configuration**:
   - A configuration file (`.env`) with your credentials
   - Node.js installed on your computer (version 18 or newer)

2.3.2 Basic Usage

```
# Preview the migration without making changes
npm start -- import --source <your-project-id> --destination <worksp

# Validate your Project Online connection
npm start -- validate --source <your-project-id>

# Run the actual migration
npm start -- import --source <your-project-id> --destination <worksp

# See detailed progress information
npm start -- import --source <your-project-id> --destination <worksp
```

2.4 What Gets Migrated

The tool migrates all essential project data:

2.4.1 Project Information

- Project name, description, and status
- Start and finish dates
- Priority levels
- Project owner information
- Completion percentage

2.4.2 Tasks

- All tasks with their hierarchical structure
- Task names, descriptions, and notes
- Start dates, end dates, and durations
- Status and priority for each task
- Dependencies between tasks (predecessors)
- Milestones
- Constraint types and dates
- Work hours (planned and actual)

2.4.3 Resources

- Team member names and contact information
- Resource types (people, equipment, costs)
- Rates and availability
- Department assignments
- Active status

2.4.4 Assignments

- Which resources are assigned to which tasks
- Assignment types properly distinguished
- Team member assignments enable collaboration features

2.5 Migration Quality

The tool maintains high data integrity:

- **Data Accuracy**: All data is validated before and after migration
- **Relationship Preservation**: Task hierarchies, dependencies, and assignments are maintained
- **Error Handling**: Automatic retry logic handles temporary issues
- **Resume Capability**: Can continue if the migration is interrupted

2.6 Security and Privacy

Your data remains secure throughout the migration:

- **Read-Only Access**: The tool only reads from Project Online, never modifies your original data

- **Encrypted Transfer**: All data transfers use secure HTTPS connections
- **Credential Protection**: Your credentials are stored locally and never logged
- **Audit Trail**: Smartsheet tracks who created and modified data

2.7 Next Steps

Ready to learn more about how the migration works? The next guide explains the technical architecture and components.

Start of Series Next: System Design →

---

3 System Design

4 System Design

Last Updated: 2024-12-08

4.1 How the Migration Tool Works

The migration tool is organized into six main components, each handling a specific part of moving your data from Project Online to Smartsheet.

4.1.1 1. Command Interface

What It Does: Provides the commands you use to run migrations

Available Commands:

```
# Run a complete migration
po-import import --source <your-project-id> --destination <workspace

# Check your Project Online data before migrating
po-import validate --source <your-project-id>

# Preview the migration without making changes
po-import import --source <your-project-id> --destination <workspace

# View your current configuration
po-import config
```

Key Features: - Parses your command options - Loads your configuration settings - Shows you progress as the migration runs - Displays helpful error messages if something goes wrong - Lets you preview changes before committing

4.1.2 2. Migration Coordinator

What It Does: Manages the overall migration workflow

The coordinator runs through these stages:

```
1. Initialization → Checks your configuration is valid
2. Extraction → Fetches your data from Project Online
3. Transformation → Converts your data for Smartsheet
4. Loading → Creates your workspaces and sheets in Smartsheet
5. Completion → Reports the results
```

Features: - Automatically retries if temporary issues occur - Validates your data before migration - Supports preview mode so you can test first - Handles errors gracefully with clear messages

4.1.3 3. Data Extraction

What It Does: Retrieves your project data from Project Online

How It Works: - Authenticates securely to your Project Online environment - Automatically handles large projects with many tasks - Respects rate limits to avoid overloading the system - Retries automatically if network issues occur - Tests connectivity before starting

What It Extracts: - Your project information and metadata - All tasks with their hierarchy and relationships - Your resources (team members, equipment, costs) - All assignments linking resources to tasks

Error Protection: - Automatic retry with increasing wait times on failures - Detection and handling of rate limits - Detailed logging to help troubleshoot any issues - Graceful handling if some data is missing

4.1.4 4. Data Transformation

What It Does: Converts your Project Online data into Smartsheet format

The transformation handles these conversions:

4.1.4.1 Your Project Information

- Creates workspace structure
- Generates a summary sheet with your project metadata
- Validates your project data
- Sets up consistent dropdown lists (status, priority)

Features: - Uses templates for faster workspace creation - Cleans up project names to work in Smartsheet - Configures dropdown lists that reference central standards

4.1.4.2 Your Tasks

- Creates 18 columns to hold all your task information
- Builds each task row preserving your hierarchy
- Converts task status and priority values
- Handles predecessor relationships between tasks
- Sets up dropdown lists for consistency

Features: - Maintains your task hierarchy through parent-child relationships - Enables Gantt chart view automatically - Calculates duration based on your start and end dates - Handles 8 constraint types (start as soon as possible, finish by date, etc.) - Can safely re-run if the migration is interrupted

4.1.4.3 Your Resources

- Creates 18 columns for resource information
- Builds a row for each of your team members and resources
- Discovers and lists your department values
- Validates resource data

- Sets up dropdown lists

Features: - Keeps email addresses separate from names - Handles people, equipment, and cost resources differently - Manages standard rates, overtime rates, and cost per use - Converts availability percentages properly - Can safely re-run without creating duplicates

4.1.4.4 Your Assignments

- Creates columns on your task sheet for assignments
- Groups resources by type (people vs equipment/costs)
- Generates one column per unique resource

Important Feature: Assignment columns work differently based on type: - **People resources** → Collaboration-enabled columns (you can @mention team members) - **Equipment/cost resources** → Simple selection lists (text-based)

Features: - Creates columns dynamically based on your actual assignments - Handles re-runs safely - Uses the right column type for people vs non-people resources

4.1.4.5 Standards Management

- Manages centralized reference lists for dropdown values
- Creates reference sheets for status, priority, and other standard fields
- Discovers values from your data (like department names)

Reference Sheets Created: - Project - Status (Active, Planning, etc.) - Project - Priority (Highest, High, Medium, etc.) - Task - Status (Not Started, In Progress, Complete) - Task - Priority (same levels as project priority) - Task - Constraint Type (8 different constraint types) - Resource - Type (People, Equipment, Cost) - Resource - Department (discovered from your data)

Benefits: - Enables consistent dropdown lists across all your projects - Centralizes value management - Single source of truth for your organization's standards

4.1.4.6 Common Conversions

- Cleans workspace names to remove invalid characters
- Converts ISO dates to standard format
- Converts durations to hours or days
- Maps numeric priorities to text labels
- Creates contact objects with names and emails
- Generates consistent sheet names

4.1.5 5. Smartsheet Integration

What It Does: Creates your workspaces, sheets, and data in Smartsheet

Operations: - Creates workspaces for your projects - Creates sheets with proper column definitions - Inserts your data in batches for efficiency - Configures dropdown lists and references - Sets up cross-sheet references for consistency

Smart Features: - Automatically handles rate limits - Retries operations if temporary failures occur - Provides detailed feedback if issues arise

4.1.6 6. Helper Utilities

What They Do: Provide supporting functions throughout the migration

Configuration Management: - Loads and validates your settings - Ensures all required credentials are present - Provides clear error messages for configuration issues

Progress Tracking: - Shows real-time progress as the migration runs - Breaks complex operations into trackable phases - Estimates time remaining for long operations

Error Handling: - Categorizes errors by type (configuration, network, data issues) - Provides actionable guidance for fixing problems - Makes error messages user-friendly

Retry Logic: - Configurable retry attempts (default: 3-5 tries) - Increasing wait times between retries (1 to 30 seconds) - Applied to all network operations

Re-run Support: - Reuses existing sheets by name if you run multiple times - Skips columns that already exist - Prevents data duplication - Enables safe retries if migrations are interrupted

4.2 What You'll See

4.2.1 During a Successful Migration

```
[2024-12-08 10:30:00] Starting Project Online to Smartsheet Migration
[2024-12-08 10:30:01] Configuration loaded successfully

[2024-12-08 10:30:02] ========== EXTRACTING YOUR DATA ==========
[2024-12-08 10:30:03] Connecting to Project Online...
[2024-12-08 10:30:05] □ Connected successfully
[2024-12-08 10:30:05] Extracting your project data...
[2024-12-08 10:30:08] □ Extracted project information
[2024-12-08 10:30:08] □ Extracted 25 tasks
[2024-12-08 10:30:09] □ Extracted 8 resources
[2024-12-08 10:30:10] □ Extracted 45 assignments
[2024-12-08 10:30:10] Extraction completed

[2024-12-08 10:30:11] ========== CONVERTING YOUR DATA ==========
[2024-12-08 10:30:11] Validating extracted data...
[2024-12-08 10:30:12] □ Data validation passed
[2024-12-08 10:30:12] Converting project structure...
[2024-12-08 10:30:13] □ Project converted
[2024-12-08 10:30:13] □ 25 tasks converted
[2024-12-08 10:30:14] □ 8 resources converted
[2024-12-08 10:30:14] Conversion completed

[2024-12-08 10:30:15] ========== CREATING IN SMARTSHEET ==========
[2024-12-08 10:30:15] Connecting to Smartsheet...
[2024-12-08 10:30:16] □ Connection established
[2024-12-08 10:30:16] Creating workspace: Marketing Campaign Q1
[2024-12-08 10:30:18] □ Workspace created
[2024-12-08 10:30:18] Creating sheets...
```

```
[2024-12-08 10:30:22]  Created 3 sheets
[2024-12-08 10:30:22] Loading your data...
[2024-12-08 10:30:35]  Loaded 25 tasks
[2024-12-08 10:30:35] Loading completed

[2024-12-08 10:30:36] ========== MIGRATION COMPLETE ==========
[2024-12-08 10:30:36] Summary:
[2024-12-08 10:30:36]   - Tasks: 25
[2024-12-08 10:30:36]   - Resources: 8
[2024-12-08 10:30:36]   - Assignments: 45
[2024-12-08 10:30:36] View your workspace: https://app.smartsheet.com
```

4.2.2 If Issues Occur

```
[2024-12-08 10:35:42] Issue loading data to "Tasks" sheet
[2024-12-08 10:35:42] Rate limit reached - this is temporary
[2024-12-08 10:35:42] Waiting 5 seconds before retry (Attempt 1/3)...
[2024-12-08 10:35:47] Retrying...
[2024-12-08 10:35:49]  Retry successful
```

4.3 Software Requirements

The tool uses these technologies:

For Connecting to Project Online: - Microsoft Authentication Library - Secure HTTP client

For Creating in Smartsheet: - Official Smartsheet software development kit

Supporting Tools: - Configuration file management - Progress tracking - Error handling - Command-line interface

All required software is included - you just need Node.js installed on your computer.

4.4 Security Measures

4.4.1 Managing Your Credentials

Best Practices: - All credentials are stored in a local `.env` file - This file is excluded from version control for security - A sample template is provided without actual credentials - Credentials are never displayed in logs or on screen - Configuration is validated before running

4.4.2 Protecting Your Data

Security Features: - No personal information is written to logs (only counts and identifiers) - All transfers use encrypted HTTPS connections - Smartsheet tracks all changes automatically - The tool only reads from Project Online, never modifies your original data

4.4.3 Access Requirements

What You'll Need: - **Project Online**: Read access to your project data - **Smartsheet**: API token with permission to create workspaces - **Standards Workspace**: Owner access to manage dropdown list values

4.5 Network Reliability

4.5.1 Handling Connection Issues

The tool automatically handles network problems:

Automatic Retries: - Retries failed operations with exponential backoff - Uses connection pooling for efficiency - Configurable timeout settings - Increasing delays between retry attempts - Automatic rate limit detection and throttling

4.5.2 Re-run Safety

What Happens If You Run Multiple Times: - The tool reuses existing sheets by name - Skips columns that already exist - Prevents data duplication - Safe to continue interrupted migrations - Can add new columns to existing sheets

Benefits: - Safe to re-run if the migration is interrupted - Can be used iteratively during testing - No corruption from multiple runs

4.6 Quality Assurance

4.6.1 Validation Checks

During Extraction: - Verifies all required fields are present - Checks for empty or invalid values - Validates identifier formats - Confirms date and time formats

During Transformation: - Validates converted values match expected formats - Checks text lengths don't exceed limits - Verifies parent-child relationships are valid - Validates all references between entities - Confirms numeric values are within acceptable ranges

Before Creating in Smartsheet: - Ensures sheet names are unique within each workspace - Checks column types are compatible - Verifies task hierarchy is consistent - Validates all predecessor references - Confirms required columns exist - Prevents duplicate identifiers

4.7 Next Steps

The next guide provides detailed information about how your data is converted from Project Online format to Smartsheet format.

← Previous: Migration Overview

Next: Data Transformation Guide →

5 How Your Data Transforms

6 How Your Data Transforms

Last Updated: 2024-12-08

This guide explains how your Project Online data converts to Smartsheet format, including field mappings, data type conversions, and naming patterns.

6.1 Table of Contents

6.2 Transformation Approach

6.2.1 How Your Data Maps

```
Your Project Online Data    →    What You Get in Smartsheet
═══════════════════════════════════════════════════════════

Project                          →    Workspace (one per project)
                                 →    Name matches your project name

Task (top level)                 →    Row in Tasks Sheet (level 0)
  └─ Task (subtask)              →    Indented Row (level 1)
      └─ Task (sub-subtask)      →    Indented Row (level 2)

Resource                         →    Row in Resources Sheet
                                 →    Contact with name and email
                                 →    Available for assignment selection

Assignment                       →    Column in Tasks Sheet
                                 →    Shows who's assigned to each task
```

6.2.2 Design Approach

1. **One Project = One Workspace**: Each Project Online project becomes its own Smartsheet workspace
2. **Direct Placement**: Sheets are placed directly in the workspace (no folders)
3. **Project Sheet Features**: Tasks sheet includes Gantt chart and dependency tracking

4. **Dual Identifiers**: Original identifier preserved plus readable auto-number
5. **Contact Integration**: Names and email addresses work together
6. **Embedded Assignments**: Assignments appear as columns in your task list

6.3 Field Mappings

6.4 1. Project Information

6.4.1 From Project Online to Smartsheet Workspace

What Happens: Your Project Online project becomes a dedicated Smartsheet workspace

Workspace Name: - Uses your exact project name - Removes characters that aren't allowed: `/\:*?"<>|` become − - Multiple dashes are consolidated to one - Limited to 100 characters maximum - If longer, truncates to 97 characters and adds `"..."`

Examples:

```
"Website Redesign 2024"           → "Website Redesign 2024"
"Q1/Q2 Planning & Execution"    → "Q1-Q2 Planning & Execution"
"IT Infrastructure | Phase 1"      → "IT Infrastructure - Phase 1"
```

6.4.2 Project Summary Sheet

Sheet Name: `{Your Project Name} - Summary`

Columns Created (15 columns, one row with your project information):

Column Name	What It Stores	Source	Example
Project Online Project ID	Original identifier	`Project.Id`	Hidden column
Project Name	Your project name	`Project.Name`	"Website Redesign 2024"
Description	Project description	`Project.Description`	"Complete redesign…"
Owner	Project owner with email	`Project.Owner` + Email	John Doe (john@example.com)
Start Date	When project starts	`Project.StartDate`	"2024-03-15"
Finish Date	When project completes	`Project.FinishDate`	"2024-06-30"
Status	Current status	`Project.ProjectStatus`	"Active"
Priority	Project priority	`Project.Priority`	"High"
% Complete	Completion percentage	`Project.PercentComplete`	"45%"
Project Online Created Date	When created in Project Online	`Project.CreatedDate`	"2024-03-01"
Project Online Modified Date	When last changed in Project Online	`Project.ModifiedDate`	"2024-03-15"
Created Date	When created in Smartsheet	System	Automatic
Modified Date	When last changed in Smartsheet	System	Automatic
Created By	Who created in Smartsheet	System	Automatic

Column Name	What It Stores	Source	Example
Modified By	Who last changed in Smartsheet	System	Automatic

6.5 2. Tasks

6.5.1 From Project Online Tasks to Smartsheet Task Rows

Sheet Name: `{Your Project Name} - Tasks`

Sheet Features: Includes Gantt chart view and dependency tracking

Columns Created (18+ columns for each task):

Column Name	What It Stores	Source	Example
Task Name	Your task name	`Task.TaskName`	"Design Homepage"
Task ID	Auto-generated identifier	Automatic	"WEB-00001"
Project Online Task ID	Original identifier	`Task.Id`	Hidden column
Start Date	When task starts	`Task.Start`	"2024-03-15"
End Date	When task finishes	`Task.Finish`	"2024-03-22"
Duration	How long task takes	`Task.Duration`	5.0 days
% Complete	Task completion	`Task.PercentComplete`	"45%"
Status	Current status	Calculated	"In Progress"
Priority	Task priority	`Task.Priority`	"Very High"
Work (hrs)	Planned hours	`Task.Work`	"40h"
Actual Work (hrs)	Hours completed	`Task.ActualWork`	"32h"
Milestone	Is this a milestone?	`Task.IsMilestone`	Checkmark or empty
Notes	Task notes	`Task.TaskNotes`	"Review with team"
Predecessors	Dependencies	`Task.Predecessors`	"5FS" (finish-to-start)
Constraint Type	Schedule constraint	`Task.ConstraintType`	"As Soon As Possible"
Constraint Date	Constraint date	`Task.ConstraintDate`	"2024-03-20"
Deadline	Must finish by	`Task.Deadline`	"2024-04-01"
Assignments	Who's assigned	Dynamic	Your team members

6.5.2 How Task Hierarchy Works

Your Project Online Structure: Tasks have outline levels (0 = top, 1 = subtask, 2 = sub-subtask, etc.)
In Smartsheet: Parent-child relationships show as indentation

How It Converts: 1. Tasks are sorted by their original order 2. The tool tracks each task's outline level 3. When the outline level increases, a child relationship is created 4. When the outline level

decreases, the tool returns to the parent level 5. Parent-child relationships are established in the row structure

Example:

```
Your Project Online:                    In Smartsheet:
- Task 1 (Level 0)          →          Row 1: Task 1 (top level)
- Task 1.1 (Level 1)        →        Row 2: Task 1.1 (indented under Row 1)
- Task 1.1.1 (Level 2)      →          Row 3: Task 1.1.1 (indented under Row 2)
- Task 1.2 (Level 1)        →        Row 4: Task 1.2 (indented under Row 1)
- Task 2 (Level 0)          →          Row 5: Task 2 (top level)
```

6.6 3. Resources

6.6.1 From Project Online Resources to Smartsheet Resource Rows

Sheet Name: `{Your Project Name} - Resources`

Sheet Type: Flat list (no hierarchy)

Columns Created (18 columns for each resource):

Column Name	What It Stores	Source	Example
Resource ID	Auto-generated identifier	Automatic	"WEB-00042"
Project Online Resource ID	Original identifier	`Resource.Id`	Hidden column
Contact	Name and email together	`Resource.Name` + `Email`	John Doe (john@example.com)
Resource Type	People, Equipment, or Cost	`Resource.ResourceType`	"Work" (people)
Max Units	Availability percentage	`Resource.MaxUnits`	"100%"
Standard Rate	Regular hourly rate	`Resource.StandardRate`	75.00
Overtime Rate	Overtime hourly rate	`Resource.OvertimeRate`	112.50
Cost Per Use	One-time cost	`Resource.CostPerUse`	50.00
Department	Department assignment	`Resource.Department`	"Engineering"
Code	Resource code	`Resource.Code`	"ENG-001"
Is Active	Currently active?	`Resource.IsActive`	Checkmark or empty
Is Generic	Generic resource?	`Resource.IsGeneric`	Checkmark or empty

6.7 4. Assignments

6.7.1 From Project Online Assignments to Task Sheet Columns

Important: There is no separate Assignments sheet - assignments appear as columns in your Tasks sheet.

How It Works: - **Team members (people)** → Collaboration-enabled columns (you can @mention them) - **Equipment/materials** → Simple selection lists (text-based)

Example Assignment Columns:

Column Name	Type	Contains	How It's Set Up
Team Members	Contact list	Your team members	Options come from Resources Sheet
Equipment	Selection list	Equipment items	Text-based options
Cost Centers	Selection list	Cost allocations	Text-based options

Benefits: - See assignments directly in your task list - Validated against your resource list - Enable collaboration features (notifications, mentions) - Simpler structure with fewer sheets

6.8 Data Type Conversions

6.8.1 Duration Conversion

From Project Online: ISO 8601 Duration format (e.g., `PT40H` means 40 hours)

To Smartsheet Duration Column: Decimal days

```
// PT40H → 5.0 (40 hours ÷ 8-hour day = 5 days)
// P5D → 5.0 (5 days)
// PT480M → 1.0 (480 minutes = 8 hours = 1 day)
```

To Work Hour Columns: Hours with "h" suffix

```
// PT40H → "40h"
// PT80H → "80h"
```

6.8.2 Date and Time Conversion

From Project Online: ISO 8601 DateTime (e.g., `2024-03-15T09:00:00Z`)

To Smartsheet: Date in `YYYY-MM-DD` format

```
// 2024-03-15T09:00:00Z → "2024-03-15"
// 2024-12-31T23:59:59-08:00 → "2024-12-31"
```

6.8.3 Priority Conversion

From Project Online: Number from 0 to 1000

To Smartsheet: Text label from dropdown list

Your Project Online Value	Becomes in Smartsheet
1000 or higher	Highest
800-999	Very High
600-799	Higher

Your Project Online Value	Becomes in Smartsheet
500-599	Medium
400-499	Lower
200-399	Very Low
0-199	Lowest

6.8.4 Status Conversion

From Project Online: Calculated from completion percentage

Conversion Rules: - `0%` → "Not Started" - `1-99%` → "In Progress" - `100%` → "Complete"

6.8.5 Contact Information

From Project Online: Separate name and email fields

To Smartsheet: Single contact field with both

```
// Project Online
Owner: "John Doe"
OwnerEmail: "john@example.com"

// Smartsheet (stored as one contact)
{
  "email": "john@example.com",
  "name": "John Doe"
}
```

6.8.6 Currency Values

From Project Online: Decimal number (e.g., `75.00`)

To Smartsheet: Numeric value (Smartsheet formats it as currency)

```
// Store as number, Smartsheet displays with currency symbol
75.0 → Displayed as "$75.00" in Smartsheet
112.5 → Displayed as "$112.50" in Smartsheet
```

6.8.7 Yes/No Fields

From Project Online: Boolean (true/false)

To Smartsheet: Checkbox column

```
// IsActive = true → ☑ (checked box)
// IsActive = false → ☐ (empty box)
```

6.8.8 Percentage Values

From Project Online: Decimal where 1.0 = 100%

To Smartsheet: Percentage with “%” symbol

```
// 1.0 → "100%"
// 0.5 → "50%"
// 1.5 → "150%" (overallocated)
```

6.9 Naming Patterns

6.9.1 Workspace Names

- **Pattern**: Uses your project name directly
- **Cleaning**: Removes characters that aren’t allowed: `/\:*?"<>|` become –
- **Examples**:
  - `"Website Redesign 2024"` stays as `"Website Redesign 2024"`
  - `"Q1/Q2 Planning & Execution"` becomes `"Q1-Q2 Planning & Execution"`
  - `"IT Infrastructure | Phase 1"` becomes `"IT Infrastructure - Phase 1"`

6.9.2 Sheet Names

- **Pattern**: `{Your Project Name} - {Sheet Type}`
- **Examples**:
  - `"Website Redesign - Tasks"`
  - `"Website Redesign - Resources"`
  - `"Website Redesign - Summary"`

Note: All sheets are placed directly in the workspace (no folders).

6.9.3 Column Names

Standard Format: - Each Word Starts With Capital Letter - Words are separated by spaces - Units are shown in parentheses

Examples: - `"Task Name"` (not “TaskName”) - `"Start Date"` (not “StartDate”) - `"Work (hrs)"` (not “WorkHours”) - `"% Complete"` (not “PercentComplete”)

Identifier Columns: - Always include ” ID” in the name - Hidden by default

Examples: - `"Task ID"` - `"Resource ID"` - `"Project ID"`

6.9.4 Value Formats

Dates: Year-Month-Day (e.g., `"2024-03-15"`)

Durations: Number + unit (e.g., `"5d"`, `"40h"`, `"2w"`)

Percentages: Number + `%` symbol (e.g., `"0%"`, `"50%"`, `"100%"`)

Currency: Numeric value (e.g., `75.00`, `112.50`) - Smartsheet formats it with currency symbol

Yes/No: Checkbox (☐ checked or ☐ empty)

6.10 Standards Architecture

6.10.1 Centralized Reference Values

A single “Standards” workspace contains reference sheets with all the dropdown list values used across your projects.

```
Standards Workspace (Centralized)
├── Standard Reference Sheets
│   ├── Project - Status (Active, Planning, etc.)
│   ├── Project - Priority (Highest, High, Medium, etc.)
│   ├── Task - Status (Not Started, In Progress, Complete)
│   ├── Task - Priority (same levels as project priority)
│   ├── Task - Constraint Type (8 different types)
│   └── Resource - Type (People, Equipment, Cost)
│
└── Discovered Reference Sheets
    └── Resource - Department (departments found in your data)
```

6.10.2 Reference Sheet Names

Pattern: `{Data Type} - {Field Name}`

Examples: - `"Project - Status"` (clearly indicates it’s for projects) - `"Task - Priority"` (separate from project priority) - `"Resource - Type"` - `"Resource - Department"`

Why: Prevents confusion when the same field name appears in different contexts

6.10.3 Reference Sheet Structure

Each reference sheet is simple: - **First Column**: “Name” (main column) - **Rows**: One row for each valid option

Example - Task Priority Reference Sheet:

Name
Highest
Very High
Higher
Medium
Lower
Very Low
Lowest

6.10.4 How Dropdown Lists Work

Your project sheets reference the standards workspace for dropdown values:

```
// Example: Task Priority Column Setup
{
  "title": "Priority",
  "type": "PICKLIST",
  "options": [{
    "sheetId": 234567890,       // Standards workspace Task - Priorit
    "columnId": 345678901        // "Name" column in that sheet
  }],
  "validation": true             // Only values from list are allowed
}
```

6.10.5 Benefits of This Approach

1. **Centralized Management**: Update values in one place, affects all projects
2. **Consistency Across Projects**: All projects use the same validated values
3. **Easy Updates**: Change a reference sheet once, all projects reflect the change
4. **Single Source of Truth**: One place for your organization's standard values
5. **Unlimited Scale**: Works no matter how many projects you migrate
6. **Automatic Discovery**: The tool finds and populates reference sheets for you

6.10.6 Migration Process

```
1. When You Start
   └─> Tool sets up or verifies the Standards workspace exists

2. For Each Project You Migrate
    └─> Creates your project workspace
    └─> Creates sheets (Tasks, Resources, Summary)
    └─> Connects dropdown lists to Standards workspace
    └─> Loads your data with validation

3. As You Migrate More Projects
    └─> New values automatically added to reference sheets
    └─> Existing projects reference the updated values
```

6.11 Data Validation

6.11.1 When Extracting from Project Online

- Checks that required fields have values
- Validates that identifiers are in the correct format
- Confirms dates and times are valid
- Ensures no critical data is missing

6.11.2 When Converting Your Data

- Validates converted values match expected formats
- Checks text doesn't exceed column limits (4000 characters max)
- Verifies parent tasks exist for child tasks
- Validates all references between tasks, projects, and resources
- Confirms numeric values are within acceptable ranges

6.11.3 Before Creating in Smartsheet

- Ensures each sheet has a unique name in the workspace
- Checks column types are compatible
- Verifies task hierarchy is consistent
- Validates all predecessor (dependency) references
- Confirms all required columns exist
- Prevents duplicate identifiers

6.12 Summary of Your Migration

What Gets Migrated: 4 main types of data (Projects, Tasks, Resources, Assignments)

Fields Mapped: 50+ individual pieces of information with detailed conversion rules

Data Conversions: 8 major types (Identifiers, Dates, Durations, Priority, Status, Contacts, Currency, Yes/No)

Relationships Maintained: - Your project-task connections - Task parent-child relationships (hierarchy) - Task dependencies (what must finish before what starts) - Resource assignments (who's assigned to what)

Result Structure: - 1 Workspace for each project - 2-3 sheets per project (Tasks, Resources, and optional Summary) - All sheets in workspace root (no folders) - Assignments embedded in task list (no separate sheet)

Key Patterns: 1. One workspace per project 2. Names preserved (with minor cleaning) 3. Two identifiers (original + auto-number) 4. Contacts with names and emails together 5. Assignments embedded in tasks 6. Standards workspace for consistency 7. Safe to re-run if needed

---

← Previous: System Design

Next: Template Workspace Creation →

---

7 Using Workspace Templates

8 Using Workspace Templates

8.1 What This Feature Does

When migrating your projects, the tool can use a pre-configured template to create your new Smartsheet workspaces. This means your workspaces start with all the right columns and sheet structure already set up.

8.2 Why Use a Template

Consistency: Every project workspace you migrate has the same structure, making it easier to work across multiple projects.

Pre-configured Columns: Complex column settings like dropdown lists and contact lists are already set up correctly.

Simplicity: The template defines the standard structure once, and every migration uses it.

8.3 How It Works

8.3.1 Setting Up the Template

You can optionally specify a template workspace in your configuration file (`.env`):

```
# Optional: Use a template workspace
TEMPLATE_WORKSPACE_ID=your_template_id_here

# Or leave empty to create workspaces from scratch
TEMPLATE_WORKSPACE_ID=
```

What the Tool Does: - **If you specify a template**: Copies that workspace and customizes it for each project - **If you don't specify**: Creates a blank workspace and builds it from scratch

8.3.2 Template Contents

Your template workspace should contain: - **Summary Sheet**: All 15 project information columns properly configured - **Tasks Sheet**: All 18 task columns including duration and dependencies - **Resources Sheet**: All 18 resource columns with dropdown lists and checkboxes

8.3.3 Migration Process

When you run a migration using a template:

1. **Copies the Template**: Makes a complete copy of your template workspace
2. **Renames Sheets**: Updates sheet names to match your project:
   - `Tasks` becomes `{Your Project Name} - Tasks`
   - `Resources` becomes `{Your Project Name} - Resources`

    - `Summary` becomes `{Your Project Name} - Summary`
3. **Clears Sample Data**: Removes any rows from the template, keeping only the column structure
4. **Loads Your Data**: Fills the sheets with your actual project information

8.4 Managing Your Template

8.4.1 When to Update the Template

You'll want to update your template workspace when: - You want to change which columns appear - You need to reorder columns - Column formatting needs adjustment - You want to add new sheets

8.4.2 How to Update

1. Make changes directly to your template workspace in Smartsheet
2. Test that the changes work by running a sample migration
3. Update the `TEMPLATE_WORKSPACE_ID` in your configuration if you're using a different template

No code changes are needed - the tool automatically uses whatever template you configure.

8.4.3 Creating a New Template

If you need to set up a new template:

1. Create a new workspace in Smartsheet with the structure you want
2. Add all the sheets with the columns configured properly
3. Optionally add sample data to see what it looks like (the tool will remove this during migration)
4. Note the workspace ID and add it to your `.env` configuration file:

```
TEMPLATE_WORKSPACE_ID=your_new_template_id_here
```

8.5 Testing Your Template

You can test that your template works correctly by running the migration with sample data. The tool will verify: - The new workspace is created successfully - Sheets have the correct names - All expected columns are present - Sample data is removed before your real data is loaded

← Previous: How Your Data Transforms

Next: Safe Re-runs →

9 Safe Re-runs

10 Safe Re-runs

Last Updated: 2025-12-05

10.1 What This Means

If you need to run a migration more than once for any reason, the tool protects against creating duplicate sheets or columns. This makes it safe to retry a migration if something goes wrong.

← Previous: Using Workspace Templates

Next: Sheet Connections →

11 How Sheets Connect to Each Other

12 How Sheets Connect to Each Other

Last Updated: 2024-12-05

12.1 What This Means

In your migrated Smartsheet workspaces, sheets can reference each other to maintain consistent data and enable dropdown lists. This ensures your project data stays organized and validated.

← Previous: Using Workspace Templates

Next: Authentication Setup →

13 Setting Up Authentication

14 Setting Up Authentication

This guide walks you through connecting the migration tool to your Project Online environment.

14.1 What You'll Need

To migrate your projects, the tool needs secure access to both your Project Online data and your Smartsheet account. This requires:

1. An application registration in Azure Active Directory
2. Appropriate permissions to read your Project Online data
3. Administrator approval for the application
4. Your credentials in a configuration file

14.2 Requirements

- **Azure Active Directory Admin Access**: You'll need someone with administrator privileges who can:
  - Create application registrations
  - Grant approval for application permissions
- **Project Online Access**: The application needs access to read your Project Online data
- **Your Organization's Information**: Know your Azure Active Directory tenant details

14.3 Step-by-Step Setup

14.3.1 Step 1: Create the Application Registration

1. Go to the Azure Portal
2. Navigate to **Azure Active Directory → App registrations**
3. Click **New registration**
4. Fill in the form:
   - **Name**: `Project Online Migration Tool`
   - **Supported account types**: Select "Accounts in this organizational directory only"
   - **Redirect URI**: Leave blank
5. Click **Register**

14.3.2 Step 2: Copy Your Application Information

After registering, you'll see the application overview page.

1. **Copy the Application (client) ID**
   - Save this as your `CLIENT_ID`
   - It looks like: `12345678-1234-1234-1234-123456789012`
2. **Copy the Directory (tenant) ID**

- Save this as your `TENANT_ID`
- It looks like: `87654321-4321-4321-4321-210987654321`

14.3.3 Step 3: Create a Client Secret

1. In your application registration, go to **Certificates & secrets**
2. Click **New client secret**
3. Fill in the form:
   - **Description**: `Migration Tool Secret`
   - **Expires**: Choose 12-24 months
4. Click **Add**
5. **IMPORTANT**: Copy the secret value immediately
   - This is your `CLIENT_SECRET`
   - **You can only see it once!**
   - If you close the page without copying it, you'll need to create a new one

14.3.4 Step 4: Grant Permissions

1. In your application registration, go to **API permissions**
2. Click **Add a permission**
3. Select **SharePoint** (Project Online uses SharePoint's infrastructure)
4. Choose **Application permissions**
5. Find and check **Sites.ReadWrite.All**
   - This lets the application read your Project Online data
   - Required for the migration to work
6. Click **Add permissions**
7. **IMPORTANT**: Click **Grant admin consent for [Your Organization]**
   - This requires administrator privileges
   - The application won't work without this approval
   - You should see a green checkmark after approval

14.3.5 Step 5: Verify the Setup

After granting consent, verify you see:

Permission	Type	Status
SharePoint / Sites.ReadWrite.All	Application	□ Granted for [Your Organization]

If you don't see the green checkmark, click "Grant admin consent" again.

14.3.6 Step 6: Configure the Tool

1. Copy the sample configuration file:

```
cp .env.sample .env
```

2. Edit the `.env` file and add your credentials:

```
# Azure Active Directory Configuration
TENANT_ID=your-tenant-id-from-step-2
CLIENT_ID=your-client-id-from-step-2
CLIENT_SECRET=your-client-secret-from-step-3
PROJECT_ONLINE_URL=https://your-organization.sharepoint.com/site

# Smartsheet Configuration
SMARTSHEET_API_TOKEN=your-smartsheet-token
```

3. Replace each placeholder value with your actual information

14.3.7 Step 7: Test the Connection

Test that everything is configured correctly:

```
npm run dev validate -- --source [your-project-id]
```

Replace `[your-project-id]` with one of your Project Online project identifiers.

If successful, you'll see:

```
□ Validating Project Online data

□ Project ID format is valid
□ Azure Active Directory tenant ID is configured
□ Azure Active Directory client ID is configured
□ Azure Active Directory client secret is configured
□ Project Online URL is configured
□ Smartsheet access token is configured

□ Testing Project Online connection...
□ Connection successful
□ Can access Project Online data

□ Validation passed
```

14.4 Troubleshooting

14.4.1 "Authentication failed" Error

What this means: Your credentials aren't working

How to fix: 1. Double-check that you copied all credentials correctly 2. Verify your client secret hasn't expired (check in Azure Portal) 3. Create a new client secret if needed

14.4.2 "Access forbidden" Error

What this means: The application doesn't have the necessary permissions

How to fix: 1. Go to Azure Portal → Your app registration → API permissions 2. Make sure **Sites.ReadWrite.All** shows with a green checkmark 3. If not approved, click "Grant admin consent for [Organization]" 4. Contact your administrator if you don't have permission to grant consent

14.4.3 Common Error Messages

Error	What It Means	How to Fix
`invalid_client`	Credentials are wrong	Verify `CLIENT_ID` and `CLIENT_SECRET` are correct
`invalid_resource`	URL is wrong	Check `PROJECT_ONLINE_URL` format
`unauthorized_client`	Not authorized	Grant admin consent for the application
`invalid_grant`	Token request failed	Create a new client secret

14.4.4 "Resource not found" Error

What this means: - Your Project Online URL is incorrect - The site doesn't exist - The application doesn't have access

How to fix: 1. Verify your Project Online URL format: `https://[your-organization].sharepoi name]` 2. Test the URL in your web browser (you should be able to access it) 3. Ensure your application has been granted access to the SharePoint site

14.4.5 "Cannot get token" Error

What this means: - Network connectivity issues - Firewall blocking Microsoft authentication - Invalid tenant ID

How to fix: 1. Check your internet connection 2. Verify your firewall allows access to: - `https://login.microsoftonline.com` - `https://*.sharepoint.com` 3. Confirm your `TENANT_ID` is correct

14.5 Security Best Practices

14.5.1 Protecting Your Credentials

1. **Never commit `.env` files to source control**
   - The file is excluded by default
   - Never share your `.env` file with others
2. **Rotate credentials regularly**
   - Set an expiration when creating secrets (12-24 months is recommended)

- Create a new secret before the old one expires
- Update your `.env` file with the new secret
- Delete the old secret after confirming the new one works

3. **Use separate credentials for testing and production**
   - Create different application registrations for testing versus actual use
   - Never use production credentials for testing

14.5.2 Managing Permissions

1. **Grant only necessary permissions**
   - Sites.ReadWrite.All is required to read Project Online data
   - Don't grant additional permissions you don't need
2. **Review regularly**
   - Periodically review which applications have permissions
   - Remove application registrations you're no longer using
   - Check who has administrative consent approval

14.5.3 Monitoring Access

1. **Review sign-in activity**
   - Azure Portal → Azure Active Directory → Sign-ins
   - Filter by your application name
   - Watch for failed authentication attempts
2. **Set up alerts** for unusual activity:
   - Multiple failed sign-in attempts
   - Unexpected sign-in patterns
   - Permission changes

14.6 Additional Resources

14.6.1 Microsoft Documentation

- Creating App Registrations
- Application Permissions
- Project Online Documentation

14.6.2 Getting Help

If you encounter issues:

1. Read the error message carefully - it often explains what's wrong
2. Review the troubleshooting section above
3. Verify all your configuration values are correct
4. Test the connection using the `validate` command
5. Contact your Azure Active Directory administrator for permission issues

15 Using the Migration Tool

16 Using the Migration Tool

Last Updated: 2024-12-08

16.1 Overview

The migration tool runs from the command line and provides real-time feedback as it moves your data from Project Online to Smartsheet. This guide explains how to use the tool effectively.

← Previous: Setting Up Authentication

Next: Troubleshooting →

17 Troubleshooting Common Issues

18 Troubleshooting Common Issues

Last Updated: 2024-12-08

This guide helps you diagnose and resolve common issues you might encounter when migrating from Project Online to Smartsheet.

18.1 How to Use This Guide

Each issue includes:

- **What you're seeing**: The symptoms or error messages
- **Why it's happening**: The underlying cause
- **How to diagnose**: Steps to confirm the issue
- **How to fix it**: Solutions that work
- **How to prevent it**: Avoiding the issue in future migrations

← Previous: Using the Migration Tool

Next: Code Conventions →

19 Code Conventions (Developer Documentation)

Developer Documentation

This document is intended for developers working on the migration tool code. If you're a Project Online user evaluating migration options, you can skip this section. The troubleshooting guide in the previous section covers everything you need for using the tool.

20 Code Conventions

This document describes coding standards, naming conventions, file organization patterns, and development workflows used in this project.

20.1 Language & Style

20.1.1 TypeScript Configuration

Target: ES2020 **Module System**: CommonJS **Strict Mode**: Enabled

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true,
    "declaration": true,
    "outDir": "./dist",
    "rootDir": "./src"
  }
}
```

20.1.2 ESLint & Prettier

ESLint: TypeScript-aware linting **Prettier**: Consistent code formatting

```
npm run lint          # Check for issues
npm run lint:fix      # Auto-fix issues
npm run format        # Format all code
```

20.2 Naming Conventions

20.2.1 Files and Directories

Source Files: PascalCase for classes, camelCase for utilities - `ProjectTransformer.ts` - Class exports - `SmartsheetHelpers.ts` - Utility functions - `utils.ts` - Shared utilities - `index.ts` - Barrel exports

Test Files: Match source filename with `.test.ts` suffix - `ProjectTransformer.test.ts` - `SmartsheetHelpers.test.ts`

Directory Structure: Lowercase with hyphens for multi-word - `src/lib/auth/` - Lowercase singular/plural as appropriate - `src/transformers/` - Plural for collections - `test/integration/` - Kebab-case for multi-word - `test/unit/builders/` - Nested organization

20.2.2 Variables and Functions

Variables: camelCase

```
const projectName = 'Sample Project';
const isActive = true;
const maxRetries = 3;
```

Constants: UPPER_SNAKE_CASE

```
// Configuration from environment (loaded via ConfigManager)
const TEMPLATE_WORKSPACE_ID = configManager.get().templateWorkspaceI
const MAX_RETRIES = 3;
const DEFAULT_TIMEOUT = 30000;
```

Functions: camelCase with verb prefix

```
// Good
function getProject(id: string): Promise<Project>
function createWorkspace(name: string): Promise<Workspace>
function validateTask(task: Task): ValidationResult
function transformProject(project: Project): Workspace

// Avoid
function project(id: string)  // Missing verb
function workspace(name: string)  // Unclear action
```

Boolean Variables: Use `is`, `has`, `should` prefixes

```
const isActive = true;
const hasChildren = false;
const shouldRetry = true;
const canCreate = user.hasPermission();
```

20.2.3 Classes and Interfaces

Classes: PascalCase, singular nouns

```
class ProjectTransformer { }
class ErrorHandler { }
class ExponentialBackoff { }
class Logger { }
```

Interfaces: PascalCase, often with descriptive suffix

```
interface ProjectOnlineProject { }
interface SmartsheetClient { }
interface ValidationResult { }
interface ProgressOptions { }
```

Type Aliases: PascalCase

```
type SmartsheetColumnType = 'TEXT_NUMBER' | 'DATE' | 'CONTACT_LIST';
type ODataQueryOptions = { $select?: string[]; $filter?: string };
```

20.2.4 Enums

Enum Names: PascalCase **Enum Values**: UPPER_SNAKE_CASE

```
enum LogLevel {
  DEBUG = 0,
  INFO = 1,
  WARN = 2,
  ERROR = 3,
  SILENT = 4,
}
```

20.3 File Organization

20.3.1 Source Code Structure

```
src/
├── cli.ts                      # CLI entry point
├── index.ts                    # Library entry point
├── lib/                        # Core libraries
│   ├── importer.ts            # Main ETL orchestrator
│   ├── ProjectOnlineClient.ts # OData API client
│   └── auth/                  # Authentication
│       └── MSALAuthHandler.ts
├── transformers/              # Data transformers
│   ├── ProjectTransformer.ts
│   ├── TaskTransformer.ts
│   ├── ResourceTransformer.ts
│   ├── AssignmentTransformer.ts
│   ├── PMOStandardsTransformer.ts
│   └── utils.ts              # Shared transformation utilities
├── types/                      # Type definitions
│   ├── ProjectOnline.ts       # Source types
│   ├── Smartsheet.ts          # Destination types
```

```
│   └── SmartsheetClient.ts   # Client interface types
└── util/                      # Cross-cutting utilities
    ├── ConfigManager.ts
    ├── ErrorHandler.ts
    ├── ExponentialBackoff.ts
    ├── Logger.ts
    ├── ProgressReporter.ts
    └── SmartsheetHelpers.ts
```

20.3.2 Test Structure

```
test/
├── setup.ts                    # Jest setup
├── importer.test.ts           # Importer validation tests
├── unit/                      # Unit tests (mocks only, no API calls)
│   ├── MockSmartsheetClient.ts       # Mock Smartsheet SDK
│   ├── MockODataClient.ts         # Mock Project Online oData client
│   ├── builders/                  # Fluent test data builders
│   │   ├── ODataProjectBuilder.ts
│   │   ├── ODataTaskBuilder.ts
│   │   ├── ODataResourceBuilder.ts
│   │   └── ODataAssignmentBuilder.ts
│   └── transformers/              # Transformer unit tests
│       ├── PMOStandardsTransformer.test.ts
│       ├── ProjectTransformer.test.ts
│       ├── ResourceTransformer.test.ts
│       ├── TaskTransformer.test.ts
│       └── utils.test.ts
├── integration/                # Integration tests (real Smartsheet A
│   ├── load-phase.test.ts         # Main ETL integration tests
│   ├── pmo-standards-integration.test.ts
│   ├── helpers/                # Integration test utilities
│   │   ├── smartsheet-setup.ts    # Workspace lifecycle management
│   │   └── odata-fixtures.ts       # Complete test fixtures
│   └── scenarios/                 # Pre-built test scenarios
│       ├── project-scenarios.ts
│       ├── task-scenarios.ts
│       ├── resource-scenarios.ts
│       └── assignment-scenarios.ts
├── lib/                                # Library tests
│   └── ProjectOnlineClient.test.ts
```

```
└── util/                                    # Utility tests
    ├── ExponentialBackoff.test.ts
    └── SmartsheetHelpers.test.ts
```

20.3.3 Import Organization

Order: External → Internal → Types → Relative

```typescript
// 1. External dependencies (Node.js built-ins first)
import * as fs from 'fs';
import * as path from 'path';

// 2. External packages (alphabetical)
import axios from 'axios';
import { Command } from 'commander';

// 3. Internal absolute imports (by category)
import { ProjectOnlineProject, ProjectOnlineTask } from '../types/Pr
import { SmartsheetClient } from '../types/SmartsheetClient';
import { Logger } from '../util/Logger';
import { ErrorHandler } from '../util/ErrorHandler';

// 4. Relative imports (closest first)
import { sanitizeWorkspaceName, convertDateTimeToDate } from './uti
import { getOrCreateSheet } from '../util/SmartsheetHelpers';
```

Barrel Exports: Use `index.ts` for clean imports

```typescript
// src/transformers/index.ts
export { ProjectTransformer } from './ProjectTransformer';
export { TaskTransformer } from './TaskTransformer';
export { ResourceTransformer } from './ResourceTransformer';
export * from './utils';

// Usage elsewhere
import { ProjectTransformer, sanitizeWorkspaceName } from './transfo
```

20.4 Code Style

20.4.1 Function Declarations

Prefer explicit return types for public APIs:

```typescript
// Good - Clear contract
export function createWorkspace(name: string): Promise<Workspace> {
  // ...
}

// Acceptable - Simple/obvious return type
```

```
function sanitizeName(name: string) {
  return name.trim().toLowerCase();
}
```

Use async/await over Promises:

```
// Good
async function loadProject(id: string): Promise<Project> {
  const data = await client.getProject(id);
  return transformProject(data);
}

// Avoid
function loadProject(id: string): Promise<Project> {
  return client.getProject(id)
    .then(data => transformProject(data));
}
```

20.4.2 Error Handling

Always handle errors explicitly:

```
// Good
try {
  await operation();
} catch (error) {
  errorHandler.handle(error, 'Operation Context');
  throw error;  // Re-throw after handling
}

// Avoid
await operation();  // Unhandled promise rejection
```

Use typed errors:

```
// Good
throw ErrorHandler.configError('SMARTSHEET_API_TOKEN', 'is required'

// Avoid
throw new Error('Missing token');  // No guidance
```

20.4.3 Comments

Use JSDoc for public APIs:

```
/**
 * Transform Project Online project to Smartsheet workspace
 *
 * @param project - Project Online project entity
```

```
 * @param workspaceId - Optional existing workspace ID
 * @returns Transformed workspace with sheet information
 * @throws {ValidationError} If project data is invalid
 */
export async function transformProject(
  project: ProjectOnlineProject,
  workspaceId?: number
): Promise<WorkspaceResult> {
  // Implementation
}
```

Use inline comments for complex logic:

```
// Convert Project Online priority (0-1000) to 7-level picklist
// Ranges: 1000+: Highest, 800-999: Very High, 600-799: Higher,
//         500-599: Medium, 400-499: Lower, 200-399: Very Low, 0-19
export function mapPriority(priority: number): string {
  if (priority >= 1000) return 'Highest';
  // ... rest of mapping
}
```

Avoid obvious comments:

```
// Bad
const name = project.Name;  // Get the name

// Good - No comment needed
const name = project.Name;
```

20.4.4 Variable Declarations

Prefer `const`, use `let` only for reassignment:

```
// Good
const projectName = 'Sample';
let retryCount = 0;

// Avoid
var projectName = 'Sample';  // Never use var
let projectName = 'Sample';  // Use const if not reassigned
```

Destructure for clarity:

```
// Good
const { Name, Description, Priority } = project;

// Acceptable
const projectName = project.Name;
const projectDescription = project.Description;
```

Use meaningful names:

```
// Good
const maxRetryAttempts = 3;
const resourcesByType = groupBy(resources, r => r.ResourceType);

// Avoid
const n = 3;  // What is n?
const temp = groupBy(resources, r => r.ResourceType);  // Temporary
```

20.5 Type Safety

20.5.1 Strict Type Checking

Enable all strict checks: - `strict: true` in tsconfig.json - No `any` types (use `unknown` if needed) - Explicit return types for public functions - Non-null assertions only when guaranteed

Avoid type assertions:

```
// Good - Type guard
function isProject(data: unknown): data is Project {
  return typeof data === 'object' &&
         data !== null &&
         'Name' in data;
}

if (isProject(data)) {
  console.log(data.Name);  // Type safe
}

// Avoid
const project = data as Project;  // Unsafe
```

20.5.2 Interface vs Type

Prefer interfaces for object shapes:

```
// Good
interface ProjectOnlineProject {
  Id: string;
  Name: string;
  Priority?: number;
}

// Use type for unions/intersections
type Result = Success | Failure;
type EnhancedProject = ProjectOnlineProject & { metadata: Metadata }
```

20.5.3 Optional vs Undefined

Use optional parameters consistently:

```
// Good
function createSheet(
  name: string,
  columns?: SmartsheetColumn[]
): Promise<Sheet>

// Avoid mixing
function createSheet(
  name: string,
  columns: SmartsheetColumn[] | undefined
): Promise<Sheet>
```

20.6 Testing Conventions

20.6.1 Test Structure

Follow AAA pattern (Arrange-Act-Assert):

```
describe('ProjectTransformer', () => {
  it('should transform project to workspace', async () => {
    // Arrange
    const mockClient = new MockSmartsheetClient();
    const transformer = new ProjectTransformer(mockClient);
    const project = new ODataProjectBuilder()
      .withName('Test Project')
      .build();

    // Act
    const result = await transformer.transformProject(project);

    // Assert
    expect(result.workspace.name).toBe('Test Project');
    expect(mockClient.createWorkspaceCalls).toHaveLength(1);
  });
});
```

20.6.2 Test Naming

Describe behavior, not implementation:

```
// Good
it('should create workspace with sanitized name')
it('should throw ValidationError for invalid project')
it('should retry failed API calls up to 3 times')
```

```
// Avoid
it('calls createWorkspace()')  // Implementation detail
it('test project transformer')  // Too vague
```

20.6.3 Test Organization

Group related tests:

```
describe('ProjectTransformer', () => {
  describe('transformProject', () => {
    it('should create workspace for valid project');
    it('should throw for invalid project');
    it('should handle missing optional fields');
  });

  describe('validateProject', () => {
    it('should pass validation for complete project');
    it('should fail validation for missing required fields');
  });
});
```

20.6.4 Mock Usage

Use mocks for external dependencies only:

```
// Good - Mock external service
const mockSmartsheetClient = new MockSmartsheetClient();

// Avoid - Don't mock pure functions
const mockSanitizer = jest.fn(sanitizeWorkspaceName);  // Test real
```

20.7 Git Workflow

20.7.1 Commit Messages

Format: `type(scope): description`

Types: - `feat`: New feature - `fix`: Bug fix - `docs`: Documentation changes - `test`: Test additions/modifications - `refactor`: Code refactoring - `perf`: Performance improvements - `chore`: Build/tooling changes

Examples:

```
feat(transformer): add support for Material resources
fix(api): handle pagination for large result sets
docs(readme): update authentication setup instructions
test(integration): add scenario for complex project hierarchy
refactor(util): extract common validation logic
```

20.7.2 Branch Naming

Format: `type/short-description-feature/add-resource-transformer-fix/rate-limiting-bug-docs/update-api-reference-test/integration-scenarios`

20.7.3 Pull Request Guidelines

Title: Same format as commit messages **Description**: Include: - What changed - Why it changed - How to test - Related issues

20.8 Documentation

20.8.1 Code Documentation

Public APIs: Always document with JSDoc **Private functions**: Document if complex **Types**: Document non-obvious fields **Configuration**: Document all options

20.8.2 README Updates

When to update: - New dependencies added - New commands added - Configuration changes - Breaking changes - Installation steps change

20.8.3 Changelog

Keep `CHANGELOG.md` current: - Document all user-facing changes - Group by version - Include migration guides for breaking changes

20.9 Performance Guidelines

20.9.1 Async Operations

Prefer parallel over sequential when safe:

```
// Good - Parallel when independent
const [tasks, resources] = await Promise.all([
  client.getTasks(projectId),
  client.getResources()
]);

// Good - Sequential when dependent
const project = await client.getProject(id);
const tasks = await client.getTasks(project.Id);
```

20.9.2 API Calls

Batch operations:

```
// Good - Single batch call
await client.addRows(sheetId, allRows);

// Avoid - Individual calls
for (const row of allRows) {
  await client.addRow(sheetId, row);  // N API calls
}
```

Cache expensive operations:

```
// Good - Cache column map
const columnMap = await getColumnMap(client, sheetId);
for (const task of tasks) {
  const row = createTaskRow(task, columnMap);  // Reuse cached map
}

// Avoid - Repeated lookups
for (const task of tasks) {
  const columnMap = await getColumnMap(client, sheetId);  // N lookup
}
```

20.10 Security

20.10.1 Credential Handling

Never log credentials:

```
// Good - Mask in logs
logger.info(`Token: ${maskToken(token)}`);

// Avoid
logger.info(`Token: ${token}`);  // Exposes credential
```

Use environment variables:

```
// Good
const apiToken = process.env.SMARTSHEET_API_TOKEN;

// Avoid
const apiToken = 'abc123...';  // Hardcoded
```

20.10.2 Input Validation

Validate all external input:

```
// Good
if (!project.Name || project.Name.trim() === '') {
  throw ErrorHandler.validationError('Project Name', 'non-empty stri
}
```

```
// Avoid
const workspaceName = project.Name;  // Assumes valid
```

20.11 Best Practices Summary

20.11.1 Do □

- Use TypeScript strict mode
- Write tests for new code
- Handle errors explicitly
- Document public APIs
- Use meaningful variable names
- Follow the style guide
- Run linter before committing
- Keep functions focused and small
- Use dependency injection
- Log at appropriate levels

20.11.2 Don't □

- Use `any` type
- Ignore TypeScript errors
- Commit commented-out code
- Use `var` declarations
- Write functions > 50 lines
- Catch errors without handling
- Hardcode configuration values
- Commit console.log statements
- Mix async/sync patterns
- Skip tests for "simple" code

20.12 Code Review Checklist

Before submitting PR: - [ ] All tests pass - [ ] Linter passes (no warnings) - [ ] New code has tests - [ ] Public APIs have JSDoc - [ ] Error handling is present - [ ] No hardcoded values - [ ] No debugging statements - [ ] Follows naming conventions - [ ] TypeScript strict mode satisfied - [ ] Documentation updated if needed

20.13 Questions?

If you're unsure about any convention: 1. Check existing code for examples 2. Review this guide and the patterns guide 3. Ask in PR comments 4. Discuss in team meeting

When in doubt, consistency with existing code takes precedence.

← Previous:
Troubleshooting Playbook

Next: Code Patterns →

21 Code Patterns (Developer Documentation)

Developer Documentation

This document is intended for developers working on the migration tool code. If you're a Project Online user evaluating migration options, you can skip this section.

22 Code Patterns

This document catalogs recurring code patterns used throughout the project. Understanding these patterns will help you write code that's consistent with the existing codebase.

22.1 Architectural Patterns

22.1.1 1. ETL Pipeline Pattern

Context: Multi-stage data transformation from source (Project Online) to destination (Smartsheet).

Implementation:

```
// src/lib/importer.ts
export class ProjectOnlineImporter {
  async importProject(data: ProjectImportData): Promise<ImportResult
    // Stage 1: Setup
    const pmoStandards = await this.setupPMOStandards();

    // Stage 2: Extract & Transform Project
    const workspace = await this.transformProject(data.project);

    // Stage 3: Load Tasks
    const tasks = await this.loadTasks(data.tasks, workspace);

    // Stage 4: Load Resources
    const resources = await this.loadResources(data.resources, worksp

    // Stage 5: Load Assignments
    await this.loadAssignments(data.assignments, tasks, resources);

    return { workspace, tasks, resources };
  }
}
```

When to use: Any multi-step data processing workflow that requires ordered execution.

22.1.2 2. Dependency Injection Pattern

Context: Decouple classes from their dependencies for testability and flexibility.

Implementation:

```
// Class receives dependencies via constructor
export class TaskTransformer {
  constructor(
    private client: SmartsheetClient,
    private logger?: Logger
  ) {}
}

// Usage
const transformer = new TaskTransformer(
  smartsheetClient,
  logger
);
```

Benefits: - Easy to mock dependencies in tests - Explicit dependency declarations - Flexible configuration

When to use: Any class that depends on external services or utilities.

22.1.3 3. Template Method Pattern

Context: Define algorithm skeleton in base class, let subclasses implement specific steps.

Implementation:

```
// Base pattern for transformers
export abstract class BaseTransformer<TSource, TTarget> {
  async transform(source: TSource): Promise<TTarget> {
    this.validate(source);
    const target = this.convert(source);
    await this.load(target);
    return target;
  }

  protected abstract validate(source: TSource): void;
  protected abstract convert(source: TSource): TTarget;
  protected abstract load(target: TTarget): Promise<void>;
}
```

When to use: Multiple classes share common workflow but differ in implementation details.

22.2 Data Transformation Patterns

22.2.1 4. Builder Pattern for Test Data

Context: Create complex test objects with fluent API.

Implementation:

```typescript
// test/unit/builders/ODataProjectBuilder.ts
export class ODataProjectBuilder {
  private project: Partial<ProjectOnlineProject> = {};

  withName(name: string): this {
    this.project.Name = name;
    return this;
  }

  withPriority(priority: number): this {
    this.project.Priority = priority;
    return this;
  }

  build(): ProjectOnlineProject {
    return {
      Id: this.project.Id || uuid(),
      Name: this.project.Name || 'Test Project',
      ...this.project,
    } as ProjectOnlineProject;
  }
}

// Usage
const project = new ODataProjectBuilder()
  .withName('Sample Project')
  .withPriority(800)
  .build();
```

When to use: Creating test data with many optional fields.

22.2.2 5. Functional Transformation Pattern

Context: Transform data through pure functions without side effects.

Implementation:

```typescript
// src/transformers/utils.ts
export function sanitizeWorkspaceName(projectName: string): string {
  return projectName
    .replace(/[/\\:*?"<>|]/g, '-')  // Replace invalid chars
    .replace(/-+/g, '-')             // Consolidate dashes
    .trim()                          // Remove leading/trailing spac
    .replace(/^-+|-+$/g, '')         // Remove leading/trailing dash
    .substring(0, 100);              // Truncate to limit
}

export function convertDateTimeToDate(isoDateTime: string): string {
```

```
  const date = new Date(isoDateTime);
  const year = date.getUTCFullYear();
  const month = String(date.getUTCMonth() + 1).padStart(2, '0');
  const day = String(date.getUTCDate()).padStart(2, '0');
  return `${year}-${month}-${day}`;
}
```

Benefits: - Easy to test (no dependencies) - Composable - Predictable output

When to use: Data format conversions, sanitization, mapping.

22.2.3 6. Mapping Pattern

Context: Convert between different data models (OData ↔ Smartsheet).

Implementation:

```
// Priority mapping: 0-1000 scale → 7 text levels
export function mapTaskPriority(priority: number): string {
  if (priority >= 1000) return 'Highest';
  if (priority >= 800) return 'Very High';
  if (priority >= 600) return 'Higher';
  if (priority >= 500) return 'Medium';
  if (priority >= 400) return 'Lower';
  if (priority >= 200) return 'Very Low';
  return 'Lowest';
}
```

Pattern: Range-based mapping with clear boundaries.

When to use: Converting between incompatible value systems.

22.3 API Integration Patterns

22.3.1 7. Retry with Exponential Backoff

Context: Handle transient failures in API calls gracefully.

Implementation:

```
// src/util/ExponentialBackoff.ts
export class ExponentialBackoff {
  constructor(
    private maxTries: number,
    private backoffTime: number,
    private maxDelayMs: number = 60000
  ) {}

  async execute<T>(operation: () => Promise<T>): Promise<T> {
    let lastError: Error;
```

```
    for (let attempt = 0; attempt < this.maxTries; attempt++) {
      try {
        return await operation();
      } catch (error) {
        lastError = error as Error;

        if (attempt < this.maxTries - 1) {
          const delay = Math.min(
            this.backoffTime * Math.pow(2, attempt),
            this.maxDelayMs
          );
          await this.delay(delay);
        }
      }
    }

    throw lastError!;
  }
}

// Usage
const backoff = new ExponentialBackoff(3, 1000);
const result = await backoff.execute(() => client.getProjects());
```

When to use: Any external API call that might fail transiently.

22.3.2 8. Rate Limiting Pattern

Context: Respect API rate limits to avoid throttling.

Implementation:

```
// src/lib/ProjectOnlineClient.ts
export class ProjectOnlineClient {
  private requestTimestamps: number[] = [];
  private rateLimitPerMinute: number;

  private async checkRateLimit(): Promise<void> {
    const now = Date.now();
    const oneMinuteAgo = now - 60000;

    // Remove old timestamps
    this.requestTimestamps = this.requestTimestamps.filter(
      t => t > oneMinuteAgo
    );

    // Wait if at limit
    if (this.requestTimestamps.length >= this.rateLimitPerMinute) {
```

```
      const oldestTimestamp = this.requestTimestamps[0];
      const waitTime = 60000 - (now - oldestTimestamp);
      await delay(waitTime);
    }

    this.requestTimestamps.push(now);
  }
}
```

When to use: High-volume API interactions with known rate limits.

22.3.3 9. Pagination Pattern

Context: Handle paginated API responses automatically.

Implementation:

```
// src/lib/ProjectOnlineClient.ts
async fetchAllPages<T>(initialUrl: string): Promise<T[]> {
  const results: T[] = [];
  let nextUrl: string | undefined = initialUrl;

  while (nextUrl) {
    const response = await this.httpClient.get<ODataCollectionRespons
      nextUrl
    );

    results.push(...response.data.value);
    nextUrl = response.data['@odata.nextLink'];
  }

  return results;
}

// Usage
const allTasks = await client.fetchAllPages<Task>('/api/tasks');
```

When to use: APIs that return paginated results with continuation tokens.

22.4 Error Handling Patterns

22.4.1 10. Typed Error with Actionable Messages

Context: Provide clear, actionable guidance when errors occur.

Implementation:

```
// src/util/ErrorHandler.ts
export class ImportError extends Error {
  constructor(
```

```
    message: string,
    public readonly code: string,
    public readonly actionable: string,
    public readonly details?: unknown
  ) {
    super(message);
    this.name = 'ImportError';
  }
}

// Factory methods for common errors
export class ErrorHandler {
  static configError(field: string, issue: string): ConfigurationErr
    return new ConfigurationError(
      `Configuration error: ${field} - ${issue}`,
      `Update your .env file to set ${field} correctly. ` +
      `Copy .env.sample to .env if you haven't already.`
    );
  }

  static rateLimitError(retryAfterMs?: number): ConnectionError {
    const waitTime = retryAfterMs
      ? `Wait ${Math.ceil(retryAfterMs / 1000)} seconds`
      : 'Wait a few minutes';

    return new ConnectionError(
      'API rate limit exceeded',
      `${waitTime} before retrying. Consider reducing batch size.`,
      { retryAfterMs }
    );
  }
}
```

Benefits: - Users know exactly what went wrong - Clear next steps for resolution - Consistent error messaging

When to use: All error scenarios, especially user-facing errors.

22.4.2 11. Error Context Pattern

Context: Add context to errors as they bubble up the call stack.

Implementation:

```
export class ErrorHandler {
  handle(error: unknown, context?: string): void {
    const prefix = context ? `[${context}] ` : '';

    if (error instanceof ImportError) {
```

```
      this.logger.error(`${prefix}${error.message}`);
      this.logger.info(`□ What to do: ${error.actionable}`);
    } else if (error instanceof Error) {
      this.logger.error(`${prefix}${error.message}`);
      const guidance = this.inferActionableGuidance(error);
      if (guidance) {
        this.logger.info(`□ What to do: ${guidance}`);
      }
    }
  }
}

// Usage
try {
  await operation();
} catch (error) {
  errorHandler.handle(error, 'Task Import');
  throw error;
}
```

When to use: Multi-layer applications where knowing the operation context helps debugging.

22.5 Resiliency Patterns

22.5.1 12. Get-or-Create Pattern

Context: Support re-running operations without errors or duplicates.

Implementation:

```
// src/util/SmartsheetHelpers.ts
export async function getOrCreateSheet(
  client: SmartsheetClient,
  workspaceId: number,
  sheetConfig: { name: string; columns: SmartsheetColumn[] }
): Promise<SmartsheetSheet> {
  // Check if sheet already exists
  const existingSheet = await findSheetInWorkspace(
    client,
    workspaceId,
    sheetConfig.name
  );

  if (existingSheet) {
    // Return existing sheet
    const fullSheet = await client.sheets.getSheet({
      id: existingSheet.id
    });
```

```
    return fullSheet.data;
  }

  // Sheet doesn't exist - create it
  const createdSheet = await client.sheets.createSheetInWorkspace({
    workspaceId,
    body: sheetConfig,
  });

  return createdSheet.data;
}
```

Benefits: - Idempotent operations - Safe to retry failed imports - No duplicate resource creation

When to use: Any create operation that might be retried.

22.5.2 13. Existence Check Before Creation

Context: Verify resource doesn't exist before attempting creation.

Implementation:

```
// Pattern: Check → Create if missing
const existingColumn = await findColumnInSheet(
  client,
  sheetId,
  columnTitle
);

if (!existingColumn) {
  const newColumn = await client.sheets.addColumn({
    sheetId,
    body: columnConfig,
  });
  return newColumn;
}

return existingColumn;
```

When to use: Operations that fail if resource already exists.

22.5.3 14. Template Copy Pattern

Context: Create pre-configured resources faster and more reliably.

Implementation:

```
// src/transformers/ProjectTransformer.ts
// Template workspace ID is configured via TEMPLATE_WORKSPACE_ID en
// If not set, creates blank workspace from scratch
```

```
constructor(private client: SmartsheetClient, configManager?: ConfigM
  this.templateWorkspaceId = configManager?.get().templateWorkspaceI
}

async transformProject(project: ProjectOnlineProject): Promise<Worksp
  if (this.templateWorkspaceId) {
    // Copy from template if configured
    const workspace = await copyWorkspace(
      this.client,
      this.templateWorkspaceId,
      project.Name
    );
  } else {
    // Create blank workspace from scratch
    const workspace = await this.client.workspaces.createWorkspace({
      name: project.Name,
    });
  }

  // Find and rename pre-existing sheets
  const summarySheet = await findSheetByPartialName(
    this.client,
    workspace.id,
    'Summary'
  );
  await renameSheet(this.client, summarySheet.id, `${project.Name} -

  // Clear template data
  await deleteAllRows(this.client, summarySheet.id);

  return workspace;
}
```

Benefits: - Faster than creating from scratch - Consistent structure - Fewer API calls

When to use: Creating complex resources with standard structures.

22.6 Logging Patterns

22.6.1 15. Structured Logging with Levels

Context: Provide appropriate logging detail for different audiences.

Implementation:

```
// src/util/Logger.ts
export enum LogLevel {
  DEBUG = 0,   // Detailed diagnostic information
  INFO = 1,    // General information
```

```
  WARN = 2,     // Warning conditions
  ERROR = 3,    // Error conditions
  SILENT = 4,   // No logging
}

export class Logger {
  debug(message: string, ...args: unknown[]): void {
    if (this.level <= LogLevel.DEBUG) {
      this.log('DEBUG', message, args, '\x1b[90m');
    }
  }

  info(message: string, ...args: unknown[]): void {
    if (this.level <= LogLevel.INFO) {
      this.log('INFO', message, args, '\x1b[36m');
    }
  }

  success(message: string, ...args: unknown[]): void {
    if (this.level <= LogLevel.INFO) {
      this.log('SUCCESS', message, args, '\x1b[32m');
    }
  }
}
```

Usage Guidelines: - **DEBUG**: Implementation details, variable values - **INFO**: Progress updates, major milestones - **WARN**: Recoverable issues, deprecated usage - **ERROR**: Failures, exceptions - **SUCCESS**: Successful completion of operations

When to use: All significant operations, especially long-running ones.

22.6.2 16. Progress Reporting Pattern

Context: Provide user feedback during long-running operations.

Implementation:

```
// src/util/ProgressReporter.ts
export class MultiStageProgressReporter {
  defineStages(stages: { name: string; total: number }[]): void {
    this.stages = stages.map(s => ({
      name: s.name,
      total: s.total,
      completed: 0,
    }));
  }

  startStage(stageName: string): void {
    this.currentStage = stageName;
```

```
    this.logger.info(`\n Stage: ${stageName} (0/${stage.total})`);
  }

  updateStage(count: number = 1, message?: string): void {
    const stage = this.stages.get(this.currentStage);
    stage.completed += count;

    const percentage = Math.round((stage.completed / stage.total) *
    this.logger.info(`  ${stage.completed}/${stage.total} (${percent
  }
}

// Usage in importer
const progress = new MultiStageProgressReporter(logger);
progress.defineStages([
  { name: 'PMO Standards', total: 1 },
  { name: 'Project Setup', total: 1 },
  { name: 'Task Import', total: tasks.length },
  { name: 'Resource Import', total: resources.length },
]);

progress.startStage('Task Import');
for (const task of tasks) {
  await processTask(task);
  progress.updateStage(1, task.name);
}
```

When to use: Operations that process multiple items or take significant time.

22.7 Testing Patterns

22.7.1 17. Mock with Spy Pattern

Context: Track method calls while providing canned responses.

Implementation:

```
// test/unit/MockSmartsheetClient.ts
export class MockSmartsheetClient implements SmartsheetClient {
  public createSheetInWorkspaceCalls: any[] = [];

  createSheetInWorkspace = jest.fn(async (workspaceId, sheet) => {
    this.createSheetInWorkspaceCalls.push({ workspaceId, sheet });
    return {
      id: this.nextId++,
      name: sheet.name,
      columns: sheet.columns,
    };
```

```
  });
}

// Usage in tests
const mockClient = new MockSmartsheetClient();
const transformer = new ProjectTransformer(mockClient);

await transformer.transform(project);

expect(mockClient.createSheetInWorkspace).toHaveBeenCalledTimes(3);
expect(mockClient.createSheetInWorkspaceCalls[0].sheet.name)
  .toBe('Test Project - Summary');
```

When to use: Testing classes that depend on external services.

22.7.2 18. Scenario-Based Test Pattern

Context: Test realistic end-to-end workflows.

Implementation:

```
// test/integration/scenarios/project-scenarios.ts
export const scenarios = {
  simpleProject: {
    project: new ODataProjectBuilder()
      .withName('Simple Project')
      .withPriority(500)
      .build(),
    tasks: [
      new ODataTaskBuilder().withName('Task 1').build(),
      new ODataTaskBuilder().withName('Task 2').build(),
    ],
    resources: [
      new ODataResourceBuilder().withName('John Doe').build(),
    ],
  },

  complexProject: {
    // ... complex scenario
  },
};

// Usage in integration tests
describe('Project Import', () => {
  it('should import simple project', async () => {
    const { project, tasks, resources } = scenarios.simpleProject;
    const result = await importer.importProject({
      project,
```

```
      tasks,
      resources,
    });

    expect(result.workspace.name).toBe('Simple Project');
    expect(result.tasksCreated).toBe(2);
  });
});
```

When to use: Integration tests validating complete workflows.

22.8 Best Practices

22.8.1 Pattern Selection Guidelines

1. **ETL Pipeline**: Use for multi-stage data transformations
2. **Dependency Injection**: Use for all classes with dependencies
3. **Retry/Backoff**: Use for all external API calls
4. **Get-or-Create**: Use for resource creation that might retry
5. **Typed Errors**: Use for all error conditions
6. **Progress Reporting**: Use for operations > 5 seconds
7. **Logging**: Use at all significant decision points
8. **Builders**: Use for complex test data creation

22.8.2 Anti-Patterns to Avoid

- □ Creating resources without checking existence
- □ Direct API calls without retry logic
- □ Generic errors without actionable messages
- □ Hardcoding values that should be configurable
- □ Mutations in transformation functions
- □ Synchronous operations for I/O
- □ Test data inline in test files
- □ Missing error context in logs

22.8.3 Pattern Evolution

When you need to add a new pattern:

1. Implement it in one place first
2. Refine based on real usage
3. Extract to reusable utility
4. Document with examples
5. Add tests
6. Update this guide

← Previous: Code Conventions

Next: Anti-Patterns →

23 Anti-Patterns (Developer Documentation)

Developer Documentation

This document is intended for developers working on the migration tool code. If you're a Project Online user evaluating migration options, you can skip this section.

24 Anti-Patterns

This document identifies common mistakes, code smells, and anti-patterns to avoid. Learning what NOT to do is as important as learning best practices.

24.1 Architecture Anti-Patterns

24.1.1 1. God Object / Swiss Army Knife Class

Problem: Single class that does too many things.

Bad Example:

```
class ProjectManager {
  // Authentication
  authenticate() { }

  // API calls
  getProjects() { }
  getTasks() { }

  // Transformation
  transformProject() { }
  transformTasks() { }

  // Validation
  validateProject() { }

  // Logging
  logInfo() { }
  logError() { }

  // Configuration
  loadConfig() { }

  // 500+ more lines...
}
```

Why It's Bad: - Difficult to test - Hard to understand - Violates Single Responsibility Principle - Changes in one area affect everything - Cannot reuse components independently

Correct Approach:

```
// Separate concerns into focused classes
class AuthHandler { }
class ProjectOnlineClient { }
class ProjectTransformer { }
class Validator { }
class Logger { }
class ConfigManager { }

// Compose them together
class ProjectImporter {
  constructor(
    private client: ProjectOnlineClient,
    private transformer: ProjectTransformer,
    private logger: Logger
  ) {}
}
```

24.1.2 2. □ Tight Coupling to External Services

Problem: Directly instantiating dependencies instead of injecting them.

Bad Example:

```
class ProjectTransformer {
  transform(project: Project) {
    // Hardcoded dependency
    const client = new SmartsheetClient(process.env.TOKEN);
    const logger = new Logger();

    // Now impossible to test without real API
    return client.createWorkspace(project.name);
  }
}
```

Why It's Bad: - Cannot test without real Smartsheet API - Cannot swap implementations - Difficult to configure - Hidden dependencies

Correct Approach:

```
class ProjectTransformer {
  constructor(
    private client: SmartsheetClient,  // Injected
    private logger: Logger             // Injected
  ) {}

  transform(project: Project) {
    // Easy to mock in tests
    return this.client.createWorkspace(project.name);
  }
```

```
}

// Usage
const transformer = new ProjectTransformer(
  new SmartsheetClient(config.token),
  new Logger()
);
```

24.1.3 3. □ Leaky Abstractions

Problem: Implementation details leak through the interface.

Bad Example:

```
// Exposes implementation detail (OData)
class ProjectClient {
  async getProjects(): Promise<ODataResponse<Project>> {
    // Returns OData-specific structure
  }
}

// Users must know about OData
const response = await client.getProjects();
const projects = response.value;  // OData structure leaked
```

Why It's Bad: - Clients depend on implementation details - Hard to change underlying technology - Violates abstraction principle

Correct Approach:

```
// Clean interface, hide implementation
class ProjectOnlineClient {
  async getProjects(): Promise<Project[]> {
    const response = await this.httpClient.get<ODataResponse<Project
    // Handle OData internally
    return response.value;
  }
}

// Simple usage
const projects = await client.getProjects();  // Just get projects
```

24.2 Data Transformation Anti-Patterns

24.2.1 4. □ Mutation of Input Data

Problem: Modifying input parameters instead of creating new objects.

Bad Example:

```
function sanitizeWorkspaceName(project: Project): Project {
  // MUTATES input parameter
  project.Name = project.Name.trim()
    .replace(/[invalid]/g, '-');
  return project;  // Returns modified input
}

// Caller's data is changed!
const project = { Name: 'Test/Project' };
sanitizeWorkspaceName(project);
console.log(project.Name);  // 'Test-Project' (unexpected!)
```

Why It's Bad: - Unexpected side effects - Hard to debug - Not composable - Breaks referential transparency

Correct Approach:

```
function sanitizeWorkspaceName(name: string): string {
  // Pure function, no mutation
  return name.trim().replace(/[invalid]/g, '-');
}

// Original unchanged
const project = { Name: 'Test/Project' };
const sanitizedName = sanitizeWorkspaceName(project.Name);
console.log(project.Name);  // 'Test/Project' (unchanged)
console.log(sanitizedName);  // 'Test-Project'
```

24.2.2 5. □ Silent Data Loss

Problem: Dropping data without warning when transformation fails.

Bad Example:

```
function transformTasks(tasks: Task[]): SmartsheetRow[] {
  return tasks.map(task => {
    try {
      return createTaskRow(task);
    } catch (error) {
      // SILENTLY IGNORE ERROR
      return null;
    }
  }).filter(row => row !== null);  // Lost tasks!
}
```

Why It's Bad: - Data loss is invisible - Debugging is impossible - Users don't know data is missing - Violates fail-fast principle

Correct Approach:

```
function transformTasks(tasks: Task[]): SmartsheetRow[] {
  const rows: SmartsheetRow[] = [];
  const errors: Array<{ task: Task; error: Error }> = [];

  for (const task of tasks) {
    try {
      rows.push(createTaskRow(task));
    } catch (error) {
      errors.push({ task, error: error as Error });
      logger.error(`Failed to transform task ${task.Id}`, error);
    }
  }

  if (errors.length > 0) {
    throw new DataError(
      `Failed to transform ${errors.length} of ${tasks.length} tasks
      'Review task data for validation errors'
    );
  }

  return rows;
}
```

24.2.3 6. □ Magic Numbers and Strings

Problem: Hardcoded values without explanation.

Bad Example:

```
function mapPriority(priority: number): string {
  if (priority >= 1000) return 'Highest';
  if (priority >= 800) return 'Very High';  // Why 800?
  if (priority >= 600) return 'Higher';     // Why 600?
  // ...
}

// Hardcoded ID
const workspace = await copyWorkspace(9002412817049476, name);
```

Why It's Bad: - No context for values - Hard to maintain - Easy to introduce errors - Not self-documenting

Correct Approach:

```
// Named constants with documentation
const PRIORITY_THRESHOLDS = {
  HIGHEST: 1000,
  VERY_HIGH: 800,
  HIGHER: 600,
```

```
  MEDIUM: 500,
  LOWER: 400,
  VERY_LOW: 200,
} as const;

function mapPriority(priority: number): string {
  if (priority >= PRIORITY_THRESHOLDS.HIGHEST) return 'Highest';
  if (priority >= PRIORITY_THRESHOLDS.VERY_HIGH) return 'Very High';
  // Clear intent
}

// Configuration from environment variable (via ConfigManager)
// Set in .env: TEMPLATE_WORKSPACE_ID=9002412817049476
const templateId = configManager.get().templateWorkspaceId ?? 900241
const workspace = await copyWorkspace(templateId, name);
```

24.3 Error Handling Anti-Patterns

24.3.1 7. □ Swallowing Exceptions

Problem: Catching errors and not handling them properly.

Bad Example:

```
async function importProject(id: string) {
  try {
    const project = await client.getProject(id);
    return transform(project);
  } catch (error) {
    // SWALLOWED - user never knows what happened
    return null;
  }
}
```

Why It's Bad: - Errors disappear - Debugging is impossible - User gets no feedback - System continues in invalid state

Correct Approach:

```
async function importProject(id: string) {
  try {
    const project = await client.getProject(id);
    return transform(project);
  } catch (error) {
    // Log error with context
    errorHandler.handle(error, `Import Project ${id}`);

    // Re-throw or return meaningful error
    throw ErrorHandler.dataError(
```

```
      `Failed to import project ${id}`,
      'Verify the project ID exists and you have access'
    );
  }
}
```

24.3.2 8. □ Generic Error Messages

Problem: Error messages that don't help users resolve the issue.

Bad Example:

```
throw new Error('Import failed');
throw new Error('Invalid data');
throw new Error('Something went wrong');
```

Why It's Bad: - No actionable information - User can't fix the issue - Support burden increases

Correct Approach:

```
// Specific, actionable errors
throw ErrorHandler.validationError(
  'Project Name',
  'non-empty string',
  project.Name
);

throw ErrorHandler.configError(
  'SMARTSHEET_API_TOKEN',
  'Copy .env.sample to .env and add your 26-character API token'
);

throw ErrorHandler.connectionError(
  'Project Online',
  'Check network connectivity and verify PROJECT_ONLINE_URL is corre
);
```

24.3.3 9. □ Catching Everything

Problem: Overly broad catch blocks that handle all errors the same way.

Bad Example:

```
try {
  await operation1();
  await operation2();
  await operation3();
} catch (error) {
  // Which operation failed? Can't tell!
```

```
  logger.error('Operation failed');
}
```

Why It's Bad: - Can't determine what failed - Can't handle different errors appropriately - Loses error context

Correct Approach:

```
try {
  await operation1();
} catch (error) {
  errorHandler.handle(error, 'Operation 1');
  throw error;
}

try {
  await operation2();
} catch (error) {
  errorHandler.handle(error, 'Operation 2');
  throw error;
}

// Or handle specific error types differently
try {
  await apiCall();
} catch (error) {
  if (error instanceof RateLimitError) {
    await delay(error.retryAfterMs);
    return apiCall();  // Retry
  }
  if (error instanceof AuthError) {
    await refreshToken();
    return apiCall();  // Retry with new token
  }
  throw error;  // Other errors propagate
}
```

24.4 API Integration Anti-Patterns

24.4.1 10. □ No Retry Logic

Problem: Failing immediately on transient errors.

Bad Example:

```
async function getProjects(): Promise<Project[]> {
  // Single attempt - any network blip fails permanently
  const response = await httpClient.get('/api/projects');
  return response.data;
```

```
}
```

Why It's Bad: - Transient failures cause permanent errors - Poor user experience - Unnecessary manual retries

Correct Approach:

```
async function getProjects(): Promise<Project[]> {
  // Automatic retry with exponential backoff
  return this.backoff.execute(async () => {
    const response = await this.httpClient.get('/api/projects');
    return response.data;
  });
}
```

24.4.2 11. □ Ignoring Rate Limits

Problem: Making requests without considering API rate limits.

Bad Example:

```
// Send 1000 requests as fast as possible
for (const task of tasks) {
  await client.createRow(sheetId, task);  // Quickly hit rate limit
}
```

Why It's Bad: - Gets throttled by API - Forces manual retry - Wastes time and resources

Correct Approach:

```
// Batch operations
await client.addRows(sheetId, tasks);  // Single API call

// Or implement rate limiting
await this.rateLimiter.checkLimit();
await client.createRow(sheetId, task);
```

24.4.3 12. □ Not Handling Pagination

Problem: Only processing first page of results.

Bad Example:

```
async function getAllTasks(): Promise<Task[]> {
  const response = await client.get('/api/tasks');
  return response.data.value;  // Only first 100 tasks!
}
```

Why It's Bad: - Missing data - Silent truncation - Incorrect totals

Correct Approach:

```
async function getAllTasks(): Promise<Task[]> {
  const tasks: Task[] = [];
  let nextLink = '/api/tasks';

  while (nextLink) {
    const response = await client.get(nextLink);
    tasks.push(...response.data.value);
    nextLink = response.data['@odata.nextLink'];
  }

  return tasks;
}
```

24.5 Testing Anti-Patterns

24.5.1 13. □ Testing Implementation Details

Problem: Tests that break when refactoring without behavior change.

Bad Example:

```
it('should call sanitizeWorkspaceName', () => {
  const spy = jest.spyOn(utils, 'sanitizeWorkspaceName');
  transformer.transformProject(project);
  expect(spy).toHaveBeenCalled();  // Tests HOW, not WHAT
});
```

Why It's Bad: - Brittle tests - Prevents refactoring - Doesn't test actual behavior

Correct Approach:

```
it('should create workspace with sanitized name', () => {
  const project = { Name: 'Test/Project<Invalid>' };
  const result = transformer.transformProject(project);
  expect(result.workspace.name).toBe('Test-Project-Invalid');  // Te
});
```

24.5.2 14. □ No Assertions / Weak Assertions

Problem: Tests that don't actually verify anything.

Bad Example:

```
it('should transform project', async () => {
  await transformer.transformProject(project);
  // No assertions! Test always passes!
});

it('should create workspace', async () => {
  const result = await transformer.transformProject(project);
```

```
  expect(result).toBeTruthy();  // Too weak - what about result?
});
```

Why It's Bad: - False sense of coverage - Doesn't catch bugs - Wastes time

Correct Approach:

```
it('should transform project', async () => {
  const result = await transformer.transformProject(project);

  // Verify specific expected outcomes
  expect(result.workspace.name).toBe('Test Project');
  expect(result.sheets.summarySheet).toBeDefined();
  expect(result.sheets.taskSheet).toBeDefined();
  expect(mockClient.createWorkspace).toHaveBeenCalledWith({
    name: 'Test Project',
  });
});
```

24.5.3 15. □ Test Interdependence

Problem: Tests that depend on execution order or shared state.

Bad Example:

```
let sharedProject: Project;

it('should create project', () => {
  sharedProject = createProject();  // Sets shared state
});

it('should transform project', () => {
  // DEPENDS on previous test running first!
  const result = transform(sharedProject);
});
```

Why It's Bad: - Tests can't run in isolation - Order matters - Parallel execution fails - Hard to debug

Correct Approach:

```
describe('ProjectTransformer', () => {
  let project: Project;

  beforeEach(() => {
    // Each test gets fresh state
    project = new ODataProjectBuilder().build();
  });

  it('should create project', () => {
    const result = createProject(project);
```

```
    expect(result).toBeDefined();
  });

  it('should transform project', () => {
    const result = transform(project);
    expect(result).toBeDefined();
  });
});
```

24.6 Performance Anti-Patterns

24.6.1 16. □ N+1 Query Problem

Problem: Making N API calls when 1 would suffice.

Bad Example:

```
// Get tasks
const tasks = await client.getTasks(projectId);

// Then get each task's details individually
for (const task of tasks) {
  const details = await client.getTask(task.Id);  // N API calls!
  processTask(details);
}
```

Why It's Bad: - Extremely slow - Wastes API quota - Doesn't scale

Correct Approach:

```
// Get all tasks with details in one call
const tasks = await client.getTasks(projectId, {
  $expand: 'Details',  // Include details in response
});

for (const task of tasks) {
  processTask(task);  // Already has details
}
```

24.6.2 17. □ Unnecessary Await in Loops

Problem: Sequential async operations that could be parallel.

Bad Example:

```
// Sequential - takes 3 seconds total
for (const project of projects) {
  await processProject(project);  // 1 second each
}
```

Why It's Bad: - Unnecessarily slow - Doesn't utilize parallelism - Poor user experience

Correct Approach:

```
// Parallel - takes 1 second total (if independent)
await Promise.all(
  projects.map(project => processProject(project))
);

// Or controlled concurrency
const concurrency = 5;
for (let i = 0; i < projects.length; i += concurrency) {
  const batch = projects.slice(i, i + concurrency);
  await Promise.all(batch.map(p => processProject(p)));
}
```

24.6.3 18. □ Excessive Logging

Problem: Logging everything, including sensitive data.

Bad Example:

```
logger.debug('API Token:', apiToken);  // Exposes credential!
logger.debug('Request:', JSON.stringify(hugeObject));  // 10MB log e
logger.debug('Processing...', item, details, metadata, ...);  // Eve
```

Why It's Bad: - Security risk - Performance overhead - Fills disk/log service - Makes finding important logs harder

Correct Approach:

```
logger.debug('API Token:', maskToken(apiToken));  // Masked
logger.debug('Request size:', getSize(object));  // Summary only
logger.info(`Processing ${items.length} items`);  // Aggregate info
logger.debug('Item processed', item.Id);  // Only IDs, not full obje
```

24.7 Configuration Anti-Patterns

24.7.1 19. □ Hardcoded Configuration

Problem: Configuration values embedded in code.

Bad Example:

```
const client = new SmartsheetClient('abc123token456');
const apiUrl = 'https://api.projectonline.microsoft.com';
const maxRetries = 3;
const timeout = 30000;
```

Why It's Bad: - Can't change without code changes - Different environments need different builds - Credentials in source control - No configuration management

Correct Approach:

```
// Environment variables
const client = new SmartsheetClient(
  process.env.SMARTSHEET_API_TOKEN!
);
const apiUrl = process.env.PROJECT_ONLINE_URL!;
const maxRetries = parseInt(process.env.MAX_RETRIES || '3');
const timeout = parseInt(process.env.TIMEOUT || '30000');

// Or ConfigManager
const config = configManager.get();
const client = new SmartsheetClient(config.smartsheetApiToken);
```

24.7.2 20. □ Missing Validation

Problem: Using configuration without validation.

Bad Example:

```
const token = process.env.SMARTSHEET_API_TOKEN;
// Use token without checking if it exists or is valid
const client = new SmartsheetClient(token);
```

Why It's Bad: - Runtime errors in production - Confusing error messages - Hard to diagnose

Correct Approach:

```
const token = process.env.SMARTSHEET_API_TOKEN;

if (!token) {
  throw ErrorHandler.configError(
    'SMARTSHEET_API_TOKEN',
    'is required but not set in .env file'
  );
}

if (!/^[a-zA-Z0-9]{26}$/.test(token)) {
  throw ErrorHandler.configError(
    'SMARTSHEET_API_TOKEN',
    'must be a 26-character alphanumeric token'
  );
}

const client = new SmartsheetClient(token);
```

24.8 Summary: Anti-Pattern Red Flags

□ **Watch for these warning signs**: - Classes > 500 lines - Functions > 50 lines - Try-catch blocks without error handling - No tests for "simple" code - Commented-out code - TODO comments left

indefinitely - Generic variable names (temp, data, obj) - Copy-pasted code blocks - console.log statements - Hardcoded values - Silent failures - Type assertions (`as`) - `any` types - Disabled linter rules - Skipped tests

24.9 Refactoring Anti-Patterns

When you see anti-patterns: 1. **Don't ignore them** - Technical debt compounds 2. **Do refactor incrementally** - Small, safe changes 3. **Do add tests first** - Safety net for refactoring 4. **Do document why** - Explain the improvement 5. **Don't refactor and add features** - One thing at a time

24.10 Questions or Concerns?

If you find yourself thinking: - "This is just a quick hack" → It will become permanent - "I'll fix it later" → You won't - "It works, why change it?" → Maintainability matters - "The tests are too hard" → Design issue, not test issue

Stop and refactor. Future you will thank you.

Next: API Services Catalog →

← Previous: Code Patterns

---

25 API Services Reference (Developer Documentation)

Developer Documentation

This document is a technical API reference intended for developers working on the migration tool. If you're a Project Online user evaluating migration options, you can skip this section.

26 API Services Reference

This catalog provides a comprehensive reference for all APIs and services used by the Project Online to Smartsheet ETL application.

26.1 External APIs

26.1.1 Project Online OData API

Primary API: Microsoft Project Online REST API (OData v4)
Documentation: Project Online OData Reference
Base URL Pattern: `https://{tenant}.sharepoint.com/sites/{site}/_api/Proje`

Endpoint	HTTP Method	Purpose	Used By Components
`/Projects`	GET	List all projects	`ProjectOnlineClient.getProjec`
`/Projects('{GET}')`	GET	Get single project by ID	`ProjectOnlineClient.getProjec`
`/Tasks`	GET	List all tasks	`ProjectOnlineClient.getTasks(`
`/Resources`	GET	List all resources	`ProjectOnlineClient.getResour`
`/Assignments`	GET	List all assignments	`ProjectOnlineClient.getAssign`

26.1.1.1 OData Query Options

The API supports standard OData v4 query parameters:

Parameter	Purpose	Example
`$select`	Select specific fields	`?$select=Id,Name,StartDate`
`$filter`	Filter results	`?$filter=ProjectId eq guid'xxx'`
`$orderby`	Sort results	`?$orderby=Name asc`
`$top`	Limit results	`?$top=100`
`$skip`	Skip results (pagination)	`?$skip=100`
`$expand`	Expand related entities	`?$expand=Tasks`

26.1.1.2 Pagination

- API returns paginated results automatically
- Response includes `@odata.nextLink` for next page
- Client handles pagination automatically via `fetchAllPages()`

26.1.1.3 Rate Limiting

- **Default Limit:** 300 requests per minute (configurable)
- **Implementation:** Client-side rate limiting in `checkRateLimit()`
- **Response Code:** 429 Too Many Requests
- **Retry-After Header:** Indicates wait time in seconds

26.1.1.4 Error Handling

Status Code	Meaning	Client Handling
200 OK	Success	Return data
401 Unauthorized	Token expired	Clear cache, retry
403 Forbidden	Insufficient permissions	Throw auth error with guidance
404 Not Found	Resource doesn't exist	Throw not found error
429 Too Many Requests	Rate limit exceeded	Wait and retry
500+ Server Error	Server issue	Retry with backoff

← Previous: Anti-Patterns Next: Test Suite Guide →

27 Test Suite (Developer Documentation)

Developer Documentation

This document describes the testing framework for developers working on the migration tool. If you're a Project Online user evaluating migration options, you can skip this section.

28 Test Suite

This directory contains the complete test suite for the Project Online to Smartsheet ETL tool, including both **unit tests** (using mocks) and **integration tests** (using real Smartsheet API).

28.1 Test Organization

```
test/
├── README.md                                # This file
├── setup.ts                                 # Jest configuration and .env.test loadi
├── importer.test.ts                         # Importer validation tests
├── unit/                                    # Unit tests (mocks only, no API calls)
│   ├── MockSmartsheetClient.ts              # Mock Smartsheet SDK
│   ├── MockODataClient.ts                   # Mock Project Online oData client
│   ├── builders/                            # Fluent test data builders
│   │   ├── ODataProjectBuilder.ts
│   │   ├── ODataTaskBuilder.ts
│   │   ├── ODataResourceBuilder.ts
│   │   └── ODataAssignmentBuilder.ts
│   └── transformers/                        # Transformer unit tests
│       ├── PMOStandardsTransformer.test.ts
│       ├── ProjectTransformer.test.ts
│       ├── ResourceTransformer.test.ts
│       ├── TaskTransformer.test.ts
│       └── utils.test.ts
├── integration/                             # Integration tests (real Smartsheet A
│   ├── load-phase.test.ts                   # Main ETL integration tests
│   ├── pmo-standards-integration.test.ts
│   ├── helpers/                             # Integration test utilities
│   │   ├── smartsheet-setup.ts              # Workspace lifecycle management
│   │   └── odata-fixtures.ts                # Complete test fixtures
│   └── scenarios/                           # Pre-built test scenarios
│       ├── project-scenarios.ts
│       ├── task-scenarios.ts
```

```
│       ├── resource-scenarios.ts
│       └── assignment-scenarios.ts
├── lib/                                  # Library tests
│   └── ProjectOnlineClient.test.ts
└── util/                                 # Utility tests
    ├── ExponentialBackoff.test.ts
    └── SmartsheetHelpers.test.ts
```

← Previous: API Services Catalog

End of Series
